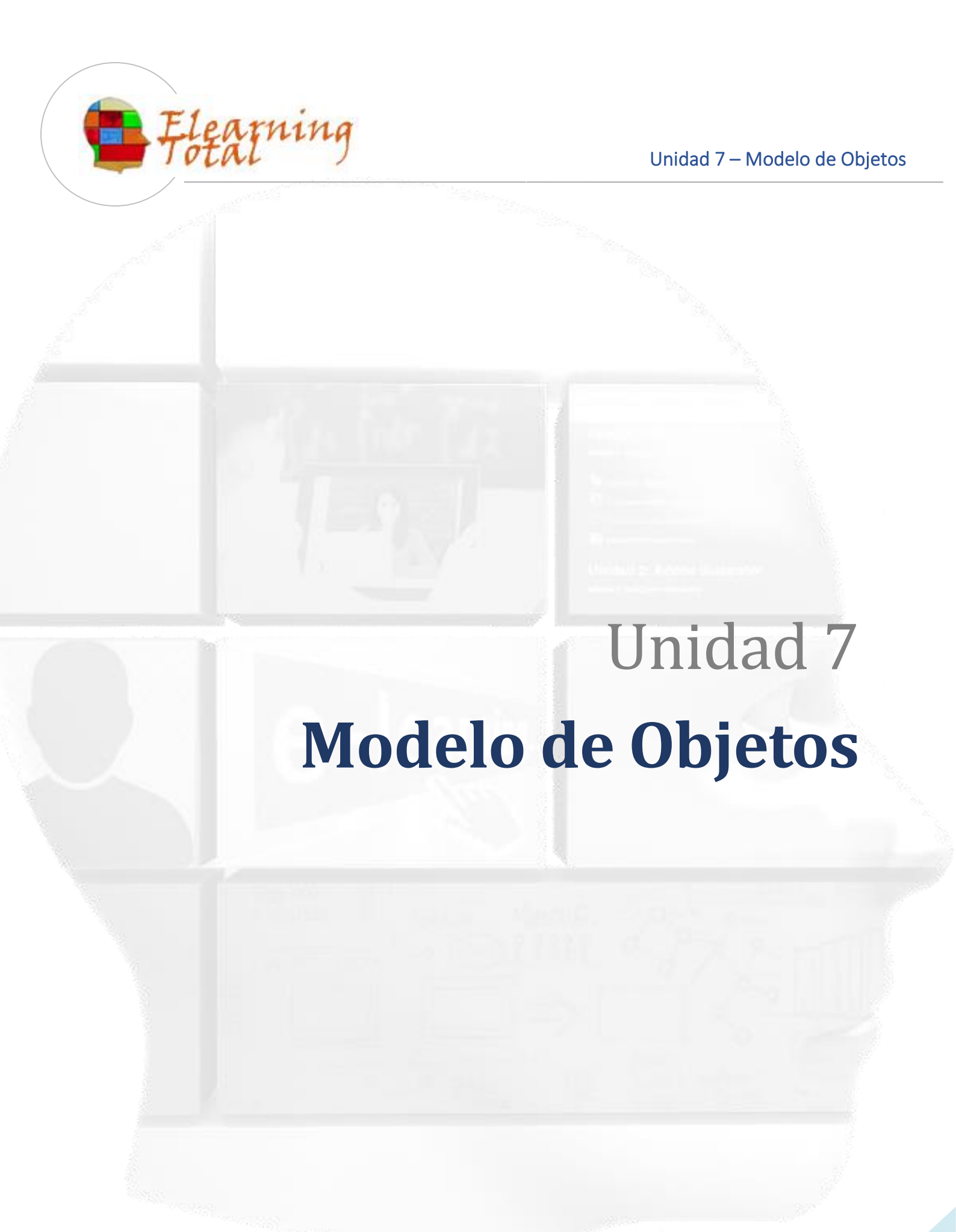

Unidad 7

Modelo de Objetos

Contacto: consultas@elearning-total.com

Web: www.elearning-total.com

Modelo de Objetos

Vamos a ver algunos conceptos más que forman parte del paradigma de programación orientado a objetos.

Polimorfismo

El polimorfismo, junto con el encapsulamiento y la herencia, forman parte de los pilares básicos de la programación orientada a objetos.

El polimorfismo, como su nombre indica, sugiere múltiples formas. En programación cuando hablamos de polimorfismo nos referimos a la capacidad de acceder a múltiples funciones a través del mismo interfaz. Es decir que un mismo identificador, o función puede tener diferentes comportamientos en función del contexto en el que sea ejecutado.

El polimorfismo es un concepto difícil de entender en un primer momento y en mi opinión PHP no nos ayuda demasiado en este aspecto, al no ser este un lenguaje de programación fuertemente tipado.

Su implementación varía en función del lenguaje de programación. En algunos casos para establecer una relación polimórfica es necesario que cada uno de los objetos implicados compartan una misma raíz, siendo entonces necesario establecer una jerarquía de clases. Este es el caso de los lenguajes de programación fuertemente tipados como Java.

Para el caso que nos ocupa vamos a estudiar el polimorfismo en PHP estableciendo una jerarquía de clases, ya que nos resultara más fácil portar el mismo ejemplo a otros lenguajes.

¿Como se implementa el polimorfismo?

A continuación veremos un ejemplo de cómo implementar una llamada polimórfica al método de un objeto.

Para ello vamos a crear una sencilla jerarquía de clases donde tendremos una clase base llamada "classPoligono" y sus respectivas clases extendidas: "classTriangulo", "classCuadrado", "classRectangulo". Cada una de estas clases tendrá un método en común que se llamará "calculo" y cuya función será la de mostrar la fórmula matemática para el cálculo del área de la figura geométrica en cuestión.

Una vez que hemos definido nuestras clases crearemos la función que se encargara de hacer la llamada polimórfica al método "calculo" cuya ejecución variara dependiendo del objeto que lo implementa.

```
<?php
/*
Empezaremos definiendo la jerarquía de clases
*/
class classPoligono
{
function calculo()
{
echo 'El área depende del tipo de polígono';
}
}

class classCuadrado extends classPoligono
{
function calculo()
{
echo 'área de un cuadrado : a=l*l<br>';
}
}

class classRectangulo extends classPoligono
{
function calculo()
{
echo 'área de un rectángulo : a=b*h<br>';
}
}

class classTriangulo extends classPoligono
{
function calculo()
{
echo 'área de un triangulo : a=(b*h)/2<br>';
}
}
/* fin definición de la jerarquía de clases */
```

```
/*
definición de la función encargada de realizar las llamadas polimórficas al método "calculo"
A destacar que en la definición de la función definimos el tipo parámetro que pasamos por
referencia, esto no es obligatorio en PHP, pero nos ayuda a entender el concepto y así poder
aplicarlo en otros lenguajes más estrictos.*/
function area(classPoligono $obj)
{
$obj->calculo();
}

/*
Creamos los objetos necesarios
*/
$cuadrado = new classCuadrado;
$rectangulo = new classRectangulo;
$triangulo = new classTriangulo;

/*
Ejecutamos la funcion encargada de realizar la llamada polimorfica
*/
area($cuadrado);
area($rectangulo);
area($triangulo);
?>
```

Conclusiones finales

Al ejecutar el ejemplo anterior vemos que la función "area" nos muestra la formula correcta en cada una de sus ejecuciones para cada tipo de figura geométrica, pese a que en su definición inicial hayamos especificado que el objeto es del tipo "classPoligono", haciendo referencia a la clase base de cada objeto. Esto sería necesario en algunos lenguajes en los cuales nos hace falta un nexo común a cada uno de los objetos, y la única forma es de utilizar la clase base común a cada uno de ellos.

Como he comentado anteriormente en PHP esta definición no sería necesario, ya que no es un lenguaje en el que sea obligatorio especificar el tipo de una variable en su definición, por lo que en este aspecto es muchos menos estricto que otros lenguajes de programación. Sin embargo, sí que sería necesario en otros lenguajes como Java mucho más estrictos.

Interfases

En el punto anterior hablamos del polimorfismo en PHP y vimos como podíamos establecer una relación polimórfica creando una jerarquía de clases. Establecimos un vínculo entre diferentes objetos a través de un elemento común, en este caso su clase base o ancestra.

Ahora hablaremos de un sistema bastante común por los programadores, para establecer un punto de unión entre objetos de diferente naturaleza, logrando el polimorfismo necesario en una determinada función.

La interfaz es el recurso ideal para la implementación del polimorfismo, ya que la interfaz es un conjunto de declaraciones de funciones o métodos sin incluir su codificación, dejando a la clase que implementa la interfaz esta tarea.

A continuación, modificaremos el ejemplo visto en el punto anterior para adaptarlo a la utilización de interfaces.

```
<?php
 /* Empezaremos definiendo la interface */
 interface Poligono
 {
 function calculo();
 }

 /* A continuacion defino las clases que implementan la
interface */
 class classCuadrado implements Poligono
 {
 function calculo()
 {
 echo 'area de un cuadrado : a=l*l<br>';
 }
 }

 class classRectangulo implements Poligono
 {
 function calculo()
 {
 echo 'area de un rectangulo : a=b*h<br>';
 }
 }
```

```
class classTriangulo implements Poligono
{
function calculo()
{
echo 'area de un triangulo : a=(b*h)/2<br>';
}
}

/* definición de la función encargada de realizar las llamadas polimórficas al método "calculo"
A destacar que en la definición de la función definimos el tipo parámetro que pasamos por referencia, esto no es obligatorio en PHP, pero nos ayuda a entender el concepto y así poder aplicarlo en otros lenguajes más estrictos. */
function area(Poligono $obj)
{
$obj->calculo();
}

/*
Creamos los objetos necesarios
*/
$cuadrado = new classCuadrado;
$rectangulo = new classRectangulo;
$triangulo = new classTriangulo;

/*
Ejecutamos la funcion encargada de realizar la llamada polimórfica
*/
area($cuadrado);
area($rectangulo);
area($triangulo);
?>
```

Conclusiones finales

Al probar el ejemplo comprobamos el comportamiento polimórfico de la función "área", mostrando diferentes resultados en función del contexto en el que se está ejecutando dicha función. Por lo tanto, hemos logrado establecer una relación polimórfica con objetos de distinta naturaleza gracias a la utilización de interfaces, sin la necesidad de establecer una jerarquía de clases.

Otras características sobre interfaces

Otras características propias de las interfaces a tener en cuenta:

- Todos los métodos definidos en una interfaz, deben ser codificados en la clase que implementa dicha interfaz.
- La clase que implemente la interfaz debe utilizar exactamente las mismas estructuras de métodos que fueron definidos en la interfaz.
- Las interfaces se pueden extender al igual que las clases mediante el operador extends.
- Una clase puede implementar diferentes interfaces.
- Una interfaz no se puede instanciar y todos sus métodos son públicos dada la propia naturaleza de la interfaz.
- Una interfaz no puede contener ni atributos, ni métodos implementados, solo declaraciones de métodos y constantes.

Clases abstractas

Las clases abstractas son similares a las clases normales en su construcción y concepto, aunque se diferencian de estas en 2 aspectos fundamentales:

- Una clase abstracta no puede ser instanciada, no podremos crear objetos a partir de ellas,
- Una clase abstracta puede incorporar métodos abstractos. Los métodos abstractos son aquellos que solo existe su declaración, dejando su implementación a las futuras clases extendidas o derivadas.

Importante

Todos los métodos declarados como abstractos, deberán pertenecer necesariamente a una clase abstracta. Es decir que una clase normal no podremos definir un método como abstracto.

Una clase abstracta tiene la misma estructura que una clase normal, solo es necesario añadir la palabra clave abstract al inicio de su declaración.

Veamos un sencillo ejemplo para comprobarlo:

```
<?php
 /* Definimos la clase abstracta */
 abstract class Poligono
```

```
{
 //A continuacion declaramos el metodo abstracto a
implementar en las clases derivadas
 abstract function calculo();
 }

 /*
 A continuacion defino las clases dereivadas que van a
extender el funcionamiento de la clase base Poligono
(abstracta)
 */
 class classCuadrado extends Poligono
 {
 function calculo()
 {
 echo 'area de un cuadrado : a=l*l<br>';
 }
 }
 class classRectangulo extends classPoligono
 {
 function calculo()
 {
 echo 'area de un rectangulo : a=b*h<br>';
 }
 }

 class classTriangulo extends classPoligono
 {
 function calculo()
 {
 echo 'area de un triangulo : a=(b*h)/2<br>';
 }
 }

 /*
 definicion de la funcion encargada de realizar las
llamadas polimorficas al metodo "calculo"
 A destacar que en la definicion de la funcion definimos
el tipo parametro que pasamos por referencia, esto no es
obligatorio en PHP, pero nos ayuda a entender el concepto y
asi poder aplicarlo en otros lenguajes mas estrictos.
```

```
*/
function area(Poligono $obj)
{
$obj->calculo();
}

/*
Creamos los objetos necesarios
*/
$cuadrado = new classCuadrado;
$rectangulo = new classRectangulo;
$triangulo = new classTriangulo;

/*
Ejecutamos la funcion encargada
de realizar la llamada polimorfica
*/
area($cuadrado);
area($rectangulo);
area($triangulo);
?>
```

Clases abstractas vs Interfaces

Analizando este ejemplo anterior, podríamos no tener muy claro dónde está la diferencia entre una clase abstracta y una interfaz, y cuando utilizar un recurso u otro, ya que aparentemente tienen la misma funcionalidad.

La principal diferencia entre ambas reside en el concepto para el que fueron concebidos. Una interfaz no es ni más ni menos que un conjunto de declaraciones a codificar en las diferentes clases que las implementan. Mientras que una clase abstracta es creada como primer paso a la creación de una jerarquía de clases. La interfaz es utilizada como punto de unión entre objetos de diferente naturaleza, mientras que la clase abstracta sirve como punto de partida para la definición de un objeto a través de una jerarquía de clases.

Existen también las correspondientes diferencias técnicas, derivadas de su concepción:

- Una interfaz no puede contener métodos o atributos implementados, mientras que en las clases abstractas sí que es posible incluirlos.

- Una clase solo puede heredar de una clase abstracta, mientras que una misma clase puede implementar varias interfaces.
- El encapsulamiento es posible en las clases abstractas (public, private, protected, final, etc), mientras que en una interfaz dada su naturaleza todos los métodos son declarados como public.

Ejemplo práctico ABM

Pensar en términos de objetos es muy parecido a cómo lo haríamos en la vida real. Por ejemplo, vamos a pensar en un coche para tratar de modelizarlo en un esquema de POO. Diríamos que el coche es el elemento principal que tiene una serie de características, como podrían ser el color, el modelo o la marca. Además, tiene una serie de funcionalidades asociadas, como pueden ser ponerse en marcha, parar o aparcar.

Pues en un esquema POO el coche sería el objeto, las propiedades serían las características como el color o el modelo y los métodos serían las funcionalidades asociadas como ponerse en marcha o parar.

Por poner otro ejemplo vamos a ver cómo modelizaríamos en un esquema POO una fracción, es decir, esa estructura matemática que tiene un numerador y un denominador que divide al numerador, por ejemplo 3/2.

La fracción será el objeto y tendrá dos propiedades, el numerador y el denominador. Luego podría tener varios métodos como simplificarse, sumarse con otra fracción o número, restarse con otra fracción, etc.

Estos objetos se podrán utilizar en los programas, por ejemplo, en un programa de matemáticas harás uso de objetos fracción y en un programa que gestione un taller de coches utilizarás objetos coche. Los programas Orientados a objetos utilizan muchos objetos para realizar las acciones que se desean realizar y ellos mismos también son objetos. Es decir, el taller de coches será un objeto que utilizará objetos coche, herramienta, mecánico, recambios, etc.

Teniendo en cuenta esto vamos a armar un ABM de productos con login de usuarios. Para ello vamos a definir una clase Base de Datos, una clase Usuario y una clase Producto. Tanto la clase Usuario como la clase Producto usarán la clase Base de Datos.

MYSQLI ORIENTADO A OBJETOS Y PDO

¿Qué es una API?

Una Interfaz de Programación de Aplicaciones, o API, define las clases, métodos, funciones y variables que necesitará llamar una aplicación para llevar a cabo una tarea determinada. En el caso de las aplicaciones de PHP que necesitan comunicarse con un servidor de bases de datos, las APIs necesarias se ofrecen generalmente en forma de extensiones de PHP.

Las APIs pueden ser procedimentales u orientadas a objetos. Con una API procedimental invocan funciones para llevar a cabo las tareas, mientras con una API orientada a objetos se instancian clases, y entonces se invocan a métodos de los objetos creados. Entre ambas opciones, la segunda es generalmente la vía recomendada, puesto que está más actualizada y conlleva una mejor organización de código.

Cuando se escriben aplicaciones PHP que necesitan conectar a un servidor MySQL, existen varias opciones disponibles respecto a API. Este documento abarca esas opciones, y ayuda a elegir la mejor solución para cada aplicación.

¿Qué es un Conector?

En la documentación de MySQL, el término conector hace referencia al software que permite a una aplicación conectarse a un servidor de bases de datos MySQL. MySQL proporciona conectores para ciertos lenguajes, entre ellos PHP.

Si una aplicación de PHP necesita comunicarse con un servidor de bases de datos, necesitará escribir el código PHP que realice tareas tales como conectar al servidor de bases de datos, realizar consultas y otras funciones relacionadas con bases de datos. Es necesario tener un software instalado en el sistema que proporcione a la aplicación en PHP la API, que manejará la comunicación entre el servidor de bases de datos y la aplicación, posiblemente empleando en caso necesario otras bibliotecas. A este software generalmente se le conoce como conector, dado que permite a una aplicación conectar con un servidor de bases de datos.

¿Qué es un Driver?

Un driver es un software diseñado para comunicarse con un tipo específico de servidor de bases de datos. Podría también invocar a una biblioteca, como por ejemplo la Biblioteca Cliente de

MySQL o el Driver Nativo de MySQL. Estas bibliotecas implementan el protocolo de bajo nivel que se utiliza para comunicarse con el servidor de bases de datos.

A modo de ejemplo, la capa de abstracción de bases de datos Objetos de Datos de PHP (PDO) utilizará alguno de los drivers para bases de datos disponibles. Uno de ellos es el driver PDO MYSQL, que permite comunicarse con un servidor MySQL.

A menudo la gente utiliza los términos conector y driver indistintamente. Esto puede dar lugar a confusión. En la documentación de MySQL, el término "driver" queda reservado para el software que proporciona la parte específica de una base de datos dentro de un conector.

¿Qué es una Extensión?

En la documentación de PHP aparece otro término - extensión. El código fuente de PHP consiste por un lado de un núcleo, y por otro de extensiones opcionales para el núcleo. Las extensiones de PHP relacionadas con MySQL, tales como mysqli, y mysql, están implementadas utilizando el framework de extensiones de PHP.

Típicamente, una extensión ofrece una API al programador de PHP para permitirle hacer uso de sus utilidades mediante código. Sin embargo, algunas de las extensiones que utilizan el framework de extensiones de PHP no ofrecen ninguna API al programador.

La extensión del driver PDO MySQL, por ejemplo, no proporciona ninguna API al programador PHP, pero en su lugar ofrece una interfaz a la capa de PDO que tiene por encima.

No deben confundirse los términos API y extensión, dado que una extensión no debe necesariamente proporcionar una API al programador.

¿Cuáles son las principales APIs que PHP ofrece para utilizar MySQL?

Hay tres APIs principales a la hora de considerar conectar a un servidor de bases de datos MySQL:

- Extensión MySQL de PHP
- Extensión mysqli de PHP
- Objetos de Datos de PHP (PDO)

Cada una tiene sus ventajas e inconvenientes. El siguiente apartado trata de dar una breve introducción a los aspectos clave de cada API.

¿Qué es la Extensión MySQL de PHP?

Esta es la extensión original diseñada que permite desarrollar aplicaciones PHP que interactúan con bases de datos MySQL. La extensión mysql proporciona una interfaz procedural, y está pensada para usar sólo con versiones de MySQL anteriores a la 4.1.3. Si bien esta extensión se puede utilizar con versiones de MySQL 4.1.3 o posteriores, no estarán disponibles todas las nuevas funcionalidades del servidor MySQL.

¿Qué es la extensión mysqli de PHP?

La extensión mysqli, o como a veces se le conoce, la extensión de MySQL mejorada, se desarrolló para aprovechar las nuevas funcionalidades encontradas en los sistemas MySQL con versión 4.1.3 o posterior. La extensión mysqli viene incluida en las versiones PHP 5 y posteriores.

La extensión mysqli contiene numerosos beneficios, siendo estas las mejoras principales respecto a la extensión mysql:

- Interfaz orientada a objetos
- Soporte para Declaraciones Preparadas
- Soporte para Múltiples Declaraciones
- Soporte para Transacciones
- Mejoradas las opciones de depuración
- Soporte para servidor empotrado

Además de la interfaz orientada a objetos, esta extensión también proporciona una interfaz procedural.

La extensión mysqli está desarrollada mediante el framework de extensiones de PHP

¿Qué es PDO?

Los Objetos de Datos de PHP, o PDO, son una capa de abstracción de bases de datos específicas para aplicaciones PHP. PDO ofrece una API homogénea para las aplicaciones PHP, independientemente del tipo de servidor de bases de datos con el que se vaya a conectar la aplicación. En teoría, si se utiliza la API PDO, se podría cambiar el servidor de bases de datos en uso, por ejemplo, de Firebird a MySQL, y sólo se necesitarían algunos cambios menores en el código PHP.

Otros ejemplos de capas de abstracción de bases de datos son JDBC para aplicaciones Java o DBI para Perl.

A pesar de que PDO tiene sus ventajas, tales como una API limpia, sencilla y portable, su mayor inconveniente es que no permite utilizar todas las funcionalidades avanzadas en la última versión del servidor MySQL. Por ejemplo, PDO no permite hacer uso de las Declaraciones Múltiples de MySQL.

PDO está desarrollado utilizando el framework de extensiones de PHP.

Comparación de Funcionalidades

La siguiente tabla compara las funcionalidades de los tres principales métodos para conectar a MySQL desde PHP:

Comparación de las opciones de la API de MySQL para PHP

	Extensión mysqli de PHP	PDO (Usando el driver PDO MySQL y el Driver Nativo MySQL)	Extensión MySQL de PHP
Versión de PHP en que se introdujo	5.0	5.0	Antes de 3.0
Incluído con PHP 5.x	Sí	Sí	Sí
Estado de desarrollo de MySQL	Desarrollo activo	Desarrollo activo, desde PHP 5.3	Sólo se le mantiene
Recomendado por MySQL para nuevos proyectos	Sí - opción recomendada	Sí	No
Soporte para juegos de caracteres	Sí	Sí	No
Soporte para Declaraciones Preparadas en el lado del servidor	Sí	Sí	No
Soporte para Declaraciones Preparadas en el lado del cliente	No	Sí	No
Soporte para Procedimientos Almacenados	Sí	Sí	No
Soporte para Declaraciones Múltiples	Sí	Mayormente	No
Suporte para todas las funcionalidades de MySQL 4.1+	Sí	Mayormente	No