PROGRAMACIÓN WEB EN ENTORNO SERVIDOR UD-11

OBJETOS

ÍNDICE

[VARIABLES DE SERVIDOR 3](#_Toc147867406)

[A. $\_SERVER[] 3](#_Toc147867407)

[VALIDACIÓN DE DATOS 4](#_Toc147867408)

[TEXTO IMPRIMIBLE 4](#_Toc147867409)

[INTRODUCCIÓN DE <br /> 4](#_Toc147867410)

[FORMATEO SALIDA CON PRINTF 5](#_Toc147867411)

[FORMATEO SALIDA CON NUMBER\_FORMAT 5](#_Toc147867412)

[FUNCIONES Y CONSTANTES MATEMÁTICAS 6](#_Toc147867413)

# CLASES

Una clase es una plantilla que define las propiedades y métodos para poder crear objetos. De esta manera, un objeto es una instancia de una clase.

Tanto las propiedades como los métodos se definen con una visibilidad (quien puede acceder)

* Privado - private: Sólo puede acceder la propia clase.
* Protegido - protected: Sólo puede acceder la propia clase o sus descendientes.
* Público - public: Puede acceder cualquier otra clase.

Para declarar una clase, se utiliza la palabra clave class seguido del nombre de la clase. Para instanciar un objeto a partir de la clase, se utiliza new.

Para acceder a un atributo o un método utilizamos el operador ->

Desde dentro de una clase, podemos referirnos a ella misma con el nombre $this.

Las clases se nombran en mayúsculas y es recomendable usar un fichero para cada una.

Texto

Descripción generada automáticamente

# ENCAPSULACIÓN

Las propiedades se definen privadas o protegidas (si queremos que las clases heredadas puedan acceder).

Para cada propiedad, se añaden métodos públicos (getter/setter).

Las constantes se definen públicas para que sean accesibles por todos los recursos.

Texto

Descripción generada automáticamente

Los objetos que se envían y reciben como parámetros siempre se pasan por referencia.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza baja

# CONSTRUCTOR

El constructor de los objetos se define mediante el método mágico \_\_construct. Puede o no tener parámetros, pero sólo puede haber un único constructor.

Una de las grandes novedades que ofrece PHP 8 es la simplificación de los constructores con parámetros, lo que se conoce como promoción de las propiedades del constructor.

Para ello, en vez de tener que declarar las propiedades como privadas o protegidas, y luego dentro del constructor tener que asignar los parámetros a estás propiedades, el propio constructor promociona las propiedades. No es recomendable su uso, por quedar los atributos menos visibles.

Texto

Descripción generada automáticamente

Escala de tiempo

Descripción generada automáticamente

# CLASES ESTÁTICAS

Son aquellas que tienen propiedades y/o métodos estáticos (también se conocen como de clase, porque su valor se comparte entre todas las instancias de la misma clase).

Se declaran con static y se referencian con ::.

Si queremos acceder a un método estático, se antepone el nombre de la clase: Producto::nuevoProducto() y no el del objeto.

Si desde un método queremos acceder a una propiedad estática de la misma clase, se utiliza la referencia self: self::$numProductos.

Las propiedades y métodos estáticos son útiles cuando deseas tener datos o funcionalidad compartida entre todas las instancias de la clase, o cuando necesitas acceder a ellos sin crear objetos de la clase. Las propiedades estáticas pueden ser compartidas entre todas las instancias de la clase, mientras que las propiedades de instancia son específicas de cada objeto creado a partir de la clase.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

# FUNCIONES

Al trabajar con clases y objetos, 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 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 de lo indicado, que sea accesible desde dónde se llama.

Texto

Descripción generada automáticamente

# CLONACIÓN DE OBJETOS

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.

Si queremos modificar el clonado por defecto, hay que definir el método mágico \_\_clone( ) que se llamará después de copiar todas las propiedades.

Este método se suele utilizar si el objeto tiene referencias a otros objetos en sus atributos. En este caso lo más probable es que queramos generar una copia de estos atributos del otro objeto, y no apuntar a los atributos del objeto clonado.

Se pueden realizar otras acciones con el método \_\_clone( ), como por ejemplo hacer un cambio en la base de datos…

# HERENCIA

PHP soporta herencia simple, de manera que una clase solo puede heredar de otra, no de dos clases a la vez. Para ello se utiliza la palabra clave extends. Si queremos que la clase A hereda de la clase B haremos: class A extends B El hijo hereda los atributos y métodos públicos y protegidos. Existen unas funciones para comprobar la herencia:

* get\_parent\_class(object): string: de Vuelve la clase padre.
* is\_subclass\_of(object, string): bool: indica si el objeto es de una subclase de la clase pasada como parámetro.

Podemos crear métodos en los hijos con el mismo nombre que el padre, cambiando su comportamiento. Para invocar a los métodos del padre -> parent::nombreMetodo()

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.

Texto

Descripción generada automáticamentePantalla de computadora con un texto en blanco

Descripción generada automáticamente con confianza mediaTexto

Descripción generada automáticamente

# CLASES ABSTRACTAS

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.

Cuando una clase hereda de una clase abstracta, obligatoriamente debe implementar los métodos que tiene el padre marcados como abstractos.

Captura de pantalla de un celular

Descripción generada automáticamente

# CLASES FINALES

Son clases opuestas a abstractas, ya que evitan que se pueda heredar una clase o método para sobreescribirlo.

Texto

Descripción generada automáticamente

# INTERFACES

Permite definir un contrato con las firmas de los métodos a cumplir (que es lo que va a hacer). Así pues, sólo contiene declaraciones de funciones y todas deben ser públicas.

Se declaran con la palabra clave interface y luego las clases que cumplan el contrato lo realizan mediante la palabra clave implements.

Se permite la herencia de interfaces. Además, una clase puede implementar varios interfaces (en este caso, sí soporta la herecia múltiple, pero sólo de interfaces).

Texto

Descripción generada automáticamente

# MÉTODOS MÁGICOS

Todas las clases PHP ofrecen un conjunto de métodos, también conocidos como

magic methods que se pueden sobreescribir para sustituir su comportamiento

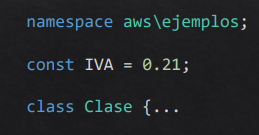
* \_\_construct( )
* \_\_destruct( ) → se invoca al perder la referencia. Permite cerrar una conexión a la BD, un fichero, ...
* \_\_toString( ) → representación del objeto como cadena. Al ejecutar echo $objeto, se ejecuta automáticamente este método.
* \_\_get(propiedad), \_\_set(propiedad, valor) → Permitiría acceder a las propiedad privadas, aunque siempre es más legible/mantenible codificar los getter/setter.
* \_\_isset(propiedad), \_\_unset(propiedad) → Averiguar o quitar el valor a un atributo
* \_\_sleep( ), \_\_wakeup( ) → Se ejecutan al recuperar (unserialize^) o almacenar un objeto que se serializa (serialize), y se utilizan para permite definir qué propiedades se serializan.
* \_\_call(), \_\_callStatic() → Se ejecutan al llamar a un método que no es público. Permiten sobrecargan métodos.

# NAMESPACES

Los Namespaces permiten organizar las clases/interfaces, funciones y/o constantes de forma similar a los paquetes en Java.

Es recomendable usar un sólo namespace por archivo y crear una estructura de carpetas respectando los niveles/subniveles (igual que se hace en Java).

Se declaran en la primera línea mediante la palabra clave namespace seguida del nombre del espacio de nombres asignado (cada subnivel se separa con la barra invertida \): Por ejemplo, para colocar la clase Producto dentro del namespace aws\ejemplos lo haríamos así



Para referenciar a un recurso que contiene un namespace, primero hemos de tenerlo disponible haciendo uso de include o require. Si el recurso está en el mismo namespace, se realiza un acceso directo (se conoce como acceso sin cualificar).

Existen tres tipos de acceso:

* Sin cualificar: recurso
* Cualificado: rutaRelativa\recurso → no hace falta poner el namespace completo
* Totalmente cualificado: \rutaAbsoluta\recurso

# use

Para evitar la referencia cualificada podemos declarar el uso mediante use (similar a hacer import en Java). Se hace en la cabecera, tras el namespace:

Los tipos posibles son:

* use const nombreCualificadoConstante
* use function nombreCualificadoFuncion
* use nombreCualificadoClase
* use nombreCualificadoClase as NuevoNombre // para renombrar elementos

En resumen, use permite acceder sin cualificar a recursos que están en otro namespace. Si estamos en el mismo espacio de nombre, no necesitamos use.

Por ejemplo, si queremos utilizar la clase \aws\ejemplos\Producto desde un recurso que se encuentra en la raíz, por ejemplo, en inicio.php, haríamos:

Imagen que contiene texto, celular, teléfono, sostener

Descripción generada automáticamente

# ORGANIZACIÓN DEL CÓDIGO

Los archivos que interactúan con el navegador se colocan en el raíz, y las clases que definamos van dentro de un namespace (y dentro de su propia carpeta src o app).

Colocaremos cada recurso en un fichero aparte.

En la primera línea indicaremos su namespace (si no está en el raíz).

Si utilizamos otros recursos, haremos un include\_once de esos recursos (clases, interfaces, etc...).

Cada recurso debe incluir todos los otros recursos que referencie: la clase de la que hereda, interfaces que implementa, clases utilizadas/recibidas como parámetros, etc...

Si los recursos están en un espacio de nombres diferente al que estamos, emplearemos use con la ruta completa para luego utilizar referencias sin cualificar.