UNIDAD DIDÁCTICA 5:

POO CON PHP

Módulo profesional:

Desarrollo Web en Entorno Servidor

Duración: 160 Horas.

Código: 0613

Ciclo Formativo de Grado Superior:

Desarrollo Aplicaciones Web

Contenido

Resumen Introductorio

En esta unidad nos introduciremos en el mundo de la Programación Orientada a Objetos dentro del lenguaje PHP.

Igual que ocurre en otros lenguajes, resulta imprescindible programar de una forma orientada a objetos por muchos motivos, pero los principales los podemos resumir en el mantenimiento futuro del código, escalabilidad e inteligibilidad de nuestras aplicaciones.

Introducción

El lenguaje PHP ha sido y continua siendo un lenguaje típicamente procedimental y en el que pocos desarrolladores invierten esfuerzos en un desarrollo orientado a objetos. Esto es debido principalmente a la característica de compatibilidad hacia atrás del lenguaje para poder permitir este tipo de programación.

Sin embargo a partir de la versión 5 (y en la actualidad estamos en la versión 7, tras la fallida versión 6) el cambio en este aspecto fue espectacular ya que la transformación del lenguaje y la incorporación del paradigma Orientado a Objetos es una realidad que hizo que evolucionase el lenguaje y aparecieran por ejemplo frameworks ampliamente utilizados hoy en día como Symfony.

Veremos en esta unidad como desarrollar código Orientado a Objetos mediante PHP y aprenderemos la sintaxis de definición y uso.

Caso Introductorio

Queremos definir clases que nos permitan datos de jugadores de un equipo de baloncesto y además que sea suficientemente mantenible y flexible para poder ampliarlo en un futuro.

Desarrollo:

1. Programación orientada a objetos

Uno de los cambios más espectaculares y maravillosos que sufrió PHP es adaptarse a los lenguajes de programación modernos incluyendo dentro de su sintaxis el paradigma de Programación Orientada a Objetos.

Libro recomendado, PHP 7 Programming CookBook (Bierer, 2016) En el capítulo 4 encontraremos una fantástica referencia a la programación orientada a objetos con PHP7 http://www.hackingwithphp.com/5/0/0/arrays	

	***POO en PHP*** http://www.hackingwithphp.com/6/0/0/objects Antes de que llegara PHP 5, el soporte de OOP en PHP era bastante escaso. Como resultado, los pocos que lo usaban a menudo lamentaban la elección, y no es sorprendente que todo el sistema tuviera una reescritura completa en PHP 5 - ahora es mucho más avanzado y flexible, y debería agradar a casi todo el mundo.

Muchos desarrolladores en PHP aún siguen programando con una metodología muy secuencial y que por lo tanto les provoca muchos problemas cuando se enfrentan al mantenimiento y actualización de su código. Comenzaremos con nuestras explicaciones, evidentemente explicando qué es un objeto.

1.1. ¿Qué es un objeto?

Además de la definición que hemos planteado anteriormente, un objeto lo podemos entender como la representación mínima de cualquier resolución a un problema que tengamos a nivel de software. Es difícil dar una definición sin aterrizarlo con un ejemplo de código, pero en breve comenzaremos a plantear ejemplos.

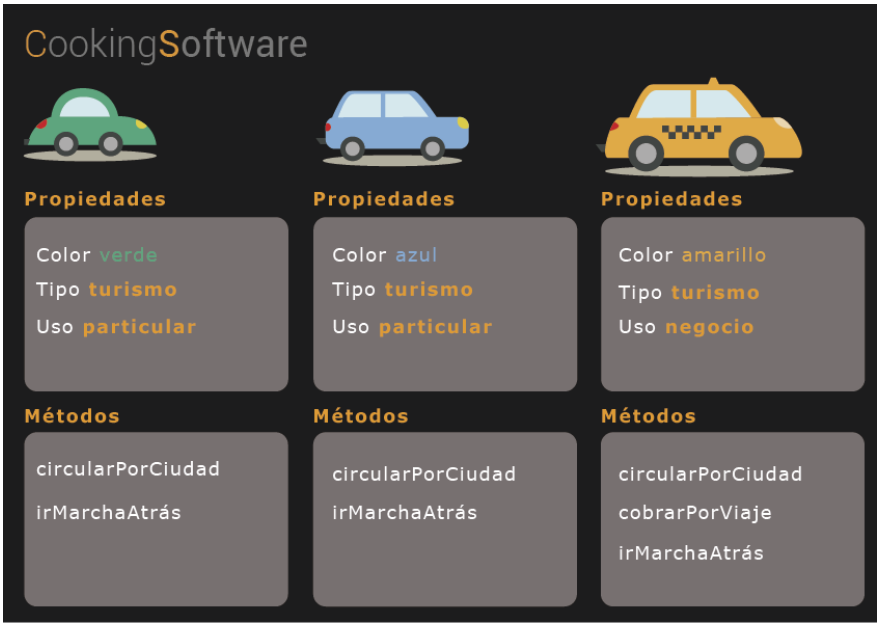


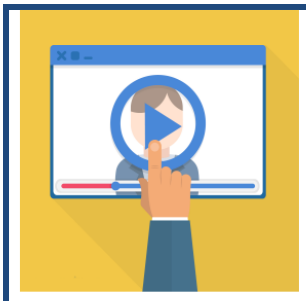
Imagen 1: Ejemplo objetos

Fuente de la imagen: propia

Como observamos en el ejemplo, un objeto tiene dos partes claramente diferenciadas:

- Propiedades/atributos, que hace referencia a las características del objeto
- Métodos/funciones, que hace referencia a las acciones que puede desarrollar.

Veamos la explicación completa a través de un vídeo:



En el vídeo que encontrarás una introducción a los objetos

https://youtu.be/B6eIVywYFcQ

1.2. POO en PHP

Clásicamente y de manera tradicional, cuando alguien se plantea programar con el lenguaje PHP nunca se plantea hacerlo de una manera orientada a objetos. Esto es así por varios motivos:

- El cambio se produjo a partir de la versión 5, donde el lenguaje produce un salto enorme para la introducción de la programación orientada a objetos.
- Compatibilidad hacia atrás es uno de los mayores retos de cualquier lenguaje de programación y en particular el de PHP, un lenguaje muy maduro donde infinidad de proyectos estaban escritos en PHP, por lo que avanzar hacia una nueva versión no debía ser inconveniente para mantener los antiguos proyectos, y más desde la filosofía de PHP.
- Los cambios de versiones en PHP no se dan con la misma velocidad que en otros lenguajes, y donde podemos tener versiones de lenguajes durante 5 o más años debido a la intencionada necesidad de mantener compatibilidades con proyectos anteriores.

La introducción de la orientación a objetos con PHP hace que podamos y debamos pensar en este paradigma para poder diseñar y programar nuestras aplicaciones con la arquitectura Orientada a Objetos. Pero no debemos dejar de lado que la propia arquitectura web nos permite programar código no orientado a objetos y en muchos casos nos impide pensar en una arquitectura 100% POO.

De aquí en adelante intentaremos:

- Aprovechar la potencia de la arquitectura POO para diseñar y desarrollar todas nuestras librerías de una forma más abstracta y que de esa forma sea mantenible en el tiempo y flexible.
- Aprovechar las opciones de desarrollo secuencial allá donde no haya más remedio (en las vistas principalmente), pero no por elo llegando al extremo de generar "spagheti code".

1.3. Definición de una Clase

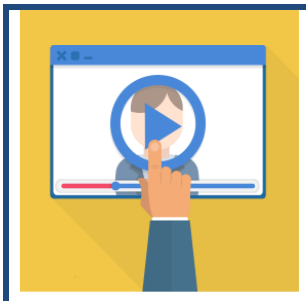
El primer paso en este mundo apasionante de la programación orientada a objetos es la creación de un objeto valga la redundancia mediante el uso de las clases.

Clases
http://php.net/manual/es/language.oop5.basic.php

La definición básica de una clase comienza con la palabra reservada ***class***, seguida de un nombre de clase, y continuando con un par de llaves que encierran las definiciones de las propiedades y métodos pertenecientes a dicha clase.

El nombre de clase puede ser cualquier etiqueta válida, siempre que no sea una palabra reservada de PHP. Un nombre válido de clase comienza con una letra o un guión bajo, seguido de una cantidad arbitraria de letras, números o guiones bajos. Como expresión regular, se expresaría de la siguiente forma: ^[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*$.

Veámoslo a través de un ejemplo:



En el vídeo que encontrarás una introducción a los objetos

https://youtu.be/jyX91bQsFSI

El código utilizado es el siguiente:

Ejemplo creación de una clase

```php
<?php
/**
 * Clase creada por Paco Gómez
 * clase ejemplo de definicion
 */
class ClaseCoche
{

  // Declaración de una propiedad
  public $color = 'Verde';
  public $tipo = 'turismo';

  // Declaración de un método
  public function mostrarColor() {
       echo '<br>';
       echo 'El color del coche es:';
       echo $this->color;
       echo '<br>';
  }
  public function getColor(){
    return $this->color;
  }
  public function mostrarTipo() {
       echo $this->tipo;
  }
}

?>
```

Qué tenemos en el anterior código

1. Una clase definida a través de la palabra reservada class
2. El nombre de la clase se denomina ClaseCoche
3. Esta clase tiene dos propiedades denominadas color y tipo

4. Además tiene 3 métodos denominados mostrarColor, getColor y mostrarTipo

	Tras iniciarse en la definición de clases, vamos a generar una primera clase que nos modele un equipo de baloncesto: 1) Generamos la estructura de ficheros dentro con una nueva carpeta y un documento php denominado equipo.php 2) Crearemos una clase utilizando la palabra reservada class, y denominando a la clase Equipo 3) Definiremos como única propiedad el nombre de equipo.

Qué pasaría si hubiéramos definido nuestro código sin ningún tipo de clase, este se podría ver de la siguiente forma:

Comparativo de código sin hacer uso de POO

```php
<?php
$color = 'Verde';
$tipo = 'turismo';

function mostrarColor() {
      echo '<br>';
      echo 'El color del coche es:';
      echo $color;
      echo '<br>';
  }
function getColor(){
    return $color;
}

?>
```

Aquí no hay forma de agrupar qué información y cuáles acciones van juntas. También tenemos otros problemas:

- No es posible prevenir que el programador cambie sin el color o el tipo del vehículo. De hecho, y dado que PHP no es un lenguaje tipificado, cualquier programador que utilice nuestro código podría cambiar el tipo a uno diferente.
- Siempre tendremos que recordar qué argumentos pasarle a cada función y en cual orden y esto se vuelve difícil si una función depende de 3 o más argumentos.
- Si quisiéramos crear un segundo vehículo entonces ¿Tendríamos que crear las variables $color2, $tipo2? Y si queremos un tercer vehículo.

Con las clases y con la POO resolvemos los anteriores problemas, pero es más, ganamos en abstracción, mantenimiento, seguridad y legibilidad de código.

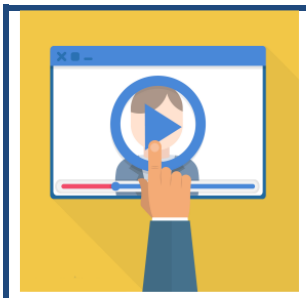
1.4. Include contra Require

Antes de comenzar a crear objetos debemos saber cómo podemos:

- Dividir el código en múltiples ficheros
- Cómo utilizar dicho código desde un fichero

En todos los lenguajes de programación modernos existe el concepto de librería, y por lo tanto de la utilización o inclusión de estas librerías en otras partes de nuestro código. De esta forma hacemos más eficiente el uso de código y además hacemos presente la denominada "reutilización" de código.

Veámoslo a través de un ejemplo:



En el vídeo que encontrarás un ejemplo de *include y require*

https://youtu.be/cg1oSluIY9s

En el siguiente enlace tienes el código utilizado:

https://github.com/pacogomezarnal/codigoEjemploCursoPHP/tree/master/pruebas

Como hemos visto en el ejemplo, dos son los nuevos conceptos aprendidos, pudiendo encontrar más documentación en la referencia de php.net:

Include
http://php.net/manual/es/function.include.php

La sentencia include incluye y evalúa el archivo especificado.
La siguiente documentación también se aplica a require.
Los archivos son incluidos con base en la ruta de acceso dada o, si ninguna es dada, el include_path especificado. Si el archivo no se encuentra en el include_path, include finalmente verificará en el propio directorio del script que hace el llamado y en el directorio de trabajo actual, antes de fallar. El constructor include emitirá una advertencia si no puede encontrar un archivo, éste es un comportamiento diferente al de require, el cual emitirá un error fatal..

El ejemplo básico además del anterior propuesto, podría ser:

Ejemplo de include

vars.php

```php
<?php

$color = 'verde';
$fruta = 'manzana';
```

```
?>

test.php
<?php

echo "Una $fruta $color"; // Una

include 'vars.php';

echo "Una $fruta $color"; // Una manzana verde

?>
```

Como vemos, la función *include* nos permite incluir el código definido en otro fichero, lo cual nos permite por lo tanto dividir nuestro código mediante librerías reutilizables a lo largo de nuestra aplicación.

Las cuatro estructuras que disponemos para poder incluir código desde otro fichero serían:

- **include**, incluye y evalúa el archivo especificado.
- **include_once**, la sentencia include_once incluye y evalúa el fichero especificado durante la ejecución del script. Tiene un comportamiento similar al de la sentencia include, siendo la única diferencia de que si el código del fichero ya ha sido incluido, no se volverá a incluir, e include_once devolverá TRUE. Como su nombre indica, el fichero será incluido solamente una vez.
- **require**, require es idéntico a include excepto que en caso de fallo producirá un error fatal de nivel E_COMPILE_ERROR. En otras palabras, éste detiene el script mientras que include sólo emitirá una advertencia (E_WARNING) lo cual permite continuar el script.
- **require_once**, la sentencia require_once es idéntica a require excepto que PHP verificará si el archivo ya ha sido incluido y si es así, no se incluye (require) de nuevo.

COMPRUEBA LO QUE SABES:

Una vez estudiado cómo definir una clase y como utilizarla, ¿sabrías demostrar con un ejemplo las diferencias entre require y require_once?
Coméntalo en el foro.

1.5. Creación de objetos

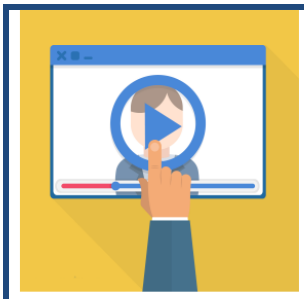
Una vez que hemos creado nuestra clase, estamos preparados para poder usar y utilizar las clases. Para ello veámoslo a través de un ejemplo y un vídeo:

Creación de un objeto

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title></title>
  </head>
  <body>
    <?php
      include 'ClaseCoche.php';

      $coche1 = new ClaseCoche();
      $coche1->getColor();
      $coche1->mostrarColor();

      $coche2 = new ClaseCoche();
      $coche2->mostrarColor();
      $coche2->mostrarTipo();
    ?>
  </body>
</html>
```



En el vídeo que encontrarás un ejemplo de uso y creación de objetos

https://youtu.be/aVw534tLPc0

Mientras que la clase representa el concepto genérico, un objeto es la representación específica de una clase. Cuando tenemos una clase podemos declarar uno o más objetos de dicha clase de esta forma:

Creación de un objeto

```php
$coche1 = new ClaseCoche();

$coche2 = new ClaseCoche();
```

Continuaremos con el ejemplo del equipo de baloncesto antes definido:

1) Dentro de la estructura de ficheros antes generada, crearemos un segundo fichero denominado liga.php
2) Incluiremos la clase equipo.php mediante un require
3) Crearemos dos equipos mediante la palabra reservada new, con dos nuevos objetos que se denominen VLCBasket y RMDBasket

new

http://php.net/manual/es/language.oop5.basic.php

Para crear una instancia de una clase, se debe emplear la palabra reservada new.

2. Características del POO en PHP

Una vez que ya conocemos las bases de la POO dentro del lenguaje PHP, avanzaremos en conceptos esenciales y que nos permitirá comenzar a usar los objetos y clases de una forma más intensiva para nuestras aplicaciones.

Libro recomendado, Hacking with PHP (Hudson, 2015) En el capítulo 6 encontramos la referencia a ejemplos y documentación sobre los objetos http://www.hackingwithphp.com/6/0/0/objects	

	Interesantísimo artículo sobre las diferencias de rendimiento entre las versiones 5 y 7 de PHP. http://www.aerospike.com/blog/php7_php5/

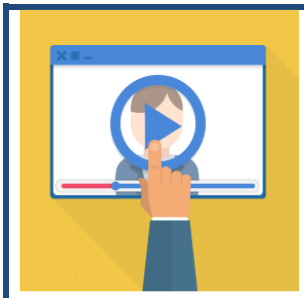
2.1. Propiedades o atributos

Las variables pertenecientes a una clase se llaman "propiedades". También se les puede llamar usando otros términos como "atributos" o "campos", pero para los propósitos de esta referencia se va a utilizar "propiedades". Ya hemos trabajado las variables en unidades anteriores y por lo tanto no nos va a resultar complicado entender este concepto y usarlo dentro de las clases.

RECUERDA…

En PHP las variables se representan con un signo de dólar seguido por el nombre de la variable. El nombre de la variable es sensible a minúsculas y mayúsculas.

Veamos ejemplo y comparativos de uso entre las variables y los atributos:



En el vídeo que encontrarás un ejemplo de uso de los atributos

https://youtu.be/OBTj8M0NGgk

En el siguiente enlace tienes el código utilizado:

https://github.com/pacogomezarnal/codigoEjemploCursoPHP/tree/master/clases

Vamos a copiar un código directamente de la referencia de php.net y lo vamos a analizar:

Ejemplos de atributos

```php
<?php
class ClaseSencilla
{
    // Válido a partir de PHP 5.6.0:
    public $var1 = 'hola ' . 'mundo';
    // Válido a partir de PHP 5.3.0:
    public $var2 = <<<EOD
hola mundo
EOD;
    // Válido a partir de PHP 5.6.0:
    public $var3 = 1+2;
```

```php
    // Declaraciones de propiedades inválidas:
    public $var4 = self::miMétodoEstático();
    public $var5 = $myVar;

    // Declaraciones de propiedades válidas:
    public $var6 = miConstante;
    public $var7 = array(true, false);

    // Válido a partir de PHP 5.3.0:
    public $var8 = <<<'EOD'
hola mundo
EOD;
}
?>
```

Podemos ver múltiples formas de definir diferentes atributos dentro de una clase, aunque la más cómun y habitual es la típica estructura *public $variable=0;*

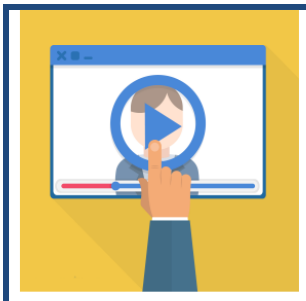
COMPRUEBA LO QUE SABES:

Una vez estudiado cómo definir una clase y como definir las propiedades, ¿serías capaz de definir una clase y sus propiedades para modelar una bicicleta? Al menos deberías definir una propiedad de tipo numérica, de carácter y booleana.
Coméntalo en el foro.

2.2. Funciones o métodos

El otro gran apartado dentro de la generación de clases y objetos en PHP es la definición de funciones.

Nuevamente tenemos un ejemplo para ver el uso y definición de las funciones:



En el vídeo que encontrarás un ejemplo de la creación de funciones

https://youtu.be/nSx297f0XRs

En el siguiente enlace tienes el código utilizado:

https://github.com/pacogomezarnal/codigoEjemploCursoPHP/tree/master/clases

Una función, esté definida o no dentro de una clase, tiene la siguiente estructura:

Funciones

```php
<?php
function foo($arg_1, $arg_2, /* ..., */ $arg_n)
{
    echo "Función de ejemplo.\n";
    return $valor_devuelto;
}
?>
```

Como podemos observar consta de:

- La palabra reservada *function*
- El nombre de la función definida por el usuario (en este caso se denomina *foo*)
- Los argumentos de entrada de parámetros, los cuales pueden ser tantos como se necesiten o se requieran para el buen funcionamiento de la función.
- El cuerpo de la función
- La devolución del resultado mediante la palabra reservada *return*

Características también diferenciales que observamos en la creación y utilización de las funciones en php:

- No necesitamos identificar el tipo de los argumentos de entrada, al igual que el resto de variables dentro del lenguaje y tal y como hemos descrito anteriormente, PHP es un lenguaje muy poco tipificado.
- No necesitamos indicar si la función devuelve o no valores o resultados, por lo que también será opcional el uso de *return*.

	Ampliaremos nuestro ejemplo de equipo de baloncesto añadiendo las siguientes propiedades y métodos a la clase: 1) Utilizaremos la clase antes definida equipo.php. 2) Añadiremos una nueva propiedad de tipo público denominado posición. 3) Añadiremos una nueva función denominada mostrarEquipo y que realice un echo de la propiedad nombre. 4) Añadiremos una nueva función denominada ponerEquipo y que modifique la propiedad nombre de la clase a partir del parámetro introducido. 5) Por último, dentro de nuestro fichero liga.php, usaremos la función ponerEquipo para definir el nombre del equipo y mostrarEquipo para mostrarlo.

2.3. Diagrama de clases

Una manera reconocida y estándar de representar las clases y sus relaciones es mediante los diagramas UML de clases.

Tal y como hemos explicado al inicio de esta Unidad Didáctica, y como se aprendió en Programación de primer curso, si echamos un vistazo a nuestro alrededor todo lo podemos representar como un objeto, que contiene propiedades, características y métodos.

Desde el punto de vista del desarrollador, necesitamos analizar las problemáticas que queremos resolver, dividir el problema y diseñar la solución con los recursos disponibles.

Los diagramas de clases colaboran en ese análisis del problema, permitiendo poder tener un lenguaje común entre cliente, desarrollador, compañero de trabajo y demás personas involucradas en un proyecto.

No es el objetivo en este módulo estudiar cómo representar una aplicación y el diagrama de clases, pero es interesante introducir o explicar cómo se representa mediante este estándar una clase ya que podremos utilizarlo de aquí en adelante para representarlas. Veamos la siguiente imagen:

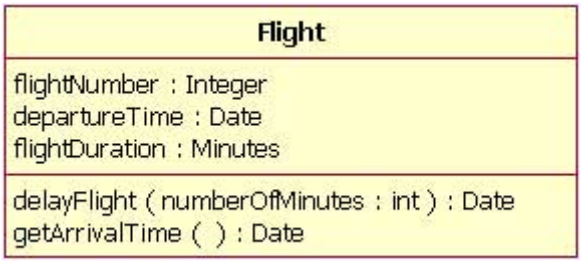


Imagen 1: Clase

Fuente de la imagen: https://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/

Como podemos ver en la imagen, tenemos tres partes claramente representadas imprescindibles de una clase:

- El nombre de la clase, ***Flight***
- Tres atributos de la misma, ***flightNumber, departureTime y flightDuration***. En el caso en el que hemos cogido el ejemplo, los

tipos utilizados son de otro lenguaje de programación, pero nos sirve para visualizar cómo representar los atributos.

- Dos métodos, ***delayFlight y getArrivalTime***, ambos con sus argumentos de entrada y salida.

Por lo tanto será un buen mecanismo gráfico que utilizaremos de aquí en adelante.

2.4. $this

Esta pseudovariable se utiliza dentro de cualquier clase/objeto para hacer referencia a la propia clase (decimos clase ya que está definida en la clase, pero sólo tiene sentido cuando se crea el objeto). Veamos la definición que hace php.net y después aterricémosla con varios ejemplos:

this
http://php.net/manual/es/language.oop5.basic.php

La pseudovariable $this está disponible cuando un método es invocado dentro del contexto de un objeto. $this es una referencia al objeto invocador (usualmente el objeto al cual el método pertenece, aunque puede que sea otro objeto si el método es llamado estáticamente desde el contexto de un objeto secundario).

Un ejemplo sobre la pseudovariable $this:

Ejemplo

```php
<?php
class ClaseSencilla
{
    // Declaración de una propiedad
    public $var = 'un valor predeterminado';

    // Declaración de un método
    public function mostrarVar() {
        echo $this->var;
    }
}
?>
```

Como se puede observar en el ejemplo:

- Tenemos una clase denominada *ClaseSencilla*
- Dentro de la clase tenemos definido un único atributo denominado *$var*
- Para poder acceder a dicha variable a lo largo de la clase, y por supuesto dentro de cualquier función, debemos utilizar la pseudovariable denominada *$this->var*

Como vemos será $this quien tenga el $, como símbolo de referencia a una variable, mientras que le sigue el símbolo -> para poder acceder a la variable dentro de la clase a la cual queremos acceder.

Muy muy importante es tener en cuenta que si dentro de una clase utilizamos $var no accederemos al mismo atributo definido dentro de la clase, y por lo tanto es imprescindible utilizar $this->var. Veámoslo con un ejemplo:

Ejemplo

```
<?php
class ClaseSencilla
{
    // Declaración de una propiedad
    public $var = 3;

    // Declaración de un método
    public function mostrarVar() {
        echo $this->var; //Acceso correcto al atributo var. Por
pantalla se mostrará 3
    }

    // Declaración de un método
    public function mostrarVar2() {
        echo $var; //Acceso INCORRECTO al atributo var. Por pantalla
no se mostrará nada
    }
}
?>
```

COMPRUEBA LO QUE SABES:

Una vez estudiado cómo definir una clase y como definir las propiedades, ¿serías capaz de provocar un error en la utilización de una propiedad al no utilizar el modificador $this?

Coméntalo en el foro.

Muy importante también será tener en cuenta que es un error de escritura o sintaxis escribir $this->$var.

Tarea 1
Título: Objetos. Creación

2.5. Ámbito de las variables

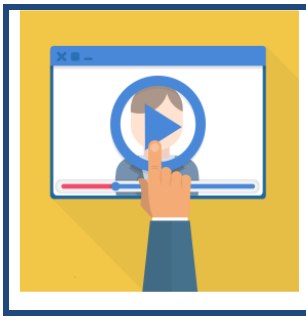
Dentro de la Programación Orientada a Objetos, el ámbito y la visibilidad lo podríamos interpretar cómo vamos a organizar nuestro código, nuestras variables y métodos y de qué forma va a ser accesibles por el resto de código o ficheros.

De acuerdo a la propia documentación de php.net, el ámbito de una variable es el contexto dentro del que la variable está definida. La mayor parte de las variables PHP sólo tienen un ámbito simple. Este ámbito simple también abarca los ficheros incluídos y los requeridos. Veamos esta definición con un ejemplo:

Ejemplo

```
<?php
  //variable de calculo de areas
    $area=0;
    $lado=5;
  ?>
  <div class="resultado">
    <?php
      //area del cuadrado
      $area=$lado*$lado;
    ?>
```

```
<?="El resultado del area es: ".$area?>
```



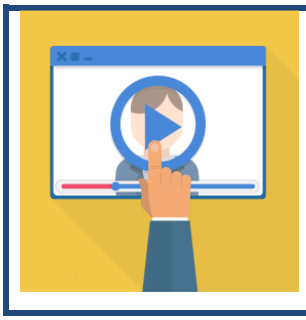
En el vídeo que encontrarás la explicación del anterior código

https://youtu.be/F22cBrrLSR0

El anterior ejemplo o código se modifica cuando utilizamos funciones dentro del código. No hemos profundizado en la utilización de funciones en php fuera de las clases y objetos ya que es una programación mucho más antigua y a extinguir, pero que nos sirve para ilustrar el ámbito de las variables dentro de php.

Ejemplo

```
<?php
    //variable de calculo de areas
      $area=0;
      $lado=5;
    //Definimos el calculo de areas
      function calcularAreaCuadrado(){
        $area=10;
        echo "El area dentro de la funcion es: ".$area."<br>";
      }
      calcularAreaCuadrado();
      echo "El area fuera de la funcion es: ".$area;
    ?>
```



En el vídeo que encontrarás la explicación del anterior código

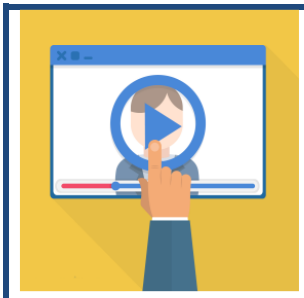
https://youtu.be/h9w2-UYwrdM

2.6. Visibilidad dentro de las clases

La visibilidad nos permite definir dónde y quien puede usar un atributo o un método. La visibilidad es un concepto dentro de orientación a objetos que permite emplear o "ocultar" varaibles y funciones de objetos dependiendo de las necesidades del desarrollo.

visibilidad
http://php.net/manual/es/language.oop5.visibility.php

La visibilidad de una propiedad, un método o (a partir d PHP 7.1.0) una constante se puede definir anteponiendo a su declaración una de las palabras reservadas public, protected o private. A los miembros de clase declarados como 'public' se puede acceder desde donde sea; a los miembros declarados como 'protected', solo desde la misma clase o mediante clases heredadas. A los miembros declarados como 'private' únicamente se puede acceder desde la clase que los definió.



En el vídeo que encontrarás la explicación sobre la visibilidad dentro las clases

https://youtu.be/W-z2G9g2Bzw

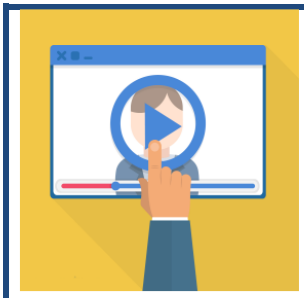
En el siguiente enlace tienes el código utilizado:

https://github.com/pacogomezarnal/codigoEjemploCursoPHP/tree/master/ambito

Volveremos en breve sobre el concepto de visibilidad después de analizar las funciones llamadas getter y setter.

2.7. Getter

Una función de tipo get, es aquella que sólo devuelve un parámetro normalmente un atributo y que no recibe ningún parámetro. Y digo normalmente ya que conforme vayamos avanzando en nuestros desarrollos, los métodos y sus casuísticas se complican.



En el vídeo que encontrarás la explicación sobre el uso de las funciones get

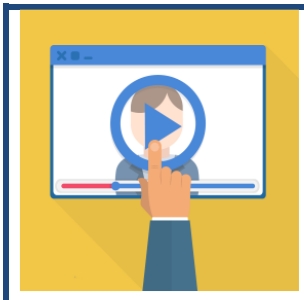
https://youtu.be/OKxH2BS0O-k

En el siguiente enlace tienes el código utilizado:

https://github.com/pacogomezarnal/codigoEjemploCursoPHP/tree/master/funcionesClases

2.8. Setter

Quizá más difícil de comprender pero no de usar son los métodos o funciones setter. En este caso la función o método recibe una única variable cuyo cometido es cambiar el contenido de una propiedad del objeto.



En el vídeo que encontrarás la explicación sobre el uso de las funciones set

https://youtu.be/W4Se3AxM3OY

En el siguiente enlace tienes el código utilizado:

https://github.com/pacogomezarnal/codigoEjemploCursoPHP/tree/master/funcionesClases

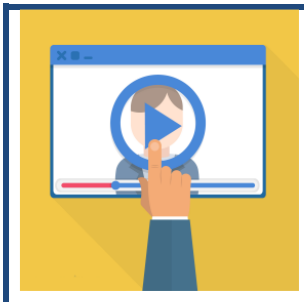
Ampliaremos nuestro ejemplo de equipo de baloncesto añadiendo las siguientes propiedades y métodos a la clase:

1) Utilizaremos la clase antes definida equipo.php.
2) Definiremos el getter y el setter de la propiedad posición.
3) Comprobaremos la correcta funcionalidad de estos métodos dentro del fichero liga.php

Compara la definición de estos métodos con los primeros definidos contra la propiedad nombre y coméntalo en el foro.

2.9. Visibilidad y setters

El uso de private dentro de un objeto tiene mucho sentido cuando estamos "setteando" una variable, ya que si una variable es pública no podemos controlar al 100% qué valor le va a colocar el usuario, mientras que cuando la variable es privada y su acceso se realiza mediante el setter, el control es total. Veámoslo en el siguiente ejemplo:



En el vídeo que encontrarás la explicación sobre el uso de los setters y private

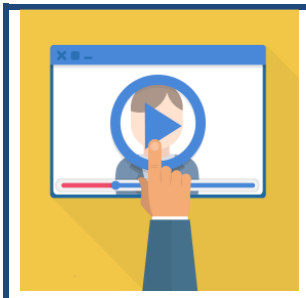
https://youtu.be/NfEtXfiJMNM

COMPRUEBA LO QUE SABES:

Una vez estudiado la visibilidad de propiedades y métodos, ¿serías capaz de comprobar con un ejemplo el uso del modificador *private* en un método?
Coméntalo en el foro.

2.10. Visibilidad y funciones

De igual forma que hemos visto que en las propiedades de una clase podemos utilizar los calificadores de visibilidad public y private, podemos aplicarlo a la definición de los métodos o funciones. Pero la pregunta en este caso es ¿para qué definiríamos un método privado?



En el vídeo que encontrarás la explicación sobre el uso de la visibilidad en las funciones

https://youtu.be/tAoy1Chx4ZA

2.11. Diagrama de clases y visibilidad

Una vez vistos los modificadores de visibilidad dentro de las clases, es interesante ver cómo se pueden representar y utilizar con los diagramas de clase UML ya que es muy sencillo y nos proporciona una visualización rápida de la definición de la visibilidad de propiedades y métodos.

Símbolo	Descripción
+	Público
-	Privado
#	Protegido

Veámoslo con un ejemplo donde podemos observar cómo se le añaden los modificadores tanto a las propiedades como a los métodos:

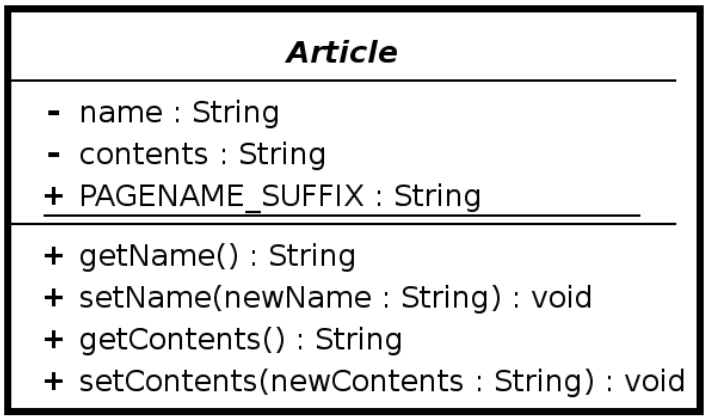


Imagen 2: Clase con visibilidad

Fuente de la imagen: https://upload.wikimedia.org/wikipedia/commons/thumb/3/34/UML_class_diagram_example.svg/640px-UML_class_diagram_example.svg.png

Tarea 2
Título: Objetos. Getters y Setters

2.12. Constructor

En cualquier lenguaje y dentro de la definición de las clases existe una función especial y sumamente importante que se denomina constructor. Decimos que es un método/función especial ya que por un lado es un método, pero por otro no sólo se puede denominar de una forma concreta.

constructor
http://php.net/manual/es/language.oop5.decon.php

PHP 5 permite a los desarrolladores declarar métodos constructores para las clases. Aquellas que tengan un método constructor lo invocarán en cada nuevo objeto creado, lo que lo hace idóneo para cualquier inicialización que el objeto pueda necesitar antes de ser usado.

Como se observa en la definición realizada por php.net, los constructores se llaman automáticamente cuando se genera un nuevo objeto, por lo que son muy interesantes para realizar inicializaciones o acciones que se deban realizar nada más crear un objeto.

Ejemplo de constructor

```php
<?php
class BaseClass {
   function __construct() {
       print "En el constructor BaseClass\n";
   }
}


// En el constructor BaseClass
$obj = new BaseClass(); //Sacará por pantalla, En el constructor
BaseClass

?>
```

Por último definiremos el contructor en nuestro ejemplo usado a lo largo de la unidad:

1) Utilizaremos la clase antes definida equipo.php.
2) Definiremos el constructor inicializando las propiedades nombre a "Equipo sin nombre" y posición a 0

	3) Comprobaremos la correcta funcionalidad del constructor a través de los getter dentro del fichero liga.php

Como se observa, el constructor se define en la clase con la palabra reservada __construct y recibirá los argumentos necesarios para el buen funcionamiento de la clase.

Caso Práctico 1
Título: Dado de juego

Resumen final:

En esta unidad hemos comenzado con la programación orientada a objetos, imprescindible dentro de cualquier lenguaje moderno y desarrollo de aplicaciones.

PHP, lenguaje históricamente procedural, realizó un cambio muy importante a partir de su versión 5, que finaliza, desde el punto de vista de la orientación a objetos, con la versión 7.

Al igual que ocurre con otras estructuras de PHP, la flexibilidad de PHP hace que la programación orientada a objetos nos permita múltiples soluciones para un mismo problema. La utilización de Objetos se hace imprescindible por lo tanto a partir de este punto, sobre todo en el desarrollo de estructuras tipo MVC que veremos más adelante.