Objetos:

**Tienen Clases y Propiedades**

**Class Persona{**

**}**

**$alumno { $nombre, $apellido1, ap2…**

**$ nombre = $\_POST[‘nombre’]**

**}**

Clase persona

**Propiedades**:

Nombre

Edad

Peso // esto seria un array pq va cambiando, no es fijo al igual que la altura

Edadactual (fechaactual-fechanacimiento)

Dni….

**Hay distintas formas de definir objetos, mediante el mismo objeto:**

**Básica: (mal)**

$miCoche = {  
 public $color = ‘rojo’ ;  
 private $extras = ‘techo solar’ ;

}

**Mediante instancias (creando primero una clase que es la forma más correcta):**

Class coche {  
 $color;  
 $extras;  
}

$miCoche = new Coche();  
$miCoche -> color = ‘Rojo’; /// $\_POST[‘color’] si es un form  
$miCoche -> potencia = ‘120’;

O $miCoche -> color = $\_POST[‘color’] (en formulario).

**Las instancias pueden ir cogidas de la mano de un constructor.**

**También con una función constructora:**

function \_\_construct ($color,$pot){  
 $this -> color = $color;  
 $this -> potencia = $pot;

}  
**Este function tiene que estar DENTRO de la clase.**

**Y luego haciendo una instancia (puede estar fuera de la clase con los parámetros del constructor:**

$miNuevoCoche = new Coche (‘rojo’ , ‘120’);

O $miNuevoCoche = new Coche (“$\_POST[‘color’], $\_POST[‘potencia’];

**\*\* Si es una propiedad del objeto es privada, solo se puede acceder a ella desde dentro del objeto, si se quiere acceder desde fuera, se tiene que usar un método (o función).**

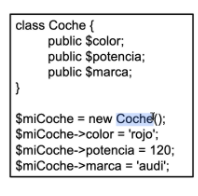
**Si asignamos una propiedad como $color-> rojo, sin poner el tipo, por defecto será pública (aunq es una mala práctica). Lo correcto seria public $color = ‘rojo’ o**

**private $color = ‘rojo’.**

**Métodos seters y getters**

Setters son los métodos que introducen un dato ( como en el peso que va cambiando)

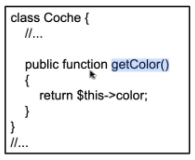
Getters otros métodos con los que se extraen esos datos.



**El New coche() es una función llamada constructor**, que crea un nuevo objeto con la clase llamado mi coche y le da las propiedades color potencia y marca. Ese proceso se llama Setter ya que le estamos aplicando valores a esas propiedades.



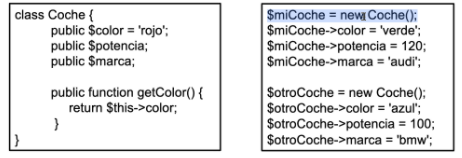
Esta función te devuelve el color del objeto donde se esta aplicando y luego con un echo puedes llamar a la función:

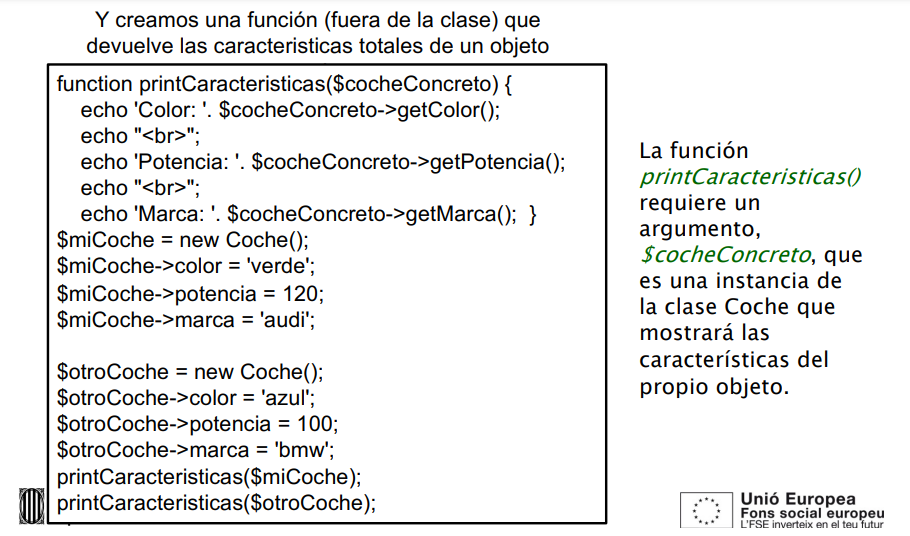




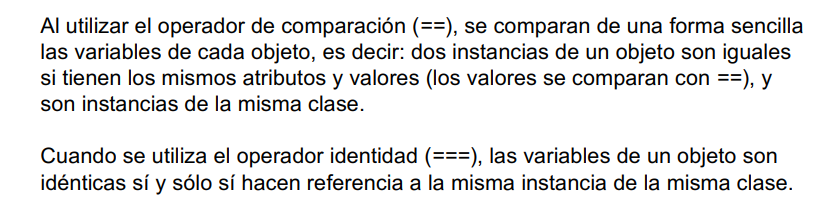
En este caso se aplica el color rojo a todos los coches con la clase coche a no ser que luego se cambie el valor. Es decir cuando creas $miCoche = newCoche() en este momento el color es rojo pero al poner $miCoche-> color =’verde’; se cambia el valor, pero si no se introduce nada, el color saldrá rojo por defecto.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***





**Diferencia entre == y === en programación orientada a objetos:**



Es decir si yo y mi hermano nos compramos el mismo coche:

$micoche = Audi, 100cv, color rojo

$cocheHermano = Audi, 100cv, color rojo

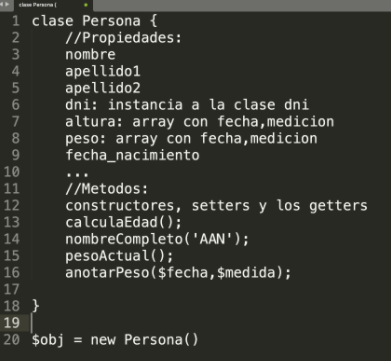
**$micoche == $cocheHermano (TRUE)** ya que son mismas propiedades y mismo valor, marca audi potencia 100 cv y color rojo

Pero **$micoche === $cocheHermano (FALSE)** porque pese a tener las mismas propiedades y valor son una instancia distinta de la clase coche es decir, **SON OBJETOS DIFERENTES.**

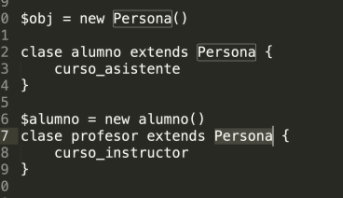
Para generar un nuevo objeto podemos usar constructores y para eliminarlo destructores…

**HERENCIA:**

Puedes hacer una clase alumno y otra clase profesor que tendrán propiedades comunes, como nombre, apellidos, edad, dni…



Ambos tienen propiedades de la clase persona.

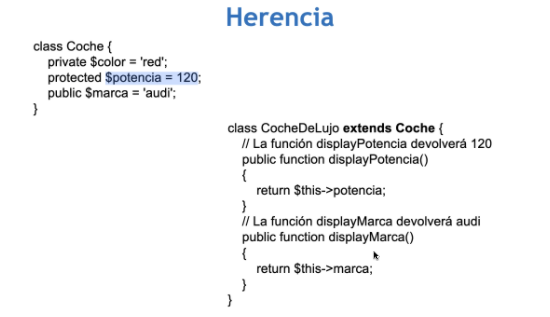
aquí por ejemplo dices que la class alumno extens persona es decir que coja las propiedades de persona y

aquí podríamos poner   
$alumno = new alumno(){

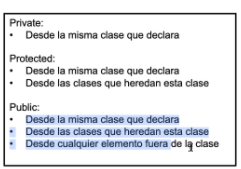
curso\_asistente

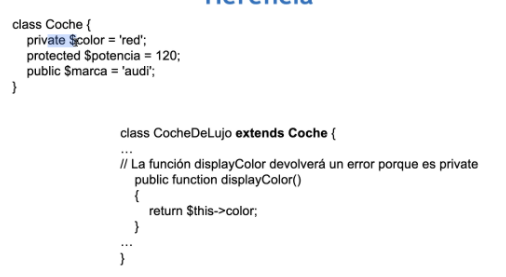
}

EJEMPLO COCHES: cada cocheDeLujo tendrá color rojo, 120 de potencia y será audi



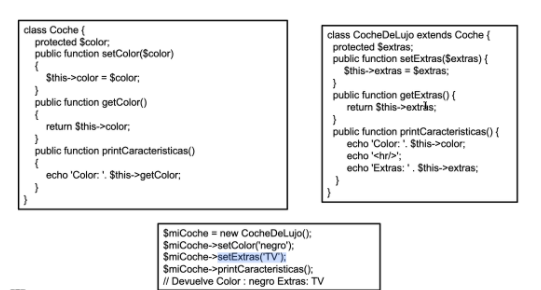
PROTECTED: Es como private pero con una herencia, solo podras acceder desde una herencia, pero desde una herencia no podras acceder a una privada:



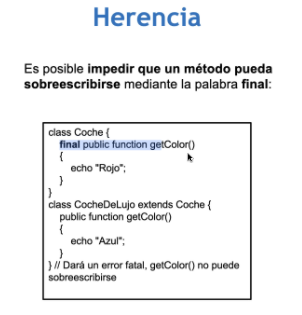


Aquí nos daría error, en todo caso deberíamos definir el color como protected.

EJEMPLO FUNCIONAL:



En este caso (abajo) $micoche es una clase coche de lujo y en la segunda línea hacemos setColor(‘negro’) cuando en la clase CocheDeLujo no hay un setter para el color. Como no hay, se ira al elemento padre class Coche que si tiene el setter y la declarará y se transferirá.

 Para que en una herencia no se sobreescriba, pondremos la palabra final y esa propiedad no se podrá sobreescribir y saldrá error.

**Modelo Vista Controlador (MVC)**

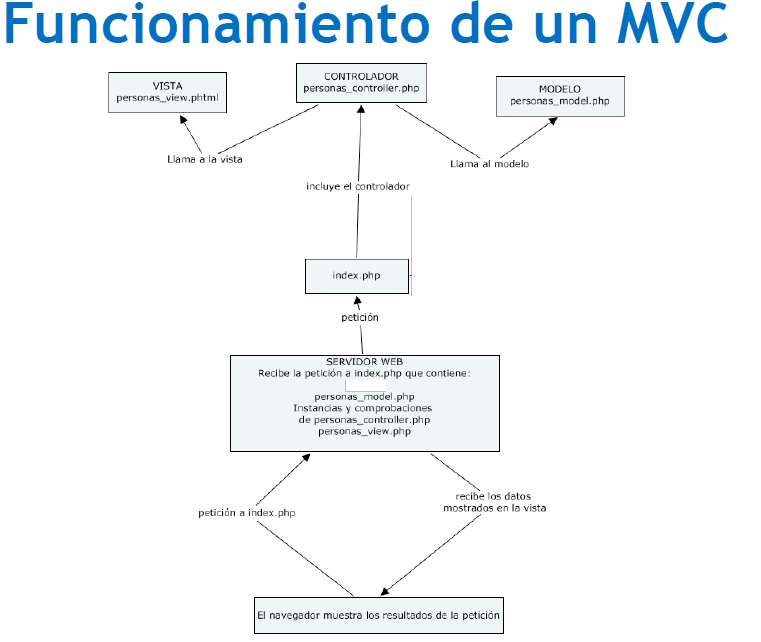
**Modelo**: gestiona todo lo relacionado con la info e iteración con los datos. Es decir contendrá las clases con sus propiedades y métodos que se comunican con los datos.

**Vista:** es la capa que mostrará la información formateada. Lo que es la parte visual: html,css… (front-end).

**Controlador**: Une la vista y el modelo. El usuario solicitará la información por medio de la vista (html) y esta hará la petición al controlador. Posteriormente este realizará la petición al modelo y lo mostrará.

**FUNCIONAMIENTO:**

1. El usuario realiza una petición.
2. El controlador captura la petición
3. El modelo hace la llamada correspondiente.
4. El modelo interactua con los datos
5. El controlador recibe la info y la envia a la vista
6. La vista muestra la información.



Clase Profesor 🡪 Profesores.dat

Propiedades : identificador + descripción del curso.

Para que salga cada curso en el select del formulario a la hora de elegir el curso.

**Hacer un gestionar Profesores** donde en podremos añadir al profesor a cada curso, y/o modificarlo.

Profesor hereda de persona.

Propiedad curso (o asignatura)