línea horizontal

## Producción web

Profesor: Fernando Gonzalo Gaitán  
Carrera: Analista de sistemas  
Año: 2024

Primera escuela de arte multimedial Da Vinci

Programación orientada a objetos con PHP - Parte 2

[Producción web](#_heading=h.gjdgxs) 1

[**Constantes**](#_heading=h.428y96nq1ybb) **1**

[**Métodos estáticos y propiedades estáticas**](#_heading=h.ugy8iyukvvlm) **2**

[Métodos estáticos](#_heading=h.8jxlutdt4z6d) 2

[Propiedades estáticas](#_heading=h.5ex56cgz5qka) 4

[**Interfaces**](#_heading=h.4tnk4y4eeq8) **5**

[**Excepciones**](#_heading=h.5qk8jauq15zr) **6**

[¿Cómo crear una excepción?](#_heading=h.4hlhs2331hlu) 7

# Constantes

Dentro de una clase se puede definir una constante:

class MiClase

{

const CONSTANTE = 'Soy una constante.';

}

Y llamarla desde la clase sin tener que instanciar un objeto:

echo MiClase::CONSTANTE;

Recordar que el valor de una constante no se puede modificar.

También puedo acceder a una constante dentro de un método como cualquier propiedad, pero en lugar de usar: “$this->” debo usar la palabra reservada: “self::”:

class MiClase

{

const CONSTANTE = 'Soy una constante.';

public function getConstante()

{

return self::CONSTANTE;

}

}

# Métodos estáticos y propiedades estáticas

Se usa la palabra reservada static para definir propiedades y métodos que pueden ser accedidos desde la clase sin tener que crear un objeto.

## Métodos estáticos

Podemos definir un método estático de la siguiente forma:

<?php

class MiClase

{

public static function metodoEstatico()

{

return 'Soy un método estático';

}

}

Y para llamar a ese método estático lo haremos desde la misma clase seguido por dos puntos y el nombre del método:

Dentro de un método estático no podemos acceder a los métodos que no son estáticos, pero sí en cambio podemos acceder a los métodos que sí lo son. En este caso usaremos la palabra reservada “self::” en lugar de “$this->”:

<?php

class MiClase

{

public static function realizarActividad()

{

self::hacerPaso1();

self::hacerPaso2();

self::hacerPaso3();

}

private static function hacerPaso1()

{

//

}

private static function hacerPaso2()

{

//

}

private static function hacerPaso3()

{

//

}

}

Tener en cuenta también, que aunque el método sea estático, la visibilidad, en caso de que sea un método privado o protected, tampoco podrá accederse desde afuera de la clase. Es decir, podemos llamar solamente a un método estático público desde la clase:

echo MiClase::metodoEstaticoPublico();

## Propiedades estáticas

Una propiedad estática puede accederse desde un método estático con “self::”:

class MiClase

{

private static $propiedad\_estatica = 'Soy una propiedad estática';

public static function getPropiedadEstatica()

{

return self::$propiedad\_estatica;

}

}

También podemos llamar a una propiedad estática desde un método que no sea estático, pero siempre con la palabra reservada: “self::”, en lugar de “$this->”.

Por ejemplo, podemos crear una propiedad estática para contar la cantidad de instancias de la clase e incrementar dentro del constructor:

class MiClase

{

private static $contador\_de\_objetos = 0;

public function \_\_construct()

{

self::$contador\_de\_objetos++;

}

public static function getContadorDeObjetos()

{

return self::$contador\_de\_objetos;

}

}

$objeto1 = new MiClase();

$objeto2 = new MiClase();

$objeto3 = new MiClase();

echo 'En total se han creado: ' . MiClase::getContadorDeObjetos() . ' objetos.';

# 

# Interfaces

Define los métodos que debe tener una clase en forma obligatoria.

También aclarar que la interfaz sólo muestra los métodos que hay que declarar, pero no cómo se implementan estos, ya que será la clase la que se encargue de esto.

Una interfaz se escribe con la palabra reservada “interface” y por convención con la primer letra “i” en minúscula:

<?php

interface iAuto {

public function revisarMotor();

}

Para implementar una clase de esta interfaz debemos usar la palabra reservada “implements”:

class Auto implements iAuto {

private $motor;

}

En este ejemplo debería indicarnos error, porque estamos implementando de la iAuto, pero no estamos creando los métodos (o el método) que nos exige dicha interfaz:

class Auto implements iAuto {

private $motor;

public function revisarMotor()

{

return $this->motor;

}

}

# 

# 

# Excepciones

Una excepción está formada por tres partes (la tercera es opcional):

* try: Un bloque de código que puede llegar a provocar una situación inesperada.
* catch: Se va a ejecutar en caso que la excepción exista.
* finally: Luego de terminados los dos anteriores, aunque no haya una excepción.

La más común es la conexión a una base de datos.

Nuestra aplicación y la base de datos funcionan como dos servicios independientes, por tanto, nuestro código debe estar preparado para realizar una acción excepcional si no se puede conectar a la base de datos:

try {

$pdo = new PDO('mysql:host=localhost;dbname=test;charset=utf8', 'root', '');

}catch(PDOException $e){

echo $e->getMessage();

}

## ¿Cómo crear una excepción?

También podemos crear nuestras propias excepciones.

supongamos que un archivo llamado archivo.txt contiene un texto: “Hola, soy un archivo.”

Para leerlo creamos una función como tal:

function leerArchivo($p\_filename)

{

return file\_get\_contents($p\_filename);

}

echo leerArchivo(archivo.txt');

Hasta ahí todo bien, ¿pero y si el archivo se borrase accidentalmente?

Con una excepción podríamos hacer algo como esto:

function leerArchivo($p\_filename)

{

$texto = null;

//is\_file devuelve TRUE si el archivo existe.

if(is\_file($p\_filename)){

$texto = file\_get\_contents($p\_filename);

}else{

//Generamos la excepción.

throw new Exception("El archivo {$p\_filename} no existe.");

}

return $texto;

}

try{

$texto = leerArchivo('archivo.txt');

echo $texto;

}catch(Exception $e){

echo $e->getMessage();

}