Contenido

17/02/2020 (Presencial 5): Unidad 4: Programación orientada a objetos en PHP.

En la tutoría presencial hemos repasado la teoría viendo algunos de los ejemplos que se adjunta.

CREACIÓN DE CLASES

- Una clase se declara en PHP con la palabra **class** seguida del nombre que queramos darle **(es conveniente buscar un nombre que identifique lo mejor posible lo que representa)** y a continuación, entre llaves, se definen los miembros de la clase. Conviene hacerlo de forma ordenada, primero los atributos y después los métodos.
- Es preferible que **cada clase esté en su propio fichero**
- **Los nombres de las clases suelen comenzar por letra mayúscula** para de esta forma distinguirlos de los objetos y otras variables.

Ejercicio carpeta 22_Clases_00_PrimeraClase

Definición sencilla de una clase Coche, y como se acceden a los atributos desde los métodos de la clase. Creación de un objeto de la clase y acceso a las propiedades y métodos.

Visibilidad de los atributos de una clase

Cuando se declara un atributo de una clase debe definirse su nivel de acceso (estos niveles de acceso son también válidos para las constantes definidas en una clase y los métodos). Puede ser :

- **Public:** pueden usarse directamente por los objetos de la clase.
- **Private:** los atributos privados solo son accedidos y modificados por los métodos definidos en la clase, no por los objetos instanciados.
- **Protected:** solo son accesibles y modificables desde la clase donde se han definido y desde las clases que heredan de la clase donde están definidos.

Motivos para crear los atributos privados:

- El valor de los atributos forma parte de la información interna del objeto y no de su interface.
- Mantener un cierto control sobre sus valores.

Cuando un objeto invoca a un método de la clase, siempre se le pasa una referencia al objeto que hizo la llamada. Esta referencia se almacena en la variable **$this** que podemos utilizar dentro de los métodos de la clase para acceder a sus atributos.

Os recomiendo que miréis el ejemplo de la documentación oficial sobre la visibilidad en los métodos de una clase:

https://www.php.net/manual/es/language.oop5.visibility.php

Ejercicio carpeta 22_Clases_01_Menus:

Página del ejercicio:

http://www.tutorialesprogramacionya.com/phpya/poo/temarios/descripcion.php?cod=37&punto=3&inicio=0

Este ejercicio consiste en guardar en dos arrays los elementos de un menú típico de una página web. Para ello tenemos dos archivos:

Fichero Menu.php

Se define una clase Menu.php en la que se declaran dos atributos de tipo privado que se inicializan con un array vacío. En ellos se almacenarán los enlaces y nombres de los enlaces a páginas web.

Además se definen dos métodos, uno para cargar opciones a los arrays anteriormente mencionados, y otro para generar los enlaces HTML de forma dinámica utilizando un bucle.

Fichero index.php

Crea un objeto de la clase Menu, e invoca el método cargarOpcion para asignar valores a las propiedades.

Después invoca al método mostrar() para que generar de forma dinámica los enlaces a partir del contenido de los atributos.

Observa que hay que utilizar métodos de la clase para poder asignar o acceder a los valores de las propiedades. Esto es debido a que los atributos están declarados como **private**.

Métodos get() y set()

Aunque no es obligatorio, los métodos que nos permiten obtener y/o modificar los valores de los atributos privados se suelen llamar igual que la propiedad precedidos de **"get" o "set" respectivamente.**

Definir constantes en una clase

Se definen utilizando la palabra **const**. Características

- Las constantes no pueden cambiar su valor,
- No usan el carácter $ en su definición.

- Su valor va siempre entre comillas y está asociado a la clase, es decir, no existe una copia del mismo en cada objeto.
- 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** (que se utiliza para acceder a los elementos de una clase). *Persona::PERSONA_MUJER*
- Los nombres de las constantes **se suelen escribir en mayúsculas** para diferenciarlos de las propiedades**.**
- Para acceder a las constantes de una clase, no es necesario que exista un objeto creado.

Métodos constructor y destructor de la clase

Método constructor:

- En PHP se pueden definir en las clases métodos constructores que se ejecutan cuando se crea el objeto (es el primer método que se ejecuta al creare un objeto de una clase).
- Puede llamarse **__construct()** o con el nombre de la clase.
- Solo puede haber existir un constructor.
- Se suelen usar para asignar valores a las propiedades.
- Puede recibir parámetros (que deben pasarse cuando se crea el objeto con la palabra new), o llamar a otros métodos.
- Los constructores no devuelven datos.

Método destructor:

- También puede definirse un método destructor que permite definir acciones que se ejecutarán cuando se elimine el objeto.
- Debe llamarse **__destruct()**.
- No lleva argumentos.
- Normalmente se ejecuta de forma automática al finalizar el script o función que usa el objeto.
- Podemos forzar a que se ejecute si liberamos la memoria que ocupa del objeto mediante la instrucción **unset($obj).**

Ejercicio carpeta 22_Clases_01_crearClases:

Fichero Persona.php

Contiene la definición de la clase Persona y sus atributos y métodos:

Atributos: $nombre y $apellidos (privados) y $edad (público)

Constantes: PERSONA_HOMBRE y PERSONA_MUJER.

Métodos: construct, destruct, getNombre, setNombre, getApellidos, setApellidos.

Fichero crearClase.php

- Se muestra el valor de una constante definida en la clase sin instanciarla.
- Se crea un objeto de la clase Persona y se asignan valores a los atributos Nombre y Apellidos del objeto creado utilizando los métodos **set()** definidos en la clase, ya que los atributos han sido declarados como private y por tanto no pueden ser accedidos directamente desde el objeto creado.
- En cambio el atributo edad, al ser público puede ser accedido directamente desde el objeto.
- Se muestran por pantalla los valores del objeto llamando a los métodos getNombre() y getApellidos() y accediendo directamente al atributo público.
- Forzamos que se ejecute el destructor borrando el objeto instanciado. En el ejemplo al ejecutar el destructor (tanto de forma manual o automáticamente cuando finaliza el script) muestra un mensaje.

Atributos y métodos estáticos.

- Una clase puede tener atributos y/o métodos estáticos. Se definen anteponiendo la palabra **static** después del modificador de visibilidad.
- Declarar propiedades o métodos como estáticos hacen que sean accesibles sin necesidad de instanciar la clase.
- Los atributos estáticos no pueden ser llamados desde un objeto de la clase. Deben llamarse utilizando el nombre de la clase y el operador de resolución de ámbito.
- Si además de estático, el atributo es privado, solo se podrá acceder a él desde los métodos de la propia clase usando la palabra **self** seguida del operador de resolución de ámbito. Del mismo modo que $this hace referencia al objeto actual, self hace referencia a la clase actual.
- Los atributos estáticos de una clase se usan para almacenar información general sobre la misma, por ejemplo el número de objetos de la clase que hay instanciados. Solo existe un único valor para ese atributo que se almacena a nivel de clase (no de objeto)
- Los métodos estáticos se usan para realizar alguna tarea específica o devolver un objeto concreto. Por ejemplo las clases matemáticas suelen tener métodos estáticos para hacer operaciones como logaritmos o raíces cuadradas. No tiene sentido crear un objeto para realizar una operación matemática.
- Si el método estático es público se puede acceder con el nombre de la clase y el operador de resolución de ámbito o con una instancia de un objeto y el

operador flecha, aunque se suele utilizar el operador de resolución de ámbito.

- Como pueden ser invocados sin necesidad de instanciar un objeto, la pseudovariable $this no está disponible dentro de los métodos estáticos. En su lugar se usa **::self**
- Si queremos acceder a un método estático

Ejercicio carpeta 22_Clases_02_static

Esta carpeta contiene varios ejemplos para verla forma de trabajar con propiedades y métodos estáticos de una clase:

Ficheros Coche.php e index.php

Este ejemplo muestra cómo se puede acceder a las propiedades y métodos estáticos de una clase tanto desde los métodos definidos en ella, usando **self::** como desde un script externo a la clase mediante el nombre de la clase seguido del operador de resolución y el nombre del método o atributos al que deseamos acceder.

Fichero Llamar_metodos_desde_metodos.php

Este ejemplo muestra:

- Cómo llamar a un método estático desde un método estático dentro de la clase.
- Cómo llamar a un método estático desde un método no estático dentro de la clase.
- Cómo llamar a un método no estático desde un método estático dentro de la clase.

Finalmente, desde fuera de la definición de la clase se crea un objeto y se invoca a un método no estático y a un método estático tanto usando la instancia de la clase como sin utilizarla.

Ejercicio carpeta 22_Clases_02_static_01

- Este ejemplo lo podéis encontrar en el canal de YouTube de Píldoras informáticas.
- Este ejemplo simula (de forma muy básica), cómo podría ser el funcionamiento de un concesionario de coches.
- Tenemos una clase Vehículo en la que están definidos dos atributos, precio (privado) y descuento (privado y estático, por lo que solo existirá a nivel de clase, no de objeto).

- Dentro de la clase se definen una serie de métodos (incluido el constructor) que establecerán un precio de partida del vehículo en función de un tipo que se indica en el constructor, y a partir de las opciones de configuración que el cliente quiera, el precio se irá incrementando.
- En un script externo, en primer lugar se invoca al método estático que comprueba si hay algún descuento a aplicar al precio.
- A continuación se crean varias instancias de la clase, una por cliente, y se establecen las opciones de configuración que cada cliente desee.
- Finalmente se muestra el precio total del vehículo.

Ejercicio Carpeta 22_Clases_03_funciones

Este ejemplo muestra el uso de las funciones de clase y objeto mencionadas en la teoría y los resultados que se muestran dependiendo desde el lugar en que se invoquen y la visibilidad de los elementos de la clase.

Aspectos a tener en cuenta sobre la utilización de los objetos

- Cuando un objeto llama a un método de una clase **se le pasa siempre una referencia** al objeto que hizo la llamada.
- Esta referencia se almacena en la variable **$this**. Se utiliza, por ejemplo, para acceder a los atributos privados del objeto (que sólo son accesibles desde los métodos de la clase).
- **Una clase puede tener atributos o métodos estáticos** que se definen anteponiendo la palabra **static.**
- **Definir las propiedades y métodos de una Clase en PHP como static hace que podamos acceder a ellos sin necesidad de crear una instancia de la misma (sin tener que crear un Objeto).**
- **Los atributos y métodos estáticos no pueden ser llamados desde un objeto de la clase utilizando el operador ->.**
- Si el método o atributo es público, deberá accederse utilizando el **nombre de la clase y el operador de resolución de ámbito ":: ".**
- Si es privado, **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**.
- Usaremos **parent::** para acceder a las **propiedades definidas como const o static de la Clase padre, así como a sus métodos** (aunque no sean de tipo static).
- si usamos self:: para acceder a métodos o propiedades que no existen en la clase hijo pero sí en la clase padre, se accederá a ellos como si usáramos parent::.

- Los atributos estáticos de una clase se utilizan para guardar información general sobre la misma, como puede ser el número de objetos que se han instanciado. Sólo existe un valor del atributo (NO HAY UNO POR CADA OBJETO QUE SE HAYA CREADO), que se almacena a nivel de clase.
- **Los métodos estáticos se llaman desde la clase. No es posible llamarlos desde un objeto y, por tanto, no podemos usar $this dentro de un método estático.**
- Una vez creado un objeto, puedes utilizar el operador **instanceof** para comprobar si es o no una instancia de una clase determinada.
- Puedes 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.
- Si cuando se realiza la llamada, el parámetro no es del tipo adecuado, se produce un error que podrías capturar. Además, ten en cuenta que sólo funciona con objetos (y a partir de PHP5.1 también con arrays).
- Cuando pasamos como parámetro un objeto, este se pasa siempre por referencia, es decir, Es decir los cambios que hagamos quedan reflejados en el objeto original.
- Ocurre lo mismo si asignamos un objeto a una variable. Tendremos dos identificadores del mismo objeto (solo tenemos un objeto al que podemos acceder mediante dos nombres).
- Para obtener una copia de un objeto que podamos manipular sin que los cambios afecten al original, usaremos **clone(obj)**.
- A veces tienes dos objetos y quieres saber su relación exacta. Para eso, en PHP5 puedes utilizar los operadores **==** y **===**.
- Si utilizas el operador de comparación **==**, comparas los valores de los atributos de los objetos. Por tanto dos objetos serán iguales si son instancias de la misma clase y, además, sus atributos tienen los mismos valores.
- Sin embargo, si utilizas el operador ===, el resultado de la comparación será true sólo cuando las dos variables sean referencias al mismo objeto.
- **Si utilizas objetos en tus programas, es recomendable que te asegures de la versión del intérprete que los va a ejecutar, utilizando por ejemplo la función phpversion porque no funcionan de la misma manera en versiones anteriores.**

Ejercicio carpeta 22_Clases_04_copiarObjetos

Este es un ejemplo que muestra cómo hacer una copia de un objeto creando dos instancias distintas, además de hacer comprobaciones entre objetos para ver si son el mismo o no.

Mecanismos de mantenimiento del estado

- **A partir de la versión 7 de PHP se puede guardar un objeto en una variable de sesión sin necesidad de serializarlo (incluso guarda las propiedades estáticas)**. Después se puede recuperar directamente en una variable.

Ejercicio carpeta 22_Clases_05_serialize

- Tenemos una clase "Menú" definida, en la que tenemos dos atributos privados que son arrays, y uno público static. Además hemos definido dos métodos, uno para cargar enlaces en el menú y otro para mostrar el contenido.
- Vamos a ver el mismo ejemplo serializando y sin serializar.

En el ejemplo hay dos carpetas:

- En la carpeta conSerialize se muestra cómo guardar objetos en una variable de sesión utilizando **serialize** y recuperar su contenido con **unserialize.**
- En la carpeta sinSerialize se muestra que ya no es necesario utilizar serialize y unserialize para guardar/recuperar objetos en variables de sesión (a partir de la versión 7 de PHP)

Relacionar clases

- Lo normal, cuando desarrollamos una aplicación web, es que tengamos varias clases y que algunas estén relacionadas entre sí.
- Hay varias formas de relacionar clases:
    - Una forma sencilla de relacionar dos clases es instanciando una en otra.
    - Otra forma de relacionar clases es pasar como parámetro a un método de una clase un objeto de otra.
    - Otra manera de relacionar clases, es creando objetos de una en la definición de otra.

Ejercicio carpeta 22_Clases_06_relacionar01

En este ejemplo se muestra la forma de relacionar dos clases instanciando una en otra. Tenemos definidas dos clases y un script principal.

El fichero ClaseDos.php define dos atributos privados, num1 y num2, el método constructor, que recibe dos números como parámetros, y el

método multiplica que devuelve el resultado de multiplicar los valores de esas propiedades.

El fichero ClaseUno.php tiene un atributo privado y un método que recibe dos parámetros y en el que se crea un objeto de la clase2 (hay que incluir el script ClaseDos.php) al que se le pasan los parámetros recibidos. Este método devuelve el resultado de invocar el método multiplica de la ClaseDos.

El script index.php crea dos objetos de la ClaseUno e invoca al método de la ClaseUno con dos números como parámetros

Ejercicio carpeta 22_Clases_06_relacionar02_01

Este ejemplo muestra la forma de relacionar dos clases pasando como parámetro a un método de una clase, un objeto de otra.

ClaseUno.php: contiene dos atributos privados, el constructor (en este caso tiene el mismo nombre que la clase), que recibe dos parámetros y se los asigna a las propiedades de la clase y los métodos get que devuelven el valor de cada una de las propiedades.

ClaseDos.php: define un método "multiplica" que recibe como parámetro un objeto de la ClaseUno. Devuelve el resultado de multiplicar los atributos de la ClaseUno obtenidos mediante sus respectivos métodos get.

Index.php: crea un objeto de la ClaseDos y llama al método "multiplica" pasando como parámetro un objeto de la ClaseUno que se crea en ese mismo momento.

Ejercicio carpeta 22_Clases_06_relacionar02_02

Otro ejemplo de relación de objetos pasándolos como parámetros.

En este caso crearemos menús de navegación.

La clase **Configuracion.php** recibe como parámetros el título (nombre), el link y el color de fondo de cada enlace del menú. El método mostrar() los muestra por pantalla.

La clase **Menu.php** recibe en el constructor un parámetro que indicará si el menú se va a mostrar en horizontal o en vertical.

El método insertar() añade a un array lo que recibe como parámetro que es un objeto de la clase Configuración. Por tanto tiene acceso a sus propiedades públicas y métodos.

Los métodos mostrarHorizontal() y mostrarVertical() muestran por pantalla los elementos del menú llamando al método de la clase Configuración mostrar(). En el caso de que el menú sea vertical simplemente añade un elemento de tipo
 entre cada enlace.

El método mostrar() comprueba el valor de la propiedad de la clase Menu que indica la dirección en la que se va a mostrar el menú y llama al método correspondiente.

El script **index.php** crea varios objetos de las clases Menu y Configuración con diferentes parámetros e invoca al método insertar() para añadir al menú un objeto de la clase Configuración y al método mostrar() de la clase Menú para mostrarlo por pantalla.

Ejercicio carpeta 22_Clases_06_relacionar02_03

Este ejemplo muestra cómo relacionar clases creando objetos de una en la definición de otra. Consiste en crear los diferentes componentes de una página web mediante clases.

Tenemos los siguientes ficheros:

Cabecera.php: este script asigna en el constructor a la propiedad que tiene definida el valor que le llega por parámetro. Tiene además, un método que lo muestra dentro de una etiqueta HTML de tipo h1.

Cuerpo.php: en esta clase se guardan en un array los elementos que le llegan como parámetro al método "insertar". El método "mostrar" recorres ese array mostrando cada elemento dentro de una etiqueta HTML de tipo párrafo.

Pie.php: construye el footer de la página

Pagina.php: en esta clase, el constructor recibe dos parámetros, con los que creará instancias de la cabecera, el cuerpo, y el pie de una página web.

Cuenta además con dos métodos, uno para insertar texto en el cuerpo de la página y el otro para mostrar por pantalla la página creada.

Index.php: crea un objeto de tipo página y hace la inserción de varios párrafos.

Parámetros opcionales en los métodos de una clase

- Un parámetro de un método es opcional, si en su declaración le asignamos un valor por defecto. Recuerda que los parámetros opcionales deben ir al final de la lista de argumentos.
- Si llamamos al método sin enviarle ese parámetro, tomará el valor por defecto.
- Se pueden utilizar tanto en programación estructurada como en POO.

Ejercicio carpeta 22_Clases_07_paramOpcionales

En el ejemplo tenemos una clase "Cabecera" definida en la que hay cuatro propiedades definidas. El constructor debe recibir cuatro parámetros, pero si no se indican el segundo, tercero y cuarto, tomarán los valores por defectos indicados en el método.

El método mostrar() saca por pantalla el contenido de las propiedades dentro de etiquetas HTML.

En el archivo index.php crea diferentes instancias de la clase "Cabecera" pasándole uno, dos tres o los cuatro parámetros.

HERENCIA

Recomendaciones

- Si la relación entre dos clases responde a una pregunta de tipo **"Clase A es un Clase B"**, es posible que el tipo de relación entre ellas sea de herencia.
- En cambio, si la relación entre dos clases responde a una pregunta de tipo **"Clase A tiene un Clase B"**, quizá en lugar de implementar una herencia es mejor declarar en la Clase A un atributo de la Clase B (relación de colaboración).

Ejercicio carpeta 22_Clases_08_herencia_01

En este ejemplo tenemos definida una clase Coche con una serie de propiedades y métodos.

Además hay una clase CochedeLujo que hereda de la clase Coche, que tiene definidos unos métodos que utilizan propiedades y métodos heredados de la clase padre.

En el script index.php creamos una instancia de la clase CocheDeLujo e invocamos a los métodos definidos en ella y también de forma directa a los métodos públicos de la clase padre.

Ejercicio carpeta 22_Clases_08_herencia_01B

Tenemos dos clases definidas, Clase1 y Clase2, que hereda de Clase1.

En ambas clases hay definidas varias propiedades con distinta visibilidad y algunas de ellas estáticas.

El método mostrar() se muestra cómo acceder a las constantes y variables de la propia clase o de la clase padre desde él.

El archivo index.php se crea un objeto de la Clase2 y llama al método mostrar() de la clase.

Ejercicio carpeta 22_Clases_08_herencia_02_sobrescribir

Este ejemplo muestra cómo se sobrescriben métodos de la clase padre en una clase hija y cómo hacer referencia a los métodos padre desde los métodos sobrescritos.

Ejercicio carpeta 22_Clases_08_herencia_02_sobrescribir

En este ejemplo se ha definido una clase Operación, que tiene tres propiedades de tipo protected que corresponden a dos números y un resultado. Además tiene definidos tres métodos, dos guardan en las propiedades de los números los valores que llegan por parámetro, y el otro, llamado operar, realiza la multiplicación de los números.

Tenemos también dos clases, Suma y Resta, que heredan de la clase Operación y que sobrescriben el método operar de la clase padre, sumando o restando los números respectivamente.

- En el script index.php se crean objetos de las tres clases y se llama desde cada uno de ellos al método operar, para comprobar que llama al método sobrescrito.

Clases abstractas

Ejercicio carpeta 22_Clases_09_abstract_01

Tenemos definida la clase abstracta VehiculoAbstracto, que tiene definida una propiedad, el método set de esa propiedad, y el método get, y dos métodos más de tipos abstracto.

Existen otras dos clases Audi y Moto que heredan de VehiculoAbstracto en la que se implementan los métodos definidos como abstractos en la clase abstracta.

En el index.php se crean objetos de la clase Audi y de la Clase Moto y se invocan a los métodos de la clase abstracta que se implementaron.

Ejercicio carpeta 22_Clases_09_abstract_02

Ejemplo de uso de una clase abstracta:

Fichero OperacionAritmetica

- Define una clase abstracta que tiene dos propiedades privadas.
- Los métodos definidos en la clase son:
    - El constructor con el mismo nombre que la clase.
    - Métodos para obtener y asignar los atributos (set y get correspondientes).
    - Define un método abstracto protected llamado "realizarOperacion" que deberá ser redefinido en las clases.

Fichero Sumar.php

- Incluye la clase Operacionaritmetica.
- Define la clase Sumar que hereda de la clase abstracta OperacionAritmetica
- No define ninguna propiedad.
- Define los siguientes métodos:
    - El constructor sumar().
    - Redefine el método "realizarOperacion" que realiza la suma de los dos atributos.

Fichero Restar.php

- Prácticamente igual que el fichero anterior, salvo que el método redefinido realiza la resta de los dos atributos.

Fichero index.php

- Incluye las clases sumar y restar.
- Crea dos objetos uno de cada clase y se asignan valores a sus atributos.
- Muestra en pantalla el resultado de invocar al método redefinido desde el objeto de cada clase.

Interfaces

- **En PHP5 no existe la herencia múltiple. Sin embargo, sí es posible crear clases que implementen varios interfaces, simplemente separando la lista de interfaces por comas después de la palabra implements.** La única restricción es que los nombres de los métodos que se deban implementar en los distintos interfaces no coincidan.

¿Interfaces o clases abstractas?

Una de las dudas más comunes en POO, es qué utilizar: interfaces o clases abstractas.

- Ambas permiten definir reglas para las clases que los implementen o hereden respectivamente. Y ninguna permite instanciar objetos. Las diferencias principales entre ambas opciones son:
    - En las clases abstractas, los métodos pueden contener código. En las interfaces no, habría que repetir el código en todas las clases que lo implemente.
    - Las clases abstractas pueden contener atributos, y los interfaces no.
    - No se puede crear una clase que herede de dos clases abstractas, pero sí se puede crear una clase que implemente varios interfaces.

Ejercicio carpeta 22_Clases_10_interfaces_01

Este ejemplo simula de forma muy básica el comportamiento de un coche.

En él tenemos una interfaz llamada Auto en la que se definen dos métodos arrancarMotor() y paraMotor() que deberán desarrollarse en las clases que implementen la interfaz.

Hay otra interface definida llamada Combustible que hereda de la clase Auto que define a su vez otros dos métodos, vaciarDeposito() y llenarDeposito(), que implementarán las clases que utilicen esa interfaz.

Además hay una clase definida llamada Deportivo que implementa la interfaz Combustible (y por tanto la clase Auto, ya que hereda de esta).

Esta clase ha de implementar los métodos definidos en las interfaces, además de los suyos propios.

- En el index.php se crea un objeto de la clase Deportivo y se van llamando a sus diferentes métodos.

Sobrecarga de propiedades y métodos

- En la mayoría de los lenguajes de programación orientados a objetos, la sobrecarga ofrece la posibilidad de tener varios métodos con el mismo nombre, pero con un tipo o número distinto de parámetros.
- PHP interpreta la sobrecarga de métodos y propiedades de forma distinta: permite crear de forma dinámica propiedades y métodos mediante el uso de los métodos mágicos.

- PHP invoca automáticamente a los métodos de sobrecarga cuando se intenta interactuar con métodos o propiedades que, o bien no han sido declarados (no existen), o bien no tenemos acceso a ellos por sus restricciones de visibilidad.
- Los métodos mágicos que se pueden emplear para la sobrecarga de propiedades son:
    - **__set()** se ejecuta al escribir datos sobre propiedades inaccesibles.
    - **__get()** se utiliza para consultar datos a partir de propiedades inaccesibles.
    - **__isset()** se lanza al llamar a isset() o a empty() sobre propiedades inaccesibles.
    - **__unset()** se invoca cuando se usa unset() sobre propiedades inaccesibles.
- Los métodos mágicos que se pueden emplear para la sobrecarga de métodos son:
    - **__call():** es lanzado al invocar un método inaccesible en un contexto de objeto.
    - **__callStatic():** es lanzado al invocar un método inaccesible en un contexto estático.

(entendemos como inaccesible al atributo o método que o no está definido, o al que no podemos acceder debido a sus restricciones de visibilidad)

Métodos mágicos

__get() y __set():

- Normalmente los atributos de una clase suelen tener una visibilidad restrictiva como protected o private por lo que es necesario tener definidos unos métodos get() y set() que nos permitan acceder a ellos
- Pero si el nº de atributos es muy grande, declarar por cada atributo un método de acceso (get) y otro de modificación (set) puede ser muy tedioso. Una solución está en la utilización de un IDE como Netbeans que automáticamente crea los 'getters/setters' básicos (ctrl + insert).

- La otra forma es utilizando los métodos mágicos. Con los métodos mágicos __set y __get podemos implementar un funcionamiento para poder modificar o acceder a los distintos atributos de la clase evitando tener que crear un método para cada uno de ellos.
- Una vez declarados estos métodos, si se intenta acceder a un atributo como si este fuera público, PHP llamara automáticamente a __get(). Y si asignamos un valor a un atributo, llamará a __set().
- Evidentemente, __set() necesita dos parámetros de entrada: nombre del atributo y el valor a asignar. Y __get() solo necesita el nombre del atributo a obtener su valor.
- Para usar esos métodos de forma automática han de estar definidos por el programador, (si no, no hacen nada), accedemos a propiedades privadas como si fueran públicas (desde una instancia de la clase invocando directamente a la propiedad).

Ejercicio carpeta 22_Clases_11_magicos_01_SetGet_01

Este ejemplo es una muestra del uso de los métodos mágicos __set() y _get().

Hay una clase definida con propiedades con visibilidad privada. Se definen los métodos mágicos __get() y _set() en los que se comprueba si existe en la clase la propiedad que reciben por parámetro. De ser así devuelven o asignan un nuevo valor respectivamente.

Por otra parte, en el otro script se crea un objeto de la clase y se accede con el operador flecha a las propiedades (como si fueran públicas). Esto es posible al tener definidos los métodos mágicos, PHP se encarga de llamarlo automáticamente. Si no, se produciría un error.

Ejercicio carpeta 22_Clases_11_magicos_01_SetGet_02

Este ejemplo es similar al anterior aunque, en este caso, en la definición del método mágico __set() lo que haremos es comprobar si existe en la clase la propiedad que se ha recibido como parámetro y si no existe se crea de forma dinámica (IMPORTANTE: la propiedad se crea en el objeto no en la clase).

Ejercicio carpeta 22_Clases_11_magicos_01_SetGet_03

En este ejemplo se utilizan los métodos set() y get() con un array en el que se van guardando las propiedades que se vayan creando en el objeto.

__clone():

- Anteriormente vimos que si queríamos hacer una copia independiente de un objeto debíamos usar **"clone"**.
- Pero puede darse el caso de que queramos hacer algún cambio en el objeto al clonarlo, limpiar algunas variables que contentan parámetros que ya no necesitemos, etcétera. Para este caso es que existe el **método mágico __clone()**.
- __clone() se dispara automáticamente (si está definido) cuando se ejecuta la instrucción **clone**.
- No recibe parámetros.

Ejercicio carpeta 22_Clases_11_magicos_02_clone

En este ejemplo se hace una copia del objeto utilizando el método mágico __clone() que es invocado automáticamente por PHP si el programador lo ha definido.

En la clase está definido _-clone(), en el que se incrementa el valor de una propiedad estática que contiene un número que corresponde con el número de objeto, y además cambia el nombre a la copia creada del objeto, añadiendo al nombre que tenía un guion y el nuevo número de instancia.

__isset() y __unset():

- __isset() se lanza cuando se llama a isset() o empty() sobre propiedades inaccesibles.
- __unset() se lanza cuando se usa unset() sobre propiedades inaccesibles.

Ejercicio carpeta 22_Clases_11_magicos_03_IssetUnset

Ejemplo de uso de los métodos mágicos **__isset()** y **__unset()**. Estos métodos se definen en la clase, y lo que hacen es ejecutar la función isset() o unset() sobre la propiedad recibida como parámetro (son propiedades privadas por lo que no se podría acceder a ellas desde una instancia de la clase si no es a través de objetos).

__toString():

- El método mágico **__toString()** es invocado automáticamente (si existe) cuando se realiza un echo o un print del objeto.
- Devuelve un string, si no se produce un error.

- De esta forma podemos, por ejemplo devolver el contenido de las propiedades del objeto como si fuera un string.

Ejercicio carpeta 22_Clases_11_magicos_04_toString

En este ejemplo se muestra el funcionamiento del método mágico __toString(). En primer lugar debemos definir el comportamiento que queramos que tenga en la clase. En ese caso, devuelve una cadena de caracteres con el valor de las propiedades "nombre" y "email".

Cuando creemos un objeto de la clase, y hagamos un "echo" o un "print" de él, PHP llamará al método __toString() definido en la clase, y mostrará la cadena que habíamos definido que retornara.

__call() y _callStatic():

- Otros métodos mágicos muy interesantes son **__call() y __callStatic()** que capturan llamadas a métodos que no están implementados en la clase.
- __call() se lanza cuando se invoca a un método inaccesible en un contexto de objeto.
- __callStatic() se lanza cuando se invoca a un método inaccesible en un contexto estático (es decir, si el método al que se quiere llamar es estático).
- Estos métodos reciben dos parámetros, el primero es el nombre del método al que se va a llamar, y el segundo es un array con todos los argumentos que hay que pasarle a ese método.
- En función del nombre del método y del número de parámetros que se pasen, se podrían realizar unas acciones u otras.

Ejercicio carpeta 22_Clases_11_magicos_05_call

- Tenemos una clase Cajero definida con sus propiedades de tipo privadas.
- En la clase están definidos los métodos __construct() y __destruct() que serán ejecutados al crear y destruir el objeto respectivamente.
- También se han implementado las funcionalidades para los métodos __call() y __callStatic() que se ejecutarán uno u otro en función del contexto de la llamada (objeto o estático).
- Ambos reciben dos parámetros, el nombre del método a ejecutar y un array con los parámetros enviados.
- En este ejemplo, los métodos simplemente mostrarán el nombre del método y los parámetros recibidos.

Ejercicio carpeta 22_Clases_11_magicos_05_call

- En este ejercicio vamos a ver cómo usar el método mágico __call(). Este método mágico es llamado automáticamente cuando se intenta llamar a un

método que no está definido en la clase o es inaccesible dentro del objeto (por ejemplo un método privado).

- Este método recibe dos parámetros, uno es el nombre del método al que se intenta invocar, y el otro los parámetros que le hemos pasado al método al que hemos intentado llamar.
- Este método recibe dos argumentos, el primero es el método al que se va a llamar y el segundo es un array con todos los argumentos que hay que pasarle a ese método.
- En el ejemplo vamos a crear un objeto de tipo Persona y vamos a intentar acceder a métodos que no son de la clase Persona sino de la clase Direccion, que no es una clase de la que hereda Persona. Son independientes.
- En la clase Persona se crea un objeto de la clase dirección. Así los métodos de esta clase podrían acceder a los métodos de la clase Dirección, pero desde otro sitio fuera de la clase no se podría acceder ya que no estamos usando extend.
- Se podría hacer definiendo en la clase Persona para cada método de la clase Dirección un método que llame al método de la clase Dirección.
- Con el método mágico __call (), con una sola llamada podemos invocar a todos los métodos de la clase Dirección desde otras clases. Esto facilita la reutilización del código que es tan importante en programación.

Fichero index.php

- Incluye el archivo de la clase Persona.
- Crea un objeto de la clase Persona.
- Asigna valores a las propiedades nombre, ciudad y país.
- Muestra el contenido de esas propiedades utilizando los métodos correspondientes.

Fichero Direccion.php

- Define una clase, Dirección.
- Contiene dos atributos protected, ciudad y país.
- Contiene los métodos get() y set() correspondientes que asignan o recuperan valores de los atributos de la clase,

Fichero Persona.php

- Incluye la clase Direccion.php
- Define la clase Persona.
- Contiene dos atributos de tipo protected: nombre y dirección.
- Define los siguientes métodos:

- Constructor: utiliza el método mágico __construct(). En este método se crea un nuevo objeto de la clase "Direccion" y se guarda en la propiedad dirección de la clase Persona.
- Métodos setNombre() y getNombre() que obtienen o asignan, respectivamente valores a la propiedad nombre de la clase Persona.
- Define el método mágico __call(). En primer lugar comprueba si el método que queremos usar está en la clase a la que queremos acceder, en nuestro caso la clase Dirección, usando para ello el método **method_exists()** al que le pasamos como argumento la propiedad dirección de la clase Persona. Si es así, procederemos a llamar al método. devuelve el resultado de llamar al método de la clase Dirección pasándole los parámetros correspondientes.
  La función PHP call_user_func_array() llama a una función que indicamos en el primer parámetro, y en el segundo parámetro recibe un array de argumentos que contiene todos los argumentos que debe recibir la función. La función call_user_func_array() devuelve el resultado de ejecutar la función que hemos indicado.
  En nuestro ejemplo hay métodos que reciben argumentos y otros no. Utilizando esta función no tenemos que indicar exactamente los argumentos. Nos llega en un array.
  Por otra parte en nuestro caso, estamos intentando llamar al método de un objeto en lugar de llamar a una función, la sintaxis para hacerlo es pasar en el primer parámetro de la función un array con dos elementos, el objeto (con las propiedades que tiene) y el método a llamar.

Enlaces con ejemplos Clases
www.tutorialesprogramacionya.com

https://www.w3schools.com/php/php_oop_inheritance.asp

PROGRAMACIÓN EN CAPAS

Intenta separar la lógica de la aplicación de la presentación.

El patrón de diseño web MVC, conocido como arquitectura MVC está compuesto por tres niveles:

- El **Modelo**: Es la representación de la información con la cual el sistema opera, por lo tanto **gestiona todos los accesos a dicha información**, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica

de negocio). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'.

- El **Controlador**: **Responde a eventos** (usualmente acciones del usuario) **e invoca peticiones al 'modelo'** cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede **enviar comandos a su 'vista' asociada** si se solicita un cambio en la forma en que se presenta el 'modelo' (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el 'controlador' hace de **intermediario entre la 'vista' y el 'modelo'**.
- La **Vista**: **Presenta el 'modelo'** (información y *lógica de negocio*) **en un formato adecuado** para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho 'modelo' la información que debe representar como salida.

IMPORTANTE: La utilización MVC requiere una buen planificación de la aplicación, por lo que de cara al examen si hay algún ejercicio relacionado con NMVC será en su forma más básica, es decir que exista un controlador que pida datos a un modelo y que genere una vista, sin necesidad de generar de forma dinámica todos los componentes.

En la plataforma os he dejado un par de ejemplos de programación siguiendo el patrón MVC, uno mucho más sencillo que el otro.

- En el primero, simplemente se muestra como se separan la lógica del programa de la presentación teniendo un archivo controlador, un modelo y dos vistas. El funcionamiento básicamente consiste en que al ejecutar el controlador por primera vez se mostrará una vista que consiste en un formulario. Al pulsar el botón del formulario se vuelve a ejecutar el controlador y al comprobar si ha legado algo por $_REQUEST muestra una vista diferente.
- El segundo ejemplo es más complicado de entender, pero muy interesante y que refleja la forma ideal de uso de este patrón. En él se ven creando de forma dinámica los controladores e invocando a los métodos de los controladores (cuyo nombre también se forma de manera dinámica) en función de las elecciones que vaya realizando el usuario.

Se utiliza en él además, el patrón “Singleton” que se encarga de garantizar que solo haya una instancia de un objeto. Os adjunto un par de enlaces donde viene explicado.

Ejemplo de la plataforma con explicación:

http://www.juntadeandalucia.es/servicios/madeja/sites/default/files/historico/1.4.0/contenido-recurso-257.html

Podéis ver el concepto de Singleton en:

http://pabletoreto.blogspot.com/2015/04/ejemplo.html

http://pabletoreto.blogspot.com/2015/04/php-pdo-singleton-crud.html

REALIZAR LA TAREA DE LA UNIDAD CON SMARTY

En vista de que os ha dado problemas a varios, en este apartado comentaré la configuración que tengo en mi ordenador (en el que me funciona la tarea con smarty). No obstante como comenté en el foro podéis entregar la tarea sin utilizar smarty siempre y cuando apliquéis el patrón MVC.

También os adjunto la carpeta smarty que yo tengo.

Pasos:

- Copiar la carpeta smarty que os adjunto con los ejemplos a la ruta donde tengáis la carpeta de xampp (en mi caso c:\xampp)
- Editamos el archivo php.ini y buscamos la directiva **include_path** en mi caso yo tengo Windows. Quito el punto y coma del principio si lo tiene (indica que está comentado) y añado al final, separado por un punto y coma la ruta absoluta a la carpeta libs de smarty y reiniciamos el servidor. En mi caso:
  ; Windows: "\path1;\path2"
  include_path = ".;c:\php\includes**;c:\xampp\smarty\libs**"
- Creamos un nuevo proyecto en NetBeans, y en ella crearemos 3 carpetas más configs, templates y templates_c. dentro de la carpeta templates ponemos las páginas html (o las tpl) que es donde va el código html y utilizaremos variables php con la sintaxis smarty, por ejemplo {$nombre}
- templates_c es un directorio temporal donde se crean dinámicamente los nuevos html con las variables que hemos usado cargadas.

- Para que lo ejecute hay que poner en propiedades del proyecto, run configuration la carpeta “smarty” entre localhost y el directorio. Si no, no lo ejecuta correctamente en el navegador.
- En el Código php que va en la raíz del proyecto, debemos incluir la clase Smarty, crear una instancia de la misma y añadir las rutas de las carpetas de nuestra plantilla. Después nuestro código php, y por último debemos asignar las variables que serán usadas en el archivo “tpl o hmtl. Por último imprimimos por pantalla la plantilla (index.html).
- El fichero login quedaría algo así:

```php
<?php
session_start();
require_once('include/DB.php');

// Cargamos la librería de Smarty
require_once('Smarty.class.php');
$smarty = new Smarty();
$smarty->template_dir = 'templates';
$smarty->compile_dir = 'templates_c';
$smarty->config_dir = 'configs';
$smarty->cache_dir = 'cache';

//se añade la siguiente instrucción, porque sio no da error al ejecutar el archivo.
$smarty->assign('error','');
// Comprobamos si ya se ha enviado el formulario
if (isset($_POST['enviar'])) {

    if (empty($_POST['usuario']) || empty($_POST['password']))
        $smarty->assign('error','Debes introducir un nombre de usuario y una contraseña');
    else {
        // Comprobamos las credenciales con la base de datos
        if (DB::verificaCliente($_POST['usuario'], $_POST['password'])) {
            $_SESSION['usuario']=$_POST['usuario'];
            header("Location: productos.php");
        }
        else {
            // Si las credenciales no son válidas, se vuelven a pedir
            $smarty->assign('error','Usuario o contraseña no válidos!');
        }
    }
}

// Mostramos la plantilla
$smarty->display('login.tpl');
?>
```