Desarrollo Web en entorno servidor
Unidad 3:
Aplicaciones Web en PHP
Programación orientada a objetos.

Tabla de contenido

1. Programación orientada a objetos

Esta primera parte del tema reproduce el capítulo dedicado a la programación orientada a objetos del libro APRENDER A DESARROLLAR UN SITIO WEB CON PHP Y MYSQL (2ª ED.) de Olivier Rollet, publicado en 2015por el Editorial ENI.

La Programación Orientada a Objetos, en adelante POO, es un paradigma de programación según el cual los sistemas a desarrollar se modelan creando clases. Las clases son las definiciones de un conjunto de datos (atributos) y funciones (métodos) y nos permiten crear objetos.

Los objetos son ejemplares de una clase determinada y como tal, disponen de los datos (atributos) y funciones (métodos) definidos.

Pongamos un ejemplo, pensemos en un animal cualquiera. De todos los animales podemos distinguir ciertos atributos como su color o su peso. Por otra parte, los animales pueden hacer ciertas funciones como moverse o comer.

Por tanto, podemos definir la clase “Animal”, la cual tendrá por atributos el color y el peso, y como métodos moverse y comer.

Definida una clase, podemos pensar en los objetos de dicha clase.

Dos objetos de la clase “Animal” son, por ejemplo, el objeto “perro” y el objeto “gato”.

Cuando se crean ejemplares de animales en la clase Animal, decimos que se crea una instancia de esta clase. Crear una instancia de una clase significa que se crea un objeto (“perro” o “gato”) de un tipo determinado (“Animal”) con ciertos atributos (“color”, “peso”) y ciertos métodos (“comer”, “moverse”)

CLASE	ANIMAL
ATRIBUTOS	Color Peso
MÉTODOS	Comer() Moverse()

Tabla 1 - Clase animal

1.1. Creación de una clase en PHP

El código en PHP para crear una clase es el siguiente.

```
<?php
class Animal // palabra clave class seguida del nombre de la clase.
{
// Declaración de atributos y métodos.
}
```

Es recomendable crear una clase por cada archivo PHP que tenga el mismo nombre que la clase.

1.2. La encapsulación

Todos los atributos en POO deben estar ocultos para otras personas que utilizan las clases que definimos. Si estoy trabajando en equipo y creo la clase Animal, los otros programadores no deben poder cambiar directamente los atributos de mi clase.

De esta forma, los atributos color y peso se ocultan en otras clases; se declaran privadas. La clase Animal tiene métodos para leer o escribir en estos atributos. Este es el principio de encapsulación, el cual permite proteger el código cuando se trabaja en equipo.

La clase Animal, que tiene como propiedades el color y el peso, a de disponer de un método para modificar su color, un método para leer el color, un método para modificar su peso, un método para leer su peso, así como otros métodos tales como comer o moverse.

1.3. Visibilidad de los atributos y de los métodos

Hay tres tipos de palabra clave para definir la visibilidad de un atributo o de un método:

- private: solo el código de su clase puede ver y acceder a este atributo o método.
- public: todas las demás clases pueden acceder a este atributo o método.
- protected: solo el código de su clase y de sus subclases pueden acceder a este atributo o método.

Vamos a crear la clase en PHP con sus atributos:

```php
<?php
class Animal // palabra clave seguida del nombre de la clase.
{
// Declaración de atributos.
private $color;
private $peso;
}
```

Podemos definir los valores por defecto de sus atributos:

```php
<?php
class Animal // palabra clave seguida del nombre de la clase.
{
// Declaración de atributos.
private $color = "gris";
private $peso = 10;
}
```

Añadimos los métodos a la clase, las normas de visibilidad son las mismas que en los atributos:

```php
<?php
class Animal // palabra clave seguida del nombre de la clase.
{
    // Declaración de atributos y métodos.
    private $color = "gris";
    private $peso = 10;
    public function comer()
    {
```

```
        //Método que puede acceder a las propiedades color y peso
    }
    public function moverse()
    {
        //Método que puede acceder a las propiedades color y peso
    }
}
```

1.4. Añadir un método en la clase

Añadir el código de un método a la clase significa aplicar la clase. Para acceder a los atributos de la clase, debemos utilizar la pseudovariable $this, que representa el objeto sobre el que va a escribir.

Para acceder al atributo o al método del objeto, utilizamos el operador ->.

Por ejemplo, vamos a añadir el código del método añadir_un_kilo() en la clase Animal:

```
<?php
class Animal // palabra clave seguida del nombre de la clase.
{
    // Declaración de atributos y métodos.
    private $color = "gris";
    private $peso = 10;
    public function comer()
    {
        //Método que puede acceder a las propiedades color y peso
    }
    public function moverse()
    {
        //Método que puede acceder a las propiedades color y peso
    }
    public function añadir_un_kilo()
    {
        $this->peso = $this->peso + 1;
    }
}
```

Cuando llamamos al método añadir_un_kilo(), añadiremos 1 al peso actual y por lo tanto el peso final será 11.

Las propiedades se declaran con el símbolo `$`, pero se llaman con `$this` sin este símbolo.

1.5. Utilización de la clase

Como primer paso, tenemos que crear un archivo que contenga el código PHP de nuestra clase Animal.class.php.

Para utilizar la clase Animal, debemos incluirla en la página donde la queramos llamar.

Vamos a crear una página uso.php y en ella escribimos el siguiente código:

```
<?php
//carga de la clase
```

```php
include('Animal.class.php');
//instanciar la clase Animal
$perro = new Animal();
?>
```

Vemos como hemos cargado la clase y la hemos instanciado, es decir, hemos creado un objeto que tiene como modelo la clase Animal:

La variable `$perro` es una instancia de la clase Animal, con los atributos propios de color, peso, y como métodos comer, moverse, añadir_un_kilo.

1.6. Actualizar y leer los atributos de la instancia

El principio de encapsulación requiere que todos los atributos sean privados. Por lo tanto, debemos crear métodos públicos que permitan leer o escribir en sus atributos desde otra página PHP.

Estos métodos se denominan **accesos** y, generalmente, sus nombres van precedidos del prefijo **get** para leer el valor del atributo y **set** para escribir el valor del atributo.

La clase Animal con los accesos es:

```php
<?php
class Animal // palabra clave seguida del nombre de la clase.
{
    // Declaración de atributos
    private $color = "gris";
    private $peso = 10;
    //accesos
    public function getColor()
    {
        return $this->color; //devuelve el color
    }
    public function setColor($color)
    {
        $this->color = $color; //escrito en el atributo color
    }
    public function getPeso()
    {
        return $this->peso; //devuelve el peso
    }
    public function setPeso($peso)
    {
        $this->peso = $peso; //escrito en el atributo peso
    }
    //métodos
    public function comer()
    {
        //Método que puede acceder a las propiedades color y peso
    }
    public function moverse()
    {
```

```
        //Método que puede acceder a las propiedades color y peso
    }
    public function añadir_un_kilo()
    {
        $this->peso = $this->peso + 1;
    }
}
```

Los accesos son públicos y por lo tanto permiten leer o escribir en los atributos desde cualquier otra clase o página PHP.

Ejemplo con la página uso.php:

```
<?php
//carga de la clase
include('Animal.class.php');
//instanciar la clase Animal
$perro = new Animal();
//leer el peso
echo "El peso del perro es:".$perro->getPeso()." kg<br>";
//añadir un kilo al perro
$perro->añadir_un_kilo();
//leer el peso
echo "El peso del perro es:".$perro->getPeso()." kg<br>";
//actualizar el peso del perro
$perro->setPeso(15);
//leer el peso
echo "El peso del perro es:".$perro->getPeso()." kg<br>";
?>
```

Da como resultado:

El peso del perro es:10 kg
El peso del perro es:11 kg
El peso del perro es:15 kg

En efecto, el peso del perro se inicializa a 10. A continuación el método añadir_un_kilo() añade 1; por lo tanto, su peso se convierte en 11. Para terminar, el método setPeso(15) ajusta el peso a 15.

Podemos crear tantas instancias de una clase como queramos.

Por ejemplo, para crear un gato blanco de 5 kg y un perro negro de 18 kg:

```
<?php
//carga de la clase
include('Animal.class.php');
//instanciar la clase Animal
$perro = new Animal();
//actualizar el peso del perro
$perro->setPeso(18);
//leer el peso
```

```
echo "El peso del perro es:".$perro->getPeso()." kg<br>";
//actualizar el color del perro
$perro->setColor("negro");
//leer el color
echo "El color del perro es:".$perro->getColor()."<br>";
//instanciar la clase Animal
$gato = new Animal();
//actualizar el peso del gato
$gato->setPeso(5);
//leer el peso
echo "El peso del gato es:".$gato->getPeso()." kg<br>";
//actualizar el color del gato
$gato->setColor("blanco");
//leer el color
echo "El color del gato es:".$gato->getColor()."<br>";
?>
```

Da como resultado:

El peso del perro es:18 kg
El color del perro es:negro
El peso del gato es:5 kg
El color del gato es:blanco

1.7. Paso como argumento de tipo objeto

Los métodos son como las funciones, pueden tomar argumentos de tipos diferentes (Integer, String…) e incluso de tipo Objeto.

`$gato` y `$perro` son objetos de tipo Animal. Pueden pasar como argumento un método, siempre y cuando acepte este tipo de objeto.

Para probar este ejemplo, cambiamos el método `comer()` de la clase Animal por `comer_animal(Animal $animal_comido)`. El método toma ahora como argumento un objeto de tipo Animal.

La página Animal.class.php queda como sigue:

```
<?php
class Animal
{
    // Declaración de atributos
    private $color = "gris";
    private $peso = 10;
    //accesos
    public function getColor()
    {
        return $this->color; //devuelve el color
    }
    public function setColor($color)
    {
        $this->color = $color; //escrito en el atributo color
```

```
    }
    public function getPeso()
    {
        return $this->peso; //devuelve el peso
    }
    public function setPeso($peso)
    {
        $this->peso = $peso; //escrito en el atributo peso
    }
    //Métodos
    public function comer_animal(Animal $animal_comido)
    {
        //el animal que come aumenta su peso tanto como el del animal comido
        $this->peso = $this->peso + $animal_comido->peso;
        //el peso del animal comido y su color se restablecen a 0
        $animal_comido->peso = 0;
        $animal_comido->color = "";
    }
    public function moverse()
    {
        //método que pueda acceder a las propiedades color y peso
    }
    public function añadir_un_kilo()
    {
        $this->peso = $this->peso + 1;
    }
}
```

Para probar este método, la página uso.php queda como sigue:

```
<?php
//carga de la clase
include('Animal.class.php');
//instanciar la clase Animal
$gato = new Animal();
//actualizar el peso del gato
$gato->setPeso(8);
//leer el peso
echo "El peso del gato es:".$gato->getPeso()." kg<br>";
//actualizar el color del gato
$gato->setColor("negro");
//leer el color
echo "El color del gato es:".$gato->getColor()."<br>";
//instanciar la clase Animal
$pez = new Animal();
//actualizar el peso del pez
$pez->setPeso(1);
//leer el peso
echo "El peso del pez es:".$pez->getPeso()." kg<br>";
//actualizar el color del pez
```

```
$pez->setColor("blanco");
//leer el color
echo "El color del pez es:".$pez->getColor()."<br><br>";
//el gato come al pez
$gato->comer_animal($pez);
//leer el peso
echo "El nuevo peso del gato es:".$gato->getPeso()." kg<br>";
//leer el peso
echo "El peso del pez es:".$pez->getPeso()." kg<br>";
//leer el color
echo "El color del pez es:".$pez->getColor()."<br><br>";
?>
```

Da como resultado:
El peso del gato es:8 kg
El color del gato es:negro
El peso del pez es:1 kg
El color del pez es:blanco
El nuevo peso del gato es:9 kg
El peso del pez es:0 kg
El color del pez es:

El objeto `$gato` llama al método `comer_animal ($pez)` y pasa como argumento el objeto de tipo Animal `$pez`. Es decir, el objeto `$pez` con sus atributos y sus métodos se pasan como argumento. Esto permite pasar como argumento varios valores con un único parámetro. El método `comer_animal(Animal $animal_comido)` solo acepta un argumento de tipo Animal.

Por lo tanto, no puede llamar al método de la siguiente manera:

```
$gato->comer_animal("Rana");
```

O de esta manera:

```
$gato->comer_animal(4);
```

Ya que los tipos "Rana" (String) y 4 (Integer) no son de tipo Animal.

1.8. El constructor

El constructor, como su nombre indica, sirve para construir un objeto de la clase que necesitemos. En los ejemplos anteriores, cuando escribíamos new Animal(), por defecto llamabos al constructor de la clase Animal.

Podemos crear nuestros propios constructores y así pasar como argumento el valor de los atributos que deseamos asignar a su objeto.

El constructor se designa como __construct y no tiene return. Tanto __construct como otros métodos “mágicos” llevan un doble guion bajo delante.

Para añadir un constructor que toma como argumentos el peso y el color, la página Animal.class.php queda como.

```
<?php
class Animal
```

```php
{
    // Declaración de atributos
    private $color = "gris";
    private $peso = 10;
    public function __construct($color, $peso)
    //Constructor que solicita 2 argumentos.
    {
        echo 'Llamar al constructor.<br>';
        $this->color = $color; // Inicialización del
        // color.
        $this->peso = $peso; // Inicialización del peso.
    }
    //accesos
    public function getColor()
    {
        return $this->color; //devuelve el color
    }
    public function setColor($color)
    {
        $this->color = $color; //escrito en el atributo color
    }
    public function getPeso()
    {
        return $this->peso; //devuelve el peso
    }
    public function setPeso($peso)
    {
        $this->peso = $peso; //escrito en el atributo peso
    }
    //Métodos
    public function comer_animal(Animal $animal_comido)
    {
        //el animal que come aumenta su peso tanto
        //como el del animal comido
        $this->peso = $this->peso + $animal_comido->peso;
        //el peso del animal comido y su color se restablecen a 0
        $animal_comido->peso = 0;
        $animal_comido->color = "";
    }
    public function moverse()
    {
        //método que pueda acceder a las propiedades
        //color y peso
    }
    public function añadir_un_kilo()
    {
        $this->peso = $this->peso + 1;
    }
}
```

Llamamos al constructor en la página uso.php:

```php
<?php
//carga de la clase
include('Animal.class.php');
//instanciar la clase Animal con su constructor
$perro = new Animal("beige",7);
//leer el peso
echo "El peso del perro es:".$perro->getPeso()." kg<br>";
//leer el color
echo "El color del perro es:".$perro->getColor()."<br>";
//actualizar el color del perro
$perro->setColor("negro");
//leer el color
echo "El color del perro es:".$perro->getColor()."<br>";
?>
```

Da como resultado:

Llamar al constructor.
El peso del perro es:7 kg
El color del perro es:beige
El color del perro es:negro

Se muestra en primer lugar "Llamar al constructor", ya que la instrucción echo que se ha escrito en el constructor __construct de su clase Animal se llama cada vez que ejecuta new Animal().

El constructor toma como argumento los valores de sus atributos. Esto evita llamar los métodos setColor() y setPeso().

En PHP no se puede declarar dos constructores en la misma clase.

1.9. El destructor

El destructor sirve para destruir el objeto con el fin de liberarlo de la memoria. Se llama automáticamente al final del script PHP o cuando se destruye el objeto.

Para destruir un objeto, utilizamos la función unset(), pasándole el objeto a destruir en la misma página donde hemos creado el objeto.

Podemos modificarlo si añadimos la función __destruct() en la clase.

Ejercicio 1
Crea dos peces en la página uso.php: **pez1, gris, 10 kg** **pez2, rojo, 7 kg** **Muestra su peso y a continuación haz que el pez1 se coma al pez2.** **Vuelve a mostrar su peso.**

1.10. Las constantes de clase

Una constante de clase es similar a una constante normal, es decir, un valor asignado a un nombre y que no cambia nunca. Recordemos que las constantes normales se crean con la función define()

```
define('PI',3.1415926535);
```

Una constante de clase representa una constante pero que está unida a dicha clase.

Por ejemplo, si creamos un animal con el constructor `__construct($color, $peso)`, no podemos saber inmediatamente que el $peso se corresponde con el peso del animal, ya que la instancia de la clase se realiza asignando un número, por ejemplo el 10.

```
$perro = new Animal("gris",10);
```

Para solventar esto, podemos usar constantes que representen, cada una, un peso distinto:

```
const PESO_LIGERO = 5;
const PESO_MEDIO = 10;
const PESO_PESADO = 15;
```

Las constantes siempre están en mayúsculas, sin el símbolo $ y precedidas de la palabra clave const.

La clase Animal.class.php se convierte en:

```
<?php
class Animal
{
    // Declaración de atributos
    private $color = "gris";
    private $peso = 10;
    //constantes de clase
    const PESO_LIGERO = 5;
    const PESO_MEDIO = 10;
    const PESO_PESADO = 15;

  …

?>
```

Para llamar a estas constantes desde la página uso.php, la sintaxis es algo peculiar, ya que debemos escribir **::** entre la clase y su constante:

```
<?php
//carga de la clase
include('Animal.class.php');
//instanciar la clase Animal con su constructor
$pez1 = new Animal("gris",Animal::PESO_MEDIO);
$pez2 = new Animal("rojo",Animal::PESO_LIGERO);
//leer el peso
echo "El peso del pez1 es:".$pez1->getPeso()." kg<br>";
//leer el peso
```

```
echo "El peso del pez2 es:".$pez2->getPeso()." kg<br>";
//el pez1 se come al pez2
$pez1->comer_animal($pez2);
//leer el peso
echo "El nuevo peso del pez1 es:".$pez1->getPeso()." kg<br>";
//leer el nuevo peso
echo "El nuevo peso del pez2 es:".$pez2->getPeso()." kg<br>";
?>
```

Da como resultado:

Llamada al constructor.
Llamada al constructor.
El peso del pez1 es:10 kg
El peso del pez2 es:5 kg
El nuevo peso del pez1 es:15 kg
El nuevo peso del pez2 es:0 kg

`Animal::PESO_MEDIO` siempre es 10, sea cual sea la instancia. Por lo tanto, la constante no está unida a la instancia, sino a la clase. Por este motivo la sintaxis es peculiar.

1.11. Los atributos y métodos estáticos

a. Método estático

Un método estático está unido a la clase, pero no al objeto.

Para convertir un método a estático, debemos añadir la palabra clave static delante de function.

Modificamos el método moverse() para convertirlo en estático y mostrar "El animal se mueve.".

La clase Animal.class.php se convierte en:

```
<?php
class Animal
{
    // Declaración de atributos
    private $color = "gris";
    private $peso = 10;
    //constantes de clase
    const PESO_LIGERO = 5;
    const PESO_MEDIO = 10;
    const PESO_PESADO = 15;
    public function __construct($color, $peso)
    // Constructor que solicita 2 argumentos.
    {
        echo 'Llamada al constructor.<br>';
        $this->color = $color; // Inicialización del color.
        $this->peso = $peso; // Inicialización del peso.
    }
    //accesos
```

```
    public function getColor()
    {
        return $this->color; //devuelve el color
    }
    public function setColor($color)
    {
        $this->color = $color; //escrito en el atributo color
    }
    public function getPeso()
    {
        return $this->peso; //devuelve el peso
    }
    public function setPeso($peso)
    {
        $this->peso = $peso; //escrito en el atributo peso
    }
    //métodos
    public function comer_animal(Animal $animal_comido)
    {
        //el animal que come aumenta su peso tanto como
        //el del animal comido
        $this->peso = $this->peso + $animal_comido->peso;
        //el peso del animal comido y su color se restablecen a 0
        $animal_comido->peso = 0;
        $animal_comido->color = "";
    }
    public static function moverse()
    {
        echo "El animal se mueve.";
    }
    public function añadir_un_kilo()
    {
        $this->peso = $this->peso + 1;
    }
}
```

Es imposible escribir en un método estático la palabra clave `$this`, ya que esta palabra representa al objeto, y el método estático está unido a la clase.

Por tanto, para llamar a este método desde la página uso.php, debemos utilizar la misma sintaxis que en las constantes (también unidas a la clase), es decir, introducir **::** entre la clase y su método estático:

```
<?php
//carga de la clase
include('Animal.class.php');
//llamada al método estático
Animal::moverse()
?>
```

Da como resultado:

El animal se mueve.

Podemos llamar al método estático desde un objeto, pero el método estático no puede cambiar nada de este objeto:

```php
<?php
//carga de la clase
include('Animal.class.php');
//instanciar la clase Animal con su constructor
$perro1 = new Animal("gris",Animal::PESO_MEDIO);
//llamada al método estático
$perro1->moverse();
?>
```

Da como resultado:

Llamada al constructor
El animal se mueve.

b. Atributo estático

Un atributo estático es un atributo propio de la clase y no del objeto, al igual que en los métodos estáticos. **Es el mismo principio que en una constante, salvo que el atributo está en una variable y puede cambiar su valor**.

Un atributo estático se escribe añadiendo la palabra clave static delante de su nombre.

Por ejemplo, para añadir un atributo estático que represente un contador que indica el número de veces que se instancia la clase, la clase Animal.class.php se convierte en:

```php
<?php
class Animal
{
    // Declaración de atributos
    private $color = "gris";
    private $peso = 10;
    //constantes de clase
    const PESO_LIGERO = 5;
    const PESO_MEDIO = 10;
    const PESO_PESADO = 15;
    // Declaración de la variable estática $contador
    private static $contador = 0;

  …

?>
```

Para cambiar el valor de este contador no podemos utilizar `$this`. Ya que `$this` representa un objeto (perro, gato), y no la clase Animal. El contador es de tipo estático y por lo tanto está unido a

la clase. Para llamar a este atributo en la clase, debemos utilizar la palabra clave **self**, que representa la clase.

Para añadir 1 al contador cada vez que vayamos a instanciar la clase Animal, debemos modificar el constructor. A continuación debemos añadir un método que permita leer este atributo privado con ayuda de un método de tipo public static y getContador().

La clase Animal.class.php se convierte en:

```php
<?php
class Animal
{
    // Declaración de atributos
    private $color = "gris";
    private $peso = 10;
    //constantes de clase
    const PESO_LIGERO = 5;
    const PESO_MEDIO = 10;
    const PESO_PESADO = 15;
    // Declaración de la variable estática $contador
    private static $contador = 0;
    public function __construct($color, $peso) // Constructor
    //que solicita 2 argumentos.
    {
        echo 'Llamada al constructor.<br>';
        $this->color = $color; // Inicialización del color.
        $this->peso = $peso; // Inicialización del peso.
        self::$contador = self::$contador + 1;
    }
    // método estático que devuelve el valor del contador
    public static function getContador()
    {
        return self::$contador;
    }
  …
?>
```

La página uso.php:

```php
<?php
//carga de la clase
include('Animal.class.php');
//instanciar la clase Animal
$perro1 = new Animal("rojo",10);
//instanciar la clase Animal
$perro2 = new Animal("gris",5);
//instanciar la clase Animal
$perro3 = new Animal("negro",15);
//instanciar la clase Animal
$perro4 = new Animal("blanco",8);
//llamada al método estático
```

```
echo "Número de animales que se han instanciado:".Animal::getContador();
?>
```

Da como resultado:

Llamada al constructor.
Llamada al constructor.
Llamada al constructor.
Llamada al constructor.
Número de animales que se han instanciado:4

2. La Herencia en PHP

2.1. Protected

2.2. Sustitución

2.3. Herencia en cascada

2.4. Las clases abstractas

2.5. Las clases finales

3. Los métodos mágicos en PHP