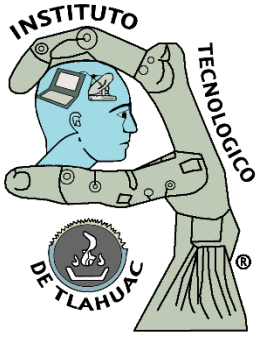
Tecnológico Nacional de México

Instituto Tecnológico de Tláhuac



Tópicos de programación de MVC

Curso PHP POO

Docente:

Roldan Aquino Segura

Alumno:

Grande López Adrián

No. de control:

16106146

Grupo:

7S2

Fecha:

5 de noviembre de 2020

Índice

Introducción

El siguiente curso tiene como fin obtener el conocimiento para poder realizar las tareas de creación y uso de clases, métodos, objetos y propiedades orientadas a PHP con POO (Orientado a objetos).

Así como la realización de una práctica para comprobar los temas estudiados.

PHP y POO

clases.php

La forma de declarar una clase es inicializando con la palabra reservada “class” seguido del nombre de la clase que debes iniciar con minúsculas, una vez iniciada dentro de los paréntesis se colocan los parámetros como variables, atributos o métodos. No existe un límite en la creación de clases.

```php
<?php
/*
Una clase es un molde de trabajo
para los metodos y atributos.
*/

    class miClase{
        //definir los atributos
        //definir los metodos
    }

    class MiClase2{

    }
        /*
        podemos declarar cualquier numero
        de clases dentro de un archivo php
        */
?>
```

metodosyatributos.php

Para declarar un atributo se inicia con un identificador seguido del nombre del que llevara el atributo empezando con el símbolo “$”, recordando que php no es un lenguaje tipado no requiere especificar el tipo de atributo que será.

La declaración de un método empieza por un identificador, seguido de la frase reservada “function” que es la encargada de declarar el método y el nombre del método en caso de llevar su constructor se indica dentro de los paréntesis. Dentro del método puede llamarse atributos con la pseudovariable “this” instanciado por un return.

```php
<?php

    class miClase{
        //Declaracion de atributos
        public $atributo1="hola mundo";

        //Declaracion de metodos
        public function miMetodo(){
            /*
            Llamar a un atributo
            utilizando la pseudo variable "this"
            */
            return $this->atributo;
        }
    }

?>
```

objetos.php

Los objetos tienen la función de instanciar clases o métodos, dichos objetos se crean con un nombre para el objeto iniciado con el símbolo “$” y seguido de la palabra “new” con el nombre de la clase u objeto a llamar.

En el ejemplo se observa un método con un constructor, esos datos no están declarados y deben ser agregados cuando se realiza el “echo” del objeto creado.

```php
<?php
    class miClase{
        //Atributos
        public $resultado=0;

        //Metodo
        public function miMetodo($v1,$v2){
            $this->resultado=$v1 + $v2;
            return $this->resultado;
        }
    }
    //Fin de clase

    //Instanciar o declarar un objeto
    $miObjeto= new miClase();
    echo $miObjeto->miMetodo(5,10);
?>
```

retornodatos.php

Cada método cuenta con una función “return” el cual se encarga de retornar el dato que se desee, siempre y cuando este sea llamado con una pseduovariable o directamente señalado después de la palabra return.

Esta función solo retorna datos, pero no puedo mostrarlos, para esta función se debe usar en “echo” para realizar la acción.

Para el siguiente ejemplo podemos observar que se realizó una serie de métodos con diferentes datos (String, entero, arreglo y json), cada método cuenta con su constructor de acuerdo al dato asignado y que se retornara.

```php
class retornoDatos{

    public function retornaString($edad){ //Tipo String
        if($edad > 18){
            return "Es mayor de edad";
        }else{
            return "No es mayor de edad";
        }
    }

    public function retornaEntero($valor){  //Tipo entero
        if($valor > 0){
            return 1;
        }else{
            return 0;
        }
    }

    public function retornaArreglo($ciclos){    //Tipo arreglo
        $datos=array();

        for ($i=0; $i < $ciclos ; $i++) {
            $datos[$i]=$i;
        }

        return $datos;
    }
    public function retornaJson(){  //Tipo json
        $arr=array(
            "hdd" => 500,
            "pantalla" => 21,
            "ram" => 8
                );

        return json_encode($arr);
    }
}
```

Para poder hacer una vista de los datos retornados, debemos crear un objeto y realizar un “echo” para poder visualizarlo.

```php
$cadena= new retornoDatos();
var_dump($cadena->retornaString(20));
echo "<br>";
var_dump($cadena->retornaEntero(1));
echo "<br>";
var_dump($cadena->retornaArreglo(10));
echo "<br>";
var_dump($cadena->retornaJson());
```

Para crear el objeto se crea una variable y se procede a instanciar la clase. En este caso al ser más de un método, para visualizarse se usará un “var_dump” que su función es mostrar información de la variable (tipo y tamaño de la variable); en cada uno se instanciara al objeto, después se llama al método con el dato correspondiente de su constructor.

La vista de los datos retornado queda con la información del tipo de datos retornado y el tamaño de esta misma.

```
string(16) "Es mayor de edad"
int(1)
array(10) { [0]=> int(0) [1]=> int(1) [2]=> int(2) [3]=> int(3) [4]=> int(4) [5]=> int(5) [6]=> int(6) [7]=> int(7) [8]=> int(8) [9]=> int(9) }
string(33) "{"hdd":500,"pantalla":21,"ram":8}"
```

Calculadora con php

Para conocer el uso de la clases y aplicaciones se realizó una calculadora con php.

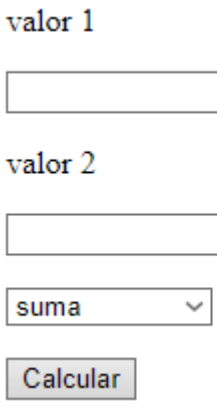
index.php

Se realiza una plantilla de tipo HTML y se utilizara un form del método “POST” dirigido a calcular.php que será creada a continuación.

Dentro del formulario se insertan los “label” para agregar los valores, el “input” respectivo a cada uno para recibir valores, un menú “option” para seleccionar la acción y un “button” para enviar los valores del formulario.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Calculadora con PHP</title>
    </head>
    <body>
        <form action="procesar.php" method="post">
            <label>valor 1</label>
            <p></p>
            <input type="text" name="val1">
            <p></p>
            <label>valor 2</label>
            <p></p>
            <input type="text" name="val2">
            <p></p>
            <select name="opcion">
                <option value="suma">suma</option>
                <option value="resta">resta</option>
                <option value="division">division</option>
                <option value="multiplicacion">multiplicacion</option>
            </select>
            <p></p>
            <button>Calcular</button>
        </form>
    </body>
</html>
```

Esto nos genera una vista en el navegador sencilla, debido a que no cuenta con el uso de hojas de estilo.



claseCalculo.php

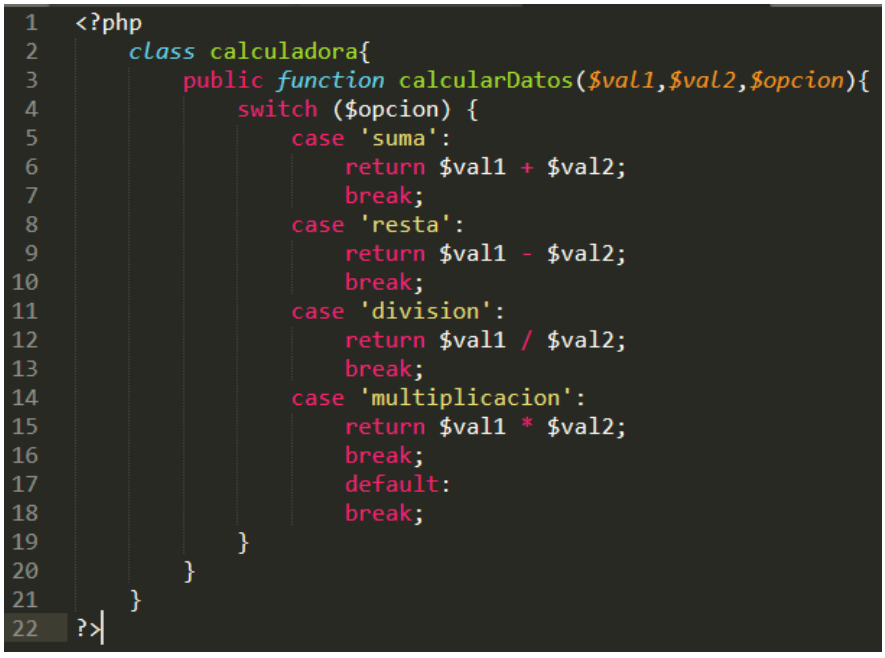
Se realiza una clase para realizar los cálculos de la calculadora.

Primero se crea una clase con el nombre “calculadora” con el método “calcularDatos” con un constructor de 3 variables (“$val1”, “$val2”, “$opcion”), para trabajar cada operación, se usará un switch usando la variable “$opcion” como identificador.

Cada opción representara una operación:

- suma
  - Sumará las variables “$val1” y “val2”.
- resta
  - Restará las variables “$val1” y “val2”.
- division
  - Dividirá las variables “$val1” y “val2”.
- multiplicación
  - Multiplicará las variables “$val1” y “val2”.

Default no contara con código, ya que para el caso no tiene un uso.



```php
<?php
    class calculadora{
        public function calcularDatos($val1,$val2,$opcion){
            switch ($opcion) {
                case 'suma':
                    return $val1 + $val2;
                    break;
                case 'resta':
                    return $val1 - $val2;
                    break;
                case 'division':
                    return $val1 / $val2;
                    break;
                case 'multiplicacion':
                    return $val1 * $val2;
                    break;
                    default:
                    break;
            }
        }
    }
?>
```

procesar.php

En esta clase se recibirán los datos del formulario por el método “POST”.

Para iniciar con una plantilla PHP se realizó un “requiere_once”, para llamar al archivo de “calculoClases.php”.

Se instancia un nuevo objeto de clase con la variable “$calcular” haciendo referencia al método “calculadora”.

Se crean los métodos “POST” para la recepción de datos de las variables y asignadas a las variables determinadas, se realiza un “echo” para imprimir en pantalla los datos.

```php
<?php
    require_once "claseCalculo.php";
    $calcular = new calculadora();
    $val1=$_POST['val1'];
    $val2=$_POST['val2'];
    $opcion=$_POST['opcion'];

    echo $calcular->calcularDatos($val1,$val2,$opcion);
?>
```

Propiedad self

Esta propiedad se utiliza para llamar métodos dentro de otros métodos sin la necesidad e instanciar de nuevo.

self.php

Se utiliza una clase “Metodos”, con dos métodos “mandarColor”, con un código de if y “daDeAltaColor” para procesar los datos del método anterior.

```php
<?php
    class Metodos{
        public function mandarColor($valor){
            if($valor==1){
                return "rojo";
            }else if($valor==2){
                return "negro";
            }else if($valor==3){
                return "azul";
            }
        }

        public function darDeAltaColor($tipocolor){
            return self::mandarColor($tipocolor);
        }
    }
    $obj= new Metodos();
    echo $obj->darDeAltaColor(2);
?>
```

Para evitar instanciar el objeto se usa la función “self::” y servirá de la misma manera que se instanciara un objeto, ahorrando líneas de código y usándolo más veces.

Herencia y propiedad parent

La herencia sirve para obtener las propiedades de una clase padre a otra clase hijo, todo mediante la palabra reservada “extends”

herencia.php

Se crea una clase llamada “clasePadre” con un método simple y una “claseHijo”, donde se le agrego la palabra reservada “extends”, de la siguiente manera, “claseHijo” heredo las propiedades del metodo “clasePadre”.

La función parent es parecida a la herencia, su función es llamar un método de otra clase aun cuando estos lleven el mismo nombre.

```php
<?php
    class clasePadre{
        public function metodoPadre(){
            return "hola desde el padre";
        }
        public function metodo1(){
            return "este es metodo padre";
        }
    }

    class claseHijo extends clasePadre{
        public function metodohijo(){
            return parent::metodo1();
        }
        public function metodo1(){
            return "este es metodo hijo";
        }
    }
    /*
    La herencia es obtener todas las propiedades de una clase a otra
    mediante la palabra reservada extends
    */
    //instancia rapida o de doble puntuacion
    echo claseHijo::metodohijo();
?>
```

Metodos privados

Un método privado no puede ser accedido o instanciado desde fuera de la clase que fue creado. Debe manejarse dentro de los métodos de la misma clase.

privado.php

Dentro de la clase se ubican 2 métodos, uno de manera privado con “private” y un público con “public”.

```php
<?php
    /*
    Un metodo privado , solo puede ser accedido
    en la misma clase donde fue creado, al mismo tiempo
    no puede mostrarse fuera o en instancia.
    Debe manejarse dentro de un metodo de la misma clase
    */
    class clase1{
        private function saludar(){
            return "saludando";
        }
        public function mandaSaludo(){
            return self::saludar();
        }
    }
    echo clase1::saludar();
?>
```

Cuando el método privado es llamado o instanciado este genera un error.

Fatal error: Uncaught Error: Call to private method clase1::saludar() from context '' in C:\xampp\htdocs\TopicosMVC\Curso PHP POO\privados.php:16 Stack trace: #0 {main} thrown in **C:\xampp\htdocs\TopicosMVC\Curso PHP POO\privados.php** on line **16**

Como se observa el error en la línea 16 del código fue cuando se instancio el método privado. Dado que solo puede ser usado en el mismo método que fue creado y no fuera de él.

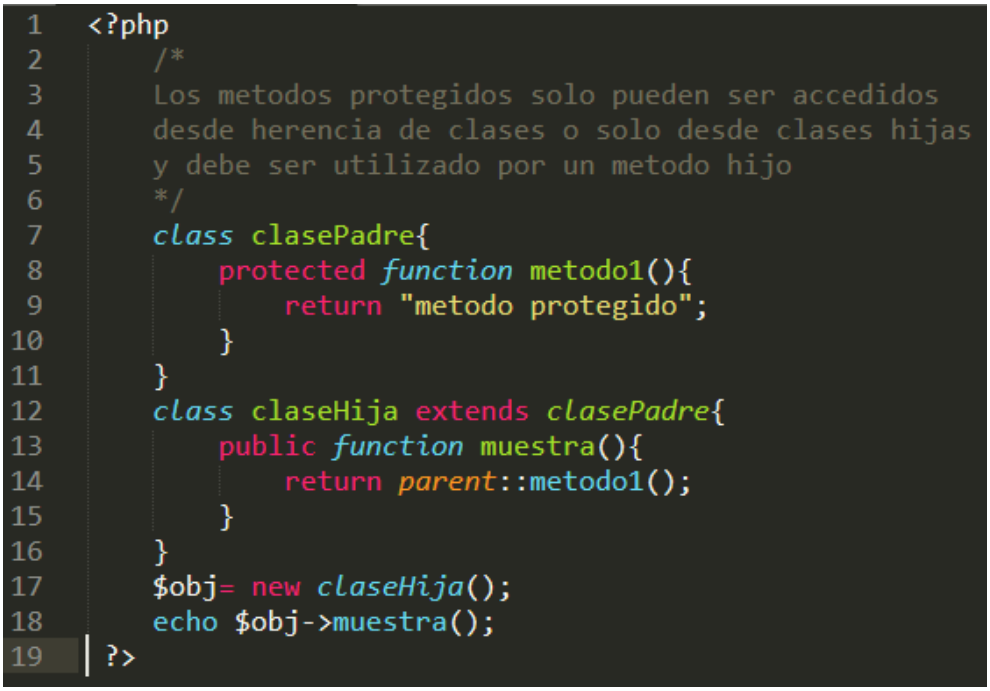
Métodos protegidos

Los métodos protegidos solo pueden ser usando herencia de entre los métodos de una misma clase o entre hijas y debe ser utilizado solo por clases hijas.

protected.php

Se utiliza una clase “clasePadre” con un método retornando datos. La clase “claseHija“ se le aplica un extends de la clase anterior y se le aplica un parent al método de la clase anterior.

Para llamarlo debemos instanciar el objeto de la clase que heredo el método protegido, en este caso “claseHija” y llamar el método que está en esta clase, de esa manera, lo que se mostrara será lo ingresado en el metodo de la clase “clasePadre”.



```php
<?php
    /*
    Los metodos protegidos solo pueden ser accedidos
    desde herencia de clases o solo desde clases hijas
    y debe ser utilizado por un metodo hijo
    */
    class clasePadre{
        protected function metodo1(){
            return "metodo protegido";
        }
    }
    class claseHija extends clasePadre{
        public function muestra(){
            return parent::metodo1();
        }
    }
    $obj= new claseHija();
    echo $obj->muestra();
?>
```

Metodos estáticos

Son funciones estructuras de php que se mantienen en una clase y sirven para mandar a llamar un proceso únicamente, no pueden acceder propiedades y no se puede acceder a métodos.

statico.php

Se utiliza clase “miclase” para crear un metodo de modo estatico con la palabra reservada “static”.

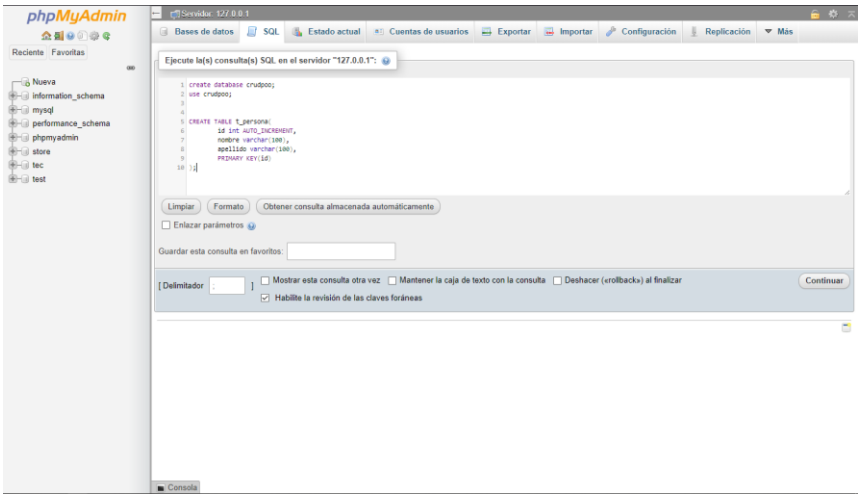
Se utiliza un variable para demostrar la forma correcta de llamar un método estático, la cual consiste en llamar un metodo en vez de la variable. Ya que al ser estático este solo devuelve datos, no los ingresa.

```php
<?php
    class miclase{
        public $mensaje=1;

        public function hola(){
            return "hola";
        }
        public static function metodo(){
            return self::hola();
        }
    }
    echo miclase::metodo();
?>
```

Conexión a MySQL con PHP y POO

Nos centramos en la conexión entre MySQL y php.

Para eso primero debemos crear la base de datos accediendo a nuestro localhost de phpMyAdmin ingresando el código “sql”.



conexion.php

Este documento consiste en realizar la conexión entre el servicio de MySQL y PHP llamando las variables de atributo privado para tener acceso a la base de datos.

Se usa un método “conexion” en la cual se llaman las variables de nuevo. Usando la librería “mysqli_conect” para ejecutar las variables

```php
<?php
    class conectar{
        private $servidor="localhost";
        private $usuario="root";
        private $bd="crudpoo";
        private $password="";

        public function conexion(){
            $conexion=mysqli_connect($this->servidor,
                                     $this->usuario,
                                     $this->password,
                                     $this->bd);
            return $conexion;
        }
    }
?>
```

metodosCrud.php

La clase “metodos” contendrá las metodos ocupar:

- mostrarDatos($sql)
    - Instanciar un objketo “c” para la clase conectar, para realizar la conexión
    - Mediante una variable “$result” se usa un “mysqlli_query” para pedir una “$conexión” y un “$sql” que se pide en el constructor.
    - Para el retorno de datos se usa “mysql_fetch_all” el cual pide “$result” y “MYSQL_ASSOC” para entregar los datos en un arreglo asociativo.
- insertarDatosNombres($datos)
    - Se instacia el objeto “c” para la conexión de la base de datos.
    - Se crea la variable “$sql” que será el “insert” de los datos de la tabla
    - Para retornar los datos se usa “$result” con un “mysqli_query” para pedir una “$conexión” y un “$sql”.
- actualizaDatosNombre($datos)
    - Objeto “$c” instanciado para la conexión.
    - Variable “$sql” para realizar el update en la tabla de las columnas a modificar.
    - Retorno de datos se mediante “$result” con un “mysqli_query” para pedir la “$conexión” y “$sql”.
- eliminarDatosNombre($id)
    - Conexión instanciada por el objeto “$c”.
    - Variable “$sql” para eliminar los dato seleccionados.
    - Retorno de datos usando una variable “$result” solicitando un “mysqli_query” para pedir la “$conexión” y “$sql”.

```php
<?php
    class metodos{
        public function mostrarDatos($sql){
            $c= new conectar();
            $conexion=$c->conexion();

            $result=mysqli_query($conexion,$sql);

            return mysqli_fetch_all($result,MYSQLI_ASSOC);
        }
        public function insertarDatosNombre($datos){
            $c= new conectar();
            $conexion=$c->conexion();

            $sql="INSERT into t_persona (nombre,apellido)
                                values ('$datos[0]','$datos[1]')";

            return $result=mysqli_query($conexion,$sql);
        }

        public function actualizaDatosNombre($datos){
            $c= new conectar();
            $conexion=$c->conexion();

            $sql="UPDATE t_persona set nombre='$datos[0]',
                                        apellido='$datos[1]'
                                    where id='$datos[2]'";
            return $result=mysqli_query($conexion,$sql);

        }
        public function eliminarDatosNombre($id){
            $c= new conectar();
            $conexion=$c->conexion();
            $sql="DELETE from t_persona where id='$id'";
            return $result=mysqli_query($conexion,$sql);
        }
    }
?>
```

index.php

Sera la plantilla php que se visualiza en la vista del navegador.

Se usa el objeto “$bjt” para llamar la clase metodos y a su vez usar “mostraDatos”; definida la variable “$sql” con el código sql de la tabla. Haciendo uso del for each mostramos los datos obtenidos de la variable “$datos” como key.

Para ingresar más datos; será un cuestionario, con la acción de “procesos/insertar” con un metodo de “post”. Cuenta con 2 campos y un botón, un campo de nombre y un campo de apellidos, el botón es de tipo “summit”.

Para actualizar los datos, se usará el archivo “editar.php” con la estructura del formulario y se pedirá el “id”.

Para la eliminación de datos se usará el archivo “elimnar.php”.

```php
<?php
    require_once "conexion.php";
    require_once "metodosCrud.php";
?>
<!DOCTYPE html>
<html>
    <head>
        <title>crud</title>
    </head>
    <body>
    <form action="procesos/insertar.php" method="post">
        <label>Nombre</label>
        <p></p>
        <input type="text" name="txtnombre">
        <p></p>
        <label>Apellido</label>
        <p></p>
        <input type="text" name="txtapellido">
        <p></p>
        <button>Agregar</button>
    </form>
    <br><br>
    <table style="border-collapse: collapse;" border="1">
        <tr>
            <td>Nombre</td>
            <td>Apellido</td>
            <td>Actualizar</td>
            <td>Eliminar</td>
        </tr>
        <?php
            $obj= new metodos();
            $sql="SELECT id,nombre,apellido from t_persona";
            $datos=$obj->mostrarDatos($sql);

            foreach ($datos as $key ) {
         ?>
        <tr>
            <td><?php echo $key['nombre']; ?></td>
            <td><?php echo $key['apellido']; ?></td>
            <td>
                <a href="editar.php?id=<?php echo $key['id'] ?>">
                Editar
                </a>
            </td>
            <td>
                <a href="procesos/eliminar.php?id=<?php echo $key['id'] ?>">
                eliminar
                </a>
            </td>
        </tr>
    <?php
        }
     ?>
    </table>
    </body>
</html>
```

editar.php

Se requiere la conexión; se usa “require_once” para acceder al archivo “conexión.php” y se instancia la conexión con la base de datos. Mediante “GET” se obtiene la “id” obtenida de url.

Se usa un “$sql” para la edición de datos en la tabla.

Con el cuerpo del formulario se agregó la forma de visualizar los datos a editar agregando una sección php con arreglo en la posición de los datos.

```php
<?php
    require_once "conexion.php";
    $obj= new conectar();
    $conexion=$obj->conexion();
    $id=$_GET['id'];
    $sql="SELECT nombre,apellido
            from t_persona where id='$id'";
    $result=mysqli_query($conexion,$sql);
    $ver=mysqli_fetch_row($result);
?>

<!DOCTYPE html>
<html>
<head>
    <title></title>
</head>
<body>


<form action="procesos/actualizar.php" method="post">
    <input type="text" hidden="" value="<?php echo $id ?>" name="id">
    <label>Nombre</label>
    <p></p>
    <input type="text" name="txtnombre" value="<?php echo $ver[0] ?>">
    <p></p>
    <label>Apellido</label>
    <p></p>
    <input type="text" name="txtapellido" value="<?php echo $ver[1] ?>">
    <p></p>
    <button>Agregar</button>
</form>
</body>
</html>
```

insertar.php

El método de ingresar datos será por medio este archivo.

Se requiere la conexión a la base datos mediante un “require_once” hacía “conexión.php” de igual manera, se requiere los metodos y se usar el “require_once” en “metodosCrud.php”.

Se definen las variables para acceder a los datos “post” del formulario. La variable “$datos” se usará para almacenar los datos recibidos del formulario. Se instancia un objeto “$obj” para llamar a metodos; ya que pide “$datos” el método, se integra un if para regresar al sitio “index.php” y en caso de fallar, enviara error.

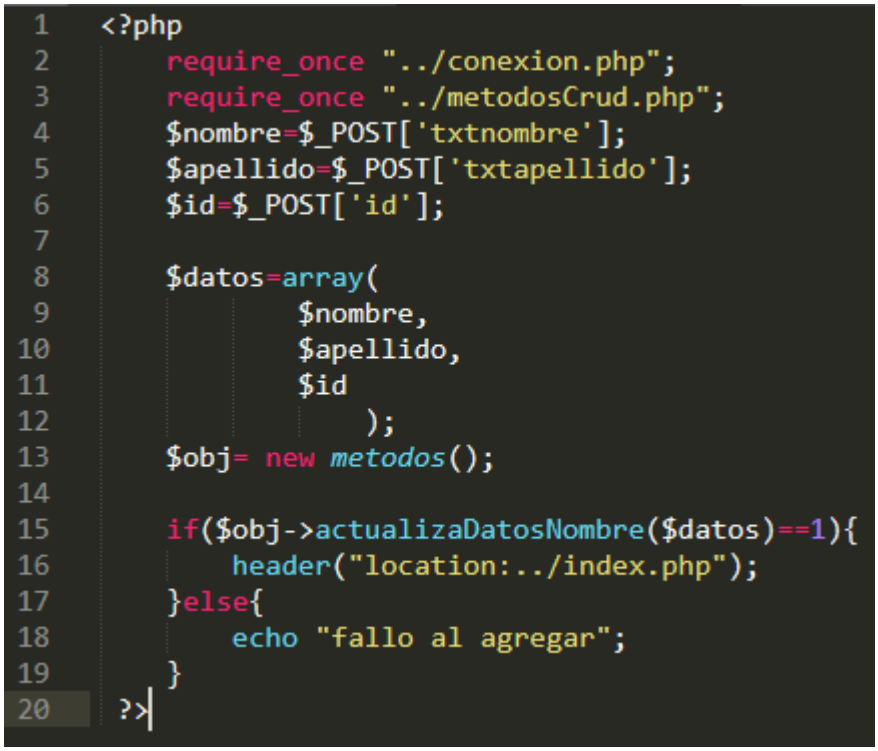
```php
<?php
    require_once "../conexion.php";
    require_once "../metodosCrud.php";
    $nombre=$_POST['txtnombre'];
    $apellido=$_POST['txtapellido'];

    $datos=array(
            $nombre,
            $apellido
                );
    $obj= new metodos();
    if($obj->insertarDatosNombre($datos)==1){
        header("location:../index.php");
    }else{
        echo "fallo al agregar";
    }
?>
```

actualizar.php

El metodo para editar datos será mediante este archivo.

Se requiere la conexión a la base datos mediante un “require_once” hacía “conexión.php” de igual manera, se requiere los metodos y se usar el “require_once” en “metodosCrud.php”.

Se definen las variables del metodo “post”; con la diferencia que se solicita la “id”, agregando una variable “$id” del metodo “post”. La variable “$datos” dirijida para almacenar los datos en forma de arreglo se compara con if que tiene instanciado un objeto a la clase “metodosCrud.php” que es el encargado de realizar la operación y se encarga de retornar al archivo “index.php”; “else” mandara un mensaje de error.



```php
<?php
    require_once "../conexion.php";
    require_once "../metodosCrud.php";
    $nombre=$_POST['txtnombre'];
    $apellido=$_POST['txtapellido'];
    $id=$_POST['id'];

    $datos=array(
            $nombre,
            $apellido,
            $id
                );
    $obj= new metodos();

    if($obj->actualizaDatosNombre($datos)==1){
        header("location:../index.php");
    }else{
        echo "fallo al agregar";
    }
?>
```

eliminar.php

El metodo para eliminar datos será mediante este archivo.

Se requiere la conexión a la base datos mediante un “require_once” hacía “conexión.php” de igual manera, se requiere los metodos y se usar el “require_once” en “metodosCrud.php”.

Se definen las variables ocupadas por el metodo “post”. Los datos son almacenados en una variable “$datos” de tipo arreglo. El if usado esta instanciado con la clase de “metodosCrud” para realizar la acción del metodo; de igual manera, realiza el regreso al archivo “index.php”; en el caso de “else” este mandara un mensaje de error.

```php
<?php
    $id=$_GET['id'];

    require_once "../conexion.php";
    require_once "../metodosCrud.php";

    $obj= new metodos();
    if($obj->eliminarDatosNombre($id)==1){
        header("location:../index.php");
    }else{
        echo "fallo al agregar";
    }
?>
```

Conclusión

Mediante el curso, la idea de poder instanciar los metodos de distintas maneras y poder aprovechar el encapsulamiento de los metodos, permite aprovechar de distita manera las variables y datos que estos se encuentran.

Anexos

GitHub

Curso PHP POO

Canal del autor