UNIVERSIDAD PERUANA LOS ANDES

FACULTAD DE INGENIERÍA

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

PRACTICA N°6

ASIGNATURA: ARQUITECTURA DE SOFTWARE

DOCENTE: FERNÁNDEZ BEJARANO, RAUL ENRIQUE

INTEGRANTE: AQUINO CANO, MOISES ISRAEL

CICLO: VII

HUANCAYO-2024
PERÚ

INFORME DE TRABAJO N°6

Título del proyecto

Sistema CRUD en Java utilizando MVC con SQL Server

El proyecto tiene como objetivo desarrollar un sistema de gestión de alumnos utilizando un enfoque de arquitectura MVC (Modelo-Vista-Controlador) en Java. El sistema permitirá realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre un registro de alumnos almacenado en una base de datos SQL Server, facilitando la administración eficiente de la información estudiantil. La aplicación proporcionará una interfaz gráfica de usuario (GUI) construida en Swing para gestionar los datos de los alumnos y estará conectada a una base de datos relacional que almacenará toda la información relevante.

Requerimientos funcionales

Registro de alumnos: El sistema debe permitir al usuario registrar un nuevo alumno, ingresando sus datos (nombre, apellido, DNI y teléfono).
Listado de alumnos: La aplicación debe mostrar en una tabla todos los alumnos registrados en la base de datos, permitiendo al usuario visualizar la información de manera clara y ordenada.
Edición de datos: El sistema debe permitir seleccionar un alumno de la lista y cargar sus datos en los campos de texto para su posterior modificación. Luego de editar la información, los cambios deben guardarse en la base de datos.
Eliminación de alumnos: El sistema debe permitir al usuario eliminar el registro de un alumno seleccionado de la tabla. Esta operación debe ser irreversible y eliminar el alumno de la base de datos.
Interfaz gráfica: El sistema debe contar con una interfaz gráfica intuitiva que permita a los usuarios realizar las operaciones CRUD de manera fácil. Los botones disponibles deben ser:

- **Guardar**: Almacena los datos de un nuevo alumno en la base de datos.
- **Listar**: Muestra todos los alumnos registrados.
- **Editar**: Carga los datos del alumno seleccionado para editarlos.
- **Eliminar**: Borra el registro de un alumno seleccionado.
- **OK**: Confirma los cambios al editar un alumno.

Validación de datos: El sistema debe realizar validaciones básicas, como el lDNI y el teléfono, para asegurar que los datos ingresados sean correctos antes de ser almacenados en la base de datos.
Conexión a base de datos: La aplicación debe conectarse a una base de datos SQL Server donde se almacenarán todos los registros de alumnos, utilizando la clase `Conexion` para gestionar la conexión a la base de datos.

MANUAL DEL PROYECTO:

Índice

Introducción

Objetivo: Describir de manera clara el propósito del sistema (por ejemplo, gestión de alumnos mediante un sistema CRUD). Incluir qué problema resuelve y por qué es importante.

Alcance: Definir el alcance del proyecto, mencionando sus principales funcionalidades (CRUD, conexión a base de datos SQL Server, interfaz gráfica en Java Swing, etc.).

Público objetivo: Indicar quiénes serán los principales usuarios del sistema y de este manual (desarrolladores, personal técnico, administradores del sistema).

Arquitectura del Proyecto (MVC):

- **Modelo**: Describe cómo se manejan los datos (ej. clases `Alumno` y `AlumnoDAO`).
- **Vista**: Explica la interfaz gráfica (ej. formularios Swing).
- **Controlador**: Detalla cómo se gestionan las interacciones entre la vista y el modelo (ej. clase `Controlador`).

Requisitos del Sistema

Hardware:

- Procesador: i5 o superior
- Memoria RAM: 8GB o superior
- Espacio en disco: 500MB de espacio disponible para la base de datos y la aplicación.

Software:

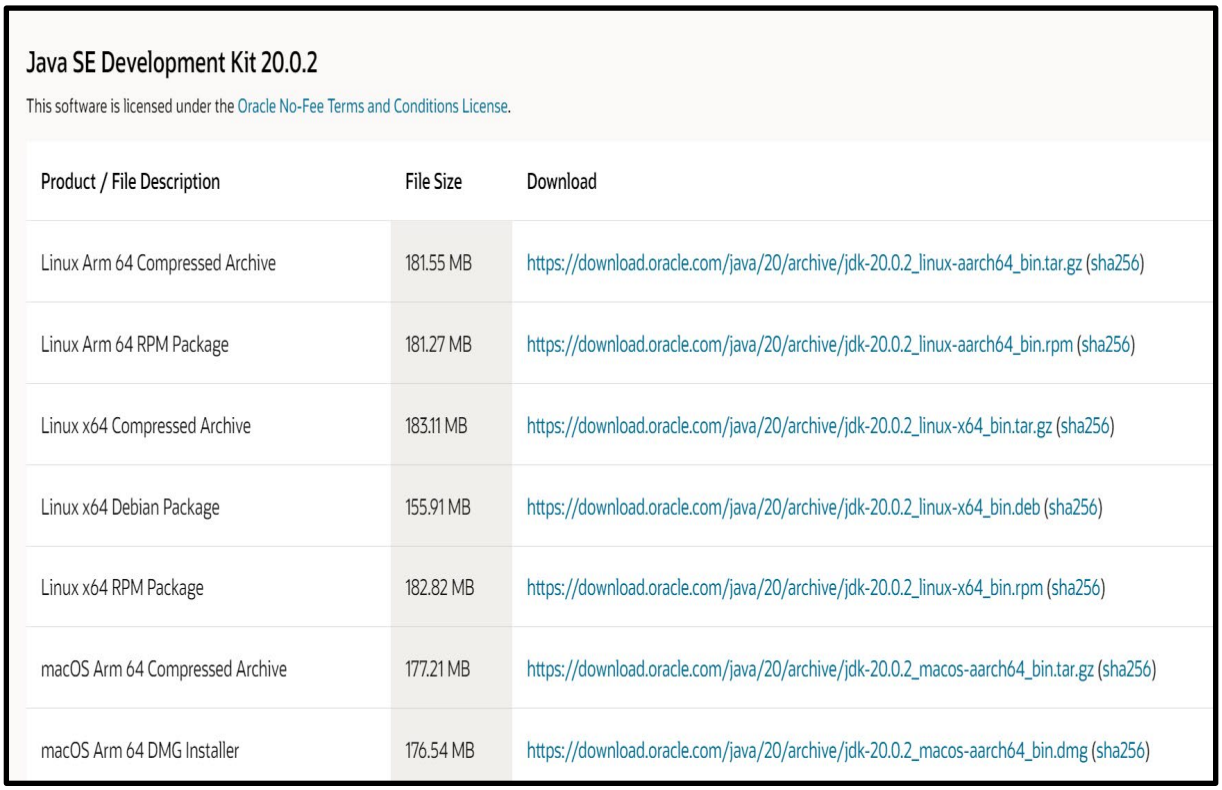
- **Java**: JDK versión 20 o superior.
- **NetBeans IDE**: Versión 14 o superior.
- **SQL Server**: Versión compatible con el driver JDBC.
- **Maven**: Gestión de dependencias y compilación.
- **Sistema Operativo**: Windows 10 o superior, o Linux

Instalación y Configuración

6.1 Instalación del Entorno de Desarrollo

Java y Maven:

1. Instalar JDK 20 desde el sitio oficial de Oracle.



Java SE Development Kit 20.0.2

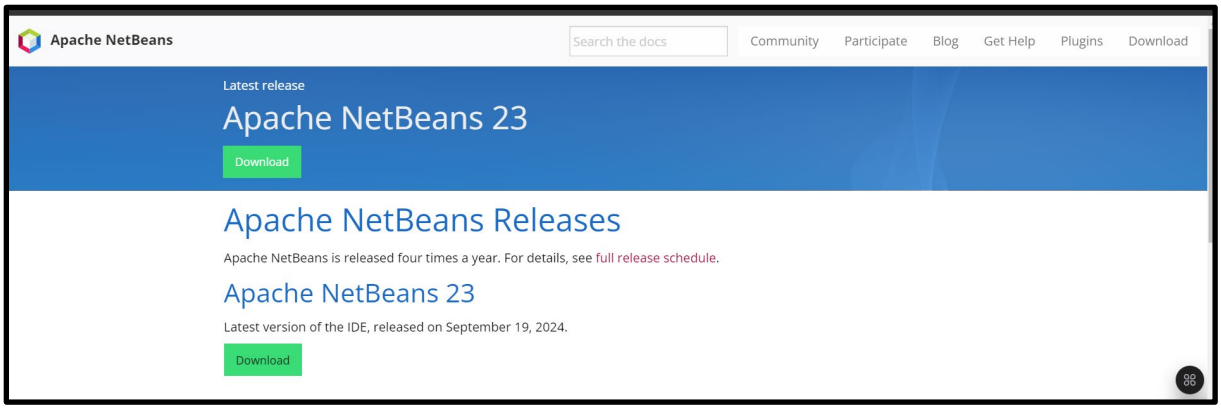
This software is licensed under the Oracle No-Fee Terms and Conditions License.

Product / File Description	File Size	Download
Linux Arm 64 Compressed Archive	181.55 MB	https://download.oracle.com/java/20/archive/jdk-20.0.2_linux-aarch64_bin.tar.gz (sha256)
Linux Arm 64 RPM Package	181.27 MB	https://download.oracle.com/java/20/archive/jdk-20.0.2_linux-aarch64_bin.rpm (sha256)
Linux x64 Compressed Archive	183.11 MB	https://download.oracle.com/java/20/archive/jdk-20.0.2_linux-x64_bin.tar.gz (sha256)
Linux x64 Debian Package	155.91 MB	https://download.oracle.com/java/20/archive/jdk-20.0.2_linux-x64_bin.deb (sha256)
Linux x64 RPM Package	182.82 MB	https://download.oracle.com/java/20/archive/jdk-20.0.2_linux-x64_bin.rpm (sha256)
macOS Arm 64 Compressed Archive	177.21 MB	https://download.oracle.com/java/20/archive/jdk-20.0.2_macos-aarch64_bin.tar.gz (sha256)
macOS Arm 64 DMG Installer	176.54 MB	https://download.oracle.com/java/20/archive/jdk-20.0.2_macos-aarch64_bin.dmg (sha256)

2. Configurar variables de entorno para JAVA_HOME y MAVEN_HOME.

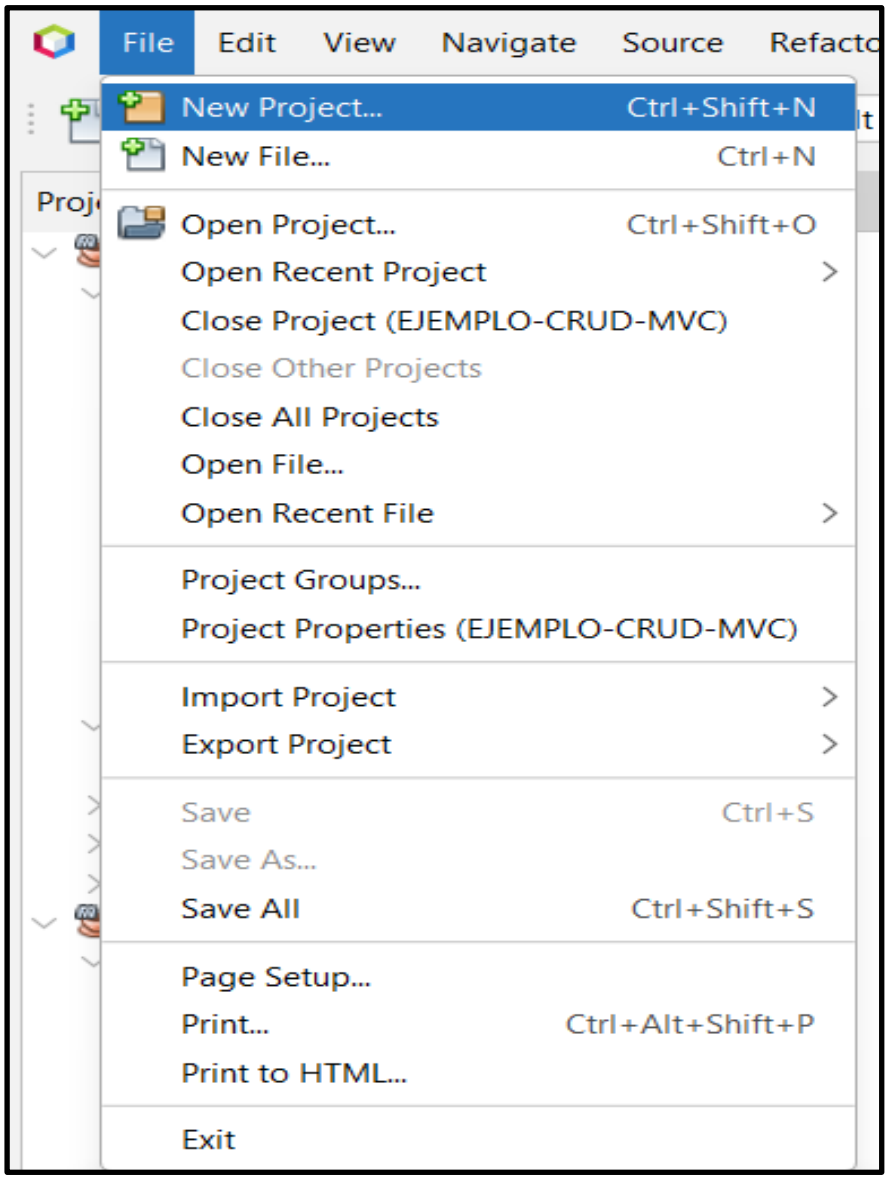
NetBeans:

1. Descargar e instalar NetBeans desde su sitio oficial.

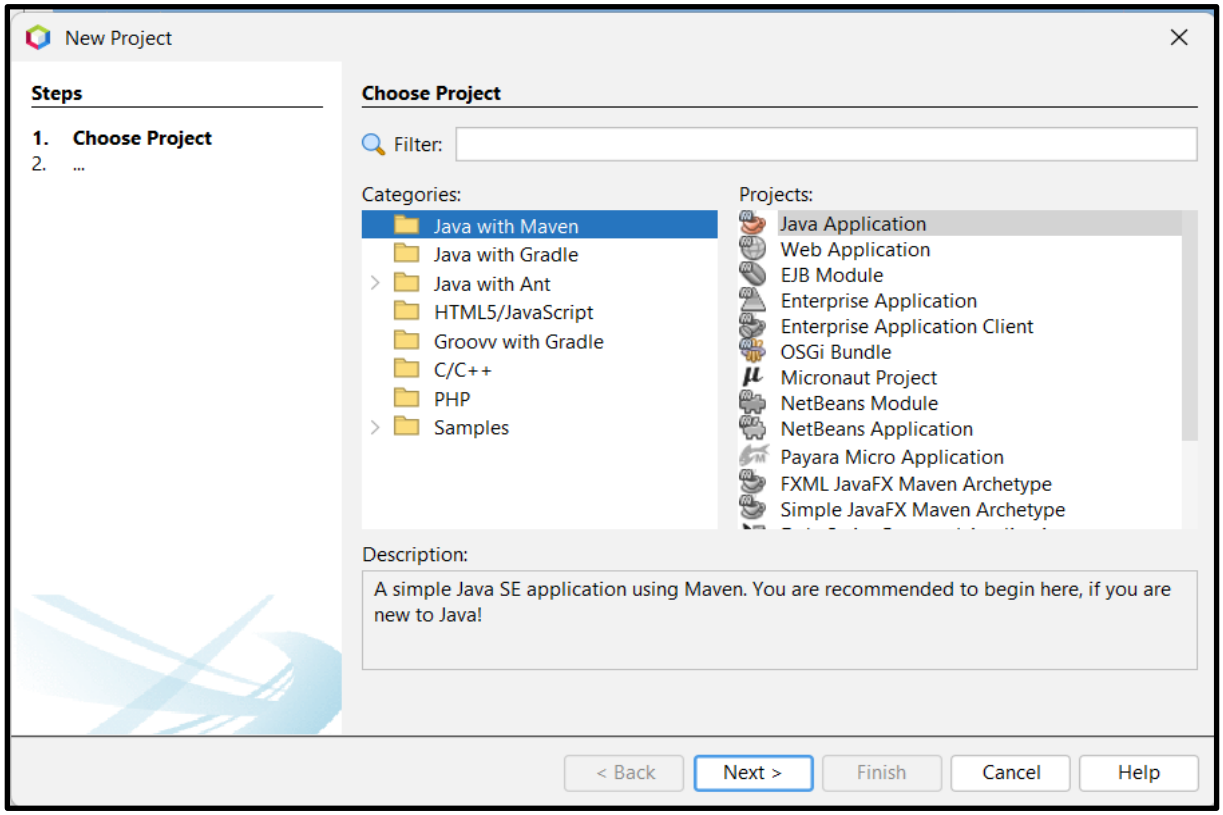


2. Configurar NetBeans para trabajar con Maven.

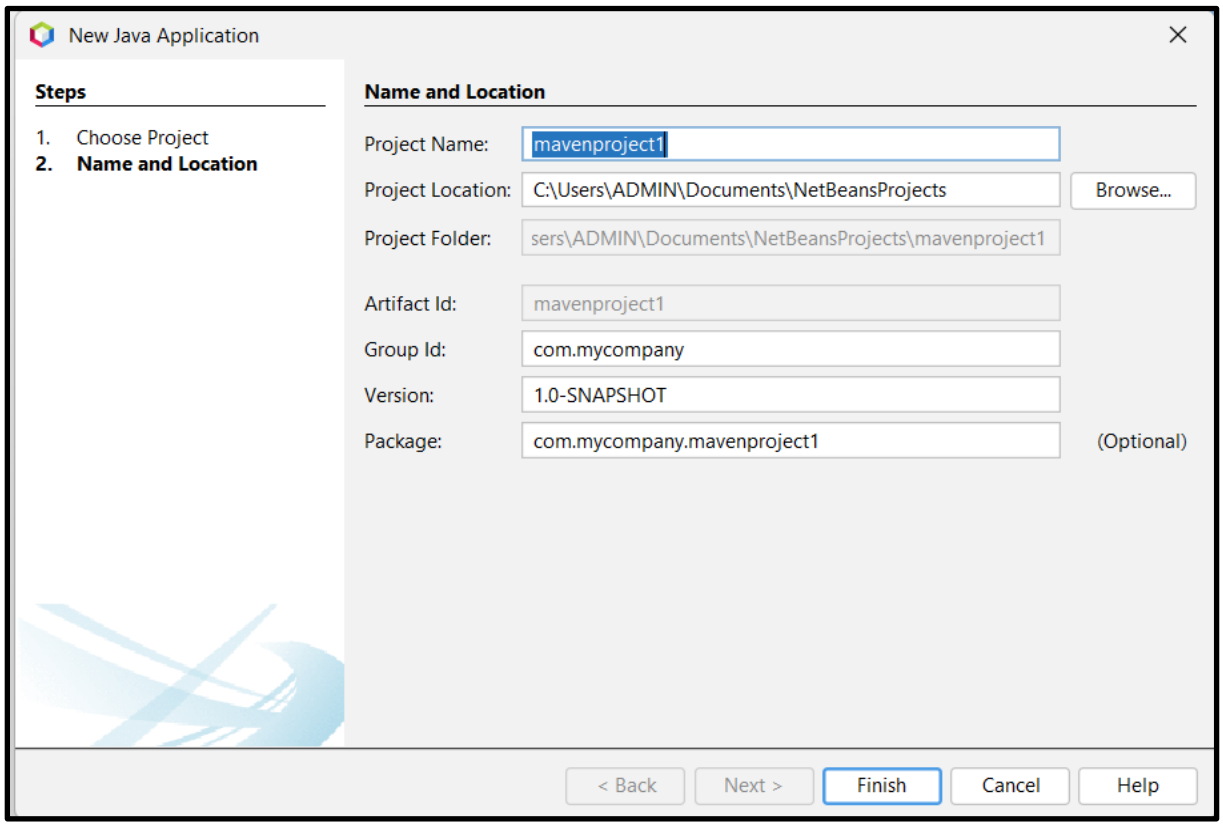
Abrimos un nuevo proyecto



Seleccionamos la tecnología Maven y Java Application



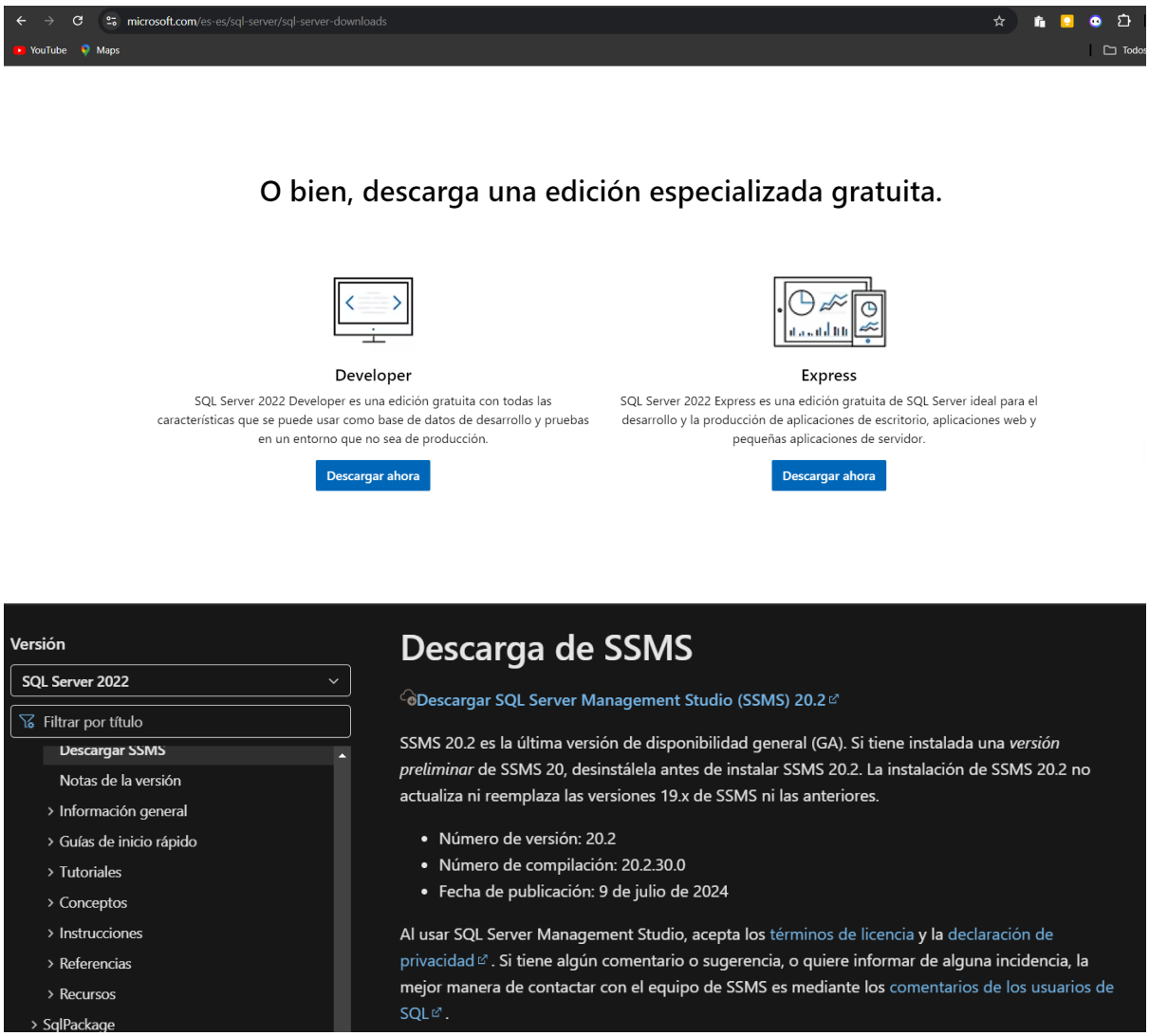
Colocas el nombre y la ubicación del proyecto



Configuración de la Base de Datos SQL Server

Pasos:

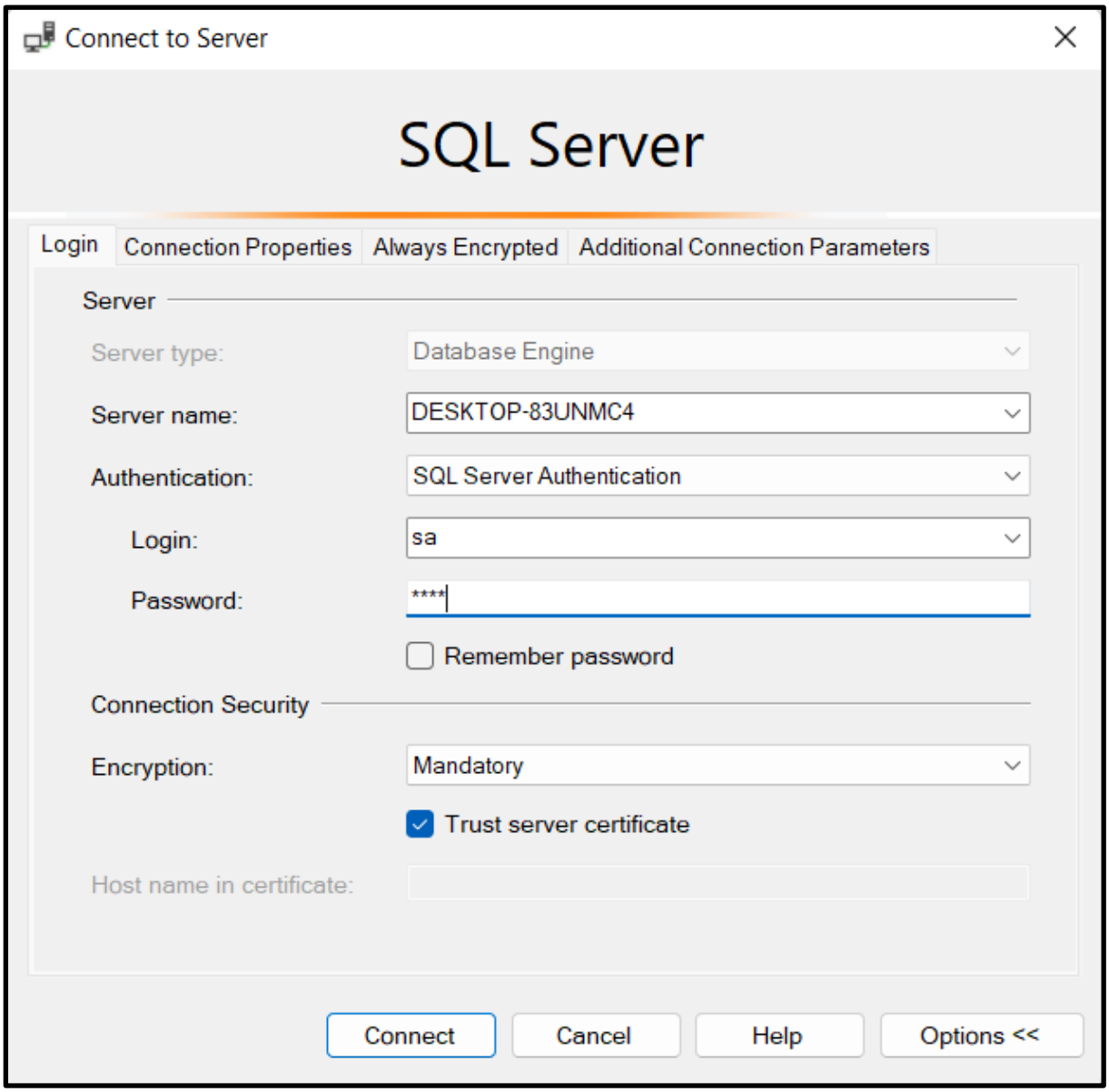
1. Instalar SQL Server.



- Instalar SQL Server Developer te permite tener un entorno de base de datos completo para desarrollar y probar aplicaciones.
- SSMS facilita la administración de bases de datos, permitiendo realizar consultas, gestionar esquemas y supervisar el rendimiento.

2. Crear la base de datos

Nos conectamos a la base de datos

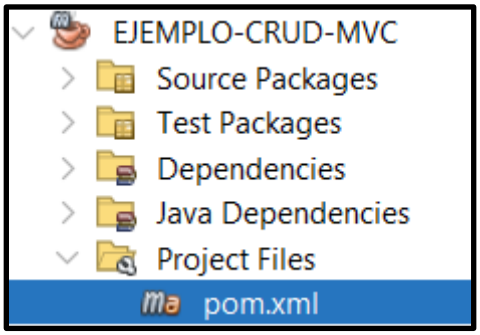


Abrimos una nueva consulta (New Query) y creamos nuestra base de datos con el nombre que desees. Y también le añadimos una tabla llamada Alumno con sus diferentes campos y sus respectivos atributos.

```sql
CREATE DATABASE EjemploCRUDDB;

USE EjemploCRUDDB;

CREATE TABLE Alumno (
    id INT PRIMARY KEY IDENTITY(1,1),
    nombre VARCHAR(100) NOT NULL,
    apellido VARCHAR(100) NOT NULL,
    dni VARCHAR(8) NOT NULL,
    telefono VARCHAR(15)
);

select * from Alumno;
```

Mantenimiento y Actualizaciones

Bien un punto importante es la agregación de dependencias debes incluir el bloque correspondiente en la sección `<dependencies>` de tu `pom.xml`




```xml
<dependencies>
    <!-- Dependencia de SQL Server JDBC Driver -->
    <dependency>
        <groupId>com.microsoft.sqlserver</groupId>
        <artifactId>mssql-jdbc</artifactId>
        <version>10.2.1.jre17</version> <!-- Versió
    </dependency>
```

El propósito de esta dependencia es permitir que tu aplicación Java se conecte y realice operaciones (como consultas, actualizaciones, etc.) en una base de datos de

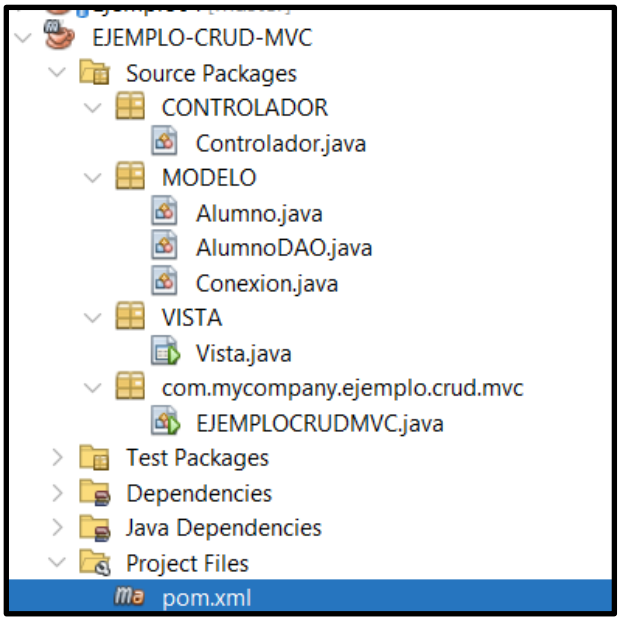
Microsoft SQL Server. Al incluir esta dependencia, Maven descargará automáticamente el controlador necesario y lo hará disponible en tu proyecto.

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package MODELO;

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

/**
 *
 * @author ADMIN
 */
public class Conexion {
    private static final String URL = "jdbc:sqlserver://localhost:1433;databaseName=EjemploCRUDDB;encrypt=false;trustServerCertificate=true;";
    private static final String USER = "sa";
    private static final String PASSWORD = "1234";
    private Connection con = null;

    public Connection getConnection() {
        try {
            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");  // Asegúrate de cargar el driver
            con = DriverManager.getConnection(URL, USER, PASSWORD);
            System.out.println("Conexión exitosa a la base de datos.");
        } catch (ClassNotFoundException | SQLException e) {
            System.err.println("Error de conexión: " + e.getMessage());
        }
        return con;
    }
}
```

Estructura del Proyecto

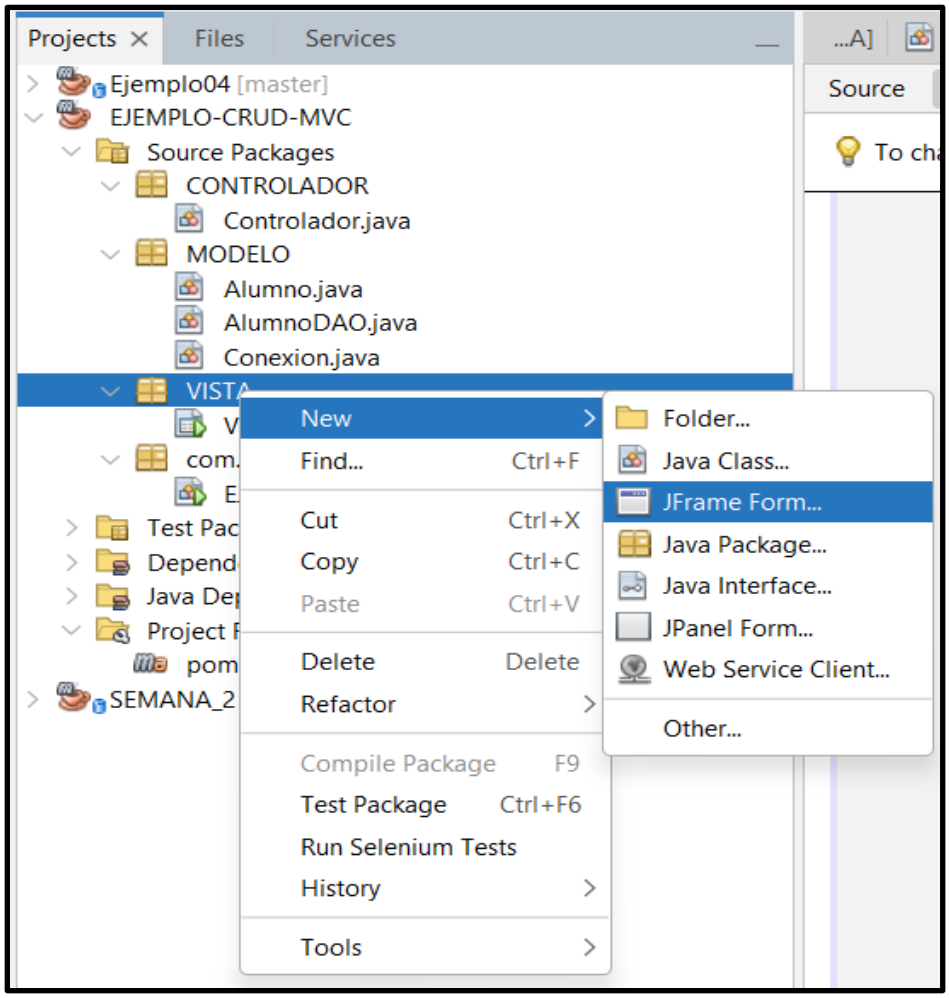
Paquetes:

- **MODELO**: Contiene las clases de conexión a la base de datos y el DAO (Data Access Object) para interactuar con SQL Server.
    - Ejemplo: `Alumno.java`, `AlumnoDAO.java`, `Conexion.java`.
- **VISTA**: Contiene la interfaz gráfica (formularios Swing).
    - Ejemplo: `Vista.java`.
- **CONTROLADOR**: Implementa la lógica de control que interactúa con la vista y el modelo.
    - Ejemplo: `Controlador.java`.

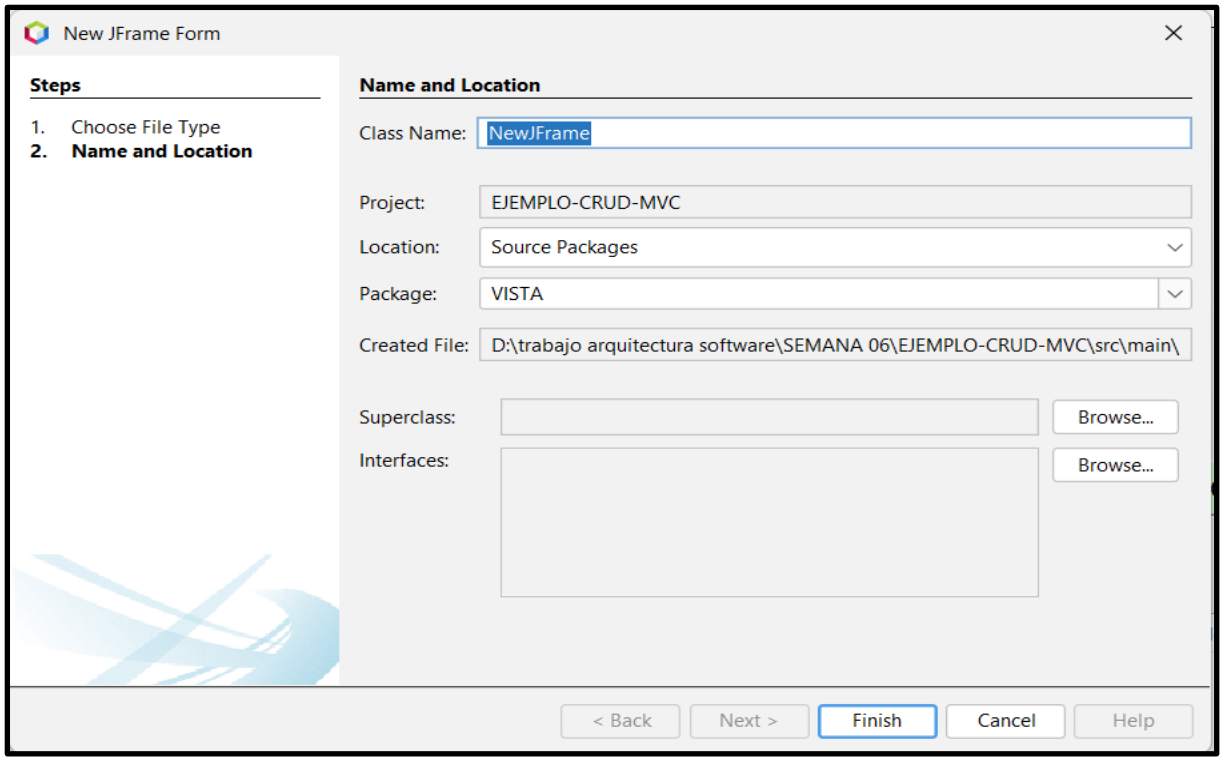


En esta creamos nuestros paquetes y dentro de ellos tenemos nuestras clases y formularios.

En el paquete de vista agregaremos un nuevo formulario



Y pondrás el el nombre que desees


New JFrame Form
Steps
1. Choose File Type
2. Name and Location
Name and Location
Class Name: NewJFrame
Project: EJEMPLO-CRUD-MVC
Location: Source Packages
Package: VISTA
Created File: D:\trabajo arquitectura software\SEMANA 06\EJEMPLO-CRUD-MVC\src\main\
Superclass:
Browse...
Interfaces:
Browse...
< Back
Next >
Finish
Cancel
Help


Design Preview [Vista]
REGISTRO
ID
NOMBRE
APELLIDO
DNI
TELEFONO
GUARDAR
LISTAR
ELIMINAR
EDITAR
OK
Id
Nombre
Apellido
DNI
Telefono

Botones del formulario

Guardar
Función: Este botón se utiliza para almacenar la información ingresada en los campos de la interfaz. Al hacer clic en él, los datos se guardan en la base de datos o en la memoria del programa.

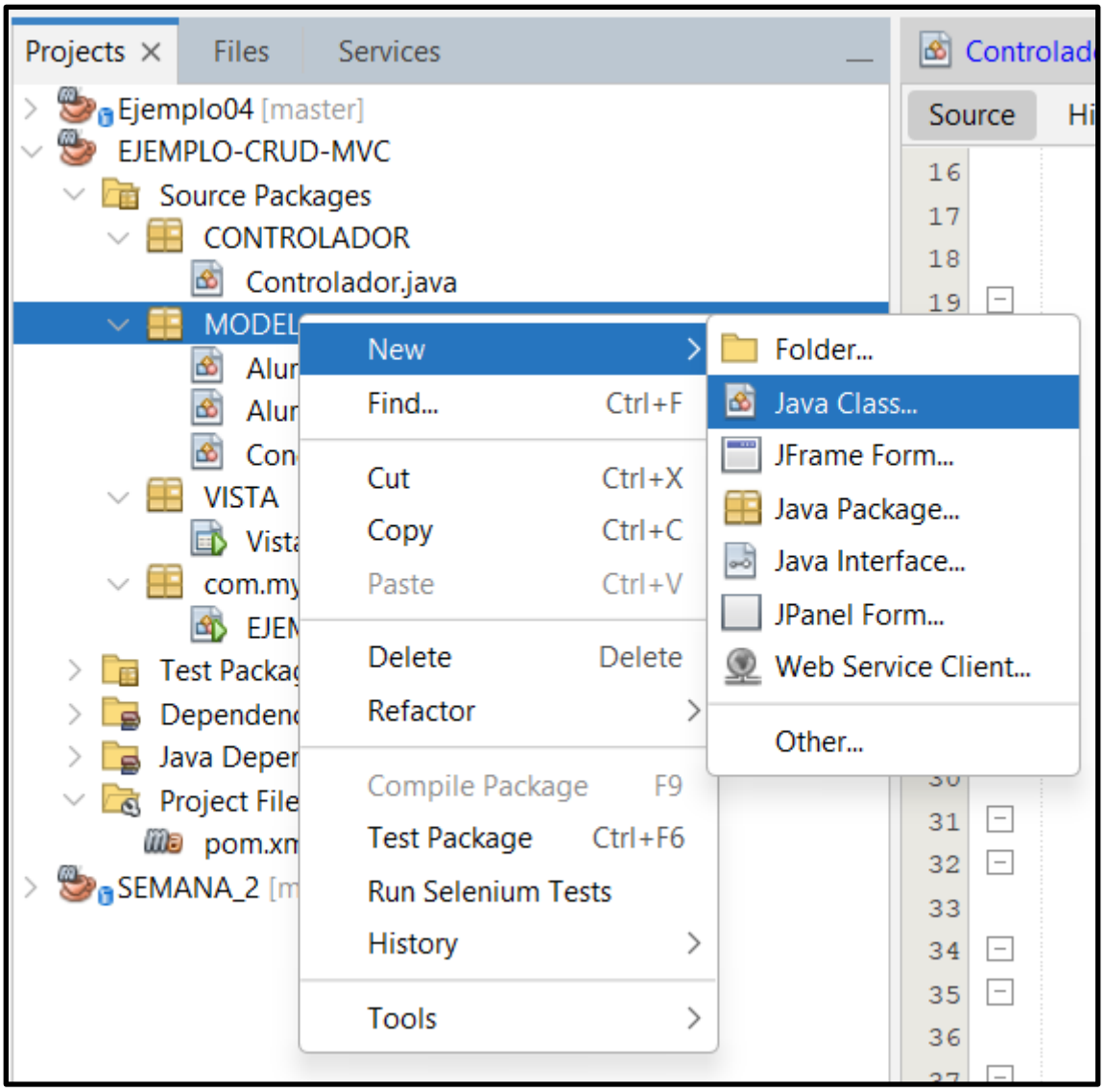
Listar
Función: Este botón permite visualizar todos los registros almacenados en la base de datos. Al hacer clic, se despliega una lista de elementos que cumplen con ciertos criterios.

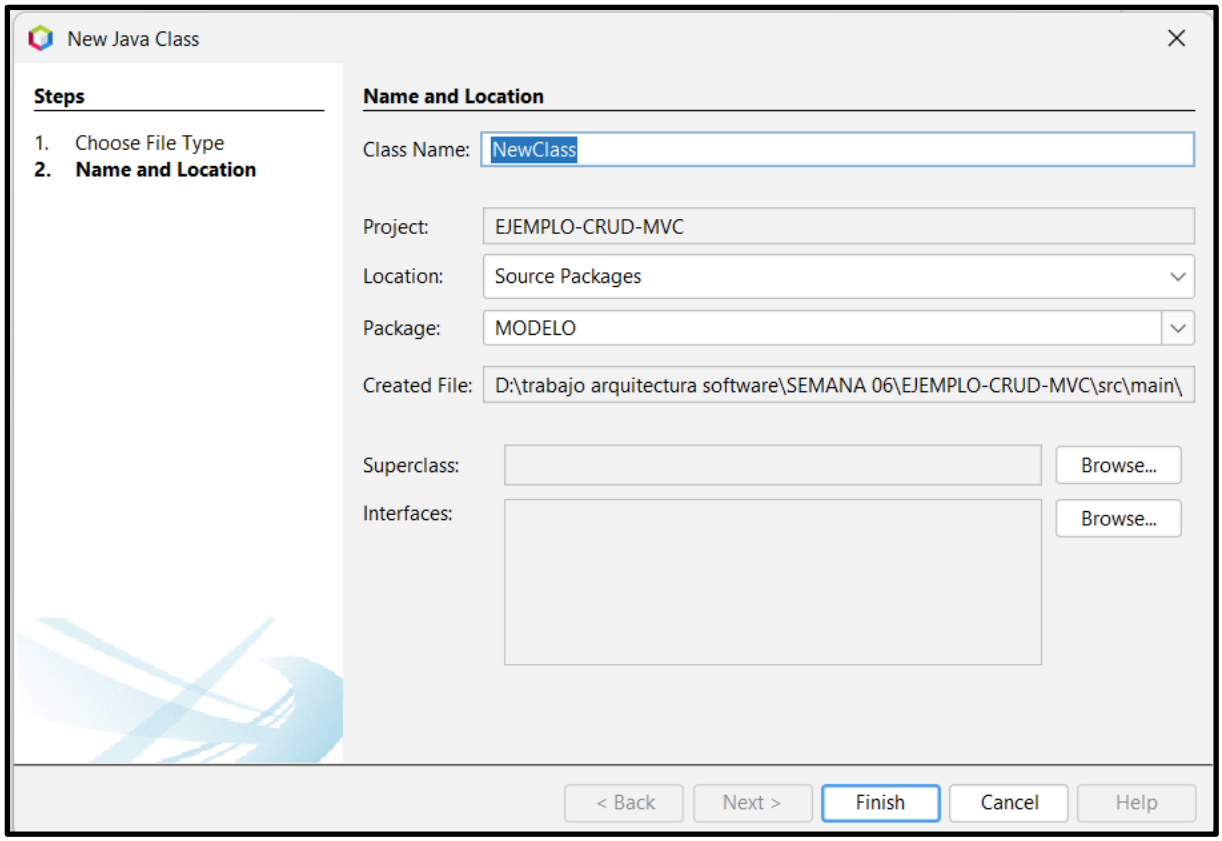
Eliminar
Función: Este botón se utiliza para borrar un registro seleccionado de la base de datos. Al hacer clic en él, se solicita confirmación antes de realizar la eliminación para evitar borrados accidentales.

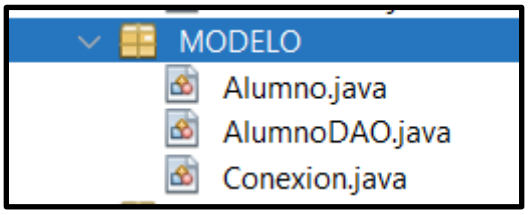
Editar
Función: Este botón permite modificar un registro previamente guardado. Al hacer clic, se habilitan los campos de entrada para que el usuario pueda realizar cambios.
Uso: Se utiliza para corregir errores o actualizar información en los registros existentes, asegurando que los datos sean precisos y estén al día.

OK
Función: Este botón se utiliza para confirmar una acción, como guardar cambios, cerrar un diálogo o aceptar una operación. Generalmente indica que el usuario ha finalizado su entrada de datos.

Después en el paquete Modelo agregaremos tres clases (Alumno, AlumnoDAO, Conexión)


Projects
Files
Services
Ejemplo04 [master]
EJEMPLO-CRUD-MVC
Source Packages
CONTROLADOR
Controlador.java
VISTA
Test Packag
Dependenc
Java Deper
Project File
pom.xm
SEMANA_2
Source
16
17
18
19
31
32
33
34
35
36
New
Find... Ctrl+F
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Refactor
Compile Package F9
Test Package Ctrl+F6
Run Selenium Tests
History
Tools
Folder...
Java Class...
JFrame Form...
Java Package...
Java Interface...
JPanel Form...
Web Service Client...
Other...





Clase Alumno

Representa la entidad "Alumno" en el sistema. Contiene los atributos que describen a un alumno. Se utiliza para crear objetos que representan a los alumnos y manejar su información en el programa.

Atributos:

- id: Identificador único del alumno (int).
- codigo: Código del alumno (String).
- nombre: Nombre del alumno (String).
- apellido: Apellido del alumno (String).
- dni: Documento Nacional de Identidad (String).
- telefono: Teléfono del alumno (String).

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package MODELO;

/**
 *
 * @author ADMIN
 */
public class Alumno {
     private int id;
    private String nombre;
    private String apellido;
    private String dni;
    private String telefono;

    // Constructor vacío
    public Alumno() {}

    // Constructor con parámetros
    public Alumno(int id, String nombre, String apellido, String dni, String telefono) {
        this.id = id;
        this.nombre = nombre;
        this.apellido = apellido;
        this.dni = dni;
        this.telefono = telefono;
    }

    // Getters y Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    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; }

    public String getDni() { return dni; }
    public void setDni(String dni) { this.dni = dni; }

    public String getTelefono() { return telefono; }
    public void setTelefono(String telefono) { this.telefono = telefono; }
}
```

Clase AlumnoDAO

Encapsula la lógica de acceso a los datos relacionados con los alumnos. Facilita las operaciones de CRUD (Crear, Leer, Actualizar, Eliminar) en la base de datos.Facilita la interacción con la base de datos, separando la lógica de negocio de la lógica de acceso a datos.

Métodos Comunes:

- guardar(Alumno alumno): Para insertar un nuevo alumno en la base de datos.
- listar(): Para recuperar todos los alumnos de la base de datos.
- eliminar(int id): Para eliminar un alumno específico.
- editar(Alumno alumno): Para actualizar la información de un alumno existente.

Facilita la interacción con la base de datos, separando la lógica de negocio de la lógica de acceso a datos.

```
package MODELO;

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

```
public class AlumnoDAO {
     Conexion conectar = new Conexion();
    Connection con;
    PreparedStatement ps;
    ResultSet rs;

    // Método para listar alumnos
    public List<Alumno> listar() {
    List<Alumno> datos = new ArrayList<>();  // Creas una lista para almacenar los datos
    String sql = "SELECT * FROM Alumno";
    try {
        con = conectar.getConnection();  // Abres la conexión
        ps = con.prepareStatement(sql);  // Preparas la consulta
        rs = ps.executeQuery();  // Ejecutas la consulta
        while (rs.next()) {
            Alumno alumno = new Alumno();  // Creas un objeto Alumno
            alumno.setId(rs.getInt(1));  // Asignas el valor de las columnas
            alumno.setNombre(rs.getString(2));
            alumno.setApellido(rs.getString(3));
            alumno.setDni(rs.getString(4));
            alumno.setTelefono(rs.getString(5));
            datos.add(alumno);  // Agregas el objeto a la lista
        }
    } catch (SQLException e) {
        System.err.println("Error al listar alumnos: " + e.getMessage());
    }
    return datos;  // Retornas la lista de alumnos
}
```

El método listar() es fundamental para obtener la información de todos los alumnos almacenados en la base de datos, permitiendo que otros componentes del sistema puedan acceder a esta información de manera organizada y eficiente.

```java
    // Método para agregar un alumno (sin el ID, porque es autoincremental)
public int agregar(Alumno alumno) {
    String sql = "INSERT INTO Alumno(nombre, apellido, dni, telefono) VALUES (?, ?, ?, ?)";
    try {
        con = conectar.getConnection();  // Asegúrate de abrir la conexión
        if (con != null) {
            ps = con.prepareStatement(sql);
            ps.setString(1, alumno.getNombre());
            ps.setString(2, alumno.getApellido());
            ps.setString(3, alumno.getDni());
            ps.setString(4, alumno.getTelefono());
            ps.executeUpdate();
            System.out.println("Alumno agregado correctamente.");
        } else {
            System.err.println("Conexión no establecida. Verifica la configuración.");
            return 0; // Indica un fallo en la conexión
        }
    } catch (SQLException e) {
        System.err.println("Error al agregar alumno: " + e.getMessage());
        return 0; // Indica un error SQL
    }
    return 1; // Indica éxito
}
```

El método agregar() es fundamental para permitir la incorporación de nuevos alumnos en la base de datos. Garantiza que se manejen adecuadamente las conexiones y las excepciones, proporcionando una forma estructurada de agregar datos al sistema.

```java
// Método para actualizar un alumno
public int actualizar(Alumno alumno) {
    String sql = "UPDATE Alumno SET nombre=?, apellido=?, dni=?, telefono=? WHERE id=?";
    try {
        con = conectar.getConnection();
        ps = con.prepareStatement(sql);
        ps.setString(1, alumno.getNombre());
        ps.setString(2, alumno.getApellido());
        ps.setString(3, alumno.getDni());
        ps.setString(4, alumno.getTelefono());
        ps.setInt(5, alumno.getId());
        ps.executeUpdate();
    } catch (SQLException e) {
        System.err.println("Error al actualizar alumno: " + e.getMessage());
        return 0;
    }
    return 1;
}
```

El método actualizar() es esencial para modificar la información de un alumno en la base de datos. Asegura que se manejen adecuadamente las conexiones y excepciones, proporcionando una forma estructurada de actualizar datos en el sistema.

```java
// Método para eliminar un alumno
 public void eliminar(int id) {
     String sql = "DELETE FROM Alumno WHERE id = ?";
     try {
         con = conectar.getConnection(); // Obtener la conexión a la base de datos
         ps = con.prepareStatement(sql); // Preparar la consulta
         ps.setInt(1, id); // Establecer el ID del alumno en la consulta
         int filasAfectadas = ps.executeUpdate(); // Ejecutar la consulta

         if (filasAfectadas > 0) {
             System.out.println("Alumno con ID " + id + " eliminado correctamente.");
         } else {
             System.err.println("No se encontró ningún alumno con ID " + id + " para eliminar.");
         }
     } catch (SQLException e) {
         System.err.println("Error al eliminar alumno: " + e.getMessage()); // Manejar error
     }
 }
```

El método eliminar() es fundamental para permitir la eliminación de registros de alumnos en la base de datos. Maneja adecuadamente las conexiones y excepciones, y proporciona retroalimentación sobre el éxito o fracaso de la operación de eliminación.

- Los métodos de AlumnoDAO preparan las consultas y cargan los datos en memoria, pero la ejecución de las operaciones en la base de datos se realiza solo cuando se invoca explícitamente el método para ejecutar la consulta. Esto permite manejar los datos de manera controlada y eficiente.

Clase Controlador

```java
package CONTROLADOR;

import MODELO.Alumno;
import MODELO.AlumnoDAO;
import VISTA.Vista;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;

public class Controlador implements ActionListener {

    private Vista vista;
    private AlumnoDAO dao;
    private DefaultTableModel modelo;
    private int idAlumnoEditar = -1; // Variable para guardar el ID del alumno a editar

    public Controlador(Vista vista, AlumnoDAO dao) {
        this.vista = vista;
        this.dao = dao;
        this.modelo = (DefaultTableModel) this.vista.getTabla().getModel();

        // Asignar listeners a los botones
        this.vista.getBtnGuardar().addActionListener(this);
        this.vista.getBtnListar().addActionListener(this);
        this.vista.getBtnEliminar().addActionListener(this);
        this.vista.getBtnEditar().addActionListener(this);
        this.vista.getBtnOk().addActionListener(this);
    }
```

La clase Controlador implementa la interfaz ActionListener, lo que le permite manejar eventos de acción (como clics de botones) en la interfaz gráfica. Cuando un usuario interactúa con la interfaz (por ejemplo, al hacer clic en un botón), el controlador recibe el evento y puede ejecutar la lógica correspondiente, como agregar, listar, editar o eliminar alumnos.

```java
    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == vista.getBtnGuardar()) {
            agregar(); // Guardar y limpiar campos
        } else if (e.getSource() == vista.getBtnListar()) {
            listar(); // Mostrar todos los alumnos en la tabla
        } else if (e.getSource() == vista.getBtnEliminar()) {
            eliminar(); // Eliminar el alumno seleccionado
        } else if (e.getSource() == vista.getBtnEditar()) {
            cargarDatosEnCampos(); // Cargar datos en los campos para editar
        } else if (e.getSource() == vista.getBtnOk()) {
            actualizar(); // Actualizar los datos del alumno
        }
    }
```

Este método se encarga de gestionar las acciones que se producen en la interfaz gráfica cuando el usuario interactúa con los botones. Es parte de la implementación del patrón MVC. El método actionPerformed actúa como el controlador de eventos, dirigiendo las acciones del usuario a los métodos específicos que manejan la lógica del negocio. Esto permite que la interfaz de usuario y la lógica de la aplicación estén desacopladas, manteniendo así la claridad y la organización en el diseño del software.

```
private void agregar() {
    Alumno alumno = new Alumno();
    alumno.setNombre(vista.getTxtNombre().getText());
    alumno.setApellido(vista.getTxtApellido().getText());
    alumno.setDni(vista.getTxtDni().getText());
    alumno.setTelefono(vista.getTxtTelefono().getText());

    dao.agregar(alumno); // Guardar alumno en la base de datos
    limpiarCampos(); // Limpiar los campos después de guardar
    listar(); // Actualizar la lista para ver el nuevo alumno
}
```

Este método se encarga de crear un nuevo objeto Alumno, capturar los datos ingresados en la interfaz gráfica y guardarlos en la base de datos.

```
// Método para listar todos los alumnos en la tabla
private void listar() {
    modelo.setRowCount(0);  // Limpiar la tabla antes de listar
    List<Alumno> lista = dao.listar(); // Obtener todos los alumnos
    for (Alumno alumno : lista) {
        modelo.addRow(new Object[]{
            alumno.getId(),
            alumno.getNombre(),
            alumno.getApellido(),
            alumno.getDni(),
            alumno.getTelefono()
        });
    }
}
```

Este método se encarga de mostrar a todos los alumnos en la tabla de la interfaz gráfica.

```
private void eliminar() {
    int fila = vista.getTabla().getSelectedRow(); // Obtener la fila seleccionada
    if (fila == -1) {
        JOptionPane.showMessageDialog(vista, "Debe seleccionar un alumno para eliminar.");
    } else {
        int id = Integer.parseInt(vista.getTabla().getValueAt(fila, 0).toString()); // Obte
        System.out.println("ID a eliminar: " + id); // Agregar esta línea para ver el ID er
        dao.eliminar(id); // Llamar al método DAO para eliminar el alumno
        JOptionPane.showMessageDialog(vista, "Alumno eliminado correctamente.");
        listar(); // Refrescar la tabla para actualizar la lista
    }
}
```

Este método se encarga de eliminar un alumno seleccionado de la tabla.

```
private void cargarDatosEnCampos() {
    int fila = vista.getTabla().getSelectedRow();
    if (fila != -1) {
        vista.getTxtId().setText(vista.getTabla().getValueAt(fila, 0).toString()); // Carg
        vista.getTxtNombre().setText(vista.getTabla().getValueAt(fila, 1).toString());
        vista.getTxtApellido().setText(vista.getTabla().getValueAt(fila, 2).toString());
        vista.getTxtDni().setText(vista.getTabla().getValueAt(fila, 3).toString());
        vista.getTxtTelefono().setText(vista.getTabla().getValueAt(fila, 4).toString());

        // Guardamos el ID del alumno que vamos a editar
        idAlumnoEditar = Integer.parseInt(vista.getTxtId().getText());
    }
}
```

Este método carga los datos del alumno seleccionado en los campos de texto para su edición.

- Se obtiene el índice de la fila seleccionada.
- Se cargan los datos del alumno seleccionado en los campos de texto correspondientes.
- Se almacena el ID del alumno que se va a editar.

```
private void actualizar() {
    if (idAlumnoEditar != -1) {  // Verificamos que se seleccio
        Alumno alumno = new Alumno();
        alumno.setId(idAlumnoEditar); // Aseguramos que el ID a
        alumno.setNombre(vista.getTxtNombre().getText());
        alumno.setApellido(vista.getTxtApellido().getText());
        alumno.setDni(vista.getTxtDni().getText());
        alumno.setTelefono(vista.getTxtTelefono().getText());

        dao.actualizar(alumno);  // Actualizamos los datos del
        listar();  // Refrescamos la lista de alumnos para ver
        limpiarCampos();  // Limpiamos los campos después de ed
        idAlumnoEditar = -1;  // Reseteamos la variable del ID
    }
}
```

Este método actualiza los datos del alumno en la base de datos.

- Se verifica que se haya seleccionado un alumno para editar.
- Se crea un nuevo objeto Alumno y se asignan los valores de los campos de texto.
- Se llama al método actualizar del DAO para guardar los cambios.
- Se refresca la tabla y se limpian los campos de entrada.

```
    // Método para limpiar los campos de texto en la vista
    private void limpiarCampos() {
        vista.getTxtNombre().setText("");
        vista.getTxtApellido().setText("");
        vista.getTxtDni().setText("");
        vista.getTxtTelefono().setText("");
    }
}
```

Limpia los campos. Se establece el texto de cada campo en vacío para que el usuario pueda ingresar nuevos datos.

Codigo Completo

Clase Alumno

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package MODELO;

/**
 *
 * @author ADMIN
 */
public class Alumno {
    private int id;
    private String nombre;
    private String apellido;
    private String dni;
    private String telefono;

    // Constructor vacío
    public Alumno() {}

    // Constructor con parámetros
    public Alumno(int id, String nombre, String apellido, String dni, String telefono) {
        this.id = id;
        this.nombre = nombre;
        this.apellido = apellido;
        this.dni = dni;
        this.telefono = telefono;
    }

    // Getters y Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    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; }
```

```
    public String getDni() { return dni; }
    public void setDni(String dni) { this.dni = dni; }

    public String getTelefono() { return telefono; }
    public void setTelefono(String telefono) { this.telefono = telefono; }
}
```

Clase AlumnoDAO

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package MODELO;

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

/**
 *
 * @author ADMIN
 */
public class AlumnoDAO {
     Conexion conectar = new Conexion();
    Connection con;
    PreparedStatement ps;
    ResultSet rs;

    // Método para listar alumnos
    public List<Alumno> listar() {
    List<Alumno> datos = new ArrayList<>();  // Creas una lista para almacenar los datos
    String sql = "SELECT * FROM Alumno";
    try {
        con = conectar.getConnection();  // Abres la conexión
```

```
            ps = con.prepareStatement(sql);  // Preparas la consulta
            rs = ps.executeQuery();  // Ejecutas la consulta
            while (rs.next()) {
                Alumno alumno = new Alumno();  // Creas un objeto Alumno
                alumno.setId(rs.getInt(1));  // Asignas el valor de las columnas
                alumno.setNombre(rs.getString(2));
                alumno.setApellido(rs.getString(3));
                alumno.setDni(rs.getString(4));
                alumno.setTelefono(rs.getString(5));
                datos.add(alumno);  // Agregas el objeto a la lista
            }
        } catch (SQLException e) {
            System.err.println("Error al listar alumnos: " + e.getMessage());
        }
        return datos;  // Retornas la lista de alumnos
    }

        // Método para agregar un alumno (sin el ID, porque es autoincremental)
    public int agregar(Alumno alumno) {
        String sql = "INSERT INTO Alumno(nombre, apellido, dni, telefono) VALUES (?,
?, ?, ?)";
        try {
            con = conectar.getConnection();  // Asegúrate de abrir la conexión
            if (con != null) {
                ps = con.prepareStatement(sql);
                ps.setString(1, alumno.getNombre());
                ps.setString(2, alumno.getApellido());
                ps.setString(3, alumno.getDni());
                ps.setString(4, alumno.getTelefono());
                ps.executeUpdate();
                System.out.println("Alumno agregado correctamente.");
            } else {
                System.err.println("Conexión no establecida. Verifica la configuración.");
                return 0; // Indica un fallo en la conexión
            }
        } catch (SQLException e) {
            System.err.println("Error al agregar alumno: " + e.getMessage());
            return 0; // Indica un error SQL
        }
        return 1; // Indica éxito
    }


    // Método para actualizar un alumno
```

```
    public int actualizar(Alumno alumno) {
        String sql = "UPDATE Alumno SET nombre=?, apellido=?, dni=?, telefono=?
WHERE id=?";
        try {
            con = conectar.getConnection();
            ps = con.prepareStatement(sql);
            ps.setString(1, alumno.getNombre());
            ps.setString(2, alumno.getApellido());
            ps.setString(3, alumno.getDni());
            ps.setString(4, alumno.getTelefono());
            ps.setInt(5, alumno.getId());
            ps.executeUpdate();
        } catch (SQLException e) {
            System.err.println("Error al actualizar alumno: " + e.getMessage());
            return 0;
        }
        return 1;
    }


    // Método para eliminar un alumno
    public void eliminar(int id) {
        String sql = "DELETE FROM Alumno WHERE id = ?";
        try {
            con = conectar.getConnection(); // Obtener la conexión a la base de datos
            ps = con.prepareStatement(sql); // Preparar la consulta
            ps.setInt(1, id); // Establecer el ID del alumno en la consulta
            int filasAfectadas = ps.executeUpdate(); // Ejecutar la consulta

            if (filasAfectadas > 0) {
                System.out.println("Alumno con ID " + id + " eliminado correctamente.");
            } else {
                System.err.println("No se encontró ningún alumno con ID " + id + " para
eliminar.");
            }
        } catch (SQLException e) {
            System.err.println("Error al eliminar alumno: " + e.getMessage()); // Manejar
error
        }
    }
}
```

Clase Conexion

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package MODELO;

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

/**
 *
 * @author ADMIN
 */
public class Conexion {
    private static final String URL = "jdbc:sqlserver://localhost:1433;databaseName=EjemploCRUDDB;encrypt=false;trustServerCertificate=true;";
    private static final String USER = "sa";
    private static final String PASSWORD = "1234";
    private Connection con = null;

    public Connection getConnection() {
        try {
            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");                //Asegúrate de cargar el driver
            con = DriverManager.getConnection(URL, USER, PASSWORD);
            System.out.println("Conexión exitosa a la base de datos.");
        } catch (ClassNotFoundException | SQLException e) {
            System.err.println("Error de conexión: " + e.getMessage());
        }
        return con;
    }
}
```

Clase Controlador

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package CONTROLADOR;

import MODELO.Alumno;
import MODELO.AlumnoDAO;
import VISTA.Vista;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;

public class Controlador implements ActionListener {

    private Vista vista;
    private AlumnoDAO dao;
    private DefaultTableModel modelo;
    private int idAlumnoEditar = -1; // Variable para guardar el ID del alumno a editar

    public Controlador(Vista vista, AlumnoDAO dao) {
        this.vista = vista;
        this.dao = dao;
        this.modelo = (DefaultTableModel) this.vista.getTabla().getModel();

        // Asignar listeners a los botones
        this.vista.getBtnGuardar().addActionListener(this);
        this.vista.getBtnListar().addActionListener(this);
        this.vista.getBtnEliminar().addActionListener(this);
        this.vista.getBtnEditar().addActionListener(this);
        this.vista.getBtnOk().addActionListener(this);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == vista.getBtnGuardar()) {
            agregar(); // Guardar y limpiar campos
```

```
    } else if (e.getSource() == vista.getBtnListar()) {
        listar(); // Mostrar todos los alumnos en la tabla
    } else if (e.getSource() == vista.getBtnEliminar()) {
        eliminar(); // Eliminar el alumno seleccionado
    } else if (e.getSource() == vista.getBtnEditar()) {
        cargarDatosEnCampos(); // Cargar datos en los campos para editar
    } else if (e.getSource() == vista.getBtnOk()) {
        actualizar(); // Actualizar los datos del alumno
    }
}

private void agregar() {
    Alumno alumno = new Alumno();
    alumno.setNombre(vista.getTxtNombre().getText());
    alumno.setApellido(vista.getTxtApellido().getText());
    alumno.setDni(vista.getTxtDni().getText());
    alumno.setTelefono(vista.getTxtTelefono().getText());

    dao.agregar(alumno); // Guardar alumno en la base de datos
    limpiarCampos(); // Limpiar los campos después de guardar
    listar(); // Actualizar la lista para ver el nuevo alumno
}



// Método para listar todos los alumnos en la tabla
private void listar() {
    modelo.setRowCount(0);  // Limpiar la tabla antes de listar
    List<Alumno> lista = dao.listar(); // Obtener todos los alumnos
    for (Alumno alumno : lista) {
        modelo.addRow(new Object[]{
            alumno.getId(),
            alumno.getNombre(),
            alumno.getApellido(),
            alumno.getDni(),
            alumno.getTelefono()
        });
    }
}

private void eliminar() {
    int fila = vista.getTabla().getSelectedRow(); // Obtener la fila seleccionada
    if (fila == -1) {
```

```
            JOptionPane.showMessageDialog(vista, "Debe seleccionar un alumno para
eliminar.");
        } else {
            int id = Integer.parseInt(vista.getTabla().getValueAt(fila, 0).toString()); //
Obtener el ID del alumno de la tabla
            System.out.println("ID a eliminar: " + id); // Agregar esta línea para ver el ID en
consola
            dao.eliminar(id); // Llamar al método DAO para eliminar el alumno
            JOptionPane.showMessageDialog(vista, "Alumno eliminado correctamente.");
            listar(); // Refrescar la tabla para actualizar la lista
        }
    }

    private void cargarDatosEnCampos() {
        int fila = vista.getTabla().getSelectedRow();
        if (fila != -1) {
            vista.getTxtId().setText(vista.getTabla().getValueAt(fila, 0).toString()); //
Cargar el ID en txtId
            vista.getTxtNombre().setText(vista.getTabla().getValueAt(fila, 1).toString());
            vista.getTxtApellido().setText(vista.getTabla().getValueAt(fila, 2).toString());
            vista.getTxtDni().setText(vista.getTabla().getValueAt(fila, 3).toString());
            vista.getTxtTelefono().setText(vista.getTabla().getValueAt(fila, 4).toString());

            // Guardamos el ID del alumno que vamos a editar
            idAlumnoEditar = Integer.parseInt(vista.getTxtId().getText());
        }
    }



    private void actualizar() {
        if (idAlumnoEditar != -1) {  // Verificamos que se seleccionó un alumno
            Alumno alumno = new Alumno();
            alumno.setId(idAlumnoEditar); // Aseguramos que el ID a editar es el correcto
            alumno.setNombre(vista.getTxtNombre().getText());
            alumno.setApellido(vista.getTxtApellido().getText());
            alumno.setDni(vista.getTxtDni().getText());
            alumno.setTelefono(vista.getTxtTelefono().getText());

            dao.actualizar(alumno);  // Actualizamos los datos del alumno en la base de
datos
            listar();  // Refrescamos la lista de alumnos para ver los cambios
            limpiarCampos();  // Limpiamos los campos después de editar
            idAlumnoEditar = -1;  // Reseteamos la variable del ID a editar
```

```
        }
    }


    // Método para limpiar los campos de texto en la vista
    private void limpiarCampos() {
        vista.getTxtNombre().setText("");
        vista.getTxtApellido().setText("");
        vista.getTxtDni().setText("");
        vista.getTxtTelefono().setText("");
    }
}
```

Clase de Jframe Vista

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java to edit this template
 */
package VISTA;

import CONTROLADOR.Controlador;
import MODELO.AlumnoDAO;
import javax.swing.table.DefaultTableModel;

/**
 *
 * @author ADMIN
 */
public class Vista extends javax.swing.JFrame {

    /**
     * Creates new form Vista
     */
    private DefaultTableModel modelo;
    private AlumnoDAO gestor;
    private Controlador controlador;
```

```java
// Getters de los botones para el controlador
public javax.swing.JButton getBtnGuardar() {
    return btnGuardar;
}

public javax.swing.JButton getBtnEliminar() {
    return btnEliminar;
}

public javax.swing.JButton getBtnListar() {
    return btnListar;
}

public javax.swing.JButton getBtnEditar() {
    return btnEditar;
}

public javax.swing.JButton getBtnOk() {
    return btnOk;
}

// Añadir el getter del campo txtId
public javax.swing.JTextField getTxtId() {
    return txtId;
}

public javax.swing.JTextField getTxtNombre() {
    return txtNombre;
}

public javax.swing.JTextField getTxtApellido() {
    return txtApellido;
}

public javax.swing.JTextField getTxtDni() {
    return txtDni;
}

public javax.swing.JTextField getTxtTelefono() {
    return txtTelefono;
}

// Getter para la tabla
public javax.swing.JTable getTabla() {
```

```
        return jTable1;
    }

    public Vista() {
        initComponents();
        setLocationRelativeTo(null); // Centrar la ventana

        // Inicializar el modelo de la tabla
        modelo = new DefaultTableModel();
        modelo.addColumn("Id");
        modelo.addColumn("Nombre");
        modelo.addColumn("Apellido");
        modelo.addColumn("DNI");
        modelo.addColumn("Teléfono");
        jTable1.setModel(modelo); // Asignar el modelo a la tabla

        txtId = new javax.swing.JTextField();
        txtId.setEditable(false); // Evitar que se edite
    }
```

Clase Main

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 */

package com.mycompany.ejemplo.crud.mvc;

import CONTROLADOR.Controlador;
import MODELO.AlumnoDAO;
import MODELO.Conexion;
import VISTA.Vista;

/**
 *
 * @author ADMIN
 */
public class EJEMPLOCRUDMVC {

    public static void main(String[] args) {
        Vista vista = new Vista();
```

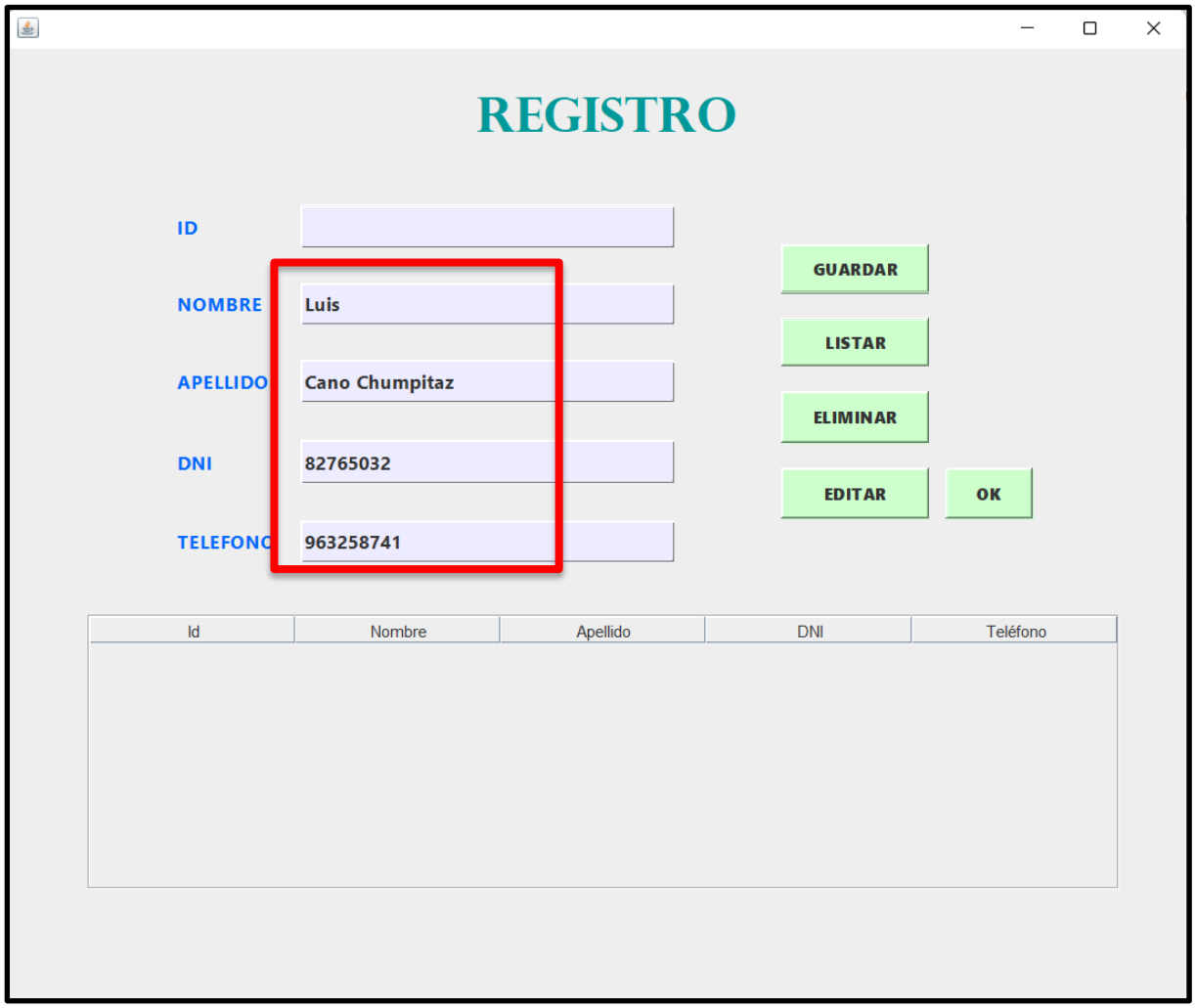
```
        // Instanciamos el modelo (DAO) para manejar los datos de los alumnos
        AlumnoDAO modelo = new AlumnoDAO();

        // Creamos el controlador, pasándole la vista y el modelo
        Controlador controlador = new Controlador(vista, modelo);

        // Hacemos visible la ventana principal (Vista)
        vista.setVisible(true);

        // Usamos la nueva instancia de Conexion
        Conexion conexion = new Conexion();
        conexion.getConnection(); // Establecer la conexión con la base de datos


    }
}
```

Base de datos

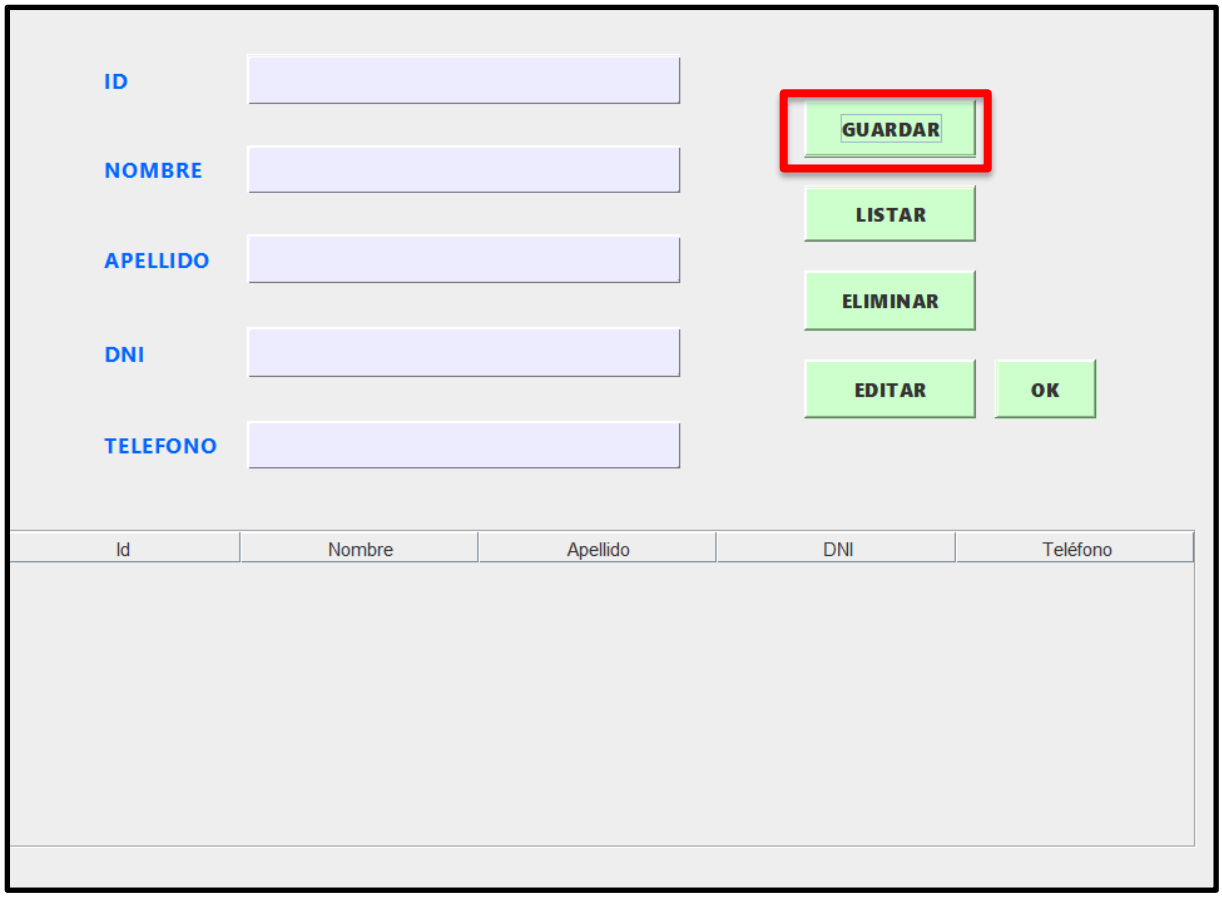
```
CREATE DATABASE EjemploCRUDDB;

USE EjemploCRUDDB;

CREATE TABLE Alumno (
    id INT PRIMARY KEY IDENTITY(1,1),
    nombre VARCHAR(100) NOT NULL,
    apellido VARCHAR(100) NOT NULL,
    dni VARCHAR(8) NOT NULL,
    telefono VARCHAR(15)
);

select * from Alumno;
```

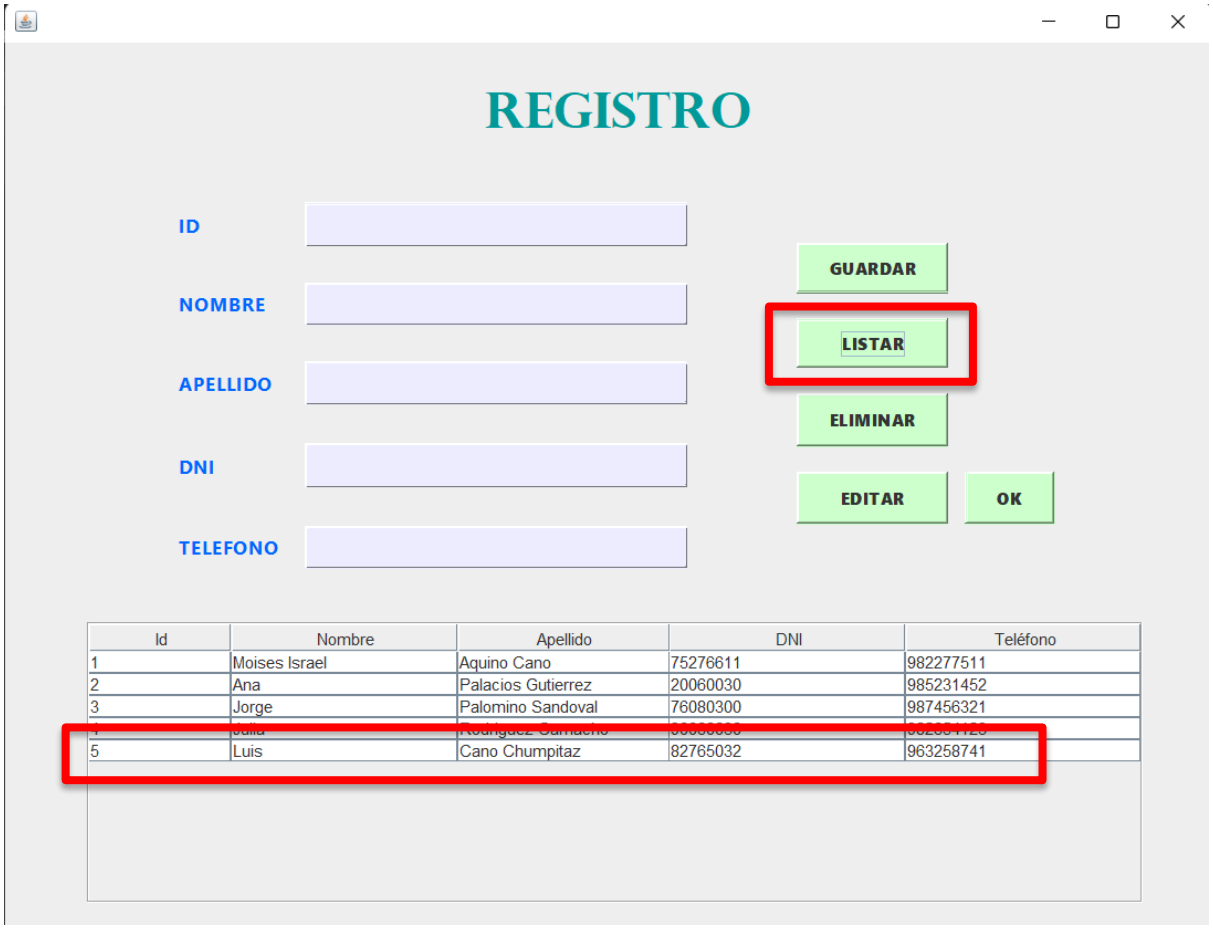
Ejecución del proyecto

Primero ingresamos los datos

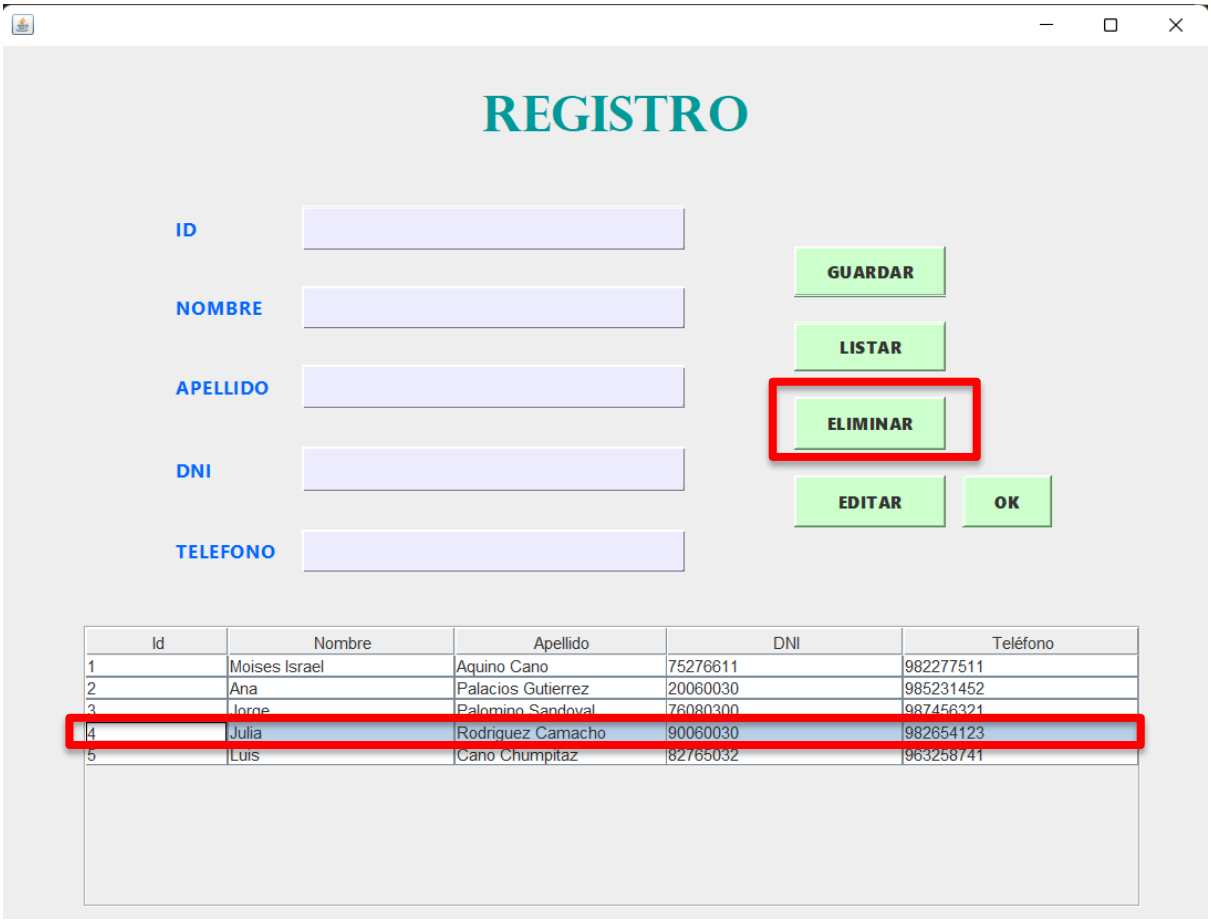


Después presionamos el botón guardar, se limpian los campos. Para poder ver los datos que fueron guardados presionaremos el botón Listar


ID
NOMBRE
APELLIDO
DNI
TELEFONO
GUARDAR
LISTAR
ELIMINAR
EDITAR
OK
Id
Nombre
Apellido
DNI
Teléfono



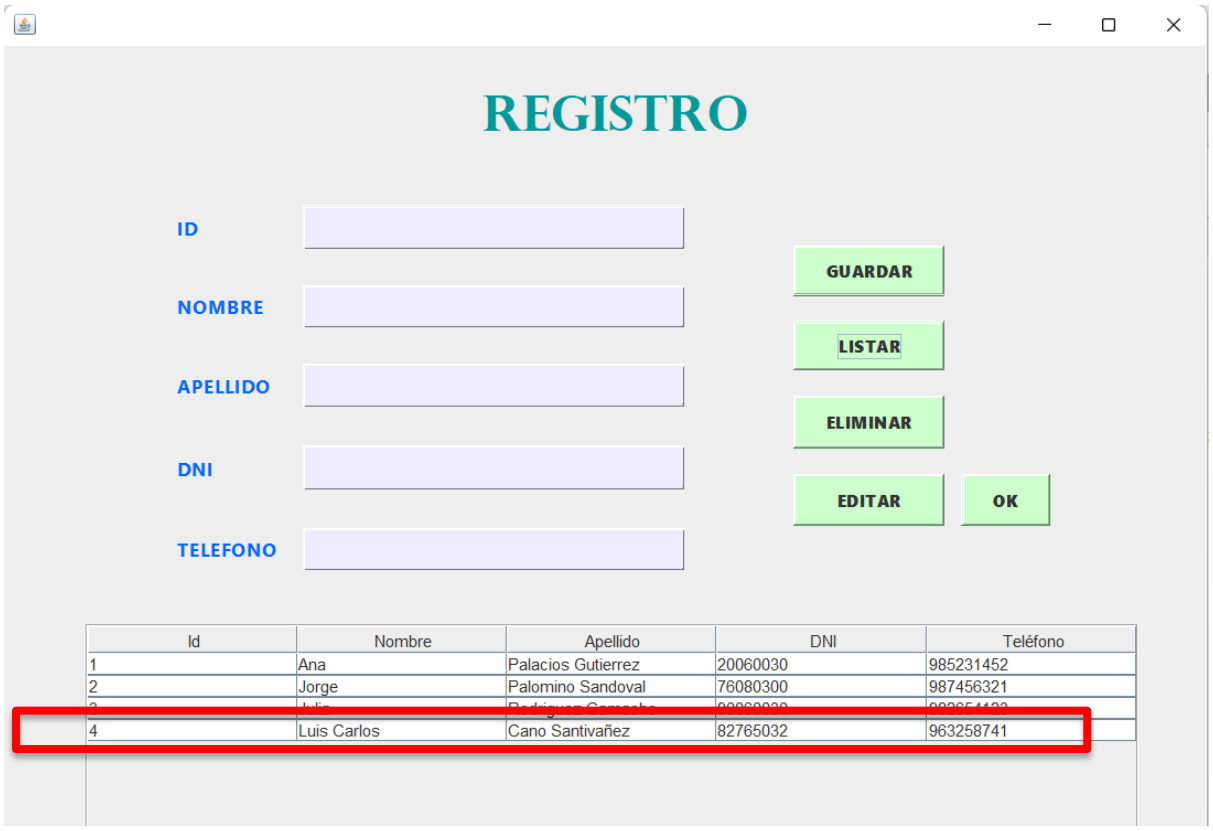
Para eliminar solo tendremos que seleccionar la fila y presionar el boton eliminar



Y se borro el alumno

Y para editar los datos del alumno se presiona el botón Editar, se sube los campos editas y con el botón OK se guarda la modificación.


REGISTRO
ID
NOMBRE
Luis Carlos
APELLIDO
Cano Santivañez
DNI
82765032
TELEFONO
963258741
GUARDAR
LISTAR
ELIMINAR
EDITAR
OK
Id
Nombre
Apellido
DNI
Teléfono
1
Ana
Palacios Gutierrez
20060030
985231452
2
Jorge
Palomino Sandoval
76080300
987456321
3
Julia
Rodriguez Camacho
90060030
982654123
4
Luis
Cano Chumpitaz
82765032
963258741


REGISTRO
ID
NOMBRE
APELLIDO
DNI
TELEFONO
GUARDAR
LISTAR
ELIMINAR
EDITAR
OK
Id
Nombre
Apellido
DNI
Teléfono
1
Ana
Palacios Gutierrez
20060030
985231452
2
Jorge
Palomino Sandoval
76080300
987456321
4
Luis Carlos
Cano Santivañez
82765032
963258741

Bien y revisamos en SQL los datos y si hay conexión con Java Netbeans.