CAPÍTULO I

Introducción a la Programación Orientada a Objetos (POO)

INTRODUCCIÓN A LA PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

La POO es un **paradigma de programación** (o técnica de programación) que utiliza objetos e interacciones en el diseño de un sistema.

ELEMENTOS DE LA POO

La POO está compuesta por una serie de elementos que se detallan a continuación.

CLASE

Una clase es un **modelo** que se utiliza para crear objetos que comparten un mismo comportamiento, estado e identidad.

Metáfora

Persona es la metáfora de una clase (la abstracción de Juan, Pedro, Ana y María), cuyo comportamiento puede ser caminar, correr, estudiar, leer, etc. Puede estar en estado despierto, dormido, etc. Sus características (propiedades) pueden ser el color de ojos, color de pelo, su estado civil, etc.

```
class Persona {
    # Propiedades
    # Métodos
}
```

OBJETO

Es una **entidad** provista de métodos o mensajes a los cuales responde (comportamiento); atributos con valores concretos (estado); y propiedades (identidad).

```
$persona = new Persona();
/*
   El objeto, ahora, es $persona,
   que se ha creado siguiendo el modelo de la clase Persona
*/
```

MÉTODO

Es el **algoritmo** asociado a un objeto que indica la capacidad de lo que éste puede hacer.

```
function caminar() {
    #...
}
```

EVENTO Y MENSAJE

Un **evento** es un suceso en el sistema mientras que un **mensaje** es la comunicación del suceso dirigida al objeto.

PROPIEDADES Y ATRIBUTOS

Las propiedades y atributos, son **variables** que contienen datos asociados a un objeto.

```
$nombre = 'Juan';
$edad = '25 años';
$altura = '1,75 mts';
```

Características conceptuales de la POO

La POO debe guardar ciertas características que la identifican y diferencian de otros paradigmas de programación. Dichas características se describen a continuación.

Abstracción

Aislación de un elemento de su contexto. Define las características esenciales de un objeto.

Encapsulamiento

Reúne al mismo nivel de abstracción, a todos los elementos que puedan considerarse pertenecientes a una misma entidad.

Modularidad

Característica que permite dividir una aplicación en varias partes más pequeñas (denominadas módulos), independientes unas de otras.

Ocultación (aislamiento)

Los objetos están aislados del exterior, protegiendo a sus propiedades para no ser modificadas por aquellos que no tengan derecho a acceder a las mismas.

Polimorfismo

Es la capacidad que da a diferentes objetos, la posibilidad de contar con métodos, propiedades y atributos de igual nombre, sin que los de un objeto interfieran con el de otro.

Herencia

Es la relación existente entre dos o más clases, donde una es la principal (madre) y otras son secundarias y dependen (heredan) de ellas (clases "hijas"), donde a la vez, los objetos heredan las características de los objetos de los cuales heredan.

Recolección de basura

Es la técnica que consiste en destruir aquellos objetos cuando ya no son necesarios, liberándolos de la memoria.

CAPÍTULO II

Programación Orientada a Objetos en PHP 5

PROGRAMACIÓN ORIENTADA A OBJETOS EN PHP 5

En este capítulo veremos como aplicar los conceptos de la POO en el entorno del lenguaje PHP 5+.

CLASES Y OBJETOS EN PHP 5

DEFINICIÓN DE CLASES

Según el **Manual Oficial de PHP**, una **Clase** es:

> *[…] "una colección de variables y funciones que trabajan con estas variables. Las variables se definen utilizando **var** y las funciones utilizando **function**" [...]*[5]

Para definir una clase, el **Manual Oficial de PHP**, continúa diciendo:

> *[…] "La definición básica de clases comienza con la palabra clave class, seguido por un nombre de clase, continuado por un par de llaves que encierran las definiciones de las propiedades y métodos pertenecientes a la clase. El nombre de clase puede ser cualquier etiqueta válida que no sea una palabra reservada de PHP. Un nombre válido de clase comienza con una letra o un guión bajo, seguido de la cantidad de letras, números o guiones bajos que sea." […]*[6]

Veamos un ejemplo de **definición de clase**:

```
class NombreDeMiClase {
    #...
}
```

Reglas de Estilo sugeridas

Utilizar **CamelCase** para el nombre de las clases.

La **llave de apertura en la misma línea** que el nombre de la clase, permite una mejor legibilidad del código.

DECLARACIÓN DE CLASES ABSTRACTAS

Las clases abstractas son aquellas que **no necesitan ser instanciadas** pero sin embargo, **serán heredadas en algún momento**. Se definen anteponiendo la palabra clave ***abstract*** *a class*:

```
abstract class NombreDeMiClaseAbstracta {
    #...
}
```

Este tipo de clases, será la que contenga **métodos abstractos** (que veremos más adelante) y generalmente, su finalidad, es la de declarar clases "genéricas" que necesitan ser declaradas pero a las cuales, no se puede otorgar una definición precisa (de eso, se encargarán las clases que la hereden).

5 Fuente de la cita: http://php.net/manual/es/keyword.class.php

6 Fuente de la cita: http://www.php.net/manual/es/language.oop5.basic.php

HERENCIA DE CLASES

Los **objetos pueden heredar propiedades y métodos de otros objetos**. Para ello, PHP permite la "extensión" (herencia) de clases, cuya característica representa la relación existente entre diferentes objetos. Para definir una clase como extención de una clase "madre" se utiliza la palabra clave ***extends***.

```
class NombreDeMiClaseMadre {
    #...
}

class NombreDeMiClaseHija extends NombreDeMiClaseMadre {
    /* esta clase hereda todos los métodos y propiedades de
       la clase madre NombreDeMiClaseMadre
    */
}
```

DECLARACIÓN DE CLASES FINALES EN PHP

PHP 5 incorpora clases finales que **no pueden ser heredadas por otra**. Se definen anteponiendo la palabra clave ***final***.

```
final class NombreDeMiClaseFinal {
    #esta clase no podrá ser heredada
}
```

¿QUÉ TIPO DE CLASE DECLARAR?

Hasta aquí, han quedado en claro, cuatro tipos de clases diferentes: **Instanciables**, **astractas**, **heredadas** y **finales**. ¿Cómo saber qué tipo de clase declarar? Todo dependerá, de lo que necesitemos hacer. Este cuadro, puede servirnos como guía básica:

Necesito...	Instanciable	Abstracta	Heredada	Final
Crear una clase que pueda ser instanciada y/o heredada	X			
Crear una clase cuyo objeto guarda relación con los métodos y propiedades de otra clase			X	
Crear una clase que solo sirva de modelo para otra clase, sin que pueda ser instanciada		X		
Crear una clase que pueda instanciarse pero que no pueda ser heredada por ninguna otra clase				X

OBJETOS EN PHP 5

Una vez que las clases han sido declaradas, será necesario crear los objetos y utilizarlos, aunque hemos visto que algunas clases, como las clases abstractas son solo modelos para otras, y por lo tanto no necesitan instanciar al objeto.

INSTANCIAR UNA CLASE

Para instanciar una clase, solo es necesario utilizar la palabra clave ***new***. El objeto será creado, asignando esta instancia a una variable (la cual, adoptará la forma de objeto). Lógicamente, la clase debe haber sido declarada antes de ser instanciada, como se muestra a continuación:

```
# declaro la clase
class Persona {
    #...
}

# creo el objeto instanciando la clase
$persona = new Persona();
```

Reglas de Estilo sugeridas

Utilizar nombres de variables (objetos) descriptivos, siempre en letra minúscula, separando palabras por guiones bajos. Por ejemplo si el nombre de la clase es **NombreDeMiClase** como variable utilizar **$nombre_de_mi_clase**. Esto permitirá una mayor legibilidad del código.

PROPIEDADES EN PHP 5

Las propiedades representan ciertas características del objeto en sí mismo. Se definen anteponiendo la palabra clave ***var*** al nombre de la variable (propiedad):

```
class Persona {
    var $nombre;
    var $edad;
    var $genero;
}
```

Las propiedades pueden gozar de diferentes características, como por ejemplo, la visibilidad: pueden ser **públicas, privadas** o **protegidas**. Como veremos más adelante, la visiblidad de las propiedades, es aplicable también a la visibilidad de los métodos.

PROPIEDADES PÚBLICAS

Las propiedades públicas se definen anteponiendo la palabra clave ***public*** al nombre de la variable. Éstas, **pueden ser accedidas desde cualquier parte de la aplicación**, sin restricción.

```
class Persona {
    public $nombre;
    public $genero;
}
```

PROPIEDADES PRIVADAS

Las propiedades privadas se definen anteponiendo la palabra clave ***private*** al nombre de la variable. Éstas solo **pueden ser accedidas por la clase que las definió**.

```
class Persona {
    public $nombre;
    public $genero;
    private $edad;
}
```

PROPIEDADES PROTEGIDAS

Las propiedades protegidas **pueden ser accedidas por la propia clase que la definió, así como por las clases que la heredan**, pero no, desde otras partes de la aplicación. Éstas, se definen anteponiendo la palabra clave ***protected*** al nombre de la variable:

```php
class Persona {
    public $nombre;
    public $genero;
    private $edad;
    protected $pasaporte;
}
```

PROPIEDADES ESTÁTICAS

Las propiedades estáticas representan una característica de "variabilidad" de sus datos, de gran importancia en PHP 5. Una propiedad declarada como estática, **puede ser accedida sin necesidad de instanciar un objeto**. y su valor es estático (es decir, no puede variar ni ser modificado). Ésta, se define anteponiendo la palabra clave ***static*** al nombre de la variable:

```php
class PersonaAPositivo extends Persona {
    public static $tipo_sangre = 'A+';
}
```

ACCEDIENDO A LAS PROPIEDAD DE UN OBJETO

Para acceder a las propiedad de un objeto, existen varias maneras de hacerlo. Todas ellas, dependerán del ámbito desde el cual se las invoque así como de su condición y visibilidad.

ACCESO A VARIABLES DESDE EL ÁMBITO DE LA CLASE

Se accede a una propiedad **no estática** dentro de la clase, utilizando la pseudo-variable ***$this*** siendo esta pseudo-variable una referencia al objeto mismo:

```php
return $this->nombre;
```

Cuando la variable **es estática**, se accede a ella mediante el operador de resolución de ámbito, doble dos-puntos **::** anteponiendo la palabra clave ***self*** o ***parent*** según si trata de una variable de la misma clase o de otra de la cual se ha heredado, respectivamente:

```php
print self::$variable_estatica_de_esta_clase;
print parent::$variable_estatica_de_clase_madre;
```

ACCESO A VARIABLES DESDE EL EXTERIOR DE LA CLASE

Se accede a una propiedad **no estática** con la siguiente sintáxis: `$objeto->variable`

Nótese además, que este acceso dependerá de la visibilidad de la variable. Por lo tanto, solo variables públicas pueden ser accedidas desde cualquier ámbito fuera de la clase o clases heredadas.

```php
# creo el objeto instanciando la clase
$persona_a_positivo = new PersonaAPositivo();

# accedo a la variable NO estática
print $persona_a_positivo->nombre;
```

Para acceder a una propiedad pública y **estática** el objeto **no necesita ser instanciado**, permitiendo así, el acceso a dicha variable mediante la siguiente sintáxis:

`Clase::$variable_estática`

```
# accedo a la variable estática
print PersonaAPositivo::$tipo_sangre;
```

CONSTANTES DE CLASE

Otro tipo de "propiedad" de una clase, son las constantes, aquellas que **mantienen su valor de forma permanente y sin cambios**. A diferencia de las propiedades estáticas, **las constantes solo pueden tener una visibilidad pública**.

Puede declararse una constante de clase como cualquier constante normal en PHP 5. El acceso a constantes es exactamente igual que al de otras propiedades.

Reglas de Estilo sugeridas

Utilizar **NOMBRE_DE_CONSTANTE** en letra MAYÚSCULA, ayuda a diferenciar rápidamente constantes de variables, haciendo más legible el código.

```
const MI_CONSTANTE = 'Este es el valor estático de mi constante';
```

MÉTODOS EN PHP 5

Cabe recordar, para quienes vienen de la programación estructurada, que el método de una clase, es un algoritmo igual al de una función. La única diferencia entre método y función, es que **llamamos método a las funciones de una clase** (en la POO), mientras que llamamos funciones, a los algoritmos de la programación estructurada.

Reglas de Estilo sugeridas

Utilizar **nombres_de_funciones_descriptivos**, en letra minúscula, separando palabras por guiones bajos, ayuda a comprender mejor el código fuente haciéndolo más intuitivo y legible.

La forma de declarar un método es anteponiendo la palabra clave ***function*** al nombre del método, seguido por un par paréntesis de apertura y cierre y llaves que encierren el algoritmo:

```
# declaro la clase
class Persona {
    #propiedades

    #métodos
    function donar_sangre() {
        #...
    }
}
```

Al igual que cualquier otra función en PHP, los métodos recibirán los parámetros necesarios indicando aquellos requeridos, dentro de los paréntisis:

```
# declaro la clase
class Persona {
    #propiedades

    #métodos
    function donar_sangre($destinatario) {
        #...
    }
}
```

MÉTODOS PÚBLICOS, PRIVADOS, PROTEGIDOS Y ESTÁTICOS

Los métodos, al igual que las propiedades, pueden ser **públicos, privados, protegidos** o **estáticos**. La forma de declarar su visibilidad tanto como las características de ésta, es exactamente la misma que para las propiedades.

```
static function a() { }
protected function b() { }
private function c() { }
# etc...
```

MÉTODOS ABSTRACTOS

A diferencia de las propiedades, los métodos, pueden ser **abstractos** como sucede con las clases.

El **Manual Oficial de PHP**, se refiere a los métodos abstractos, describiéndolos de la siguiente forma:

> *[…] "Los métodos definidos como abstractos simplemente declaran la estructura del método, pero no pueden definir la implementación. Cuando se hereda de una clase abstracta, todos los métodos definidos como abstract en la definición de la clase parent deben ser redefinidos en la clase child; adicionalmente, estos métodos deben ser definidos con la misma visibilidad (o con una menos restrictiva). Por ejemplo, si el método abstracto está definido como protected, la implementación de la función puede ser redefinida como protected o public, pero nunca como private." […]*[7]

Para entender mejor los métodos abstractos, podríamos decir que a grandes rasgos, los **métodos abstractos son aquellos que se declaran inicialmente en una clase abstracta, sin especificar el algoritmo que implementarán**, es decir, que solo son declarados pero no contienen un "código" que especifique qué harán y cómo lo harán.

Tal vez, te preguntes **¿Cuál es la utilidad de definir métodos abstractos y clases abstractas?** Para responder a esta pregunta, voy enfocarme en un caso de la vida real, en el cual estuve trabajando hace poco tiempo.

Ejemplo

Se trataba de hacer un sistema de gestión informática, para las farmacias de los Hospitales del Gobierno de la Ciudad de Buenos Aires. Un punto fundamental, era pensar en los insumos farmacéuticos como "un todo abstracto". ¿Por qué? Fundamentalmente, porque si bien existen insumos farmacéuticos de todo tipo y especie, cada uno de ellos, comparte características comunes, que por sí solas no pueden definirse con precisión. Por ejemplo, todos los insumos

7 Fuente de la cita: http://www.php.net/manual/es/language.oop5.abstract.php

farmacéuticos requieren de un tipo de conservación especial. Algunos requieren refrigeración a determinada temperatura que solo puede ser satisfecha conservándolos en una heladera; otros requieren conservarse en un ambiente seco; otros, no pueden tomar contacto con el exterior, a fin de conservar su capacidad estéril; etc. ¿Podía definirse con exactitud una clase Insumo? La respuesta a esa pregunta, es justamente su pregunta retórica ¿irías a la farmacia a pedirle al farmacéutico "deme un insumo de 500 mg"? Insumo, representa la entidad "abstracta" y para eso, sirven las clases abstractas. Con ellas declaramos aquellos "objetos" que no pueden ser definidos con presición pero aseguramos allí, todas aquellas características que dichos objetos, guardarán entre sí. Declarar un método conservar_insumo() como abstracto, serviría para luego definir con exactitud, en una clase heredada, el algoritmo exacto que determinado insumo necesitaría para procesar su conservación. Es así entonces, que una clase InsumoRefrigerado heredaría de Insumo, y redefiniría el método conservar_insumo() indicando un algoritmo que solicitara la temperatura a la cual debía conservarse en heladera, etc.

MÉTODOS MÁGICOS EN PHP 5

PHP 5, nos trae una gran cantidad de auto-denominados **"métodos mágicos"**. Estos métodos, otorgan una funcionalidad pre-definida por PHP, que pueden aportar valor a nuestras clases y ahorrarnos grandes cantidades de código. Lo que muchos programadores consideramos, ayuda a convertir a PHP en un lenguaje orientado a objetos, cada vez más robusto.

Entre los métodos mágicos, podemos encontrar los siguientes:

El Método Mágico `__construct()`

El método `__construct()` es aquel que será invocado de manera automática, al instanciar un objeto. Su función es la de ejecutar cualquier inicialización que el objeto necesite antes de ser utilizado.

```
# declaro la clase
class Producto {
    #defino algunas propiedades
    public $nombre;
    public $precio;
    protected $estado;

    #defino el método set_estado_producto()
    protected function set_estado_producto($estado) {
        $this->estado = $estado;
    }

    # constructor de la clase
    function __construct() {
        $this->set_estado_producto('en uso');
    }
}
```

En el ejemplo anterior, el constructor de la clase se encarga de definir el estado del producto como "en uso", antes de que el objeto (Producto) comience a utilizarse. Si se agregaran otros métodos, éstos, podrán hacer referencia al estado del producto, para determinar si ejecutar o no determinada función. Por ejemplo, no podría mostrarse a la venta un producto "en uso por el sistema", ya que a éste, se le podría estar modificando el precio.

El método mágico `__destruct()`

El método `__destruct()` es el encargado de liberar de la memoria, al objeto cuando ya no es referenciado. Se puede aprovechar este método, para realizar otras tareas que se

estimen necesarias al momento de destruir un objeto.

```
# declaro la clase
class Producto {
    #defino algunas propiedades
    public $nombre;
    public $precio;
    protected $estado;

    #defino el método set_estado_producto()
    protected function set_estado_producto($estado) {
        $this->estado = $estado;
    }

    # constructor de la clase
    function __construct() {
        $this->set_estado_producto('en uso');
    }

    # destructor de la clase
    function __destruct() {
        $this->set_estado_producto('liberado');
        print 'El objeto ha sido destruido';
    }
}
```

Otros métodos mágicos

PHP nos ofrece otros métodos mágicos tales como `__call`, `__callStatic`, `__get`, `__set`, `__isset`, `__unset`, `__sleep`, `__wakeup`, `__toString`, `__invoke`, `__set_state` y `__clone`.

Puede verse una descripción y ejemplo de su uso, en el sitio Web oficial de PHP:

http://www.php.net/manual/es/language.oop5.magic.php