PROGRAMACIÓN ORIENTADA A OBJETOS

PHP sigue un paradigma de programación orientada a objetos (POO) basada en clases.

La **estructura de los objetos** se define **en las clases**. En ellas se escribe el código **que define el comportamiento de los objetos** y s**e indican los miembros que formarán parte de los objetos de dicha clase**. Entre los miembros de una clase puede haber:

- *Métodos*. Son los miembros de la clase que contienen el código de la misma. *Un método es como una función.* Puede recibir parámetros y devolver valores.

- *Atributos o propiedades*. *Almacenan información acerca del estado del objeto al que pertenecen* (y por tanto, *su valor puede ser distinto para cada uno de los objetos de la misma clase*).

A la **creación de un objeto basado en una clase** se le llama **instanciar una clase** y **al objeto obtenido** también se le conoce como **instancia de esa clase.**

Los pilares fundamentales de la POO son:

Herencia. Es el proceso de crear una clase a partir de otra, heredando su comportamiento y características y pudiendo redefinirlos.

Abstracción. Hace referencia a que cada clase oculta en su interior las peculiaridades de su implementación, y presenta al exterior una serie de métodos (interface) cuyo comportamiento está bien definido. Visto desde el exterior, cada objeto es un ente abstracto que realiza un trabajo.

Polimorfismo. Un mismo método puede tener comportamientos distintos en función del objeto con que se utilice.

Encapsulación. En la POO se juntan en un mismo lugar los datos y el código que los manipula.

MOTIVACIÓN

Las ventajas más importantes que aporta la programación orientada a objetos son:

Modularidad. La POO permite dividir los programas en partes o módulos más pequeños, que son independientes unos de otros pero pueden comunicarse entre ellos.

Extensibilidad. Si se desean añadir nuevas características a una aplicación, la POO facilita esta tarea de dos formas: añadiendo nuevos métodos al código, o creando nuevos objetos que extienden el comportamiento de los ya existentes.

Mantenimiento. Los programas desarrollados utilizando POO son más sencillos de mantener, debido a la modularidad antes comentada. También ayuda seguir ciertas convenciones al escribirlos, por ejemplo, escribir cada clase en un fichero propio. No debe haber dos clases en un mismo fichero, ni otro código aparte del propio de la clase.

Creación de clases

La **declaración de una clase en PHP** se hace utilizando la palabra **class**. **A continuación y entre llaves, deben figurar los miembros de la clase.**

Conviene **hacerlo de forma ordenada;**

1. Primero las **propiedades o atributos**→ Los atributos de una clase son similares a las variables de PHP.

   Es posible indicar un valor en la declaración de la clase. En este caso, *todos los objetos que se instancian a partir de esa clase, partirán con ese valor por defecto en el atributo.*

2. Después los **métodos**, cada uno con su código respectivo.

```php
<?php
class Producto {
private $codigo;
public $nombre;
public $PVP;
public function muestra() {
print "<p>" . $this->codigo . "</p>";
}
}
?>
```

NOTA; *Es preferible que cada clase figure en su propio fichero* (producto.php). Además, muchos programadores prefieren utilizar para las clases nombres que comienzan por letra mayúscula, para de esta forma, *distinguirlos de los objetos y otras variables*.

Cómo instanciar objetos

Una vez definida la clase, podemos usar la palabra **new** para *instanciar objetos*.

```php
$p = new Producto();
```

Y haber declarado la clase en el programa en el que la usemos con include/require (include_once/requiere_once)

```php
require_once('producto.php');
$p = new Producto();
```

Cómo acceder desde un objeto a los atributos o métodos de una clase

Utilizar el **operador flecha** (NOTA, *sólo se pone el símbolo $ delante del nombre del objeto*):

```
require_once('producto.php');
$p = new Producto();
$p->nombre = 'LG';
$p->muestra();
```

Si accedemos dentro de la clase a una propiedad o método de la misma clase, utilizaremos la referencia **$this**

```
private $codigo;
public function setCodigo($nuevo_codigo) {
if (noExisteCodigo($nuevo_codigo)) {
$this->codigo = $nuevo_codigo;
return true;
}
return false;
}
public function getCodigo() {
return $this->codigo;
}
```

Declarar atributos

Ya hemos visto que los atributos de una clase son similares a las variables de PHP. *Es posible indicar un valor en la declaración de la clase*. En este caso, *todos los objetos que se instancian a partir de esa clase, partirán con ese valor por defecto en el atributo.*

Además, cuando se declara un atributo, se debe indicar su nivel de acceso, siendo los principales niveles:

public. Los atributos declarados como public pueden utilizarse directamente por los objetos de la clase. ej, $nombre

private. Los atributos declarados como private s*ólo pueden ser accedidos y modificados por los métodos definidos en la clase,* no directamente por los objetos de la misma. ej, $codigo.

Uno de los motivos para crear atributos privados es que su valor forma parte de la información interna del objeto. Otro motivo es mantener cierto control sobre sus posibles valores

ej.Por ejemplo, no quieres que se pueda cambiar libremente el valor del código de un producto. O necesitas conocer cuál será el nuevo valor antes de asignarlo.En estos casos, se suelen definir esos atributos como privados y además se crean dentro de la clase métodos para permitirnos obtener y/o modificar los valores de esos atributos.

```php
<?php
class Persona {
    private string $nombre;
    public function setNombre(string $nom) {
        $this->nombre=$nom;
    }
    public function imprimir(){
        echo $this->nombre;
        echo '<br>';
    }
}
$bruno = new Persona(); // creamos un objeto
$bruno->setNombre("Bruno Díaz");
$bruno->imprimir();
?>
```

Aunque no es obligatorio, el nombre del método que nos permite o*btener el valor de un atributo* suele **empezar por get,** y el que nos permite *modificarlo* por **set**.

```php
<?php
class MayorMenor {
    private int $mayor;
    private int $menor;
    public function setMayor(int $may) {
        $this->mayor = $may;
    }
    public function setMenor(int $men) {
        $this->menor = $men;
    }
    public function getMayor() : int {
        return $this->mayor;
    }
    public function getMenor() : int {
        return $this->menor;
    }
}
```

En PHP5 se introdujeron los llamados métodos mágicos, entre ellos __set y __get. Si se declaran estos dos métodos en una clase, PHP los invoca automáticamente cuando desde un objeto se intenta usar un atributo no existente o no accesible. Por ejemplo, el código siguiente simula que la clase Producto tiene cualquier atributo que queramos usar.

Ej. el código siguiente simula que la clase Producto tiene cualquier atributo que queramos usar.

```php
class Producto {
    private $atributos = array();
    public function __get($atributo) {
    return $this->atributos[$atributo];
    }
    public function __set($atributo, $valor) {
    this->atributos[$atributo] = $valor;
    }
    }
```

```php
$p= new Producto();
    print_r($p);
    $p->noexisto = "Ahora si que existo";
    print_r($p);
```

Encapsulación

Las propiedades se definen privadas o protegidas (si queremos que las clases heredadas puedan acceder).
Para cada propiedad, **se añaden métodos públicos.**

Constantes

Además de métodos y propiedades, en una clase también se pueden definir constantes, utilizando la palabra **const.**

No confundas los atributos con las constantes:

Las constantes no pueden cambiar su valor

No usan el carácter $

Su valor va siempre entre comillas y está asociado a la clase, es decir, no existe una copia del mismo en cada objeto. Por tanto, *para acceder a las constantes de una clase*, se debe utilizar el **nombre de la clase** y el **operador ::,** *llamado operador de resolución de ámbito*

```php
class DB {
        const USUARIO = 'dwes';
        …
        }
        print DB::USUARIO;
```

NOTA; no es necesario que exista ningún objeto de una clase para poder acceder al valor de las constantes que defina. Además, sus nombres suelen escribirse en mayúsculas.

Clases estáticas

Se definen utilizando la palabra clave **static** y se referencian con **::**

Los atributos estáticos en PHP **son variables que pertenecen a la clase en sí misma** y **no a una instancia específica de esa clase.** Esto significa que *su valor es compartido por todos los objetos de esa clase,* y *puede ser accedido y modificado sin necesidad de crear una instancia.*

EJ de uso :

- Datos globales a la clase: Cuando necesitas almacenar información que sea común a todos los objetos de una clase, como un contador de instancias o una configuración global.
- Variables de caché: Para almacenar resultados de cálculoso datos que se utilizan con frecuencia.

```
class Producto {
        private static $num_productos = 0;
        public static function nuevoProducto() {
        self::$num_productos++;
        }
       }
```

Los atributos y métodos estáticos **no pueden ser llamados** desde un objeto de la clase **utilizando** el operador **->**. Ni **es posible llamarlos desde un objeto usando $this** dentro de un método estático.

Si el método o atributo es público, **deberá accederse utilizando el nombre de la clase y el operador de resolución de ámbito,** ej. Producto::nuevoProducto()

Si **es privado,** como el atributo $num_productos en el ejemplo anterior, sólo se podrá acceder a él desde los métodos de la propia clase, *utilizando* la palabra **self**. De la misma forma que $this hace referencia al objeto actual, self hace referencia a la clase actual. Es decir, Si desde un método queremos acceder a una propiedad estática de la misma clase, se utiliza la referencia **self**: ej self::$numProductos

```
    Producto::nuevoProducto();
    self::$num_productos ++;
```

Constructores

El constructor de una clase debe llamarse **__construct**. Se pueden utilizar, por ejemplo, para asignar valores a atributos.

Es un método especial dentro de una clase que se ejecuta automáticamente cada vez que se crea un nuevo objeto de esa clase. Su principal función es **inicializar las propiedades (atributos) del objeto con valores iniciales o realizar cualquier otra tarea necesaria para preparar el objeto para su uso.**

```php
class Producto {
        private static $num_productos = 0;
        private $codigo;
        public function __construct() {
        self::$num_productos++;
        }
        }
```

El constructor de una clase p**uede llamar a otros métodos o tener parámetros,** en cuyo caso deberán pasarse cuando se crea el objeto. Sin embargo, s***ólo puede haber un método constructor en cada clase.***

```php
class Producto {
        private static $num_productos = 0;
        private $codigo;
        public function __construct($codigo) {
        $this->$codigo = $codigo;
        self::$num_productos++;
        }
        }
```

```php
 $p = new Producto('MOTOROLA5G');
```

También es posible definir un método destructor, que debe llamarse **__destruct** y permite **definir acciones que se ejecutarán cuando se elimine el objeto**

```php
class Producto {
        private static $num_productos = 0;
        private $codigo;
        public function __construct($codigo) {
        $this->codigo = $codigo;
        self::$num_productos++;
        }
        public function __destruct() {
        self::$num_productos--;
        }
        }
```

```php
 $p = new Producto('MOTOROLA5G');

        unset($p);
```

Introspección

DEFINICIÓN - Introspección es la capacidad de un programa para examinar su propia estructura y comportamiento en tiempo de ejecución.

En el contexto de las clases, esto significa que podemos obtener información sobre:

Propiedades: Nombre, tipo y valor de las propiedades de una clase.

Métodos: Nombre, argumentos y valor de retorno de los métodos.

Clases: Jerarquía de clases, interfaces implementadas y rasgos utilizados.

Ej de uso, **generar documentación, crear frameworks y bibliotecas flexibles, depurar código, metaprogramación (**escribir código que genera o modifica otro código en tiempo de ejecución)

En PHP existen un conjunto de funciones ya definidas por el lenguaje que permiten obtener información sobre los objetos:

- **instanceof**: permite comprobar si un objeto es de una determinada clase
- **get_class:** devuelve el nombre de la clase
- **get_declared_class:** devuelve un array con los nombres de las clases definidas
- **class_alias**: crea un alias
- **class_exists / method_exists / property_exists**: true si la clase / método / propiedad está definida
- **get_class_methods / get_class_vars / get_object_vars**: Devuelve un array con los nombres de los métodos / propiedades de una clase / propiedades de un objeto que son accesibles desde dónde se hace la llamada (es decir, que sean públicos).

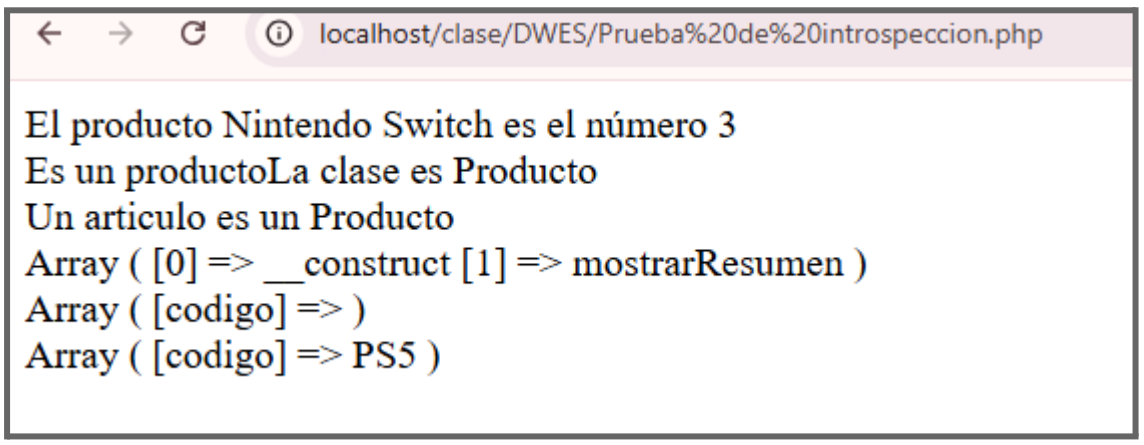
introspección.php

```php
<?php
include_once('claseProducto.php');
$p = new Producto("PS5");
if ($p instanceof Producto) {
    echo " Es un producto";
    echo "La clase es ".get_class($p)."<br>";

    class_alias("Producto", "Articulo");
    $c = new Articulo("Nintendo Switch");
    echo "Un articulo es un ".get_class($c)."<br>";

    print_r (get_class_methods("Producto"));
    print_r(nl2br("\n"));
    print_r(get_class_vars("Producto"));
```

```
    print_r(nl2br("\n"));
    print_r(get_object_vars($p));

    if (method_exists($p, "mostrarResumen")) {
        $p->mostrarResumen();
    }
}
?>
```

claseProducto.php

```
<?php
class Producto {
    const IVA = 0.23;
    private static $numProductos = 0;
    public $codigo;

    public function __construct(string $cod) {
        self::$numProductos++;
        $this->codigo = $cod;
    }
    public function mostrarResumen() : string {
                return "El producto ".$this->codigo." es el número
".self::$numProductos;
    }
}
$prod1 = new Producto("PS5");
$prod2 = new Producto("XBOX Series X");
$prod3 = new Producto("Nintendo Switch");
echo $prod3->mostrarResumen();
print_r(nl2br("\n"));

?>
```

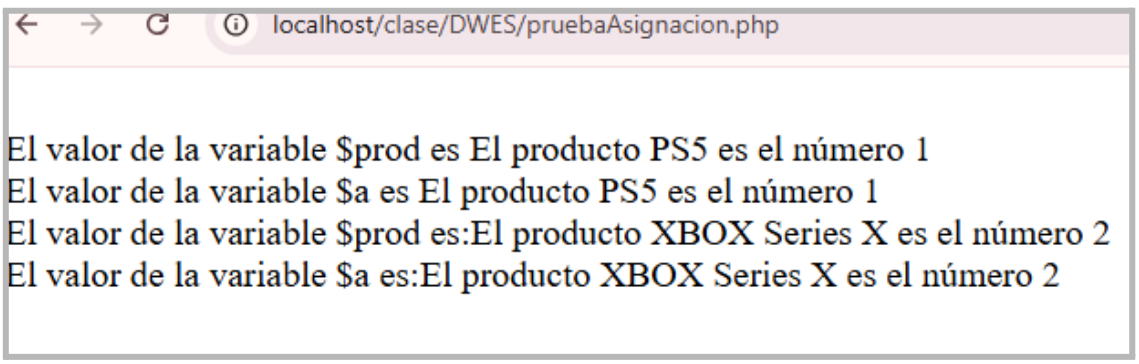


Clonado

¿Qué sucede cuando en PHP se ejecuta un código como el siguiente?

pruebaAsignacion.php

```php
<?php
class Producto {
    const IVA = 0.23;
    private static $numProductos = 0;
    public $codigo;


    public function __construct(string $cod) {
        self::$numProductos++;
        $this->codigo = $cod;
    }
    public function mostrarResumen() : string {
                return "El producto ".$this->codigo." es el número
".self::$numProductos;
    }
}
$prod = new Producto("PS5");
$a=$prod;
print_r(nl2br(" \nEl valor de la variable \$prod es "));
echo $prod->mostrarResumen();
print_r(nl2br(" \nEl valor de la variable \$a es "));
echo $prod->mostrarResumen();
$prod = new Producto("XBOX Series X");
print_r(nl2br(" \nEl valor de la variable \$prod es:"));
echo $prod->mostrarResumen();
print_r(nl2br(" \nEl valor de la variable \$a es:"));
echo $prod->mostrarResumen();
?>
```



En PHP el código anterior simplemente crea un nuevo identificador del mismo objeto. Es decir, cuando se utilice uno cualquiera de los identificadores para cambiar el valor de algún atributo, este cambio se vería también reflejado al acceder utilizando el otro identificador. **Recuerda que en realidad todos se refieren a la única copia que se almacena del mismo**.

Al asignar dos objetos no se copian, se crea una nueva referencia. Si queremos una copia, hay que clonarlo mediante el método **clone(object) : object**

Para el ej

```php
$p = new Producto();
$p->nombre = 'Samsung Galaxy S';
$a = clone($p);
```

Si queremos modificar el clonado por defecto, existe una forma sencilla de personalizar la copia para cada clase particular (por ejemplo, copiar todos los atributos menos alguno), hay que definir el método mágico **__clone()** que se llamará después de copiar todas las propiedades (después de copiar todos los atributos en el nuevo objeto).

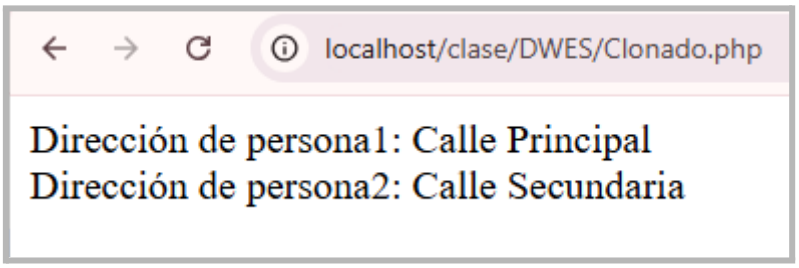
clonado.php

```php
<?php
class Persona {
    public $nombre;
    public $edad;
    public $direccion; // Un objeto de la clase Direccion
    public function __construct($nombre, $edad, Direccion $direccion) {
        $this->nombre = $nombre;
        $this->edad = $edad;
        $this->direccion = $direccion;
    }
    public function __clone() {
        // Clonamos la dirección para evitar referencias compartidas
        $this->direccion = clone $this->direccion;
    }
}
class Direccion {
    public $calle;
    public $numero;
    public $ciudad;
}
// Creamos una dirección
$direccion1 = new Direccion();
$direccion1->calle = "Calle Principal";
$direccion1->numero = 123;
$direccion1->ciudad = "Madrid";
// Creamos una persona
```

```php
$persona1 = new Persona("Juan", 30, $direccion1);
// Clonamos la persona
$persona2 = clone $persona1;
// Modificamos la dirección de la persona clonada
$persona2->direccion->calle = "Calle Secundaria";
// Imprimimos las direcciones
echo "Dirección de persona1: " . $persona1->direccion->calle . "<br>";
echo "Dirección de persona2: " . $persona2->direccion->calle . "<br>";
?>
```

NOTA,Desde PHP5, se puede indicar en las funciones y métodos de qué clase deben ser los objetos que se pasen como parámetros. Para ello, debes especificar el tipo antes del parámetro. En el ej.

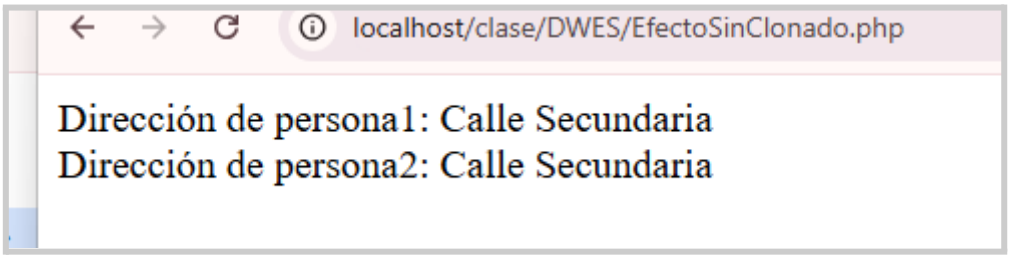
```php
    public function __construct($nombre, $edad, Direccion $direccion)
```



efectoSinClonado.php

```php
<?php
class Persona {
    public $nombre;
    public $edad;
    public $direccion; // Un objeto de la clase Direccion
    public function __construct($nombre, $edad, Direccion $direccion) {
        $this->nombre = $nombre;
        $this->edad = $edad;
        $this->direccion = $direccion;
    }
    /*public function __clone() {
        // Clonamos la dirección para evitar referencias compartidas
        $this->direccion = clone $this->direccion;
    }*/
}
class Direccion {
    public $calle;
    public $numero;
    public $ciudad;
}
// Creamos una dirección
```

```php
$direccion1 = new Direccion();
$direccion1->calle = "Calle Principal";
$direccion1->numero = 123;
$direccion1->ciudad = "Madrid";
// Creamos una persona
$persona1 = new Persona("Juan", 30, $direccion1);
// Clonamos la persona
$persona2 = clone $persona1;
// Modificamos la dirección de la persona clonada
$persona2->direccion->calle = "Calle Secundaria";
// Imprimimos las direcciones
echo "Dirección de persona1: " . $persona1->direccion->calle . "<br>";
echo "Dirección de persona2: " . $persona2->direccion->calle . "<br>";
?>
```



IMPORTANTE, En PHP puedes crear referencias a variables (como números enteros o cadenas de texto), utilizando el operador &:

```php
$p = new Producto();
$p->nombre = 'Samsung Galaxy S';
$a = clone($p);
// El resultado de comparar $a === $p da falso
// pues $a y $p no hacen referencia al mismo objeto
$a = & $p;
// Ahora el resultado de comparar $a === $p da verdadero
// pues $a y $p son referencias al mismo objeto.
```

Herencia

La herencia es un mecanismo de la POO que nos permite definir nuevas clases en base a otra ya existente. L**as nuevas clases** que heredan también se conocen con el nombre de **subclases**. La clase **de la que heredan** se llama c**lase base o superclase.**

ej. de herencia → Una tienda web que vende electrónica de distintos tipos.

claseProducto.php ->Para manejar la entrada de productos genéricos (entrada en el almacen) creamos una clase llamada Producto, con algunos atributos y un método que genera una salida personalizada en formato HTML del código.

```php
<?php
class Producto
{
public $codigo;
public $nombre;
public $nombre_corto;
public $PVP;
public function muestra()
{
echo "<p>" . $this->codigo . "</p>";
}
}
?>
```

claseExtProducto.php→ Para almacenar características de los productos (ej, pulgadas de monitores, capacidad y procesado de ordenadores, tecnología …). Esta clase será una **clase heredada de** la clase **Producto.**

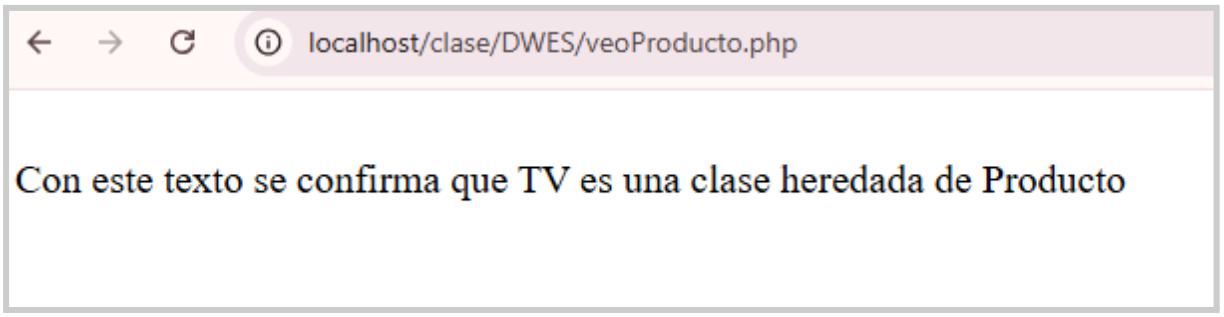
```php
<?php
class TV extends Producto
{
public $pulgadas;
public $tecnologia;
}
?>
```

Para definir una clase que herede de otra, utilizar la palabra **extends *indicando la superclase (*** `class TV extends Producto)`. *Los nuevos objetos que se instancian a partir de la subclase* **son también objetos de la clase base**; se puede *comprobar utilizando el operador instanceof (ver ejemplo a continuación)*.

veoProducto.php

```php
<?php
include_once ('claseProductoElectronica.php');
include_once ('claseExtProducto.php');
$t = new TV();
if ($t instanceof Producto)
{
// Este código se ejecuta pues la condición es cierta
echo "<br>Con este texto se confirma que TV es una clase heredada de
Producto";
}
?>
```

RESULTADO

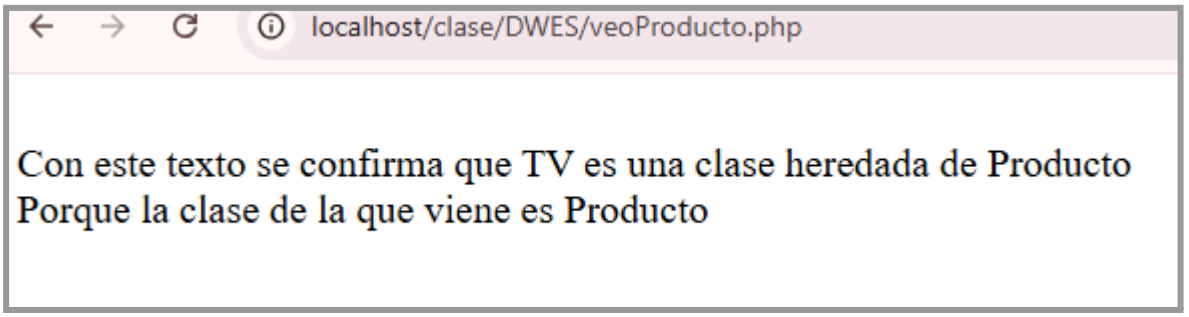


De las funciones definidas en PHP que permiten obtener información sobre los objetos y que están relacionadas con la herencia de clases destacamos:

get_parent_class ;Devuelve el nombre de la clase padre del objeto o la clase que se indica.

is_subclass_of ; Comprueba si el objeto tiene esta clase como uno de sus padres.

```php
<?php
include_once ('claseProductoElectronica.php');
include_once ('claseExtProducto.php');
$t = new TV();
if ($t instanceof Producto)
{
// Este código se ejecuta pues la condición es cierta
echo "<br>Con este texto se confirma que TV es una clase heredada de Producto";
$claseOriginal=get_parent_class($t);
if (is_subclass_of($t, 'Producto')) {
echo "<br>Porque efectivamente la clase de la que viene es ". $claseOriginal."<br>";
}
}
?>
```



IMPORTANTE, **La nueva clase hereda todos los atributos y métodos públicos** de la clase base, **pero no los privados**.

> *Para crear en la clase base un método no visible al exterior* (como los privados) *que se herede a las subclases,* debes *utilizar la palabra* **protected** en lugar de private.

Constructor en hijos

En los hijos no se crea ningún constructor de manera automática. Por lo que si no lo hay, *se invoca automáticamente al del padre.* En cambio, **si lo definimos en el hijo**, hemos de **invocar al del padre de manera explícita**.

EJ,claseProductoElectronica.php

```php
<?php
class Producto
{
public function __construct(public int $codigo, public $nombre=null,public $nombre_corto=null, public $PVP=null) {
    $this->codigo = $codigo;
}
final public function muestra()
{
echo "<br> El código del producto desde la superclase es ". $this->codigo . "<br>";
}
}
?>
```

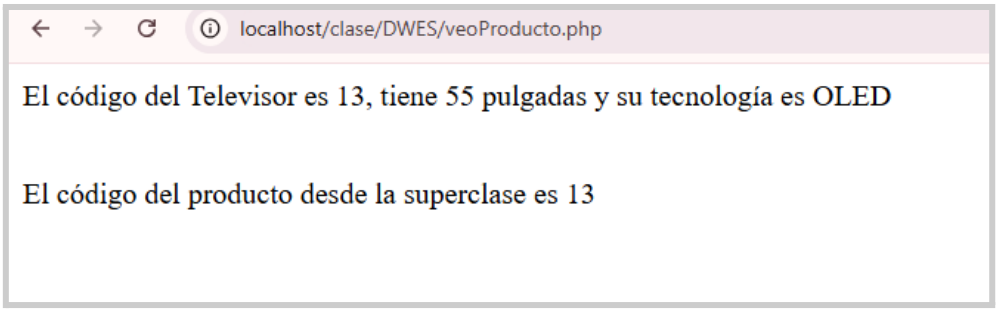
claseExtProducto.php

```php
<?php
include_once ('claseProductoElectronica.php');
class TV extends Producto
{
public function __construct(int $codigo, private int $pulgadas, private string $tecnologia){
    parent::__construct($codigo);
}
public function muestraHerencia()
{
echo "<p>El código del Televisor es {$this->codigo}, tiene {$this->pulgadas} pulgadas y su tecnología es {$this->tecnologia} </p>";
}
}
```

```php
?>
```

veoProducto.php

```php
<?php
include_once ('claseProductoElectronica.php');
include_once ('claseExtProducto.php');

//constructor en la clase heredada
$t = new TV(13,55,"OLED");
$p = new Producto(13);
$t->muestraHerencia();
$p->muestra();

?>
```



Sobrescribir métodos

Para **redefinir el comportamiento de los métodos existentes en la clase base,** crear **en la subclase** un n**uevo método con el mismo nombre**.

claseProductoElectronica.php

```php
<?php
class Producto
{
public $codigo;
public $nombre;
public $nombre_corto;
public $PVP;
public function __construct($codigo=null) {
    $this->codigo = $codigo;
}
public function muestra()
{
echo "<br> El código del producto es ". $this->codigo . "<br>";
}
}
```

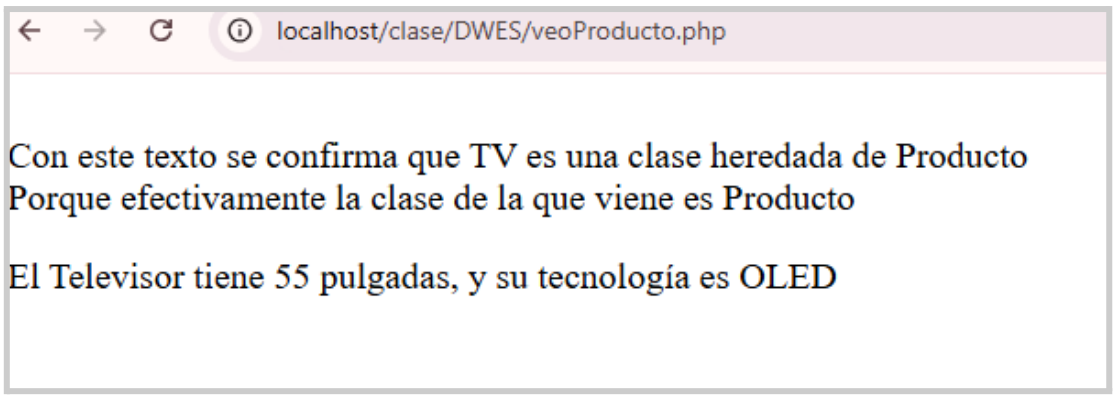
```
?>
```

claseExtProducto.php

```
<?php
include_once ('claseProductoElectronica.php');
class TV extends Producto
{
public $pulgadas;
public $tecnologia;
public function muestra()
{
echo "<p>El Televisor tiene {$this->pulgadas} pulgadas, y su tecnología
es {$this->tecnologia} </p>";
}
}
?>
```

veoProducto.php

```
<?php
include_once ('claseProductoElectronica.php');
include_once ('claseExtProducto.php');
$t = new TV();
$p = new Producto(13);

$t->pulgadas = 55;
$t->tecnologia = "OLED";
if ($t instanceof Producto)
{
// Este código se ejecuta pues la condición es cierta
echo "<br>Con este texto se confirma que TV es una clase heredada de
Producto";
$claseOriginal=get_parent_class($t);
if (is_subclass_of($t, 'Producto')) {
echo "<br>Porque efectivamente la clase de la que viene es ".
$claseOriginal."<br>";
$t->muestra();
}
}
?>
```



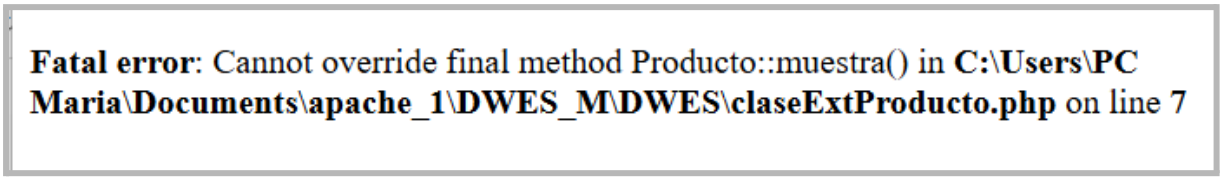
Utilizando la palabra **final** se evita que las clases heredadas puedan redefinir el comportamiento de los métodos existentes en la superclase. La palabra **final** puede usarse en la definición del método e incluso en la definición de la clase **final class Producto { ... }**

EJ, redefiniendo el método muestra() de claseProductoElectronica.php con

```
final public function muestra()
```

```php
<?php
class Producto
{
public $codigo;
public $nombre;
public $nombre_corto;
public $PVP;
public function __construct($codigo=null) {
    $this->codigo = $codigo;
}
final public function muestra()
{
echo "<br> El código del producto desde la superclase es ".
$this->codigo . "<br>";
}
}
?>
```

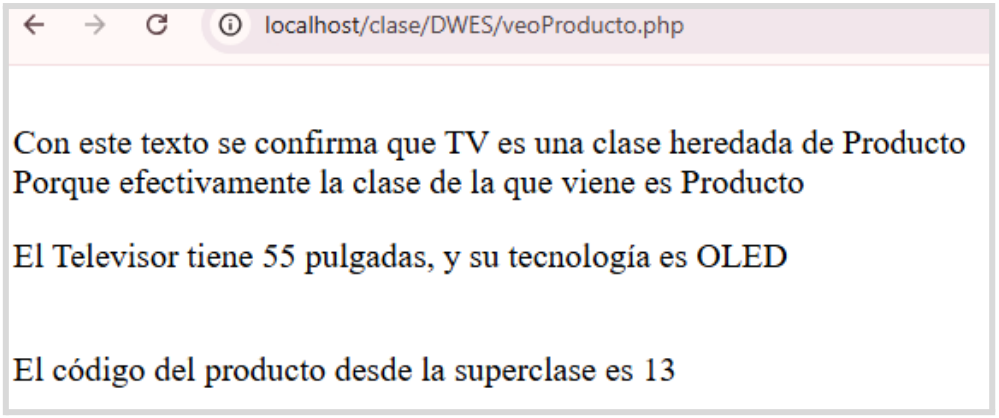
El resultado al ejecutar el código es un Fatal error por redefinición del método muestra en la clase heredada



Tenemos que corregir el código en claseExtProducto.php y veoProducto.php como sigue

```php
<?php
include_once ('claseProductoElectronica.php');
class TV extends Producto
{
public $pulgadas;
public $tecnologia;
public function muestraHerencia()
{
echo "<p>El Televisor tiene {$this->pulgadas} pulgadas, y su tecnología
es {$this->tecnologia} </p>";
}
}
?>
```

```php
<?php
include_once ('claseProductoElectronica.php');
include_once ('claseExtProducto.php');
$t = new TV();
$p = new Producto(13);

$t->pulgadas = 55;
$t->tecnologia = "OLED";
if ($t instanceof Producto)
{
// Este código se ejecuta pues la condición es cierta
echo "<br>Con este texto se confirma que TV es una clase heredada de
Producto";
$claseOriginal=get_parent_class($t);
if (is_subclass_of($t, 'Producto')) {
echo "<br>Porque efectivamente la clase de la que viene es ".
$claseOriginal."<br>";
$t->muestraHerencia();
$p->muestra();
}
}
?>
```



Clases abstractas¶

Son clases opuestas a **final.**

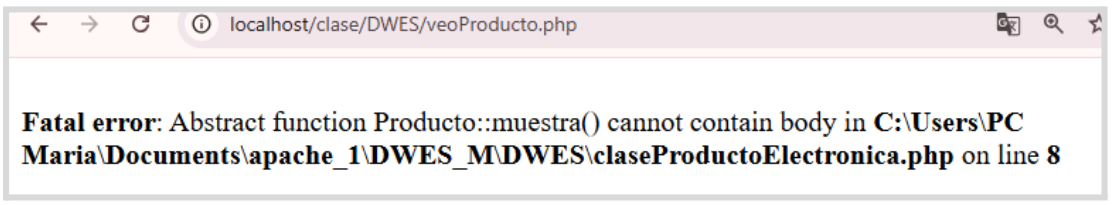
Las clases abstractas obligan a heredar de una clase, ya que no se permite su instanciación. Se define mediante **abstract class NombreClase {.**

Una clase abstracta **puede contener** *propiedades y métodos no-abstractos. y/o métodos abstractos.*

Ej. si modificamos nuestro código de claseProductoElectronica.php

```php
<?php
abstract class Producto
{
public function __construct(public int $codigo, public $nombre=null,public $nombre_corto=null, public $PVP=null) {
    $this->codigo = $codigo;
}
// Método abstracto
abstract public function muestra();
}
?>
```

El resultado en el cliente será



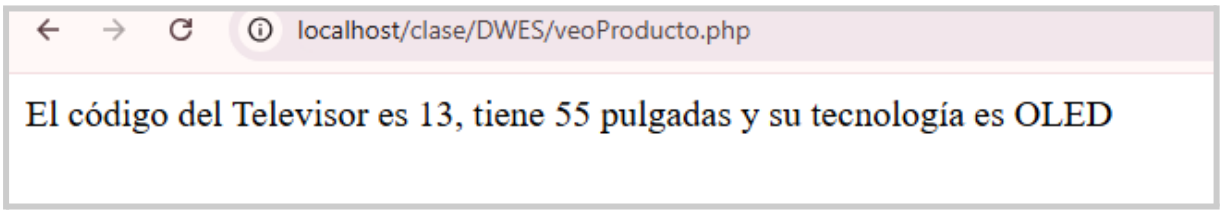
Para corregir este error debemos modificar el código de claseExtProducto.php como sigue

```php
<?php
include_once ('claseProductoElectronica.php');
class TV extends Producto
{
public function __construct(int $codigo, private int $pulgadas, private
string $tecnologia){
    parent::__construct($codigo);
}
public function muestra()//obligado a implementarlo al ser abstracta la
clase Producto
{
echo "<p>El código del Televisor es {$this->codigo}, tiene
{$this->pulgadas} pulgadas y su tecnología es {$this->tecnologia}
</p>";
}
}
?>
```

veoProducto.php

```php
<?php
include_once ('claseProductoElectronica.php');
include_once ('claseExtProducto.php');
//constructor en la clase heredada
$t = new TV(13,55,"OLED");
$t->muestra();
?>
```

Siendo el resultado:



Introducimos arrays en el constructor

Para facilitar la creación de nuevos objetos, crearemos un constructor al que se le pasará un array con los valores de los atributos del nuevo producto.

ej.claseProductoElectronica.php

```php
<?php
class Producto
{
    public $codigo;
    public $nombre;
    public $nombre_corto;
    public $PVP;

public function __construct($row) {
    $this->codigo = $row['codg'];
    $this->nombre = $row['nomb'];
    $this->nombre_corto = $row['nom_cort'];
    $this->PVP = $row['PVP'];
}

public function muestra()
{
echo "<br> El código del producto desde la superclase es ".
$this->codigo . "<br>";
echo "<br> El nombre del producto desde la superclase es ".
$this->nombre . "<br>";
echo "<br> El nombre corto del producto desde la superclase es ".
$this->nombre_corto . "<br>";
echo "<br> El PVP del producto desde la superclase es ". $this->PVP.
"<br>";
}
}
?>
```

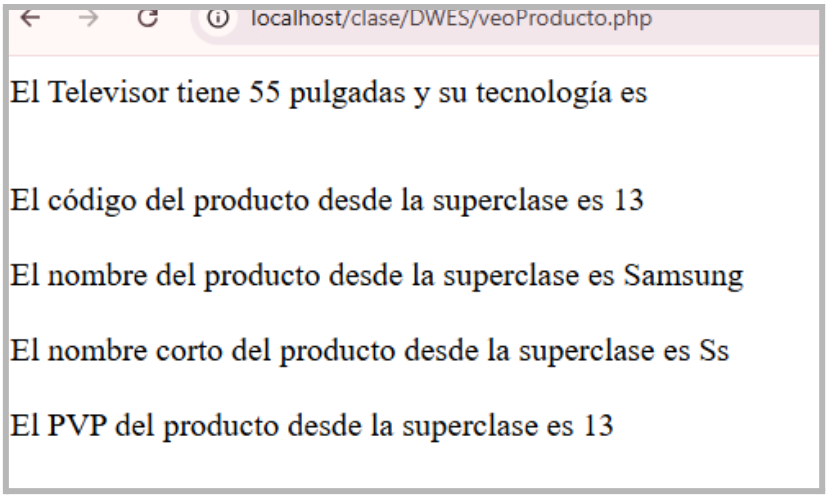
claseExtProducto.php

```php
<?php
include_once ('claseProductoElectronica.php');
class TV extends Producto
{
public $pulgadas;
public $tecnologia;
public function __construct($row){
    parent::__construct($row);
    $this->pulgadas = $row['pulgadas'];
}
public function muestra()
{
```

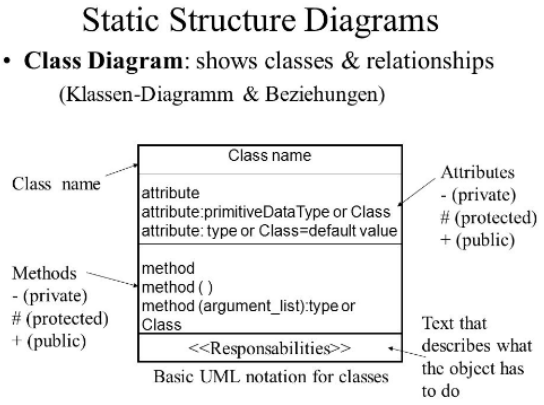
```
echo "<p>El Televisor tiene {$this->pulgadas} pulgadas y su tecnología
es {$this->tecnologia} </p>";
}
}
?>
```

veoProducto.php

```
<?php
include_once ('claseProductoElectronica.php');
include_once ('claseExtProducto.php');
$datosProducto = [
    'codg' => 13,
    'nomb' => 'Samsung',
    'nom_cort' => 'Ss',
    'PVP' => 13,
    'pulgadas'=>55
];
$t = new TV($datosProducto);
$p = new Producto($datosProducto);
$t->muestra();
$p->muestra();
?>
```



NOTA; Cuando un proyecto crece, es normal **modelar las clases mediante UML.** La clases se representan mediante un cuadrado, separando el nombre, de las propiedades y los métodos:



Elementos básicos de un diagrama de clases:

- **Clases:** Representadas como rectángulos divididos en tres partes: nombre de la clase, atributos y métodos.
- **Atributos:** Las propiedades de los objetos de una clase. Se representan como nombre_atributo: tipo_de_dato.
- **Métodos:** Las operaciones que los objetos pueden realizar. Se representan como nombre_método(parámetros): tipo_de_retorno.
- **Relaciones:**
    - **Asociación:** Representa una relación entre dos clases. Se muestra como una línea que conecta las clases.
    - **Agregación:** Representa una relación "tiene un". Se muestra como una línea con un rombo en el extremo de la clase que contiene.
    - **Composición:** Representa una relación "está compuesto por". Se muestra como una línea sólida con un rombo relleno en el extremo de la clase que contiene.
    - **Herencia:** Representa una relación "es un". Se muestra como una flecha hueca que apunta de la subclase a la superclase.

Ej. Gimnasio (Volveremos sobre esta idea posteriormente)

Class Diagram of Gym Management System :

Class Diagram Image: