Taller Patrón Singleton + NVC

CRUD de Estudiante

Integrantes:

Dennison Chalacan
Jeffrey Manobanda
Jhordy Marcillo

Tecnologías:

Java SE 8+ — Swing — Serialización

24 de noviembre de 2025

1. Implementación del Singleton

1.1. Código EstudianteRepository

El patrón Singleton garantiza una única instancia del repositorio compartida por toda la aplicación.

Listing 1: EstudianteRepository.java - Singleton

```java
public class EstudianteRepository {
    // 1. Instancia unica estatica
    private static EstudianteRepository instance;
    private List<Estudiante> estudiantes;

    // 2. Constructor privado
    private EstudianteRepository() {
        this.estudiantes = loadFromFile();
    }

    // 3. Metodo getInstance() sincronizado
    public static synchronized EstudianteRepository getInstance()
        {
        if (instance == null) {
            instance = new EstudianteRepository();
        }
        return instance;
    }

    // Metodos CRUD
    public boolean agregar(Estudiante e) {
        if (buscar(e.getId()) != null) return false;
        estudiantes.add(e);
        saveToFile();
        return true;
    }

    public boolean editar(Estudiante e) {
        Estudiante existente = buscar(e.getId());
        if (existente == null) return false;
        existente.setNombres(e.getNombres());
        existente.setEdad(e.getEdad());
        saveToFile();
        return true;
    }

    public boolean eliminar(String id) {
        Estudiante e = buscar(id);
        if (e == null) return false;
        estudiantes.remove(e);
        saveToFile();
        return true;
    }

```

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

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

1.2. Explicación del Singleton

Componentes clave:

1. **Instancia estática privada:** Almacena la única instancia
2. **Constructor privado:** Previene instanciación externa con `new`
3. **getInstance() sincronizado:** Punto de acceso único, thread-safe

Ventajas:

- Una sola lista de estudiantes en memoria
- Todos los componentes acceden a los mismos datos
- Evita inconsistencias y pérdida de información
- Control centralizado de persistencia

2. Integración del Patrón NVC

2.1. Diagrama de Arquitectura

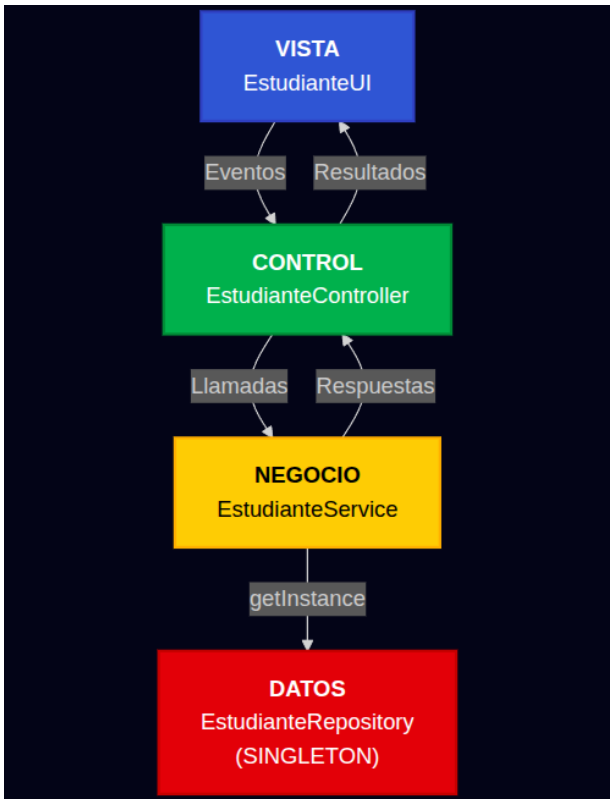

Figura 1: Diagrama de arquitectura NVC + Singleton

2.2. Código por Capas

Capa de Datos - Modelo:

Listing 2: Estudiante.java

```java
public class Estudiante implements Serializable {
    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
    public String getId() { return id; }
    public void setId(String id) { this.id = id; }
    public String getNombres() { return nombres; }
    public void setNombres(String n) { this.nombres = n; }
    public int getEdad() { return edad; }
```

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

Capa de Negocio:

Listing 3: EstudianteService.java

```
public class EstudianteService {
    private final EstudianteRepository repository;

    public EstudianteService() {
        // Usa la instancia unica del Singleton
        this.repository = EstudianteRepository.getInstance();
    }

    public ResultadoOperacion agregar(Estudiante e) {
        if (repository.buscar(e.getId()) != null) {
            return ResultadoOperacion.error("ID duplicado");
        }
        return repository.agregar(e)
            ? ResultadoOperacion.exito("Agregado")
            : ResultadoOperacion.error("Error al agregar");
    }

    public ResultadoOperacion editar(Estudiante e) {
        return repository.editar(e)
            ? ResultadoOperacion.exito("Editado")
            : ResultadoOperacion.error("ID no encontrado");
    }

    public ResultadoOperacion eliminar(String id) {
        return repository.eliminar(id)
            ? ResultadoOperacion.exito("Eliminado")
            : ResultadoOperacion.error("ID no encontrado");
    }

    public List<Estudiante> listar() {
        return repository.listar();
    }
}
```

Capa de Control:

Listing 4: EstudianteController.java

```
public class EstudianteController {
    private final EstudianteService service;

    public EstudianteController() {
        this.service = new EstudianteService();
    }

    public ResultadoOperacion guardarEstudiante(
            String id, String nombres, String edadStr) {
        try {
```

```
            int edad = Integer.parseInt(edadStr.trim());
            Estudiante e = new Estudiante(
                id.trim(), nombres.trim(), edad);
            return service.agregar(e);
        } catch (NumberFormatException ex) {
            return ResultadoOperacion.error("Edad inválida");
        }
    }

    public ResultadoOperacion editarEstudiante(
            String id, String nombres, String edadStr) {
        try {
            int edad = Integer.parseInt(edadStr.trim());
            Estudiante e = new Estudiante(
                id.trim(), nombres.trim(), edad);
            return service.editar(e);
        } catch (NumberFormatException ex) {
            return ResultadoOperacion.error("Edad inválida");
        }
    }

    public ResultadoOperacion eliminarEstudiante(String id) {
        return service.eliminar(id.trim());
    }

    public List<Estudiante> obtenerTodos() {
        return service.listar();
    }
}
```

Capa de Vista:
La clase `EstudianteUI` (interfaz Swing) incluye:

- Formulario: campos ID, Nombres, Edad
- Botones: Guardar, Editar, Eliminar, Limpiar
- Tabla para mostrar estudiantes
- Delega todas las acciones al `EstudianteController`

3. Pruebas de Persistencia Compartida

3.1. Código de Prueba

Listing 5: TestSingleton.java

```
public class TestSingleton {
    public static void main(String[] args) {
        System.out.println("=== PRUEBA SINGLETON ===\n");

        // Crear dos controladores diferentes
```

```
        EstudianteController controller1 =
            new EstudianteController();
        EstudianteController controller2 =
            new EstudianteController();

        // Controller1 agrega estudiantes
        System.out.println("Controller1 agregando:");
        controller1.guardarEstudiante("001", "Juan", "20");
        controller1.guardarEstudiante("002", "Maria", "22");

        // Controller2 lista (sin agregar nada)
        System.out.println("\nController2 listando:");
        controller2.obtenerTodos().forEach(e ->
            System.out.println("  " + e.getId() +
                " - " + e.getNombres()));

        // Controller2 agrega uno mas
        System.out.println("\nController2 agregando:");
        controller2.guardarEstudiante("003", "Carlos", "21");

        // Controller1 lista de nuevo
        System.out.println("\nController1 listando:");
        controller1.obtenerTodos().forEach(e ->
            System.out.println("  " + e.getId() +
                " - " + e.getNombres()));

        System.out.println("\ n  Ambos ven los 3 estudiantes");
        System.out.println("   Singleton funcionando!");
    }
}
```

3.2. Resultado de la Prueba

```
=== PRUEBA SINGLETON ===

Controller1 agregando:

Controller2 listando:
  001 - Juan
  002 - Maria

Controller2 agregando:

Controller1 listando:
  001 - Juan
  002 - Maria
  003 - Carlos

 Ambos ven los 3 estudiantes
 Singleton funcionando!
```

Evidencia: Los dos controladores comparten la misma lista. Controller2 ve los datos de Controller1 y viceversa, demostrando que ambos usan la misma instancia del repositorio (Singleton).

4. Singleton + NVC: Complemento y Prevención de Pérdida de Datos

4.1. Problema sin Singleton

Sin Singleton, cada `EstudianteService` crearía su propio repositorio:

```
Controller1 -> Service1 -> Repository1 (Lista A)
Controller2 -> Service2 -> Repository2 (Lista B)
```

Resultado: Datos inconsistentes, cada uno con su lista separada.

4.2. Solución con Singleton

Con Singleton, todos comparten la misma instancia:

```
Controller1 --\
               \
                --> Repository.getInstance() -> Lista ÚNICA
               /
Controller2 --/
```

Resultado: Consistencia global, sin pérdida de datos.

4.3. Cómo se Complementan

- **NVC:** Separa responsabilidades en capas independientes
- **Singleton:** Garantiza que la capa de Datos sea única y compartida
- **Juntos:** Arquitectura limpia + datos consistentes

Beneficios:

1. Vista y Control pueden existir en múltiples instancias sin problema
2. Todos acceden al mismo repositorio de datos
3. No hay pérdida ni duplicación de información
4. Facilita mantenimiento y escalabilidad

5. Estructura del Código Fuente

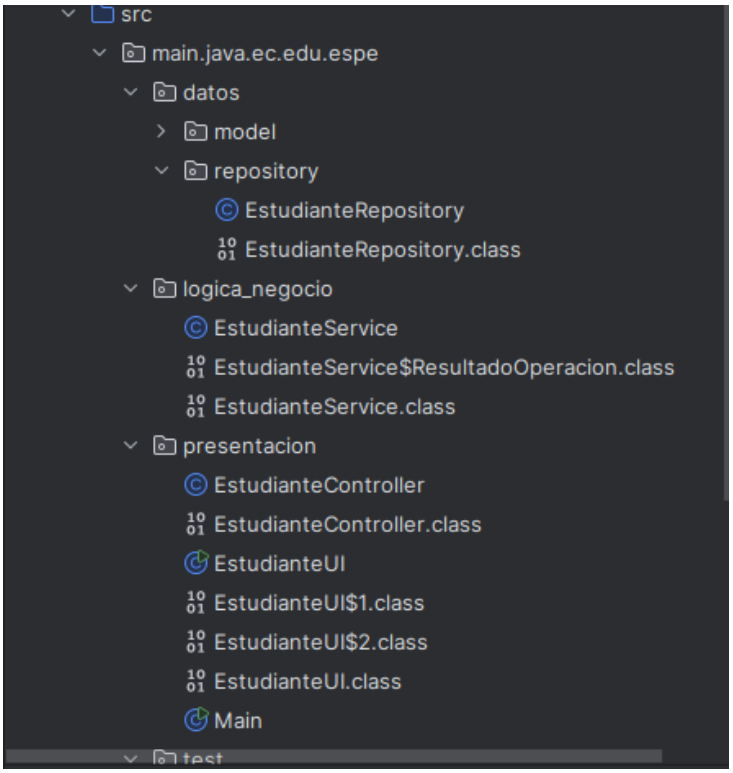

Figura 2: Estructura final del proyecto

6. Resultados

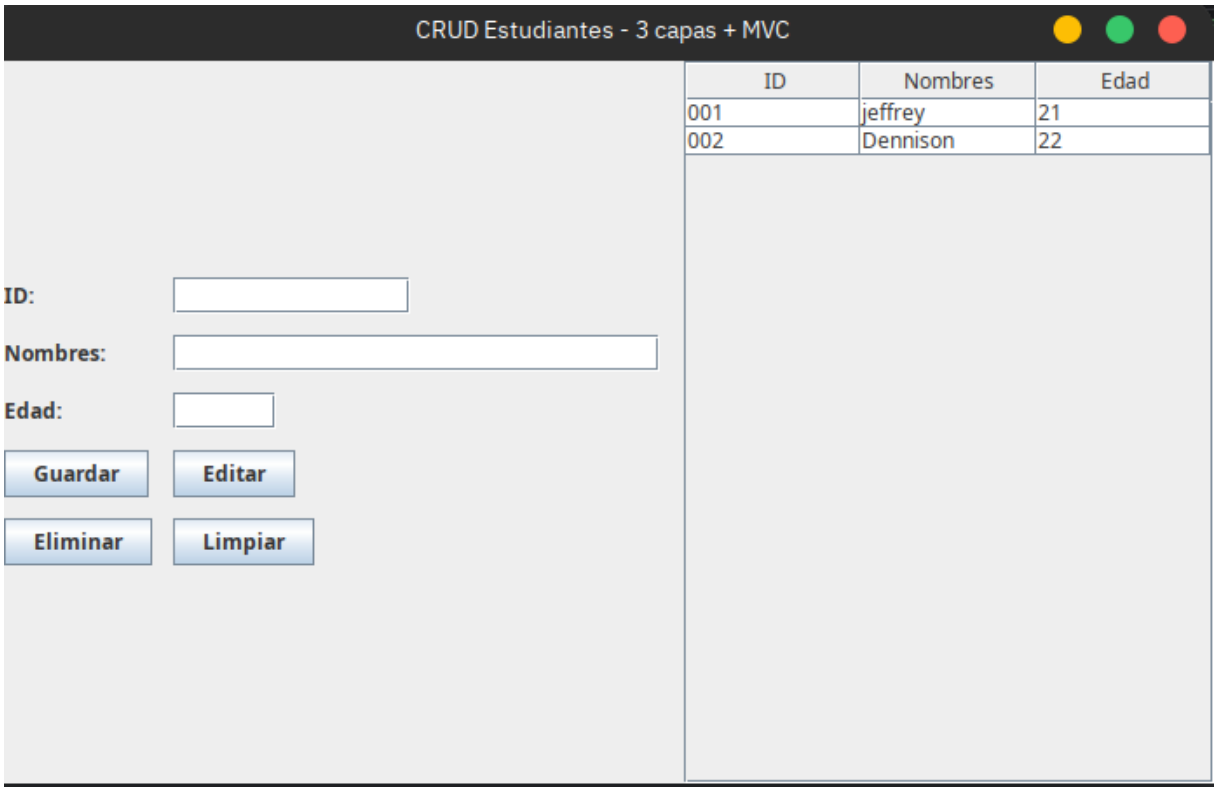

Figura 3: Estructura final del proyecto

El Singleton en el repositorio garantiza que todos los componentes de la arquitectura NVC accedan a los mismos datos, evitando inconsistencias y pérdida de información.