****

**UNIVERSIDAD DE LAS FUERZAS ARMADAS  
ESPE**

Departamento de Ciencias de la Computación



Programación Orientada a Objetos

Principios SOLID

**Nombre:** Jorge Galarza Cruz  
**Docente:** Tatiana Gualotuña

**NRC:** 4024  
**Fecha de entrega:** 24 de Junio del 2019

Sangolquí  
S-I: Marzo 2019 – Julio 2019

**Contenido**

[**1.** **Introducción:** 2](#_Toc14333116)

[**2.** **Objetivos:** 3](#_Toc14333117)

[Objetivo General: 3](#_Toc14333118)

[Objetivos Específicos: 3](#_Toc14333119)

[**3.** **Desarrollo** 3](#_Toc14333120)

[**4.** **Conclusiones** 18](#_Toc14333121)

[**5.** **Recomendaciones** 18](#_Toc14333122)

[**6.** **Bibliografía** 18](#_Toc14333123)

# **Introducción:**

El uso de patrones de diseño en un problema de programación facilita la forma de entender a dicho problema, además de ello, permite al programador realizar la reutilización del código mediante diseños, permite al programa resistirse al cambio, mejorar la calidad, interpretación, etc.

En general, la gran ventaja del uso de patrones es que minimizan el riesgo de generar un mal diseño y permiten comunicar experiencias entre diseñadores.

Por otra parte, la utilización de MVC, permite dividir la lógica del proyecto, haciéndolo más entendible, facilita la abstracción en base de datos, controlar los recursos e incluso ayuda a controlar problemas que se pueden presentar en el proyecto, El uso de MVC, es muy útil si lo usamos acompañado de patrones de diseño, como Observador, fachada, etc.

# **Objetivos:**

## Objetivo General:

Solucionar el problema planteado en clase mediante el uso de patrones de diseño, y MVC.

## Objetivos Específicos:

* Interpretar el mejor uso de un patrón de diseño.
* Solucionar el problema aplicando BD MySql.
* Aplicar MVC, acompañado de patrones y base de datos.

# **Desarrollo**

**Problema:**

La facturación de un taller mecánico requiere de un conjunto de clases que modelen las piezas que se utilizan en una reparación. Sobre cada PIEZA se debe reflejar su número de pieza, su nombre, su precio y el tiempo de mano de obra que se facturará por su instalación.

Para ciertas reparaciones se utilizan conjuntos de piezas denominados «kits» (aceite, tapa y filtro de aceite para un cambio de aceite). Los «kits» tienen su propio número, nombre y tiempo de mano de obra, pero su precio resulta de la suma de los precios de las piezas que lo componen menos un 10 % de descuento.

Considerando que nombre y número de pieza son muy estables, el precio de las piezas se actualiza con frecuencia y que las piezas que componen un «kit» pueden a su vez set «kits», elaborar el diagrama de clases de la solución propuesta para esta situación e implementar totalmente las clases que modelan los tipos de piezas.

Se pide:

• Realizar el patrón MVC

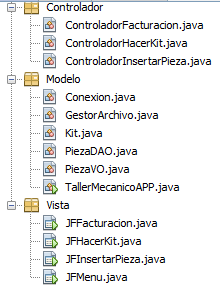
• Aplicar el patrón de diseño que crea conveniente para lograr un aplicativo efectivo

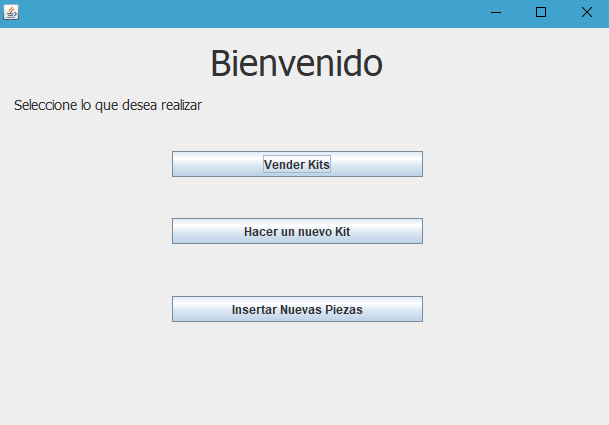
**Interpretación del problema:**

Nos pide crear la facturación de un taller mecánico, en el que se venden kits de piezas, estas piezas tienen sus respectivos atributos, al igual que los kits.

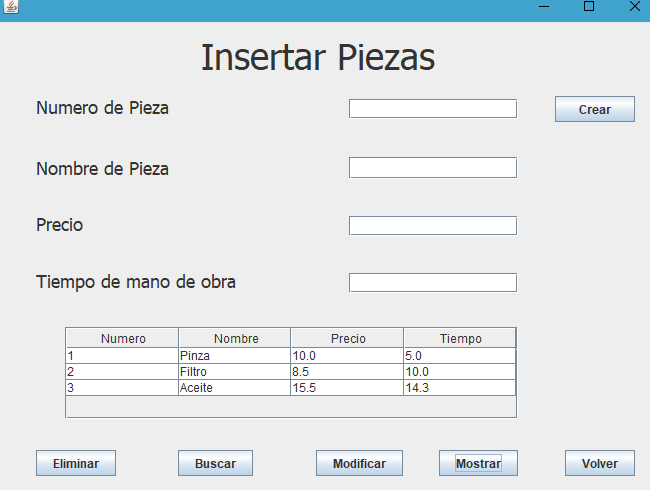
Desde este momento, se aplica el patrón de diseño fachada, ya que habrá un menú que controle la interacción entre el usuario y el programa, de tal manera que pueda elegir a su gusto la elección deseada.

Sin embargo, antes de eso debemos considerar el MVC, y crear las respectivas clases que nos permite llevar a cabo este problema.



Realizaremos 4 Interfaces, siendo JFMenu la fachada del programa.

Empezando por JFInsertarPieza que permite insertar nuevas piezas con sus atributos para mandar a la venta.

Tendremos los botones de crear, Eliminar, Buscar, Modificar, Mostrar y Volver para regresar al fachada.

Cada uno de estos botones viene dado por el patrón de diseño **OBSERVADOR.** Que se encuentra en la clase ControladorInsertarPieza, esta clase se encarga de mantener una comunicación entre el Modelo y la Vista, sin ser una comunicación directa entre las ambas, el código del controlador es el siguiente:

package Controlador;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.ArrayList;

import javax.swing.table.DefaultTableModel;

import Vista.JFInsertarPieza;

import Modelo.\*;

public class ControladorInsertarPieza implements ActionListener{

JFInsertarPieza objVista = new JFInsertarPieza();

PiezaDAO objDAO= new PiezaDAO();

PiezaVO objVO= new PiezaVO();

public ControladorInsertarPieza(JFInsertarPieza objVista)

{

this.objVista = objVista;

this.objVista.btnCrear.addActionListener(this);

this.objVista.btnEliminar.addActionListener(this);

this.objVista.btnModificar.addActionListener(this);

this.objVista.btnMostrar.addActionListener(this);

this.objVista.btnBuscar.addActionListener(this);

}

@Override

public void actionPerformed(ActionEvent e) {

if(e.getSource()==objVista.btnCrear)

{

System.out.println("Ingreso en crear");

int numero = Integer.parseInt(objVista.txtNumeroPieza.getText());

String nombre = objVista.txtNombrePieza.getText();

double precio = Double.parseDouble(objVista.txtPrecio.getText());

double tiempo = Double.parseDouble(objVista.txtTiempo.getText());

objVO = new PiezaVO(numero,nombre,precio, tiempo);

objDAO.insertPieza(objVO);

objVista.txtNumeroPieza.setText("");

objVista.txtNombrePieza.setText("");

objVista.txtPrecio.setText("");

objVista.txtTiempo.setText("");

}else if(e.getSource()==objVista.btnEliminar)

{

int numerofila = objVista.jTable1.getSelectedRow();

ArrayList<PiezaVO> arr = new ArrayList<>();

arr = objDAO.showPieza();

objVO = arr.get(numerofila);

objDAO.deletePieza(objVO);

}else if(e.getSource()==objVista.btnBuscar)

{

int codigo = Integer.parseInt(objVista.txtNumeroPieza.getText());

PiezaVO aux = objDAO.searchPieza(codigo);

if(aux.getNumerodepieza()== codigo){

objVista.txtNombrePieza.setText(aux.getNombre());

objVista.txtPrecio.setText(Double.toString(aux.getPrecio()));

objVista.txtTiempo.setText(Double.toString(aux.getTiempo()));

}

else{

System.out.println("No existe el administrador con ese codigo");

}

}else if (e.getSource()==objVista.btnModificar)

{

int numero = Integer.parseInt(objVista.txtNumeroPieza.getText());

String nombre = objVista.txtNombrePieza.getText();

double precio = Double.parseDouble(objVista.txtPrecio.getText());

double tiempo = Double.parseDouble(objVista.txtTiempo.getText());

objVO = new PiezaVO(numero,nombre,precio, tiempo);

objDAO.modifyPieza(objVO);

}else if (e.getSource()==objVista.btnMostrar)

{

ArrayList<PiezaVO> arr = new ArrayList<>();

arr = objDAO.showPieza();

System.out.println(arr);

DefaultTableModel modelo = new DefaultTableModel();

objVista.jTable1.setModel(modelo);

modelo.addColumn("Numero");

modelo.addColumn("Nombre");

modelo.addColumn("Precio");

modelo.addColumn("Tiempo");

Object[] columna = new Object[4];

for( int i=0; i<arr.size();i++){

columna[0] = arr.get(i).getNumerodepieza();

columna[1] = arr.get(i).getNombre();

columna[2] = arr.get(i).getPrecio();

columna[3] = arr.get(i).getTiempo();

modelo.addRow(columna);

}

objVista.txtNumeroPieza.setText("");

objVista.txtNombrePieza.setText("");

objVista.txtPrecio.setText("");

objVista.txtTiempo.setText("");

}

}

}

La siguiente interfaz, fue JFHacerKit, que nos permite realizar un kit con sus atributos y con las piezas que hemos creado en la anterior interfaz. Se usa de igual manera su respectivo CONTROLADOR, que observará la interacción del usuario con la maquina.

Se introduce en numero de kit, el nombre de kit y el tiempo de mano de obra. De igual forma se despliega una tabla con todas las piezas disponibles. Al seleccionar una pieza y pulsar el boton “AgregarPieza”, se irán añadiendo piezas al kit, una vez que se agregen todas las piezas deseadas, se creará un nuevo kit.

Codigo de ControladorJFHacerKit

package Controlador;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.ArrayList;

import javax.swing.table.DefaultTableModel;

import Vista.JFHacerKit;

import Modelo.\*;

import java.io.File;

import java.util.Arrays;

public class ControladorHacerKit implements ActionListener{

JFHacerKit objVista = new JFHacerKit();

PiezaDAO objDAO= new PiezaDAO();

PiezaVO objVO= new PiezaVO();

Kit objKit = new Kit();

public ControladorHacerKit(JFHacerKit objVista)

{

this.objVista = objVista;

this.objVista.btnMostrar.addActionListener(this);

this.objVista.btnAgregarPieza.addActionListener(this);

this.objVista.btnAgregarKit.addActionListener(this);

}

@Override

public void actionPerformed(ActionEvent e) {

if(e.getSource()==objVista.btnAgregarPieza)

{

int numerofila = objVista.jTable1.getSelectedRow();

System.out.println("Ingreso en crear");

PiezaVO aux = new PiezaVO();

GestorArchivo gs = new GestorArchivo();

aux = objDAO.showPieza().get(numerofila);

gs.crearPiezaArchivo(aux);

}else if(e.getSource()==objVista.btnAgregarKit)

{

ArrayList arr = new ArrayList();

File agendapiezas = new File("agendapiezas.txt");

GestorArchivo aux = new GestorArchivo();

GestorArchivo ga = new GestorArchivo();

arr = aux.listarPieza();

String auxString = Arrays.toString(arr.toArray());

int numero = Integer.parseInt(objVista.txtNumeroKit.getText());

String nombre = objVista.txtNombreKit.getText();

double tiempo = Double.parseDouble(objVista.txtTiempo.getText());

objKit = new Kit(numero,nombre,tiempo,auxString);

System.out.println("kit agregado");

ga.crearKit(objKit);

agendapiezas.delete();

}else if (e.getSource()==objVista.btnMostrar)

{

ArrayList<PiezaVO> arr = new ArrayList<>();

System.out.println("holaaa");

arr = objDAO.showPieza();

objVista.jTable1.setModel(modelo);

modelo.addColumn("Numero");

modelo.addColumn("Nombre");

modelo.addColumn("Precio");

modelo.addColumn("Tiempo");

Object[] columna = new Object[4];

for( int i=0; i<arr.size();i++){

columna[0] = arr.get(i).getNumerodepieza();

columna[1] = arr.get(i).getNombre();

columna[2] = arr.get(i).getPrecio();

columna[3] = arr.get(i).getTiempo();

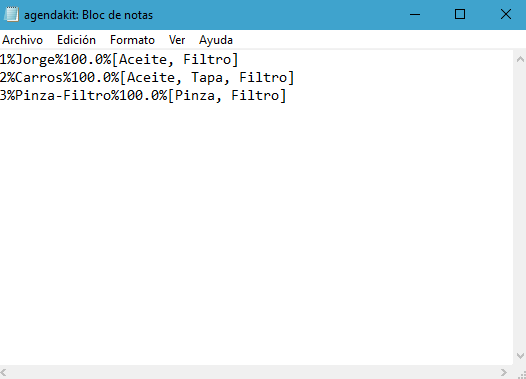
modelo.addRow(columna);

}

}

}

}

Con la intención de mantener guardados los Kits se guardarán en un archivo plano, dado por sus atributos y los nombres de las piezas utilizadas.

Debido a esto, y respetando los principios SOLID, se crea una nueva clase que se encarga únicamente de gestionar archivos, llamada GestorArchivos que vienen dado por el siguiente código:

package Modelo;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileReader;

import java.io.FileWriter;

import java.util.ArrayList;

public class GestorArchivo {

public void crearPiezaArchivo(PiezaVO nuevo)

{

try{

File f = new File("agendapiezas.txt");

FileWriter fw;

BufferedWriter bw;

if(f.exists())

{

fw = new FileWriter(f,true);

bw = new BufferedWriter(fw);

bw.newLine();

bw.write(nuevo.getNombre());

}

else{

fw = new FileWriter(f,true);

bw = new BufferedWriter(fw);

bw.write(nuevo.getNombre());

}

bw.close();

}

catch(Exception e)

{

}

}

public void crearKit(Kit nuevo)

{

try{

File f = new File("agendakit.txt");

FileWriter fw;

BufferedWriter bw;

if(f.exists())

{

fw = new FileWriter(f,true);

bw = new BufferedWriter(fw);

bw.newLine();

bw.write(nuevo.getNumero()+ "%" + nuevo.getNombre()+ "%" + nuevo.getTiempoManoObra()+

"%" + nuevo.getNombrepiezas());

}

else{

fw = new FileWriter(f,true);

bw = new BufferedWriter(fw);

bw.write(nuevo.getNumero()+ "%" + nuevo.getNombre()+ "%" + nuevo.getTiempoManoObra()+

"%" + nuevo.getNombrepiezas());

}

bw.close();

}

catch(Exception e)

{

}

}

public ArrayList listarPieza()

{

ArrayList listaC = new ArrayList();

try{

File f = new File("agendapiezas.txt");

if(f.exists())

{

FileReader fr = new FileReader(f);

BufferedReader br = new BufferedReader(fr);

String linea;

while ((linea = br.readLine()) != null)

{

listaC.add(linea);

}

br.close();

}

else

{

System.out.println("Archivo no disponible");

}

}

catch(Exception e)

{

}

return listaC;

}

public ArrayList<Kit> listarKit()

{

ArrayList<Kit> listaC = new ArrayList<>();

try{

File f = new File("agendakit.txt");

if(f.exists())

{

FileReader fr = new FileReader(f);

BufferedReader br = new BufferedReader(fr);

String linea;

while ((linea = br.readLine()) != null)

{

String [] contacto = linea.split("%");

Kit c = new Kit(Integer.parseInt(contacto[0]), contacto[1],

Double.parseDouble(contacto[2]), contacto[3]);

listaC.add(c);

}

br.close();

}

else

{

System.out.println("Archivo no disponible");

}

}

catch(Exception e)

{

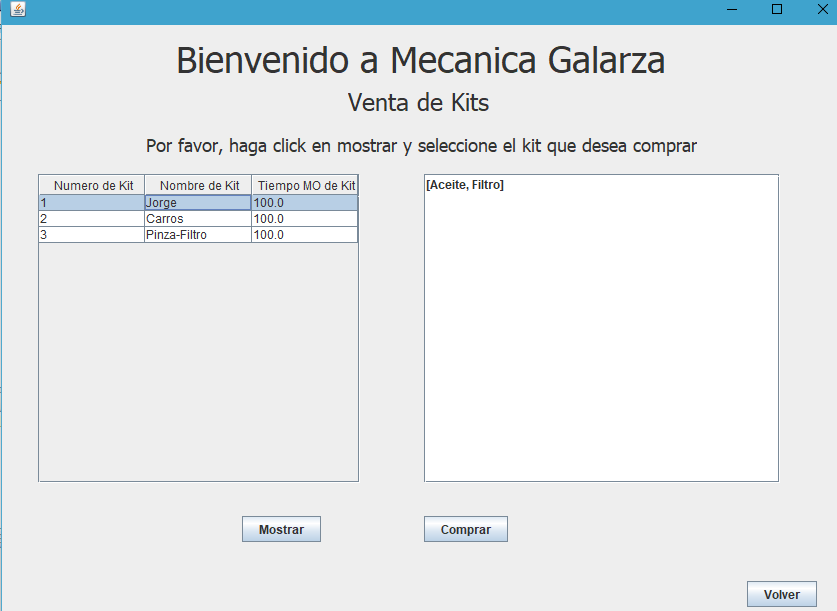
}

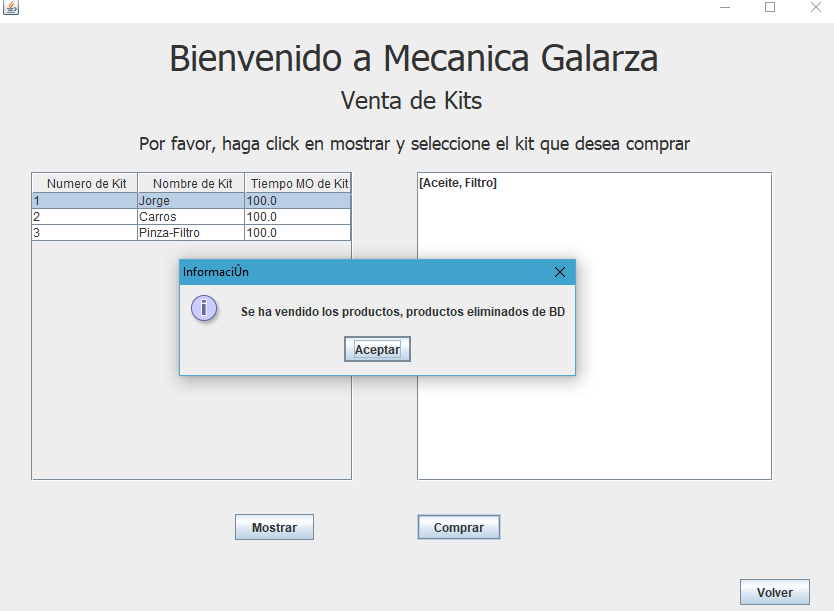
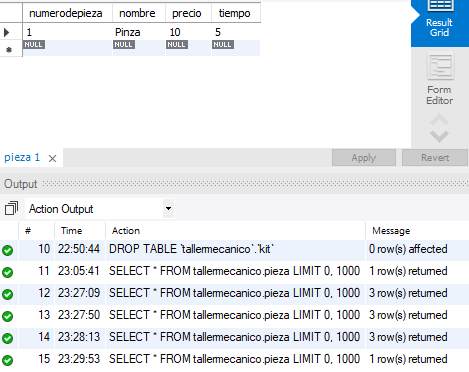
return listaC;

}

}

JFFacturacion actuará como un simulador de facturación de los kits, en esta interfaz: en la tabla se encontrará todos los kits que hemos creado, y a su lado derecho una lista de las Piezas que tienen los respectivos kits.



  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
Al comprar el producto, se aplica el descuento y se borra de la base de datos las piezas compradas, ya que ya no habrá más en su inventario.

El código de su controlador viene dado de la siguiente forma:

package Controlador;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.ArrayList;

import javax.swing.table.DefaultTableModel;

import Vista.JFFacturacion;

import Modelo.\*;

import java.io.File;

import javax.swing.DefaultListModel;

public class ControladorFacturacion implements ActionListener{

JFFacturacion objVista = new JFFacturacion();

PiezaDAO objDAO= new PiezaDAO();

PiezaVO objVO= new PiezaVO();

Kit objKit = new Kit();

public ControladorFacturacion(JFFacturacion objVista)

{

this.objVista = objVista;

this.objVista.btnMostrar.addActionListener(this);

this.objVista.btnComprar.addActionListener(this);

}

@Override

public void actionPerformed(ActionEvent e) {

if(e.getSource()==objVista.btnComprar)

{

int numerofila = objVista.jTable1.getSelectedRow();

ArrayList<Kit> arr = new ArrayList<>();

Kit aux = new Kit();

GestorArchivo ga = new GestorArchivo();

arr = ga.listarKit();

String aux1 = arr.get(numerofila).getNombrepiezas();

String aux2;

String[] aux3 = new String[objDAO.showPieza().size()];

for(int i = 0 ; i < objDAO.showPieza().size(); i++)

{

aux2 = objDAO.showPieza().get(i).getNombre();

if(aux1.contains(aux2))

{

aux3[i] = aux2;

}

}

for(int i = 0 ; i<aux3.length; i++)

{

System.out.println(aux3[i]);

}

objDAO.deletePiezabyName(aux3);

System.out.println("Gracias por comprar");

}else if (e.getSource()==objVista.btnMostrar)

{

ArrayList<Kit> arr = new ArrayList<>();

GestorArchivo gs = new GestorArchivo();

arr = gs.listarKit();

DefaultTableModel modelo = new DefaultTableModel();

objVista.jTable1.setModel(modelo);

modelo.addColumn("Numero de Kit");

modelo.addColumn("Nombre de Kit");

modelo.addColumn("Tiempo MO de Kit");

Object[] columna = new Object[3];

for( int i=0; i<arr.size();i++){

columna[0] = arr.get(i).getNumero();

columna[1] = arr.get(i).getNombre();

columna[2] = arr.get(i).getTiempoManoObra();

System.out.println(arr.get(i));

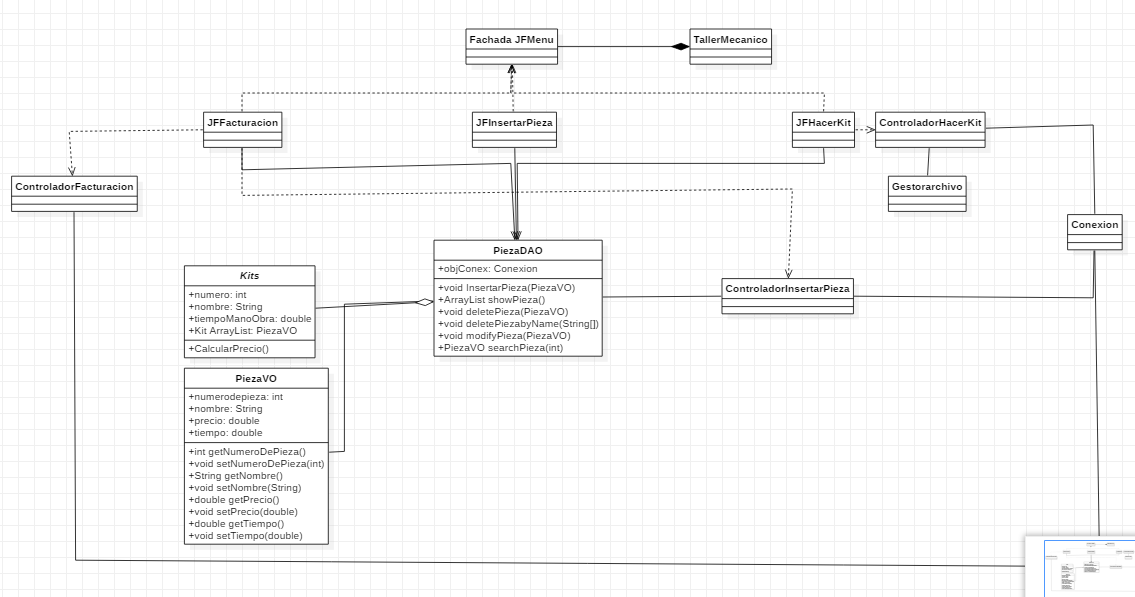
modelo.addRow(columna);

}

}

}

}

**Diagrama de clases:**

# **Conclusiones**

Cuando se desarrolla cualquier software, hay que reconocer el problema y aplicar los patrones de diseño correspondientes, de igual manera aplicar MVC, es de gran utilidad para realizar un código mas limpio y entendible.

# **Recomendaciones**

Aplicar en los problemas. todo los aprendido tales como principios SOLID, MVC, y patrones de diseño.

# **Bibliografía**

**null**