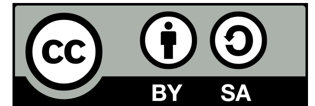
CURSO .PHP

PROGRAMACIÓN ORIENTADA A OBJETOS

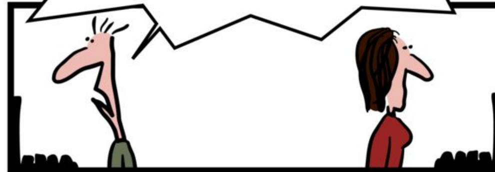


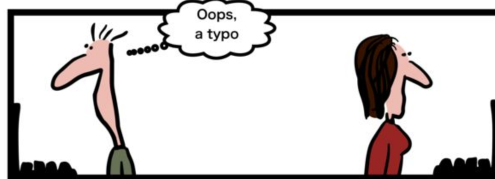
Autor: Jon Vadillo

www.jonvadillo.com

Contenidos

- Fundamentos básicos
- Definir una clase
- Crear una instancia
- Modificadores
- Herencia
- Clases abstractas
- static
- Interfaces
- Excepciones



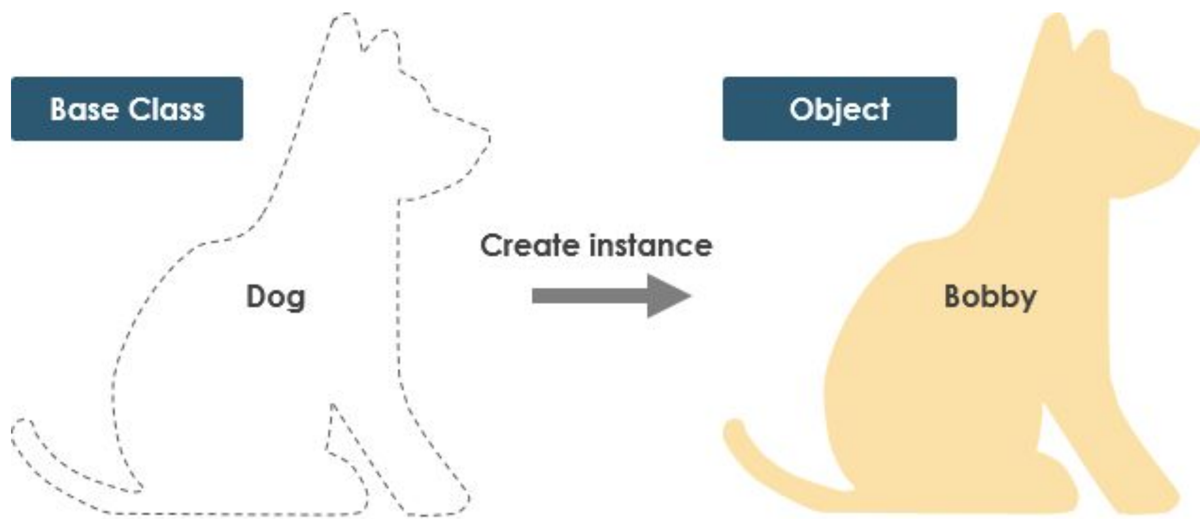


Just happened

Source: http://geek-and-poke.com/

Clase

- Concepto utilizado para **describir entidades (objetos)**.
- Están compuestos por:
  - Propiedades: definen cómo son los objetos y su **estado**.
  - Métodos: definen el **comportamiento**, lo que pueden hacer los objetos de esa clase.
- Cada objeto creado de una clase se conoce como instancia.



Properties	Methods	Property Values	Methods
Color	Sit	Color: Yellow	Sit
Eye Color	Lay Down	Eye Color: Brown	Lay Down
Height	Shake	Height: 17 in	Shake
Length	Come	Length: 35 in	Come
Weight		Weight: 24 pounds	

Source: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/

Clase

- El constructor se define con el método **`__construct()`**
  - Siempre será invocado al crear una instancia.
- Las propiedades (atributos) se declaran privados (solo accesibles desde dentro) y los métodos públicos.
- Para crear una instancia se utiliza la palabra reservada **`new`**
- **`$this`** hace referencia a la propia instancia creada.
- Una buena práctica es solo crear una clase por archivo (tendrá el mismo nombre que la clase y extensión .php)

Definir una clase

```
class Persona {

    private $nombre, $apellido;

    // constructor
    public function   construct($nombre, $apellido) {
        $this->nombre = $nombre;
        $this->apellido = $apellido;
    }

    public function saluda() {
        echo "Hola, me llamo " . $this->nombre . " " . $this->apellido;
    }
}

$alex = new Persona("Manu", "García");
$alex->saluda();
```

Crear una instancia empleando un String

```
$nombreDeClase = 'Persona';


$objeto = new Persona; // new Persona

$objeto = new $nombreDeClase; // new Persona

$objeto = new $nombreDeClase(); // new Persona
```

Hands on!

- Crea una clase con 3 atributos privados, un constructor que los inicialice y dos métodos públicos. A continuación instancia dos objetos distintos y llama a sus métodos.

Modificadores

- Public: cualquiera puede acceder a la variable.
- Private: solo accesible desde la clase que los declara.
- Protected: accesible desde la clase que los declara o sus descendientes.
- Final: sus descendientes no pueden sobreescribir el valor.
- Abstract: solo se puede utilizar una vez se ha definido en la subclase.

Modificadores

```php
class Persona
{
    public $nombre = 'Nombre Public';
    protected $apellido =
                         'Apellido Protected';
    private $edad = 'Edad Private';

    function pruebaModificadores()
    {
        echo $this->nombre;
        echo $this->apellido;
        echo $this->edad;
    }
}
```

```php
$persona = new MyClass();
echo $persona->public; // OK
echo $persona->protected; // Error
echo $persona->private; // Error
$persona->pruebaModificadores(); // OK
```

Espacios de nombres

```
Class demoClass
{

}

class demoClass
{

}
```

Can not have same name classes in one project

©www.besthinditutorials.com

All classes at one place

namespace 1

```
class demoClass
{

}
```

namespace 2

```
class demoClass
{

}
```

Classes in different different namespace can not cause error

Espacios de nombres

- En aplicaciones que utilicen varias librerías, pueden coincidir nombres de clases y causar problemas.
- Los espacios de nombres (Namespaces) resuelven el problema.
- Un namespace es un contenedor abstracto que agrupa clases, funciones, constantes, etc.

Espacios de nombres

Coche.php

```php
namespace MiApp\Modelo;

class Coche {
    //codigo de la clase
}
```

index.php

```php
include 'MiApp/Modelo/Coche.php' ;

$coche1 = new MiApp\Modelo\Coche;
```

Nota: es una buena práctica utilizar como namespace el directorio donde está la clase.

use

Coche.php

```php
namespace MiApp\Modelo;

class Coche {
    //codigo de la clase
}
```

index.php

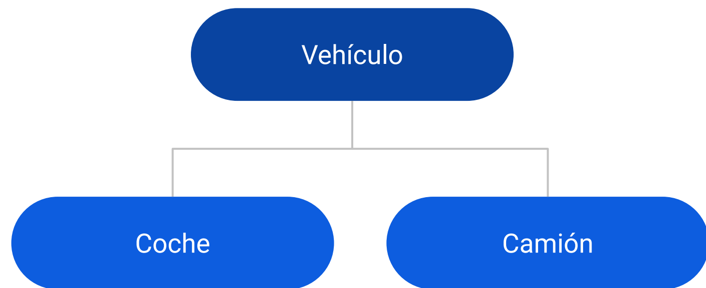
```php
include 'MiApp/Modelo/Coche.php' ;

use MiApp\Modelo\Coche;

$coche1 = new Coche;
```

Herencia

- Permite reutilizar una clase ya definida y extenderla.
- Una clase **sólo puede heredar de una única clase.**
- Se utiliza la palabra extends seguida de la clase a heredar.



Herencia

```php
class Persona {

    private $nombre;

    // constructor
    public function __construct($nombre) {
        $this->nombre = $nombre;
    }

    public function saluda() {
        echo "Hola, me llamo " . $this->nombre;
    }
}
```

```php
class Estudiante extends Persona {
    public function estudiar(){
        echo "Estoy estudiando";
    }
}

// Instancia de la subclase
$obj = new Estudiante("Mikel");
$obj->saludar();
$obj->estudiar();
```

parent::

- Las propiedades y los métodos heredados pueden ser sobreescritos (excepto los definidos como final).
    - Para ello basta con declararlos en la subclase con el mismo nombre.
    - Es posible acceder a los métodos sobreescritos utilizando la palabra reservada parent::

```php
class Estudiante extends Persona
{
    // Sobreescribir el método
    function saludar() {
        echo "Soy un estudiante\n";
        parent::saludar();
    }
}

$estudiante = new Estudiante();
$estudiante->saludar();
```

Hands on!

- Crea una clase llamada `Poligono` con 3 variables (color, altura y anchura), cada una con sus getters y setters. A continuación crea dos subclases de Polígono llamadas `Triángulo` y `Cuadrado`. Ambas tendrá un método llamado `area()` que calculará su área.

Clases abstractas

- No se pueden instanciar, se instancian las subclases.
- Toda clase que contenga un método abstracto deberá estar definida como clase abstracta, utilizando la palabra `abstract`.
- Las clases abstractas también pueden contener métodos comunes.
- **Los métodos abstractos no pueden estar implementados**, solo se declara la firma.
- Las subclases deben implementar **todos** los métodos abstractos.

Hands on!

- Convierte la clase Poligono del ejemplo anterior en clase abstracta y añade el método abstracto `area()`. ¿Qué ventaja obtenemos frente a la implementación anterior?

static

- Un método o propiedad declarado como estático es **accesible aunque no exista ninguna instancia** de la clase.
- Métodos estáticos:
  - Se ejecutan sobre la clase, no en instancias creadas.
  - No pueden emplear la pseudovariable **`$this`**
- Propiedades estáticas:
  - Solo existe una copia para la clase, no pertenece al objeto.
  - No se pueden acceder mediante **`->`**

static

```
class Persona {
    public static $miEstatica = "Variable estática";
    public static function mostrarMiEstatica() {
        return self::$miEstatica;
    }
}
class MiOtraClase extends MiClase {
    public function mostrarMiEstaticaDeNuevo(){
        return parent::$miEstatica;
    }
}
```

static

```
//Todas las formas son equivalentes:

echo MiClase::miMetodo();

echo MiClase::$miEstatica;

$clase = "MiClase";
echo $clase::miMetodo();


$miclase = new MiClase;
$miclase->miMetodo();
```

Hands on!

- Crea una clase llamada Persona con una variable estática llamada “personasEnElMundo” e incializada a cero. El constructor deberá incrementar la variable cada vez que se crea un objeto. Crea 4 objetos y a continuación muestra la variable estática por pantalla.

Interfaces

- Una interfaz (interface) es un “acuerdo” que debe cumplir la clase que lo implemente. Contiene **la declaración de métodos** que las clases tendrán que implementar.
- También **puede contener constantes** (no podrán ser sobreescritas).
- La clase que implemente una interface tendrá que contener todos sus métodos. Los métodos definidos serán todos **públicos**.
- Una clase puede implementar **múltiples interfaces.**

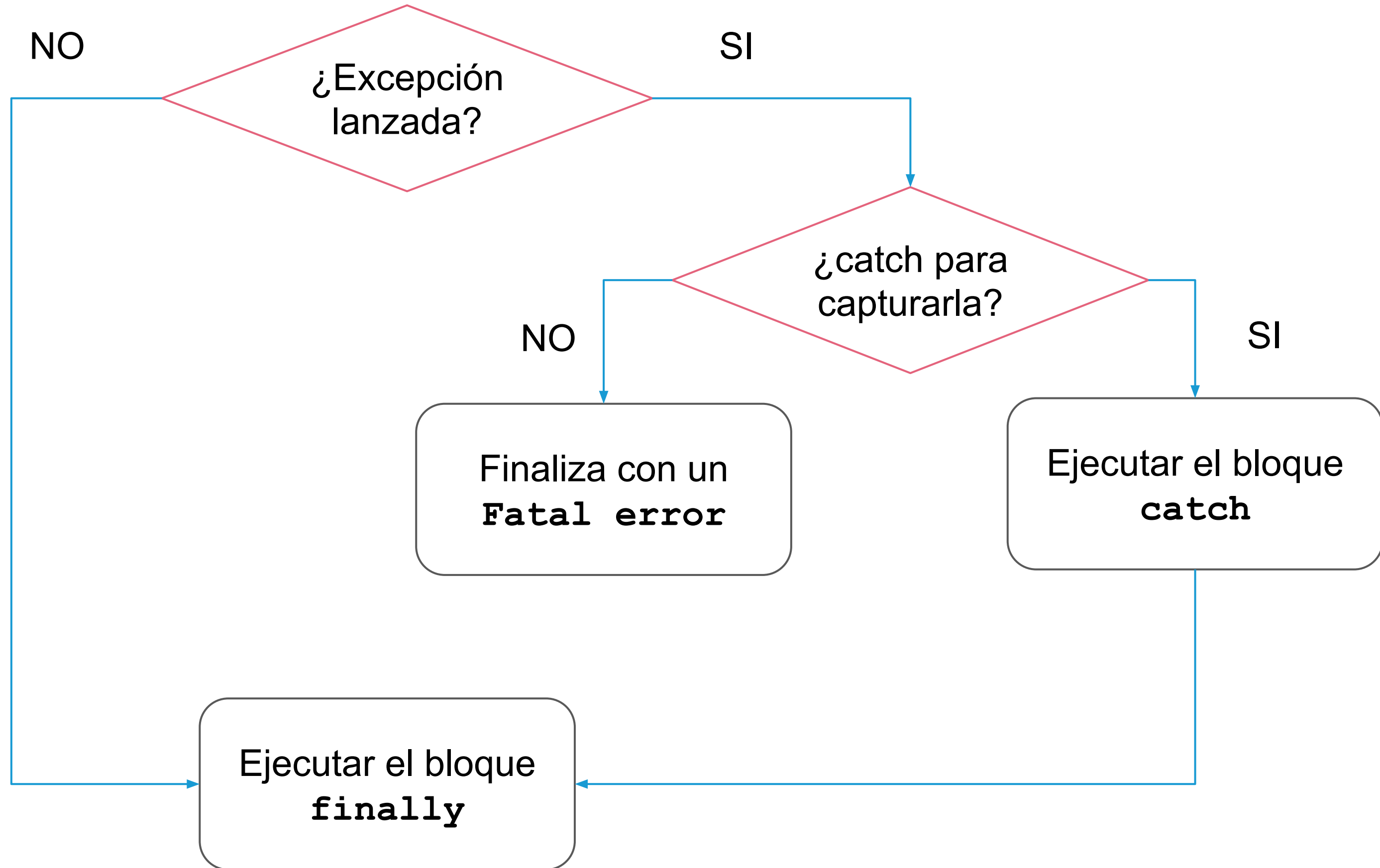
Interfaces

```
interface Vehiculo {
    public function acelerar();
    public function frenar();
}

class Coche implements Vehiculo {
    public function acelerar(){
        echo "He acelerado a 100 km/h";
    }
}
```

Excepciones

- Facilitan el **manejo de errores**. Las excepciones pueden ser lanzadas y capturadas:
  - Lanzar una excepción: es una sencilla forma de **informar** de un error de forma controlada.
  - Capturar la excepción: se define un bloque de código (**try**) a ejecutar, el cual es capaz de **capturar** las excepciones que ocurran en su interior y así **reaccionar** de forma controlada ante los errores.



NO
¿Excepción lanzada?
SI
¿catch para capturarla?
NO
SI
Finaliza con un Fatal error
Ejecutar el bloque catch
Ejecutar el bloque finally

Excepciones

```
try {
    // codigo

    // Si algo va mal, pueden saltar excepciones

    // codigo: no se ejecuta si ha ocurrido una excepción
} catch (Exception $e) {
    // la excepción es capturada y se ejecuta el bloque
    // $e->getMessage() contiene el mensaje de error.
} finally {
    // codigo: siempre se ejecuta
}
```

Hands on!

- Crea una función que reciba dos números como parámetros y realice la división del primero entre el segundo. La función debe lanzar una excepción en caso de que el valor del segundo parámetro sea igual a cero. Tendrá que capturar la excepción y mostrar un mensaje indicando que no ha podido realizarse la operación.

Sources

- PHP Group: https://www.php.net/
- PHP The Right Way: https://phptherightway.com
- WikiBooks PHP: https://en.wikibooks.org/wiki/PHP_Programming