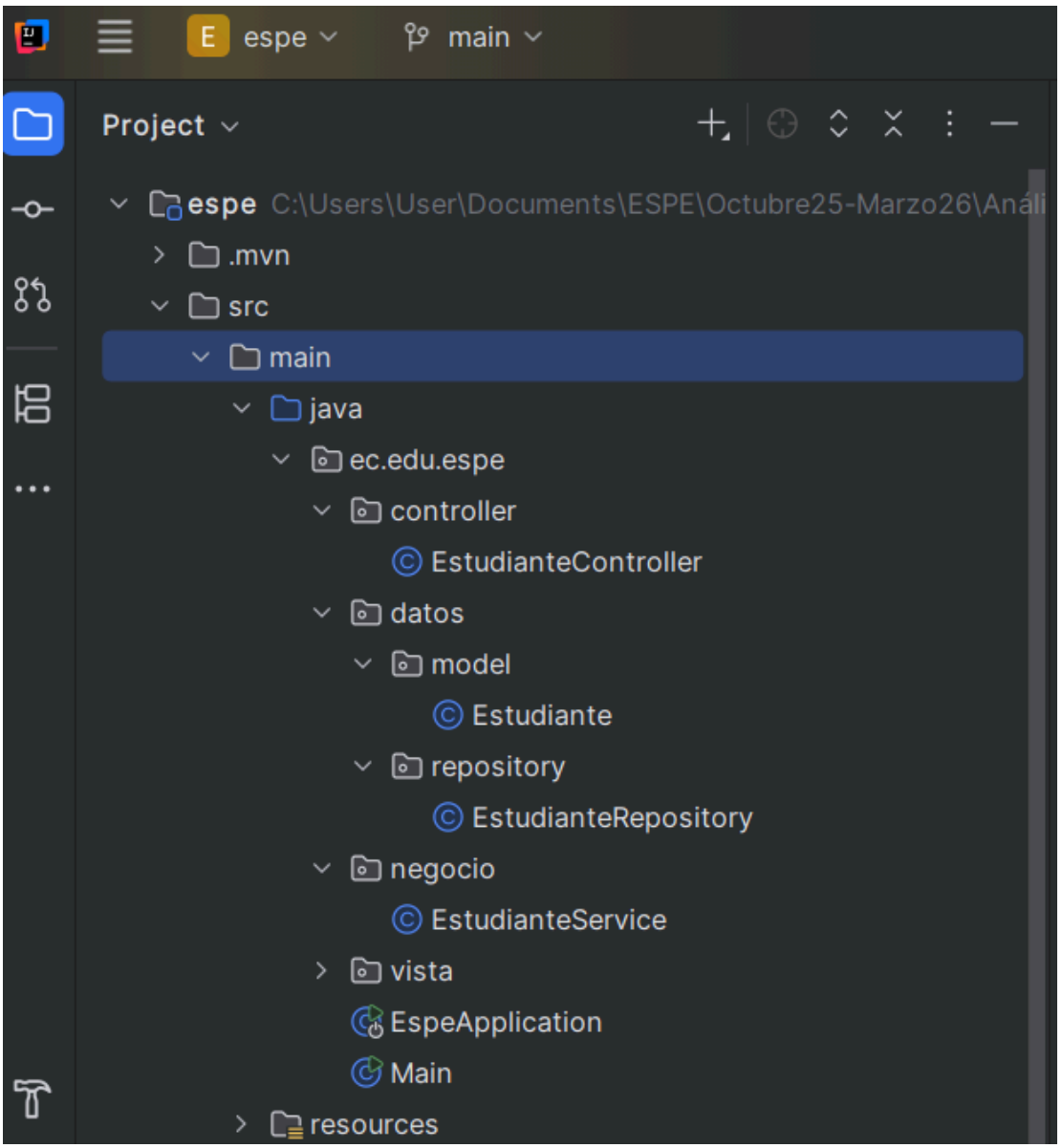
Taller 1

Nombre del estudiante:	Matias Lugmaña, Camilo Orrico, Denise Rea, Julio Viche
Docente:	Mgt. Jenny Alexandra Ruiz Robalino
Fecha:	25/11/2025
NRC:	27835

1. Objetivo del Taller

Comprender y aplicar la arquitectura de 3 capas (Modelo, Repositorio y Servicio), junto con el patron NVC para desarrollar un-CRUD de Estudiante (ID, Nombres, Edad), basado en el documento de Arquitectura con GEMA.

2. Código Fuente Organizado por Paquetes



El código fuente se organizó según la arquitectura de 3 capas, lo que permitió separar las responsabilidades de la siguiente manera:

1. Paquete ec.edu.espe.**datos** (Capa de Datos):

    - **Modelo**: La clase Estudiante contiene los atributos id, nombres y edad, junto con sus respectivos métodos getters y setters.
        i. Codigo:

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


public class Estudiante {

   private String id;

   private String nombres;

   private int edad;


   public Estudiante(String id, String nombres, int edad) {

       this.id = id;

       this.nombres = nombres;

       this.edad = edad;

   }


   // Getters y Setters [cite: 40]

   public String getId() { return id; }

   public void setId(String id) { this.id = id; }


   public String getNombres() { return nombres; }

   public void setNombres(String nombres) { this.nombres =
nombres; }
```

```
    public int getEdad() { return edad; }

    public void setEdad(int edad) { this.edad = edad; }

}
```

- **Repositorio**: La clase EstudianteRepository gestiona la persistencia de los datos, permitiendo operaciones de agregar, listar, buscar, actualizar y eliminar estudiantes. Se implementó el patrón Singleton para asegurar que solo exista una instancia del repositorio a lo largo de la aplicación.
    i. Codigo

```
package ec.edu.espe.datos.repository;


import ec.edu.espe.datos.model.Estudiante;

import java.io.*;

import java.util.ArrayList;

import java.util.List;


public class EstudianteRepository {

    // 1. Instancia estática única

    private static EstudianteRepository instance;

    private List<Estudiante> estudiantes;

    private final String FILE_NAME = "estudiantes.txt";


    // 2. Constructor privado (nadie puede hacer 'new' desde fuera)

    private EstudianteRepository() {

        this.estudiantes = new ArrayList<>();

        cargarDesdeArchivo();
```

```
}

// 3. Método de acceso global (Singleton)
public static EstudianteRepository getInstance() {
    if (instance == null) {
        instance = new EstudianteRepository();
    }
    return instance;
}

// --- MÉTODOS CRUD ---
public void crear(Estudiante estudiante) {
    estudiantes.add(estudiante);
    guardarEnArchivo();
}

public List<Estudiante> listar() {
    return estudiantes;
}

public Estudiante buscarPorId(String id) {
    return estudiantes.stream()
            .filter(e -> e.getId().equals(id))
            .findFirst().orElse(null);
}
```

```
    public void actualizar(Estudiante estActualizado) {
        for (int i = 0; i < estudiantes.size(); i++) {
            if (estudiantes.get(i).getId().equals(estActualizado.getId())) {
                estudiantes.set(i, estActualizado);
                break;
            }
        }
        guardarEnArchivo();
    }

    public void eliminar(String id) {
        estudiantes.removeIf(e -> e.getId().equals(id));
        guardarEnArchivo();
    }

    // --- PERSISTENCIA TXT ---
    private void guardarEnArchivo() {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(FILE_NAME))) {
            for (Estudiante est : estudiantes) {
                writer.write(est.getId() + "," + est.getNombres() + "," + est.getEdad());
                writer.newLine();
            }
        } catch (IOException e) { e.printStackTrace(); }
    }
```

```
   private void cargarDesdeArchivo() {
       File file = new File(FILE_NAME);
       if (!file.exists()) return;
       try (BufferedReader reader = new BufferedReader(new
FileReader(file))) {
           String line;
           while ((line = reader.readLine()) != null) {
               String[] parts = line.split(",");
               if (parts.length == 3) {
                   estudiantes.add(new Estudiante(parts[0],
parts[1], Integer.parseInt(parts[2])));
               }
           }
       } catch (IOException e) { e.printStackTrace(); }
   }
}
```

2. Paquete ec.edu.espe.**negocio** (Capa de Negocio):
    - **Servicio**: La clase EstudianteService se encarga de la lógica de negocio, que incluye la validación de datos y la delegación de operaciones CRUD al repositorio. Aquí también se manejan las excepciones relacionadas con la manipulación de estudiantes.
        i. Codigo

```
package ec.edu.espe.negocio;


import ec.edu.espe.datos.model.Estudiante;
import ec.edu.espe.datos.repository.EstudianteRepository;
```

```
import java.util.List;



public class EstudianteService {

   private EstudianteRepository repository;



   public EstudianteService() {

       // USO DEL SINGLETON

       this.repository = EstudianteRepository.getInstance();

   }



   public void guardar(String id, String nombre, String edadStr)
throws Exception {

       validar(id, nombre, edadStr);

       if (repository.buscarPorId(id) != null) throw new
Exception("Cédula repetida");

       repository.crear(new Estudiante(id, nombre,
Integer.parseInt(edadStr)));

   }



   public void editar(String id, String nombre, String edadStr)
throws Exception {

       validar(id, nombre, edadStr);

       if (repository.buscarPorId(id) == null) throw new
Exception("Estudiante no existe");

       repository.actualizar(new Estudiante(id, nombre,
Integer.parseInt(edadStr)));

   }



   public void eliminar(String id) throws Exception {
```

```
        if (repository.buscarPorId(id) == null) throw new
Exception("No existe");

        repository.eliminar(id);

    }



    public Estudiante buscar(String id) {

        return repository.buscarPorId(id);

    }



    public List<Estudiante> listar() {

        return repository.listar();

    }



    private void validar(String id, String nom, String edad) throws
Exception {

        if (id.isEmpty() || nom.isEmpty() || edad.isEmpty()) throw
new Exception("Campos vacíos");

        if (!id.matches("\\d{10}")) throw new Exception("Cédula
inválida"); // Validación simple

    }

}
```

3. Paquete ec.edu.espe.**controller** (Capa de Control):

    - **Controlador**: La clase EstudianteController maneja los eventos generados por la vista. Delegó las operaciones de CRUD al servicio y actualizó la vista con los resultados. Los métodos actionPerformed manejan las acciones de los botones, como guardar, editar, eliminar y buscar estudiantes.
        i. Codigo

```
package ec.edu.espe.controller;
```

```
import ec.edu.espe.datos.model.Estudiante;

import ec.edu.espe.negocio.EstudianteService;

import ec.edu.espe.vista.EstudianteUI;

import javax.swing.*;

import java.awt.event.*;


public class EstudianteController implements ActionListener {

   private EstudianteUI view;

   private EstudianteService service;


   public EstudianteController(EstudianteUI view,
EstudianteService service) {

       this.view = view;

       this.service = service;

       asignarEventos();

       refrescarTabla();

       this.view.setVisible(true);

   }


   private void asignarEventos() {

       view.btnGuardar.addActionListener(this);

       view.btnEditar.addActionListener(this);

       view.btnEliminar.addActionListener(this);

       view.btnBuscar.addActionListener(this);

       view.btnLimpiar.addActionListener(this);
```

```
        view.tblEstudiantes.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                llenarCampos();
            }
        });
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        try {
            if (e.getSource() == view.btnGuardar) {
                service.guardar(view.txtId.getText(),
view.txtNombres.getText(), view.txtEdad.getText());
                limpiar();
            } else if (e.getSource() == view.btnEditar) {
                service.editar(view.txtId.getText(),
view.txtNombres.getText(), view.txtEdad.getText());
                limpiar();
            } else if (e.getSource() == view.btnEliminar) {
                service.eliminar(view.txtId.getText());
                limpiar();
            } else if (e.getSource() == view.btnBuscar) {
                Estudiante est =
service.buscar(view.txtId.getText());
                if (est != null) {
                    view.txtNombres.setText(est.getNombres());
```

```
view.txtEdad.setText(String.valueOf(est.getEdad()));

                } else {

                    JOptionPane.showMessageDialog(view, "No
encontrado");

                }

                return;

            } else if (e.getSource() == view.btnLimpiar) {

                limpiar();

                // ELIMINAMOS EL 'return;' PARA QUE BAJE A
REFRESCAR TABLA

                // O llamamos explícitamente:

                refrescarTabla();

                return;

            }



            refrescarTabla(); // Se ejecuta para Guardar, Editar,
Eliminar

            JOptionPane.showMessageDialog(view, "Acción realizada
con éxito");



        } catch (Exception ex) {

            JOptionPane.showMessageDialog(view, "Error: " +
ex.getMessage());

        }

    }



    private void refrescarTabla() {

        view.tableModel.setRowCount(0);
```

```
        for (Estudiante est : service.listar()) {

            view.tableModel.addRow(new Object[]{est.getId(),
est.getNombres(), est.getEdad()});

        }

    }



    private void llenarCampos() {

        int row = view.tblEstudiantes.getSelectedRow();

        if (row >= 0) {

            view.txtId.setText(view.tableModel.getValueAt(row,
0).toString());

            view.txtNombres.setText(view.tableModel.getValueAt(row,
1).toString());

            view.txtEdad.setText(view.tableModel.getValueAt(row,
2).toString());

            view.txtId.setEditable(false);

        }

    }



    private void limpiar() {

        view.txtId.setText(""); view.txtNombres.setText("");
view.txtEdad.setText("");

        view.txtId.setEditable(true);

        view.tblEstudiantes.clearSelection();

    }

}
```

4. Paquete ec.edu.espe.**vista** (Capa de Vista):

- **Interfaz de Usuario**: La clase EstudianteUI provee la interfaz de usuario que permite interactuar con el sistema CRUD. En este caso, se desarrolló una interfaz simple en consola. Se utilizan métodos para gestionar los datos ingresados y mostrar los resultados de las operaciones.
    i. Codigo

```
package ec.edu.espe.vista;



import javax.swing.*;

import javax.swing.table.DefaultTableModel;

import java.awt.*;

import java.awt.event.ActionListener;



public class EstudianteUI extends JFrame {

   // Componentes públicos para acceso del Controlador

   public JTextField txtId, txtNombres, txtEdad;

   public JButton btnGuardar, btnEditar, btnEliminar, btnLimpiar,
btnBuscar;

   public JTable tblEstudiantes;

   public DefaultTableModel tableModel;



   public EstudianteUI() {

       initComponents();

   }



   private void initComponents() {

       setTitle("Taller Singleton + NVC");

       setSize(600, 500);
```

```
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        setLayout(new BorderLayout());



        JPanel pForm = new JPanel(new GridLayout(5, 2));

        pForm.add(new JLabel("Cédula:")); txtId = new JTextField();
pForm.add(txtId);

        pForm.add(new JLabel("Nombres:")); txtNombres = new
JTextField(); pForm.add(txtNombres);

        pForm.add(new JLabel("Edad:")); txtEdad = new JTextField();
pForm.add(txtEdad);



        btnGuardar = new JButton("Guardar");

        btnEditar = new JButton("Editar");

        btnEliminar = new JButton("Eliminar");

        btnBuscar = new JButton("Buscar");

        btnLimpiar = new JButton("Limpiar");



        JPanel pBotones = new JPanel();

        pBotones.add(btnBuscar); pBotones.add(btnGuardar);

        pBotones.add(btnEditar); pBotones.add(btnEliminar);
pBotones.add(btnLimpiar);



        add(pForm, BorderLayout.NORTH);

        add(pBotones, BorderLayout.SOUTH);



        tableModel = new DefaultTableModel(new Object[]{"ID",
"Nombres", "Edad"}, 0);

        tblEstudiantes = new JTable(tableModel);
```

```
        add(new JScrollPane(tblEstudiantes), BorderLayout.CENTER);

    }

}
```

5. **Main**: La clase Main ejecuta la aplicación, instanciando los objetos necesarios para la ejecución de la vista, el servicio y el controlador, y haciendo visible la interfaz.

Codigo:

```
package ec.edu.espe;


import ec.edu.espe.controller.EstudianteController;

import ec.edu.espe.negocio.EstudianteService;

import ec.edu.espe.vista.EstudianteUI;


public class Main {

   public static void main(String[] args) {


        EstudianteUI view1 = new EstudianteUI();

        EstudianteService service1 = new EstudianteService();

        EstudianteController controller1 = new
EstudianteController(view1, service1);

        view1.setVisible(true);

        view1.setTitle("Ventana A (Usuario 1)");



        EstudianteUI view2 = new EstudianteUI();

        EstudianteService service2 = new EstudianteService();
```

```
        EstudianteController controller2 = new
EstudianteController(view2, service2);

        view2.setVisible(true);

        view2.setTitle("Ventana B (Usuario 2) - Verificación
Singleton");



        view2.setLocation(view1.getX() + 420, view1.getY());

    }

}
```

3. Explicación de la Arquitectura

La arquitectura de 3 capas, combinada con el patrón MVC+Singleton, sigue la filosofía de separación de responsabilidades, lo que mejora la claridad y mantenibilidad del código:

- **Modelo (Model):** En esta capa, se definen las entidades que representan los datos de la aplicación. En este caso, la clase Estudiante es el modelo de datos, con los atributos id, nombres, y edad. El modelo no contiene ninguna lógica de negocio, solo la estructura de los datos.

- **Repositorio (Repository):** Esta capa maneja la persistencia de los datos. La clase EstudianteRepository almacena y recupera los estudiantes desde un archivo, proporcionando métodos para agregar, listar, actualizar y eliminar registros. El patrón Singleton se aplica aquí para garantizar que toda la aplicación utilice la misma instancia del repositorio.

- **Servicio (Service):** La capa de servicio gestiona la lógica de negocio. La clase EstudianteService valida los datos antes de enviarlos al repositorio. También maneja las excepciones y verifica que los estudiantes no existan antes de agregar nuevos o que estén presentes antes de realizar actualizaciones o eliminaciones.

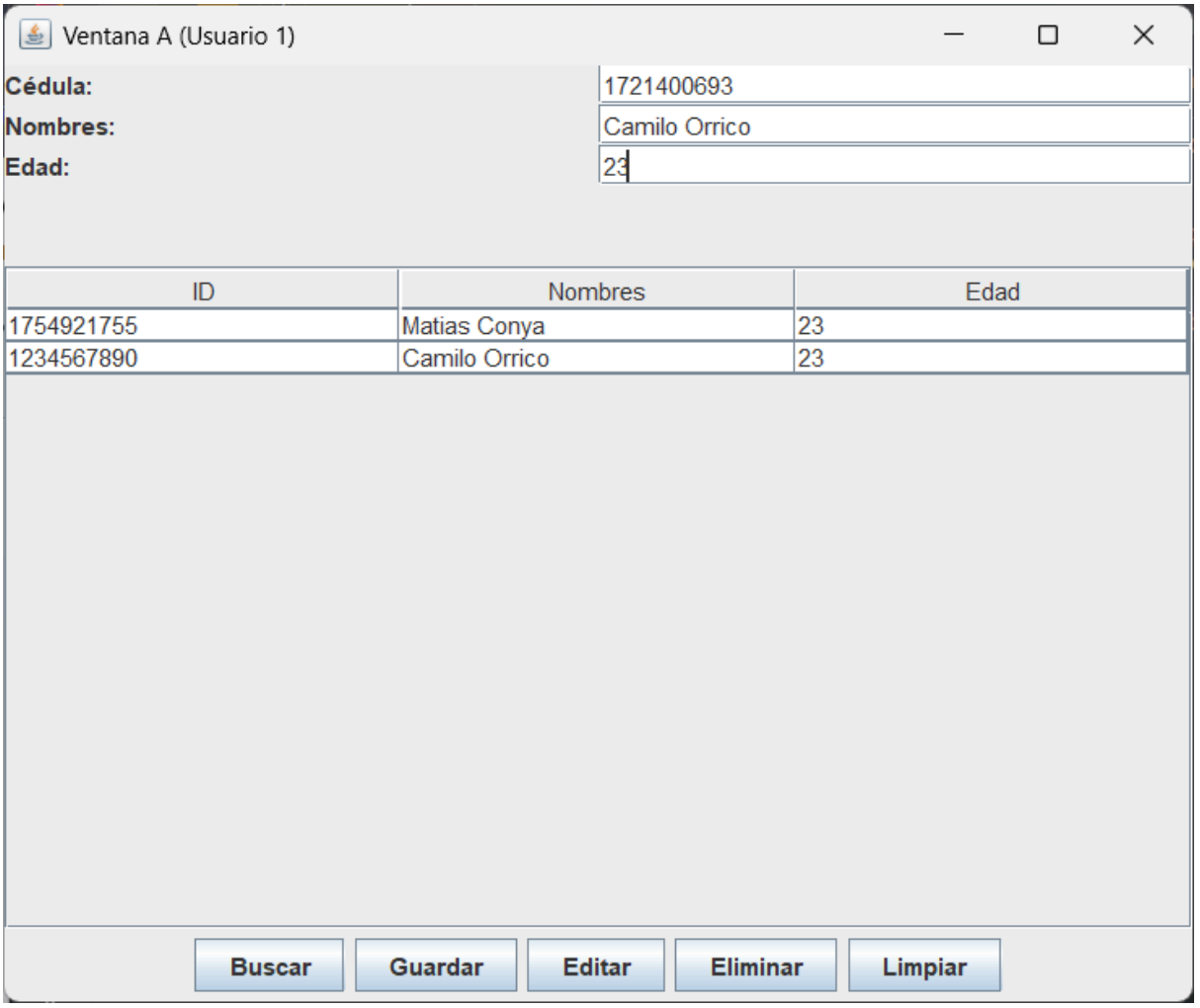
- **Vista (View):** La vista es responsable de interactuar con el usuario, solicitando entradas y mostrando los resultados. En este caso, se utilizó una interfaz de consola simple (EstudianteUI) que muestra menús y recibe entradas del usuario para realizar las operaciones de CRUD.

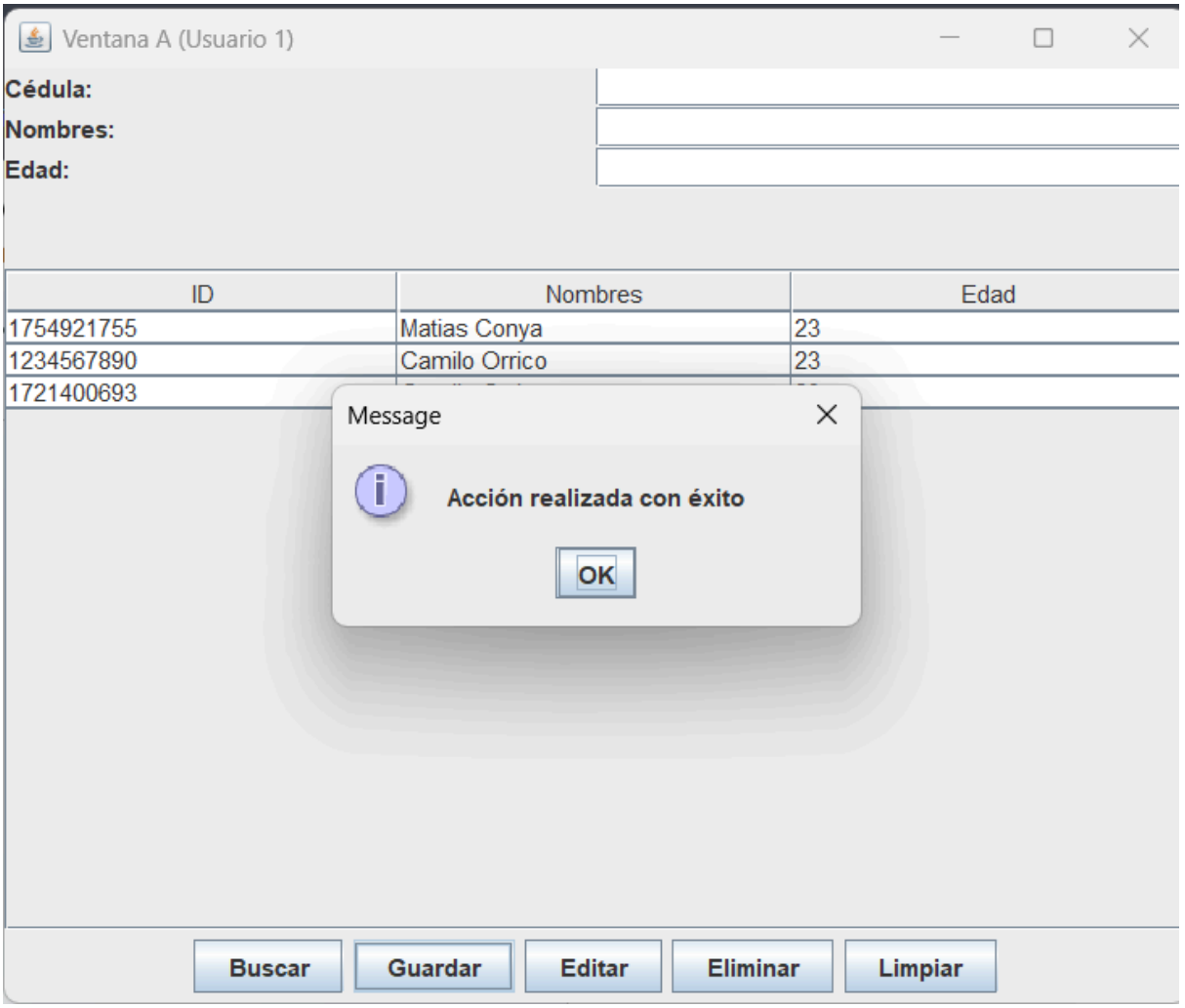
- **Controlador (Controller):** El controlador gestiona los eventos de la vista y coordina las interacciones entre el modelo y la vista. La clase EstudianteController es responsable de delegar las solicitudes de la vista a la capa de negocio y actualizar la interfaz de usuario en consecuencia.

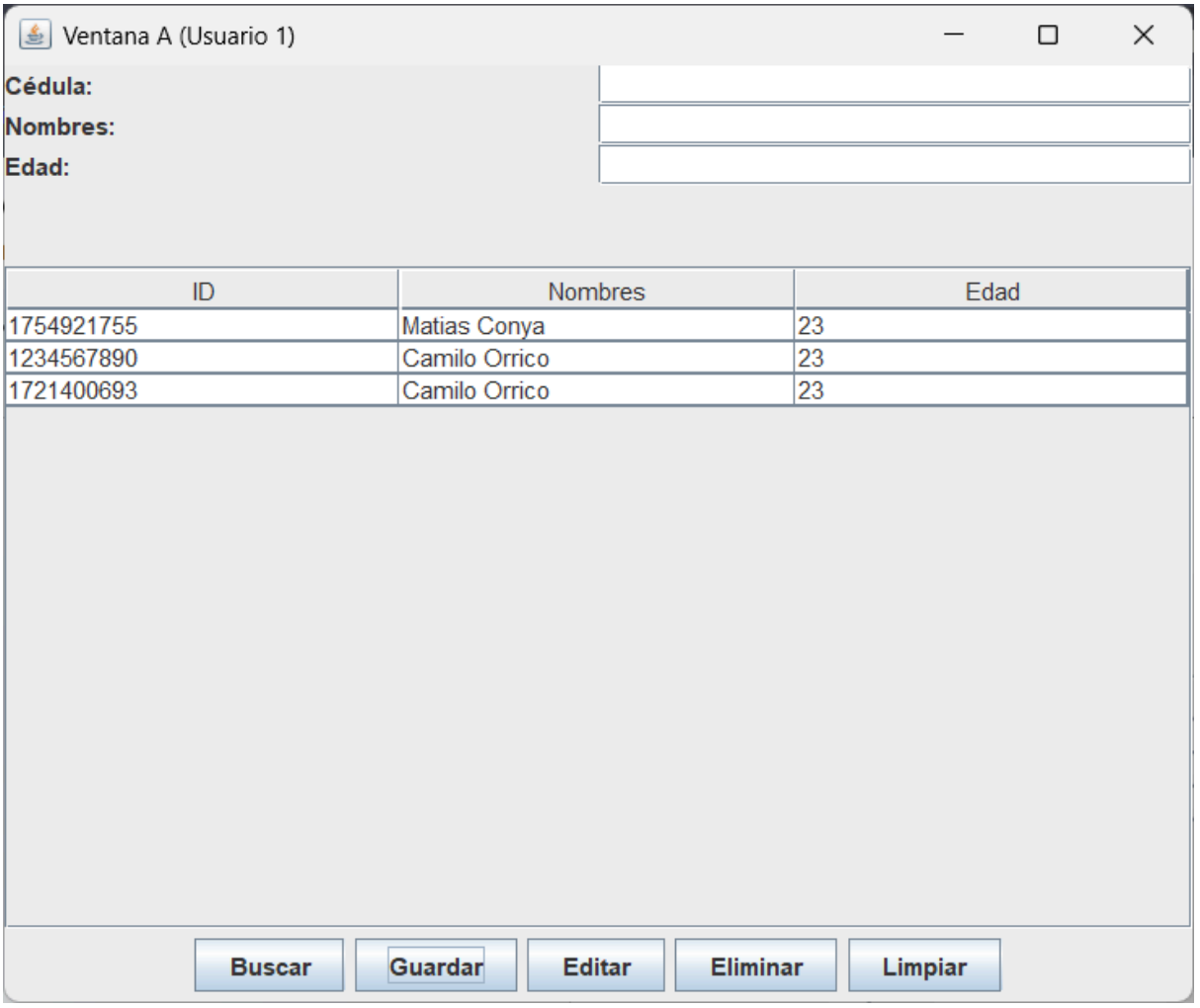
4. Evidencia de Ejecución del CRUD

A continuación, se muestran las capturas de pantalla que demuestran la ejecución de las operaciones del CRUD:

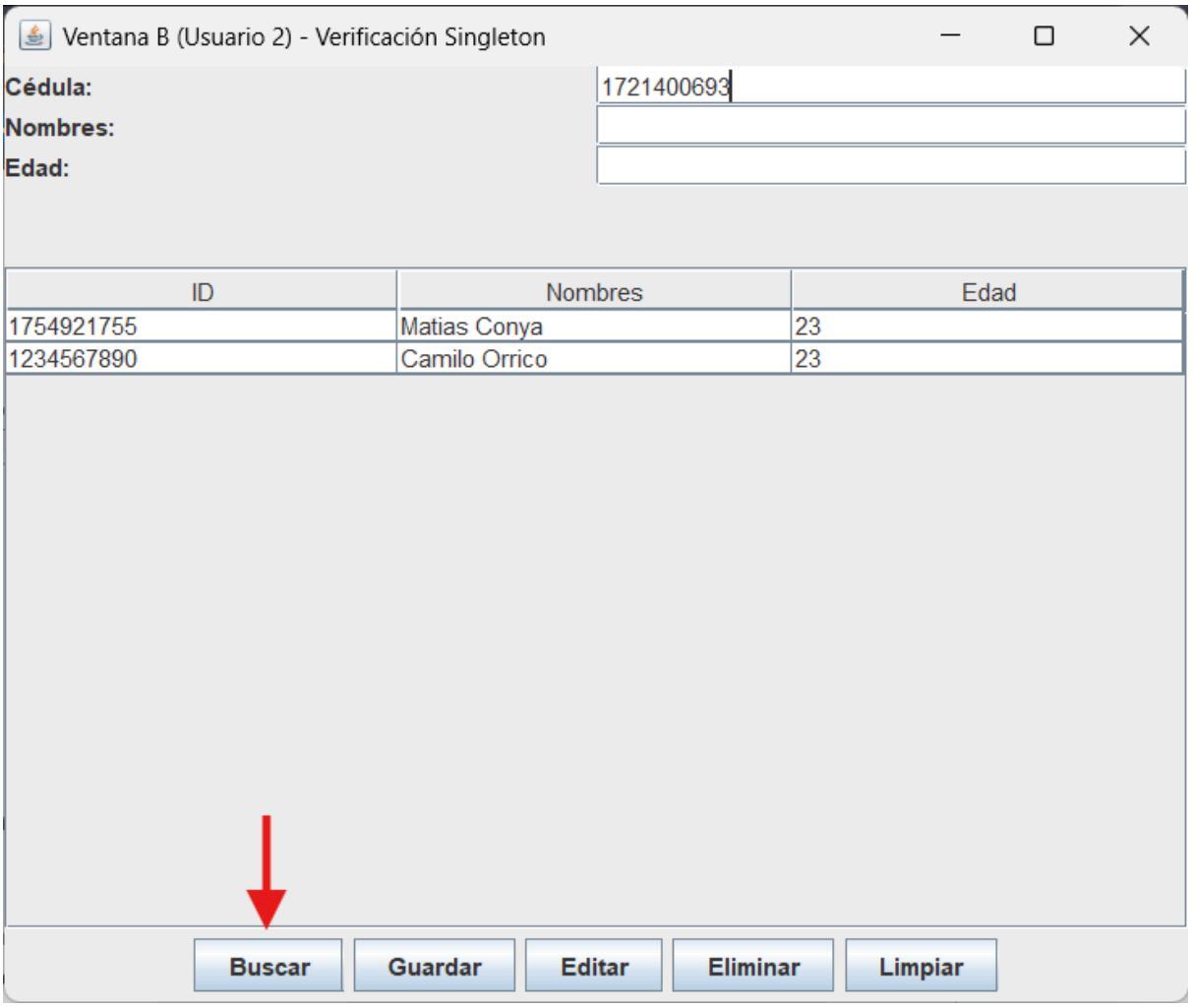
1. **Crear Estudiante:**

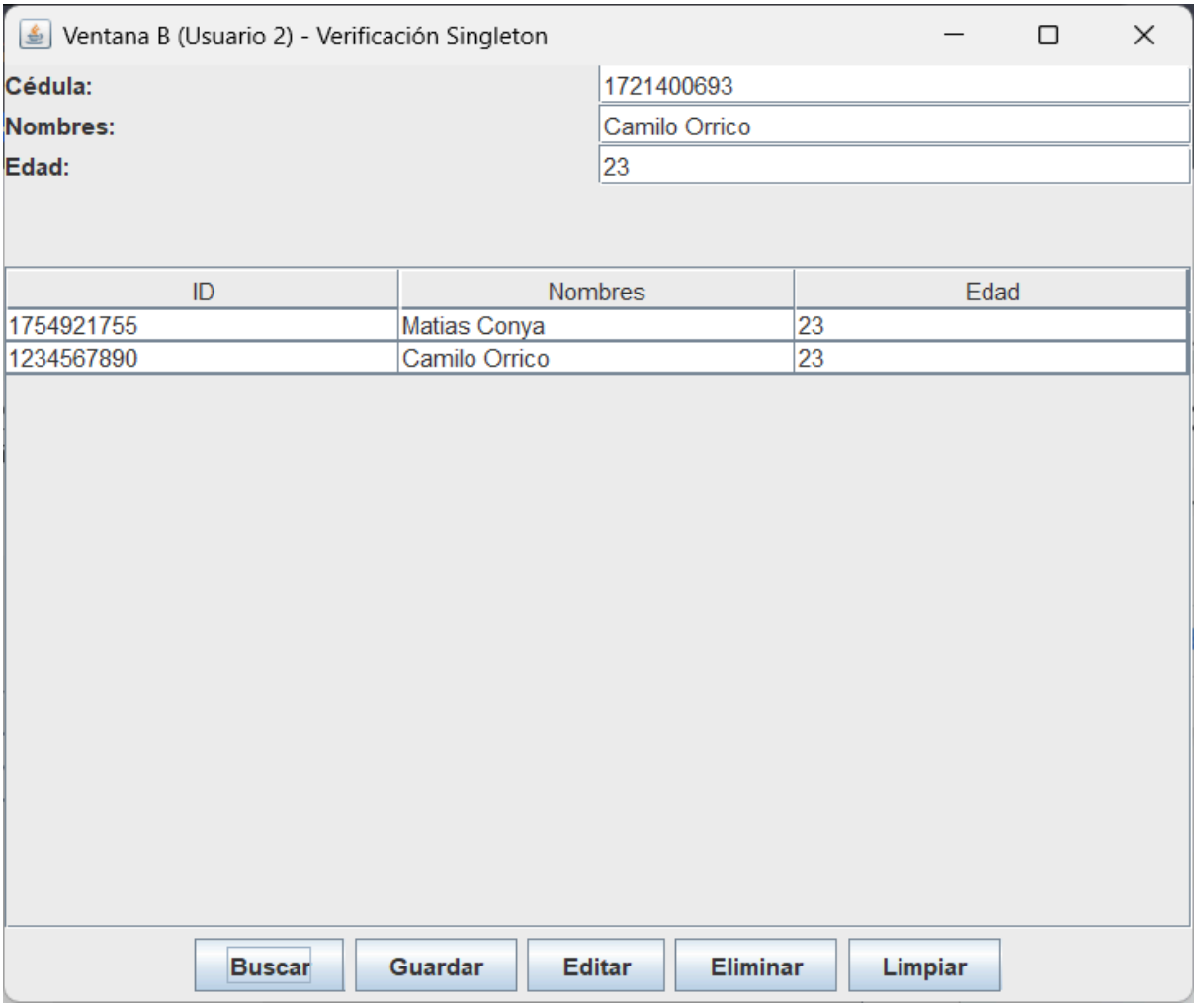

Ventana A (Usuario 1)
Cédula:
1721400693
Nombres:
Camilo Orrico
Edad:
23
ID
Nombres
Edad
1754921755
Matias Conya
23
1234567890
Camilo Orrico
23
Buscar
Guardar
Editar
Eliminar
Limpiar


Ventana A (Usuario 1)
Cédula:
Nombres:
Edad:
ID
Nombres
Edad
1754921755
Matias Conya
23
1234567890
Camilo Orrico
23
1721400693
Message
Acción realizada con éxito
OK
Buscar
Guardar
Editar
Eliminar
Limpiar

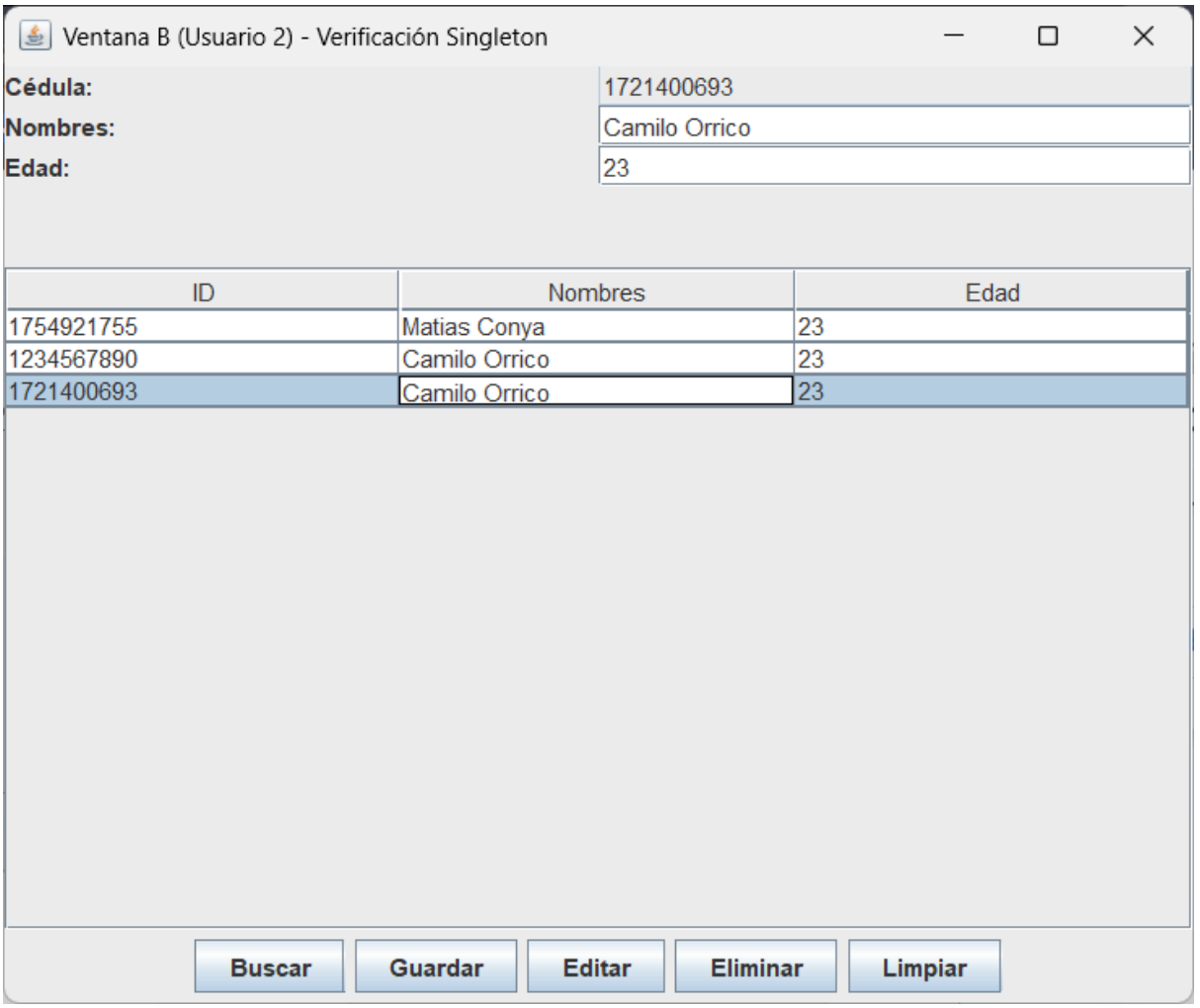

Ventana A (Usuario 1)
Cédula:
Nombres:
Edad:
ID
Nombres
Edad
1754921755
Matias Conya
23
1234567890
Camilo Orrico
23
1721400693
Camilo Orrico
23
Buscar
Guardar
Editar
Eliminar
Limpiar

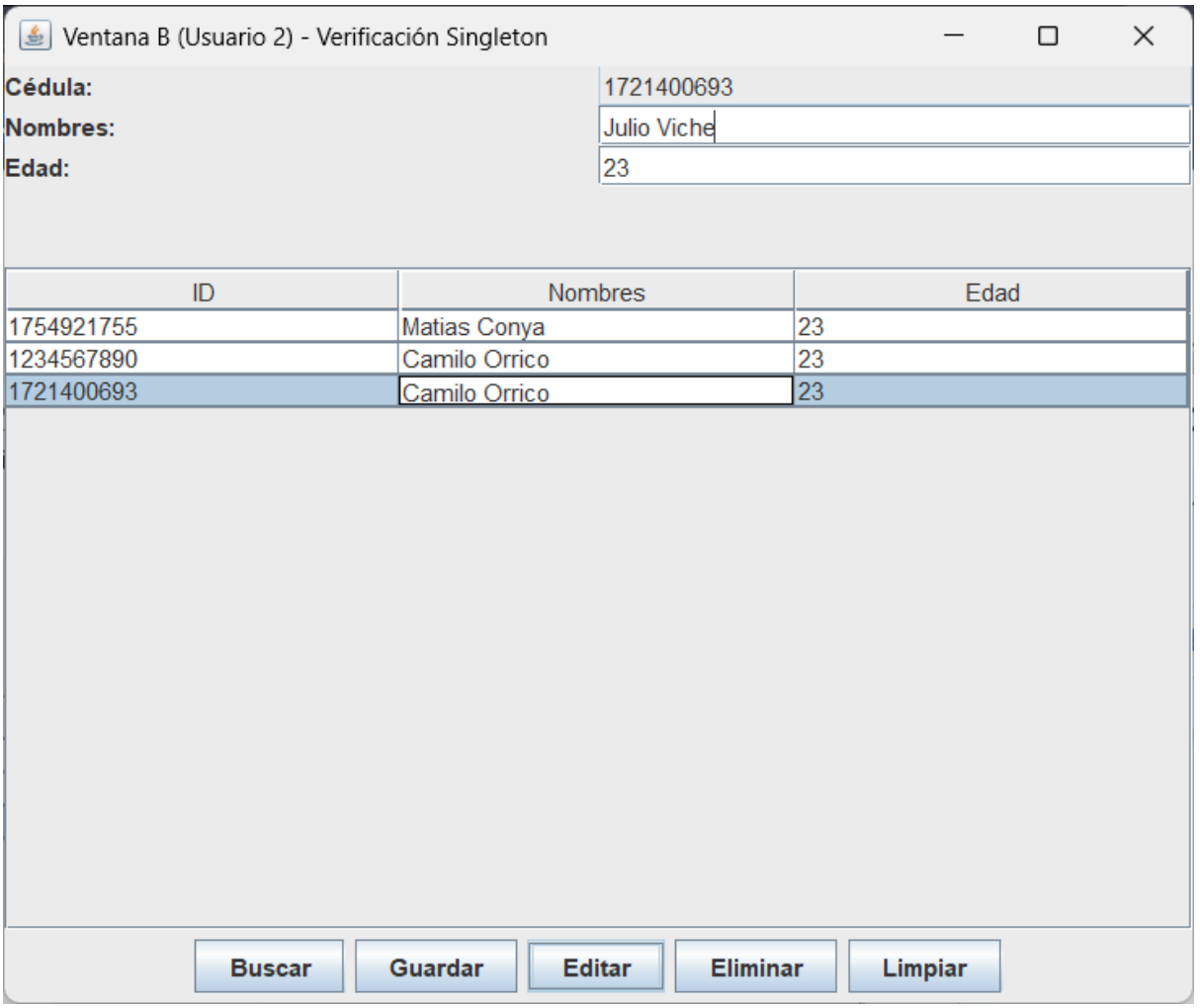
2. **Leer Estudiante:**

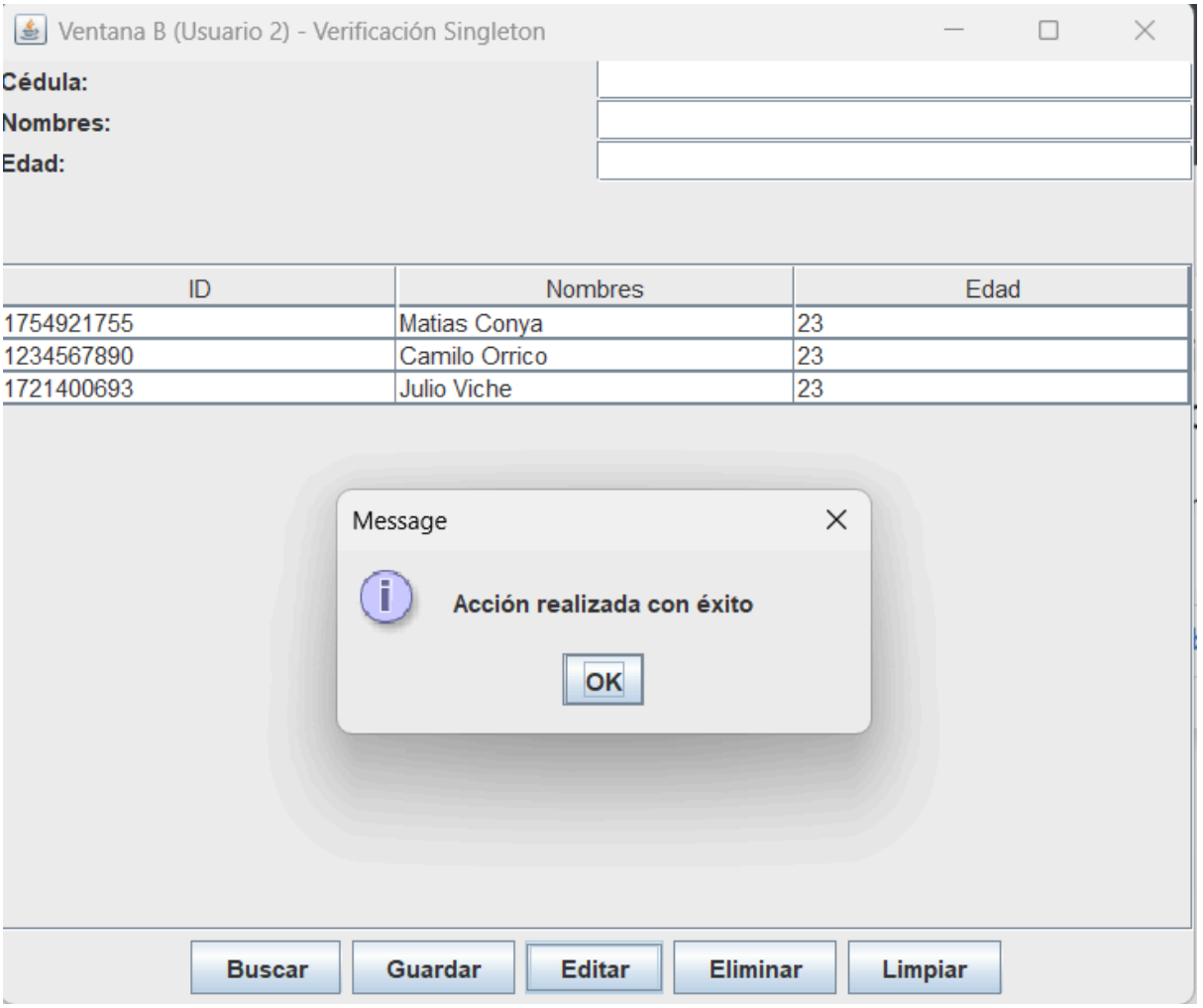



Ventana B (Usuario 2) - Verificación Singleton
Cédula:
1721400693
Nombres:
Camilo Orrico
Edad:
23
ID
Nombres
Edad
1754921755
Matias Conya
23
1234567890
Camilo Orrico
23
Buscar
Guardar
Editar
Eliminar
Limpiar

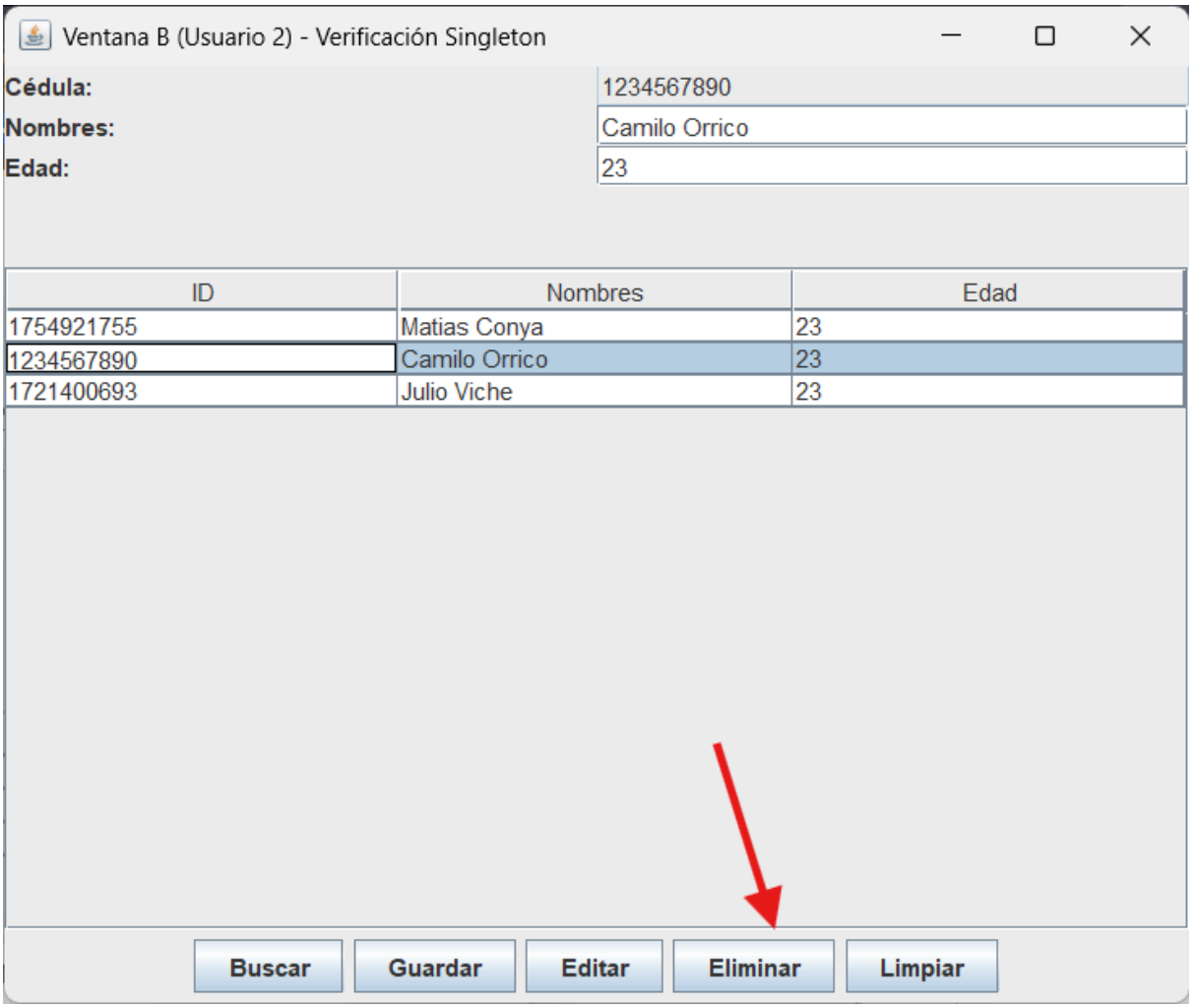
3. **Actualizar Estudiante:**

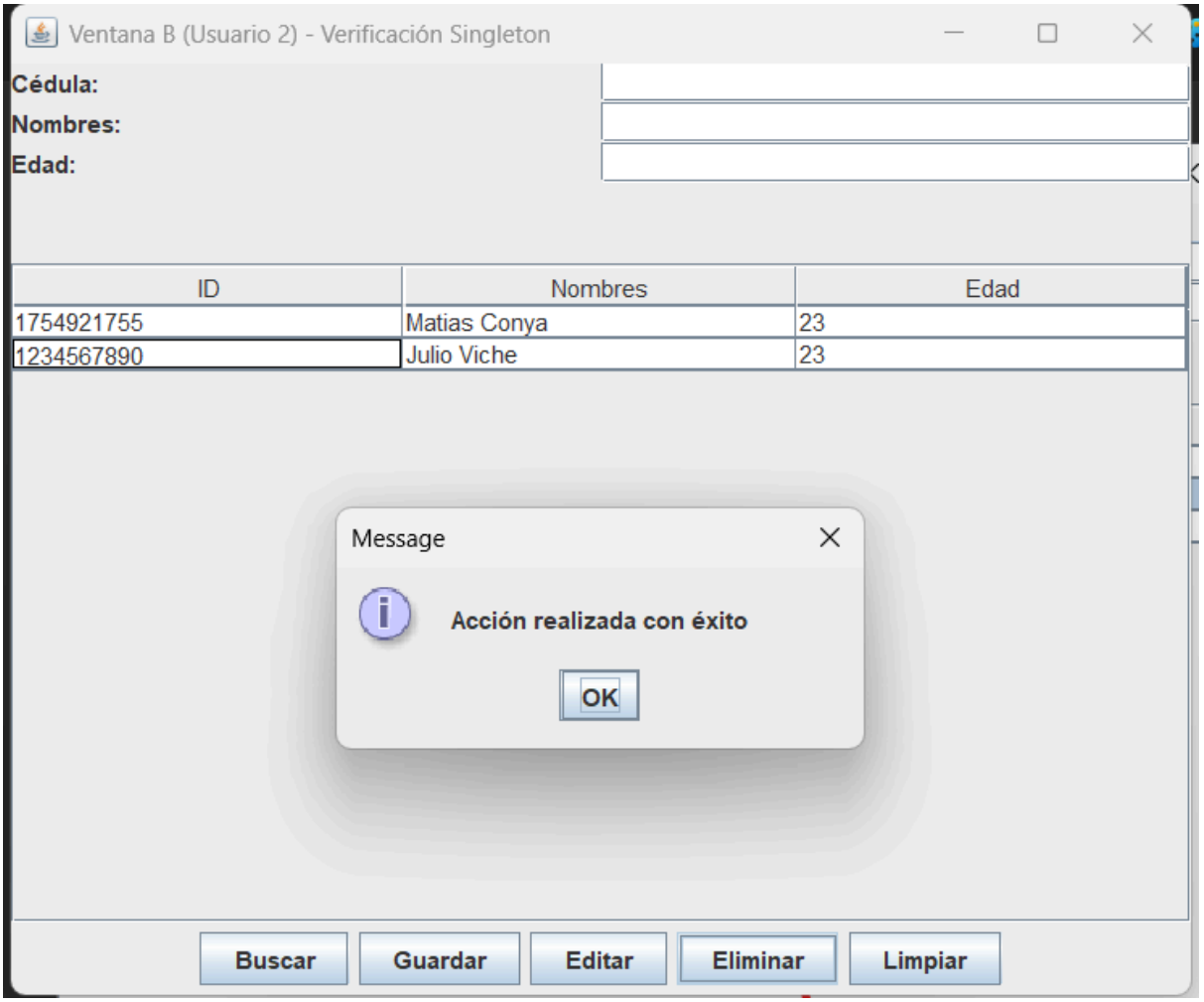



Ventana B (Usuario 2) - Verificación Singleton
Cédula:
1721400693
Nombres:
Julio Viche
Edad:
23
ID
Nombres
Edad
1754921755
Matias Conya
23
1234567890
Camilo Orrico
23
1721400693
Camilo Orrico
23
Buscar
Guardar
Editar
Eliminar
Limpiar


Ventana B (Usuario 2) - Verificación Singleton
Cédula:
Nombres:
Edad:
ID
Nombres
Edad
1754921755
Matias Conya
23
1234567890
Camilo Orrico
23
1721400693
Julio Viche
23
Message
Acción realizada con éxito
OK
Buscar
Guardar
Editar
Eliminar
Limpiar

4. **Eliminar Estudiante:**





5. Conclusión

En conclusión, este taller ha permitido comprender y aplicar la arquitectura de 3 capas junto con el patrón MVC para desarrollar un CRUD eficiente y bien estructurado para gestionar estudiantes. La separación de responsabilidades entre las capas mejora la organización del código y facilita su mantenimiento y escalabilidad. El uso del patrón Singleton en la capa de repositorio garantiza que la persistencia de los datos sea única, evitando inconsistencias en el almacenamiento de datos.

6. Recomendaciones

Para futuras mejoras y expansión del proyecto, se sugieren las siguientes recomendaciones:

- **Implementación de Validaciones Más Rigurosas:** Es importante agregar validaciones más estrictas, como verificar que el id tenga el formato adecuado o que no se ingresen datos vacíos.

- **Interfaz Gráfica:** Aunque se ha utilizado una interfaz de consola, se recomienda implementar una interfaz gráfica (por ejemplo, utilizando JavaFX o Swing) para mejorar la experiencia del usuario.

- **Persistencia en Base de Datos:** En lugar de usar un archivo de texto, sería más escalable y eficiente utilizar una base de datos para almacenar los estudiantes.

- **Pruebas Unitarias:** Es recomendable implementar pruebas unitarias para validar las operaciones de cada capa, lo que ayudará a detectar errores antes de la ejecución final.

7. Referencias

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional. (Referencia canónica para el Patrón Singleton).

Oracle. (2023). *Java Documentation: The Singleton Pattern*. Recuperado de https://docs.oracle.com/javase/tutorial/

Sommerville, I. (2011). *Software Engineering* (9th ed.). Addison-Wesley. (Referencia para Arquitectura de Software y separación de capas).