UTN.BA FACULTAD REGIONAL BUENOS AIRES
SECRETARÍA DE EXTENSIÓN UNIVERSITARIA FRBA UTN

Centro de e-Learning

EXPERTO UNIVERISTARIO EN MySQL Y PHP NIVEL AVANZADO

www.sceu.frba.utn.edu.ar/e-learning

Módulo I

PHP AVANZADO

Programación orientada a objetos.

Conceptos básicos.

Creación de clases, métodos, propiedades y objetos.

Presentación de la Unidad:

En esta unidad aprenderán los conceptos indispensables para comprender el paradigma de programación orientado a objetos.

Cual es el objetivo y cuales son las ventajas de programar bajo esta modalidad.

Aprenderán a definir una clase con sus métodos y propiedades y a instanciar un objeto.

Objetivos:

- Aprender los conceptos de clase, objeto, método y propiedad.
- Aprender a definir una clase con sus métodos y propiedades y a instanciar un objeto.
- Aprender clasificar ya sea a los métodos o propiedades en públicos, privados y protegidos para obtener el grado de encapsulamiento deseado.

Temario:

¿Qué es la Programación Orientada a Objetos?
Clases, utilidad y definición
Los objetos como instancias de las clases.
__construct y __destruct
El operador $this.
Propiedades y Métodos estáticos
Encapsulamiento en PHP5
Herencia
Ejemplo práctico Libro de visitas

CONSIGNAS PARA EL APRENDIZAJE COLABORATIVO

En esta Unidad los participantes se encontrarán con diferentes tipos de consignas 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:

1. Los foros asociados a cada una de las unidades.
2. La Web 2.0.
3. Los contextos de desempeño de los participantes.

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

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.

Pensar en términos de objetos es muy parecido a cómo lo haríamos en la vida real. Por ejemplo vamos a pensar en un coche para tratar de modelizarlo en un esquema de POO. Diríamos que el coche es el elemento principal que tiene una serie de características, como podrían ser el color, el modelo o la marca. Además tiene una serie de funcionalidades asociadas, como pueden ser ponerse en marcha, parar o estacionar.

Los cuatro primeros términos, mas básicos, que tenemos que aprender cunado pensamos en un paradigma de programación orientado a objeto son:

- clase,
- objeto,
- propiedades
- y métodos.

La clase contiene la definición del objeto, indicando cuales son sus propiedades o atributos y cuales son sus funcionalidades o métodos.

En el ejemplo del coche entonces tendríamos por un lado la clase coche que contiene como propiedades (atributos) el color y la marca, y como métodos (funcionalidades) encender, parar o estacionar.

Veamos estos términos uno por uno con mas ejemplos.

Clases, utilidad y definicion

La clase es la base de la POO. La clase es utilizada para definir un objeto. Dentro de una clase declaramos los atributos (propiedades) y métodos que serán utilizados por el objeto.

Así podemos decir que la clase es el conjunto de declaraciones que van a definir los tipos de objetos que utilizaremos en el futuro.

Este es un ejemplo sencillo para ver como tenemos que definir una clase y cual es la sintaxis que tenemos que utilizar.

```
1. //definición de la clase polígono
2. class polígono{
3.
4.    public $vertices;
5.    public $color;
6.
7.    function area()
8.    {
9.      //instrucciones a ejecutar
10.        }
11.
12.    }
```

Los objetos como instancias de las clases.

Como hemos comentado anteriormente, las clases son tan solo la definición del objeto, así que para poder trabajar con ellas es necesario instanciarlas mediante la creación de un objeto. Así el objeto es la representación útil de la clase que lo define.

Para instanciar una clase y así crear el objeto lo haremos de la siguiente forma:

```
1. $cuadrado = new Poligono();
2. $triangulo = new Poligono();
```

__construct y __destruct

A continuación vamos a hablar de la primera novedad que incorpora php5, los métodos mágicos: **__construct** y **__destruct**.

Como ya podemos imaginar son métodos que nos sirven para definir el constructor y destructor en una clase.

El constructor de una clase es utilizado para inicializar la misma, así podremos asignar algunos valores a las propiedades que se crean convenientes en el momento de creación de un objeto.

El destructor sin embargo es ejecutado en el momento en el que el objeto deja de existir. Así aquellas tareas que queramos ejecutar en el momento de liberar un objeto las definiremos en el destructor de la clase.

Cabe aclarar que ninguno de estos dos métodos son obligatorios, pueden estar como no en la definición de una clase.

A continuación vamos a extender nuestro ejemplo anterior con la inclusión del constructor y destructor.

```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);
  ?>
```

El operador $this.

Si nos fijamos en el ejemplo anterior podemos encontrar el operador $this, este operador es para referirse al objeto sobre el que se está ejecutando el método.

Así dentro de una clase para acceder a los métodos y propiedades propios siempre lo haremos a través de $this->metodo/propiedad

Propiedades y Metodos estáticos

Dentro de una clase podremos definir propiedades y métodos estáticos mediante la palabra clave **"static".** Esto quiere decir que estos métodos o propiedades podrán ser accesibles sin necesidad de instanciar la clase, es decir, que podemos acceder a ellos sin la obligación de crear el objeto correspondiente.

La forma de definir métodos o propiedades estáticos es sencilla, basta con añadir la palabra clave **static** justo después del modificador de acceso en la declaración de dicho método o propiedad.

A continuación un sencillo ejemplo práctico de su utilización, presten atención a la manera de acceder al método estático:

```
1. <?php
2.
3. class Mensajes{
4.    (...)
5.    public static function mostrar()
6.    {
7.      echo 'Hemos accedido a este método estático';
8.    }
9.    (...)
10.      }
11.
12.      /*
13.      Observen que para acceder a métodos/propiedades
   vemos que difiere de la forma habitual a cuando
   accedemos a ellos mediante la creación del objeto
14.      */
15.      echo Mensajes::mostrar();
16.
17.      ?>
```

Encapsulamiento en PHP5

El encapsulamiento es una de las características fundamentales en la programación orientada a objetos.

Así que a continuación vamos a hablar de **public**, **private** y **protected**, que son los modificadores de acceso a métodos y propiedades incorporados a partir de PHP5.

Vamos a comentar cada uno de los modificadores, analizando sus características y adjuntando un ejemplo de su implementación.

Modificador public

Este modificador es el mas permisivo de los 3, puesto que indica que el método o la propiedad es accesible desde la clase y desde cualquier otra parte de nuestro programa.

Además es el modificador aplicado por defecto en caso de que no especifiquemos ningún modificador.

Modificador private

Este modificador establece el nivel mas restrictivo, puesto que las propiedades o métodos que declaremos como **private** solo serán accesibles desde el interior de la clase.

Modificador protected

Este modificador establece un nivel de restricción medio, esto quiere decir que las propiedades o métodos declarados como **protected** solo serán accesibles desde la clase base o las clases hijas que hereden de la clase base.

Vamos a ver a continuación el concepto de Herencia para que se pueda entender mejor este modificador.

Herencia

La herencia es otro de los pilares fundamentales dentro de la programación orientada a objetos. La herencia consiste en la definición de una clase a partir de otra existente,

así esta nueva clase se le llama clase extendida o clase derivada.

La importancia de la herencia reside en que cuando creamos una clase extendida, esta a su vez hereda todos los métodos y propiedades de su clase base o ancestra, pudiendo a su vez implementar otros métodos o propiedades exclusivos de nuestra clase extendida.

Vamos a trabajar sobre un ejemplo para que comprendan mejor los conceptos de Herencia y Encapsulamiento.

Pensemos en un negocio que vende películas en DVD y juegos para distintas consolas como pueden ser PlayStation, Wii o Xbox.

Vamos a pensar en una estructura como la siguiente, por un lado tendremos la clase soporte que tiene las propiedades y métodos comunes a ambos soportes, los dvd y los juegos. Luego tendremos dos clases que extienden a soporte que son las clases dvd y juego con las propiedades y métodos particulares de cada tipo de soporte.

Comencemos con la definición de la clase soporte:

```
<?php
class soporte{
   public $titulo;
   protected $numero;
   private $precio;

   function __construct($tit,$num,$precio){
      $this->titulo = $tit;
      $this->numero = $num;
      $this->precio = $precio;
   }

   public function dame_precio_sin_iva(){
      return $this->precio;
   }

   public function dame_precio_con_iva(){
      return $this->precio * 1.21;
   }

   public function dame_numero_identificacion(){
      return $this->numero;
   }

   public function imprime_caracteristicas(){
       echo $this->titulo;
       echo "<br>" . $this->precio . " (IVA no incluido)";
   }
```

```
}

?>
```

Solo la propiedad titulo va a a poder ser accedida desde fuera de la clase, la propiedad precio solo se podrá acceder de la clase soporte y la propiedad numero podrá ser accedida tanto de la clase soporte como de las clases dvd y juego, ya que como dijimos anteriormente estas clases extenderán a soporte.

Tomemos el script imprimesoporte.php el cual

```
<?php
require ('soporte.php');

$soporte1 = new soporte("Los Intocables",22,6);
echo "<b>" . $soporte1->titulo . "</b>";
echo "<br>Precio: " . $soporte1->dame_precio_sin_iva() . "
pesos";
echo "<br>Precio IVA incluido: " . $soporte1-
>dame_precio_con_iva() . " pesos".'<br/>';

$soporte2 = new soporte("Lo que el viento se llevo",35,6);
$soporte2->imprime_caracteristicas();
echo "<br>Precio IVA incluido: " . $soporte1-
>dame_precio_con_iva() . " pesos";

?>
```

En él estamos instanciando dos veces la clase soporte, quiere decir que estamos creando dos objetos de la clase soporte 2 veces.

Una vez instanciado el objeto podemos hacer referencia a sus propiedades, siempre que estas se hayan definido con el modificador public, a través del operado -> como en la línea en la que se muestra el titulo:

```
echo "<b>" . $soporte1->titulo . "</b>";
```

En el caso de las propiedades definidas como private o protected no podremos llevar a cabo esa instrucción, no podremos hacer por ejemplo desde este script:

```
echo "<b>" . $soporte1->precio . "</b>";
```

En el caso de las propiedades definidas como private como no podemos acceder directamente a la misma lo que hacemos es definir un método public que nos devuelve el valor de dicha propiedad como en el caso de los métodos:

```
$soporte1->dame_precio_con_iva()
$soporte1->dame_precio_sin_iva()
```

Lo mismo sucedería con los métodos. Es decir un método definido como private no podrá ser accedido desde fuera de la clase. Si hubiéramos definido un método como private no sería lícito invocar a dicho método desde fuera de la clase de la siguiente forma:

```
$soporte1->método()
```

Solo podríamos utilizarlo desde dentro de la clase de la siguiente manera:

```
$this->método()
```

Veamos ahora la definición de la clase dvd:

```
<?php

class dvd extends soporte{
   public $idiomas_disponibles;
   private $formato_pantalla;

   function
__construct($tit,$num,$precio,$idiomas,$pantalla){
      parent::__construct($tit,$num,$precio);
      $this->idiomas_disponibles = $idiomas;
      $this->formato_pantalla = $pantalla;
}

   public function imprime_caracteristicas(){
      echo "Película en DVD:<br>";
      parent::imprime_caracteristicas();
      echo "<br>" . $this->idiomas_disponibles;
       echo "<br>" . $this->formato_pantalla;
   }
}
?>
```

Como dijimos las clases dvd y juego extienden a la clase soporte eso lo expresamos en la definición de la clase a través de la palabra extends soporte.

Esta clase entonces tendrá todas las propiedades y métodos de la clase soporte mas los definidos en la clase dvd.

El hecho de que la clase herede las propiedades de soporte no implica que el constructor de la clase soporte se ejecute automáticamente cuando se instancia la clase dvd. Es por eso que en el constructor de la clase dvd se llama al constructor de la clase soporte. La sintaxis para invocar ese método es:

```
parent::__construct($tit,$num,$precio);
```

Indicando que ejecute el constructor de la clase padre.

Lo mismo sucede en el método imprime_caracteristicas(), invoca al método imprime_caracteristicas de la clase padre, es decir de soporte.

Una vez instanciado un objeto dvd será lícito hacer

```
$midvd = new dvd('El Padrino',25,5,'español-ingles-
frances','4:3 y 16:9');
$midvd->dame_precio_con_iva();
```

Ya que el método dame_precio_con_iva() es heredado por la clase dvd.

Veamos el código de imprimedvd.php

```
<?php
require ('soporte.php');
require ('dvd.php');

$midvd = new dvd('El Padrino',25,5,'español-ingles-
frances','4:3 y 16:9');
$midvd->imprime_caracteristicas();
?>
```

De manera similar veamos la definición de la clase juego:

```
<?php

class juego extends soporte{
   public $consola; //nombre de la consola del juego ej:
playstation
   private $min_num_jugadores;
   private $max_num_jugadores;

   function
__construct($tit,$num,$precio,$consola,$min_j,$max_j){
      parent::__construct($tit,$num,$precio);
      $this->consola = $consola;
      $this->min_num_jugadores = $min_j;
      $this->max_num_jugadores = $max_j;
   }

   public function imprime_jugadores_posibles(){
      if ($this->min_num_jugadores == $this-
>max_num_jugadores){
         if ($this->min_num_jugadores==1)
         echo "<br>Para un jugador";
         else
         echo "<br>Para " . $this->min_num_jugadores . "
jugadores";
```

```
        }else{
         echo "<br>De " . $this->min_num_jugadores . " a " .
$this->max_num_jugadores . " Jugadores.";
        }
    }

    public function imprime_caracteristicas(){
        echo "Juego para: " . $this->consola . "<br>";
        parent::imprime_caracteristicas();
        echo "<br>" . $this->imprime_jugadores_posibles();
    }
}

?>
```

Y de imprimejuego.php

```
<?php
require ('soporte.php');
require ('juego.php');

$mijuego = new juego("Final Fantasy", 21, 2.5,
"Playstation",1,1);
$mijuego->imprime_caracteristicas();

//esta línea daría un error porque no se permite acceder a
un atributo private del objeto
//echo "<br>Jugadores: " . $mijuego->min_num_jugadores;
//habria que crear un método para que acceda a los
atributos private
$mijuego->imprime_jugadores_posibles();

echo "<p>";
$mijuego2 = new juego("GP Motoracer", 27, 3, "Playstation
II",1,2);
echo "<b>" . $mijuego2->titulo . "</b>";
$mijuego2->imprime_jugadores_posibles();


?>
```

Los invito a que hagan pruebas cambiando los modificadores y agregando llamados a los distintos métodos y propiedades para que puedan comprobar todos estos conceptos que hemos explicado.

Ejemplo práctico Libro de visitas

Tomando como base los conceptos que vimos en la Unidad cuando decimos que pensar en términos de objetos es muy parecido a cómo lo haríamos en la vida real. Y que podemos pensar en un coche como el elemento principal que tiene una serie de características, como podrían ser el color, el modelo o la marca y tiene una serie de funcionalidades asociadas, como pueden ser ponerse en marcha, parar o aparcar.

Y además teniendo en cuenta que uno de los objetivos fundamentales de la POO es la reutilización de código vamos a pensar en un Libro de Visitas y lo desarrollamos bajo estos conceptos de POO.

Vamos a tener por un lado una clase Base de Datos que tendrá todo lo que tenga que ver con la interacción con la base de datos, de esta manera esta clase podrá ser reutilizada por cualquier sistema que desarrollemos. La complejidad de la interacción con la base estará encapsulada dentro de esta clase y nosotros no tendremos mas que instanciarla como corresponde y utilizarla. Así luego en un carrito de compras, en un sistema de encuestas, en un ABM o en cualquier otro tipo de sistema que se les ocurra podremos utilizar esta misma clase para interactuar con la base.

Y luego tendremos una clase Comentario que tendrá asociados métodos como insertar_comentario() o get_comentarios() para insertar un comentario nuevo en la tabla o para recuperar todos los comentarios.

Si miramos el código del ejemplo, lo que hacemos es, instanciar un objeto Base de Datos y luego instanciamos un objeto Comentario pasándole como parámetro un objeto Base de Datos.

De esta forma, como se darán cuenta, la clase Base de Datos puede ser reutilizada en cualquier sistema. O mismo dentro de éste si tuviéramos otra clase Usuario, Producto o lo que se les ocurra podrá utilizar también esta misma clase Base de Datos.

No hace falta comentar mucho mas con respecto al resto del código. Podrán seguirlo sin problema. Espero que el ejemplo les sirva como idea para plantear sus propios ejemplos basándose en esta metodología.