

Universidad Mariano Galvez

Ingeniero Alfredo Cruz

Programacion II



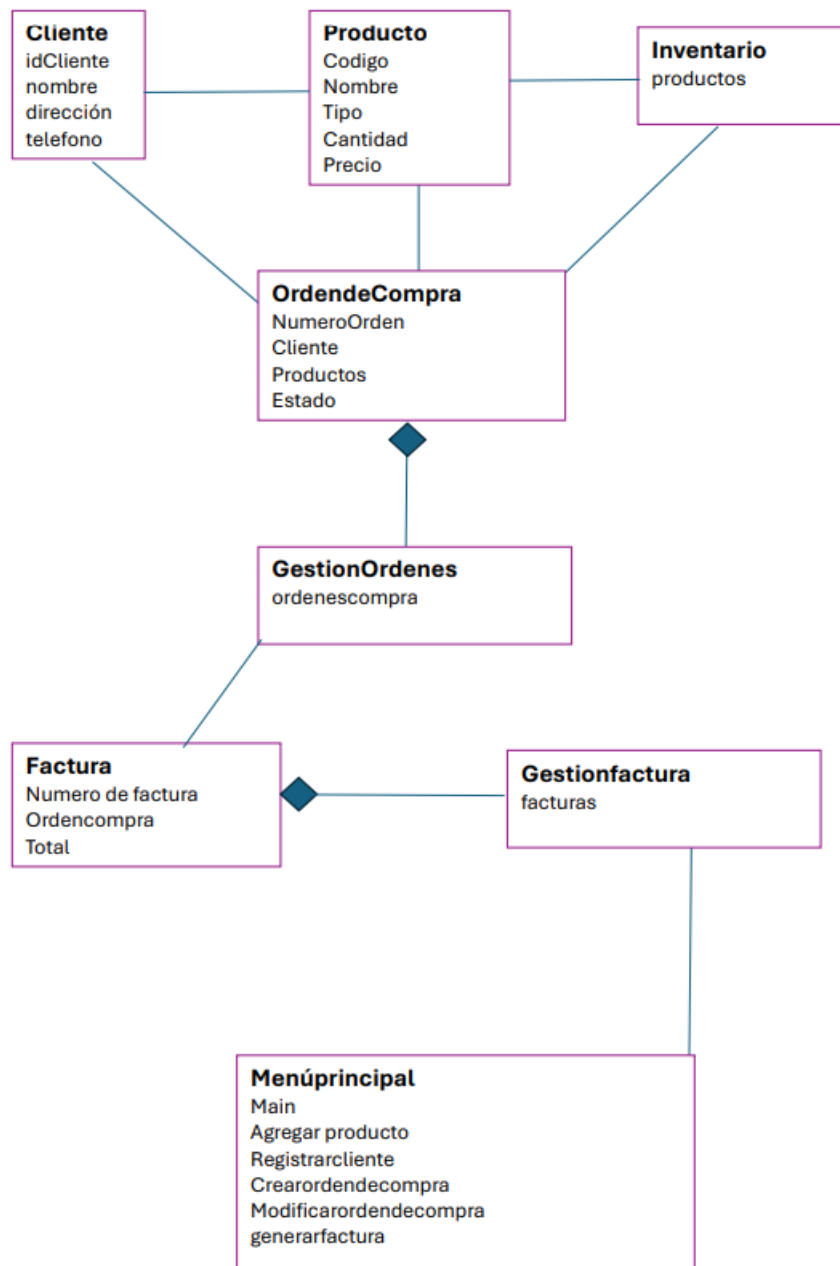
PROYECTO FINAL

Cinthia Yadira Robles Sotoj.

7690-16-13986.

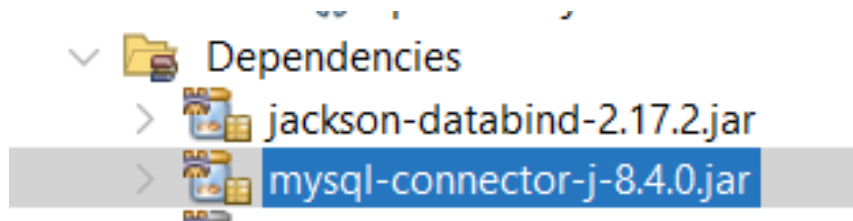
Sábados.

Diagrama de clases

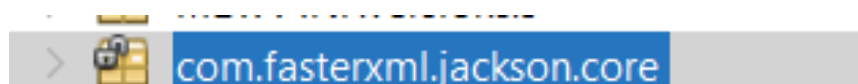


POM.XML

```
Main.java x InterfazLogicaCliente.java x ClienteGUI.java x LogicaSQLCliente.java x Cliente.java x
1 [ { effective History
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2
3   <modelVersion>4.0.0</modelVersion>
4   <groupId>com.mycompany</groupId>
5   <artifactId>PFcinthia</artifactId>
6   <version>1.0-SNAPSHOT</version>
7   <packaging>jar</packaging>
8   <dependencies>
9     <dependency>
10       <groupId>com.mysql</groupId>
11       <artifactId>mysql-connector-j</artifactId>
12       <version>8.4.0</version>
13     </dependency>
14     <dependency>
15       <groupId>com.fasterxml.jackson.core</groupId>
16       <artifactId>jackson-databind</artifactId>
17       <version>2.17.2</version>
18     </dependency>
19   </dependencies>
20   <properties>
21     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
22     <maven.compiler.source>22</maven.compiler.source>
23     <maven.compiler.target>22</maven.compiler.target>
24   </properties>
25 </project>
```

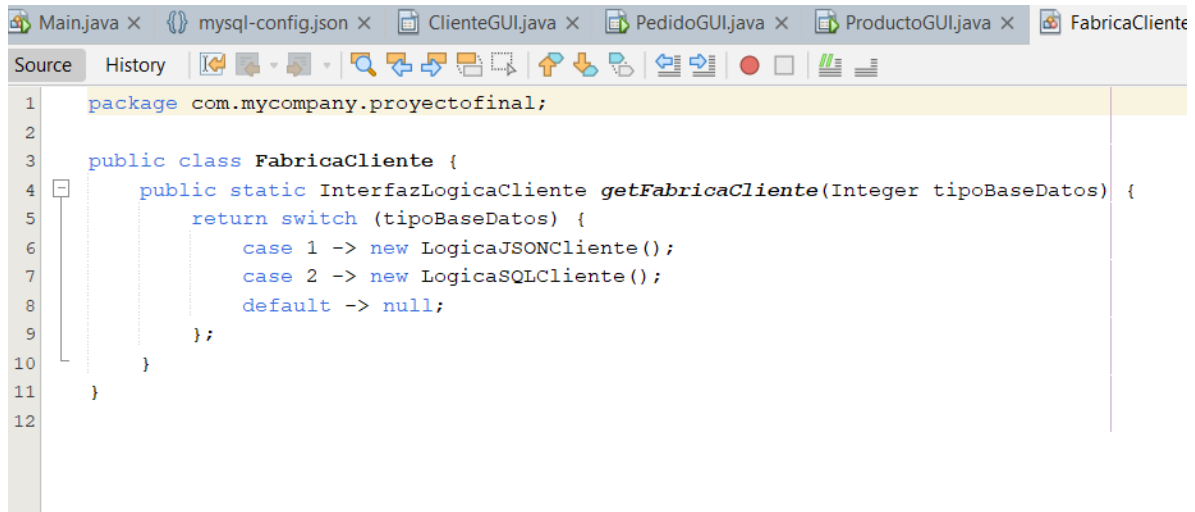


Estas librerías las utilice para poder realizar la conexión con la base de datos.



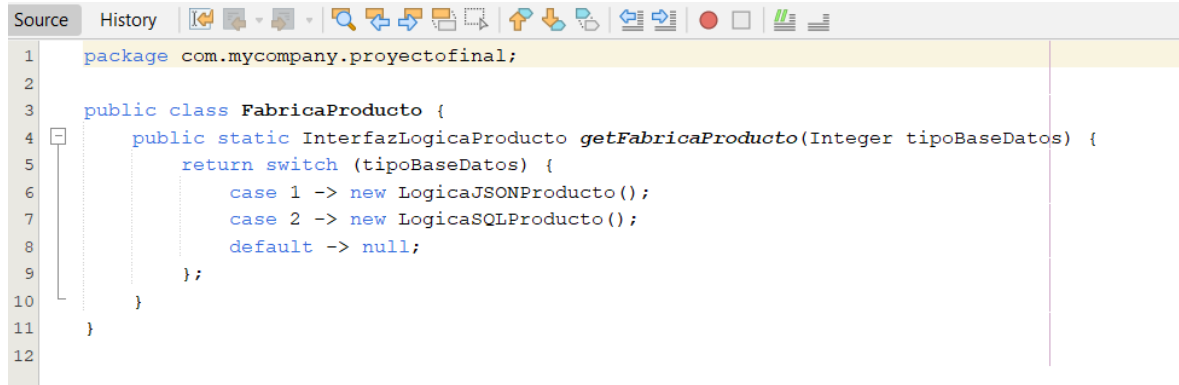
Esta librería la utilice para manejar los Json.

Patrones de diseño Factory para la clase cliente



```
1 package com.mycompany.proyectofinal;
2
3 public class FabricaCliente {
4     public static InterfazLogicaCliente getFabricaCliente(Integer tipoBaseDatos) {
5         return switch (tipoBaseDatos) {
6             case 1 -> new LogicaJSONCliente();
7             case 2 -> new LogicaSQLCliente();
8             default -> null;
9         };
10    }
11 }
12
```

Patrones de diseño Factory para la clase Producto



```
1 package com.mycompany.proyectofinal;
2
3 public class FabricaProducto {
4     public static InterfazLogicaProducto getFabricaProducto(Integer tipoBaseDatos) {
5         return switch (tipoBaseDatos) {
6             case 1 -> new LogicaJSONProducto();
7             case 2 -> new LogicaSQLProducto();
8             default -> null;
9         };
10    }
11 }
12
```

Este patrón sirve para almacenar y mostrar datos, ya sea en una base de datos MySQL o en formato JSON.

Patrones de diseño segregación de la interfaz para la clase Producto

```

package com.mycompany.proyectofinal;

import java.io.IOException;
import java.sql.SQLException;
import java.util.List;
import javax.swing.JTable;

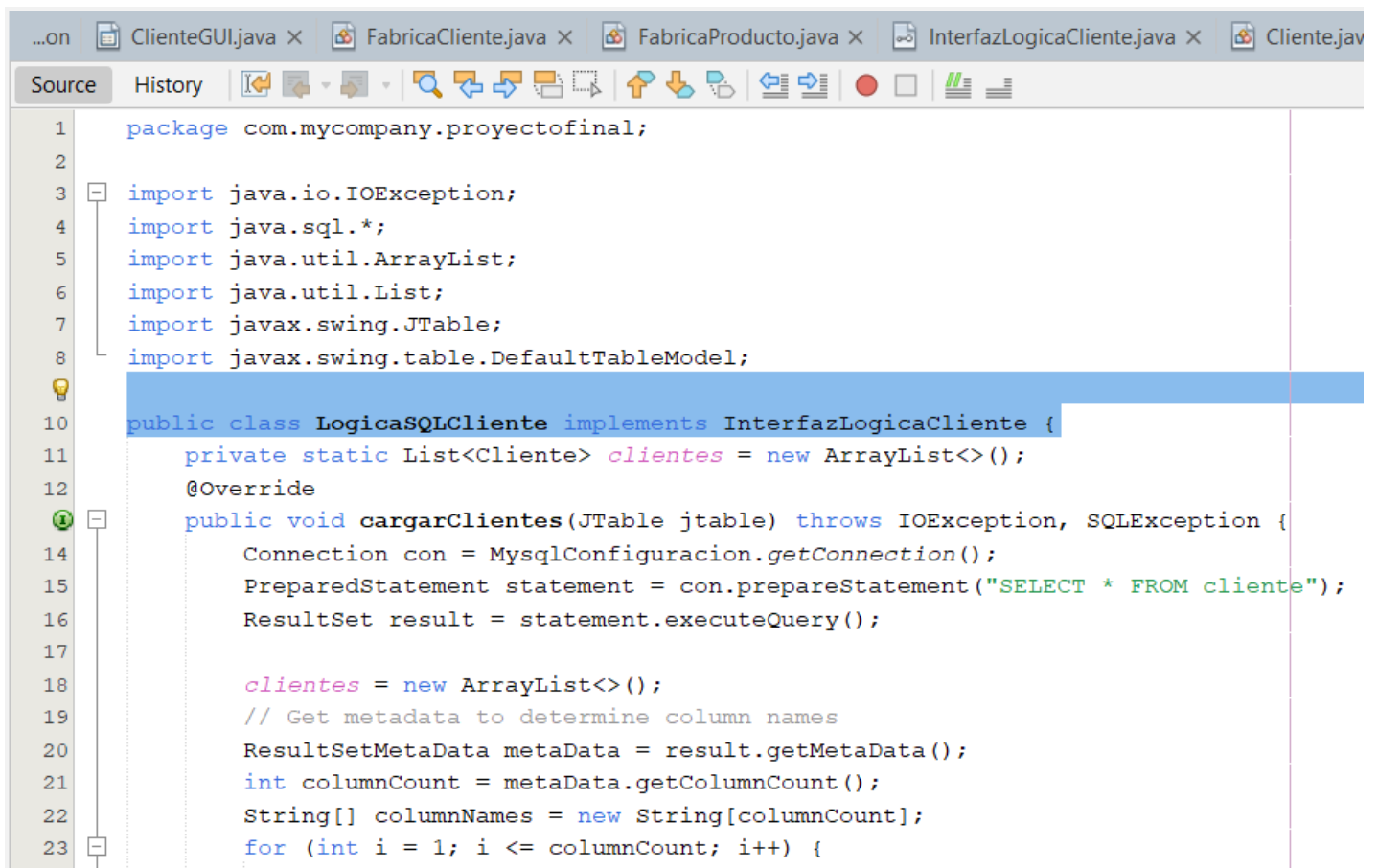
public interface InterfazLogicaCliente {
    void cargarClientes(JTable table) throws IOException, SQLException;
    void guardarClientes() throws IOException;

    void agregarCliente(Cliente cliente, Integer productoId) throws SQLException, IOException;

    List<Cliente> getClientes();
}

```

PARA LA BASE DE DATOS A CREAR



```

1 package com.mycompany.proyectofinal;
2
3 import java.io.IOException;
4 import java.sql.*;
5 import java.util.ArrayList;
6 import java.util.List;
7 import javax.swing.JTable;
8 import javax.swing.table.DefaultTableModel;
9
10 public class LogicaSQLCliente implements InterfazLogicaCliente {
11     private static List<Cliente> clientes = new ArrayList<>();
12     @Override
13     public void cargarClientes(JTable jtable) throws IOException, SQLException {
14         Connection con = MysqlConfiguracion.getConnection();
15         PreparedStatement statement = con.prepareStatement("SELECT * FROM cliente");
16         ResultSet result = statement.executeQuery();
17
18         clientes = new ArrayList<>();
19         // Get metadata to determine column names
20         ResultSetMetaData metaData = result.getMetaData();
21         int columnCount = metaData.getColumnCount();
22         String[] columnNames = new String[columnCount];
23         for (int i = 1; i <= columnCount; i++) {

```

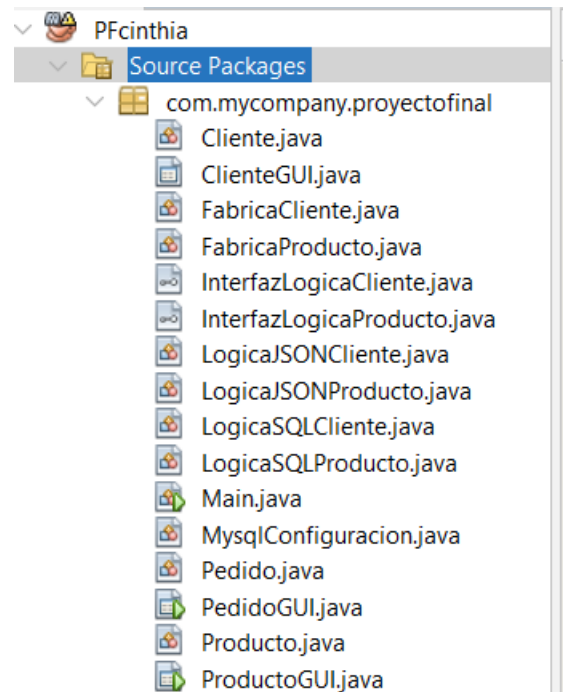
IMPLEMENTACIÓN DE LA INTERFAZ “LÓGICA CLIENTE” PARA MYSQL

```
Main.java x InterfazLogicaCliente.java x ClienteGUI.java x LogicaSQLCliente.java x
Source History
1 package com.mycompany.proyectofinal;
2
3 import java.io.IOException;
4 import java.sql.*;
5 import java.util.ArrayList;
6 import java.util.List;
7 import javax.swing.JTable;
8 import javax.swing.table.DefaultTableModel;
9
10 public class LogicaSQLCliente implements InterfazLogicaCliente {
11     private static List<Cliente> clientes = new ArrayList<>();
12     @Override
13     public void cargarClientes(JTable jTable) throws IOException, SQLException {
14         Connection con = MysqlConfiguracion.getConnection();
15         PreparedStatement statement = con.prepareStatement("SELECT * FROM cliente");
16         ResultSet result = statement.executeQuery();
17
18         clientes = new ArrayList<>();
19         // Get metadata to determine column names
20         ResultSetMetaData metaData = result.getMetaData();
21         int columnCount = metaData.getColumnCount();
22         String[] columnNames = new String[columnCount];
23         for (int i = 1; i <= columnCount; i++) {
```

IMPLEMENTACIÓN DE LA SEGREGACIÓN DE LA INTERFAZ “LÓGICA CLIENTE” PARA JSON

```
...on ClienteGUI.java x FabricaCliente.java x FabricaProducto.java x InterfazLogicaCliente.java x Cliente.java x Lo
Source History
2
3 import com.fasterxml.jackson.databind.ObjectMapper;
4
5 import java.io.File;
6 import java.io.IOException;
7 import java.net.URL;
8 import java.util.ArrayList;
9 import java.util.HashMap;
10 import java.util.List;
11 import java.util.Map;
12 import javax.swing.JTable;
13 import javax.swing.table.DefaultTableModel;
14
15 public class LogicaJSONCliente implements InterfazLogicaCliente {
16     private static List<Cliente> clientes = new ArrayList<>();
17     private static Map<String, String> map = new HashMap<>();
18
19     public void cargarClientes(JTable jTable) throws IOException {
20         URL url = new URL("file:src/main/resources/cliente.json");
21
22         ObjectMapper mapper = new ObjectMapper();
23         Cliente[] clientesTmp = mapper.readValue(new File(url.getPath()), Cliente[].class);
24         clientes = new ArrayList<>();
```

PATRÓN ÚNICA RESPONSABILIDAD, SE UTILIZA PARA SEPARAR LA LÓGICA POR MÓDULOS

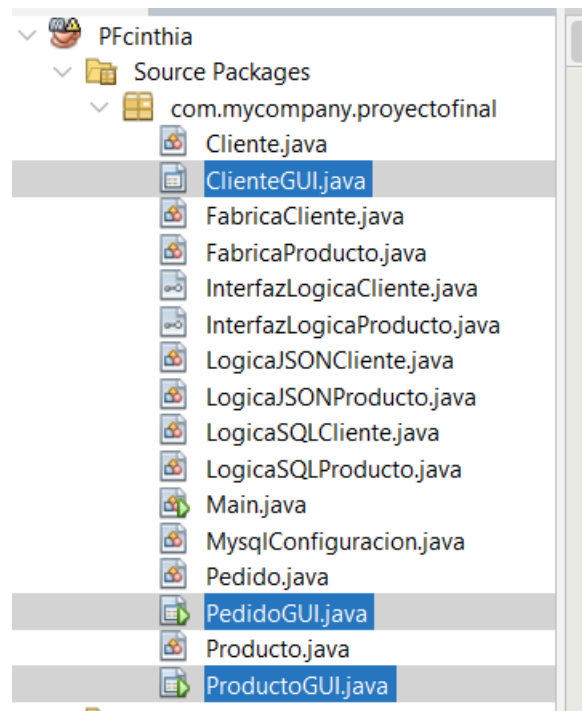


ESTRUCTURA BASE DE DATOS

```
SQL File 4* x cliente producto cliente
Limit to 1000 rows

4
5
6 • CREATE TABLE producto (
7     productoId int auto_increment primary key,
8     nombre varchar(150),
9     stock integer,
10    estado varchar(100)
11 );
12
13
14 • CREATE TABLE cliente (
15     nitCliente varchar(100) primary key ,
16     nombreCliente varchar(100),
17     direccion varchar(100),
18     celular varchar(100)
19 );
20
21 • CREATE TABLE pedido (
22     pedidoId int primary key auto_increment,
23     fechaOrden datetime,
24     estado varchar(100);
```

GUI'S, Cliente, Pedido, Producto



Utilice estos GUI'S para facilitar la interacción con el programa con los siguientes comandos:

JLabel: Se usaron 4 etiquetas para describir los campos de entrada: **Nombre**, **NIT**, **Teléfono** y **Dirección**.

TextField: Se usaron 4 campos de texto correspondientes a las etiquetas anteriores, donde el usuario puede ingresar la información de **Nombre**, **NIT**, **Teléfono** y **Dirección**

JTable: Se usaron 2 tablas para mostrar la información ingresada por el usuario o para desplegar los datos necesarios.

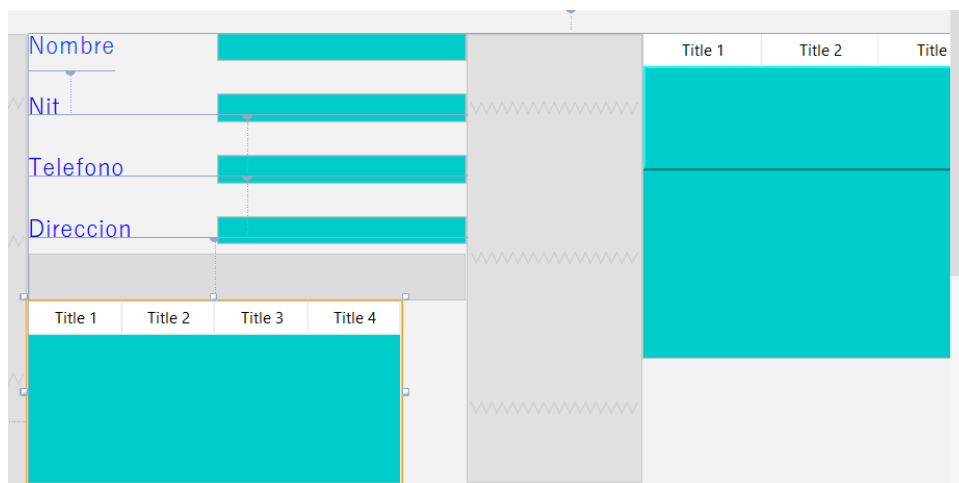


DIAGRAMA DE ENTIDAD-RELACION
PERSISTENCIA EN LA BASE DE DATOS USANDO WORKBENCH

