

Taller 2

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

El objetivo de este taller es comprender, aplicar e integrar el Patrón Singleton dentro de la capa de Datos y utilizar la arquitectura MVC (Modelo-Vista-Control) para estructurar correctamente una aplicación CRUD de Estudiantes. Se busca demostrar que la persistencia en memoria es compartida gracias al Singleton, evitando la pérdida de datos.

2. Organización de la Arquitectura MVC

La solución se ha organizado en paquetes separados para cumplir con la separación de responsabilidades:

- Datos (ec.edu.espe.datos): Contiene EstudianteRepository (Singleton).
- Negocio (ec.edu.espe.negocio): Contiene EstudianteService.
- Vista (ec.edu.espe.vista): Contiene EstudianteUI.
- Control (ec.edu.espe.controller): Contiene EstudianteController

3. Implementación de Singleton

Se aplicó el patrón de diseño Singleton en la clase EstudianteRepository.

¿Cómo funciona?

Se definió un constructor private para impedir que otras clases instancien el repositorio directamente y un atributo static que almacena la única instancia permitida. El acceso global se realiza mediante el método público getInstance().

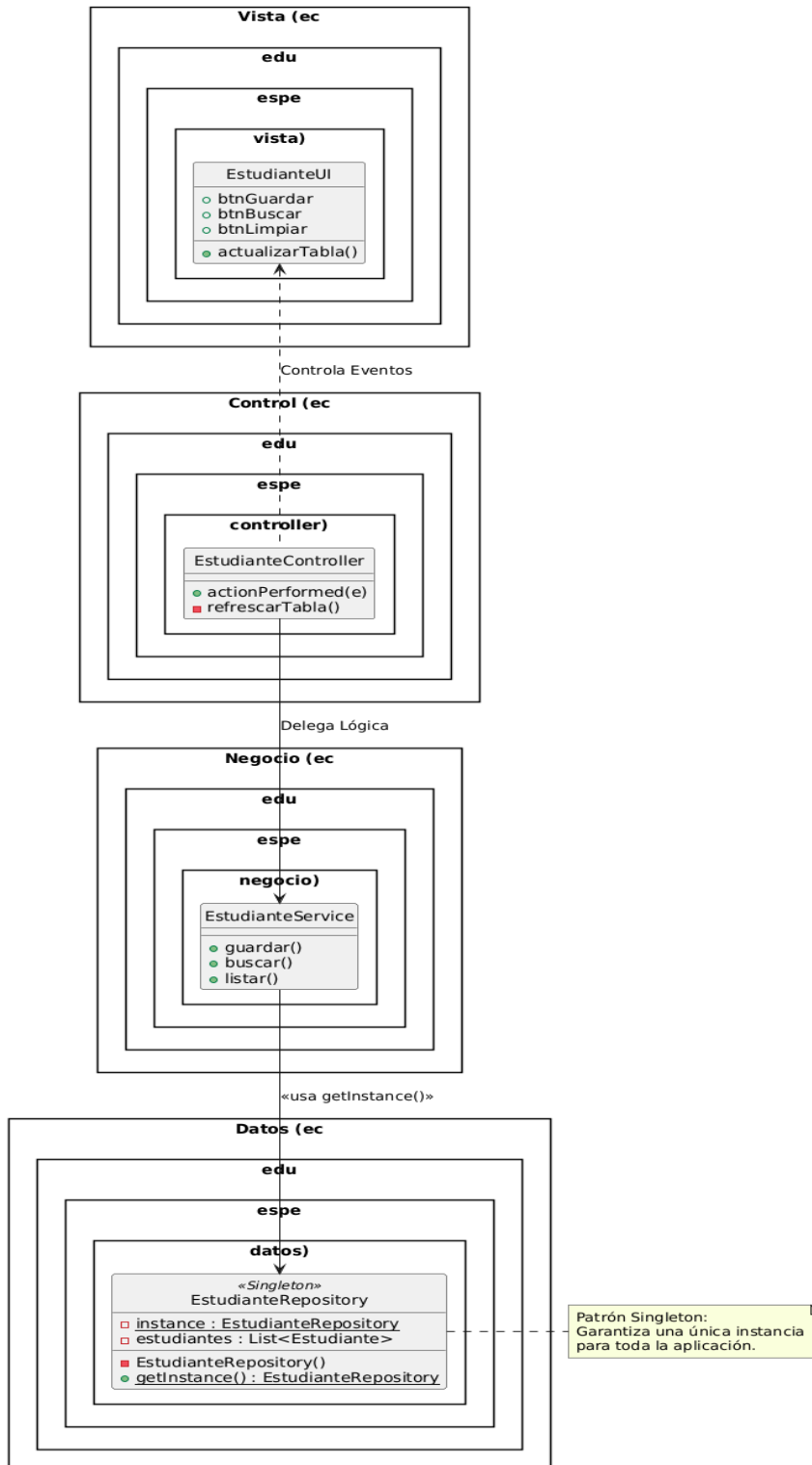
¿Por qué complementa a MVC?

En una arquitectura MVC, es crucial que el repositorio de datos sea único. Sin Singleton, si abrimos múltiples ventanas (Vistas), cada Controlador crearía su propia lista de estudiantes desconectada. Al usar Singleton,



garantizamos que toda la aplicación comparta la misma referencia en memoria y el mismo archivo físico (**estudiantes.txt**), asegurando la consistencia de los datos.

4. Diagrama de la estructura



5. Código fuente relevante

A continuación se presentan los fragmentos clave que demuestran la implementación de los patrones solicitados

A. Capa de Datos (Implementación Singleton)

Archivo: *EstudianteRepository.java*

```
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
    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(); }
}
}
```

B. Capa de Negocio (Consumo del Singleton)

Archivo: *EstudianteService.java*

```
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
    }
}
```



C. Capa de Control (Delegación)

Archivo: *EstudianteController.java*

```
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; // No refrescar tabla al buscar
    } else if (e.getSource() == view.btnLimpiar) {
        limpiar();
        return;
    }
    refrescarTabla();
    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,
1).toString());
        view.txtNombres.setText(view.tableModel.getValueAt(row,
2).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();
}
```



D. Clase Main (Prueba de Múltiples Controladores)

Archivo: *Main.java*

```
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)");

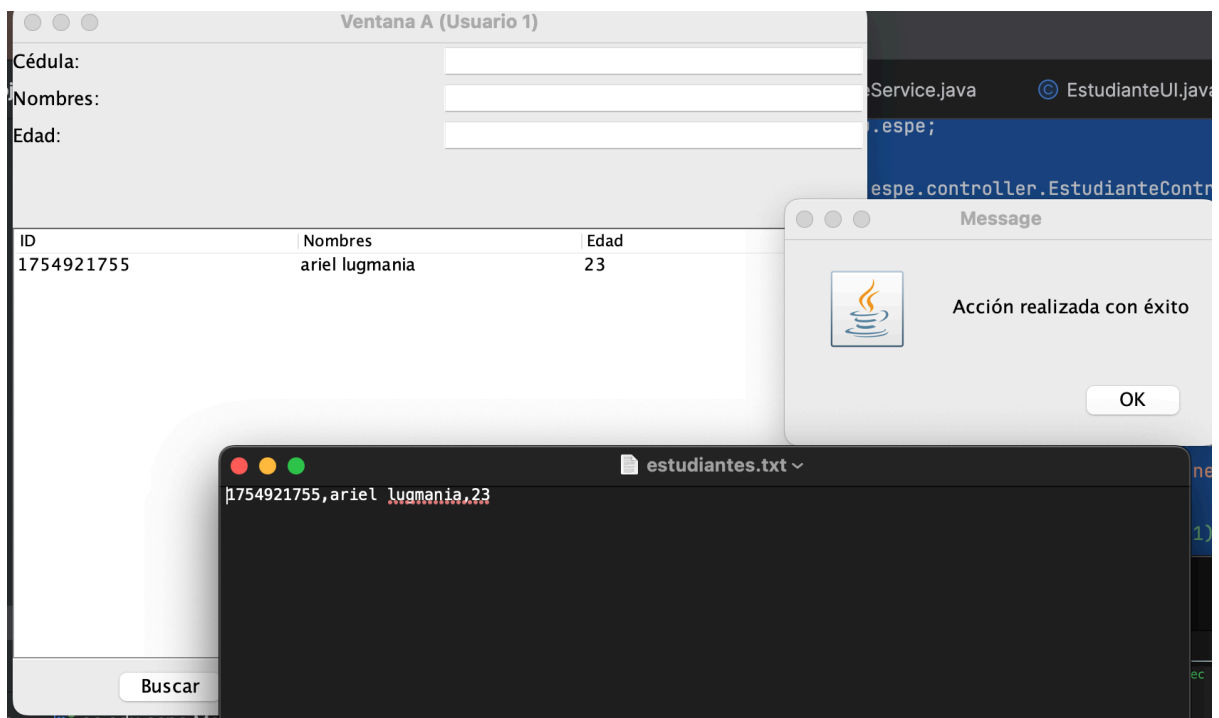
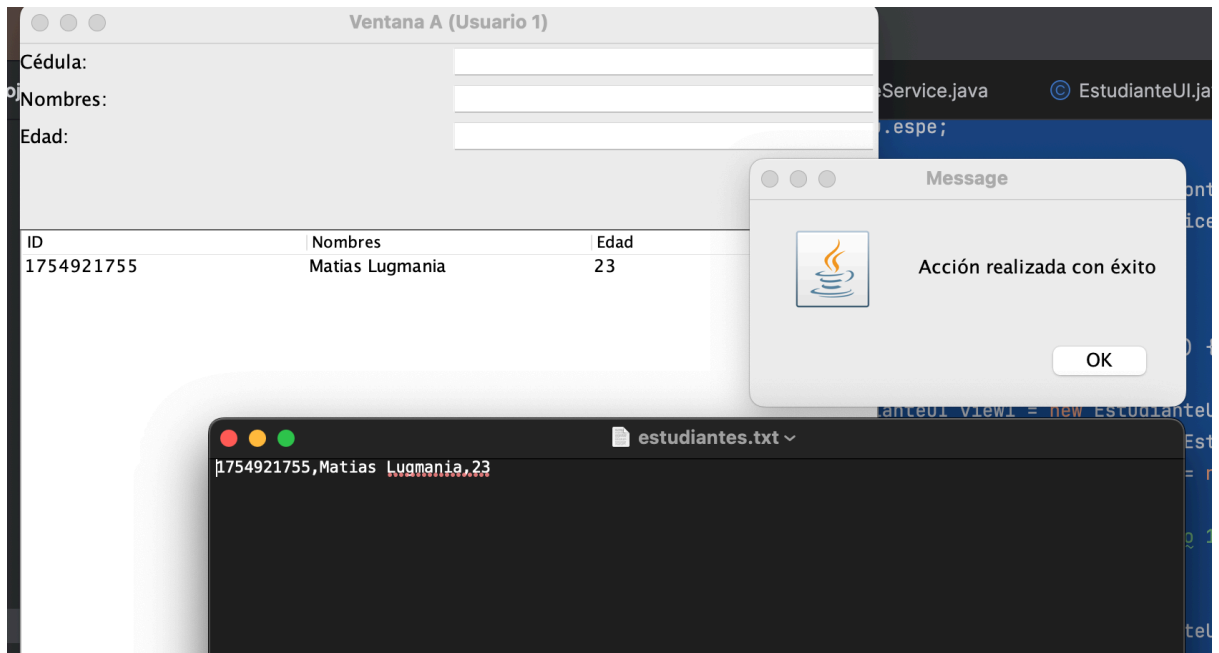
        EstudianteUI view2 = new EstudianteUI();
        EstudianteService service2 = new EstudianteService();
        EstudianteController controller2 = new EstudianteController(view2,
service2);
        view2.setVisible(true);
        view2.setTitle("Ventana B (Usuario 2) - Verificación Singleton");

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

6. Evidencias de funcionamiento

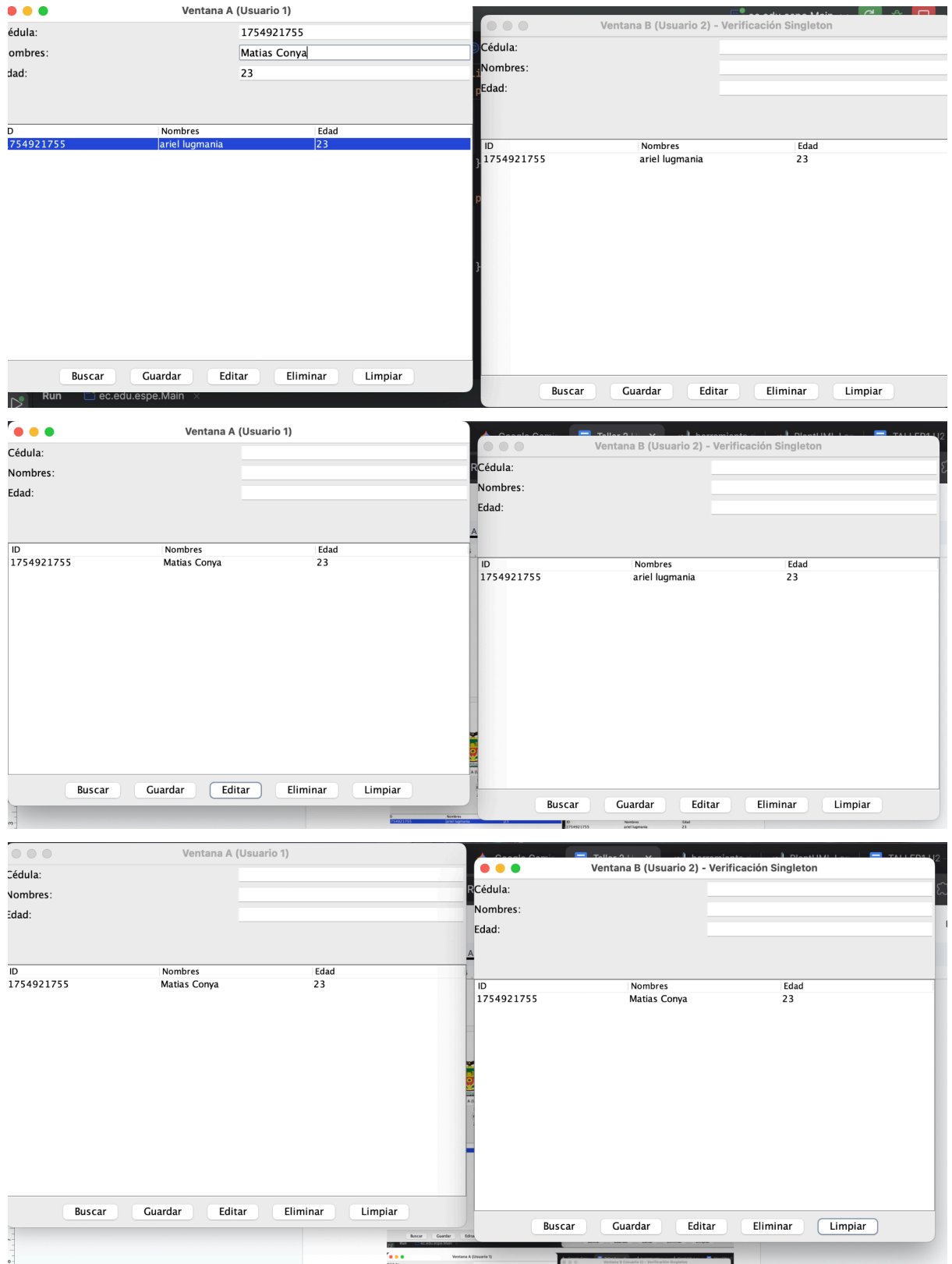
Evidencia 1: Persistencia en Archivo

La siguiente captura muestra el archivo plano (*estudiantes.txt*) actualizado correctamente tras guardar un registro desde la aplicación.



Evidencia 2: Prueba de Singleton con Múltiples Controladores

Se ejecutaron dos instancias de la Vista (Ventana A y Ventana B) simultáneamente mediante el **Main**. Como se observa, al interactuar con una ventana, la otra tiene acceso a los mismos datos, confirmando que comparten la misma instancia del Repositorio.



7. Conclusiones

Se logró implementar correctamente el patrón Singleton en la capa de persistencia, lo que solucionó los problemas de concurrencia y garantizó una única fuente de verdad para los datos.

La arquitectura MVC facilitó la organización del código, permitiendo que la interfaz gráfica sea independiente de la lógica de validación y almacenamiento.

8. 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).