Orientación a objetos en PHP

Miguel Angel Alvarez
Manu Gutierrez

desarrolloweb.com

desarrolloweb.com/manuales/manual-php.html

Introducción: Orientación a objetos con PHP

Explicamos al detalle todas las características de la orientación a objetos de PHP, la orientación a objetos avanzada que comenzó con PHP 5 y que continua vigente en PHP 7.

PHP se ha modernizado en los últimos años trayendo las características más sofisticadas para la programación orientada a objetos, potenciando todos los beneficios en el código de los lenguajes de programación más avanzados.

Encuentras este manual online en:
http://desarrolloweb.com/manuales/58.php

Autores del manual

Las siguientes personas han participado como autores escribiendo artículos de este manual.

Miguel Angel Alvarez

Miguel es fundador de DesarrolloWeb.com y la plataforma de formación online EscuelaIT. Comenzó en el mundo del desarrollo web en el año 1997, transformando su hobby en su trabajo.

Manu Gutierrez

Objetivos e introducción al de orientación a objetos de PHP

Este manual pretende dar un repaso a las nuevas características de PHP 5, en cuanto a la programación orientada a objetos, que le diferencian de versiones anteriores. La orientación a objetos con PHP 5 es muy avanzada y pone a PHP al nivel de los mejores lenguajes de programación orientados a objetos que podamos encontrar, con características y herramientas de altísimo nivel. La orientación a objetos de PHP 7 sigue siendo la misma que la de PHP 5. Comenzaremos por ver qué es PHP 5 y obtener referencias para aprender PHP general en DesarrolloWeb.com y cómo instalar PHP 5 en nuestro ordenador local para desarrollo.

Introducción a PHP 5

Introducción al manual del lenguaje PHP en su versión 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 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 y la Introducción a los lenguajes del web.

Asimismo, queremos presentarte la sección monotemática de PHP, donde se concentran todos los contenidos sobre PHP que dispone DesarrolloWeb.com. También sería interesante el Manual del lenguaje SQL y el Taller de MySQL, 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.

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.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *16/11/2004*
Disponible online en *http://desarrolloweb.com/articulos/58.php*

Introducción a la orientación a objetos en PHP 5

Aprendemos qué es la orientación a objetos y vemos cómo era la aproximación que existía en las versiones anteriores del lenguaje. Comenzaremos a explicar los conceptos más básicos de orientación a objetos en PHP 5.

Evolución del modelo de orientación a objetos de PHP

A lo largo de los años el modelo de orientación a objetos de PHP ha cambiado mucho. La programación orientada a objetos de las versiones anteriores a la 5 era muy rudimentaria y ahora es extremadamente avanzada.

Para entender la importancia de los cambios de PHP 5, en cuanto a orientación a objetos, es interesante observar cómo ha evolucionado el modelo de objetos de las versiones anteriores. Comenzamos a trabajar con objetos en PHP 3 con una propuesta extremadamente rudimentaria y aunque esto queda ya muy lejos, lo cierto es que nos da una idea de cómo el mundo de PHP se ha profesionalizado y cómo este lenguaje ha pasado de ser una herramienta ocasional a una alternativa seria, a la altura de los lenguajes de programación más avanzados.

Lo cierto es que este artículo te dará un poco de conocimiento general de PHP y te ayudará a tener una visión de pájaro del lenguaje más precisa, con un horizonte más lejano en el tiempo. Pero si no te interesa y estás con prisa podrías saltarte este capítulo del Manual de Orientación a Objetos de PHP, puesto que no es absolutamente imprescindible para poder programar en objetos con PHP.

Orientación a objetos en las versiones antiguas de PHP: PHP3 y PHP4

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.

Orientación a objetos en PHP 5 y PHP 7

Ahora vamos a conocer cómo PHP 5 implementa la reciente la orientación a objetos, ofreciendo un listado de las novedades con respecto a los objetos en versiones anteriores. Ahora la programación orientada a objetos con PHP 5 es realmente avanzada y se mantiene en PHP 7, por lo que todo lo que vas a leer a continuación tiene validez también en la versión más reciente del lenguaje. No obstante, a medida que han pasado los años todavía nos han traído novedades que han mejorado este panorama y que comentamos en futuros artículos del manual, como los namespaces o traits.

En la parte inicial de este artículo 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.

Este artículo es obra de *Miguel Angel Alvarez*

Fue publicado por primera vez en *16/11/2004*
Disponible online en *http://desarrolloweb.com/articulos/58.php*

Modelo de orientación a objetos en PHP 5

Cómo trabaja PHP 5 con la orientación a objetos. Listado de las novedades con respecto a los objetos en versiones anteriores.

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.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *16/11/2004*
Disponible online en *http://desarrolloweb.com/articulos/58.php*

Clases en PHP 5

Vemos que es una clase, y como podemos definirlas e instanciarlas.

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();
```

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.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *30/11/2004*
Disponible online en *http://desarrolloweb.com/articulos/58.php*

Constructores en PHP 5

Vamos a ver qué es un constructor y para que sirven, además de un sencillo ejemplo de una clase que define un constructor.

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

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *14/12/2004*
Disponible online en *http://desarrolloweb.com/articulos/58.php*

Sobrecarga de constructores en PHP

Cómo podemos fabricar por nosotros mismos un sistema de sobrecarga de métodos, no soportada de manera nativa en PHP, en el caso específico de constructores, tan importante para la reutilización del código.

La Programación Orientada a Objetos (POO) en PHP tiene una carencia que considero fundamental y quien venga de otros lenguajes la echará seriamente en falta. Se trata de la **sobrecarga de métodos**, que es uno de los pilares fundamentales que hace de la POO una gran forma de crear *software* más versátil y que reutilice más código.

En este artículo estudiaremos una alternativa muy sencilla, pero totalmente funcional para poder simular la sobrecarga de métodos, de modo que aprovechemos sus posibilidades tal como si fuera algo nativamente soportado por PHP. En concreto, encararemos el problema con los métodos constructores, de modo que podamos construir objetos con distintos juegos de parámetros.

¿Por qué no hay sobrecarga de métodos en PHP?

Al notar la falta de sobrecarga de métodos en la Programación Orientada a Objetos implementada en PHP, uno se puede quedar pensando ¿por qué no existe en PHP? ¿no se trata de una POO devaluada debido a la carencia de sobrecarga? ¿cómo puedo hacer clases que tengan constructores que acepten varios juegos de parámetros?

Después de analizar la situación, podemos observar que en PHP sufrimos la carencia de sobrecarga debido a que es un lenguaje poco estricto. En el caso de las funciones, PHP tiene la particularidad de permitir invocarlas enviando un juego de parámetros diferente al que fue declarada. Sin invocamos una función sin enviarle todos los parámetros, PHP dará mensajes de advertencia, pero el código se ejecuta. Si al invocar una función enviamos más parámetros de los que tocaba, ni siquiera nos da un mensaje de advertencia.

Es por ello que no pueden coexistir funciones con el mismo nombre y distintos juegos de parámetros, porque PHP siempre va a utilizar la función que se declare (una única vez), enviando los parámetros de que disponga en el momento se su invocación.

¿Cómo simular la sobrecarga de métodos constructores?

La salida que nos queda para aprovechar las posibilidades de la sobrecarga de métodos es programarla "a mano". Se trata de un proceso sencillo de incorporar, aunque no cabe duda de que es más bonito cuando está admitido por el lenguaje de manera nativa.

> **Nota:** Este ejemplo lo hemos aplicado a constructores, pero podría servirnos crear un esquema similar con el que sobrecargar cualquier método que necesitemos. Puedes ver cómo son los constructores de PHP en un artículo anterior.

El truco está en tener un método "genérico" que no recibe parámetros y un método específico para cada número de parámetros que pensemos aceptar. El método "genérico" lo declaramos

sin indicar ningún parámetro y dentro de su código utilizar la función de PHP func_get_args(), que nos permite de manera genérica extraer todos los parámetros que pueda estar recibiendo la función. Una vez sabemos el número de parámetros que nos han enviado en tiempo de ejecución, podemos invocar a la función específica que tiene el código a ejecutar cuando se recibe ese número concreto de parámetros.

Viendo el código del siguiente ejemplo y leyendo sus comentarios podremos entender mejor esta técnica.

```
class jugador
{
    private $nombre;
    private $equipo;
    function __construct()
    {
        //obtengo un array con los parámetros enviados a la función
        $params = func_get_args();
        //saco el número de parámetros que estoy recibiendo
        $num_params = func_num_args();
        //cada constructor de un número dado de parámtros tendrá un nombre de
        función
        //atendiendo al siguiente modelo __construct1() __construct2()...
        $funcion_constructor ='__construct'.$num_params;
        //compruebo si hay un constructor con ese número de parámetros
        if (method_exists($this,$funcion_constructor)) {
            //si existía esa función, la invoco, reenviando los parámetros que recibí en el constructor original
            call_user_func_array(array($this,$funcion_constructor),$params);
        }
    }
    //ahora declaro una serie de métodos constructores que aceptan diversos números de
    parámetros
    function __construct0()
    {
        $this>__construct1("Anónimo");
    }
    function __construct1($nombre)
    {
        $this>__construct2($nombre, "Sin equipo");
    }
        function __construct2($nombre, $equipo)
    {
        $this>nombre = $nombre;
        $this>equipo = $equipo;
    }
}
```

Podrían existir otros mecanismos para obtener sobrecarga de constructores en PHP, pero éste que hemos visto creo que es el más limpio y el que mejor respeta la filosofía de la Programación Orientada a Objetos.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *12/11/2013*
Disponible online en *http://desarrolloweb.com/articulos/sobrecarga-constructores-php.html*

Destructores en PHP 5

Explicación de los destructores en PHP5 y ejemplos de funcionamiento.

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.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *14/12/2004*
Disponible online en *http://desarrolloweb.com/articulos/58.php*

Modificadores de acceso a métodos y propiedades en PHP5

Son los Public, Protected y Private, que pueden conocerse porque ya se utilizan en otros lenguajes orientados a objetos.

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.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *17/01/2005*
Disponible online en *http://desarrolloweb.com/articulos/58.php*

Herencia en PHP 5

Los mecanismos de herencia son una de las herramientas fundamentales que disponen los desarrolladores en la programación orientada a objetos. Vemos cómo funciona en PHP 5.

La herencia en PHP5

Explicamos la herencia en PHP 5, un proceso por el cual los objetos pueden heredar las características de otros, de modo que se pueden hacer objetos especializados, basados en otros más generales.

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.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *16/06/2005*
Disponible online en *http://desarrolloweb.com/articulos/58.php*

La herencia en PHP 5, Segunda parte

Continuamos con los mecanismos de herencia en PHP5. Creamos clases que heredan de otra clase y aprendemos a sobrescribir métodos.

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

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *17/06/2005*
Disponible online en *http://desarrolloweb.com/articulos/58.php*

La herencia en PHP5, Tercera parte

Creamos otras clases a partir de una clase padre, para continuar con nuestro ejemplo de videoclub.

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

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *17/06/2005*
Disponible online en *http://desarrolloweb.com/articulos/58.php*

Programación orientada a objetos en PHP 5 por la práctica

Continuamos mostrando muchas de las prácticas habituales que llevan a cabo los programadores con la orientación a objetos. Vemos cómo se implementa todo a través de ejemplos en PHP 5.

Los atributos de los objetos pueden ser otros objetos

Los atributos o propiedades de los objetos pueden ser de cualquier tipo, incluso 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 , 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.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *04/07/2005*
Disponible online en *http://desarrolloweb.com/articulos/58.php*

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

Continuación del artículo sobre las propiedades de los objetos.

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

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *08/07/2005*
Disponible online en *http://desarrolloweb.com/articulos/58.php*

Repasando la creación de clases

Para afianzar los conocimientos sobre programación orientada a objetos vamos a concluir por ahora la creación de nuestro videoclub con PHP5.

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.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *18/07/2005*
Disponible online en *http://desarrolloweb.com/articulos/58.php*

Orientación a objetos avanzada

Veremos a continuación algunos conceptos, ya más avanzados, de la programación orientada a objetos en PHP 5, como son las clases y métodos abstractos y las interfaces.

Métodos y clases abstractos en PHP 5

Conoceremos lo que es una clase abstracta y los métodos abstractos. Cómo se definen y en qué situaciones se deben utilizar.

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.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *01/08/2005*
Disponible online en *http://desarrolloweb.com/articulos/58.php*

Interfaces en PHP 5

Vemos lo que son las interfaces, para qué se utilizan y cómo trabajar con ellas en PHP5.

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.

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.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *10/10/2005*
Disponible online en *http://desarrolloweb.com/articulos/58.php*

Seguridad Php (I)

PHP es una lengua muy fácil a aprender, y muchos programadores lo aprenden como manera de agregar interactividad a sus Sitio Web.

Desafortunadamente, eso significa a menudo los programadores de PHP, especialmente ésos más nuevos al desarrollo web, cometen ciertos riesgos de seguridad y desaprovechan el potencial que sus usos pueden contener. Aquí están algunos de los problemas mas comunes de seguridad y cómo evitarlos.

Regla número uno: Nunca, confiar en los usuarios

Nunca debes confiar en que los usuarios te van a mandar los datos que tu esperas. Mucha gente responde a esto con algo como“Oh, nadie estaría interesado en mi sitio”. Esta afirmación no podría ser mas incorrecta , siempre hay un usuario malévolo que quiere explotar un agujero de seguridad ademas los problemas pueden presentarse fácilmente debido a un usuario que hace algo mal inintencionalmente.

Por todo esto la regla de todo desarrollador web tiene que ser "Nunca, confiar en los usuarios" . Asumir que cada pieza de datos que tu sitio recoge de un usuario puede convertirse en un agujero de seguridad, siempre. Si la seguridad de tu sitio web es importante para ti, este un buen puntopara comenzar a aprender. Sería conveniente tener “una hoja de seguridad para PHP” al lado de tu escritorio con los puntos mas importantes en texto negrita grande.

Variables globales

En muchas lenguajes debes crear explícitamente un variable para utilizarlas. En PHP, hay una opción, las “register_globals”, que puedes fijar en php.ini y que permite que utilices variables globales.

Considera el código siguiente:

```
if ($password == "my_password") {

$authorized = 1;

}

if ($authorized == 1) {

echo "Mis cosas importantes ";

}
```

A muchos de vosotrosos puede parecer que este código esta funcionando perfectamente. ¿Sin embargo, si un servidor tiene “register_globals” encendidos, entonces simplemente agregando”? authorized=1 " al URL dará a cualquier persona el acceso libre a exactamentelo que no quisieras que todo el mundo viera. Éste es uno de los problemas mas comunes de la seguridad de PHP.

Afortunadamente, esto tiene un par de soluciones simples y posibles. La primera, y quizás la mejor, es fijar desactivar “register_globals”. La segunda es asegurarse de que utilizas solamente las variables que has fijado explícitamente tú mismo. En el ejemplo anterior, eso

significaría la adición “$authorized = 0; ” al principio de la escritura:

```
$authorized = 0;

if ($password == "my_password") {

$authorized = 1;

}

if ($authorized == 1) {

echo "Lots of important stuff.";

}
```

Mensajes de error

Los mensajes de error son una herramienta muy útil para los programadores y hackers. Un desarrollador los necesita para detectar bugs. Un hacker puede utilizarlos para descubrir todas las clases de información sobre un sitio, desde la estructura del directorio del servidor a la información de la conexión de la base de datos.En PHP para evitar esto puedes utilizar .htaccess o php.ini, fijando “error_reporting” a “0”.

Este artículo es obra de *Manu Gutierrez*
Fue publicado por primera vez en *28/08/2007*
Disponible online en *http://desarrolloweb.com/articulos/seguridad-php-i.html*

Seguridad Php (II)

Una de las ventajas más grandes de PHP es la facilidad con la cual puede comunicarse con las bases de datos, lo más normal con MySQL.

Mucha gente hace el uso excesivo de esto, y muchos grandes sitios, confía en las bases de datos para funcionar.

Sin embargo, con esa ventaja hay problemas suficientemente grandes en la seguridad a los que tendras que hacer frente. Afortunadamente, hay un montón de soluciones. El peligro más común de seguridad al que debes de hacer frente es cuando un usuario utiliza un fallo para poder atacar directamente al servidor de bases de datos con sentencias SQL.

Utilicemos un ejemplo común. Muchos sistemas utilizan un codigo muy parecido a este para comprobar el usuario y la contraseña pudiendose hacer todas las combinaciones válidas del usuario y de su contraseña, por ejemplo para controlar el acceso a un área de administración:

```
$check = mysql_query("SELECT Username, Password, UserLevel FROM Users WHERE Username = '".$_POST['username']."' and Password =
```

```
'".$_POST['password']."'");
```

¿Te parece familiar?. Y parece que no podría hacer mucho daño. Pero digamos por un momento que introduzco el siguiente ”usuario ” en el formulario:

```
'0 1=1 #
```

La pregunta que va a ser ejecutada sería esta:

```
SELECT Username, Password FROM Users WHERE Username = '' OR 1=1 #' and Password = ''
```

La almohadilla (#) le dice aMySQL que todo que le sigue es un comentario y que no debe de hacerle caso. Ejecutará SQL hasta ese punto. Despues 1 es igual a 1, SQL devolverá todos los usuarios y contraseñas de la base de datos. Y como la primera combinación del usuario y de contraseña en la mayoría de las bases de datos es la de el administrador, la persona que incorporó simplemente algunos símbolos en un formulario ahora entra como administrador de la Web , con los mismos privilegios que tendría si supiera realmente el usuario y la contraseña.

Con una poca de creatividad, este agujero de seguridad se puede explotar aun más lejos, permitiendo que un usuario cree su propia cuenta , lea números de las tarjetas de crédito o simplemente vacie la base de datos.

Afortunadamente, este tipo de vulnerabilidad es bastante fácil de solucionar. Comprobando si hay algun caracter raro cuando el usuario introduce los datos, y quitándolos o neutralizandos, podemos evitar que cualquier persona utilize su propio código del SQL en nuestra base de datos. La función que sigue sería la adecuada:

```
function make_safe($variable) {

$variable = addslashes(trim($variable));

return $variable;

}
```

Ahora debemos modificar nuestra consulta. En vez de usar variables _POST como en la consulta de arriba, ahora utilizamos todos los datos del usuario con la función make_safe, dando por resultado el código siguiente:

```
$username = make_safe($_POST['username']);

$password = make_safe($_POST['password']);

$check = mysql_query("SELECT Username, Password, UserLevel FROM Users WHERE Username = '".$username."' and Password = '".$password."'");
```

Ahora, si un usuario incorporó los datos anteriormente citados, la consulta será la siguiente, que es totalmente inofensiva. La consulta siguiente seleccionará de una base de datos los registros donde el usuario es igual a "\ 'O o 1=1 #".

```
SELECT Username, Password, UserLevel FROM Users WHERE Username = '\' OR 1=1 #' and Password = ''
```

Ahora, a menos que tengas un usuario con un nombre muy inusual y una contraseña en blanco, tu malévolo atacante no podrá hacer ningún daño en tu sitio Web. Es importante comprobar todos los datos pasados a tu base de datos. Las cabeceras de HTTP enviados por elusuario pueden ser falsificadas. Su dirección de remitente tambien puede ser falsificada. No confies en los datos enviados por el usuario, y tu y tu sitio estareis a salvo.

Este artículo es obra de *Manu Gutierrez*
Fue publicado por primera vez en *10/09/2007*
Disponible online en *http://desarrolloweb.com/articulos/seguridad-php-ii.html*

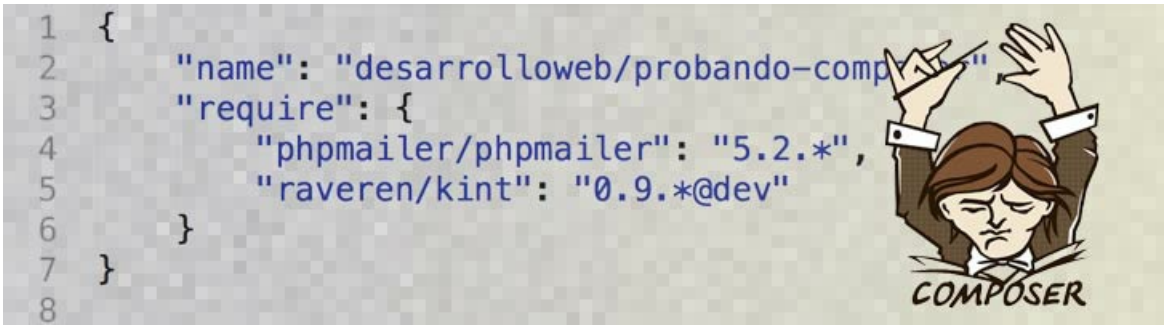
Composer, gestor de dependencias para PHP

Composer es una herramienta imprescindible para los desarrolladores en PHP, que permite gestionar de una manera ágil las dependencias de un proyecto.

Composer es un gestor de dependencias en proyectos, para programación en PHP. Eso quiere decir que nos permite gestionar (declarar, descargar y mantener actualizados) los paquetes de software en los que se basa nuestro proyecto PHP. Se ha convertido en una herramienta de cabecera para cualquier desarrollador en este lenguaje que aprecie su tiempo y el desarrollo ágil.

¿Empiezas un nuevo proyecto con PHP? echa un vistazo antes a Composer porque te puede ayudar bastante en el arranque y gracias a él podrás resumir muchas de las tareas de mantenimiento de las librerías de terceros que estés usando.

https://getcomposer.org



En este artículo te resumimos los detalles para entender qué es Composer, cómo funciona y para instalarlo en tu sistema. Primero comenzaremos explicando qué es un gestor de dependencias, luego veremos cómo funciona Composer para darnos cuenta qué aspectos de

nuestro día a día nos va a simplificar. Por último veremos cómo instalarlo y cómo usar librerías gestionadas con Composer en nuestro código PHP.

Por qué un gestor de dependencias

Cuando comienzas un proyecto en PHP, ya de cierta complejidad, no te vale solo con la librería de funciones nativa de PHP. Generalmente todos usamos alguna que otra librería de terceros desarrolladores, que nos permite evitar empezar todo desde cero. Ya sea un framework o algo más acotado como un sistema para debug o envío de email, validación de formularios, etc., cualquier cosa que puedas necesitar ya está creada por otros desarrolladores. Si no la estás usando ninguna librería posiblemente estés perdiendo tu precioso tiempo, pero eso es otra discusión.

De modo que, al comenzar el proyecto hasta ahora teníamos que ir a la página de cada uno de los componentes de software que queríamos usar, descargarlos, copiarlos en la carpeta de nuestro proyecto, etc. No solo eso, cuando estamos en mitad del desarrollo, o ya en producción, y nos cambian la versión de la librería, tenemos que volverla a descargar manualmente, actualizar los archivos, etc. Nadie se había muerto por hacer todo ese tipo de tareas de configuración y mantenimiento, pero no cabe duda que nos llevan un tiempo.

Todo eso sin contar con que ciertos softwares, como un framework como Symfony, dependen a su vez de muchas otras librerías que tendrías que instalar a mano y a su vez, mantener actualizadas.

Los gestores de paquetes nos ayudan para resumir las tareas de descarga y mantenimiento de las versiones del proyecto para que estén siempre actualizadas. Ya existían en otros lenguajes de programación y nos resultaban especialmente útiles como npm en NodeJS. Ahora los desarrolladores de PHP también contamos con esta herramienta gracias a Composer.

Cómo funciona Composer

Composer nos permite declarar las librerías que queremos usar en un proyecto. Su uso es extremadamente simple, lo que anima a cualquier persona a usarlo, sea cual sea su nivel técnico.

Para beneficiarnos del workflow que nos propone Composer simplemente tenemos que escribir un archivo de configuración en el que indicamos qué paquetes vamos a requerir. El archivo es un simple JSON en el que indicamos cosas como el autor del proyecto, las dependencias, etc.

El archivo JSON debe tener un nombre específico: composer.json

A continuación tienes un ejemplo de JSON donde declaramos varios parámetros de nuestra aplicación.

```
{
    "name": "desarrolloweb/probando-composer",
    "require": {
```

```
        "phpmailer/phpmailer": "5.2.*",
    }
}
```

Luego nos pondremos a desgranar este código para que se entienda cada una de sus partes, así como veremos qué otra información podemos colocar en este JSON. La idea es ver lo sencillo que es declarar qué librerías o software estás utilizando y con ello dejar nuestro proyecto listo para la "magia" de Composer.

Una vez tenemos definidas las dependencias en nuestro proyecto debemos instalarlas. Esto lo conseguimos con un simple comando en el terminal en el que le pedimos a Composer que las instale:

```
composer install
```

> **Nota:** Ese comando puede variar según la instalación que tengas en tu sistema de Composer. Especificaremos en un futuro artículo diversas situaciones en las que tengamos que generar variantes de este mismo comando. De momento nos vamos a quedar en una presentación de Composer, pero en seguida nos ponemos a aprender en detalle aquí en DesarrolloWeb.com

Lanzado ese comando Composer se encargará de ir a los repositorios de paquetes de software y descargar aquellas librerías mencionadas, copiándolas en la carpeta de tu proyecto.

Una vez finalizado el proceso en tu consola de comandos podrás encontrar en la carpeta de tu proyecto un directorio llamado "vendor" donde estarán las librerías declaradas. Ya solo nos queda hacer los includes para que estén disponibles en tus aplicaciones y para ello también nos ayuda Composer.

Simplemente tendremos que hacer un único include o require en nuestro código y todas las librerías estarán disponibles para usar.

```
require 'vendor/autoload.php';
```

Packagist

Para terminar de convencerte y contarte la introducción completa, debes echar un vistazo a Packagist. Se trata del repositorio de paquetes que son instalables por medio de Composer.

En la página de Packagist encontrarás un buscador que te puede dar una idea de la cantidad de material que encuentras disponible para usar en cualquier proyecto PHP.

https://packagist.org/

Simplemente busca por cualquier concepto que te interese, como email, template, wysiwyg, etc. Verás que te aparecen varias opciones clasificadas por popularidad, descargas, etc. Además sobre cada paquete encuentras información y el código necesario para declarar tu dependencia en el JSON de Composer.

Conclusión

Espero que con lo que hemos visto hasta ahora te haya llamado la atención esta herramienta. La verdad es que es muy útil y como decimos, una vez comienzas a usarla te das cuenta de todo el trabajo que te quita del medio, no solo en la descarga de los paquetes, sino también en las actualizaciones de las librerías con el comando "composer update" que veremos más adelante.

Sabemos que nos hemos dejado muchas cosas en el tintero, como el proceso de instalación y el detalle del JSON, pero lo veremos ya en próximos artículos. De momento queríamos presentarte el gestor de dependencias y que sepas por qué los desarrolladores de PHP lo hemos adoptado con tanto entusiasmo.

En el próximo artículo vamos a detallar el proceso de instalación de Composer.

Además de los próximos artículos donde vamos a explicarlos los detalles del flujo de trabajo con Composer para la gestión de dependencias, vamos a presentaros ahora un vídeo de nuestro canal de Youtube donde encuentras resumidos algunos de los pasos básicos de uso de Composer.

Para ver este vídeo es necesario visitar el artículo original en:
http://desarrolloweb.com/articulos/composer-gestor-dependencias-para-php.html

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *17/10/2014*
Disponible online en *http://desarrolloweb.com/articulos/composer-gestor-dependencias-para-php.html*

Namespaces en PHP

Guía para el trabajo con espacios de nombres en PHP. Utilidades y manera de trabajar con ellos: declararlos y usarlos.

Los espacios de nombres son una de las utilidades que han aparecido en PHP 5, en la versión PHP 5.3. Ya tienen un tiempo de vida relativamente grande y deberían estar disponibles en tu servidor de PHP. Gracias a las facilidades que nos ofrecen deberías comenzar a usarlos, así que te proponemos seguir esta guía para conocer su funcionamiento.

Básicamente sirven de contenedores para el código de PHP, de modo que cuando creemos elementos del lenguaje como constantes, funciones o clases, se queden en un ámbito más restringido, evitando colisiones o conflictos de nombres con otros elementos que puedas crear tú más adelante, otras personas de tu equipo o incluso otros desarrolladores.

Lo primero sería aclarar que en la página de PHP tienen una estupenda guía para comenzar con los namespaces. Además está traducida al español, por lo que te recomendamos tenerla a mano: Namespaces en la documentación oficial.

En este artículo pretendemos aclarar algunas cosas básicas y otras adicionales, a la vez que te ayudamos con nuevos ejemplos y explicaciones que puedan complementar la documentación oficial.

Para qué sirven los espacios de nombres

Como hemos dicho, sirven para organizar el código, de manera que los nombres que nosotros reservemos a la hora de crear clases o funciones no entren en conflicto con los que hayan podido, o puedan en el futuro, crear otras personas.

Por ejemplo, imagina que tienes una clase que se llama "Seguridad" que se encarga de validar usuarios en tu aplicación para saber si están correctamente autenticados en el sistema. Es posible que otras personas, ya sea en tu mismo proyecto, o en otras librerías que puedas llegar a utilizar, hayan imaginado ese mismo nombre para otra clase, por ejemplo una clase que se encarga de impedir que se acceda a los recursos situados dentro de una carpeta sin los permisos necesarios.

Como no puede haber en el sistema dos clases con el mismo nombre, si necesitases trabajar con ambas clases dentro de un proyecto, encontrarías que PHP te da un mensaje de error. Pero bueno, esto nos puede ocurrir con otros elementos más sencillos, como funciones. Imagina que tienes una función validar_entero() y más adelante otra persona se le ocurre el mismo nombre para una función en otra parte del código, ambas funciones entrarían en conflicto, pues deberían tener nombres distintos.

Hasta la creación de los namespaces, para evitar colisiones de nombres, los desarrolladores estaban obligados a ser un poco imaginativos con los nombres que ponían a sus piezas de software, creando nombres extra largos para sus clases, evitando así que otras personas en el futuro pudieran usar esos mismos nombres.

Obviamente, esa solución no era la mejor, así que PHP incorporó los espacios de nombres, que nos permiten reservar nombres para las funciones, clases, interfaces, traits, constantes, etc, que solo tienen validez en cierto namespace. Al crear una librería, colocarías tus funciones dentro de un espacio de nombres, de manera que otros desarrolladores puedan usar los mismos nombres de piezas de software sin que colisionen, por estar en distintos espacios de nombres.

Símil de los namespaces y el sistema de ficheros de tu ordenador

Creo que este símil hace muy fácil entender qué son los namespaces y cómo funcionan, pues nos apoyamos en algo que todo el mundo usamos, que es el sistema de carpetas y archivos de tu disco duro.

En tu ordenador los archivos están dentro de carpetas y dentro de éstas hay a su vez otras carpetas que tienen otros archivos. Una carpeta de tu ordenador se comporta como si fuera un espacio de nombres o namespace de PHP.

Por ejemplo, no puede haber dos archivos con el mismo nombre en la misma carpeta o directorio. Pero sí puede haber dos archivos con el mismo nombre que están en distintos directorios.

Es tan sencillo como eso, los espacios de nombres son como carpetas de tu disco duro, que en lugar de contener archivos contienen funciones, clases, etc. Esas clases pueden tener cualquier nombre sin colisionar con otros códigos que estén en un espacio de nombres distinto.

Cómo declarar un espacio de nombres en PHP

El concepto, como has podido ver, es muy sencillo de entender. Ahora solo falta ver la sintaxis con la que PHP define los espacios de nombres.

Cuando estás creando un archivo que tiene una serie de código que quieres situar en un espacio de nombres, tienes que indicarlo al principio del código del fichero.

Indicas el namespace de esta manera:

```
<?php
namespace Deswebcom;
```

A partir de esa línea puedes escribir cualquier tipo de código, donde cada uno de los elementos declarados se quedarán en el espacio de nombres definido.

Tienes que asegurarte que:

1) La declaración del espacio de nombres sea la primera línea de tu código fuente. Y como código fuente entendemos tanto código PHP como código HTML.

Por ejemplo, esto no funcionaría:

```
<?php
echo "probando";
namespace Deswebcom;
```

Recibirás un error fatal como este: Fatal error: Namespace declaration statement has to be the very first statement in the script.

Del mismo modo, tampoco podrás hacer algo como esto:

```
<html>
    <head>
<?php
namespace Deswebcom;
```

2) En un espacio de nombres se pueden engloban:

- Constantes
- Funciones
- Clases y otros elementos de programación orientada a objetos, como interfaces, traits o clases abstractas.

Por tanto las variables globales que creas en un namespace se van al ámbito global. Es decir, si dentro de un archivo donde has declarado estar en un espacio de nombres usas variables sueltas (sin ser locales a funciones o métodos, osea, variables globales de toda la vida), esas variables se quedan en el ámbito global.

Siguiendo con la declaración de nuestro primer namespace, podemos ver el código completo de un archivo con su espacio de nombres.

```
<?php
namespace Deswebcom;

const PI = 3.14;
function avisa(){
    echo "Te estoy avisando";
}
class MiClase{
    public function probando(){
    echo "Esto es una prueba";
    }
}
```

Ese código lo meterás en un archivo de tu proyecto, por ejemplo "primer-namespace.php".

> **Nota:** La localización donde lo pongas ese archivo, o su nombre en si, es indiferente. Es decir, por el hecho de haber definido un namespace no estás obligado a colocar el código fuente en una ruta determinada. Aunque como veremos más adelante, muchos programadores con base en las buenas prácticas colocan los archivos del espacio de nombres en carpetas que tienen los mismos nombres que el namespace declarado.

Cómo usar un espacio de nombres

Ahora que ya tenemos nuestro primer namespace vamos a usarlo en otro archivo aparte.

Primero tendrás que incluir el archivo (aquel donde tienes el código de tu namespace), para poder usarlo.

```
include "primer-namespace.php";
```

Ahora PHP conoce ese código, está en un namespace y su uso será distinto que si hubieses colocado todos esos miembros en el ámbito global, pero al menos ya puedes usarlo. Ahora bien, la manera de usar aquellos elementos del namespace puede tener diversas alternativas que veremos a continuación:

a) Colocar la ruta completa del namespace cuando nos referimos a sus elementos: Para hacer uso de los miembros de ese espacio de nombres usaremos la ruta completa dentro del namespace declarado.

Por ejemplo, en el namespace se declaró la constante PI, a la que podemos acceder de esta manera:

```
echo Deswebcom\PI;
```

Como puedes comprobar, comenzamos haciendo referencia al namespace, por su nombre, seguido de una contrabarra y el miembro al que quieres acceder, en este caso la constante PI.

De manera similar podríamos acceder a las funciones o las clases, a través de la mención del espacio de nombres donde están englobadas:

```
Deswebcom\avisa();
```

b) Declarar el uso de un miembro del namespace: Otra alternativa útil es definir que vas a usar un miembro de un namespace, esto es, una constante, función, clase o similares para, a partir de ese momento, poder usarla como si fuera un miembro declarado de manera global.

```
use const Deswebcom\PI;
echo PI;

use function Deswebcom\avisa;
avisa();

use Deswebcom\MiClase;
$prueba = new MiClase();
$prueba->probando();
```

En este código primero se declara que se va a usar una constante del espacio de nombres. Luego se declara que se va a usar una función y por último se declara que se va a usar una clase. A partir de ahí podemos hacer referencia a esos miembros sin referirnos a la ruta del namespace donde han sido creados, como si esa constante, función o clase que se hubiera

declarado de manera global.

Fíjate que al usar una constante tenemos que informar "use const" y si es una función "use function", mientras que si es una clase usamos "use" a secas.

> **Nota:** La posibilidad de declarar el uso de una función o constante de un namespace está disponible solo a partir de PHP 5.6.

Conclusión

Con esto ya conoces las bases sobre los espacios de nombres en PHP. Como introducción creo que está bastante bien y que ya tienes información suficiente para usarlos sin problemas. No obstante, el lenguaje PHP nos permite diversas otras alternativas de uso que estudiaremos en futuros artículos.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *23/06/2015*
Disponible online en *http://desarrolloweb.com/articulos/uso-namespaces-php.html*

Organizar los namespaces de PHP en niveles

Alternativas de uso y organización del código PHP por medio de los espacios de nombres: subespacios de nombres, variantes de acceso...

En un artículo anterior pudimos entender el concepto de namespace en PHP y su uso básico. Ahora vamos a seguir explorando posibilidades del lenguaje para sacarle aún más partido y conocer algunas posibilidades un poco más avanzadas.

Vamos a tocar en este texto varios puntos de diversa índole, como la posibilidad de crear subespacios de nombres, organizar los archivos con código, definir un espacio de nombres en varios ficheros diferentes y cosas de este estilo. Todo orientado a que puedas organizar mejor tu código, sacando partido a los namespaces de PHP.

Crear Sub-namespaces

Es habitual que los desarrolladores creen espacios de nombres dentro de otros espacios de nombres. Por ejemplo, todos los namespaces de una empresa o desarrollador pueden comenzar con su "nick" y luego dentro puedes tener espacios de nombres para cada tipo de utilidad que estás desarrollando.

Imagina que DesarrolloWeb.com se dedicase a fabricar librerías con código PHP, pues todas las librerías podrían estar dentro del espacio de nombres "Deswebcom". Luego podríamos tener librerías que se encargan del trabajo con sesiones, cookies, bases de datos, etc. Podrían estar en sub-namespaces como Deswebcom\Sesiones, Deswebcom\Cookies, Deswebcom\BBDD. Ya dentro de Deswebcom\Sesiones podrías tener una serie de clases, funciones y constantes, o a su vez otros espacios de nombres donde organizas otra serie de librerías. El orden es el que tú necesites y esta organización resulta muy popular. De hecho la puedes ver reproducida en numerosos paquetes de librerías o en la organización de las clases de un framework potente.

La definición de los namespaces y sub-namespaces es prácticamente la que hemos comentado en artículos anteriores. Comienzas el código indicando tu namespace (recuerda que debe ser la primera instrucción del archivo).

```
<?php
namespace Deswebcom\Galaxias\Andromeda;
```

Como puedes apreciar, la jerarquía de namespaces se indica con una barra invertida (contrabarra). Puede llegar a tantos niveles como necesites y en cada nivel puedes tener miembros del espacio de nombres. Es decir, en el namespace "Deswebcom" puedo tener clases, funciones, constantes, traits, interfaces y otros namespaces. Dentro de los otros namespaces puedo tener otra vez lo mismo: clases, funciones, otros namespaces, etc.

Organizar archivos con namespaces

La organización de tus archivos en espacios de nombres es muy flexible. Hay lenguajes donde a la hora de definir estructuras similares a los namespaces te obligan a que el código resida en carpetas del mismo nombre. En PHP no es así.

Puedes definir namespaces de cualquier profundidad en archivos que tengas en cualquier carpeta. Puedes usar el mismo namespace en diversos archivos, puedes incluso crear varios espacios de nombres en el mismo archivo.

Lógicamente, aunque PHP te permita muchas alternativas, es interesante que tengas tus propias costumbres y una buena práctica sería crear una estructura de carpetas similar a la estructura de espacios de nombres que estás generando. Eso te ayudará a saber dónde están los archivos que tienen el código de cada espacio de nombres.

Cuando trabajas con jerarquías de espacios de nombres puedes encontrar usos como estos que vamos a relatar.

Mismo espacio de nombres en dos archivos diferentes

Si tienes dos archivos que trabajan dentro de un mismo namespace, entonces, las funciones que crees en ambos espacios de nombres estarán disponibles sin declarar que vas a usar ese espacio de nombre.

Puede parecer obvio, pero veamos un ejemplo de dos archivos que trabajan sobre el mismo espacio de nombres. Como primer archivo:

```
<?php
namespace Deswebcom\Galaxias;

function observar($a){
    echo "Observando la galaxia $a";
}
```

Como segundo archivo:

```
<?php
namespace Deswebcom\Galaxias;

//como este archivo está en el mismo espacio de nombres que el anterior
//soy capaz de invocar una función declarada en este namespace sin indicar su ruta
observar("Centaurus A");
```

Acceso "relativo" a elementos de otros espacios de nombres

Otra cosa que puede surgir es que desde un espacio de nombres quieras acceder a miembros de otro espacio de nombres dependiente.

En este caso ocurre como cuando usas tu sistema de archivos, siguiendo el símil de las carpetas de un disco duro de tu ordenador. Estando situados en un directorio de tu disco duro, pueden surgir tres escenarios:

1. Acceso a archivos indicando solo su nombre. Es algo que sueles hacer cuando accedes a un fichero que está dentro de la misma carpeta donde te encuentras. En este caso no necesitamos indicar la ruta completa de esos archivos, ya que estamos en el mismo directorio.
2. Acceso a archivos indicando una ruta relativa. Es el caso que suele ocurrir cuando accedes a archivos que están dentro de subcarpetas de la carpeta donde te encuentras. En este caso como posibilidad podríamos definir la ruta relativa al archivo que queremos acceder, desde la carpeta donde estamos. Osea, si tengo un subdirectorio llamado "folder" y dentro de él hay un archivo "file.txt", podríamos acceder por "folder/file.txt".
3. Acceso indicando una ruta absoluta. Es el caso típico de acceso a archivos que están en otras rutas no dependientes del directorio donde me encuentro. En este caso lo común y más cómodo es referirnos a los archivos con su ruta absoluta, por ejemplo

"/carpeta/subcarpeta/archivo.txt".

Esas mismas situaciones se pueden dar en el uso de namespaces. La primera, acceso a archivos indicando solo su nombre la la hemos visto en el punto anterior, cuando comentábamos que podíamos acceder a funciones definidas en el mismo espacio de nombres como si fueran funciones globales.

La segunda situación la podemos ver en un ejemplo a continuación. Por ejemplo piensa en este espacio de nombres Deswebcom\Galaxias\Andromeda.

```
<?php
namespace Deswebcom\Galaxias\Andromeda;

function localizar(){
    echo "Soy la galaxia Andromeda y estoy a 3 millones de años luz de la tierra";
}
class Estrella{
    public static function pertenece(){
        echo "Pertenezco a a la galaxia Andromeda";
    }
}
```

Ahora, si estamos en un espacio de nombres como "Deswebcom\Galaxias", no necesitamos toda la ruta absoluta para llegar a los miembros de "Deswebcom\Galaxias\Andromeda":

```
<?php
namespace Deswebcom\Galaxias;
include "andromeda.php";

Andromeda\localizar();
Andromeda\Estrella::pertenece();
```

La tercera posibilidad es también fácil de entender y funcionará siempre, puesto que si indicamos la ruta completa de toda la jerarquía de namespaces, no importa en qué espacio de nombres estamos trabajando. Es decir, indicando el espacio de nombres completo, siempre vamos a poder referirnos a cualquier elemento, independientemente de nuestra posición.

```
Deswebcom\Galaxias\Andromeda\localizar();
Deswebcom\Galaxias\Andromeda\Estrella::pertenece();
Deswebcom\Galaxias\observar("Via Lactea");
```

En este caso también admitiría comenzar por una contrabarra toda la ruta jerárquica para el acceso a los namespaces, indicando que el primer namespace "Deswebcom" es de primer nivel.

```
\Deswebcom\Galaxias\Andromeda\localizar();
\Deswebcom\Galaxias\Andromeda\Estrella::pertenece();
\Deswebcom\Galaxias\observar("Via Lactea");
```

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *14/07/2015*
Disponible online en *http://desarrolloweb.com/articulos/organizar-namespaces-php-niveles.html*

Importar y apodar namespaces en PHP

Alternativas para importar namespaces o miembros de namespaces, con alias o sin él. Reglas para importar y cómo acceder al ámbito global para resolver nombres usados en espacios de nombres.

Este artículo continúa con la serie de tutoriales dedicados a conocer y dominar los namespaces de PHP. Es el tercero que escribimos sobre este asunto, así que te recomendamos empezar la lectura por la introducción a los namespaces de PHP.

Ten en cuenta que aquí damos por hecho que sabes ciertas cosas y además usamos código de espacios de nombres que fueron definidos en artículos anteriores.

Importar con alias

A la hora de importar código desde espacios de nombres es posible asignarles un alias para referirse a ellos más adelante. Esto nos puede servir para varias situaciones, como por ejemplo darle una manera resumida de acceder a una clase dentro de archivo o para asignarle un nombre diferente, en el caso de ya estar ocupado el nombre original de esa clase, de modo que podríamos evitar colisiones de nombres en determinadas ocasiones.

```
use Deswebcom\Galaxias\Andromeda\Estrella as E;
```

Así estaríamos asignando el Alias "E" a la clase "Estrella", que está en el namespace "Deswebcom\Galaxias\Andromeda".

A partir de ahí podríamos usar el nombre de la clase totalmente cualificado, "Deswebcom\Galaxias\Andromeda\Estrella" o el alias que acabamos de definir "E".

```
E::pertenece();
```

o bien:

```
Deswebcom\Galaxias\Andromeda\Estrella::pertenece();
```

Nota: Recuerda que también puedes declarar que vas a usar funciones o constantes de otros espacios de nombres. En ese caso la cosa cambia algo, porque tienes que mencionar "use function" o "use const".

Por ejemplo tengo este namespace.

```
<?php
namespace Midesweb;

function quien(){
    echo "Soy @midesweb";
}
```

Y en otro archivo quiero usar la función quien, también por un alias.

```
use function Midesweb\quien as q;
```

Ahora esa función está disponible con el alias "q". Luego podré invocarla así:

```
q();
```

Realmente este tema de los alias ya lo venías utilizando, aunque no lo habíamos mencionado todavía. Esto es porque cualquier sentencia "use" lo que hace realmente es crear un alias. Solamente que, si no le indicamos un alias diferente, crea ese recurso con su nombre original. Por ejemplo:

```
use Deswebcom\Galaxias\Andromeda\Estrella;
```

Es una línea de código equivalente a:

```
use Deswebcom\Galaxias\Andromeda\Estrella as Estrella;
```

Antes de acabar este punto quisiera dar dos aclaraciones, igual son cosas obvias, pero creo que podré solventar las dudas de algunos lectores:

1. Una sentencia use con un alias (o sin él, pues realmente ya sabemos que el alias siempre lo crea al declarar un "use") pone a disposición un código tan solo para el archivo .php donde hemos creado ese alias.
2. Esto se deriva de lo anterior: Si estás dentro de un espacio de nombres y declaras un "use" para crear un alias de una clase, función o cualquier otra cosa, no significa que estés agregando ese elemento al espacio de nombres en el que estás. Aquella función o clase, interfaz o lo que sea, permanece en el espacio de nombres donde fue definida y con ese "use" simplemente estás indicando que dentro de este fichero eres capaz de referirte a ese elemento por su alias.

Importar un namespace completo con alias

Otra tarea común es la de crear un alias para referirse a un espacio de nombres completo.

```
use Deswebcom\Galaxias;
```

Eso crea un alias llamado "Galaxias" para el namespace "Deswebcom\Galaxias".

Nota: Recuerda que es equivalente a haber escrito:

```
use Deswebcom\Galaxias as Galaxias;
```

Si dentro del namespace Deswebcom\Galaxias había una función llamada distancia(), entonces, gracias al alias, podríamos invocarla de esta manera:

```
Galaxias\distancia("Via Lactea", "Alfa Centauro");
```

Además, gracias al Alias podemos acceder también a namespaces que están dentro de Galaxias de una manera resumida.

```
Galaxias\Andromeda\localizar();
```

Aquí hemos pasado del namespace "Galaxias" al subnamespace "Andromeda" y a partir de él, invocado a su función localizar().

Podríamos crear ese namespace con un alias diferente si lo deseamos.

```
use Deswebcom\Galaxias as GA;
```

Entonces usaremos esa abreviación (alias) para acceder a los miembros del namespace.

```
GA\distancia("Via Lactea", "Alfa Centauro");
GA\Andromeda\localizar();
```

Importar solamente en el ámbito global

La instrucción use para importar un namespace o un miembro sólo se puede realizar desde el ámbito global de un fichero. Es decir, si intentamos importar desde una función nos dará un error "unexpected 'use' (T_USE)".

```
function importar_ilegal(){
    use Deswebcom;

    echo "Esto no lo puedes hacer, porque este no es el ámbito global";
}
```

Tampoco podrías hacerlo desde una clase:

```
class WebSite{
    use Deswebcom;
}
```

Pero bueno, en este caso el error sería diferente, porque aquí PHP estaría suponiendo que vas a usar un trait en la clase "WebSite" y no el namespace "Deswebcom".

Espacio de nombres global

Si estamos programando dentro de un espacio de nombres podremos crear cualquier miembro con cualquier nombre sin que colisione con otros miembros de otros namespaces. Esto ocurre con el ámbito global también.

Todas las funciones nativas de PHP están definidas dentro del ámbito global, por lo que en ocasiones podemos necesitar referirnos al ámbito global en vez del ámbito del namespace.

Por ejemplo, tengo el namespace "Deswebcom\Galaxia" y dentro de él he definido una función llamada explode(). En ese caso, dentro del namespace "Galaxia", explode() hará referencia al método expolde() que hemos definido en este ámbito.

```
<?php
namespace Deswebcom\Galaxias;

function explode($estrella){
    echo "La estrella $estrella va a explotar";
}

explode("Sol");
```

Ese código te devolverá "La estrella Sol va a explotar", porque se está invocando al explode() recién definido.

> **Nota:** Puedes reparar que si no estuviéramos dentro de un namespace PHP no nos habría permitido crear una función llamada explode(), porque ya existe una función nativa del API de PHP con ese nombre, que se encarga en separar partes del string en un array.

Ahora bien, si necesitas por algún motivo acceder a la función explode() de PHP, la nativa definida en el ámbito global, entonces tendría que especificar que esa función es aquella del ámbito global y no la definida en el namespace. Para ello no hay más que agregarle una contrabarra antes del nombre de la función (o cualquier otra cosa del ámbito global a la que queremos acceder).

```
$arraymarcas = \explode("|", "desarrolloweb|escuelait");
```

Conclusión

Con todo esto tienes bastante información sobre namespaces. Realmente ya sabes todo lo que necesitas, así que a partir de ahora procura usarlos.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *11/08/2015*
Disponible online en *http://desarrolloweb.com/articulos/importar-apodar-namespaces-php.html*