

Taller Patrón Singleton + NVC

CRUD de Estudiante

Integrantes:

Dennison Chalacan
Jeffrey Manobanda
Jhordy Marcillo

Tecnologías:

Java SE 8+ — Swing — Serialización

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

```
1 public class EstudianteRepository {
2     // 1. Instancia unica estatica
3     private static EstudianteRepository instance;
4     private List<Estudiante> estudiantes;
5
6     // 2. Constructor privado
7     private EstudianteRepository() {
8         this.estudiantes = loadFromFile();
9     }
10
11    // 3. Metodo getInstance() sincronizado
12    public static synchronized EstudianteRepository getInstance()
13    {
14        if (instance == null) {
15            instance = new EstudianteRepository();
16        }
17        return instance;
18    }
19
20    // Metodos CRUD
21    public boolean agregar(Estudiante e) {
22        if (buscar(e.getId()) != null) return false;
23        estudiantes.add(e);
24        saveToFile();
25        return true;
26    }
27
28    public boolean editar(Estudiante e) {
29        Estudiante existente = buscar(e.getId());
30        if (existente == null) return false;
31        existente.setNombres(e.getNombres());
32        existente.setEdad(e.getEdad());
33        saveToFile();
34        return true;
35    }
36
37    public boolean eliminar(String id) {
38        Estudiante e = buscar(id);
39        if (e == null) return false;
40        estudiantes.remove(e);
41        saveToFile();
42        return true;
43    }
```

```
44     public List<Estudiante> listar() {
45         return new ArrayList<>(estudiantes);
46     }
47
48     public Estudiante buscar(String id) {
49         return estudiantes.stream()
50             .filter(e -> e.getId().equals(id))
51             .findFirst().orElse(null);
52     }
53 }
```

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

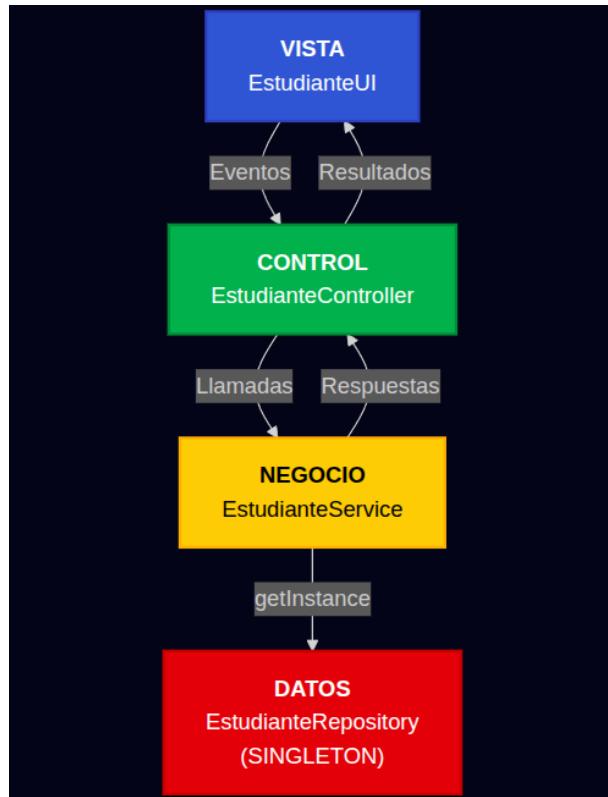


Figura 1: Diagrama de arquitectura NVC + Singleton

2.2. Código por Capas

Capa de Datos - Modelo:

Listing 2: Estudiante.java

```
1 public class Estudiante implements Serializable {
2     private String id;
3     private String nombres;
4     private int edad;
5
6     public Estudiante(String id, String nombres, int edad) {
7         this.id = id;
8         this.nombres = nombres;
9         this.edad = edad;
10    }
11
12    // Getters y Setters
13    public String getId() { return id; }
14    public void setId(String id) { this.id = id; }
15    public String getNombres() { return nombres; }
16    public void setNombres(String n) { this.nombres = n; }
17    public int getEdad() { return edad; }
```

```

18     public void setEdad(int edad) { this.edad = edad; }
19 }
```

Capa de Negocio:

Listing 3: EstudianteService.java

```

1 public class EstudianteService {
2     private final EstudianteRepository repository;
3
4     public EstudianteService() {
5         // Usa la instancia unica del Singleton
6         this.repository = EstudianteRepository.getInstance();
7     }
8
9     public ResultadoOperacion agregar(Estudiante e) {
10        if (repository.buscar(e.getId()) != null) {
11            return ResultadoOperacion.error("ID duplicado");
12        }
13        return repository.agregar(e)
14            ? ResultadoOperacion.exito("Agregado")
15            : ResultadoOperacion.error("Error al agregar");
16    }
17
18    public ResultadoOperacion editar(Estudiante e) {
19        return repository.editar(e)
20            ? ResultadoOperacion.exito("Editado")
21            : ResultadoOperacion.error("ID no encontrado");
22    }
23
24    public ResultadoOperacion eliminar(String id) {
25        return repository.eliminar(id)
26            ? ResultadoOperacion.exito("Eliminado")
27            : ResultadoOperacion.error("ID no encontrado");
28    }
29
30    public List<Estudiante> listar() {
31        return repository.listar();
32    }
33 }
```

Capa de Control:

Listing 4: EstudianteController.java

```

1 public class EstudianteController {
2     private final EstudianteService service;
3
4     public EstudianteController() {
5         this.service = new EstudianteService();
6     }
7
8     public ResultadoOperacion guardarEstudiante(
9         String id, String nombres, String edadStr) {
10        try {

```

```

11         int edad = Integer.parseInt(edadStr.trim());
12         Estudiante e = new Estudiante(
13             id.trim(), nombres.trim(), edad);
14         return service.agregar(e);
15     } catch (NumberFormatException ex) {
16         return ResultadoOperacion.error("Edad invalida");
17     }
18 }
19
20 public ResultadoOperacion editarEstudiante(
21     String id, String nombres, String edadStr) {
22     try {
23         int edad = Integer.parseInt(edadStr.trim());
24         Estudiante e = new Estudiante(
25             id.trim(), nombres.trim(), edad);
26         return service.editar(e);
27     } catch (NumberFormatException ex) {
28         return ResultadoOperacion.error("Edad invalida");
29     }
30 }
31
32 public ResultadoOperacion eliminarEstudiante(String id) {
33     return service.eliminar(id.trim());
34 }
35
36 public List<Estudiante> obtenerTodos() {
37     return service.listar();
38 }
39 }

```

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

```

1 public class TestSingleton {
2     public static void main(String[] args) {
3         System.out.println("== PRUEBA SINGLETION ==\n");
4
5         // Crear dos controladores diferentes

```

```

6     EstudianteController controller1 =
7         new EstudianteController();
8     EstudianteController controller2 =
9         new EstudianteController();
10
11    // Controller1 agrega estudiantes
12    System.out.println("Controller1\u201aagregando:");
13    controller1.guardarEstudiante("001", "Juan", "20");
14    controller1.guardarEstudiante("002", "Maria", "22");
15
16    // Controller2 lista (sin agregar nada)
17    System.out.println("\nController2\u201alistando:");
18    controller2.obtenerTodos().forEach(e ->
19        System.out.println(" \u201a" + e.getId() +
20            "\u201a" + e.getNombres()));
21
22    // Controller2 agrega uno mas
23    System.out.println("\nController2\u201aagregando:");
24    controller2.guardarEstudiante("003", "Carlos", "21");
25
26    // Controller1 lista de nuevo
27    System.out.println("\nController1\u201alistando:");
28    controller1.obtenerTodos().forEach(e ->
29        System.out.println(" \u201a" + e.getId() +
30            "\u201a" + e.getNombres()));
31
32    System.out.println("\n    \u201aAmbos\u201aven\u201alos\u201a3\u201aestudiantes");
33    System.out.println("      \u201aSingleton\u201afuncionando!");
34 }
35 }
```

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

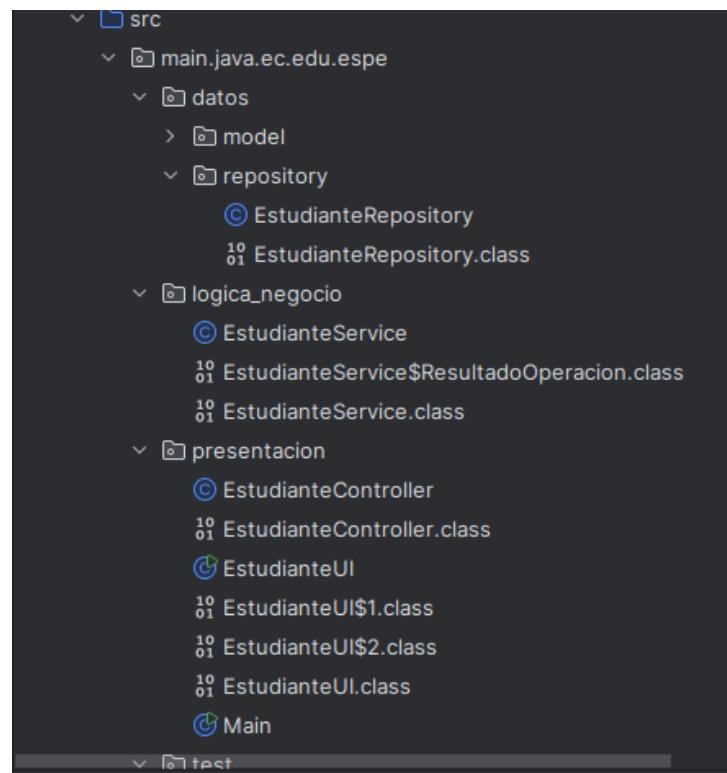


Figura 2: Estructura final del proyecto

6. Resultados

CRUD Estudiantes - 3 capas + MVC

ID	Nombres	Edad
001	jeffrey	21
002	Dennison	22

ID:

Nombres:

Edad:

Guardar **Editar**

Eliminar **Limpiar**

The screenshot displays a user interface for a CRUD application. At the top, there's a header bar with the text "CRUD Estudiantes - 3 capas + MVC" and three colored circles (yellow, green, red). Below the header is a table showing two student records: one with ID 001, name Jeffrey, and age 21; and another with ID 002, name Dennison, and age 22. To the left of the table is a form with fields for "ID", "Nombres", and "Edad", each with an associated input box. Below these fields are four buttons: "Guardar" (Save), "Editar" (Edit), "Eliminar" (Delete), and "Limpiar" (Clear). The entire interface is contained within a light gray box.

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.