



U.T.7: Programación Orientada a objetos en PHP

Contenidos

- ❑ Introducción al a programación OO
 - ❑ Modelo de orientación a objetos de PHP3 y PHP4.
 - ❑ Modelo de orientación a objetos de PHP5.
 - ❑ Clases en PHP5.
 - ❑ Constructores y destructores en PHP5.
 - ❑ Modificadores de acceso.
 - ❑ Sintaxis de la herencia.
 - ❑ Objetos como atributos de otros objetos.
 - ❑ Clases y métodos abstractos.
 - ❑ Interfaces.
 - ❑ Control de excepciones
-

Programación orientada a objetos

- ❑ PHP no es un lenguaje OO puro.
 - ❑ En las primeras versiones PHP que lo incorporaban presentaba ciertas carencias (como la ocultación, la herencia múltiple, el polimorfismo,...)
 - ❑ En las versiones más recientes todas las características de POO están disponibles.
-

Programación orientada a objetos

❑ Clase:

- ❑ Conjunto de objetos similares. Todo objeto es una instancia de una clase.
- ❑ Conjunto de variables, llamados atributos o propiedades, y funciones, llamadas métodos, que trabajan sobre esas variables.
- ❑ Especificación de propiedades y funcionalidades de elementos que van a participar en nuestros programas.

❑ Definición de una clase:

```
class NombreClase{  
    var $variables;  
    Function metodo($parametros){  
        //codigo  
    }  
}
```

Programación orientada a objetos

❑ Ejemplo: clase "Caja"

- ❑ **Atributos**: dimensiones, color, contenido,...
 - ❑ Funciones o **métodos** que podríamos incorporar a la clase "caja":
introduce(), muestra_contenido(), comprueba_si_cabe(), vaciate()...
 - ❑ Los **atributos** se definen declarando unas variables al principio de la clase.
 - ❑ Los **métodos** se definen declarando funciones dentro de la clase
-

Programación orientada a objetos

❑ Ejemplo: clase "Caja"

```
class Caja{  
    var $alto;           // public en PHP5.3  
    var $ancho;  
    var $largo;  
    var $contenido;  
    var $color;  
    function introduce($cosa){  
        $this->contenido = $cosa;  
    }  
    function muestra_contenido(){  
        echo $this->contenido;  
    }  
}
```

Dentro de un método, la variable **\$this** hace referencia al objeto sobre el que invocamos el método

Programación orientada a objetos

❑ Utilizar la clase: **creación y uso de objetos**

- ❑ Las clases solamente son definiciones.
- ❑ **Crear** un ejemplar de una clase, es decir **instanciar** un objeto de una clase:

`$micaja = new Caja;`

- ❑ Con esto hemos creado, o mejor dicho, instanciado, un objeto de la clase Caja llamado \$micaja.
-

Programación orientada a objetos

❑ Utilizar la clase: **creación y uso de objetos**

❑ Una vez creado el objeto podemos **usarlo**:

- ❑ Los métodos de un objeto se llaman utilizando el código: “->”
nombre_del_objeto->nombre_de_metodo()

❑ Ejemplo:

```
$micaja->introduce("algo");
```

```
$micaja->muestra_contenido();
```

- ❑ Para acceder a los atributos de una clase también se accede con el código “->”:

```
nombre_del_objeto->nombre_del_atributo
```

Programación orientada a objetos

```
class Cliente{  
var $nombre;   
var $direccion;  
var $codigo;  
function rellena($nom,$dir,$cod){  
$this -> nombre=$nom;  
$this -> direccion=$dir;  
$this -> codigo=$cod;  
}  
}
```

Definición de métodos y atributos
(las variables deben llevar **var**
delante) PHP3 y PHP4

Utilización de **\$this**

Operador **new** para a creación

Ojo! Sólo un \$:
\$this -> \$codigo ¡Incorrecto!

```
$cliente1 = new Cliente;  
$cliente1 -> rellena("Juan","Pérez","344")
```

Programación orientada a objetos

❑ Constructores

- ❑ Si en la creación de la clase hay un método con el mismo nombre que la clase se llama **constructor** y se ejecuta siempre cuando se crea un objeto de esa clase. Suele inicializar las variables.
- ❑ Un único constructor por clase. Esta definición está obsoleta en

PHP7.

Se recomienda
definirlo como
método mágico:
__construct()

```
class Caja{  
    var $alto;  
    var $ancho;  
    var $largo;  
    var $contenido;  
    function Caja($alto=1,$ancho=1,$largo=1,$color="negro"){  
        $this->alto=$alto;  
        $this->ancho=$ancho;  
        $this->largo=$largo;  
        $this->color=$color;  
        $this->contenido=""; }  
    function introduce($cosa){  
        $this->contenido = $cosa; }  
    function muestra_contenido(){  
        echo $this->contenido; }  
}
```

Programación orientada a objetos

❑ Herencia

- ❑ Crear una clase derivada de otra
- ❑ Hereda todas las **propiedades** (variables) y métodos (funciones) de la clase madre.
- ❑ Importante: el constructor de la clase madre NO se invoca automáticamente si existe constructor en la clase hija.

```
class ClaseDerivada extends ClaseBase{  
  // definición de métodos y variables  
  // exclusivos de ClaseDerivada,  
  // y redefinición (especialización)  
  // de métodos de ClaseBase  
}
```

Uso de **extends**

```
class CajaFuerte extends Caja {  
  var $combinacion;  
  function programa_codigo($numero){  
    $this->combinacion = $numero;}  
}
```

Programación orientada a objetos

❑ Funciones para el manejo de clases y objetos:

❑ **get_class_methods(clase)**

- ❑ Devuelve un array con los **nombres de los métodos de una** clase dada

❑ **get_class_vars(clase)**

- ❑ Devuelve un array con los **nombres de las propiedades** (inicializadas por defecto) de la clase.
- ❑ Los elementos del array están organizados de la forma
varname => value.

❑ **get_object_vars(objeto)**

- ❑ Devuelve un array con los **nombres de las propiedades** (inicializadas por defecto) de un **objeto** concreto
-

Programación orientada a objetos

❑ Funciones para el manejo de clases y objetos:

❑ **method_exists(objeto o clase, método)**

- ❑ Devuelve true si el método pasado como argumento existe en el objeto también pasado como argumento.

❑ **class_exists(clase, boolean \$autoload)**

- ❑ Esta función verifica si la clase dada ha sido definida o no.
- ❑ **\$autoload** indica si ha de llamarse a `__autoload` de forma predeterminada.

❑ **interface_exists(interface, boolean \$autoload)**

- ❑ Comprueba si la interfaz dada ha sido definida.
-

Modelo de orientación a objetos de PHP3 y PHP4.

- ❑ Sólo implementaba una parte pequeña de las características de POO.
 - ❑ En PHP3 podíamos crear clases e instanciar objetos.
 - ❑ En PHP4 no se cambia sustancialmente la versión de POO de PHP3.
 - ❑ Los objetos se pasan como parámetros por valor (se duplica su almacenamiento en memoria)
-

Modelo de orientación a objetos de PHP3 y PHP4.

```
class Caja{
    var $contenido;
    function introduce($cosa){
        $this->contenido=$cosa;
    }
    function muestra_contenido(){
        echo this->contenido;
    }
}

$micaja=new Caja;
$micaja->introduce("algo");
$micaja->muestra_contenido();
echo "<br>";
$segunda_caja=$micaja;
// $segunda_caja=&$micaja;
$segunda_caja->introduce("contenido en segunda caja");
$segunda_caja->muestra_contenido();
echo "<br>";
$micaja->muestra_contenido();
```

PHP dispone del paso de parámetros por referencia a través de &

Modelo de orientación a objetos de PHP5

- ❑ PHP5 supuso una reestructuración del núcleo de PHP (Zend engine 2.0)
- ❑ El nuevo núcleo de PHP incluye la reescritura casi total del modelo de objetos, basándose en el referente indiscutible de java.
- ❑ Documentación sobre el modelo de objetos de PHP5:
<http://php.net/manual/es/language.oop5.php>
- ❑ Esto repercute tanto en la compatibilidad como en la ejecución de los scripts de PHP3 y PHP4.
- ❑ La conectividad PHP/MySQL se provee a través de una API externa.
- ❑ Registro de cambios de POO

<http://us1.php.net/manual/es/language.oop5.changelog.php>

Modelo de orientación a objetos de PHP5

- ❑ Nombres fijos para los constructores y destructores:
__construct() y **__destruct()**
 - ❑ Acceso **public**, **private** y **protected** a propiedades y métodos.
 - ❑ Posibilidad de uso de interfaces.
 - ❑ Métodos y clases **final**
 - ❑ Operador instanceof
 - ❑ Atributos y métodos **static**
 - ❑ Clases y métodos abstractos
 - ❑ Constantes de clase
 - ❑ Funciones que especifican la clase de objeto que reciben como parámetro
-

Modelo de orientación a objetos de PHP5

- ❑ **`__autoload()`**: Intenta incluir el código de una clase que se necesita y no ha sido declarada todavía en el código que se está ejecutando.
 - ❑ **`spl_autoload_register()`** proporciona una alternativa más flexible para la carga automática de clases. Por esta razón, el uso de `__autoload()` no se recomienda y puede quedar obsoleta o ser eliminada en el futuro.
 - ❑ **`__clone()`**
Clonado de objetos:
-

Orientación a objetos de PHP5:

❑ spl_autoload_register()

```
<?php
// function __autoload($clase) {
//     include 'clases/' . 'class.' . $clase . '.php';
// }

function mi_autocargador($clase) {
    include 'clases/' . 'class.' . $clase . '.php';
}

spl_autoload_register('mi_autocargador');

// O, usando una función anónima a partir de PHP 5.3.0
spl_autoload_register(function ($clase) {
    include 'clases/' . 'class.' . $clase . '.php';
});
?>
```

Clases en PHP5

- ❑ Las clases de programación orientada a objetos son definiciones de los elementos que forman un sistema.
- ❑ Un objeto se define indicando qué propiedades y funcionalidades tiene .

```
class Hombre{  
    var $nombre;  
    var $edad;  
  
    function comer($comida){  
        // definición del método  
    }  
    function moverse($destino){  
        // definición del método  
    }  
}
```

- ❑ Para instanciar un objeto a partir de la clase:

```
$pepe=new Hombre();
```

Constructores y destructores en PHP5

- ❑ Los constructores se encargan de resumir las acciones de inicialización de los objetos.
- ❑ Debe llamarse con un nombre fijo: `__construct()`
- ❑ Los destructores son funciones que se encargan de realizar las tareas que se necesita ejecutar cuando un objeto deja de existir. El objeto ya no está referenciado por ninguna variable.
- ❑ El destructor es opcional y debe llamarse con un nombre fijo:

`__destruct()`

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

Métodos mágicos en PHP5

- ❑ Los nombres de método siguientes son mágicos en las clases PHP
 - ❑ **__construct()**, **__destruct()**,
 - ❑ **__call()**, **__callStatic()**,
 - ❑ **__get()**, **__set()**,
 - ❑ **__isset()**, **__unset()**,
 - ❑ **__sleep()**, **__wakeup()**,
 - ❑ **__toString()**,
 - ❑ **__invoke()**,
 - ❑ **__set_state()** y **__clone()**

 - ❑ No se puede tener métodos con estos nombres en cualquiera de las clases a menos que desee la funcionalidad predefinida asociada a ellos.
-

Métodos mágicos en PHP5

❑ `__get()`, `__set()`

- ❑ Una práctica en programación es proteger los atributos de una clase con una visibilidad restrictiva y acceder a dichos atributos mediante métodos 'get' y 'set'.
 - ❑ Con los métodos mágicos `__set` y `__get` PHP5 implementa un mecanismo para poder modificar o acceder a los distintos atributos privados de la clase, evitando tener que crear un método para cada uno de ellos.
 - ❑ Una vez declarados, si se intenta acceder a un atributo como si fuera público, PHP llamará automáticamente a `__get()`. Y si asignamos un valor a un atributo, llamará a `__set()`.
 - ❑ `__set()` necesita dos parámetros de entrada: nombre del atributo y el valor a asignar.
 - ❑ `__get()` sólo necesita el nombre del atributo del que obtener su valor.
-

Métodos mágicos en PHP5

□ __get(), __set()

```
class Objeto{
    private $id;
    private $nombre;
    private $email;
    function __construct($id, $nombre, $email) {
        $this->id = $id;
        $this->nombre = $nombre;
        $this->email = $email;
    }
    function __clone(){
        $this->id = ++$this->id;
    }
    public function __set($var, $valor){
        if (property_exists(__CLASS__, $var)){
            $this->$var = $valor;
        } else
            echo "No existe el atributo $var.";
    }
    public function __get($var){
        if (property_exists(__CLASS__, $var)){
            return $this->$var;
        }
        return NULL;
    }
}

$obj = new Objeto(1, "objeto1", "prueba1@ejemplo.com");
$p = $obj;
echo $p->id; //2


$p->nombre = "nombre cambiado";


echo $p->nombre; //nombre cambiado
Echo $obj->nombre;
```


Métodos mágicos en PHP5

□ __toString

- Permite asignar una cadena (string) al objeto, que será mostrada donde el objeto sea usado como un string. Es decir, el valor que mostrará si se intenta hacer *echo \$objeto*.

```
class Objeto{
    private $id;
    private $nombre;
    private $email;

    //...
    public function __toString(){
        return $this->id.$this->nombre.$this->email;
    }
}
$obj = new Objeto(1, "objeto1", "prueba1 @ejemplo.com");
$p = clone $obj;
echo $p; //2objeto2prueba1 @ejemplo.com
```

Métodos mágicos en PHP5

❑ `__clone()`

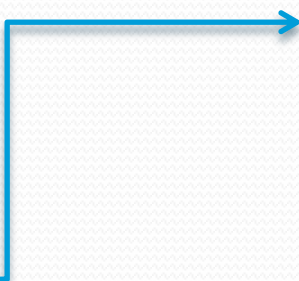
- ❑ En PHP 4 se podía copiar un objeto simplemente asignándolo de una variable a otra. Y de esta forma se podía modificar la copia sin que ello afectara al original.
 - ❑ En PHP 5 los objetos son asignados por referencia. Lo que significa que si en dos variables asignamos el mismo objeto lo que obtendremos será dos referencias al mismo objeto. Necesitamos algo más si queremos realizar una copia exacta pero no enlazada de un objeto
 - ❑ Para ello existe este método mágico.
-

Modificadores de acceso en PHP5

- ❑ Los nuevos modificadores de acceso que se han incorporado en PHP5:
 - ❑ **public** : los miembros son accedidos desde cualquier clase
 - ❑ **private**: el acceso a los miembros de la clase sólo es posible desde el propio objeto o a través de métodos de acceso.
 - ❑ **protected**: los elementos son públicos dentro de la propia clase y en sus heredadas.
 - ❑ La forma de declaración de una variable de PHP 4 con la palabra clave **var** todavía es soportado (como un sinónimo de public) por compatibilidad.
-

Constantes de clase en PHP5

- ❑ Es posible definir valores constantes en función de cada clase manteniéndola invariable. Se diferencian de variables en que no utilizan el símbolo \$ al usarlas.



```
<?php
class MyClass
{
    const CONSTANT = 'valor constante';
    function showConstant() {
        echo self::CONSTANT . "\n";
    }
}
echo MyClass::CONSTANT . "\n";
$classname = "MyClass";
echo $classname::CONSTANT . "\n"; // A partir de PHP 5.3.0
$class = new MyClass();
$class->showConstant();
echo $class::CONSTANT . "\n"; // A partir de PHP 5.3.0
?>
```

- ❑ A partir de PHP 5.3.0, es posible hacer referencia a una clase utilizando una variable. El valor de la variable no puede ser una palabra clave.

Sintaxis de la herencia

- ❑ La herencia en PHP se consigue indicando la cláusula ***extends*** seguida de la clase que proporciona la herencia.
 - ❑ Se heredan todos los métodos públicos y protegidos de la clase padre. A menos que una clase sobrescriba esos métodos, mantendrán su funcionalidad original.
 - ❑ Se pueden sobrescribir métodos heredados para adecuarlos al comportamiento de la clase hija.
 - ❑ Se utiliza la cláusula ***parent::*** para acceder a los miembros de la clase padre desde una clase heredada.
 - ❑ El método constructor de la clase hija se debe redefinir para ajustarlo a las variables miembro de dicha clase.
-

Sintaxis de la herencia

```
class cinta_video extends soporte{
    private $duracion;
    function __construct($tit,$num,$precio,$duracion){
        parent::__construct($tit,$num,$precio);
        $this->duracion=$duracion;
    }
    function imprime_caracteristicas(){
        echo "Película en VHS: <br>";
        parent::imprime_caracteristicas();
        echo "<br>Duracion: ".$this->duracion;
    }
}
```

Objetos como atributos de otros objetos

- ❑ Las características de los objetos se almacenan a través de los atributos de la clase.
- ❑ Podemos utilizar cualquier tipo para los atributos de los objetos, incluso otros objetos.
- ❑ Atributos de la clase Cliente:

[illegible]

Colaboración entre objetos

- Dentro de la clase **Pagina** definimos tres atributos de tipo objeto de las clases *Cabecera*, *Cuerpo* y *Pie* respectivamente. Luego, dentro de la clase *Pagina* creamos los tres objetos y llamaremos a sus métodos respectivos.

```
class Pagina {  
    private $cabecera;  
    private $cuerpo;  
    private $pie;  
    public function __construct($texto1,$texto2){  
        $this->cabecera=new Cabecera($texto1);  
        $this->cuerpo=new Cuerpo();  
        $this->pie=new Pie($texto2);  
    }  
    public function insertarCuerpo($texto){  
        $this->cuerpo->insertarParrafo($texto);  
    }  
}
```


Colaboración entre objetos

- ❑ Declaramos las cuatro clases siguiendo el esquema:

```
class Cabecera {  
    [atributos y métodos]  
}  
class Cuerpo {  
    [atributos y métodos]  
}  
class Pie {  
    [atributos y métodos]  
}  
class Pagina {  
    private $cabecera;  
    private $cuerpo;  
    private $pie;  
    [métodos]  
}  
$pag=new Pagina();
```

Colaboración entre objetos

□ En resumen:

- Cada clase debería servir para una sola cosa.
 - Es mejor que las clases sean lo más genéricas posibles y sus posibilidades no se reduzcan a solucionar algo en concreto.
 - Para que las clases colaboren entre sí se pueden crear objetos dentro de una clase.
-

Clases y métodos abstractos en PHP5

- ❑ Una clase abstracta es aquella que tiene métodos abstractos.
- ❑ Los métodos abstractos son aquellos que no incluyen una codificación, sino que simplemente se declaran, dejando para las clases que heredan esa codificación.
- ❑ Las clases que incorporan métodos abstractos se deben declarar como abstractas.
- ❑ Las clases abstractas no se pueden instanciar.
- ❑ Para declarar métodos y clases abstractas se utiliza ***abstract***.
- ❑ Las clases que heredan de una clase abstracta están obligadas a implementar los métodos abstractos o a volver a definirlos como abstractos.

Ejemplo 086

Ejemplo 085

Clases y métodos abstractos en PHP5

```
abstract class nombre_clase{  
    // propiedades  
    public x;  
    private y;  
  
    // métodos  
    public function __construct() {...}  
    public abstract function nombre_metodo();  
}
```

Interfaces en PHP5

- ❑ Son declaraciones de funcionalidades que tienen que cubrir las clases que implementan la interfaz.
 - ❑ En una interfaz se definen un juego de funciones, es decir, se declaran una serie de métodos o funciones sin especificar ningún código fuente asociado.
 - ❑ Las clases que implementan la interfaz serán las encargadas de proporcionar un código a los métodos que contiene esa interfaz.
 - ❑ Para definir una interfaz se utiliza la palabra clave **interface**.
 - ❑ Para implementar una interfaz se utiliza la palabra clave **implements**.
 - ❑ Se pueden implementar varias interfaces en una misma clase => herencia múltiple
-

Interfaces en PHP5

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

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

Interfaces en PHP5

```
class coche implements encendible{
    private $gasolina;
    private $batería;
    private $estado="apagado";
    function __construct() {
        $this->gasolina=0;
        $this->batería=10;
    }
    public function encender(){
        if $this->estado == "apagado"){
            if ($this->bateria > 0){
                if ($this->gasolina>0){
                    $this->estado="encendido";
                    $this->bateria--;
                    echo "<br><b>Enciendolo...</b> estoy encendido";
                }else{echo "<br>No tengo gasolina";}
            }else {"<br>No tengo bateria";}
        }else {"<br>Ya estaba encendido";}
    }
}
```

Interfaces en PHP5

```
class coche implements encendible{
    public function apagar(){
        if $this->estado == "encendido"){
            $this->estado="apagado";
            echo "<br><b>Apago...</b> estoy apagado";
        }else {"<br>No estaba encendido";}
    }
    public function cargar_gasolina($litros){
        $this->gasolina+=$litros;
        echo "<br>Cargados $litros litros";
    }
}
```

Llamadas polimórficas a funciones

- ❑ Las interfaces permiten el tratamiento de objetos sin necesidad de conocer las características internas de cada objeto o la clase a la que pertenecen, sólo la interfaz que implementan.
- ❑ En la declaración de la función especificamos que el parámetro recibido implementa una determinada interfaz que marcará el juego de acciones que puede desarrollar dicho objeto.

```
function enciende_algo (encendible $algo){  
    $algo->encender();  
}  
  
$mibombilla=new bombilla();  
$micoche=new coche();  
  
enciende_algo($mibombilla);  
enciende_algo($micoche);
```

Palabra clave *final*

PHP 5 introduce la nueva palabra clave **final**, que impide que las clases hijas sobrescriban un método, antecediendo su definición con la palabra final. Si la propia clase se define como final, entonces no se podrá heredar de ella.

```
<?php
class BaseClass {
    public function test() {
        echo "llamada a BaseClass::test()\n";
    }
    final public function moreTesting() {
        echo "llamada a BaseClass::moreTesting()\n";
    }
}

class ChildClass extends BaseClass {
    public function moreTesting() {
        echo "llamada a ChildClass::moreTesting()\n";
    }
}

// Devuelve un error Fatal: Cannot override final method BaseC
lass::moreTesting()
?>
```

Palabra clave *final*

Ejemplo clase final

```
<?php
final class BaseClass {
    public function test() {
        echo "llamada a BaseClass::test()\n";
    }

    // Aquí no importa si definimos una función como final o no
    final public function moreTesting() {
        echo "llamada a BaseClass::moreTesting()\n";
    }
}

class ChildClass extends BaseClass {
}
// Devuelve un error Fatal: Class ChildClass may not inherit from final class (BaseClass)
?>
```

Las propiedades no pueden declararse como final. Sólo pueden las clases y los métodos.

Palabra clave *static*

- ❑ Declarar propiedades o métodos de clases como estáticos los hacen accesibles sin la necesidad de instanciar la clase. Una propiedad declarada como *static* no puede ser accedida con un objeto de clase instanciado (aunque un método estático sí lo puede hacer).
 - ❑ Por motivos de compatibilidad con PHP 4, si no se utiliza ninguna declaración de visibilidad, se tratará a las propiedades o métodos como si hubiesen sido definidos como *public*.
-

Palabra clave *static*

- ❑ Las propiedades estáticas **no** pueden ser accedidas a través del objeto utilizando el operador flecha (->).
 - ❑ Invocar métodos no estáticos estáticamente genera una advertencia de nivel **E_STRICT**.
 - ❑ Debido a que los métodos estáticos se pueden invocar sin tener creada una instancia del objeto, la pseudo-variable *\$this* no está disponible dentro de los métodos declarados como estáticos.
-

Palabra clave *static*

- ❑ Las propiedades estáticas sólo pueden ser inicializadas utilizando un string literal o una constante; las expresiones no están permitidas.
- ❑ Por tanto, se puede inicializar una propiedad estática con enteros o arrays (por ejemplo), pero no se puede hacer con otra variable, con el valor de devolución de una función, o con un objeto.

Ejemplo 133

Operador de resolución de ámbito ::

- ❑ El operador de resolución de ámbito , el doble dos-puntos, es un token que permite acceder a elementos **estáticos**, **constantes**, y sobrescribir propiedades o **métodos** de una clase.
- ❑ Cuando se hace referencia a estos items desde el exterior de la definición de la clase, se utiliza el nombre de la clase.
- ❑ A partir de PHP 5.3.0, es posible hacer referencia a una clase usando una variable. El valor de la variable no puede ser una palabra clave

```
?php
class MyClass {
    const CONST_VALUE = 'Un valor constante';
}

$classname = 'MyClass';
echo $classname::CONST_VALUE; // A partir de PHP 5.3.0
echo MyClass::CONST_VALUE;
?>
```

Operador de resolución de ámbito ::

- Las tres palabras claves especiales *self* y *parent* son utilizadas para acceder a propiedades y métodos desde el interior de la definición de la clase.

```
?php
class OtherClass extends MyClass
{
    public static $my_static = 'variable estática';

    public static function doubleColon() {
        echo parent::CONST_VALUE . "\n";
        echo self::$my_static . "\n";
    }
}

$classname = 'OtherClass';
echo $classname::doubleColon(); // A partir de PHP 5.3.0

OtherClass::doubleColon();
?>
```


Control de excepciones

- ❑ Es responsabilidad del programador gestionar lógicamente los errores inevitables e impedir las terminaciones violentas de la aplicación.
 - ❑ La gestión de los errores de ejecución incluye brindar al usuario la información del error de un modo adecuado que le pueda resultar útil.
-

Tipos de error en PHP

- ❑ Los errores que se pueden producir en la ejecución de una aplicación se pueden clasificar como:
 - ❑ **Errores estándar.** Tienen diferente nivel de gravedad, algunos no permiten que prosiga la ejecución del programa y otros son simples advertencias de error.
 - ❑ **Errores personalizados.** Son errores definidos por el usuario para gestionar diversas situaciones de excepción.
-

Tipos de errores estándar en PHP

- ❑ E_COMPILE_ERROR (64)
 - ❑ E_COMPILE_WARNING (128)
 - ❑ E_CORE_ERROR (16)
 - ❑ E_CORE_WARNING (32)
 - ❑ E_ERROR (1)
 - ❑ E_NOTICE (8)
 - ❑ E_PARSE (4)
 - ❑ E_WARNING (2)
-

Tipos de errores personalizados en PHP

- ❑ `E_USER_ERROR` (256)
 - ❑ `E_USER_WARNING` (512)
 - ❑ `E_USER_NOTICE` (1024)
 - ❑ `E_ALL` : Todos los errores salvo `E_STRICT`.
 - ❑ `E_STRICT`: Avisos en tiempo de ejecución. Este nivel se habilita para que PHP sugiera cambios en el código.
-

Gestión personalizada de errores

- ❑ Para crear gestores de error en PHP se crea una función de usuario que utiliza cinco parámetros:
 - ❑ número de error a gestionar
 - ❑ descripción del error
 - ❑ archivo donde se produjo el error
 - ❑ número de línea
 - ❑ contexto

```
Function gestor_error($numero_error, $descripcion_error, $archivo_error,  
$linea_error, $contexto_error){  
    // código para el tratamiento de los errores  
}
```

Funciones para la gestión de errores

- ❑ PHP suministra un conjunto de funciones para el tratamiento de errores:
 - ❑ *error_log (pmensaje, ptipo, pdestino, pcabeceras);*
 - ptipo: 0: mensaje va al registro del servidor
 - 1: mensaje va a la dirección de correo de destino
 - 2: mensaje va a una IP de destino
 - 3: mensaje va a un fichero de destino
 - ❑ *set_error_handler(pgestor_error)*
 - ❑ *restore_error_handler()*
 - ❑ *error_reporting (pnivel): Establece qué errores se registran*
 - ❑ *trigger_error(pmensaje, ptipo): Genera un mensaje de error de usuario y pasa el control a la función gestora de errores.*

Ejemplo c801

Estructura try/throw/catch

- ❑ El código que se encuentra dentro de las palabras clave try/catch puede generar una excepción, que es una llamada a un gestor de excepciones.
- ❑ El bloque **try** es monitorizado a la espera de que se produzca una excepción.
- ❑ La excepción producida en el bloque try es tratada por uno de los bloques **catch(exception)** situados a continuación del try. Estos bloques contienen el código para gestionar la excepción o para relanzarla hacia otro gestor (**throw**)

Ejemplo c802 Ejemplo c803

Clase Exception

- ❑ La clase Exception es la clase base de todas las excepciones.
- ❑ Esta clase proporciona los métodos:
 - ❑ getMessage()
 - ❑ getPrevious()
 - ❑ getCode()
 - ❑ getFile()
 - ❑ getLine()
 - ❑ getTrace()
 - ❑ GetTraceAsString()
- ❑ Todos los métodos se han definido como final.

BIBLIOGRAFÍA

- ❑ D'andrea, Edgar. ***Cómo crear una tienda virtual con php 6 y mysql***, Inforbook's ,2010
 - ❑ Manual de PHP. Disponible en: <http://es2.php.net/manual/es/>
 - ❑ Manuales disponibles en: www.apache.org, www.php.net, www.mysql.com,
www.desarrolloweb.com
-