Orientación a objetos en PHP

Dídac Gil de la Iglesia

PID_00155710
Índice

Introducción......................................................................................................................... 5

1. ¿Porqué usar PHP OO?.................................................................................................. 7

2. Organización del código............................................................................................... 9

3. Reutilización de código.............................................................................................. 12

4. Multiplicidad............................................................................................................... 13

5. Herencia...................................................................................................................... 17

6. Visibilidad................................................................................................................... 19

7. Sobrecarga de clases................................................................................................... 22

8. Constructores y destructores...................................................................................... 24
Introducción

PHP es un lenguaje de programación estructurado con extensiones de orientación a objetos (OO). A continuación se explica cómo programar siguiendo una metodología orientada a objetos bajo PHP.
1. ¿Porqué usar PHP OO?

En PHP, una función declarada en una página PHP podrá ser usada inmediatamente después a su declaración.

Como primera aproximación, uno podría plantearse programar en cada fichero PHP todas las funciones necesarias que vayan a usarse en el fichero. En proyectos grandes, esto supone un alto coste de mantenimiento debido a la complejidad de seguimiento del código y a la duplicación del mismo código en distintas páginas. En caso de detectarse un error y el código que lo corrija, o en caso de querer aplicar una mejora en una función, deberán revisarse todos los scripts PHP para comprobar si contienen la función que hay que corregir.

Sin mucha dificultad puede observarse que la práctica de repetir códigos en scripts PHP queda lejos de ser una vía óptima para la programación en proyectos.

PHP permite definir clases, que poseen variables (también llamadas propiedades) y métodos (funciones). A continuación vemos un ejemplo donde se crea la clase Matematicas con 2 variables y dos métodos.

Ved también

Dentro de la documentación en línea disponible en la web de PHP.net, existe una sección dedicada a la programación orientada a objetos. La programación OO se introdujo en PHP versión 4 y ha sido ampliada dentro de la versión 5. En caso de querer exprimir la OO al máximo en PHP, es recomendable realizar una lectura de las funcionalidades disponibles en OO y de las que puedan aparecer en futuras versiones.
<?php

class Matematicas{
    //Declaracion de las variables internas
    $error="";

    //Funciones

    //Sumar los valores
    function suma($valor1, $valor2)
    {
        if(is_numeric($valor1) && is_numeric($valor2))
        {
            $this->error="";
            return $valor1 + $valor2;
        }
        else
        {
            $this->error="Un valor no es numerico";
            return "NaN"
        }
    }

    //Restar los valores
    function resta($valor1, $valor2)
    {
        if(is_numeric($valor1) && is_numeric($valor2))
        {
            $this->error="";
            return $valor1 - $valor2;
        }
        else
        {
            $this->error="Un valor no es numerico";
            return "NaN"
        }
    }

    function obtener_error()
    {
        return $error;
    }
}

Fichero matematicas clase.php

Las clases permiten organizar mejor el código, ampliar nuestra productividad y eficiencia mediante la reutilización de código, la multiplicidad y la versatilidad, conceptos que trataremos en los siguientes puntos.
2. Organización del código

En PHP orientado a objetos, cada clase se programa dentro de un único fichero.

Posteriormente a la creación del fichero con la definición de la clase, pueden crearse instancias de la clase, llamadas objetos. Si hacemos un símil con el diseño de una silla, el prototipo de la silla es la “clase”. Se definen las medidas, los materiales (variables) y los usos (funciones) que la silla puede tener. Cada silla física generada a partir de ese diseño será un “objeto” y podrá usarse dentro de diferentes ubicaciones, entornos y para diferentes propósitos.

En el siguiente ejemplo, se ha programado una clase llamada Texto que permite: formatear cadenas de texto para imprimirlas correctamente en HTML, formatear una cadena de caracteres que contenga una consulta SQL para minimizar el peligro de un ataque de seguridad, concatenar cadenas y eliminar espacios innecesarios.
<?php
class Texto{
    //Declaración de las variables internas
    $cadena = "";

    //Funciones
    //Obtiene el valor inicial de la cadena
    function entrar_cadena($variable){
        $this->cadena = $variable;
    }

    //Elimina los espacios existentes al inicio y fin de la cadena
    function limpiar_espacios(){
        return trim($this->cadena);
    }

    //Se inserta una cadena de caracteres delante o detrás de la cadena a trabajar
    function insertar_delante ($textoDelante){
        $this->cadena = $textoDelante . $this->cadena;
    }

    function insertar_detrás ($textoDetras){
        $this->cadena = $this->cadena . $textoDetras;
    }

    //Escape los caracteres especiales para evitar errores en SQL
    function preparar_SQL(){
        return addslashes($this->cadena);
    }

    //Escribe en formato HTML los caracteres especiales, tales como los acentos
    function preparar_HTML(){
        return htmlspecialchars($this->cadena);
    }

    function mostrar_cadena(){
        return $this->cadena;
    }
}
?>

Fichero texto_clase.php

En el fichero aplicación.php se crea un objeto de la clase texto con el nombre de $cadena. Sólo mediante la instancia $cadena, podemos acceder a las funciones existentes dentro de la clase.
Con la programación orientada a objetos, las clases quedan localizadas dentro de un único fichero PHP. Durante el proceso de programación, suelen darse errores dentro del comportamiento de las aplicaciones, por lo que es necesario efectuar un análisis para detectar la función que está incorrectamente implementada. En la programación orientada a objetos, dada su organización, solamente será necesario corregir el error dentro de la clase correspondiente, y todos los objetos de la clase creados se beneficiarán de la corrección.

Por ejemplo, el programador podría detectar que ciertas cadenas al usarse con htmlspecialchars no son convertidas a lenguaje HTML, por lo que podría preferir usar la función htmlentities. Si el programador ha creado la clase Texto, simplemente deberá modificar la función preparar_HTML.

```php
function preparar_HTML(){
    return htmlentities($this->cadena);
}
```

Sí, por el contrario, no ha usado la clase, deberá buscar todas las referencias a htmlspecialchars dentro de su proyecto y sustituirlas por la nueva función, lo que supondrá una sobrecarga de tiempo, y un alto potencial a olvidarse alguna referencia sin corregir.
3. Reutilización de código

La creación de un portal dinámico en PHP suele componerse de páginas de presentación, formularios, obtención de información existente dentro de bases de datos, tratamiento de datos obtenidos desde formularios, bases de datos, portales externos, páginas XML, RSS, etc. Hay funcionalidades comunes a la mayoría de los proyectos, como la validación de variables o el acceso a bases de datos, por lo que el programador tiende a reutilizar código implementado en proyectos anteriores, con lo que incrementa así su eficiencia y evita posibles errores de programación.

Estamos acostumbrados a reutilizar código de terceras personas continuamente, como por ejemplo la API de PHP, con funciones tales como `mysql_query()`, `require()` o hasta un simple `echo()`.

A mayor escala, un programador podrá aprovechar un subconjunto de las funciones creadas en proyectos anteriores en sus nuevos proyectos, como por ejemplo funciones para establecer y gestionar sesiones, o validar los usuarios en la base de datos.

Sin hacer uso de la programación OO tendríamos nuestro código repartido en varios ficheros PHP. Además, el código PHP que se podría reutilizar estaría entremezclado con código HTML del portal.

Usando programación OO, todas las funciones quedarán dentro de una o varias clases y, por lo tanto, el correspondiente código se encontrará en un solo fichero PHP, lo que permitirá separar el código PHP de las líneas HTML. Para usarlo en una página PHP, el programador simplemente tendrá que cargar el fichero correspondiente a la clase que quiere reutilizar en la página PHP. La carga del fichero con la clase se realizará con la llamada `require_once()`.
4. Multiplicidad

Como ya hemos detallado, una clase es un “prototipo” que define variables y funciones que pueden realizarse sobre estas variables.

Una vez definida una clase, podemos instanciar tantos objetos de la clase como sean necesarios, teniendo todos ellos las características de la clase, es decir, sus variables y funciones.

Para hacer más simple el entendimiento del concepto multiplicidad, expondremos el siguiente ejemplo.

Dentro de una aplicación web en la que se debe hacer un traspaso de dinero entre dos cuentas bancarias, existe una cuenta a la que se hará un ingreso de dinero (cuenta destino, B) que proviene de una cuenta a la que se le deberá sustraer dicho saldo (cuenta origen, A).

Un traspaso de un saldo supone una consulta de saldo en la cuenta origen y una actualización en la misma cuenta para restar el saldo que se va a traspasar. Por la parte del beneficiario, implica una consulta del saldo actual para poder realizar una actualización que lo incremente con el traspaso.

Para ello, se ha creado una clase Cuenta que permite obtener el saldo de una cuenta corriente, así como actualizar su saldo.
<?php

class Cuenta{
    //Declaración de las variables internas
    $num_cuenta;
    $saldo;

    //Declaración del número de cuenta bancaria
    function definir_cuenta($numero_cuenta)
    {
        $this->num_cuenta = $numero_cuenta;
    }

    //Consulta el saldo de la cuenta asociada
    function obtener_saldo()
    {
        $this->saldo = sql_obtener("SELECT saldo FROM cuentas WHERE numero_cuenta = "$this->num_cuenta");
        return $this->saldo;
    }

    //Añade valores a las variables
    function entrar_saldo($valor)
    {
        $this->saldo = $valor;
        sql_actualizar("UPDATE saldo FROM cuentas WHERE numero_cuenta = "$this->num_cuenta");
    }
}

//Fichero cuenta clase.php

En la aplicación PHP, se hace uso de la clase creada Cuenta y la clase Matemáticas.
<?php
require_once("matematicas_clase.php");
require_once("cuenta_clase.php");
...

//Definimos una función que permite el traspaso entre cuentas. Usa dos objetos de la clase Cuenta y un valor numérico que representa el importe a traspasar entre las cuentas. La función hace uso de la clase Matematicas para comprobar que existe saldo suficiente en la cuenta origen para el traspaso.

function traspaso_saldo($origen, $destino, $importe)
{
    $mensaje="";
    $calculadora=new Matematicas();

    //comprobar saldo origen
    if($calculadora->resta($origen->obtener_saldo(), $importe) < 0) {
        $error=$calculadora->obtener_error();
        if($error <> "") return "El campo importe no es numérico";
        else return "No hay saldo suficiente en la cuenta origen";
    }
    else{
        //hay saldo suficiente. Se realiza el traspaso
        $importe_origen=$calculadora->resta($origen->obtener_saldo(), $importe);
        $origen->entrar_saldo($importe_origen);

        $importe_destino=$calculadora->suma($importe, $destino->obtener_saldo());
        $destino->entrar_saldo($importe_destino);
        return "Importe traspasado";
    }
}"
Dado que se va a trabajar con dos cuentas bancarias, se han instanciado dos objetos de la clase Cuenta. La multiplicidad es la capacidad de usar múltiples instancias de una misma clase, con lo que se puede simplificar el código de una aplicación PHP y mantener el estado de varios objetos al mismo tiempo.
5. Herencia

En ocasiones, el código no puede reutilizarse tal cual, sino que debe especializarse o refinarse para tener en cuenta nuevas necesidades. Uno podría crear una nueva clase, reescribiendo todo el código reaprovechable en ella e incluir también las nuevas utilidades implementando las funciones necesarias. No obstante, eso limitaría la posible reutilización de código.

Una de las ventajas de la programación orientada a objetos es el uso de la herencia, que soluciona el problema anterior. La herencia permite crear una nueva clase que “herede” las características (variables y métodos) de otra clase. En PHP la herencia se indica mediante el código EXTENDS.

La clase Cuenta vista en el subapartado anterior tenía la limitación de ser monodivisa, por lo que las cuentas origen y destino deben compartir la misma moneda para poder hacer las transferencias bancarias.
<?php
require_once("cuenta_clase.php");

class Cuenta_Multidivisa extends Cuenta{
    //Declaracion de las variables internas
    $divisa;

    //Establece el numero de cuenta bancaria y la divisa por defecto con la que se
    //quieren mostrar los valores de la cuenta.
    function definir_cuenta($valor, $divisa)
        {
            $this->num_cuenta = $valor;
            $this->divisa= $divisa;
        }

    //Permite obtener la divisa por defecto usada para la cuenta
    function consultar_divisa()
        {
            return $this->divisa;
        }

    //Consulta el valor de la cuenta en la divisa
    function aplicar_cambio($divisa_destino)
        {
            $coefficiente=obtener_cambio_BancoEuropeo($this->divisa, $divisa_destino);
            return $this->saldo * $coefficiente;
        }

    //Cambia la divisa por defecto con la que se quieren mostrar valores de la
cuenta.
    Actualiza la cantidad de saldo para mostrar su correcto valor en la divisa
    correspondiente.
    function cambiar_divisa($nueva_divisa)
        {
            if ($this->saldo = 0 || $this->divisa = "")
                $this->divisa= $nueva_divisa;
            else{
                $this->saldo = $this->aplicar_cambio($nueva_divisa);
                $this->divisa= $nueva_divisa;
            }
        }
}
?>

Fichero cuenta_multidivisa_class.php

Haciendo una clase Cuenta_Multidivisa que extienda la clase Cuenta, podrá
añadirse una nueva funcionalidad que haga el cambio entre divisas consultando el estado actual del cambio en el Banco Europeo.
6. Visibilidad

En la programación orientada a objetos sobre PHP, las funciones y las variables existentes en los objetos tienen tres niveles de visibilidad, clasificados en Public, Protected y Private, de más a menos permisivo respectivamente.

Imaginemos la siguiente aplicación PHP que hace uso de la clase Cuenta_Multidivisa.

```php
<?php
    require_once(“cuenta_multidivisa_clase.php”);

    // Se consulta el saldo actual de cada una de las cuentas, probablemente desde
    // una base de datos
    $cuenta_A = new Cuenta_Multidivisa();
    $cuenta_A->definir_cuenta("0123456789012345", "Peseta");

    $cuenta_A->entrar_valores(5000);

    $cuenta_A->divisa =”Euro”;
}
?>
```

Hasta el momento sólo habíamos accedido a las funciones de la clase Cuenta_Multidivisa pero, así como podemos acceder a las funciones de la clase, también es posible acceder a sus variables.

Por defecto, la visibilidad de las funciones y las variables es Publica, lo que quiere decir que, una vez instanciado un objeto de la clase, puede accederse directamente a sus variables internas y a sus funciones.

En ciertos casos, puede ser beneficioso acceder a las variables internas de un objeto. En otros casos, como el expuesto en el código anterior, puede ser una fuente de errores. Podemos ver que el objeto cuenta_A, gracias a que ha podido accederse a la variable divisa, ha pasado de tener 5.000 pesetas a tener 5.000 euros.

Veamos ahora el mismo código mejorado para evitar tales errores.
<?php
require_once("cuenta_clase.php");

class Cuentas_Multidivisa extends Cuenta{
    //Declaración de las variables internas
    private $divisa;

    //Establece el número de cuenta bancaria y la divisa por defecto con la que
    //se quieren mostrar los valores de la cuenta.
    public function definir_cuenta($valor, $divisa)
    {
        $this->num_cuenta =$valor;
        $this->divisa=$divisa;
    }

    //Permite obtener la divisa por defecto usada para la cuenta
    public function consultar_divisa()
    {
        return $this->divisa;
    }

    //Consulta el valor en una divisa destino
    public function aplicar_cambio($divisa_destino)
    {
        $coeficiente=obtener_cambio_DancoEuropeo($this->divisa, $divisa_destino);
        return $this->saldo * $coeficiente;
    }

    //Cambia la divisa por defecto con la que se quieren mostrar valores de
    //la cuenta.
    //Actualiza la cantidad de saldo para mostrar su correcto valor en la divisa
    //correspondiente.
    public function cambiar_divisa ($nueva_divisa)
    {
        if ($this->saldo = 0 || $this->divisa = "")
            $this->divisa = $nueva_divisa;
        else{
            $this->saldo = $this->aplicar_cambio($nueva_divisa);
            $this->divisa = $nueva_divisa;
        }
    }
}
Así como en el ejemplo del cambio de divisa, existen diversidad de casos donde variables, e incluso funciones internas, deberán ser privadas para obligar a la lectura o modificación de estas variables a través de funciones donde se hayan implementado los controles deseados.

Como ejemplos, podemos pensar en las variables de usuario en la autenticación de una aplicación. El cambio de contraseña deberá ser mediante la llamada a una función, asegurando así que la nueva contraseña tiene la complejidad adecuada, se actualiza el valor tanto en la sesión como en la base de datos, etc.

```php
<?php
require_once("cuentas_multidivisa clase.php");

//Se consulta el saldo actual de cada una de las cuentas, probablemente desde una base de datos
$cuenta_A = new cuenta_Multidivisa();
$cuenta_A->definir_cuenta("0123 4557 8901 2345");
$cuenta_A->establecer_divisa("Peso");
$cuenta_A->entrar_valores(5000);

$cuenta_A->divisa ="Euro";  //Error. No es posible acceder
//a $salo_A->divisa_origen
$cuenta_A->establecer_divisa("Euro");  //El saldo ha pasado a ser 30.05
}
?>
```

Así como en el ejemplo del cambio de divisa, existen diversidad de casos donde variables, e incluso funciones internas, deberán ser privadas para obligar a la lectura o modificación de estas variables a través de funciones donde se hayan implementado los controles deseados.

Como ejemplos, podemos pensar en las variables de usuario en la autenticación de una aplicación. El cambio de contraseña deberá ser mediante la llamada a una función, asegurando así que la nueva contraseña tiene la complejidad adecuada, se actualiza el valor tanto en la sesión como en la base de datos, etc.

```php
<?php
class sesiones{

//Declaracion de las variables internas
private $login;
private $password;

public cambio_password($old, $new_password) {
    if(compleja($new_password && $this->password == $old) ) {
        actualizar_password($this->login, $new_password);
        $this->password = $new_password;
        return 0;  //Sin errores
    } else
        return -1;  //Con errores
}
}
?>
```
7. Sobrecarga de clases

Como hemos visto hasta ahora, para poder instanciar objetos pertenecientes a las clases implementadas, debemos usar alguno de los comandos “require”, “include”, “require_once” o “include_once”.

En ocasiones, la lista de clases que vamos a requerir para una aplicación PHP puede ser larga, por lo que deberemos hacer una gran cantidad de sentencias “require” para poder hacer uso de todas ellas.

En PHP versión 5, se ha creado la función __autoload() que, en caso de no disponerse de una clase en la creación de un objeto, es lanzada para tratar de evitar errores.

Un programador puede definir el contenido de la función __autoload() para tratar dichos casos.


Aprovechando la funcionalidad __autoload(), y una estructura de ficheros bien definida, es posible solicitar la carga de las clases bajo demanda.

Veamos el siguiente código.

```php
<?php
    function __autoload($nombre_clase) {
        require_once $nombre_clase . '_clase.php';
    }

    $cuenta_A = new Cuenta();
?
```

La creación del objeto cuenta_A por si solo daría un error de ejecución al no conocerse la existencia de la clase Cuenta, ya que no ha sido cargada. En la versión 5 de PHP, antes de lanzar dicho mensaje de error se ejecutará la función “__autoload()” pasando como parámetro el nombre de la clase faltante. En nuestro caso, se lanzará “__autoload(Cuenta)”, haciendo que se ejecute la siguiente sentencia:
require_once Cuenta_clase.php;

Dicha sentencia corregirá el problema cargando la clase Cuenta. Cabe notar también que la clase Cuenta se añadirá solamente cuando sea requerida para la instanciación de un objeto. Su uso nos puede ahorrar la molestia de tener que escribir el correspondiente “require_once” para cada una de las clases que vayamos a usar.
Otro de los conceptos existentes en la programación orientada a objetos son los constructores y los destructores.

Se trata de la implementación de funciones que se lanzan automáticamente en el proceso de creación y de destrucción de los objetos usados en una aplicación PHP.

Los **constructores** suelen usarse para la inicialización de variables durante el proceso de instanciación de objetos, y tienen como nombre de función __construct().

Los **destructores**, por otro lado, suelen usarse para hacer salidas controladas de aplicaciones, tales como el cierre de ficheros, cierre de sesiones con bases de datos o similares. La función destructor tiene el nombre __destruct().

La función constructora __construct permite la entrada de variables para la instanciación.

```php
<?php
    class Cuenta_2{
        private $divisa_origen;
        private $saldo;
        private $numero_cuenta;

        public function __construct($cuenta, $divisa) {
            $this->numero_cuenta=$cuenta;
            $this->divisa_origen=$divisa;
        }
    }
    ...
?>
```

Como ejemplo práctico de constructores y destructores presentamos el acceso a bases de datos. Permitirá establecer conexiones con una base de datos definida por defecto de manera automática con sólo instanciar un objeto, y asegurar su desconexión al salir del fichero PHP.
```php
<?php
require("parametros_conexion.php"); // fichero que contiene user, password,
                                        // server, DBname

class ConexiionBDDB{
    private $conexion;

    public function __construct() {
        $this->conexion = mysql_connect($server, $user, $password);
        mysql_select_db( $DBname , $this->conexion );
    }

    public function __destruct () {
        mysql_close( $this->conexion );
    }
}
```