GUÍA Nº 2 – Modelo Arquitectura MVC

FACULTAD	CURSO	AMBIENTE
INGENIERÍA	DISEÑO Y ARQUITECTURA DE SOFTWARE	LABORATORIO DE DISEÑO Y ARQUITECTURA DE SOFTWARE 77C0206

ELABORADO POR	ANÍBAL SARDÓN PANIAGUA	**APROBADO POR**	ARTURO RIVERA
VERSIÓN	001	**FECHA DE APROBACIÓN**	01/03/2022

1. LOGRO GENERAL DE UNIDAD DE APRENDIZAJE

El estudiante formula un informe descriptivo sobre los requerimientos de un producto de software para las necesidades de los Stakeholders, aplicando un lenguaje de modelado como UML y un proceso de desarrollo en la herramienta IBM RSA; definiendo la Visión del Negocio, modelo de casos de Uso y prototipos visuales.

2. OBJETIVOS ESPECÍFICOS DE LA PRÁCTICA

- Entender la Arquitectura MVC.
- Conocer lo componentes de la Arquitectura MVC y su implementación con JEE.
- Diseñar y Desarrollar una aplicación MVC con JEE.

3. MATERIALES Y EQUIPOS

- Computadoras Personales.
- Sistema Operativo Windows.
  Star UML
  Netbeans
  Glassfish
- Pizarra
- Plumón
- Mota

4. PAUTAS DE SEGURIDAD

Las computadoras y laptops deben de estar prendidas mientras se usan. Pero al terminar el laboratorio estas deben dejarse apagadas.

- En el laboratorio debe estar prendido el aire acondicionado para evitar sobrecalentamientos y averías, especialmente en épocas de altas temperaturas.
- Los estudiantes no pueden llevar alimentos que puedan derramar sobre los computadores.
- Computadoras, router, switch, puntos de acceso (caídas).
- Eléctricos, por contacto directo o indirecto, electricidad estática y por fenómeno térmico. Puede producir: electrocuciones y quemaduras.
- Procedimiento ante Corte de Energía Eléctrica
- No tocar el equipo eléctrico en el que se encuentra trabajando, puede que retorne la energía.
- Comunicarse con el Asistente de Operaciones de turno quien se comunicará con el Técnico.

5. FUNDAMENTO

La asignatura de Diseño y Arquitectura de Software es de carácter teórico-práctico y tiene el propósito de potenciar en el estudiante sus habilidades para analizar y diseñar una arquitectura de software. Se desarrolla los siguientes contenidos: Introducción a la arquitectura de software, vistas y estilos de la arquitectura, requisitos de calidad de un software, diagramación UML orientada al diseño arquitectónico de software, patrones de arquitectura, arquitectura orientada a servicios (SOA), Arquitecturas en Cloud Computing , Arquitecturas para software en dispositivos móviles y documentación de una arquitectura de software.

6. INTRODUCCIÓN (MARCO TEÓRICO)

PATRON MVC

Problema

Las aplicaciones deben estar creadas sobre una arquitectura sólida y estable que facilite su implementación y permita realizar los cambios y expansiones de la aplicación, además de facilitar la modificación y adición de nuevas funcionalidades.

La aplicación debe estar creada de manera modular, debe estar conformado por diferentes módulos independientes. Esto permite también que la aplicación sea más fácil de mantener, ya que los cambios en una parte causarán un menor impacto en las demás partes al estar aisladas unas de otras.

Solución

Como la mayor parte del costo de cada llamada está relacionado con la comunicación de ida y vuelta entre el cliente y servidor, una forma de reducir el número de llamadas es usando un objeto (el DTO) que agrega los datos que

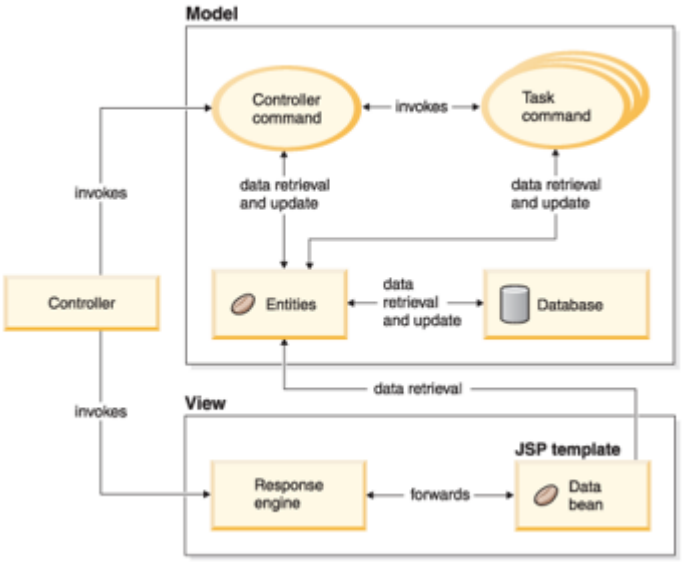
habrían sido transferidos por cada llamada, pero que son entregados en una sola llamada.

MVC separa los datos y la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones.

MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador.

Por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario.

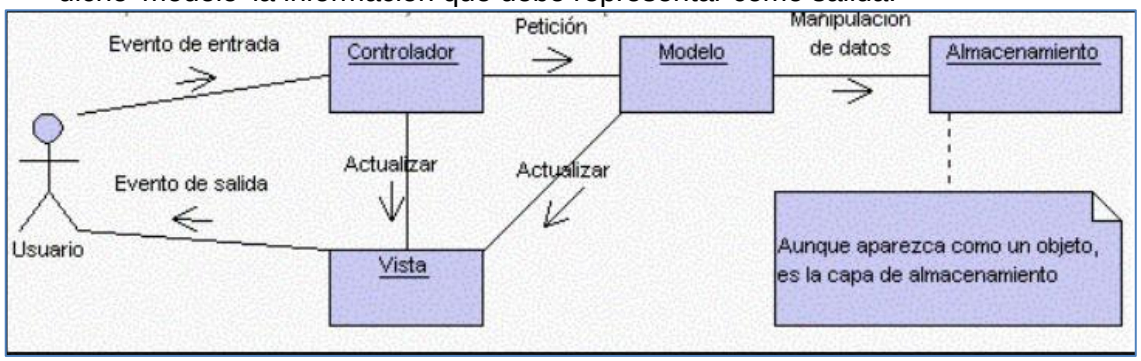
Se basa en las ideas de reutilización de código y la separación de conceptos, que buscan facilitar el desarrollo de aplicaciones y su posterior mantenimiento.

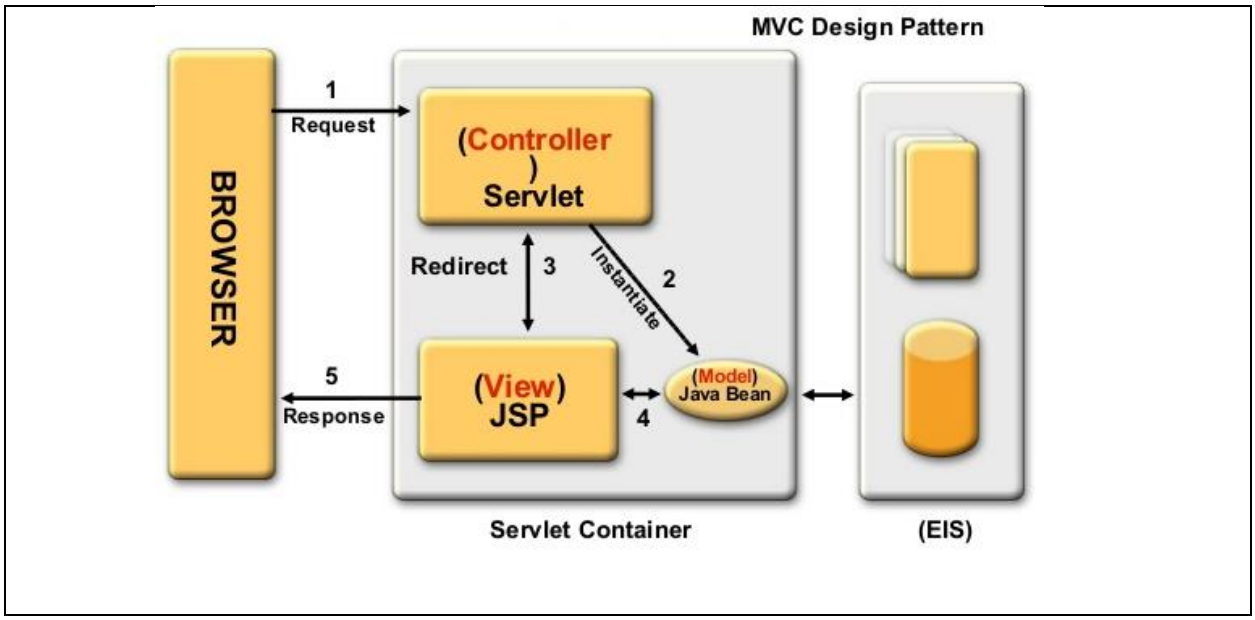


El Modelo: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio).

El Controlador: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información.

La Vista: Presenta el 'modelo' (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario), por tanto requiere de dicho 'modelo' la información que debe representar como salida.



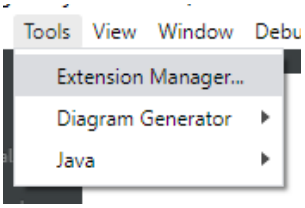


7. PROCEDIMIENTO (DESARROLLO DE LA PRÁCTICA)

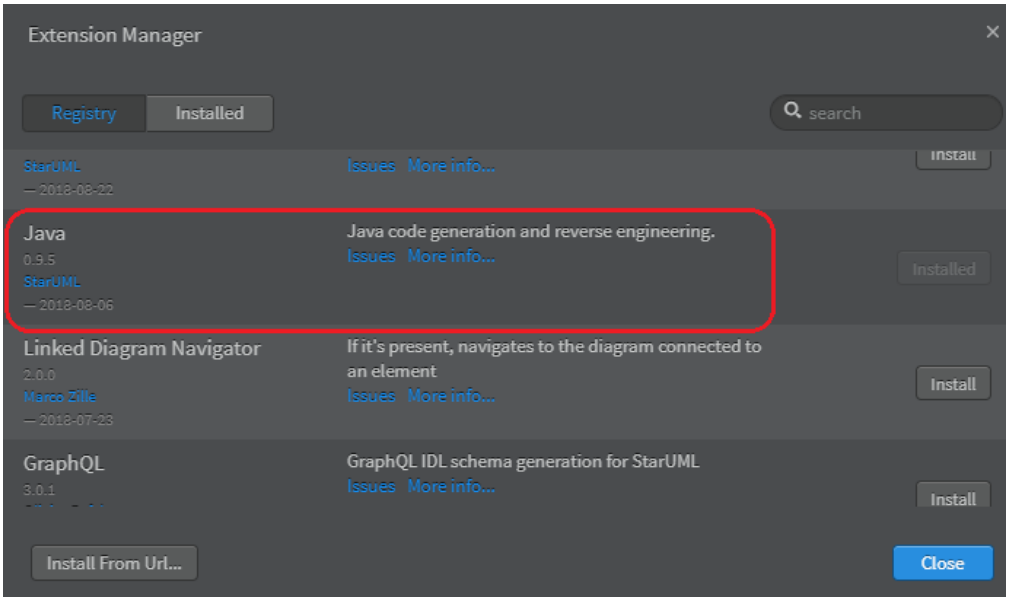
1. Agregar extensión para generar código Java.

Vamos a ingresar al Extension Manager para agregar la extensión que genera código en Java.

Hacemos clic en Tools **Extension Manager**.

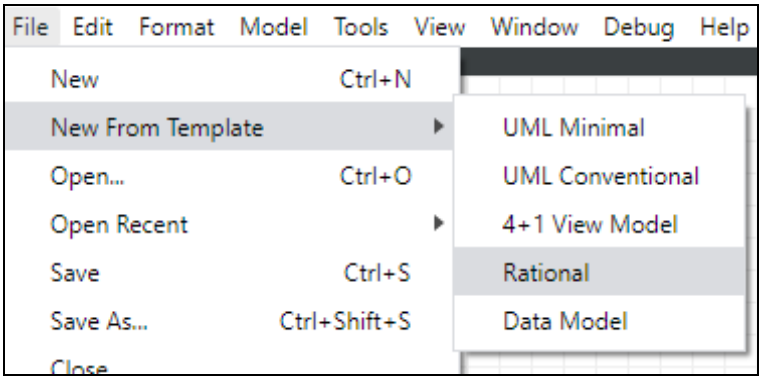


Buscamos y seleccionamos la librería Java y hacemos clic en el botón **Install**.

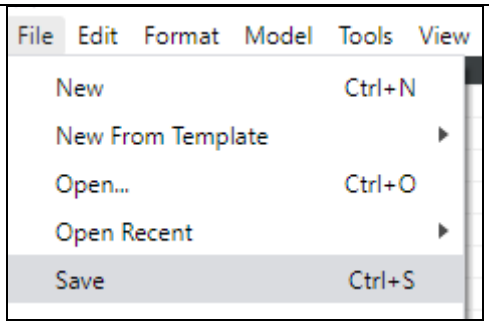


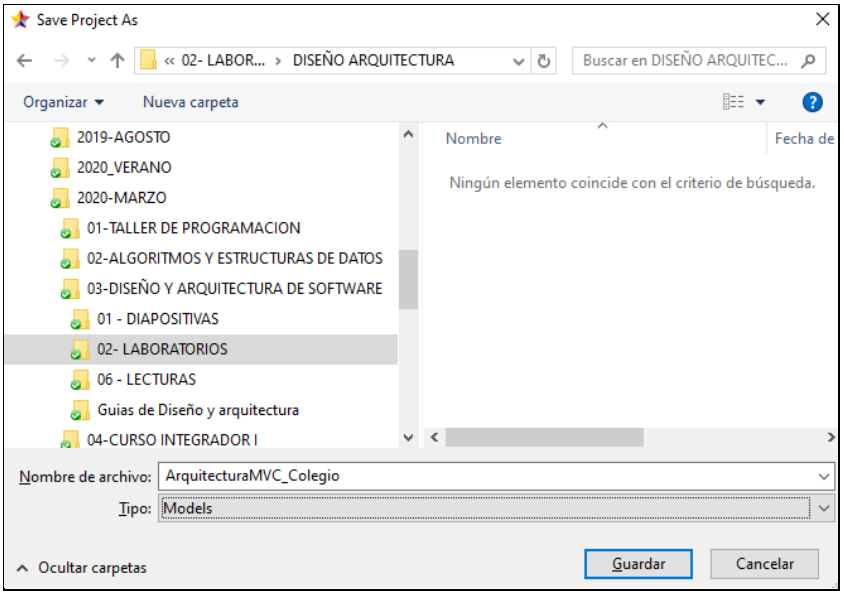
2. Creación del Proyecto Modelo con Star UML

Creamos un proyecto modelo (**ArquitecturaMVC_Colegio**) de tipo **Rational**.

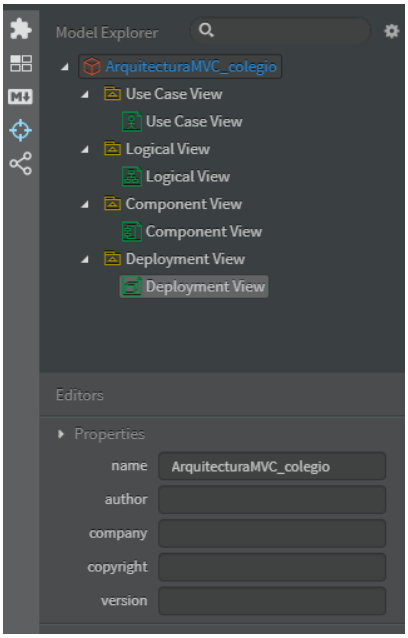


Grabamos el proyecto **ArquitecturaMVC_Colegio**

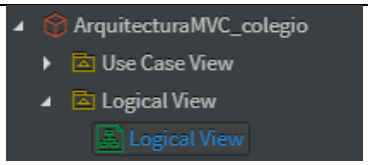




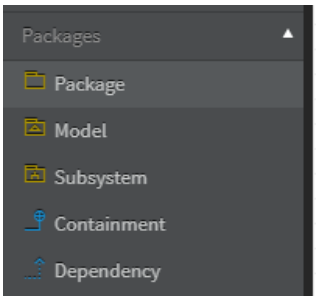
Ponemos nombre al proyecto con el **Model Explorer**

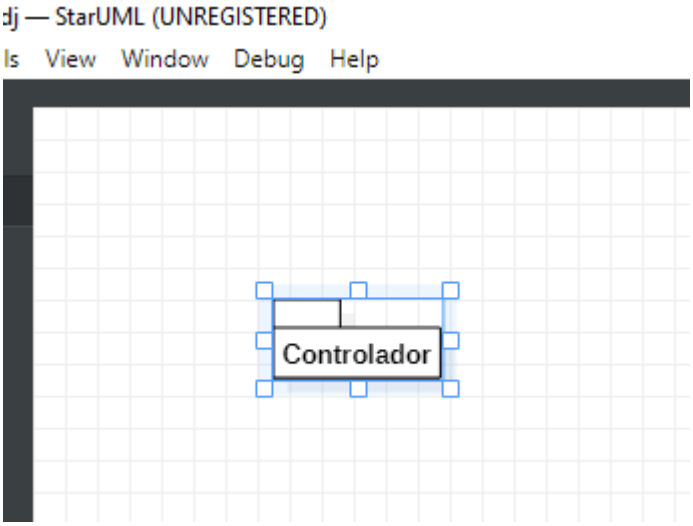


Seleccionamos el **Logical View** y vamos a agregar paquetes sobre el lienzo.

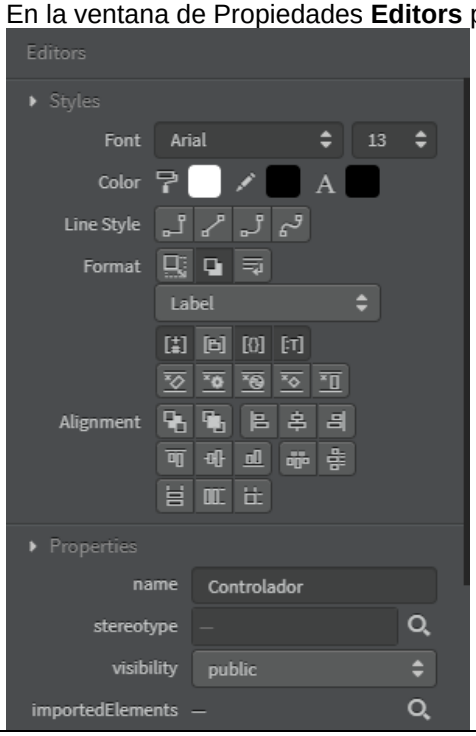


Desde el **ToolBox** seleccionamos **Package** y lo agregamos al Lienzo:

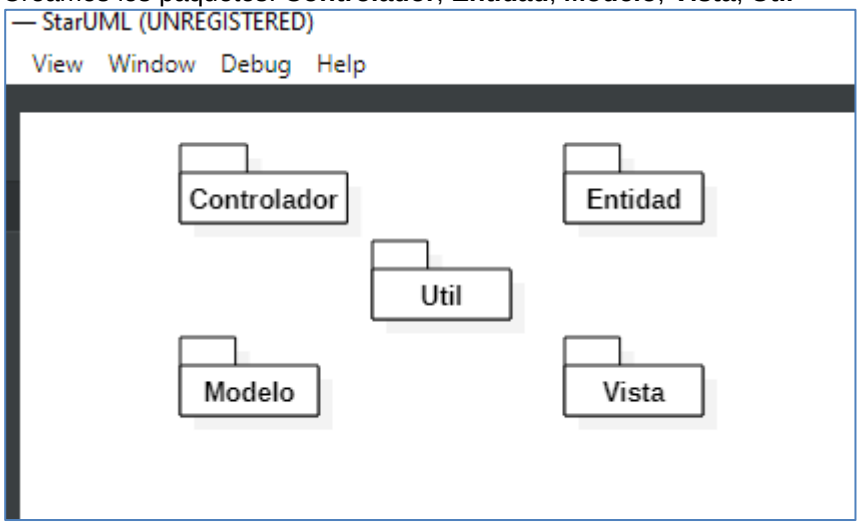


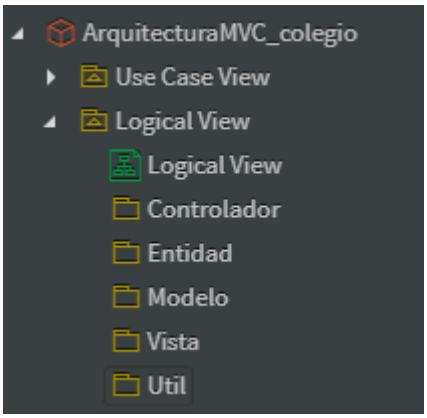


En la ventana de Propiedades **Editors** podemos colocar el nombre: **“Controlador”**

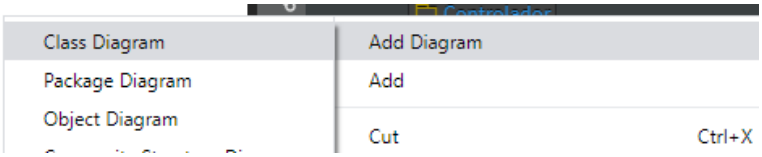


Creamos los paquetes: **Controlador**, **Entidad**, **Modelo**, **Vista**, **Util**

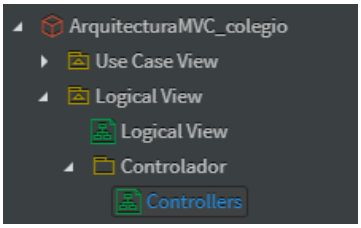


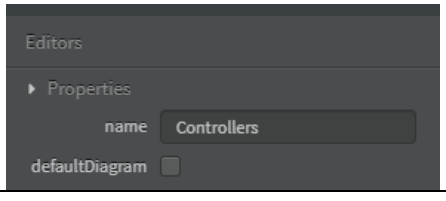


a) Seleccionamos el paquete **Controlador** y agregamos un Diagrama de Clases **Class Diagram.**



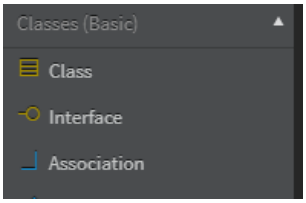
Ponemos nombre al diagrama **Controllers** en la ventana **Editors**



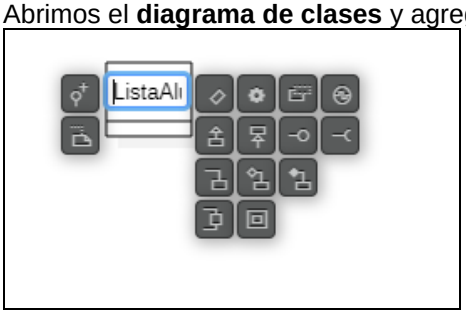


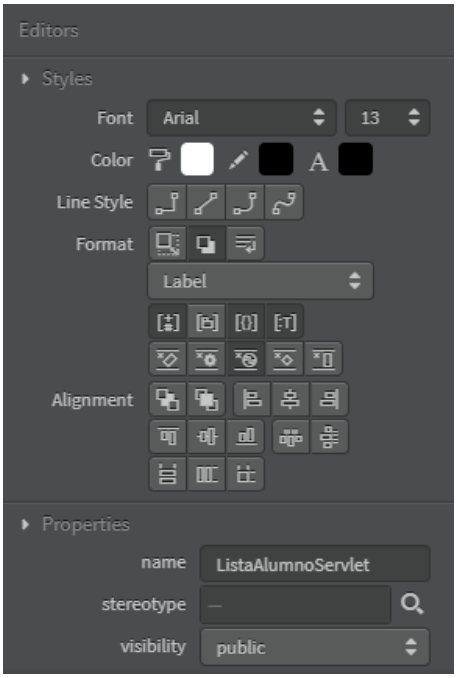
.

Seleccionamos el diagrama **Controllers** y agregamos las clases:



Abrimos el **diagrama de clases** y agregamos la **Clase ListaAlumnoServlet**:

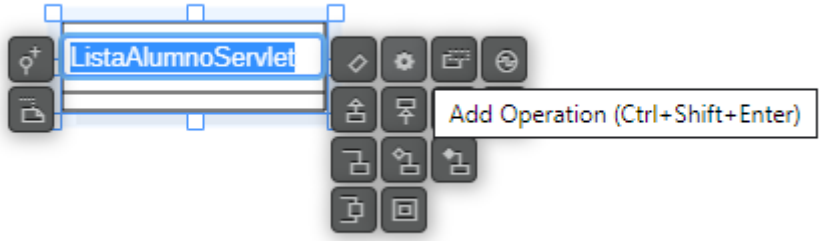




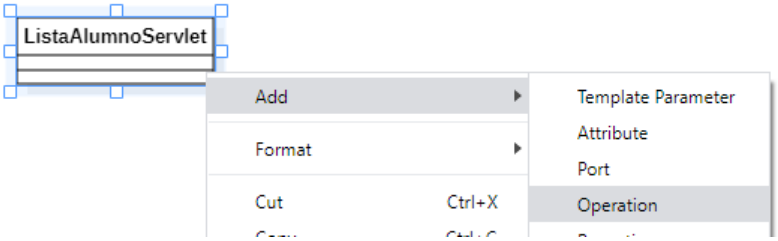
A la **Clase ListaAlumnoServlet** agregamos el Método **Service** y los parámetros de entrada: **HttpServletRequest** y **HttpServletResponse.**

Hacemos doble clic sobre la clase para mostrar los botones:

Hacemos clic sobre el botón **"Add Operation"**

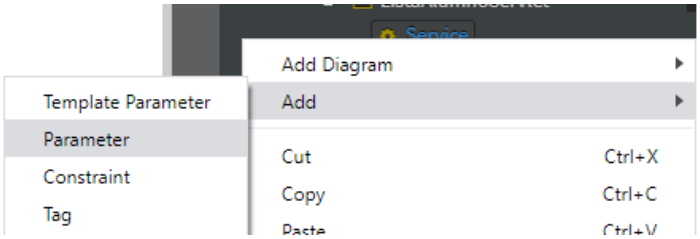


También Podemos hacer clic derecho y seleccionar la opción **Add** y **Operation**

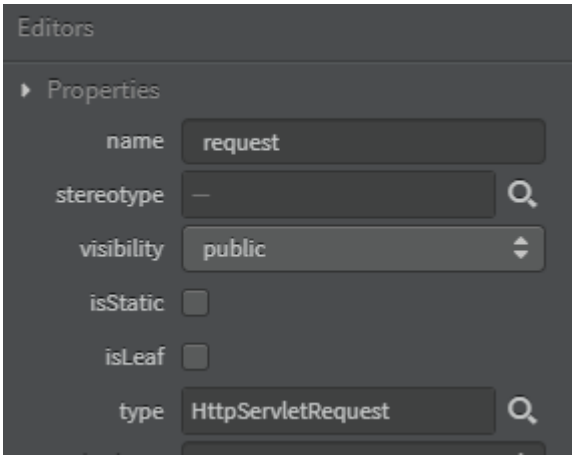


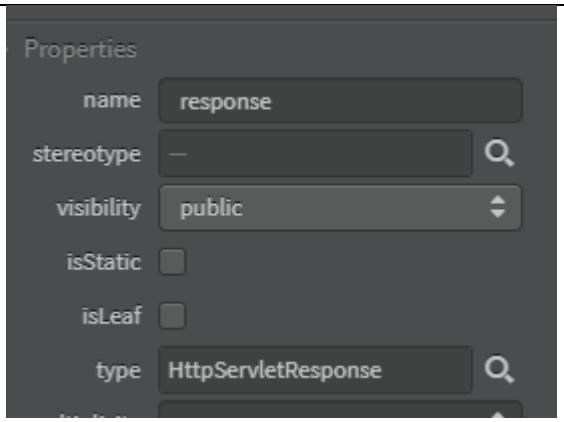
Seleccionamos el método **Service.**

Hacemos clic derecho y selccionamos la opción **Add** y **Parameter**.

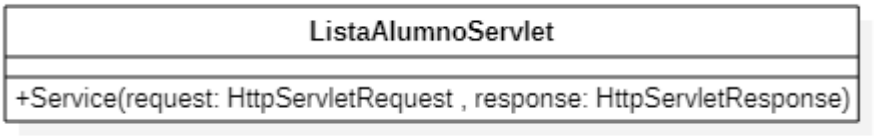


Vamos a agregar los parámetros de entrada: **HttpServletRequest** y **HttpServletResponse.**:

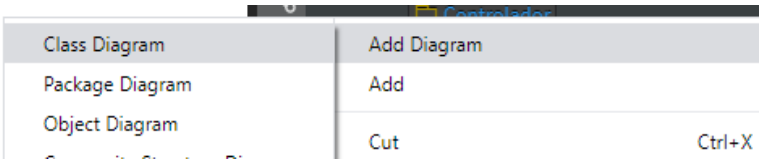




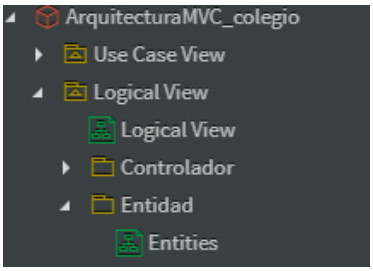
La clase se mostrará de la siguiente manera:

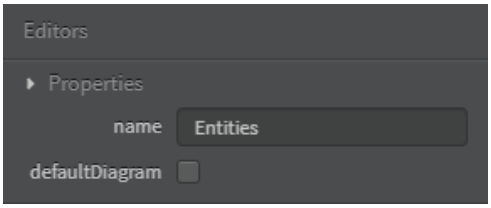


b) Seleccionamos el paquete **Entidad** y agregamos un Diagrama de Clases **Class Diagram.**



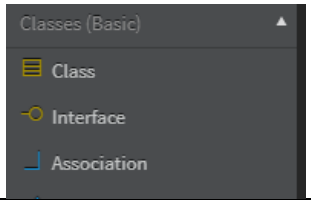
Ponemos nombre al diagrama **Entities** en la ventana **Editors**



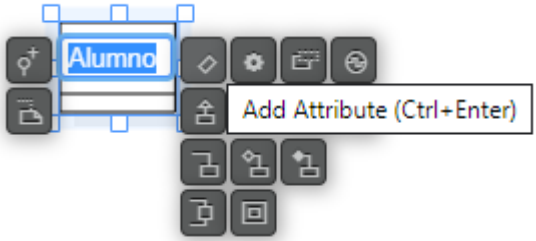


.

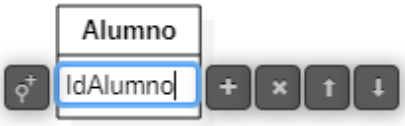
Seleccionamos el diagrama **Entities** y agregamos la **Clase Alumno**.



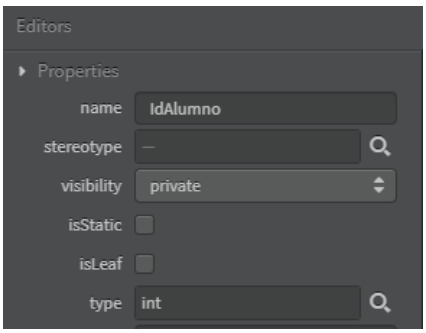
Vamos a agregar los atributos, hacemos doble clic sobre la clase y presionamos el botón **Add Attribute**



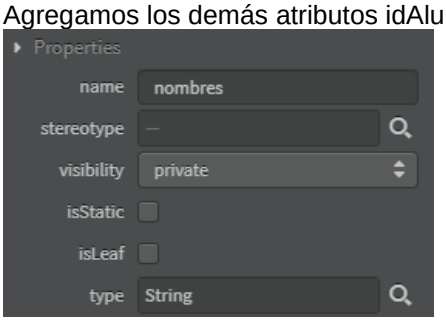
Agregamos el atributo **IdAlumno**

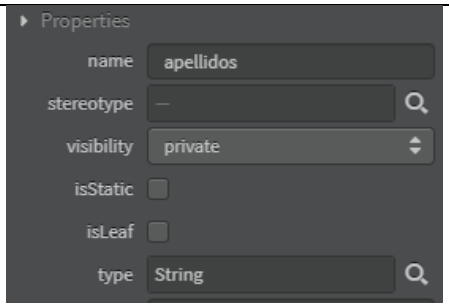


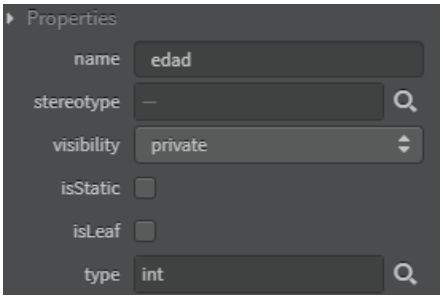
En la ventana Editors ingresamos las propiedades:



Agregamos los demás atributos idAlumno, nombres, apellidos, edad:

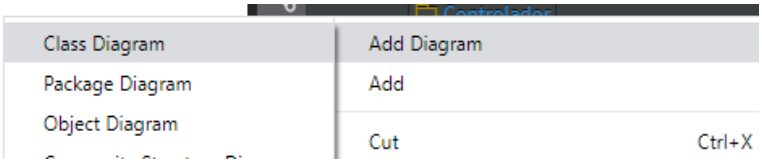




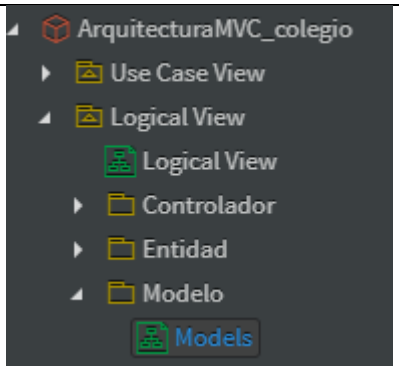


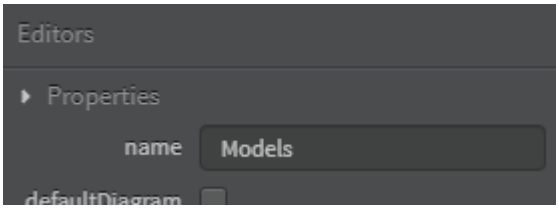
Agregamos los métodos getter & setter para los atributos privados.

c) Seleccionamos el paquete **Modelo** y agregamos un Diagrama de Clases **Class Diagram.**



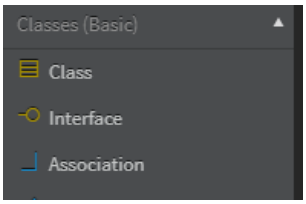
Ponemos nombre al diagrama **Models** en la ventana **Editors**



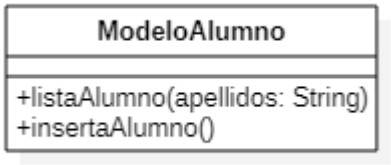


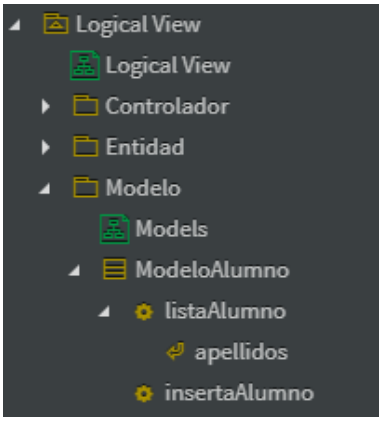
.

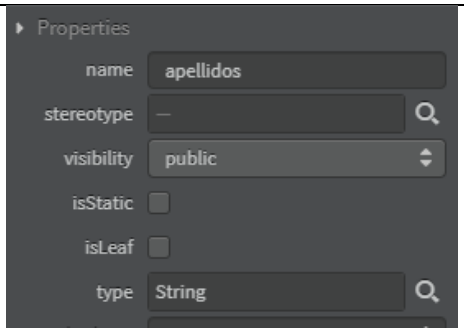
Seleccionamos el diagrama **Models** y agregamos la **Clase ModeloAlumno**.



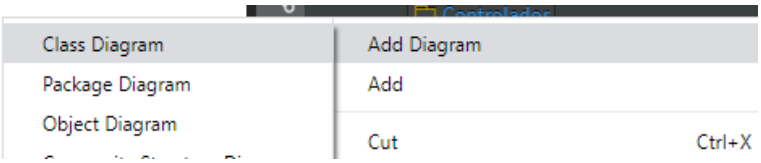
Agregamos los métodos **listaAlumno(),** recibe el parámetro (String apellido) y el método **insertaAlumno()**



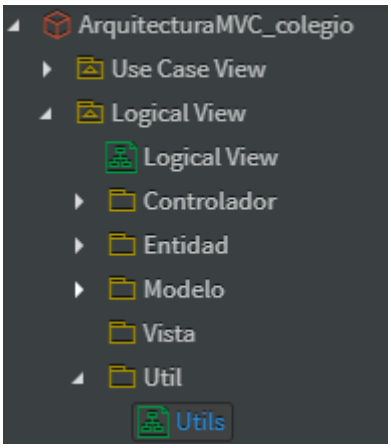


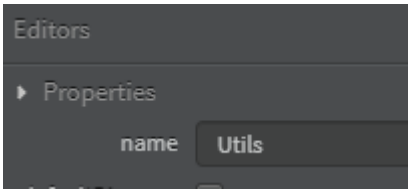


d) Seleccionamos el paquete **Util** y agregamos un Diagrama de Clases **Class Diagram.**



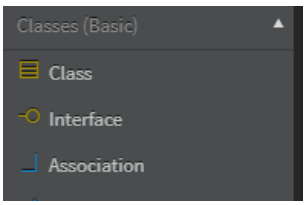
Ponemos nombre al diagrama **Utils** en la ventana **Editors**

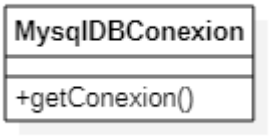




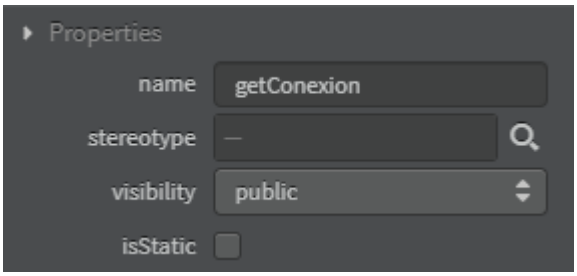
.

Seleccionamos el diagrama **Utils** y agregamos la **Clase MysqlDBConexion**.

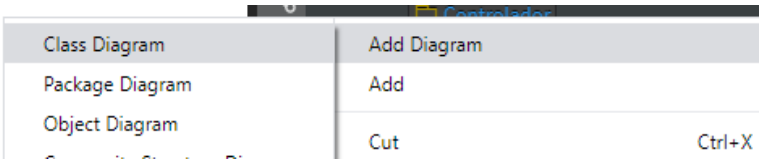




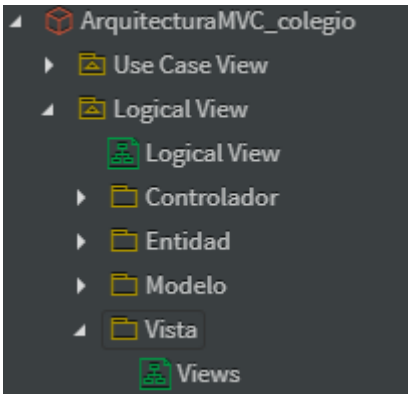
En la **Clase MysqlDBConexion** agregamos el Método **getConexion.**

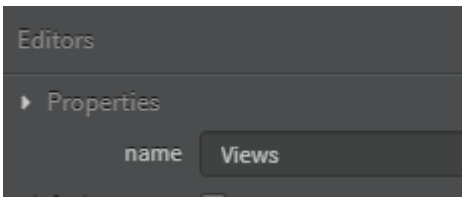


e) Seleccionamos el paquete **Vista** y agregamos un Diagrama de Clases **Class Diagram.**

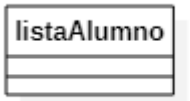


Ponemos nombre al diagrama **Views** en la ventana **Editors**



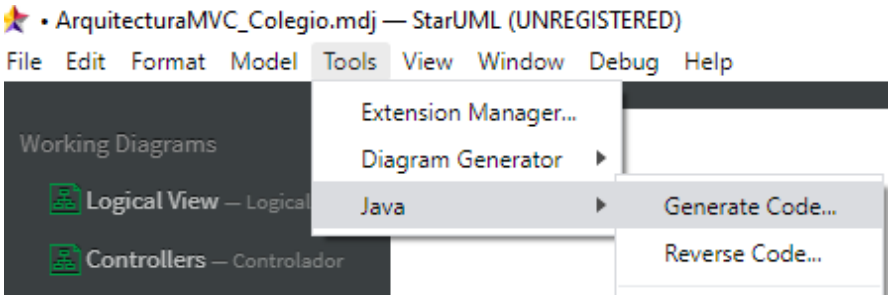


Seleccionamos el diagrama **Views** y agregamos la **Clase listaAlumno**.

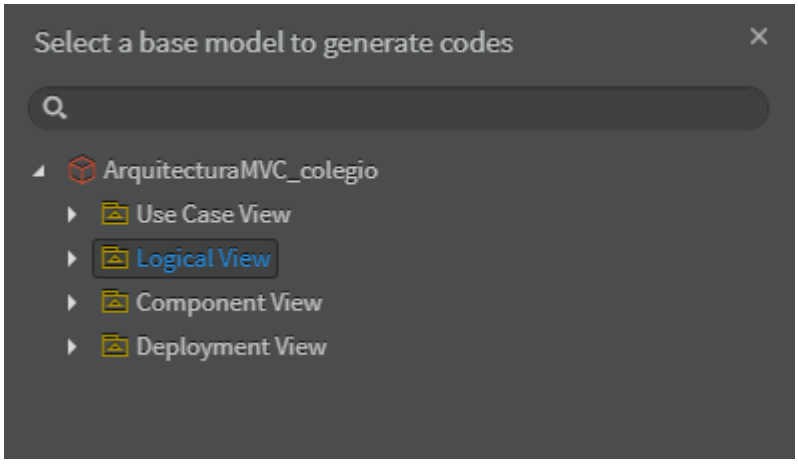


3. Generación de Código Fuente en JAVA para los elementos del MVC

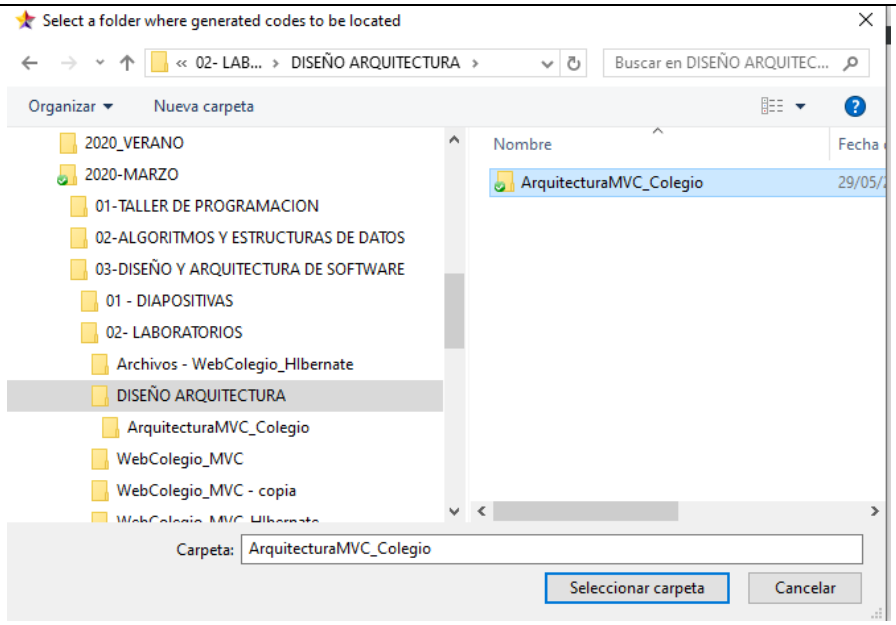
a) Hacemos Clic en el menú Tools – Java – Generate Code.



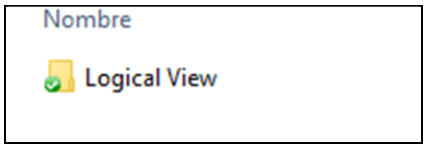
b) Seleccionamos la carpeta donde se encuentra el modelo "**Logical View"** para que transforme a código JAVA. Presionamos el **botón OK**

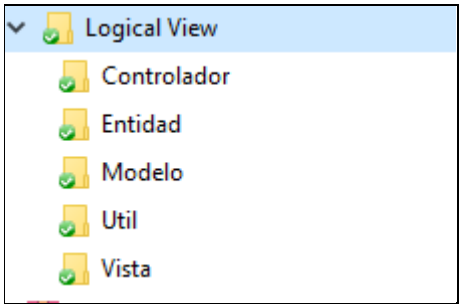


c) Selecionamos la carpeta donde se va a grabar el Proyecto con el Código Generado en Java.



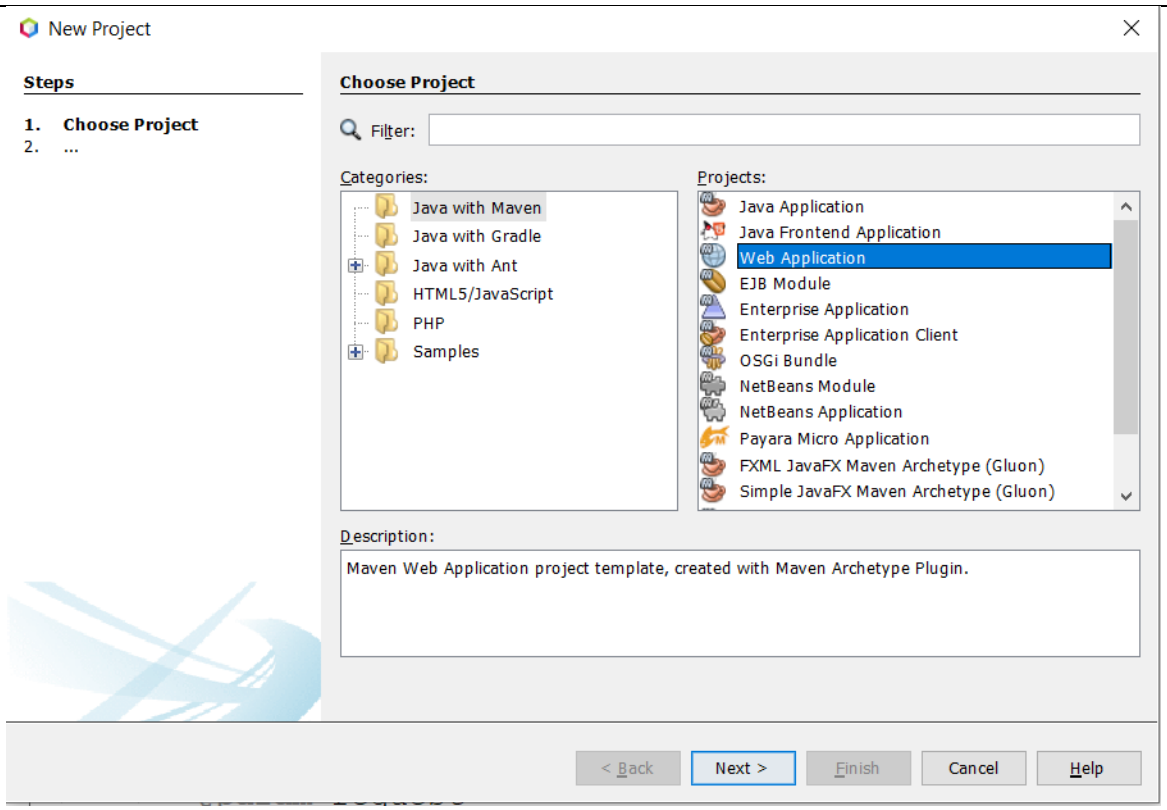
d) Se va a generar el Código en **JAVA**



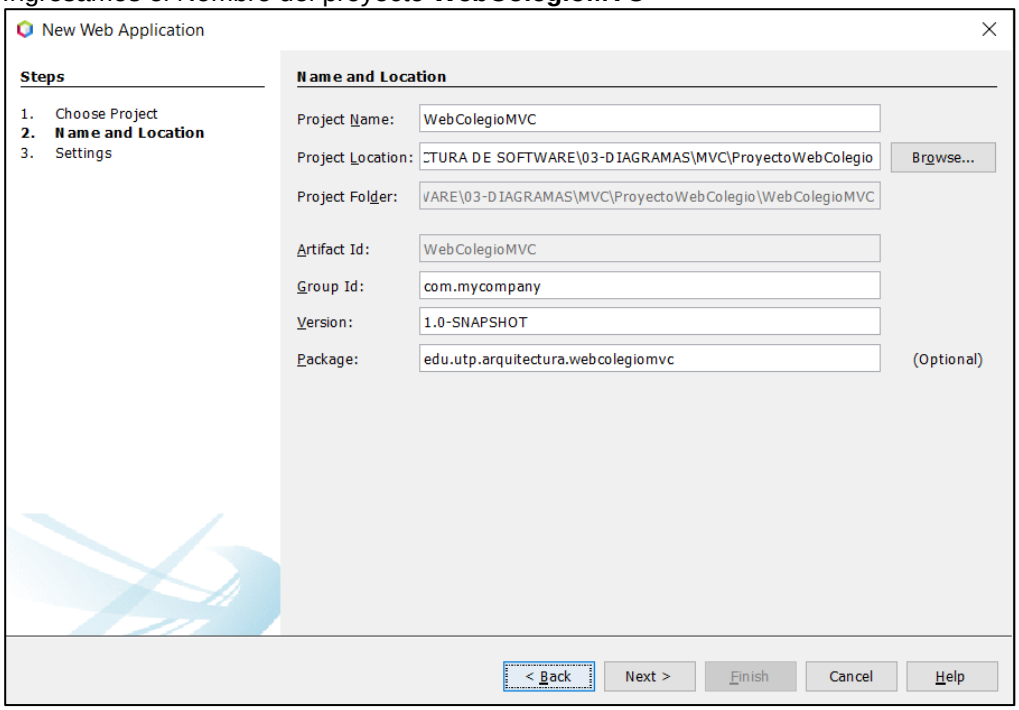


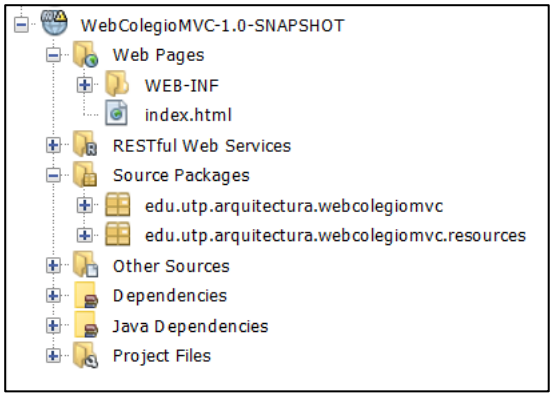
4. Creación de la Aplicación Web con los elementos MVC generados.

Ingresamos a NetBeans, y creamos un nuevo proyecto Web, hacemos Clic en el menú Archivo -> Nuevo Proyecto -> Java Web - > Web Aplication.



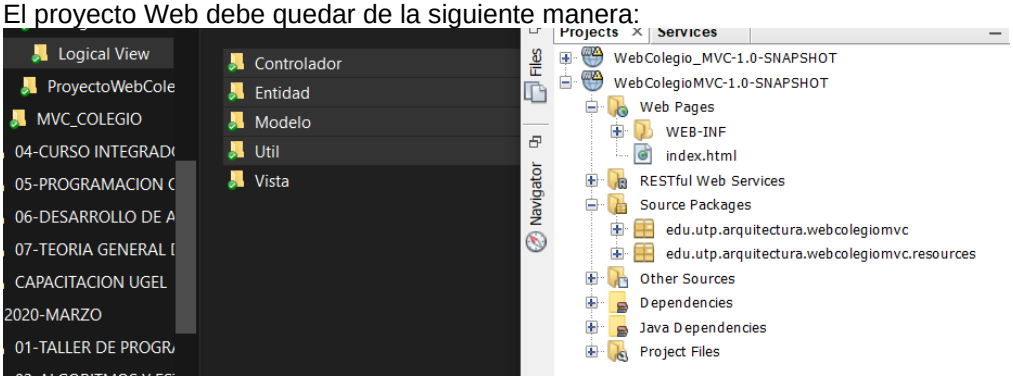
Ingresamos el Nombre del proyecto **WebColegioMVC**

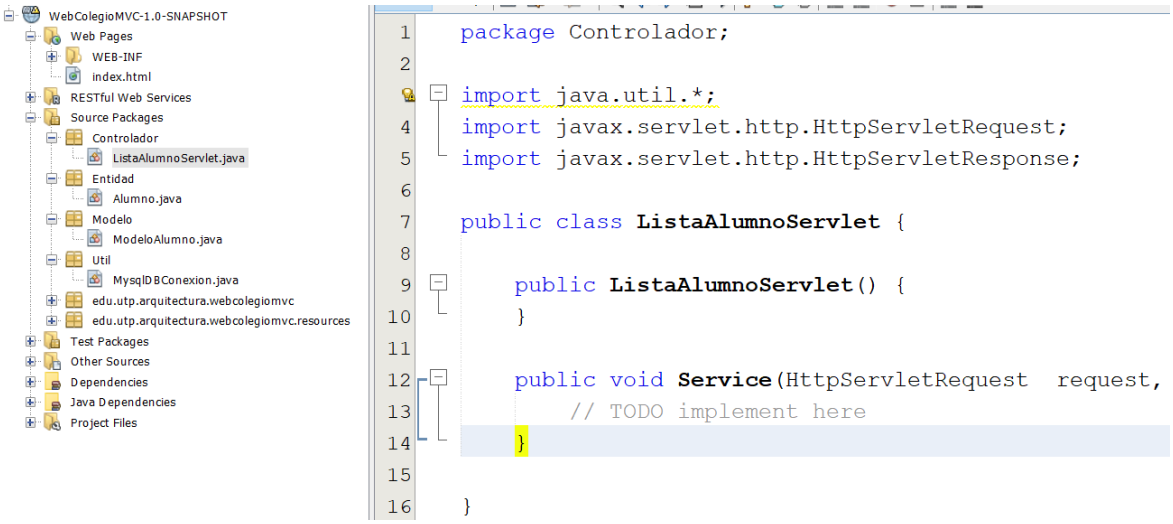




Las **carpetas y las clases generadas con Star UML**, ***las <u>vamos a arrastrar hacia el proyecto</u>*** Web en Netbeans y las vamos a copiar dentro de la **carpeta Source Pakages.**

El proyecto Web debe quedar de la siguiente manera:





Vamos a codificar las clases generadas:

a) Abrimos la clase **MysqlDBConexion** y agregamos el siguiente código:

```java
package utils;

import java.sql.Connection;
import java.sql.DriverManager;

public class MysqlDBConexion
{
        static{
                try {
                        Class.forName("com.mysql.jdbc.Driver");
                }catch (ClassNotFoundException e) {
                        e.printStackTrace();
                }
        }
        public static Connection getConexion()
        {
            Connection con=null;
            try
            {
                con=DriverManager.getConnection("jdbc:mysql://localhost/colegio","root","");
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
            return con;
        }
}
```

b) Abrimos la clase **ModeloAlumo** y agregamos el siguiente código:

```java
package modelo;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;

import utils.MysqlDBConexion;

import entidad.Alumno;

public class ModelAlumno
{
	public List<Alumno> listaAlumno(String ape)
	{
		List<Alumno> data = new ArrayList<Alumno>();

		Connection conn= null;
		PreparedStatement pstm = null;
		ResultSet rs = null;
		try
		{	//LLAMADA A METODO para conectarse a la BD
			conn = MysqlDBConexion.getConexion();
			// CONSULTA CON FILTRO like
			String sql ="select * from tbAlumno where apellido like ?";
			pstm = conn.prepareStatement(sql);
			//
			pstm.setString(1,ape + "%");
			rs = pstm.executeQuery();
			Alumno obj = null;
			while(rs.next())
			{
				obj = new Alumno();
				obj.setIdAlumno(rs.getInt("idtbAlumno"));
				obj.setNombre(rs.getString("nombre"));
				obj.setApellido(rs.getString("apellido"));
				obj.setEdad(rs.getInt("edad"));
				data.add(obj);
			}

		}
		catch (Exception e)
		{
			e.printStackTrace();
		}
		finally
		{
			try
			{
				if(rs!= null) rs.close();
				if(pstm!= null) pstm.close();
				if(conn!= null) conn.close();
			}
```

```
                    catch (Exception e2)
                    {
                    }
                }

                return data;
            }

            public int insertaAlumno(Alumno obj){
                int salida = -1;

                Connection conn= null;
                PreparedStatement pstm = null;
                try {
                    conn = MysqlDBConexion.getConexion();
                    String sql ="insert into tbalumno values(null,?,?,?)";
                    pstm = conn.prepareStatement(sql);
                    pstm.setString(1, obj.getNombre());
                    pstm.setString(2, obj.getApellido());
                    pstm.setInt(3, obj.getEdad());
                    salida = pstm.executeUpdate();
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
                finally
                {
                    try
                    {
                        if(pstm!= null) pstm.close();
                        if(conn!= null) conn.close();
                    }
                    catch (Exception e2) {
                    }
                }
                return salida;
            }
        }
```

c) Abrimos la clase **Alumno** y agregamos el siguiente código:

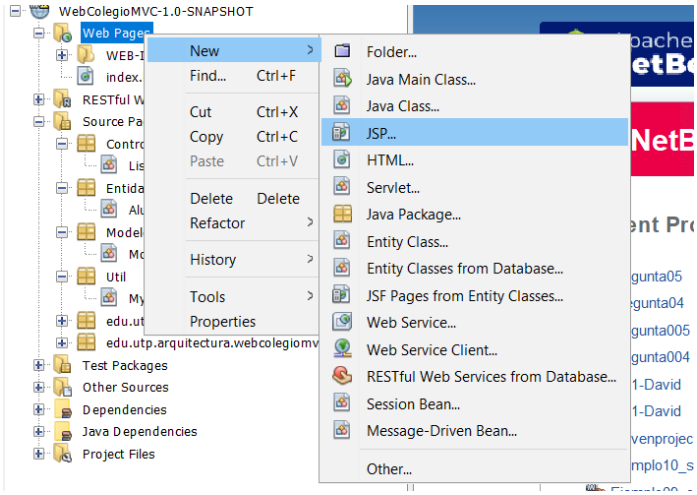
```java
package entidad;
public class Alumno
{
        private int idAlumno, edad;
        private String nombre, apellido;

        public int getIdAlumno() {
                return idAlumno;
        }
        public void setIdAlumno(int idAlumno) {
                this.idAlumno = idAlumno;
        }
        public int getEdad() {
                return edad;
        }
        public void setEdad(int edad) {
                this.edad = edad;
        }
        public String getNombre() {
                return nombre;
        }
        public void setNombre(String nombre) {
                this.nombre = nombre;
        }
        public String getApellido() {
                return apellido;
        }
        public void setApellido(String apellido) {
                this.apellido = apellido;
        }
}
```

d) Abrimos la clase **ListaAlumnoServlet** y agregamos el siguiente código:

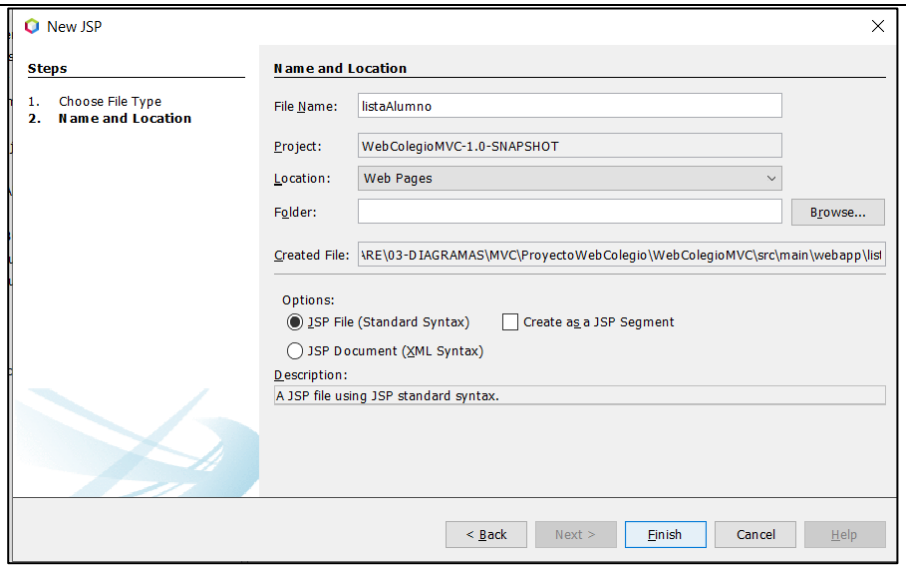
```java
package Controlador;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.util.List;
import entidad.Alumno;
import modelo.ModelAlumno;

/**
 *
 * @author Anibal
 */
@WebServlet(name = "ListaAlumnoServlet", urlPatterns = {"/ListaAlumnoServlet"})
public class ListaAlumnoServlet extends HttpServlet {
```

```
87
88      protected void service(HttpServletRequest request, HttpServletResponse response)
89              throws ServletException, IOException {
90
91              //1 Se obtiene una arraylist con los
92              //alumnos de la base de datos
93              String prmApellidos;
94              prmApellidos = request.getParameter("txtApellidos");
95              ModelAlumno m = new ModelAlumno();
96              List<Alumno> data = m.listaAlumno(prmApellidos);
97
98              //2 Se guarda el arraylist en request con el alias alumnos
99              //  El request es una memoria temporal
100             request.setAttribute("alumnos", data);
101
102             //3 Se reenvia el request al formulario
103             request.getRequestDispatcher("/listaAlumno.jsp").
104                                             forward(request, response);
105
106     }
107
108 }
```

e) Ahora vamos a implementar la **Vista**. Vamos a Crear las páginas HTML y JSP, hacemos clic derecho sobre el proyecto -> Nuevo -> JSP.



Ponemos el nombre de **listaAlumno**



Agregamos el siguiente código:

```
<!DOCTYPE html>
<%@page import="entidad.Alumno"%>
<%@page import="java.util.List"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Lista Alumnos</title>
<link rel="stylesheet" type="text/css" href="css/facilito.css" />

</head>
<body>
    <table>
            <caption class="grilla_titulo"> Lista de Alumnos</caption>

             <tr class="grilla_cabecera">
                    <th>Id</th><th>Nombre</th><th>Apellido</th><th>Edad</th>
             </tr>
             <!--
                    Scriplet: son inserciones de codigo java dentro un JSP    < % %>
                    Expression: son resultados de codigo java que se va visualizar en el JSP
              -->
        <%
            List<Alumno> a = (List<Alumno>)request.getAttribute("alumnos");
            if(a != null)
            {
                for(Alumno aux :a)
                {
        %>
```

```
                    <tr class="grilla_campo">
                        <td><%= aux.getIdAlumno() %></td>
                        <td><%= aux.getNombre()  %></td>
                        <td><%= aux.getApellido() %></td>
                        <td><%= aux.getEdad() %></td>
                    </tr>

            <%      }
                } %>

        </table>
    </body>
    </html>
```

Modificamos la página **index.html**

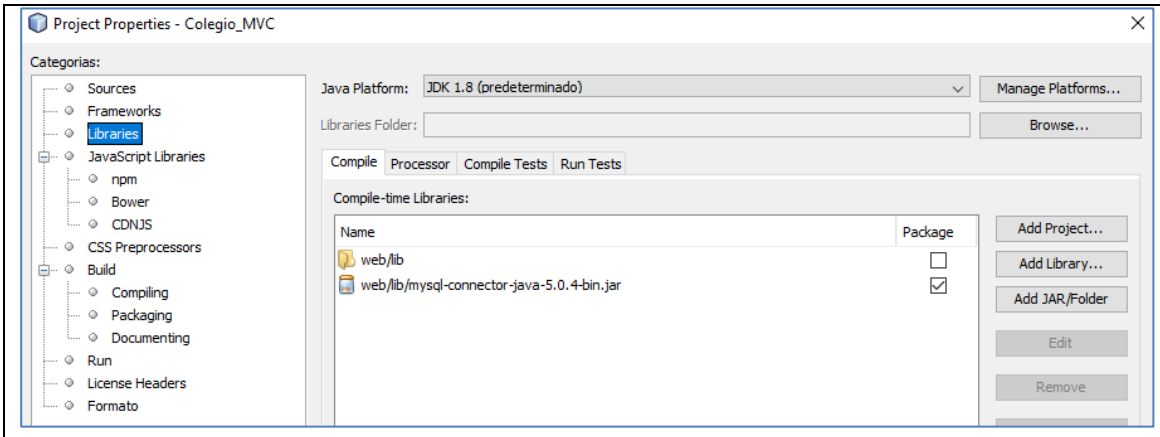
```
<!DOCTYPE html>
<!--
To change this license header, choose License Headers in Project Properties.
To change this template file, choose Tools | Templates
and open the template in the editor.
-->
<html>
    <head>
        <title>Sitio Web</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <link rel="stylesheet" type="text/css" href="css/facilito.css" />
    </head>
    <body>
        <div class="titulo">Bienvenido al sitio Web</div>

        <form action="ListaAlumnoServlet" name="frmSesion" method="post">
                <label class="loginTitle">Consulta de Alumnos</label>  <br> <br>
                <label class="loginLabel">Apellido buscado :</label>
                <input class="loginControl" type="text" name="txtApellidos">  <br>
                <input class="loginButton" type="submit" value="Enviar">
        </form>

    </body>
</html>
```

Agregamos el Driver para MySQL:

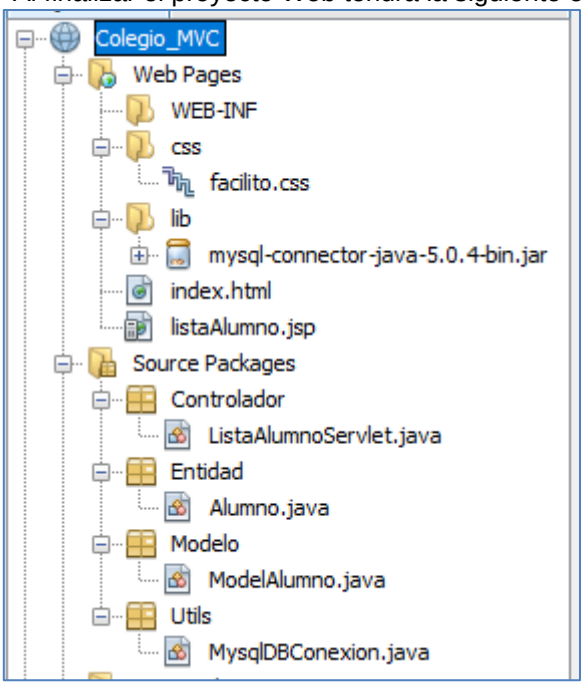
Hacemos **clic derecho** sobre el proyecto -> **propiedades.**

En la ventana de propiedades elegimos **Libraries** botón **Add JAR/Folder** y buscamos el archivo .JAR **mysql-conector-java.jar**

Elegimos y aceptamos



Al finalizar el proyecto Web tendrá la siguiente estructura:



5. Creación de la Base de Datos

Conectarse al servidor **MySQL** y ejecutar el siguiente **Script (Colegio):**

```
CREATE DATABASE  IF NOT EXISTS `colegio` /*!40100 DEFAULT CHARACTER SET utf8 */;
USE `colegio`;
-- MySQL dump 10.13  Distrib 5.6.24, for Win64 (x86_64)
--
-- Host: localhost    Database: colegio
-- ------------------------------------------------------
-- Server version    5.6.26-log

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
```

```
-- Table structure for table `curso`
--

DROP TABLE IF EXISTS `curso`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `curso` (
`idCurso` int(11) NOT NULL AUTO_INCREMENT,
`nombre` varchar(45) DEFAULT NULL,
`docente` varchar(45) DEFAULT NULL,
`lugar` varchar(45) DEFAULT NULL,
PRIMARY KEY (`idCurso`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `curso`
--

LOCK TABLES `curso` WRITE;
/*!40000 ALTER TABLE `curso` DISABLE KEYS */;
INSERT INTO `curso` VALUES (1,'Matematicas','Carlos Hurtado','Sede Arequipa'),(2,'Lenguaje','Luis Felipe','Sede
Arequipa');
/*!40000 ALTER TABLE `curso` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `Alumno`
--

DROP TABLE IF EXISTS `Alumno`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `Alumno` (
`idAlumno` int(11) NOT NULL AUTO_INCREMENT,
`nombre` varchar(45) DEFAULT NULL,
`apellido` varchar(45) DEFAULT NULL,
`edad` int(11) DEFAULT NULL,
PRIMARY KEY (`idAlumno`)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `Alumno`
--

LOCK TABLES `Alumno` WRITE;
/*!40000 ALTER TABLE `Alumno` DISABLE KEYS */;
INSERT INTO `Alumno` VALUES
(1,'Luciana','Carpio',18),(2,'Carlos','Segovia',25),(7,'Luis','Garcia',21),(8,'Roxana','Zevallos',31);
/*!40000 ALTER TABLE `Alumno` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `Administrativo`
--

DROP TABLE IF EXISTS `Administrativo`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `Administrativo` (
`cod_emp` int(11) NOT NULL AUTO_INCREMENT,
`nom_emp` varchar(30) DEFAULT NULL,
`ape_emp` varchar(30) DEFAULT NULL,
`login_emp` varchar(20) DEFAULT NULL,
`clave_emp` varchar(15) DEFAULT NULL,
PRIMARY KEY (`cod_emp`)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `Administrativo`
--

LOCK TABLES `Administrativo` WRITE;
/*!40000 ALTER TABLE `Administrativo` DISABLE KEYS */;
```

```
INSERT INTO `Administrativo` VALUES
(1,'Carlos','Macedo','cmacedo','123456'),(2,'Luis','Carpio','lcarpio','654321'),(7,'Arturo','Delgado','adelgado','12345');
/*!40000 ALTER TABLE `Administrativo` ENABLE KEYS */;
UNLOCK TABLES;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
```

8. ENTREGABLES

Luego de culminar el proyecto **ArquitecturaMVC_Colegio** y que funcione las páginas JSP de la aplicación MVC, realice los siguientes entregables:

ENTREGABLES

1. Agregar al modelo las Clases para (patrón MVC, para realizar las operaciones de búsqueda y de inserción de **tabla Cursos.**

2. Generar el código de las clases en JAVA y terminar de implementar las clases para que la Aplicación Web pueda hacer consultas e insertar nuevos **Cursos**.

9. FUENTES DE INFORMACIÓN COMPLEMENTARIA

- Cervantes, H & Velasco,P (2016). Arquitectura de Software, Conceptos y ciclo de Desarrollo. Mexico D.F. Cengage Learning Editores.
- Somerville, I.(2015). Ingeniería de Software. Madrid, España. Pearson. 7ma. Edición.
- Pressman, R.(2015). Ingeniería de Software, un enfoque práctico. Mexico DF. Mc Graw Hill. 7ma. Edición.