Tema 6. Generación dinámica de páginas

DESARROLLO EN ENTORNO SERVIDOR 2º DAW

MARÍA CRIADO DOMÍNGUEZ

Objetos

Características

- 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.

Objetos

Ventajas

- 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 extiendan 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.

Objetos

La estructura de los objetos se define en las clases. En ellas se escribe el código que define el comportamiento de los objetos y se 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).

Objetos

Nivel de acceso

- **public** los atributos o métodos public pueden utilizarse directamente por los objetos de la clase en cualquier sitio donde se instancie el objeto.
- **private** los atributos o métodos solo pueden ser accedidos y modificados por los métodos definidos en la clase, no directamente por los objetos de la clase.
- **protected** atributos o métodos (“privados”) visibles en las subclases (heredados) getAtributo() método que nos permite acceder a un atributo privado setAtributo() método que nos permite establecer un atributo privado $this. referencia al objeto que hace la llamada

Objetos

Ejemplo

```
class Producto {
     private $codigo;
     public $nombre;
     public $PVP;

     public function muestra() {
     print "" . $this->codigo . "";
     }
}
$p = new Producto();
```

Objetos

- Métodos mágicos que, si se implementan, son utilizados por defecto
- __set()
- __get()
- __construct()
- __destruct()
- __call()
- __clone()
- __toString()

Objetos

Estáticos

- const constantes de clase static atributos y métodos estáticos – atributos y métodos de clase (public static o private static)
  - :: operador de resolución e ámbito (-> para objetos :: para clases)
  - self:: clase actual (this-> para objetos self:: para clases)
- Utilizaremos self:: para acceder a las constantes de clase y a los atributos y métodos estáticos.
- Utilizaremos Clase::metodoEstatico() para acceder a los métodos estáticos y públicos de una clase.

Objetos

Operador instanceof: if ($p instanceof Producto)

Funciones útiles para el manejo de objetos y clases:

- get_class devuelve el nombre de la clase del objeto
- class_exits devuelve true si la clase está definida
- get_declared_classes devuelve un array con los nombres de las clases definidas
- class_alias crea un alias para una clase
- get_class_methods devuelve un array con los nombres de los métodos accesibles
- method_exits devuelve true si existe el método (sea accesible o no)
- get_class_vars devuelve un array con los nombres de los atributos accesibles
- get_object_vars devuelve un array con los nombres de los métodos accesible
- property_exists devuelve true si existe el atributo (sea accesible o no)

Objetos

Comparación de objetos == si son de la misma clase y tienen el mismo valor en sus atributos (pero son dos objetos distintos)

Comparación de objetos === si son el mismo objetos, dos referencias al mismo objeto Alias – el mismo objeto con varios nombres – referencias al mismo objeto

Serialización – almacenamiento de un objeto en un formato no orientado a objeto

Unserialize – recuperar un objeto de su almacenamiento no orientado a objeto

Serialización automática – cuando guardamos objetos en la sesión

Serialización explicita – para guardar objetos en una base de datos relacional

Objetos Herencia

Mecanismo de la POO que nos permite definir nuevas clases en base a otra ya existente.

```
class subclase extends superclase {}
```

Los nuevos objetos de la subclase son también objetos de la superclase.

Utilizaremos la visibilidad protected en la superclase con aquellos atributos o métodos privados que queremos que se hereden en la subclase. Los atributos o métodos privados no se heredan en la subclase.

Polimorfismo: Podemos redefinir los métodos protected creando otro método en la subclase con el mismo nombre.

Funciones asociadas a la herencia:

- get_parent_class: Devuelve el nombre de la clase padre del objeto o clase que se le pasa.
- is_subclass_of: Devuelve true si el objeto o clase del primer parámetro tiene como clase base la que se indica en el segundo parámetro.

Objetos Herencia

Sintaxis de la herencia

- La herencia en PHP se consigue indicando la cláusula extends seguida de la clase que proporciona la herencia.
- Se heredan todos los métodos públicos y protegidos de la clase padre. A menos que una clase sobrescriba esos métodos, mantendrán su funcionalidad original.
- Se pueden sobreescribir métodos heredados para adecuarlos al comportamiento de la clase hija.
- Se utiliza la cláusula parent:: para acceder a los miembros de la clase padre desde una clase heredada.
- El método constructor de la clase hija se debe redefinir para ajustarlo a las variables miembro de dicha clase.

Objetos Interface

- Son declaraciones de funcionalidades que tienen que cubrir las clases que implementan la interfaz.
- En una interfaz se definen un juego de funciones, es decir, se declaran una serie de métodos o funciones sin especificar ningún código fuente asociado.
- Las clases que implementan la interfaz serán las encargadas de proporcionar un código a los métodos que contiene esa interfaz.
- Para definir una interfaz se utiliza la palabra clave interface.
- Para implementar una interfaz se utiliza la palabra clave implements.
- Se pueden implementar varias interfaces en una misma clase => herencia múltiple

Objetos Interface

Un interfaz es como una clase vacía que solo contiene declaraciones de métodos.

Se definen utilizando la palabra interface (sin la palabra class).

interface nombreinterface {}

Se utilizan en las clases que están obligadas a implementar estos métodos (entre otros).

La clase que implementa una interfaz o varias la añade en su definición con la palabra implemets seguida de las interfaces que implementa.

class Nombredeclase implements nombreinterface {}

Todos los métodos que se declaren en un interfaz deben ser públicos. Además de métodos, lo interfaces podrán contener constantes pero no atributos. Un interfaz es como un contrato que la clase debe cumplir.

Funciones para interfaces:

get_declared_interfaces devuelve un array con el nombre de los interfaces declarados

interface_exists true si existe el interfaz

Objetos
Clase abstracta

- Son clases que no se pueden instanciar
- Sirven de base para otra clases
- Son como “plantillas”
- Se declaran con la palabra clave abstract

```php
abstract class miClaseBase {
    abstract protected function getItem();
    abstract protected function quantity($qty);
    public function listItem() {
        $result = '<p>' . $this->getItem() . '</p>';
        return $result;
    }
}
```

MVC

El patrón Modelo Vista Controlador MVC divide el código en 3 capas:

El proceso de desarrollo llamado Model View Controller (MVC), que es un estándar de programación de aplicaciones, utilizado tanto para hacer sitios web como programas tradicionales.

MVC (Model-View-Controller) es un patrón de diseño de software en torno a la interconexión de los tres tipos de componentes principales en un lenguaje de programación como PHP, a menudo con un fuerte enfoque en la programación orientada a objetos (POO).

MVC
El modelo

En el modelo mantendremos encapsulada la complejidad de nuestra base de datos y simplemente crearemos funciones para recibir, insertar, actualizar o borrar información de nuestras tablas.

Al mantenerse todas las llamadas a la base de datos en un mismo código, desde otras partes del programa podremos invocar las funciones que necesitemos del modelo y éste se encargará de procesarlas.

En el modelo nos podrán preocupar cosas como el tipo de base de datos con la que trabajamos, o las tablas y sus relaciones, pero desde las otras partes del programa simplemente llamaremos a las funciones del modelo sin importarnos qué tiene que hace éste para conseguir realizar las acciones invocadas.

El modelo representa las estructuras de datos, normalmente, en clases que contendrán métodos que le ayudarán a recuperar, insertar y actualizar información en la base de datos.

MVC
La vista

La vista es donde se encontrarán todos los elementos de la interfaz de usuario de una aplicación, esta puede contener código HTML, hojas de estilo CSS y archivos Javascript.

Cualquier cosa que el usuario pueda ver, es guardado en la vista, y algunas veces lo que ve el usuario en un momento es la combinación de varias vistas en la misma petición.

Una vista normalmente será una página web, pero una vista también puede ser un fragmento de página como un encabezado o pie de página.

MVC
El controlador

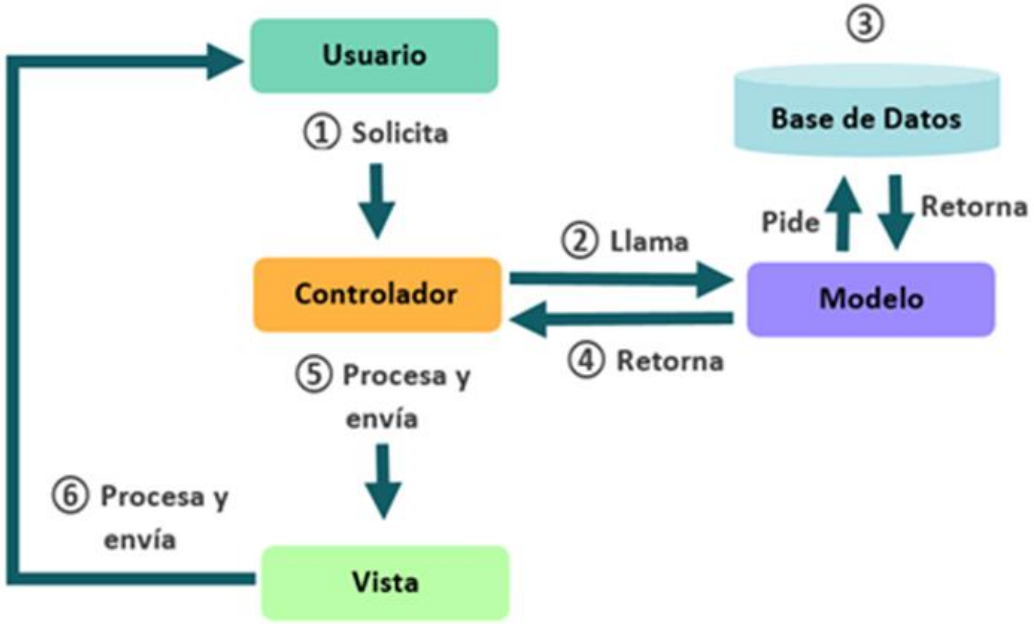
El controlador es el componente encargado de conectar el modelo con la vista.

Los controladores son el primer punto de entrada en estos componentes, ya que la primera solicitud se pasa a un controlador.

En el controlador guardamos la lógica de nuestras páginas y realizamos todas las acciones que sean necesarias para generarlas, ayudados del modelo o la vista.

MVC
Ciclo

1. El usuario realiza una petición. El controlador captura la petición del usuario.
2. El controlador llama al modelo.
3. El modelo interactúa con la base de datos, y retorna la información al controlador.
4. El controlador recibe la información, la procesa y la envía a la vista.
5. La vista procesa la información recibida y la entrega de una manera visualmente entendible al usuario



MVC
Ventajas vs Inconvenientes

1. La separación del *Modelo* y la *Vista*, lo cual logra separar los datos, de su representación visual.
2. Facilita el manejo de errores.
3. Permite que el sistema sea escalable (aumentar sus componentes o funciones sin comprometer la operabilidad y la calidad).
4. Es posible agregar múltiples representaciones de los datos.

1. La cantidad de archivos que se deben mantener incrementa considerablemente.
2. La curva de aprendizaje es más alta que utilizando otros modelos.
3. Su separación en capas, aumenta la complejidad del sistema.

MVC
El controlador