PHP - POO

(CONCEPTOS BÁSICOS)

Oscar Fernando Aristizábal Cardona
ofaczero@gmail.com

Concepto POO

El lenguaje PHP permitir programar con las siguientes metodologías:

- **Programación Lineal:** Es cuando desarrollamos el código fuente de una pagina con scripts en PHP y alternamos con código HTML.
- **Programación Estructurada:** Es cuando utilizamos funciones que realizan diferentes actividades dentro de una pagina, estas funciones pueden ser llamadas dentro del mismo archivo o en una librería externa.
- **Programación Orientada a Objetos:** Es cuando creamos clases y definimos objetos de la misma clase.

Concepto:

Un objeto es una entidad independiente con sus propios datos y programación.

Ejemplo:

El procesador de un computador puede ser considerado un objeto, en este caso, sus datos (atributos) describen sus características físicas y su programación (métodos) describen el funcionamiento interno y su interrelación con otras partes del computador (también objetos).

Clase y Objeto

Declaración de una clase y creación de un objeto:

Una clase es un molde o plantilla en el que podemos crear múltiples objetos, con similares características, podemos definir atributos (que los conocemos como variables) y métodos (que los conocemos como funciones).

La clase define los atributos y métodos comunes a los objetos de ese tipo, pero luego, cada objeto tendrá sus propios valores y compartirán las mismas funciones.

Es obligatorio crear una clase antes de poder crear objetos (instancias) de esa clase.

Al crear un objeto de una clase, se dice que se crea una instancia de la clase o un objeto propiamente dicho.

Es recomendable buscar un nombre de clase lo más próximo a lo que representa. La palabra clave para declarar la clase es **class**, seguidamente el nombre de la clase y luego encerramos entre llaves de apertura y cerrado todos sus atributos(variables) y métodos(funciones).

Los atributos normalmente son privados (**private**), esto significa que no podemos acceder al mismo atributo desde fuera de la clase. Luego para definir los métodos se utiliza la misma sintaxis que las funciones del lenguaje PHP.

Concepto POO

La sintaxis básica para declarar una clase es:

```
class Nombreclase {
  atributos
  métodos
}
```

Sintaxis para la definición de los objetos:

```
$variable = new Nombreclase();
$variable->nombreMetodo(atributos);
```

```php
<?php
     class Persona {
        private $nombreCompleto;

        public function initNombre($nom) {
          $this->nombreCompleto = $nom;
        }
        public function mostrarNombre() {
          echo $this->nombreCompleto;
          echo '<br>';
        }
   }

   $per1 = new Persona();
   $per1->initNombre('Jeremias');
   $per1->mostrarNombre();

   $per2 = new Persona();
   $per2->initNombre('Homero');
   $per2->mostrarNombre();
?>
```

Atributos y Métodos

Atributos de una Clase:

- Los atributos son las propiedades distintivas, características y cualidades de cada clase.
- Los atributos por lo general contienen información sobre el objeto.
- Cuando creamos un objeto de una clase determinada, los atributos declarados por la clase son localizadas en memoria y pueden ser modificados mediante los métodos.
- Lo más conveniente es que los atributos sean privados para que solo los métodos de la clase puedan modificarlos.

Métodos de una Clase:

- Los métodos son como las funciones en los lenguajes estructurados, pero están definidos dentro de una clase y operan sobre los atributos de dicha clase.
- El objetivo de un método es ejecutar las actividades que tiene encomendada la clase a la cual pertenece.
- Los atributos de un objeto se modifican mediante llamadas a sus métodos.

El Método Constructor

Sintaxis del constructor:

```php
<?php
     class NombreClase {

        public function __construct([parámetros]) {
            // Bloque de Código
        }
   }
?>
```

El constructor es un método especial de una clase. El objetivo fundamental del constructor es inicializar los atributos del objeto que creamos.

Básicamente el constructor debe ser un método público y reemplaza a los métodos que inicializan atributos (variables).

El Método Constructor

- Las ventajas de implementar un constructor en lugar del método inicializar son:
    1. El constructor es el primer método que se ejecuta cuando se crea un objeto.
    2. El constructor se llama automáticamente.
    3. Quien utiliza POO conoce el objetivo de este método.

- Otras características de los constructores son:
    - El constructor se ejecuta inmediatamente luego de crear un objeto y no puede ser llamado nuevamente.
    - Un constructor no puede retornar datos.
    - Un constructor puede recibir parámetros que se utilizan normalmente para inicializar atributos.
    - El constructor es un método opcional, de todos modos es muy común definirlo.

El Método Constructor

```php
<?php
     class CatalogoMusica {
        private $nom_artista = array();
        private $nom_album   = array();

        public function __construct($n_art, $n_alb) {
          $this->nom_artista = $n_art;
          $this->nom_album   = $n_alb;
        }

        public function mostrarCatalogo() {
          echo “<h3>Nombre Artista: ”.$this->nom_artista.”</h3>”;
          echo “<h3>Nombre Album: ”.$this->nom_album.”</h3>”;
        }
  }

  $msc = new CatalogoMusica(‘Carlos Vives’, ‘El rock de mi pueblo’);
  $msc->mostrarCatalogo();

?>
```

Llamar métodos dentro de la clase

Hasta el momento hemos llamado a los métodos desde donde definimos un objeto de dicha clase, por ejemplo:

```php
$variable = new nombreClase();
$variable->nombreMetodo();
```

Que pasa si queremos llamar dentro de la clase a otro método que pertenece a la misma clase, la sintaxis es la siguiente:

```php
$this->nombreMetodo();
```

Es importante tener en cuenta que esto solo se puede hacer cuando estamos dentro de la misma clase.

Llamar métodos dentro de la clase

```php
<?php
      class tablaHTML {
         private $nf;
         private $nc;
         public function __construct($nfs, $ncs) {
           $this->nf = $nfs;
           $this->nc = $ncs;
         }
         public function contenidoTabla() {
           $this->inicioTabla();
           for($i=0; $i < $this->nf; $i++) {
             echo “<tr>”;
             for($j=0; $j < $this->nc; $j++) {
               echo “<td style=‘border: 1px solid #aaa’> X </td>”;
             }
             echo “</tr>”;
           }
           $this->finalTabla();
         }
         public function inicioTabla() {
           echo “<table style=‘border: 1px solid #333’>”;
         }
         public function finalTabla() {
           echo “</table>”;
         }
   }
   $tbl = new tablaHTML(5,5);
   $tbl->contenidoTabla();
?>
```

Acceso a métodos y atributos (public - private)

Hasta el momento hemos dicho que los atributos conviene definirlos con el modificador **private** y los métodos los hemos definido todos **public**.

Pero en muchas situaciones conviene que haya métodos privados (**private**).

Un método privado (**private**) solo puede ser llamado desde otro método de la clase.

No podemos llamar a un método privados desde donde definimos un objeto.

Con la definición de métodos privados se elimina la posibilidad de llamar a métodos por error.

Entonces el modificador **private** nos permite ocultar en la clase atributos y métodos que no queremos que los accedan directamente quien definen objetos de dicha clase.

Uno de los objetivos fundamentales de la POO es el **encapsulamiento**. El **encapsulamiento** es una técnica por el que se ocultan las características internas de una clase de todos aquellos elementos (atributos y métodos) que no tienen porque conocerla otros objetos.

Cuando uno planea una clase debe poner mucha atención cuales responsabilidades (métodos) deben ser públicas y cuales responsabilidades no queremos que las conozcan los demás.

Colaboración de Objetos

Hasta el momento hemos trabajado con una sola clase, de la cual hemos definido uno o varios objetos. Pero una aplicación real consta de muchas clases.

Existen una forma de relacionar las clases "**Colaboración**".

Cuando dentro de una clase definimos un atributo o una variable de otra clase decimos que esta segunda clase colabora con la primera.

Sintaxis:

```
class nombreclase1 {
  [atributos y métodos]
}
class nombreclase2 {
  [atributos y métodos]
}
class nombreclase3 {
  private $nombreclase1;
  private $nombreclase2;
  [métodos]
}
```

Parámetros de tipo objeto y opcionales

Parámetros de tipo objeto:

Otra posibilidad que nos presenta el lenguaje PHP es pasar parámetros no solo de tipo primitivo (enteros, reales, cadenas etc.) sino parámetros de tipo objeto.

Sintaxis:

```
$variable1 = new Nombreclase1($parametros1);
$variable2 = new Nombreclase2($parametros1);
$variable1 -> nombremetodo($variable2);
```

Parámetros de tipo opcionales:

Un parámetro es opcional si en la declaración del método le asignamos un valor por defecto. Si luego llamamos al método sin enviarle dicho valor tomará el que tiene por defecto.

Sintaxis:

```
public function __construct($par, $par2 = 22, $par3 = 'Texto')

$variable1 = new Nombreclase1('parametro1');
```

Parámetros opcionales

Restricciones que debemos tener en cuenta cuando utilizamos parámetros opcionales:

• No podemos definir un parámetro opcional y seguidamente un parámetro obligatorio. Es decir los parámetros opcionales se deben ubicar a la derecha en la declaración del método.

• Cuando llamamos al método no podemos alternar indicando algunos valores a los parámetros opcionales y otros no. Es decir que debemos pasar valores a los parámetros opcionales teniendo en cuenta la dirección de izquierda a derecha en cuanto a la ubicación de parámetros.

• Podemos definir parámetros opcionales tanto para el constructor como para cualquier otro método de la clase. Los parámetros opcionales nos permiten desarrollar clases que sean más flexibles en el momento que definimos objetos de las mismas.

Herencia

La herencia significa que se pueden crear nuevas clases partiendo de clases existentes, que tendrá todas los atributos y los métodos de su 'superclase' o 'clase padre' y además se le podrán añadir otros atributos y métodos propios.

En PHP, a diferencia de otros lenguajes orientados a objetos (C++), una clase sólo puede derivar de una única clase, es decir, PHP no permite herencia múltiple.

Superclase o clase padre:

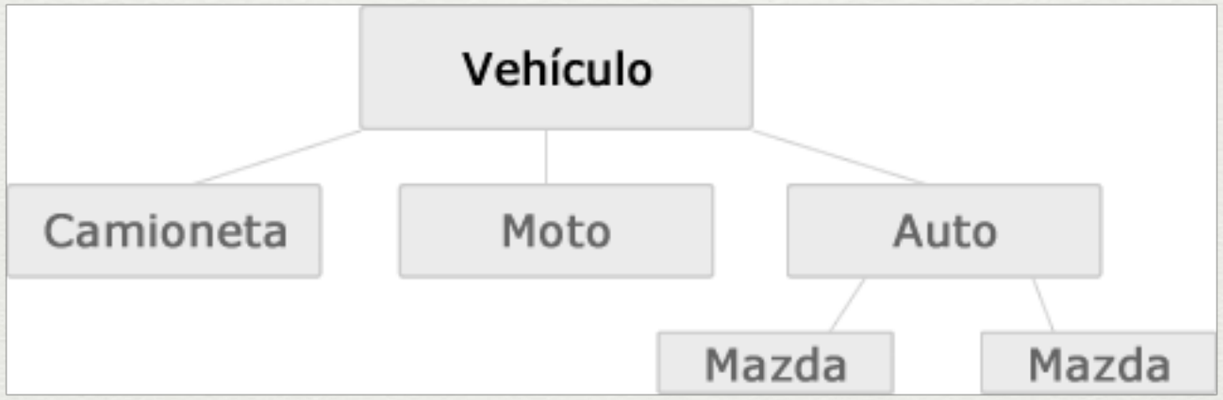
Clase de la que desciende o deriva una clase. Las clases hijas (descendientes) heredan (incorporan) automáticamente los atributos y métodos de la la clase padre.

Subclase:

Clase desciendiente de otra. Hereda automáticamente los atributos y métodos de su superclase. Es una especialización de otra clase. Admiten la definición de nuevos atributos y métodos para aumentar la especialización de la clase.

Herencia

Ejemplos teóricos de herencia:



Herencia

```php
<?php
  class Operacion {
        private $num1;
        private $num2;
        private $resultado;

        public function cargarNum1($n1) {
          $this->num1 = $n1;
        }
        public function cargarNum2($n2) {
          $this->num2 = $n2;
        }
        public function imprimirResultado() {
          echo $this->resultado.“<br>”;
        }
  }
  class Producto extends Operacion {
        public function calcular() {
          $this->resultado = $this->num1 + $this->num2;
        }
  }
  $res = new Producto();
  $res->cargarNum1(5);
  $res->cargarNum2(6);
  $res->calcular();
  $res->imprimirResultado();
?>
```

Modificadores de acceso atributos y métodos (protected)

Habíamos dicho que otro objetivo de la POO es el **encapsulamiento** (es decir ocultar todo aquello que no le interese a otros objetos), para lograr esto debemos definir los atributos y métodos como privados.

El inconveniente es cuando debemos utilizar herencia.

Una subclase no puede acceder a los atributos y métodos privados de la clase padre. Para poder accederlos deberíamos definirlos como públicos (pero esto trae como contrapartida que perdemos el **encapsulamiento** de la clase).

Aquí es donde entra en juego el modificador **protected**. Un atributo o método **protected** puede ser accedido por la clase, por todas sus subclases pero no por los objetos que definimos de dichas clases.

Sobreescritura de métodos

Una subclase en PHP puede redefinir un método, es decir que podemos crear un método con el mismo nombre que el método de la clase padre. Ahora cuando creamos un objeto de la subclase, el método que se llamará es el de dicha subclase.

Lo más conveniente es sobreescribir métodos para completar el algoritmo del método de la clase padre. No es bueno sobreescribir un método y cambiar completamente su comportamiento.

Sintaxis:

```
parent::nombremetodo();
```

Sobreescritura del constructor

Cuando creamos un objeto de una clase el primer método que se ejecuta es el constructor. Si la clase no tiene constructor pero la subclase si lo tiene, el que se ejecuta es el constructor de la clase padre.

Pero si la subclase también tiene constructor, es decir sobreescribimos el constructor de la clase padre.

Sintaxis:

```
parent::__construct(parametros);
```

Clases abstractas y concretas

Una clase abstracta tiene por objetivo agrupar atributos y métodos que luego serán heredados por otras subclases.

No es obligatorio que toda clase padre sea abstracta.

Sintaxis:

```
abstract class [nombre de clase] {
  [atributos]
  [metodos]
}
```

La ventaja de definir las clases abstractas con este modificador es que se producirá un error en tiempo de ejecución si queremos definir un objeto de dicha clase. Luego hay que tener bien en cuenta que solo podemos definir objetos de las clases concretas.

Métodos abstractos

En conceptos anteriores definimos clases abstractas que tienen por objetivo agrupar atributos y métodos comunes a un conjunto de subclases.

Si queremos que las subclases implementen comportamientos obligatoriamente podemos definir métodos abstractos.

Un método abstracto se declara en una clase pero no se lo implementa.

Sintaxis:

```
public abstract function nombremetodo();
```

Con esto logramos que todas las subclases de la clase padre deben implementar el método abstract.

Si una subclase no lo implementa se produce un error.

Métodos y clases final

Si a un método le agregamos el modificador final significa que ninguna subclase puede sobreescribirlo.

Este mismo modificador se lo puede aplicar a una clase, con esto estaríamos indicando que dicha clase no se puede heredar.

Sintaxis:

```
public final function nombremetodo()
```

Referencia y clonación de objetos

Hay que tener en cuenta que un objeto es una estructura de datos compleja. Luego cuando asignamos una variable de tipo objeto a otra variable lo que estamos haciendo es guardar la referencia del objeto. No se está creando un nuevo objeto, sino otra variable por la que podemos acceder al mismo objeto.

Si queremos crear un nuevo objeto idéntico a otro debemos utilizar el operador clone.

Sintaxis:

```
$objeto1 = new Nombreclase();
$objeto2 = clone($objeto1);
```

función __clone()

PHP nos permite crear un método que se llamará cuando ejecutemos el operador clone. Este método puede entre otras cosas inicializar algunos atributos.

Si no se define el método __clone se hará una copia idéntica del objeto que le pasamos como parámetro al operador clone.

Sintaxis:

```
public function __clone()
 {
   $this->atribruto = 20;
 }
```

Operador instanceof

Cuando tenemos una lista de objetos de distinto tipo y queremos saber si un objeto es de una determinada clase el lenguaje PHP nos provee del operador instanceof.

Sintaxis:

```
if (Objeto instanceof Nombreclase)
     //Ejecutar bloque;
```

Método destructor de una clase (__destruct)

Otro método que se ejecuta automáticamente es el **__destruct** (destructor de la clase).

Las características de este método son:

- El objetivo principal es liberar recursos que solicitó el objeto (conexión a la base de datos, creación de imágenes dinámicas etc.)
- Es el último método que se ejecuta de la clase.
- Se ejecuta en forma automática, es decir no tenemos que llamarlo.
- Debe llamarse **__destruct**.
- No retorna datos.
- Es menos común su uso que el constructor, ya que PHP gestiona bastante bien la liberación de recursos en forma automática.

Sintaxis:

```
  public function __destruct() {

 }
```

Métodos estáticos de una clase (static)

Un método estático pertenece a la clase pero no puede acceder a los atributos de una instancia. La característica fundamental es que un método estático se puede llamar sin tener que crear un objeto de dicha clase.

Un método estático es lo más parecido a una función de un lenguaje estructurado. Solo que se lo encapsula dentro de una clase.

La sintaxis para llamar un método estático como vemos es distinta a la llamada de métodos de un objeto. Indicamos primero el nombre de la clase, luego el operador '::' y por último indicamos en nombre del método estático a llamar.

Sintaxis:

```
public static function metodo($valor) {
    return strlen($valor);
}

Nombreclase::metodo($c);
```

Interfaces de Objetos

Las interfaces de objetos permiten crear código con el cual especificamos qué métodos deben ser implementados por una clase, sin tener que definir cómo estos métodos son manipulados.

Las interfaces son definidas utilizando la palabra clave interface, de la misma forma que con clases estándar, pero sin métodos que tengan su contenido definido.

Todos los métodos declarados en una interfaz deben ser públicos, ya que ésta es la naturaleza de una interfaz.

Para implementar una interfaz, se utiliza el operador implements. Todos los métodos en una interfaz deben ser implementados dentro de la clase; el no cumplir con esta regla resultará en un error fatal. Las clases pueden implementar más de una interfaz si se deseara, separándolas cada una por una coma.

```
interface Clase {
    public function metodo($parametro);
}

class NombreClase implements Clase { }
```

NameSpaces

Los espacios de nombres permiten crear aplicaciones complejas con mayor flexibilidad, evitando problemas de conflictos (colisiones) entre clases y así mejorar legibilidad del código.

Un namespace no es más que un directorio para clases, traits, interfaces, funciones y constantes.

Se crean utilizando la palabra reservada namespace al principio del archivo, antes que cualquier otro código.

Sintaxis:

```
namespace MiProyecto;

use Proyecto\Prueba\MiProyecto;
```