Tema 4: Desarrollo de aplicaciones web utilizando código embebido

¿Qué aprenderás?

- Cómo compartir información entre varias páginas web de una misma aplicación, usando sesiones y cookies.
- Cómo usar la Programación Orientada a Objetos en PHP.

¿Sabías que…?

- Si definimos en el archivo php.ini la variable session.auto_start = 1, automáticamente en todas las páginas de nuestra aplicación se inicializará la sesión.
- La programación orientada a objetos es un paradigma que fue tomando forma en varios lenguajes de programación en la década de los 80.
- PHP es un lenguaje de programación orientado a objetos que no soporta herencia múltiple. No obstante, una clase puede implementar varias interfaces.

1. Sesiones en PHP

1.1. Introducción

Las aplicaciones web dinámicas normalmente constan de varias páginas que comparten información. Ésta estará disponible durante un periodo de tiempo, llamado sesión, que comprende desde que el usuario ve la primera página hasta que deja nuestro sitio web (o se supera el tiempo máximo de sesión).

El uso de las sesiones es un buen método para mover información de una página a otra. PHP nos brinda funciones que configuran una sesión de usuario y almacenan la información correspondiente en el servidor. Podremos tener acceso a esta información desde cualquier página.

1.2. Funcionamiento de las sesiones

PHP nos permite configurar una sesión en una página web y guardar variables en ella. Para usar estas variables en otras páginas, debemos abrir la sesión. El funcionamiento es el siguiente:

1. Se asigna un número de identificación a la sesión. Esta identificación (un número largo) se almacena en una variable del sistema PHP llamada PHPSESSID.
2. Las variables de sesión se almacenan en un archivo en el servidor, cuyo nombre es el número de identificación de la sesión. El archivo se almacena en el directorio \tmp en Linux. En Windows se almacena en el directorio sessiondata.
3. Se pasa el número de identificación de la sesión a cada página.
4. Se obtienen las variables del archivo de la sesión para cada nueva página. Estas variables se incorporan al array $_SESSION.

1.3. Abrir la sesión

La función parar abrir la sesión es la siguiente:

```
session_start();
```

Esta función se debe incorporar en cada página web que queramos que use la sesión. Lo que hace es comprobar que ya haya un número de identificación de sesión, para entonces añadir en el array $_SESSION todas las variables de la sesión. Si no existe ningún identificador de sesión, crea uno nuevo y empieza una sesión.

IMPORTANTE: hay que tener en cuenta que la función session_start no se puede usar después de enviar cualquier output.

Podemos configurar PHP para que añada automáticamente en todas nuestras páginas la función session_start. Para ello hay que editar el archivo php.ini. Buscamos la variable session.auto_start y establecemos su valor en 1.

1.4. Uso de variables de sesión

Para guardar variables dentro de la sesión usaremos la siguiente instrucción:

```
$_SESSION[“nombre_variable”] = valor;
```

De esta forma, el valor almacenado estará disponible en otras páginas que usen la sesión. Para usar ese valor sólo tendremos que usar el array $_SESSION.

Veamos un ejemplo de uso de sesiones. Tendremos dos páginas:

Página 1: se crea la sesión y se almacenan en ella unos valores.

```
<?php
     session_start();
?>
<html>
<head><title>Página 1</title></head>
<body>
<?php
     $_SESSION[“mi_variable”] = “prueba”;
     echo “<form action=’pagina2.php’ method=’POST’>
               <input type=’submit’/>
          </form>”;
?>
</body>
</html>
```

Página 2: usamos la sesión y mostramos los valores almacenados en ella.

```
<?php
     session_start();
?>
<html>
<head><title>Página 2</title></head>
<body>
<?php
     echo $_SESSION[“mi_variable”];
?>
</body>
</html>
```

Uno de los usos típicos de las sesiones en PHP es controlar el acceso a ciertas páginas a usuarios que previamente se han identificado en nuestra aplicación. Veamos de forma esquemática cómo hacerlo:

- En la página de login, creamos una sesión y le pedimos al usuario su nombre de usuario y su contraseña mediante un formulario.

- El formulario de la página de login nos lleva a otra página que usará la sesión abierta. Conectaremos a una base de datos para comprobar que existe un usuario con el nombre y la contraseña especificados, y devolveremos el identificador de usuario. Éste valor lo almacenaremos en la sesión:

```
$_SESSION["id_usuario"] = $identificador;
```

- En las páginas a las cuales queramos restringir el acceso, usaremos el siguiente código:

```
<?php
     session_start();
     if(!isset($_SESSION["id_usuario"])){
          header("Location: login.php");
     }
     else{
          header("bienvenido.php");
     }
?>
```

1.5. Cerrar sesión

Para cerrar una sesión existente PHP nos ofrece la función:

```
session_destroy();
```

Esta función elimina toda la información de las variables de sesión. PHP ya no pasa el número de identificación de la sesión a la siguiente página.

Si lo que queremos hacer es mantener la sesión, pero eliminar alguna de las variables almacenadas en ella, usaremos la función:

```
unset($_SESSION["nombre_variable"]);
```

PHP también tiene una función para eliminar todas las variables de sesión, pero manteniendo la sesión actual abierta:

```
session_unset();
```

1.6. Cookies

Las cookies son parejas variable=valor. Éstas se almacenan en el PC del usuario mediante su explorador, en un archivo, de manera que cualquier página web puede acceder a la información que almacenen las cookies.

Las cookies se almacenan de forma indefinida en el PC del usuario, a no ser que éste las borre. Al estar en la máquina del usuario, las cookies están bajo su control, y este es el gran inconveniente que presentan: el usuario puede modificarlas, borrarlas, incluso configurar su exploradores para que no las usen. Si nuestra aplicación depende de ellas, no funcionará.

Por tanto, pese a ser una herramienta que nos ofrece PHP, es mejor trabajar con otras herramientas que nos ofrece el lenguaje.

Las cookies se almacenan por medio de la función setcookie, de la siguiente manera:

```
setcookie("variable", "valor");
```

Observemos que variable indica el nombre de la variable, pero no incluimos el signo del dólar.

Para usar una cookie en nuestro programa PHP, escribiremos lo siguiente:

```
$_COOKIE["variable"];
```

Las cookies creadas de la forma que acabamos de ver almacenarán la información hasta que el usuario salga del sitio web. Si queremos que la información almacenada en una cookie permanezca en un archivo del PC del usuarios después de que éste abandone el sitio web, usaremos la siguiente función:

```
setcookie("variable", "valor", tiempo);
```

El valor tiempo establece el tiempo, en segundos, que estará la cookie disponible hasta expirar.

¿Podemos borrar cookies? No exactamente. Lo que podemos hacer es usar uno de los siguientes métodos:

```
setcookie("variable", "");
setcookie("variable");
```

IMPORTANTE: hay que tener en cuenta que la función setcookie no se puede usar después de enviar cualquier output.

1.7. Otros métodos para mover información entre páginas

A parte de las herramientas vistas anteriormente, PHP ofrece otros métodos para hacer que nuestras páginas web compartan información. Podemos hacerlo usando:

- Información en la URL: una forma simple pero poco segura de mover información entre páginas es incluirla en la URL, siguiendo el formato variable=pareja.

  Podemos agregar las parejas variable=valor en cualquier parte de la URL. El inicio de la información se señala con un sigo de interrogación , y separaremos las parejas con el signo &. Un ejemplo:

  ```
  header("Location:
  pagina?nombre=Dani&ciudad=Barcelona");
  ```

- Formularios: es el modo más común. Cuando el usuario clica sobre el botón de envío asociado al formulario, la información de los campos pasa a la página incluida en la etiqueta action del formulario.

  El uso habitual de los formularios es el de pedirle datos al usuario. Pero podemos usar campos ocultos para pasar información de una página a otra sin necesidad de pedirle al usuario que la introduzca. De hecho, podemos crear formularios formados íntegramente por campos ocultos. Siempre necesitaremos el botón de envío, y la página nueva no se desplegará hasta que el usuario no haga clic en él

Ejemplo de uso de sesiones

2. Programación Orientada a Objetos en PHP

2.1. Introducción

La programación orientada a objetos es un paradigma de programación que usa la idea de clase como definición de un nuevo "tipo de dato", a partir de la cual podemos crear objetos, que serán instancias de la clase con un estado propio (valores propios).

En este documento veremos las herramientas que ofrece PHP 5 para crear objetos a partir de clases, y trabajar con ellos en nuestros programas.

2.2. Definición de clase

Una clase es una colección de variables y funciones que nos permiten trabajar con estas variables. Las variables también son llamadas propiedades de una clase.

Para definir una clase pondremos la palabra class seguida del nombre de la clase, y abriremos y cerraremos una llave:

```
class NombreClase {
    ...
}
```

Entre las dos llaves incluiremos el código de la clase, es decir, las variables o propiedades que tendrá la clase y los métodos o funciones para trabajar con ellas.

2.3. Objetos

Un objeto es una instancia de una clase. Dicho de otra forma, un objeto representa una clase con un estado, es decir, con valores en las propiedades de la clase.

Para crear un objeto usaremos la palabra clave new. De esta forma crearemos una instancia de la clase y la guardaremos en una variable:

```
$objeto = new NombreClase();
```

Obviamente, antes de crear un nuevo objeto, la clase debe estar declarada.

2.4. Propiedades

Las propiedades de una clase son variables que permiten guardar valores. Representan las características de la clase.

Las propiedades de una clase se declaran de la siguiente forma:

```
class NombreClase{
     var $variable1;
     var $variable2;
     ...
}
```

Las propiedades pueden inicializarse a la vez que se declaran, pero esta inicialización debe ser siempre un valor constante. No podemos inicializar una propiedad al valor de una variable, por ejemplo.

```
class NombreClase{
     var $variable1 = 10; //Declaración OK
     var $variable2 = $otraVarible; //Declaración KO
     var $variable3 = "hola ".“mundo”; //Declaración KO
}
```

Una característica importante sobre las propiedades es la visibilidad. La visibilidad de una propiedad se define anteponiendo a la declaración de la propiedad una de las siguientes palabras: public, private o protected. El significado de cada palabra es el siguiente:

- `public`: las propiedades `public` pueden ser accedidas desde cualquier parte de la aplicación.
- `private`: las propiedades `private` sólo pueden ser accedidas desde la clase donde se han declarado.
- `protected`: las propiedades `protected` pueden ser accedidas desde la clase donde se han declarado, y desde las clases que la hereden

Las variables que se declaren sin especificar su visibilidad, es decir, con la palabra `var`, se entenderán como `public`.

Ejemplo de clase con propiedades con visibilidad definida:

```
class MiClase{
```

```
        public $variable1 = "Valor 1";
        private $variable2 = "Valor 2";
        protected $variable3 = "Valor 3";
        var $variable4 = "Valor 4";
        function verVariables(){
            echo $this->variable1;
            echo $this->variable2;
            echo $this->variable3;
            echo $this->variable4;
        }
}
$objeto = new MiClase();
echo $objeto->variable1;
echo $objeto->variable2; //ERROR
echo $objeto->variable3; //ERROR
echo $objeto->variable4;
$objeto->verVariables();
```

En el código anterior cabe destacar la utilización de la palabra $this. Esta variable (también llamada pseudo-variable) se utiliza dentro de las clases para acceder a propiedades y métodos no estática (las propiedades estáticas se verán en el apartado 8). La pseudo-variable $this hace referencia al objeto actual con el que se está trabajando, es decir, el objeto que invoca un método o solicita el valor de una propiedad.

2.5. Constantes

Una clase puede contener alguna propiedad que sea constante, es decir, su valor no cambiará durante la ejecución del programa. Definiremos una propiedad constante con la palabra `const`. A continuación veremos un ejemplo:

```
class MiClase{
    const MICONSTANTE = 10;
    function verConstante(){
        echo self::MICONSTANTE;
    }
}
```

Las constantes suelen escribirse en mayúsculas. Cabe destacar cómo accedemos a una constante: usando la pseudo-variable self (no podemos usar this) seguida de dobles dos puntos (::) y el nombre de la constante.

A partir de la versión 5.3 de PHP podemos acceder al valor de una constante a través del nombre de la clase, sin necesidad de instanciar un objeto. Por ejemplo:

```
echo MiClase::MICONSTANTE;
```

Por último, hay que destacar que las constantes son públicas. No se puede definir ningún otro grado de visibilidad.

2.6. Métodos

Los métodos son funciones que se definen dentro de una clase. Su funcionalidad principal es la de trabajar con las propiedades de la clase. Los métodos se declaran anteponiendo la pablara function al nombre del método. Los métodos pueden recibir parámetros igual que cualquier función de PHP. Veamos un ejemplo de clase con la declaración de sus métodos:

```
class NombreClase{
     //Declaración de propiedades
     function metodo1(){
          //Código del método
     }
     function metodo2($paramtro1, $parametro2){
          //Código del método
     }
}
```

Los métodos, igual que las propiedades, pueden ser declarados como `public`, `private` o `pretected`. El funcionamiento de la visibilidad de los métodos es igual que el de las propiedades.

2.7. Métodos mágicos

En PHP hay una serie de métodos predefinidos que se llaman “mágicos”. Estos métodos podemos usarlos en nuestras clases y aportan funcionalidades útiles. Los métodos mágicos tienen un nombre que empieza con _ _ (doble guión bajo). A continuación veremos algunos métodos mágicos.

2.7.1. __construct()

Este método se llama constructor, y es el método que se invocará automáticamente cuando se cree un nuevo objeto, es decir, cada vez que se haga un new. Este método se suele utilizar para inicializar las propiedades de un objeto. Por ejemplo:

```
class MiClase{
     private $var1;
     private $var2;
     function __construct($parametro1, $parametro2){
          $this->var1 = $parametro1;
          $this->var2 = $parametro2;
     }
}
$objeto = new MiClase("valor 1", "valor 2");
```

Vemos cómo, al crear el nuevo objeto, llamamos al constructor de la clase con dos parámetros, tal y como se ha declarado en la función `__construct`.

2.7.2. __destruct()

Este método se llama destructor, y sirve para liberar memoria, eliminando un objeto cuando ya no es referenciado. En este método se pueden realizar las tareas que se estimen necesarias en el momento de destruir el objeto.

```
class MiClase{
     private $var1;
     private $var2;
     function __construct($parametro1, $parametro2){
          $this->var1 = $parametro1;
          $this->var2 = $parametro2;
     }
     function __destruct(){
          echo "Liberando espacio...";
     }
}
```

2.7.3. __toString()

Este método es útil para definir cómo se debe comportar un objeto cuando se le trata como un String, por ejemplo, lo que queremos que se muestre si ejecutamos la instrucción `echo $objeto`.

```
class MiClase{
     private $var1;
     private $var2;
     function __construct($parametro1, $parametro2){
          $this->var1 = $parametro1;
          $this->var2 = $parametro2;
     }
     function __destruct(){
          echo "Liberando espacio...";
     }
     function __toString(){
          return $this->var1." ".$this->var2;
     }
}
```

Hay más métodos mágicos predefinidos en PHP, se pueden encontrar en la web oficial de php.

2.8. La palabra static

La palabra `static` la podemos usar tanto con las propiedades como con los métodos de una clase. Las propiedades y métodos declarados como `static` pueden ser accedidos sin necesidad de instanciar un objeto.

Todos los objetos de la misma clase comparten las propiedades definidas como `static`. Todos los objetos acceden a la misma variable, por tanto, si un objeto modifica el valor de una variable `static`, el resto de objetos verán el nuevo valor.

Las propiedades `static` no se pueden invocar con la pseudo-variable `$this`, ni pueden ser accedidas con el operador `->`. Además, la pseudo-variable `$this` no está disponible en los métodos `static`.

Para acceder a una propiedad `static` de un objeto, o invocar un método `static`, usaremos el operador `::` (dobles dos puntos).

Veamos un ejemplo de declaración y uso de propiedades y métodos `static`:

```
class MiClase{
     public static $variable_estatica = 10;
     public static function metodoEstatico(){
          ...
     }
}
echo MiClase::$variable_estatica;
MiClase::metodoEstatico();
```

Podemos observar que en el código anterior no se ha instanciado ningún objeto de la clase `MiClase`. Se accede a la propiedad y método `static` anteponiendo el nombre de la clase seguido del operador `::`.

2.9. Herencia

La herencia es una forma de relacionar unas clases con otras. Mediante la herencia podemos hacer que una clase B herede las propiedades y métodos de otra clase A. Se dice que la clase B extiende a la clase A. Se establece una relación entre las clases A y B como clases padre (o madre) e hija.

Veamos un ejemplo en el que se declara una clase y luego otra que la hereda:

```
class MiClaseMadre{
     public function metodoMadreA(){
          echo "Soy el metodo A de la madre<br/>";
     }
     public function metodoMadreB(){
          echo "Soy el metodo B de la madre<br/>";
     }
}
class MiClaseHija extends MiClaseMadre{
     public function metodoHija(){
          echo "Soy el metodo de la hija<br/>";
     }
}
$hija = new MiClaseHija();
$hija->metodoHija();
$hija->metodoMadreA();
```

```
$hija->metodoMadreB();
```

En el código anterior se puede ver como se crea el objeto `$hija`, que puede invocar a los métodos declarados en la clase `MiClaseHija` y a los métodos declarados en la clase `MiClaseMadre`.

IMPORTANTE: hay que recordar que las clases hijas sólo heredaran las propiedades y métodos que no hayan sido declarados como `private` (es decir, `public` o `protected`).

Si hemos definido una clase madre con el método `__construct`, la clase hija debe llamar explícitamente a este método siguiendo la sintaxis `parent::__construct`. Veamos un ejemplo:

```
class MiClaseMadre{
     private $var1;
     private $var2;
     function __construct($parametro1, $parametro2){
          $this->var1 = $parametro1;
     $this->var2 = $parametro2;
     }
}
class MiClaseHija extends MiClaseMadre{
     private $var3;
     function __construct($param1, $param2, $param3){
          parent::__construct($param1, $param2);
          $this->var3 = $param3;
     }
}
```

PHP, al igual que otros populares lenguajes como Java, no soporta herencia múltiple. Es decir, una clase sólo puede heredar una única clase.

2.10. Clases abstractas

Las clases abstractas son un tipo especial de clase. Se caracterizan por llevar la palabra `abstract` en su definición, y tener como mínimo un método abstracto. Un método abstracto es aquel que no tiene código, es decir, sólo se especifica su definición. Un ejemplo de clase abstracta podría ser:

```
abstract class MiClaseAbstracta{
     //Definición de propiedades (si las tiene)
     //Definición de métodos no abstractos (si los tiene)
     abstract protected function miFuncionAbstracta();
}
```

Las clases abstractas se utilizan para ser heredadas por otras clases, que obligatoriamente tendrán que implementar los métodos abstractos de la clase abstracta. Por ejemplo, el código de una clase que herede la clase abstracta anterior sería:

```
class MiClase extends MiClaseAbstracta{
     //Definición de propiedades (si las tiene)
```

```
        //Definición de métodos propios (si los tiene)
        function miFuncionAbstracta(){
            //Implementación del método
        }
}
```

No es importante el orden en que se declaran los métodos. Es decir, los métodos abstractos no deben ir después de los métodos no abstractos, y viceversa.

2.11. Clases finales

Las clases definidas con la palabra reservada final no podrán ser heredadas por otras clases. La definición de una clase final sería:

```
final class MiClaseFinal {
        //Definición de propiedades
        //Definición de métodos propios
}
```

También podemos definir métodos con la palabra final. Estos métodos se caracterizan por que no podrán ser sobrescritos por clases hijas. El siguiente código es erróneo:

```
class MiClaseMadre {
        public function metodoQueSePuedeSobrescribir(){
            echo "Código en la clase madre";
        }
        final public function
        metodoQueNoSePuedeSobrescribir(){
            echo "Código FINAL en la clase madre";
        }
}
class MiClaseHija extends MiClaseMadre {
        public function metodoQueNoSePuedeSobrescribir(){
            echo "Código FINAL que intento sobrescribir";
        }
}
```

Vemos que la clase `MiClaseHija` intenta sobrescribir el método definido como final en la clase `MiClaseMadre`. Esto daría un error.

Ejemplo de Programación Orientada a Objetos

Test de autoevaluación

¿Cómo se crea una sesión en PHP?

a) session_start()
b) session()
c) $HTTP_SESSION('nombre de la sesión')
d) new_session()

¿Cómo se elimina una sesión en PHP?

a) session_stop()
b) delete_session()
c) session_destroy()
d) unset(session)

¿Cómo se accede a una variable de sesión?

a) $_nombre_de_variable
b) $_SESSION['nombre de variable']
c) session('nombre de variable')
d) $nombre_de_variable

¿Cómo puedo acceder a una constante de una clase?

a) `self::MICONSTANTE`
b) `$this->MICONSTANTE`
c) `$this::MICONSTANTE`
d) `NombreClase->MICONSTANTE`

¿Qué palabra se utiliza en PHP para indicar que una propiedad o método es de la clase?

a) `public`
b) `extends`
c) `abstract`
d) `static`

Recursos y enlaces

- Manual oficial de PHP: sesiones
- Manual oficial de PHP: Programación Orientada a Objetos

Conceptos clave

- **Sesión**: herramienta que nos brinda PHP para poder almacenar información en diferentes accesos a nuestra aplicación.
- **Cookie**: mecanismo para almacenar datos en el navegador del cliente.
- **Clase:** una clase es una colección de variables y funciones que nos permiten trabajar con estas variables.
- **Objetos:** un objeto es una instancia de una clase. Dicho de otra forma, un objeto representa una clase con un estado, es decir, con valores en las propiedades de la clase.
- **Propiedades:** las propiedades de una clase son variables que permiten guardar valores. Representan las características de la clase.
- **Método:** los métodos son funciones que se definen dentro de una clase.

Ponlo en práctica

Actividad 1

Crea en PHP una página con un formulario con el que el usuario introduzca su nombre. El formulario nos llevará a una página intermedia en la que el nombre se guardará en la sesión y automáticamente se nos redirigirá a una tercera página. En ésta se comprueba si existe un nombre de usuario en la sesión. Si es así, se mostrará el mensaje "BIENVENID@" + nombre, sino se nos mostrará el mensaje "DEBE INTRODUCIR ANTES SU NOMBRE" y se nos redirigirá a la página del formulario transcurridos 3 segundos.

SOLUCIÓN

Actividad 2

Crea en PHP la clase vehículo, con las siguientes propiedades: marca, modelo y potencia. Se deben crear los métodos necesarios para crear vehículos, consultar y modificar sus propiedades, e imprimir por pantalla la información de un vehículo (todos sus datos).

SOLUCIÓN

Actividad 3

Crea en PHP la clase coche que herede las propiedades y métodos de la clase vehículo del ejercicio anterior. Además, la clase coche tendrá las propiedades puertas y ruedas. Esta última tendrá el valor 4, y todos los coches que se creen compartirán el mismo valor. Se deben crear los métodos necesarios para crear coches, consultar y modificar sus propiedades, e imprimir por pantalla la información de un coche (todos sus datos).

SOLUCIÓN