Taller: Creación de un CRUD

Aplicando Arquitectura de 3 Capas y Patrón Modelo-Vista-Controlador (MVC)

Integrantes:
Dennison Chalacan
Jeffrey Manobanda
Jhordy Marcillo

Estructura del Proyecto:
Aplicación CRUD de Estudiante (ID, nombres, edad)

Tecnologías:
Java Swing, Arquitectura 3 Capas, Patrón MVC

Fecha: 20 de noviembre de 2025

Índice

1. Objetivo del Taller

Construir una aplicación CRUD de Estudiante (ID, nombres, edad) utilizando la arquitectura de 3 capas y el patrón de diseño Modelo-Vista-Controlador (MVC) con una interfaz gráfica en Java Swing.

2. Estructura del Proyecto

La aplicación deberá organizarse en la siguiente estructura de paquetes:

```
src/main/java/ec/edu/espe/
    - datos/
    - model/
    - Estudiante.java
    - repository/
    - EstudianteRepository.java
    - logica_negocio/
    - EstudianteService.java
    - presentacion/
    - EstudianteUI.java
    - Main.java
```

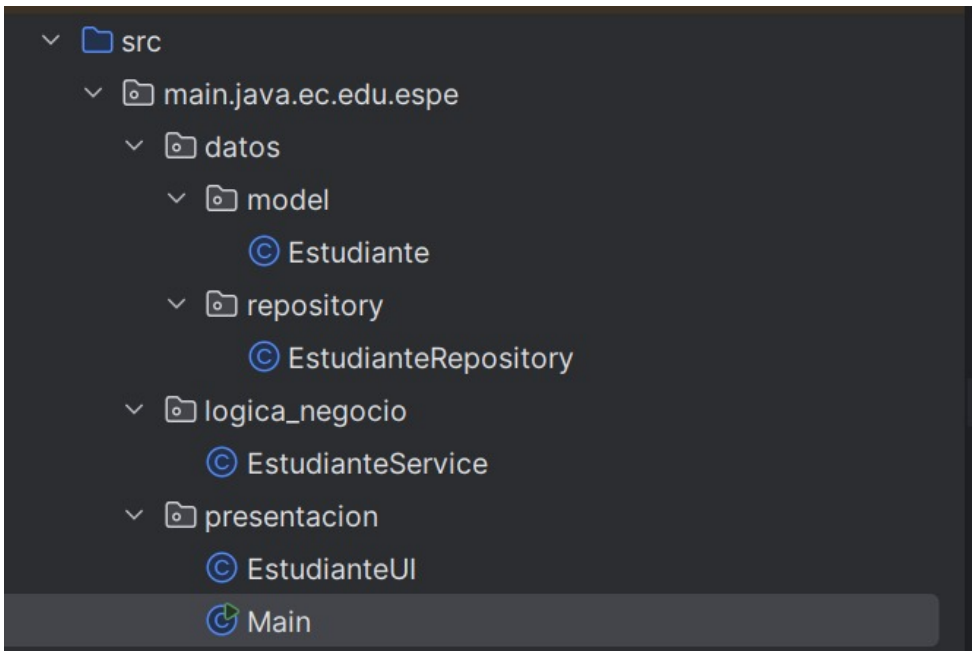


Figura 1: Estructura del proyecto

3. Descripción de las Capas

3.1. Capa Modelo (Model)

Contiene la clase `Estudiante.java`, que representa el dominio con sus atributos: ID, nombres y edad. No contiene lógica de negocio ni acceso a datos.

3.2. Capa de Acceso a Datos (Repository)

`EstudianteRepository.java` administra las operaciones CRUD utilizando una colección interna (ArrayList) o almacenamiento simple en archivo. Opcionalmente puede implementarse como Singleton.

3.3. Capa Lógica de Negocio (Service)

`EstudianteService.java` aplica reglas de negocio como validaciones (ej., edad ¿0, ID no repetido) y delega las operaciones CRUD al repositorio.

3.4. Capa Presentación – Swing (Vista y Controlador)

`EstudianteUI.java` representa la interfaz gráfica del usuario. Incluye:

- Campos de texto para ID, nombres y edad.
- Botones para las acciones CRUD.
- Una JTable que actúa como 'grid' para mostrar los estudiantes.

La UI envía las solicitudes al Service y actualiza la vista según resultados.

4. Flujo de Interacción MVC + 3 Capas

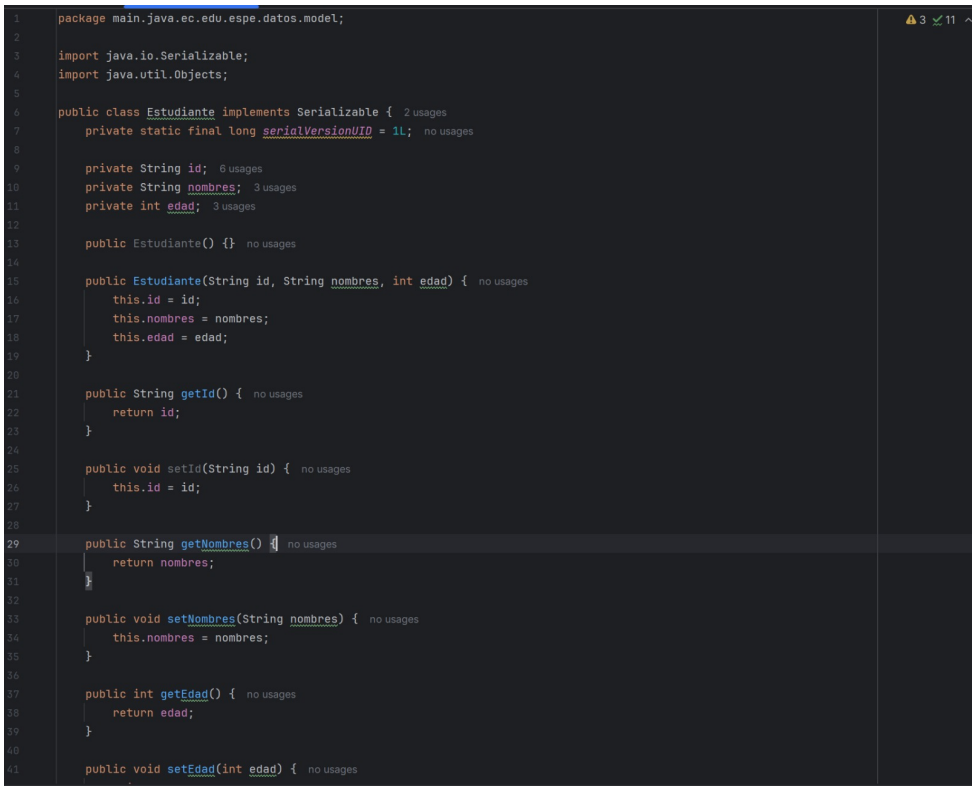
1. El usuario interactúa con `EstudianteUI`.
2. La UI llama a `EstudianteService`.
3. El Service valida datos y llama a `EstudianteRepository`.
4. El Repository ejecuta las operaciones CRUD y devuelve resultados.
5. La UI actualiza el formulario y el grid.

5. Actividades del Taller

5.1. Actividad 1: Crear el paquete datos/model

Crear la clase `Estudiante.java` con:

- Atributos: id, nombres, edad
- Constructor, getters y setters.



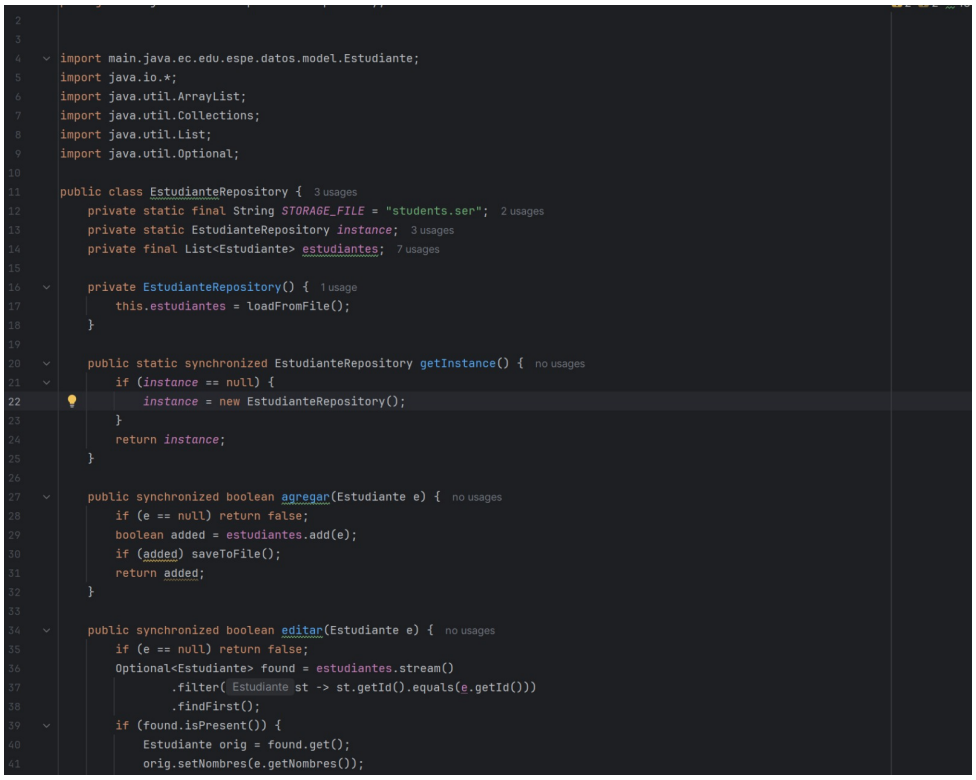
```java
package main.java.ec.edu.espe.datos.model;

import java.io.Serializable;
import java.util.Objects;

public class Estudiante implements Serializable {  2 usages
    private static final long serialVersionUID = 1L;  no usages

    private String id;  6 usages
    private String nombres;  3 usages
    private int edad;  3 usages

    public Estudiante() {}  no usages

    public Estudiante(String id, String nombres, int edad) {  no usages
        this.id = id;
        this.nombres = nombres;
        this.edad = edad;
    }

    public String getId() {  no usages
        return id;
    }

    public void setId(String id) {  no usages
        this.id = id;
    }

    public String getNombres() {  no usages
        return nombres;
    }

    public void setNombres(String nombres) {  no usages
        this.nombres = nombres;
    }

    public int getEdad() {  no usages
        return edad;
    }

    public void setEdad(int edad) {  no usages
```

Figura 2: Código de la clase Estudiante.java

5.2. Actividad 2: Implementar datos/repository

Implementar `EstudianteRepository.java`:

- Crear una lista interna de estudiantes.
- Implementar: `agregar()`, `editar()`, `eliminar()`, `listar()`.

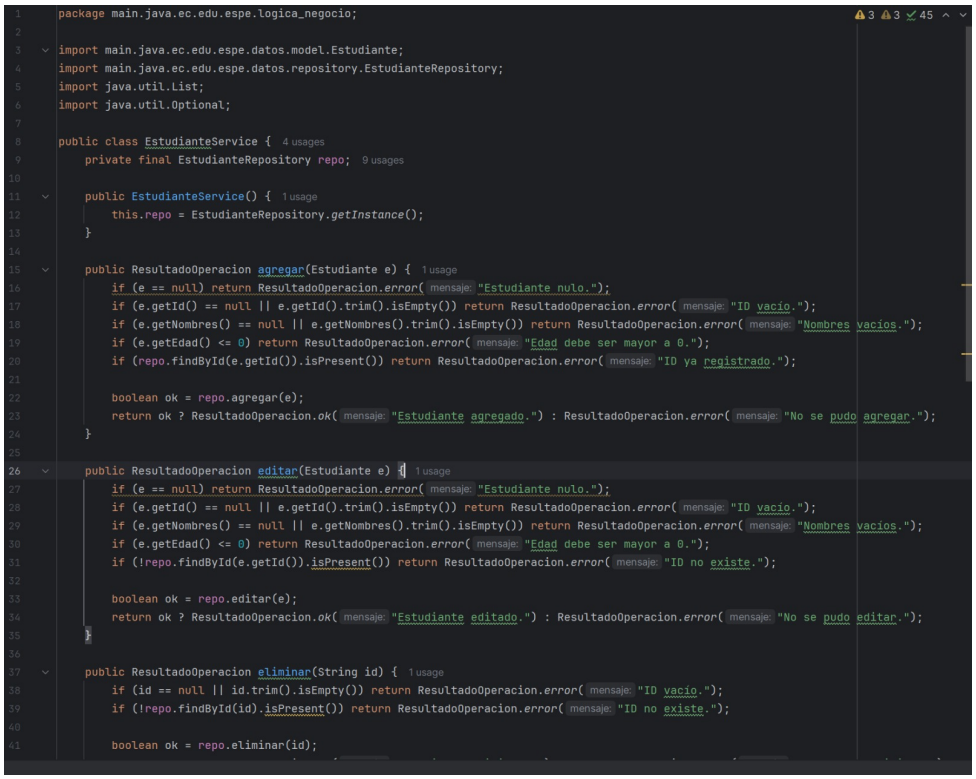


```java
import main.java.ec.edu.espe.datos.model.Estudiante;
import java.io.*;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Optional;

public class EstudianteRepository {  3 usages
    private static final String STORAGE_FILE = "students.ser";  2 usages
    private static EstudianteRepository instance;  3 usages
    private final List<Estudiante> estudiantes;  7 usages

    private EstudianteRepository() {  1 usage
        this.estudiantes = loadFromFile();
    }

    public static synchronized EstudianteRepository getInstance() {  no usages
        if (instance == null) {
            instance = new EstudianteRepository();
        }
        return instance;
    }

    public synchronized boolean agregar(Estudiante e) {  no usages
        if (e == null) return false;
        boolean added = estudiantes.add(e);
        if (added) saveToFile();
        return added;
    }

    public synchronized boolean editar(Estudiante e) {  no usages
        if (e == null) return false;
        Optional<Estudiante> found = estudiantes.stream()
                .filter( Estudiante st -> st.getId().equals(e.getId()))
                .findFirst();
        if (found.isPresent()) {
            Estudiante orig = found.get();
            orig.setNombres(e.getNombres());
```

Figura 3: Código del EstudianteRepository.java

5.3. Actividad 3: Crear logica_negocio/EstudianteService.java

- Validar reglas del negocio.
- Delegar todas las operaciones al Repository.

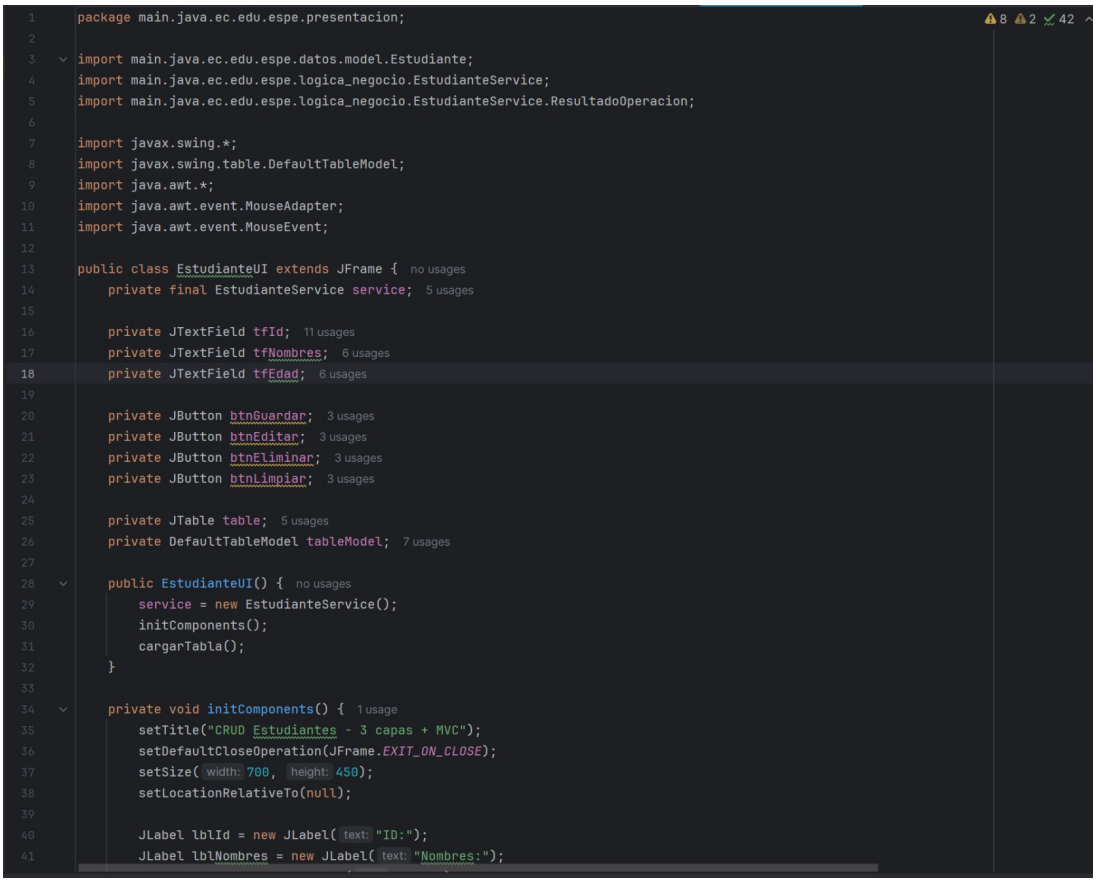


```java
package main.java.ec.edu.espe.logica_negocio;

import main.java.ec.edu.espe.datos.model.Estudiante;
import main.java.ec.edu.espe.datos.repository.EstudianteRepository;
import java.util.List;
import java.util.Optional;

public class EstudianteService {  4 usages
    private final EstudianteRepository repo;  9 usages

    public EstudianteService() {  1 usage
        this.repo = EstudianteRepository.getInstance();
    }

    public ResultadoOperacion agregar(Estudiante e) {  1 usage
        if (e == null) return ResultadoOperacion.error( mensaje: "Estudiante nulo.");
        if (e.getId() == null || e.getId().trim().isEmpty()) return ResultadoOperacion.error( mensaje: "ID vacío.");
        if (e.getNombres() == null || e.getNombres().trim().isEmpty()) return ResultadoOperacion.error( mensaje: "Nombres vacios.");
        if (e.getEdad() <= 0) return ResultadoOperacion.error( mensaje: "Edad debe ser mayor a 0.");
        if (repo.findById(e.getId()).isPresent()) return ResultadoOperacion.error( mensaje: "ID ya registrado.");

        boolean ok = repo.agregar(e);
        return ok ? ResultadoOperacion.ok( mensaje: "Estudiante agregado.") : ResultadoOperacion.error( mensaje: "No se pudo agregar.");
    }

    public ResultadoOperacion editar(Estudiante e) {  1 usage
        if (e == null) return ResultadoOperacion.error( mensaje: "Estudiante nulo.");
        if (e.getId() == null || e.getId().trim().isEmpty()) return ResultadoOperacion.error( mensaje: "ID vacío.");
        if (e.getNombres() == null || e.getNombres().trim().isEmpty()) return ResultadoOperacion.error( mensaje: "Nombres vacios.");
        if (e.getEdad() <= 0) return ResultadoOperacion.error( mensaje: "Edad debe ser mayor a 0.");
        if (!repo.findById(e.getId()).isPresent()) return ResultadoOperacion.error( mensaje: "ID no existe.");

        boolean ok = repo.editar(e);
        return ok ? ResultadoOperacion.ok( mensaje: "Estudiante editado.") : ResultadoOperacion.error( mensaje: "No se pudo editar.");
    }

    public ResultadoOperacion eliminar(String id) {  1 usage
        if (id == null || id.trim().isEmpty()) return ResultadoOperacion.error( mensaje: "ID vacío.");
        if (!repo.findById(id).isPresent()) return ResultadoOperacion.error( mensaje: "ID no existe.");

        boolean ok = repo.eliminar(id);
```

Figura 4: Código del EstudianteService.java

5.4. Actividad 4: Crear presentacion/EstudianteUI.java en Swing

- Construir el formulario (ID, nombres, edad).
- Implementar botón Guardar → llama al Service.
- Actualizar la JTable con los datos.



```java
package main.java.ec.edu.espe.presentacion;

import main.java.ec.edu.espe.datos.model.Estudiante;
import main.java.ec.edu.espe.logica_negocio.EstudianteService;
import main.java.ec.edu.espe.logica_negocio.EstudianteService.ResultadoOperacion;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

public class EstudianteUI extends JFrame {
    private final EstudianteService service;

    private JTextField tfId;
    private JTextField tfNombres;
    private JTextField tfEdad;

    private JButton btnGuardar;
    private JButton btnEditar;
    private JButton btnEliminar;
    private JButton btnLimpiar;

    private JTable table;
    private DefaultTableModel tableModel;

    public EstudianteUI() {
        service = new EstudianteService();
        initComponents();
        cargarTabla();
    }

    private void initComponents() {
        setTitle("CRUD Estudiantes - 3 capas + MVC");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize( width: 700,  height: 450);
        setLocationRelativeTo(null);

        JLabel lblId = new JLabel( text: "ID:");
        JLabel lblNombres = new JLabel( text: "Nombres:");
```

Figura 5: Interfaz gráfica EstudianteUI.java

6. Interfaz Gráfica del CRUD

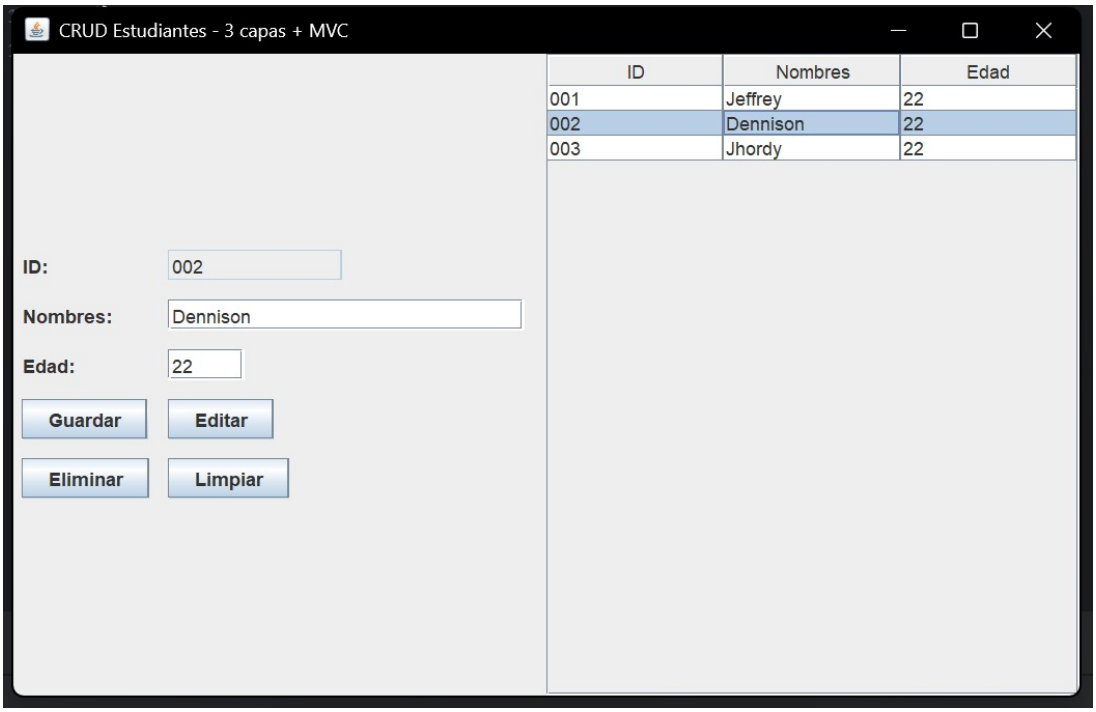


Figura 6: Interfaz del CRUD

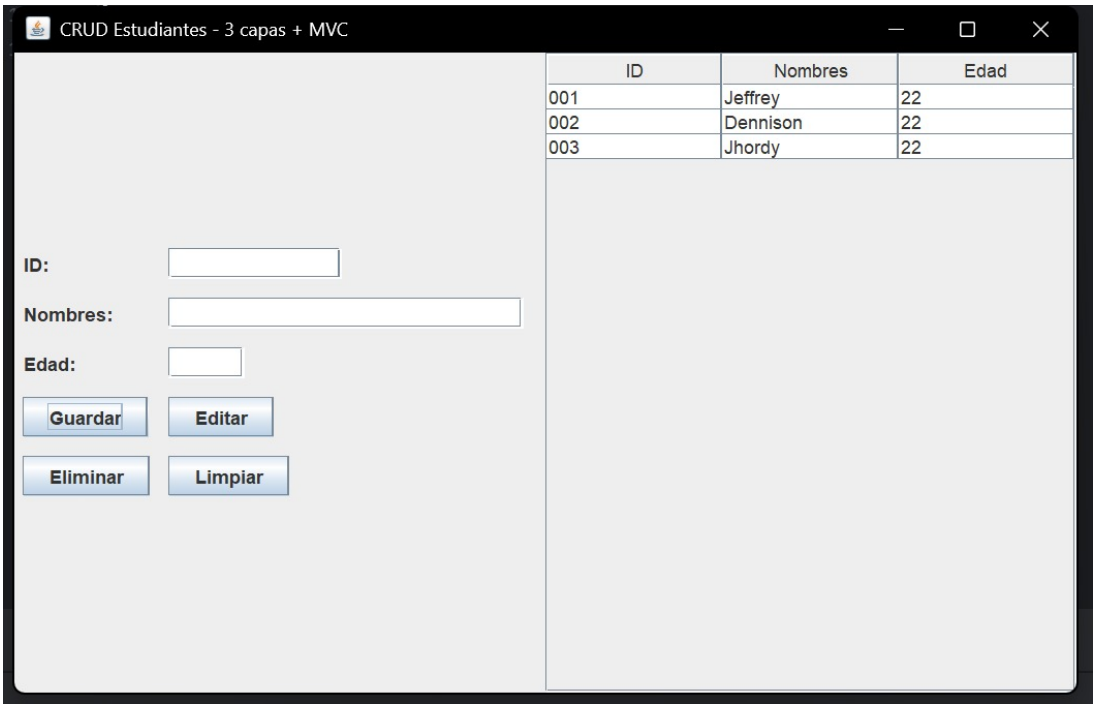


Figura 7: Interfaz del CRUD