MVC + Intro a POO

Agenda

- MVC
- POO
- POO & BBDD
- POO & MVC

Model View Controller (MVC)

Model View Controller

Patrón de arquitectura de software utilizado ampliamente en la industria.

PATRÓN DE ARQUITECTURA DE SOFTWARE

Ofrecen **soluciones estándares a problemas comunes** dentro de la ingeniería de software.

Un patrón define una *estructura esencial* para un sistema de software.

Define cuáles van a ser sus componentes y cómo se relacionan entre ellos.

Un poco de historia...

MVC fue introducido a fines de la década del 70 en pleno auge de las computadoras personales con GUI.

Padre de otros patrones:

- HMVC (MVC Jerárquico)
- MVA (Modelo-Vista-Adaptador)
- MVP (Modelo-Vista-Presentador)
- MVVM (Modelo-Vista Vista-Modelo)

Hoy en día es uno de los **patrones más presentes en la industria**, ya que lo utilizan una gran cantidad de frameworks en alguna de sus variantes.

nest

laravel

BACKBONE.JS

ember

Simple MVC Framework
Applications made easy!

ANGULAR

.NET MVC

django

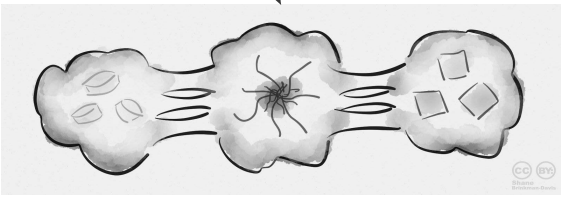
yiiframework

spring

ZEND FRAMEWORK

¿Qué problema resuelve?

Desacopla el código de programas donde toda la *lógica,* el *acceso a datos* y la *interfaz gráfica* se encuentran bajo mismos archivos sin ninguna separación clara.



¿Qué propone MVC?

Divide la lógica del programa en **tres elementos inter-relacionados**. Cada uno cumple una función determinada.

Cada componente tiene una responsabilidad determinada y trabajan de forma coordinada:

- **MODELO**: Acceso a datos
- **VISTA**: Interfaz de usuario (Front End)
- **CONTROLADOR**: Coordinador entre vista y modelo

Modelo - Responsabilidades

MODELO

En este componente manejamos la **comunicación con el modelo de datos (ej. BD)**.

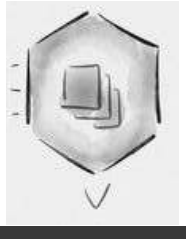
- Proteger y persistir los datos del usuario.
- Asegurar la integridad y consistencia de datos.
- Proveer métodos para:
  - Consultar datos
  - Insertar/Modificar datos
  - Borrar Datos

Vista - Responsabilidades

VISTA

En este componente generamos la **interfaz de usuario.**

- Presentar la información al usuario (front-end).
- Permitir al usuario interactuar con la aplicación



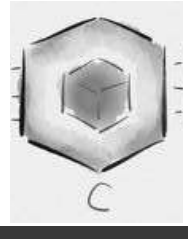
Controlador - Responsabilidades

CONTROLADOR

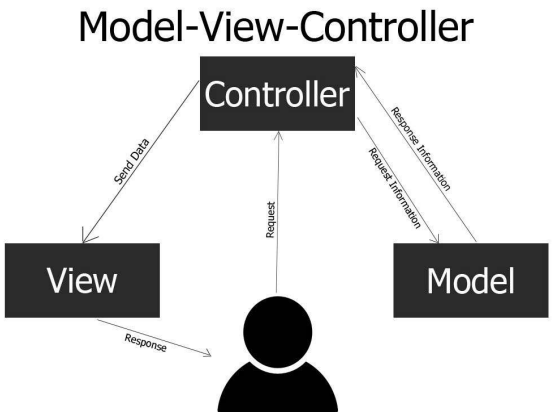
Es el **intermediario** (coordinador) entre la vista y el modelo.

- **Controla y coordina** el flujo de la aplicación.
- **Obtiene y procesa** los pedidos del usuario.
- **Valida** la entrada de datos del usuario.

EJEMPLO
Cuando una nueva entrada de un usuario llega, el **Controller** la valida y llama al **Model** para modificar los datos, luego actualiza la **View**.



MVC
Funcionamiento



Ventajas MVC

- MVC crea un sistema desacoplado
  - Reduce la complejidad de cada parte del sistema
- Permite trabajar en paralelo de forma colaborativa
  - FrontEnd
  - BackEnd
- Facilita
  - Escalabilidad
  - Mantenimiento

Desventajas MVC

- Agrega complejidad a la solución
- La estructura predefinida puede no ser lo que estábamos buscando

¿Cómo saber cuándo **no** usarlo?

- Donde hay elementos que no aplican a la tripla MVC

POO

¿Por qué POO?

- La **Programación Orientada a Objetos** (POO) es una forma de **pensar, modelar y desarrollar aplicaciones**

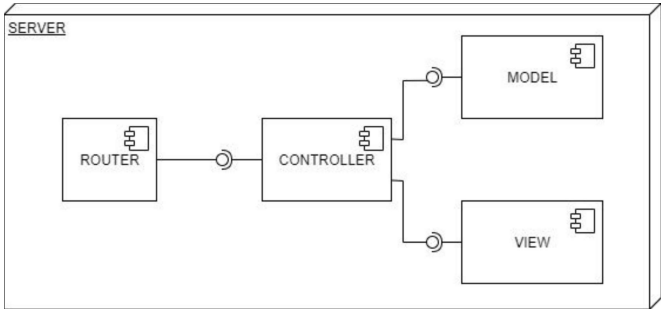
- Permiten desarrollar sistemas que puedan **crecer a gran escala** sin perder la posibilidad de ser **extendidos** y **modificados** de acuerdo a las necesidades del cliente

MVC

Diagrama de componentes

¿Cómo armar nuestro MVC?

Uno de los enfoques más clásicos es usar POO



¿Qué son los objetos?

- Podemos ver a nuestro alrededor y ver muchos objetos del mundo real: un perro, un escritorio, un televisor, una bicicleta etc…
- Cada uno de los objetos comparten dos características: **Atributos y Operaciones**

atributos: cadena, numero de cambios, cambio actual

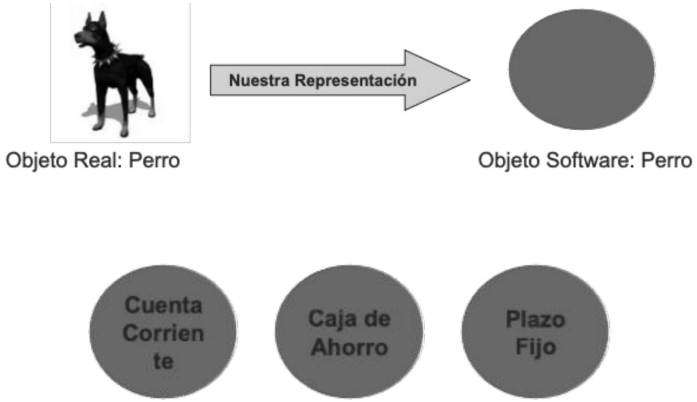
operaciones: acelerar, realizar un cambio, frenar, etc.

atributos: nombre, color, raza.

operaciones: ladrar, correr, jugar, etc.

¿Qué son los objetos?

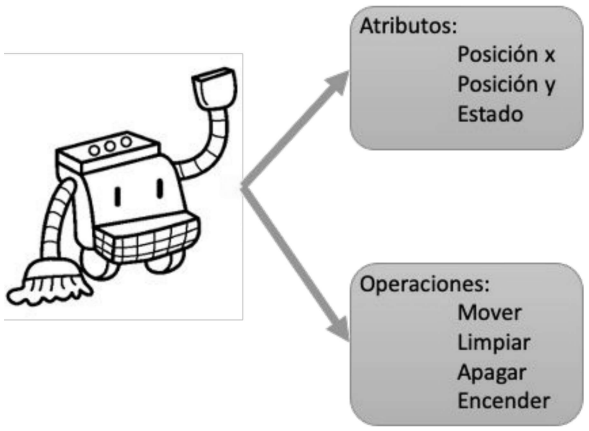
- Los objetos de software son modelos de objetos del mundo real y ellos también tienen **atributos** y **operaciones**
- Un objeto de software mantiene sus **atributos** en variables e implementa sus **operaciones** en métodos
- Podemos representar objetos del mundo real y objetos abstractos



Ejercicio de Ejemplo

- Suponga una aplicación que simula un robot limpiando una habitación
- Dicho robot conoce su posición x,y en la sala y el usuario puede moverlo a cualquier posición, simplemente indicando nuevas coordenadas
- Además, el robot tiene un “estado”. Por ejemplo, “LIMPIANDO”, “CARGAR BATERIA”, “ATASCADO”,”APAGADO”
- ¿Cómo lo modelamos?

Ejercicio de Ejemplo



Ejercicio de Ejemplo

- El paradigma orientado a objetos fomenta que tanto los datos del Robot como las operaciones que se pueden hacer sobre el Robot, estén en el mismo lugar (en el objeto Robot)
- Un objeto es, precisamente, una entidad que agrupa atributos y operaciones relacionadas entre sí
- Nuestro objeto “Robot” debería tener la posición, un estado, y operaciones que permiten cambiar la posición y cambiar el estado

Creación de objetos en PHP

- Para crear nuestro Robot, tenemos que declarar una “clase” Robot
- Una clase es un molde que permite crear objetos de esa clase
- A los objetos creados a partir de una clase se les suele denominar “instancias” de dicha clase

```
<?php
class Robot
{

}
?>
```

Clases vs. Objetos

Clase

Instancias (con diferentes atributos, en este caso, colores distintos)

Creación de objetos en PHP

- Simplemente se indica que se desea crear una nueva “instancia” de una clase:

```
$miRobot = new Robot();
```

- Mediante los paréntesis podemos pasar argumentos en la creación, por ejemplo, el nombre del Robot

Atributos

- Primero tenemos que establecer qué cosas va a tener nuestro robot:
  - Una posición *x* y otra *y*
  - Un estado
- Podría tener más cosas: color, modelo, nombre, distancia recorrida, etc. Cosas relacionadas con el Robot
- No tiene sentido poner un atributos que pertenecen otras entidades “precio de producto” o “número de ticket de recital”
- Esto se suele denominar alta cohesión (los atributos del robot tienen relación con la entidad robot)

Atributos /Properties

```
<?php
class Robot
{
   private $x;
   private $y;
   private $estado =
'APAGADO';
}
?>
```

Visibilidad

- Los atributos en PHP pueden ser definidos con un modificador de acceso
  - Public: puede ser accedido desde cualquier lugar
  - Private: solo puede ser accedido por la clase que lo declara
  - Protected: olo puede ser accedido por la clase que lo declara o subclases

Operaciones / Métodos

- Al igual que las properties, los métodos pueden tener modificadores de acceso en PHP

```
<?php
class Robot
{
   private $x;
   private $y;
   private $estado = 'APAGADO';

   public function encender(){
      $this->estado = 'ENCENDIDO';
   }
}
?>
```

Acceso a properties
Dentro de la clase debemos usar la palabra reservada $this:

- $this->nombreProperty

Creación de objetos

```
<?php
class Robot
{
   private $x;
   private $y;
   private $estado = 'APAGADO';

   public function encender(){
      $this->estado = 'ENCENDIDO';
   }
}
$robotCasa = new Robot();
$robotOficina = new Robot();
$robotOficina->encender();
?>
```

- Para “llamar” a los métodos de un objeto utilizamos ->

- ¿robotCasa y robotOficina tienen los mismos atributos? ¿Si modifico el property estado del robotOficina se modifica el de robotCasa?

¿Objeto o Clase?

- ¿Tiene sentido crear 2 **clases** RobotCasa y RobotOficina?
- ¡NO! En este caso, serían idénticas en cuanto a atributos, pero tendrían distinto nombre
- En Programación Orientada a Objetos, es **clave** discernir entre una clase (un tipo de objeto) y un objeto (una instancia/ejemplo de esa clase)
- ¿Qué problema hay en tener 2 clases iguales? Es doble mantenimiento: por ejemplo, si se quiere agregar un nuevo atributo a los robots, hay que agregarlo en 2 clases. ¿Qué sucede si tenemos 10 clases idénticas?

Constructores

- En cada clase podemos declarar un método “constructor”
- Este método se llama cada vez que se hace un new de la clase
- Es útil para cualquier inicialización que el objeto necesite

```
<?php
class Robot {
   private $x;
   private $y;
   function __construct() {
      $this->x = 0;
      $this->y = 0;
   }
}
?>
```

Interacción entre Objetos

- Para crear una aplicación más grande debemos interactuar entre distintos objetos
- Conceptualmente, en Programación Orientada a Objetos, se dice que un objeto “le envía un mensaje” a otro objeto. El mensaje indica la operación a realizar y sus parámetros
- En PHP, esta interacción es través de llamados a métodos (no son mensajes literalmente)

```
$robotOficina->encender();
```

Se envía el mensaje encender al objeto de tipo Robot

Ejemplo Interacción entre objetos

- Supongamos una casa inteligente que posee entre sus artefactos un robot de limpieza
- Cuando TODOS sus habitantes abandonan la casa, se indica al robot que se encienda
- Simplemente vamos a simular la situación. Le vamos a indicar a la casa explícitamente la salida de un habitante

Ejemplo Interacción entre objetos

- Hay formas más orientadas a objetos que otras para resolver este problema
- Opción 1: modelar la cantidad de habitantes actuales de la casa dentro del robot, y contar las salidas y entradas a la casa en el mismo
  - se pierde cohesividad en el robot
  - se mezclan atributos que son parte del robot con atributos que son de otra entidad (la casa inteligente)

Ejemplo Interacción entre objetos

- Opción 2: usar una variable global (ej. Variable huespedes)
  - Las variables globales siempre representan un riesgo
  - La modificación de sus valores se puede dar en cualquier lugar del código y en cualquier orden
  - Al no tener las modificaciones sobre un valor en un solo lugar, no se puede controlar el orden en el que modifican las variables

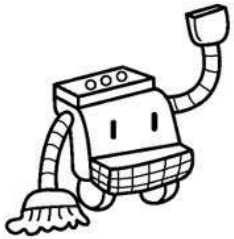
Ejemplo Interacción entre objetos

- Opción 3: modelar la casa inteligente por un lado y el robot por otro
  - Clase CasaInteligente y clase Robot
  - Para indicarle al robot que debe encenderse, la casa invoca un método del robot (el método encender)

Ejemplo Interacción entre objetos

La casa y el robot no deben modificar DIRECTAMENTE los atributos el uno al otro
¡La interacción es solo a través de métodos!

encender()



Detecta que su cantidad de habitantes es 0, y envía un mensaje al robot (invocando el método encender del robot)

Al recibir el mensaje, cambia su estado a encendido

Ejemplo Interacción entre objetos

```php
<?php
class CasaInteligente{
   private $habitantesActuales;
   private $robotLimpieza;

   function __construct(){
      $this->habitantesActuales = 2;
      $this->robotLimpieza = new Robot();
   }
   public function salirDeCasa(){
      $this->habitantesActuales = $this->habitantesActuales - 1;
      if($this->habitantesActuales == 0)
         $this->robotLimpieza->encender();
   }
}
```

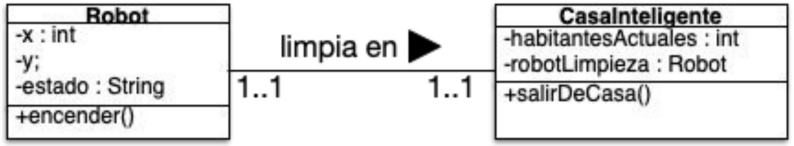
Ejemplo Interacción entre objetos

¿Cómo usaríamos la casa y el robot?

```
$casa = new CasaInteligente();
$casa->salirDeCasa();
$casa->salirDeCasa();
```

Diagrama de clases

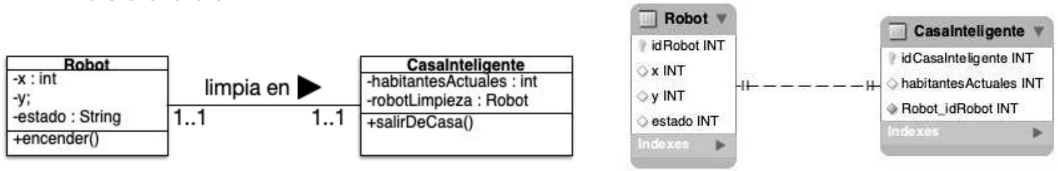
- Una herramienta útil para pensar en nuestro diseño orientado a objetos son los diagramas de clases de UML



POO & BBDD

Relación entre POO y BBDD

- Hay algunos paralelismos entre objetos y registros, así como entre clases y tablas.
  - Ej: Las tablas tienen columnas, las clases atributos y métodos.
- Las clases que se persisten suelen tener una tabla asociada



Relación entre Objeto y Registro en BBDD

POO	Entidad-Relación
Clase	Tipo Entidad
Objeto	Instancia de entidad
Asociación entre objetos	Relación entre tablas
Modelado del Comportamiento!!	**Modelado de los Datos!!**

2011 UML, Pearson Education

Relación entre POO y BBDD

- Son mundos similares, pero diferentes
- Se lo conoce como problema de “impedance mismatch”
- Existen herramientas que facilitan el mapeo (Object Relational Mapping - ORM) (BD y Arq. Web)

POO & BBDD + PDO

Al hacer un ***fetch*** sobre la consulta, podemos indicar que nos traiga un objeto e indicar de qué clase es ese obieto.

```
class User{
   public $name;
   ...
};

//Traer muchos objetos como registros
$users = $pdo->query('SELECT name FROM
users')->fetchAll(PDO::NUM, 'User');

//Traer un solo objeto y que PHP cree la clase
$stmt = $pdo->query('SELECT name FROM users LIMIT 1');
$stmt->setFetchMode(PDO::FETCH_CLASS, 'User');
$user = $stmt->fetch();
```

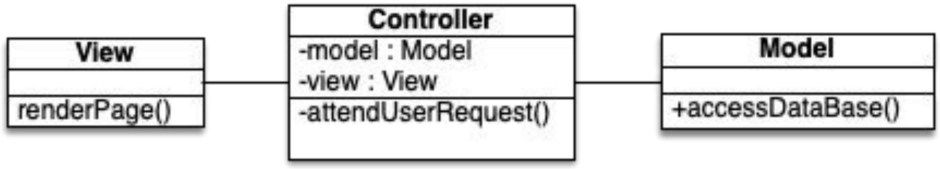
Debe coincidir con el nombre de la columna

https://phpdelusions.net/pdo/objects

POO & MVC

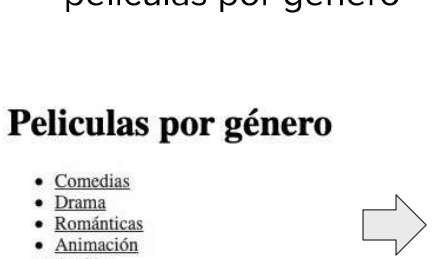
MVC

- La idea principal es encapsular cada parte del patrón en una clase



Ejemplo MVC

- Supongamos una página que nos permite mostrar películas por género



Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Movies</title>
</head>
<body>

   <h1>Peliculas por género</h1>

   <ul>
       <li><a href="peliculas.php?genre=Comedy">Comedias</a></li>
       <li><a href="peliculas.php?genre=Drama">Drama</a></li>
       <li><a href="peliculas.php?genre=Romance">Románticas</a></li>
       <li><a href="peliculas.php?genre=Animation">Animación</a></li>
       <li><a href="peliculas.php?genre=Action">Acción</a></li>
   </ul>

</body>
</html>
```

- Por simplicidad la aplicacion no esta enrutada
- Mandamos el género que queremos ver al archivo peliculas.php (se envia por GET)

Peliculas.php

```
<?php
/**Imprime la lista de peliculas por un genero determinado
* Este archivo obtiene el parametro GET 'genre' que determina el genero
solicitado*/

// verifica datos obligatorios
if (!isset($_GET['genre']) || empty($_GET['genre'])) {
   echo "<h2>Error! Género no especificado.</h2>";
   die();
}

// obtiene el genero enviado por GET
$genre = $_GET['genre'];

echo "<h1>Lista por género: $genre</h1>";
echo "<a href='index.html'> Volver </a>";

// Obtiene la lista de peliculas de la DB según género
$db = new PDO('mysql:host=localhost;'.'dbname=db_movies;charset=utf8', 'root', '');
$query = $db->prepare('SELECT * FROM movies WHERE genre = ?');
$query->execute([$genre]);
$movies = $query->fetchAll(PDO::FETCH_OBJ);

// imprime la tabla de peliculas
echo "<table>
<thead>
   <tr>
       <th>Titulo</th>
       <th>Año</th>
       <th>Estudio</th>
   </tr>
<thead>
<tbody>
";
foreach($movies as $movie) {
echo "
   <tr>
       <td>$movie->title</td>
       <td>$movie->year</td>
       <td>$movie->studio</td>
   </tr>
";
}
echo " </tbody>
</table>";
```

Ejemplo MVC

Cuando tengamos sistemas más complejos, esta forma de programar dificultad la legibilidad y el mantenimiento

¿Cómo organizamos este código con MVC?

Peliculas.php

```php
<?php
/**Imprime la lista de peliculas por un genero determinado
* Este archivo obtiene el parametro GET 'genre' que determina el genero
solicitado*/

// verifica datos obligatorios
if (!isset($_GET['genre']) || empty($_GET['genre'])) {
    echo "<h2>Error! Género no especificado.</h2>";
    die();
}

// obtiene el genero enviado por GET
$genre = $_GET['genre'];

echo "<h1>Lista por género: $genre</h1>";
echo "<a href='index.html'> Volver </a>";

// Obtiene la lista de peliculas de la DB según género
$db = new PDO('mysql:host=localhost;'.'dbname=db_movies;charset=utf8', 'root', '');
$query = $db->prepare('SELECT * FROM movies WHERE genre = ?');
$query->execute([$genre]);
$movies = $query->fetchAll(PDO::FETCH_OBJ);
```

```php
// imprime la tabla de peliculas
echo "<table>
<thead>
    <tr>
        <th>Titulo</th>
        <th>Año</th>
        <th>Estudio</th>
    </tr>
<thead>
<tbody>
";
foreach($movies as $movie) {
echo "
    <tr>
        <td>$movie->title</td>
        <td>$movie->year</td>
        <td>$movie->studio</td>
    </tr>
";
}
echo " </tbody>
</table>";
```

Refactoricemos!

1. Creemos un archivo por cada componente
   a. MovieModel
   b. MovieView
   c. MovieController
2. Movamos el código creando las clases junto a sus properties y métodos

MovieModel.php

Empecemos encapsulando el modelo

```php
<?php
class MovieModel{
    /**
     *  Obtiene la lista de peliculas de la DB según género
     */
    function getMoviesByGenre($genre) {
        $db = new PDO('mysql:host=localhost;'.'dbname=db_movies;charset=utf8', 'root', '');
        $query = $db->prepare('SELECT * FROM movies WHERE genre = ?');
        $query->execute([$genre]);
        $movies = $query->fetchAll(PDO::FETCH_OBJ);
        return $movies;
    }
}
```

MovieView.php

```php
<?php
class MovieView{
    function renderMoviesByGenre($genre, $movies){
        echo "<h1>Lista por género: $genre</h 1>";
        echo "<a href='index.html'> Volver </a>" ;
        // imprime la tabla de peliculas
        echo "<table>
        <thead>
            <tr>
                <th>Titulo</th>
                <th>Año</th>
                <th>Estudio</th>
            </tr>
        <thead>
        <tbody>
        ";
        foreach ($movies as $movie) {
            echo "
                <tr>
                    <td>$movie ->title</td>
                    <td>$movie ->year</td>
                    <td>$movie ->studio</td>
                </tr>
            ";
            }
            echo " </tbody>
            </table>";
        }

        function renderError() {
            echo "<h2>Error! Género no especificado.</h2>" ;
        }
    }
```

MovieController.php

```php
<?php
require_once 'MovieModel.php';
require_once 'MovieView.php';
class MovieController{
    private $model;
    private $view;

    public function __construct() {
        $this->model = new MovieModel();
        $this->view = new MovieView();
    }

    function showMoviesByGenre() {
        // verifica datos obligatorios
        if (!isset($_GET['genre']) || empty($_GET['genre']))
            $this->view->renderError();
            return;
        }
        // obtiene el genero enviado por GET
        $genre = $_GET['genre'];
        // obtengo las peliculas del modelo
        $movies = $this->model->getMoviesByGenre($genre);
        // actualizo la vista
        $this->view->renderMoviesByGenre($genre, $movies);
    }
}
```

Peliculas.php

```php
<?php

require_once 'MovieController.php';

// instancio la clase del controlador
$controller = new MovieController();

$controller->showMoviesByGenre();
```

Referencias

PROGRAMACION EN C, C++, JAVA Y UML y otros link

POO y MVC en PHP link

Must Read: POO php.net link

Aprender POO con PHP 5 link