UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

Diplomatura:

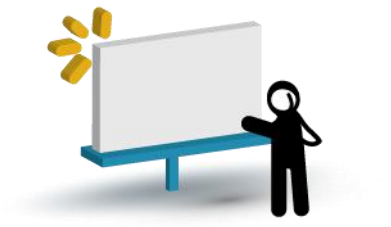
Proffessional backend developer

Curso: Programador Web avanzado

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Módulo 1: Introducción y nivelación

Unidad 2: PHP orientado a objetos



Presentación:

Hace algunos años surgió un nuevo paradigma de programación que transformó el mundo del desarrollo de software, se trata del paradigma de la programación orientada a objetos.

¿Qué es programar orientado a objetos? ¿Cuáles son sus ventajas? ¿Cómo nos cambie nuestra manera de pensar?

Estas respuestas y otras más desarrollamos en esta unidad.

Objetivos:

Que los participantes:

- Conozcan el paradigma de programación orientada a objetos.
- Aprendan a pensar en programación orientada a objetos.
- Comprendan las ventajas del paradigma.
- Conozcan la sintaxis de PHP para cumplir los objetivos anteriores.

Bloques temáticos*:

- Paradigma de programación orientada a objetos.
- Componentes dentro del POO.
- Aprendamos con la paloma pepita.
- POO con PHP.
- POO con PHP, nuevos conceptos.
- Ejercicio para entregar (Entrega obligatoria).

Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

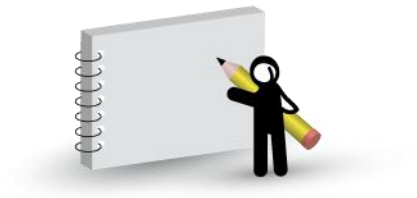
- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

** El MEC es el modelo de E-learning colaborativo de nuestro Centro.*



Tomen nota:

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

Paradigma de programación orientada a objetos

¿Qué es la programación orientada a objetos?

La programación Orientada a objetos (POO) es una forma especial de programar, más cercana a como expresaríamos las cosas en la vida real que otros tipos de programación.

Con la POO tenemos que aprender a pensar las cosas de una manera distinta, para escribir nuestros programas en términos de objetos, propiedades, métodos y otras cosas que veremos rápidamente para aclarar conceptos y dar una pequeña base que permita soltarnos un poco con este tipo de programación.

Beneficios de la POO

- Abstracción: Es un proceso mental por el que se ignoran las características de algo, quedándonos con lo que realmente nos importa. La clave para entenderlo es "proceso mental", así que nos tenemos que poner a pensar, extrayendo aquello que realmente nos importa e ignorando lo superfluo.
- Encapsulación: Es el proceso por el cual se ocultan los detalles del soporte donde se almacenan las características de una abstracción. En este punto el detalle clave para entender está en la palabra "soporte". Cuando encapsulamos estamos guardando cómo se soporta algo, como se almacena, qué medio es, cuál es su nombre, etc. Lo vas a entender mejor con un ejemplo. Es como cuando te dan un medicamente encapsulado, donde tú no ves lo que hay dentro. Solo puedes tocar la cápsula pero no ves si dentro hay polvo, un comprimido, el color que tiene, etc. Nadie quiere que manosees el medicamento y por ello te lo entregan encapsulado.
- Modularización: Es la descomposición de un sistema, creando una serie de piezas que colaboran entre sí, poco acoplados y cohesivos. Modularidad es tomar un sistema y tener la capacidad de segmentarlo en diversas partes independientes, que tengan sentido.
- Jerarquización: Es la estructuración por niveles de los módulos o elementos que forman parte de un sistema. Es la forma de organizar los módulos, existiendo jerarquías de todo tipo y con diversos grados de dependencia, responsabilidad, incumbencia, composición, entre otros.

Componentes dentro del POO

¿Qué es una clase?

Las clases son declaraciones de objetos, también se podrían definir como abstracciones de objetos. Esto quiere decir que la definición de un objeto es la clase. Cuando programamos un objeto y definimos sus características y funcionalidades en realidad lo que estamos haciendo es programar una clase.

Propiedades en clases

Las propiedades o atributos son las características de los objetos. Cuando definimos una propiedad normalmente especificamos su nombre y su tipo.

Nos podemos hacer a la idea de que las propiedades son algo así como **variables** donde almacenamos datos relacionados con los objetos.

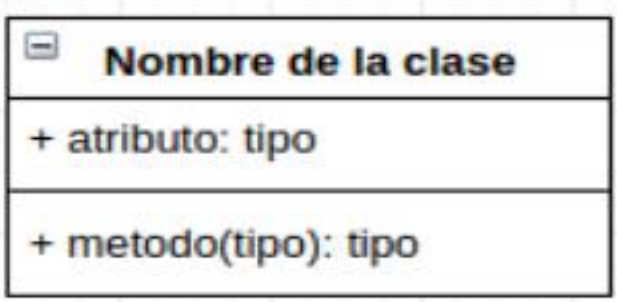
Métodos en las clases

Son las **funcionalidades (funciones)** asociadas a los objetos. Cuando estamos programando las clases las llamamos métodos. Los métodos son como funciones que están asociadas a un objeto.

La clase es el molde con el que vamos hacer una torta, el objeto seria cada torta que hacemos.

Diagrama de clases

Un diagrama de clases es un esquema en el cual podremos visualizar las propiedades y métodos que posee la misma.



Lo primer que se coloca es el nombre de la clase.

Luego (con una línea divisoria en el medio) se colocan los atributos, en caso de tener más de un atributo se separan solo por el salto de línea. Se suele especificar el tipo de dato que es.

Por último (luego de la línea divisoria con los atributos) se colocan los métodos, al igual que los atributos se separan por un salto de línea y se especifican los tipos de datos recibidos por parámetros y devueltos por el método.

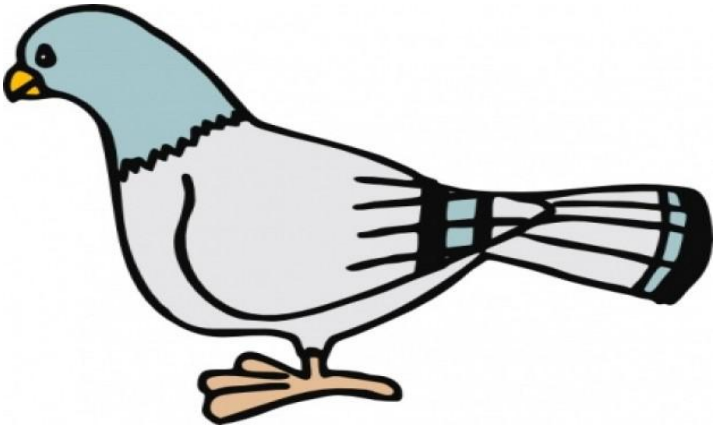
¿Qué es un objeto?

Los objetos son ejemplares de una clase cualquiera. Cuando creamos un ejemplar tenemos que especificar la clase a partir de la cual se creará. Esta acción de crear un objeto a partir de una clase se llama instanciar (que viene de una mala traducción de la palabra instace que en inglés significa ejemplar).

Como se menciono anteriormente la clase seria torta y una instancia de la clase seria la torta que definitivamente nos vamos a comer.

Aprendamos con la paloma pepita

Para comprender mejor el mundo la POO, jugaremos con nuestra amiga la paloma pepita.



¿Qué propiedades y métodos tiene pepita?

Como mencionamos anteriormente las propiedades son características de la clase y los métodos son las funcionalidades.

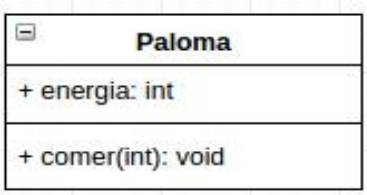
En este caso las propiedades y métodos son de pepita o de paloma….

Bien recorda que las propiedades y métodos son de la clase NO DEL OBJETO, por lo cual serán de paloma no de Pepita.

Veamos algunos ejemplos de propiedades y métodos de Pepita:

Propiedades

- Días de vida
- Km recorridos



Métodos:

- Volar

Ejercicio propuesto

- Agregar 1 atributo y 1 método más a la clase Paloma.

Programando la realidad

La POO se basa en la idea de que programamos clases de la vida real, por ejemplo podemos encontrar clases más tangibles como auto, casa, cancha, árbol o más abstractas como pensamiento, sentimiento, emoción.

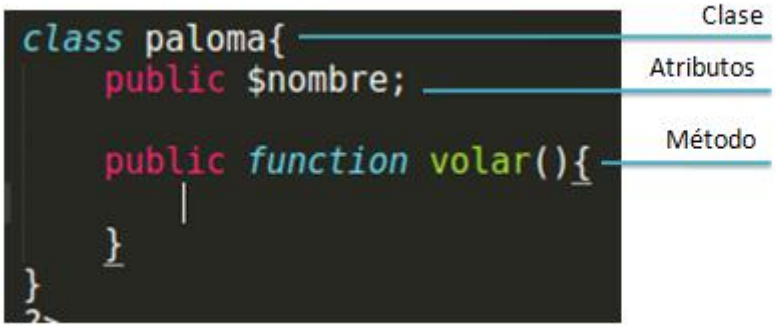
Ejercicio propuesto

Realiza el diagrama de clases de 3 clases que se te ocurran programar en la realidad. Especifica propiedades y métodos.

POO con PHP

Como es la sintaxis para desarrollar una clase

Una clase en php esta dentro de un script php, es decir en un archivo con extensión .php y dentro de <?php ?>



En la imagen anterior vemos el ejemplo de la clase Paloma con la propiedad nombre y el método volar.

Los métodos y propiedades de clase deben ir entre las {} (llaves) de apertura y cierre de la misma. En el caso de las propiedades deben finalizar con ; y no se pueden inicializar al momento de declararlas.

Instanciar una clase

Como dijimos anteriormente un objeto es una instancia de una clase, entonces veamos como instancia una clase.

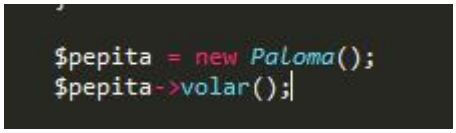
```
$pepita = new Paloma();
```

Vemos que para instancia una clase lo hacemos con el operador **new** seguido del nombre de la clase que queremos instanciar (observar que el nombre debe ser igual al seguido de la palabra class en la definición de clase). Al instanciar una clase tenemos un objeto, es decir ya no tenemos el molde de la paloma sino que tenemos la paloma viva.

Objeto -> mensaje

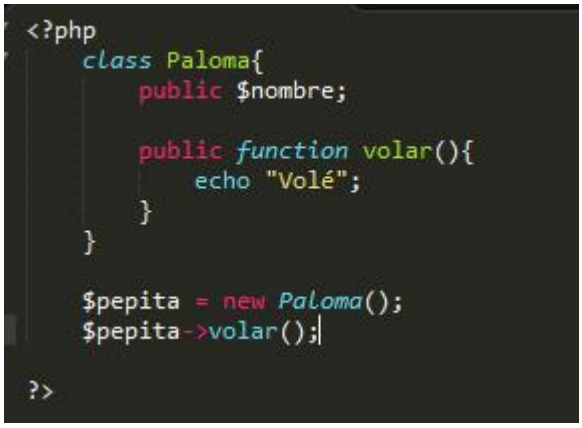
En la POO todo se resume a la relación objeto -> mensaje, es decir los objetos (instancias de clases cobran vida y se hablan entre ellos).

Vemos un ejemplo



Acá vemos como primero instanciamos la clase y luego enviamos el mensaje volar a la misma.

Veamos el ejemplo completo de definición (primer parte) e implementación (después del cierre de la llave de clase):



Ver archivo clase_paloma.php

Ejercicio propuesto

En base a las 3 clases propuestas en el ejercicio anterior, ahora debes programar su definición e implementar la misma (es decir instanciarla), adelante ya comenzaste a programar en objetos!!!

Constructor de una clase

Toda clase tiene su método constructor (a veces no está explicito, es decir no está declarado el método dentro de la implementación de la misma).

El método constructor, como su nombre lo indica, es el método que se ejecuta primero al instanciar una clase. Mediante este método podemos:

- Inicializar la clase
- Permite asignar valores a los atributos al instanciarse el objeto

Vemos un ejemplo:

```php
class Paloma{
    public $nombre;

    public function __construct($nombre_paloma){

        $this->nombre = $nombre_paloma;
    }
```

En este caso el constructor recibe el parámetro **$nombre_paloma** y se lo asigna a la variable de clase **$nombre.**

Las variables de clase (como $nombre) “viven” en toda la clase (es decir en todos los métodos de la clase) mientras que las variables locales ($nombre_paloma) solo “viven” dentro del método en el cual son utilizados (en el ejemplo el constructor).

Ejemplo completo de cómo definir la clase con el constructor y como implementar el mismo:

```php
<?php
    class Paloma{
        public $nombre;

        public function __construct($nombre_paloma){

            $this->nombre = $nombre_paloma;
        }

        public function getNombre(){
            return $this->nombre;
        }
        public function volar(){
            echo "Volé";
        }
    }

    $pepita = new Paloma("Pepita");
    $pepita->volar();
    echo $pepita->getNombre();

?>
```

Ver archivo paloma_constructor.php

En el ejemplo anterior vemos por un lado la definición de clase con el método constructor que recibe la variable **$nombre_paloma** como parámetro.

Por otro lado vemos la instancia de la clase paloma mediante **new Paloma(“Pepita”)**, es decir al momento de hacer el new parámetros entre paréntesis luego del nombre de la clase los parámetros que recibirá el método constructor. En este caso la variable **$nombre_paloma** tomara el valor **“Pepita”**.

Operador this

En el ejemplo anterior podemos observar el uso del operador $this:

```php
<?php
    class Paloma{
        public $nombre;

        public function __construct($nombre_paloma){

            $this->nombre = $nombre_paloma;
        }

        public function getNombre(){
            return $this->nombre;
        }
        public function volar(){
            echo "Volé";
        }
    }

    $pepita = new Paloma("Pepita");
    $pepita->volar();
    echo $pepita->getNombre();

?>
```

Ver método getNombre

El operador **$this** se utiliza para hacer referencia a la misma clase, en este caso **$this->nombre** estamos haciendo referencia a la variable de clase **$nombre** de la misma clase (en este caso Paloma).

Ejemplo clase polígono

```php
<?php
 //definición de la clase polígono incluyendo el
constructor y destructor

  class poligono{

     public $vertices;
     public $color;

     function __construct($ver=null)
     {
       if ($ver==null)
           $this->vertices = 4;
       else
           $this->vertices = $ver;
     }

    function __destruct()
    {
      echo 'vertices :  '.$this->vertices.' finalizado';
    }

    function muestraVertice()
    {
      echo $this->vertices;
    }

  }

  $cuadrado = new Poligono();
  $triangulo = new Poligono(3);
  ?>
```

Ejercicio propuesto

- Al desarrollo realizado sobre las 3 clases propuestas, utiliza el constructor de dichas clases para inicializar las propiedades especificadas
- Agreguemos a la clase Paloma lo siguiente:
    - Propiedades
        - Energía acumulada
        - Km
        - Días de vida.
    - Métodos
        - Comer
            - Energía aumenta en 5 por cada unidad.
        - Volar
            - Energía decrementa en 3 por km recorrido.
        - Dormir
            - Energía aumenta en 2 y aumenta 1 día de vida.

POO con PHP, nuevos conceptos

Encapsulamiento

Es uno de los conceptos fundamentales de POO.

Nos sirve para separar la definición de la implementación, es decir cuando definimos nombre de la clase y sus métodos (definición de clase):

```php
class Paloma{
    public $nombre;

    public function __construct($nombre_paloma){

        $this->nombre = $nombre_paloma;
    }

    public function getNombre(){
        return $this->nombre;
    }
    public function volar(){
        echo "Volé";
    }
}
```

Respecto de cuando instanciamos esa clase y le enviamos mensajes a la misma:

```php
$pepita = new Paloma("Pepita");
$pepita->volar();
echo $pepita->getNombre();
```

Modificadores de acceso

Modificadores de acceso:

- Public
  - El método o atributo es accesible desde la clase y cualquier parte de nuestro programa.
- Protected
  - El método o atributo será accesible desde la clase padre o sus hijas
- Private

- El método o atributo es accesible solo desde el interior de la clase

Ejercicio propuesto

- Para pensar ¿Qué propiedades de la clase Paloma serán públicos, privados o protegidos?

Herencia

La herencia consiste en la definición de una clase a partir de otra, es decir tendremos una clase hija la cual podrá utilizar la misma definición de métodos y propiedades de su clase padre ya que las “hereda”.

En el ejemplo vemos una clase AVE (clase padre) y 3 clases hijas (Paloma, canario y Gorrión).

Tanto la paloma, el canario y el gorrión son aves por lo cual podrán volar. Entonces el método volar lo definiremos dentro de la clase Ave (porque es compartido por todas sus clases hijas).

A su vez los 3 tipos de aves tienen la propiedad energía, por lo cual esta propiedad la vamos a definir en la clase AVE.

Herencia, sintaxis en php

Para definir que una clase hija A hereda de una clase padre B, debemos utilizar el modificador **extends**:

```
class Paloma extends Ave{
```

En este caso la clase Paloma hereda de la clase Ave.

Veamos un ejemplo de la clase Ave:

```php
class Ave{
    public $nombre;
    protected $energia;
    public $dias_vida;
    protected $km_recorridos;
    protected $decrementar_energia;

    public function __construct($nombre){
        $this->nombre = $nombre;
        $this->energia = 0;
        $this->dias_vida = 0;
        $this->km_recorridos = 0;
    }
    public function getEnergia(){
        return $this->energia;
    }
    public function getKmrecorridos(){
        return $this->km_recorridos;
    }

    public function comer($cantidad){

        $this->energia += ($cantidad*5);
    }


    public function volar($km){
        $this->energia -= ($km*$this->decrementar_energia);
        $this->km_recorridos += $km;
    }
    public function dormir(){
        $this->energia += 2;
        $this->dias_vida  += 1;
    }


    public function __destruct(){
        echo "<br>Adios pepita";
    }
}
```

Como vemos los métodos volar, dormir, comer son propios de la clase Ave ya que todos los tipos de ave tendrán dichos métodos.

Veamos un ejemplo de definición de la clase hija Paloma:

```php
class Paloma extends Ave{


    public function __construct($nombre){
        $this->nombre = $nombre;
        $this->energia = 0;
        $this->dias_vida = 0;
        $this->km_recorridos = 0;
        $this->decrementar_energia = 2;
    }

    public function __destruct(){
        echo "<br>Adios pepita";
    }
}
```

Ahora vemos un ejemplo de implementación de la clase hija Paloma utilizando los métodos definidos en la clase padre Ave:

```php
$pepita = new Paloma("Pepita");
echo "<br>Me llamo: ".$pepita->nombre;
$pepita->comer(5);
echo "<br>Comer, Energia: ".$pepita->getEnergia();
$pepita->volar(5);
echo "<br>Volar, Energia: ".$pepita->getEnergia();
echo "<br>Volar, Km Recorridos: ".$pepita->getKmrecorridos();
$pepita->dormir();
echo "<br>Dormir, Energia: ".$pepita->getEnergia();
echo "<br>Dormir, Dias de vida: ".$pepita->dias_vida
```

Como podemos observar instanciamos la clase Paloma.

Luego utilizamos el método comer y lo llamamos desde la instancia de la clase Paloma, pero como vimos anteriormente el método comer no estaba definido en la clase Paloma. Gracias a la herencia se utilizara el método comer definido en la clase Ave ya que Paloma es una clase hija de Ave.

Ahora vemos un ejemplo de la clase Canario

```php
class Canario extends Ave{


    public function __construct($nombre){
        $this->nombre = $nombre;
        $this->energia = 0;
        $this->dias_vida = 0;
        $this->km_recorridos = 0;
        $this->decrementar_energia = 2;
    }
    public function cantar(){
        echo "<br> El canario canta";
    }

    public function __destruct(){
        echo "<br>Adios canario";
    }
}
```

Como podemos observar la clase Canario tiene el método cantar, el cual no está definido en la clase Paloma.

Esto se debe a que solo el canario canta, entonces este método no estará disponible para todas las aves solo lo estará para el canario.

Un método definido en la clase padre puede ser redefinido en la clase hija para obtener un comportamiento diferente. Por ejemplo podemos volver a definir el método volar en la clase Paloma:

```php
class Paloma extends Ave{


    public function __construct($nombre){
        $this->nombre = $nombre;
        $this->energia = 0;
        $this->dias_vida = 0;
        $this->km_recorridos = 0;
        $this->decrementar_energia = 2;
    }
    public function volar($km){
        $this->energia -= ($km*$this->decrementar_energia);
        $this->km_recorridos += $km;
    }
    public function __destruct(){
        echo "<br>Adios pepita";
    }
}
```

Ejercicio propuesto

- Definir una nueva clase que sea hija de la clase Ave
- Definir propiedades y métodos propios de esta nueva clase de ave (que no herede de la clase padre, como el método cantar para el canario)
- Implementar la nueva clase utilizando los métodos propios y los heredades de la clase AVE

Parent

Mediante la utilización de parent, podemos desde una clase hija hacer referencia directa a un método de la clase padre (el operador $this servía para hacer referencia a métodos o propiedades propias de la clase, parent tiene el mismo sentido pero en lugar de hacer referencia a la clase propia lo hace a su clase padre). Ejemplo:

```php
class Paloma extends Ave{


    public function __construct($nombre){
        $this->nombre = $nombre;
        $this->energia = 0;
        $this->dias_vida = 0;
        $this->km_recorridos = 0;
        $this->decrementar_energia = 2;
    }
    public function volar($km){
        parent::volar($km);
        echo "vole";
    }
    public function __destruct(){
        echo "<br>Adios pepita";
    }
}
```

En el ejemplo vemos que la clase Paloma tiene su propio método volar. En la definición de dicho método vemos que hace **parent::volar($km)**, esto quiere decir que llama al método volar de la clase Ave. En la siguiente sentencia vemos que hacer un **echo "volé";**

En resumen la clase Paloma vuela igual que todas las aves, pero también imprime algo por pantalla por lo cual utilizamos parent para llamar al método en común y luego realizamos la impresión por pantalla.

Ejercicio para entregar (Entrega obligatoria)

Ejercicio 1

Crear una clase vehículos:

- Atributos:
  - Combustible
  - Marca
  - Modelo
  - Cantidad de ruedas
  - Color
- Métodos:
  - Encender (decrementa 2 unidades el combustible)
  - Andar (decrementa 5 unidades por km recorrido)
  - Cargar (Incrementa el combustible N unidades. N debe recibirse por parámetro en el método)
  - Pintar (cambia el color del vehículo)

Ejercicio 2

- Crear una clase auto que herede de vehículos:
  - Instanciar el objeto y definir las propiedades de vehículos
  - Agregar un atributo que indique si el auto es taxi (solo para la clase auto).
- Crear la clase moto que herede de vehículos:
  - Instanciar el objeto y definir las propiedades de vehículos
  - Agregar un método wheelie (decrementa la cantidad de combustible en 5)
- Crear la clase camioneta que herede de vehículos, definir 2 atributos y 2 métodos propios de camioneta que no sean compartidos por auto y moto.

Bibliografía utilizada y sugerida

- https://desarrolloweb.com/articulos/499.php
- http://php.net/manual/es/language.oop5.php
- https://styde.net/aprende-programacion-orientada-a-objetos-poo-con-php/
- https://styde.net/php-y-programacion-orientada-a-objetos/
- https://codigofacilito.com/cursos/php_oo
- http://www.tutorialesprogramacionya.com/phpya/poo/

Lo que vimos:

En esta unidad hemos visto qué es la programación orientada a objetos, qué ventajas tiene programar bajo este paradigma y cómo modifica nuestra forma de programar.

Luego hemos aplicado la POO bajo el lenguaje PHP para poder entender su sintaxis.

Lo que viene:

En la próxima unidad veremos el manejo de excepciones y errores en PHP, la utilización de XML y JSON.