Desarrollo Web en Entorno Servidor

4.- Objetos en PHP II

IES Severo Ochoa

Índice

- Metodos encadenados
- Traits
- Métodos mágicos
- Namespaces
- Gestión de Errores
  - Niveles de Error
  - Excepciones
- SPL

Métodos encadenados

- Programación funcional
- Sobre un objeto se realizan varias llamadas
- Los métodos mutadores (que cambian el contenido) devuelven `$this`.

```php
$p1 = new Libro();
$p1->setNombre("Harry Potter");
$p1->setAutor("JK Rowling");
echo $p1;

// Method chaining
$p2 = new Libro();
$p2->setNombre("Patria")->setAutor("Aramburu");
echo $p2;
```

POO con encadenamiento

ies severo ochoa

```php
<?php
class Libro {
    private string $nombre;
    private string $autor;

    public function getNombre() : string {
        return $this->nombre;
    }
    public function setNombre(string $nombre) : Libro {
        $this->nombre = $nombre;
        return $this;
    }

    public function getAutor() : string {
        return $this->autor;
    }
    public function setAutor(string $autor) : Libro {
        $this->autor = $autor;
        return $this;
    }

    public function __toString() : string {
        return $this->nombre." de ".$this->autor;
    }
}
```

Traits

ies severo ochoa

- Permite reutilizar código
- No se puede instanciar
- Solución para la herencia múltiple

```
trait Nombre {
  …
}
```

- Dentro de la clase → `use NombreTrait`
- Similar a copiar y pegar el código dentro la clase que lo usa

Ejemplo Trait Singleton

```php
<?php
trait Singleton {
  private static $instance;

  public static function getInstance() {
   if ( is_null( self::$instance ) ) {
    self::$instance = new self();
   }

   return self::$instance;
  }
}
```

```php
class Cero {
}
class Uno extends Cero  {
  use Singleton;
}
class Dos extends Cero {
  use Singleton;
}

$a = Uno::getInstance();
$b = Dos::getInstance();
```

Magic methods I

- Todas las clases PHP ofrecen un conjunto de métodos que se pueden sobreescribir para sustituir su comportamiento.
- https://www.php.net/manual/es/language.oop5.magic.php
- `__construct()`
- `__destruct()` → se invoca al perder la referencia
  - Cerrar una conexión a BD, cerrar un fichero
- `__toString()` → representación del objeto como cadena

Magic Methods II

- `__get(propiedad), __set(propiedad, valor)`
  - Permitiría acceder a las propiedad privadas
  - Mejor codificar los getter/setter
- `__isset(propiedad), __unset(propiedad)`
- `__sleep(), __wakeup()`
  - Se ejecutan al recuperar (unserialize) o almacenar un objeto que se serializa (serialize).
  - Permite definir que propiedades se serializan
- `__call(), __callStatic()`
  - Se ejecutan al llamar a un método que no es público. Permiten sobrecargan métodos

Namespaces

ies severo ochoa

- >= PHP 5.3
- Similar a los paquetes Java
- Permite organizar las clases/interfaces/traits, funciones y/o constantes.

- Recomendable: un sólo namespace por archivo y colocarlos en carpetas
  - Igual que Java

Declaración

- Se declara en la primera línea
- **`namespace`** `Nombre;`
  - `namespace Nombre\Subnombre;`

```php
<?php
namespace Dwes\Ejemplos;

const IVA = 0.21;

class Producto {
  public $nombre;

  public function muestra() : void {
      print "<p>Prod:" . $this->nombre . "</p>";
  }
}
```

Uso de Namespaces

- Para acceder a un elemento (constante, función, clase ...):
  - Primero hemos de hacerlo disponible: `include/require`.
  - Después podemos acceder a él.
    - Si tenemos dos recursos en el mismo NS, podemos acceder a él directamente (sin cualificar)

- El acceso puede ser:
  - Sin cualificar: `recurso`
  - Cualificado: `rutaRelativa/recurso`
  - Totalmente cualificado: `/rutaAbsoluta/recurso`

Acceso

```php
<?php
namespace Dwes\Ejemplos;

include_once("ejemploNamespace.php");

echo IVA; // sin cualificar
echo Utilidades\IVA; // daría error, no existe utilidades
// cualificado -> \Dwes\Ejemplos\Utilidades\IVA
echo \Dwes\Ejemplos\IVA; // totalmente cualificado

$p1 = new Producto(); // \Dwes\Ejemplos\Producto
$p2 = new Model\Producto(); // daría error, no existe model
 // \Dwes\Ejemplos\Model\Producto
$p3 = new \Dwes\Ejemplos\Producto();
// \Dwes\Ejemplos\Producto
```

use

- Para evitar la referencia cualificada podemos declarar el uso.
- Se hace en la cabecera, tras el namespace

```
use const nombreCualificadoConstante
use function nombreCualificadoFuncion
use nombreCualificadoClase
```

- También podemos renombrar elementos

```
use nombreCualificadoClase as NuevoNombre
```

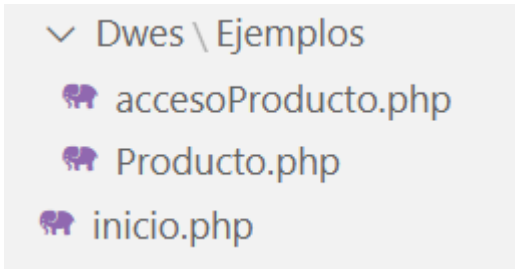
Ejemplo use

```php
<?php
include_once("ejemploNamespace.php");

use const Dwes\Ejemplos\IVA;
use \Dwes\Ejemplos\Producto;

echo IVA;
$p1 = new Producto();
```

- No necesitamos estar en el mismo namespace

Organización

- Crear NS anidados para separar funcionalidades
- Crear una organización de carpetas similar



Dwes > Ejemplos > Producto.php > ...

```php
<?php
namespace Dwes\Ejemplos;

const IVA = 0.21;

class Producto {
    public $nombre;

    public function muestra() : void {
        print "<p>Prod:" . $this->nombre . "</p>";
    }
}
```

inicio.php > ...

```php
<?php
include_once("Dwes/Ejemplos/Producto.php");

use const Dwes\Ejemplos\IVA;
use Dwes\Ejemplos\Producto;

echo IVA;
$p1 = new Producto();
```

Autoload

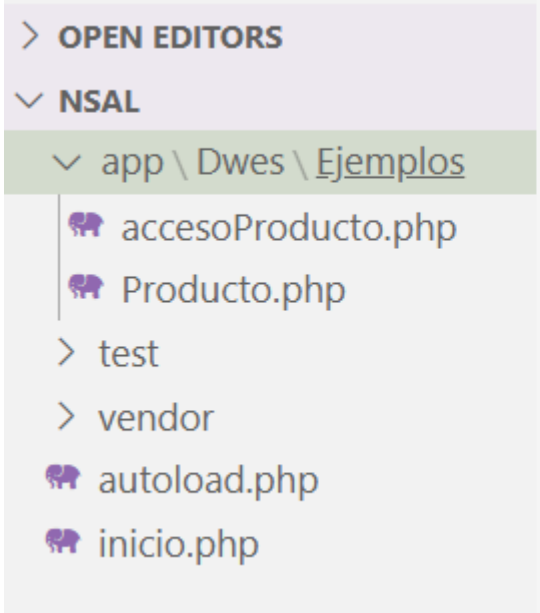
- Permite cargar las clases (no las constantes ni las funciones) que se van a utilizar y evitar tener que hacer el `include_once` de cada una de ellas.
  - **`spl_autoload_register`**

```php
spl_autoload_register( function( $NombreClase ) {
    include_once $NombreClase.'.php';
} );
```

- Antes se hacía con `__autoload()`
  - *Deprecated* desde PHP7.2

Organizando con autoload

- Colocamos el código dentro de una carpeta `app`
  - Más tarde colocaremos las pruebas en `test`
  - Librerías en `vendor` → Composer



inicio.php

```php
<?php
include("autoload.php");

use Dwes\Ejemplos\Producto;

echo Producto::IVA;
$p1 = new Producto();

```

autoload.php

```php
<?php
spl_autoload_register( function( $NombreClase ) {
    include_once "app/".$NombreClase.'.php';
} );
```

Gestión de errores

- PHP clasifica los errores que ocurren en diferentes niveles.
- Cada nivel se identifica con una constante.
  - `E_ERROR`: errores fatales, no recuperables. Se interrumpe el script.
  - `E_WARNING`: advertencias en tiempo de ejecución. El script no se interrumpe.
  - `E_NOTICE`: avisos en tiempo de ejecución.

- Listado de constantes: https://www.php.net/manual/es/errorfunc.constants.php

Configuración de errores

- A nivel de `php.ini`:
- `error_reporting`: indica los niveles de errores a notificar
  - `error_reporting = E_ALL & ~E_NOTICE`
    - Todos los errores menos los avisos en tiempo de ejecución.
- `display_errors`: indica si mostrar o no los errores por pantalla
  - En entornos de producción es común ponerlo a `off`

Configurando errores en código

ies severo ochoa

- `error_reporting(codigo)`
  - Controla qué errores notificar
- `set_error_handler(nombreManejador)`
  - Indica que función se invocará cada vez que se encuentre un error
  - El manejador recibe como parámetros el nivel del error y el mensaje

Ejemplo manejo errores

```php
<?php
error_reporting(E_ALL & ~E_NOTICE & ~E_WARNING);
$resultado = $dividendo / $divisor;

error_reporting(E_ALL & ~E_NOTICE);
set_error_handler("miManejadorErrores");
$resultado = $dividendo / $divisor;
restore_error_handler(); // vuelve al anterior

function miManejadorErrores($nivel, $mensaje) {
    switch($nivel) {
        case E_WARNING:
            echo "<strong>Warning</strong>: $mensaje.<br/>";
            break;
        default:
            echo "Error de tipo no especificado: $mensaje.<br/>";
    }
}
```

Error de tipo no especificado: Undefined variable: dividendo.
Error de tipo no especificado: Undefined variable: divisor.
Error de tipo Warning: Division by zero.

Excepciones

- PHP >= 5
- Similar a Java
  - `try / catch / finally`
  - throw new `Exception`
    - Adjuntar mensaje[, código][y excepción] que lo ha provocado.

```php
<?php
try {
  if ($divisor == 0) {
    throw new Exception("División por cero.");
  }
  $resultado = $dividendo / $divisor;
} catch (Exception $e) {
  echo "Se ha producido el siguiente error: ".$e->getMessage();
}
```

Exception

- Clase padre de todas las excepciones
- Constructor: `mensaje[,codigoError][,excepcionPrevia]`
- A partir de un objeto `Exception`:
  - `getMessage()`
  - `getCode();`

Creando excepciones

- Heredan de `Exception`
- Consejos:
  - Sobrecargar el constructor y llamar al constructor del padre
  - Sobrecargar `__toString` ¿y llamar al método del padre?

```php
<?php
class MiExcepcion extends Exception {
    public function __construct($msj, $codigo = 0, Exception $previa = null) {
        // código propio
        parent::__construct($msj, $codigo, $previa);
    }
    public function __toString() {
        return __CLASS__ . ": [{$this->code}]: {$this->message}\n";
    }
    public function funciónPersonalizada() {
        echo "Una función personalizada para este tipo de excepción\n";
    }
}
```

Excepciones múltiples

- Se pueden usar excepciones múltiples para comprobar diferentes condiciones.
  - De más específica a más general.

```php
<?php
$email = "ejemplo@ejemplo.com";
try {
    // Comprueba si el email es válido
    if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE) {
        throw new MiExcepcion($email);
    }
    // Comprueba la palabra ejemplo en la dirección email
    if(strpos($email, "ejemplo") !== FALSE) {
        throw new Exception("$email es un email de ejemplo");
    }
} catch (MiExcepcion $e) {
    echo $e->miFuncion();
} catch(Exception $e) {
    echo $e->getMessage();
}
```

¿Qué pasaría…?

```php
<?php
class MainException extends Exception {}
class SubException extends MainException {}


try {
    throw new SubException("Lanzada SubException");
} catch (MainException $e) {
    echo "Capturada MainException " . $e->getMessage();
} catch (SubException $e) {
    echo "Capturada SubException " . $e->getMessage();
} catch (Exception $e) {
    echo "Capturada Exception " . $e->getMessage();
}
```

Capturada MainException Lanzada SubException

Capturando múltiples Excepciones

ies severo ochoa

- Operador |

```php
<?php
class MainException extends Exception {}
class SubException extends MainException {}

try {
    throw new SubException("Lanzada SubException");
} catch (MainException | SubException $e ) {
    echo "Capturada Exception " . $e->getMessage();
}
```

Relanzar excepciones

- Capturar una excepción de sistema y lanzar una de aplicación.

```php
<?php
class AppException extends Exception {}

try {
    // Código de negocio que falla
} catch (Exception $e) {
    throw new AppException("AppException: ".$e->getMessage(),
        $e->getCode(), $e);
}
```

Throwable

- PHP >= 7
- Interfaz que implementan tanto los errores como las excepciones

```php
<?php
try {
    // tu codigo
} catch (Throwable $e) {
    echo 'Forma de capturar errores y excepciones a la vez';
}
```

Error

- PHP >= 7
- Clase que agrupa todos los errores fatales.

```php
try {
  // Genera una notificación que no se captura
  echo $variableNoAsignada;
  // Error fatal que se captura
  funcionQueNoExiste();
} catch (Error $e) {
  echo "Error capturado: " . $e->getMessage();
}
```

SPL

ies severo ochoa

- Standard PHP Library
- https://www.php.net/manual/es/book.spl.php
- Estructuras de datos
  - Pila, cola, cola de prioridad, lista doblemente enlazada, etc...
- Conjunto de iteradores diseñados para recorrer estructuras agregadas
  - arrays, resultados de bases de datos, árboles XML, listados de directorios, etc.
  - https://diego.com.es/tutorial-de-la-libreria-spl-de-php

Excepciones SPL

- `LogicException` (extends `Exception`)
  - `BadFunctionCallException`
  - `BadMethodCallException`
  - `DomainException`
  - `InvalidArgumentException`
  - `LengthException`
  - `OutOfRangeException`
- `RuntimeException` (extends `Exception`)
  - `OutOfBoundsException`
  - `OverflowException`
  - `RangeException`
  - `UnderflowException`
  - `UnexpectedValueException`

¿Alguna pregunta?