TALLER N2 - U2

1. DATOS INFORMATIVOS

Carrera: Ingeniería de Software

Asignatura: Análisis y Diseño de Software

Tema del taller: Taller del Patrón Singleton con Patrón NVC

Docente: Mgt. Jenny Alexandra Ruiz Robalino

Integrantes:

- Amaguaña Casa Kevin Fernando
- Bonilla Hidalgo Jairo Smith
- Guamán Pulupa Alexander Daniel
- Tipán Ávila Reishel Dayelin

Fecha: 24/11/2025 Paralelo: 27835

2. DESARROLLO

• Explicación de la implementación del Singleton.

Codigo: Implementacion de Singleton en la clase EstudianteRepository asegurando una única instancia

```java
/**
 * Obtiene la instancia única del repositorio (Singleton)
 * @return la instancia única de EstudianteRepository
 */
public static EstudianteRepository getInstance() {
    if (instance == null) {
        synchronized (EstudianteRepository.class) {
            if (instance == null) {
                instance = new EstudianteRepository();
            }
        }
    }
    return instance;
}
```

Se realiza uso de getInstace() en la capa de servicio (EstudianteService)

```java
public class EstudianteService {
    private final EstudianteRepository repo;

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

Uso de EstudianteService en el controlador

```java
public class EstudianteControlador {
    private final EstudianteService service;
    private final String identificador;

    public EstudianteControlador(String identificador) {
        this.service = new EstudianteService();
        this.identificador = identificador;
        System.out.println("[Controlador " + identificador + "] Creado y conectado al servicio.");
    }
```

Pruebas:

Creando múltiples controladores y verifica que comparten la misma lista

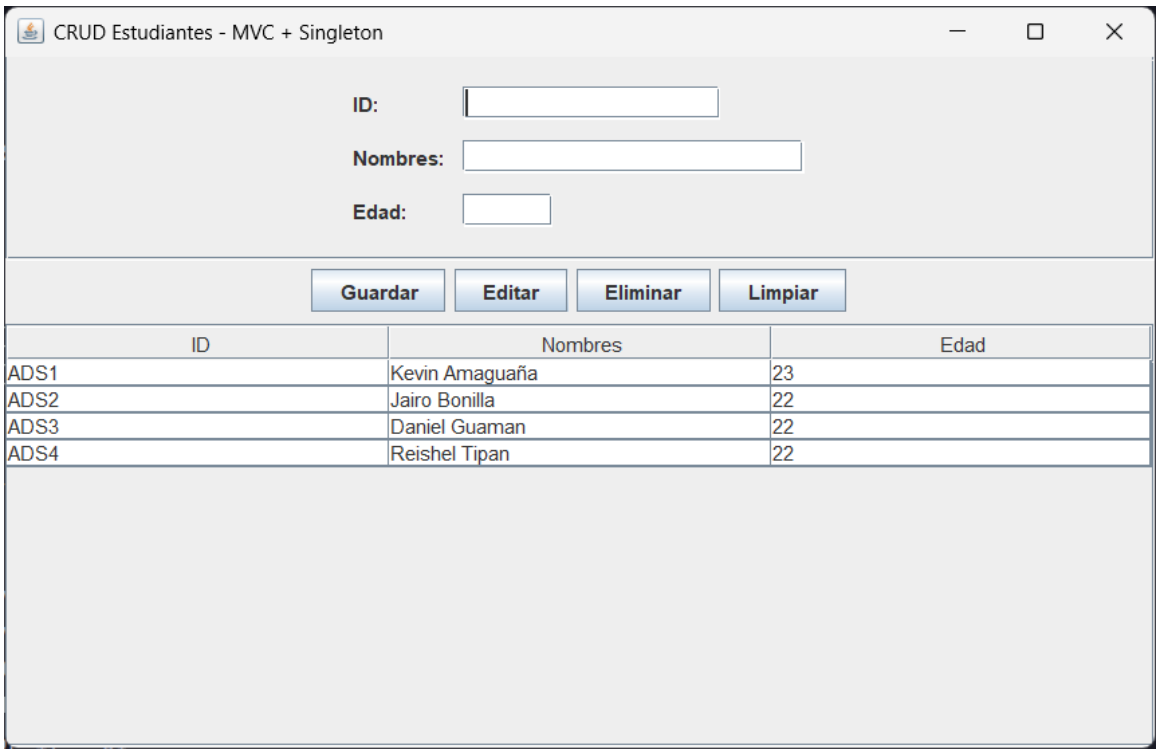
```
--- Prueba de Múltiples Controladores ---
[Controlador 1] Creado y conectado al servicio.
[Controlador 2] Creado y conectado al servicio.
[Controlador 1] CREAR: Estudiante{id='DEMO1', nombres='Estudiante Demo', edad=25} - Éxito
[Controlador 2] BUSCAR: Encontrado - Estudiante{id='DEMO1', nombres='Estudiante Demo', edad=25}
¿Controlador 2 ve estudiante de Controlador 1? ✓ SÍ
[Controlador 1] ELIMINAR: ID=DEMO1 - Éxito
--- Controladores verificados correctamente ---
```

Explicacion: cómo Singleton complementa a NVC y evita pérdida de datos en el repositorio.

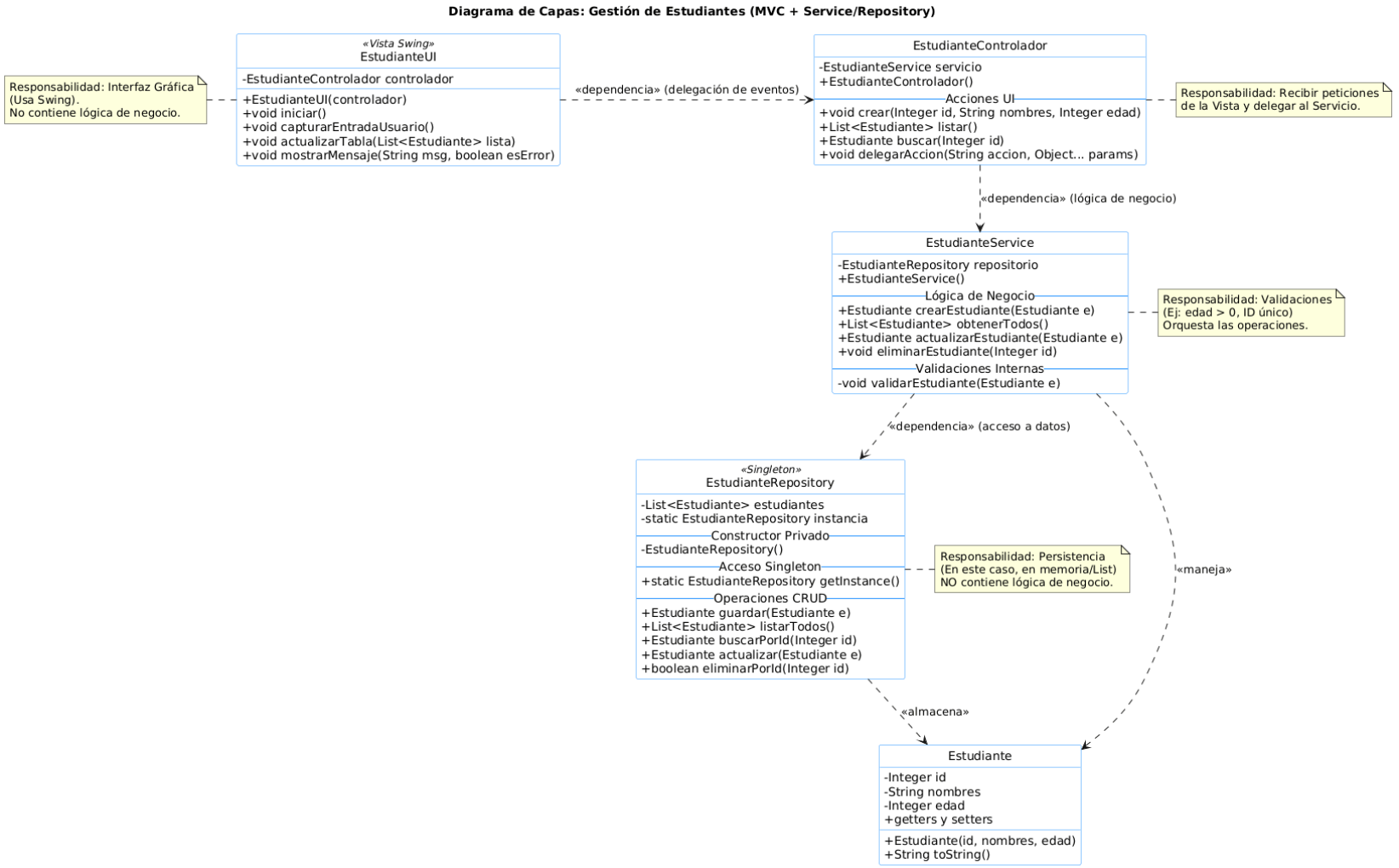
El patrón Singleton garantiza que existe una única instancia del EstudianteRepository en toda la aplicación, evitando la pérdida de datos en memoria. Cuando múltiples componentes (EstudianteService, EstudianteUI, u otras vistas) llaman a EstudianteRepository.getInstance(), todos obtienen la misma referencia al repositorio, compartiendo así el mismo HashMap<String, Estudiante> donde se almacenan los datos. Sin Singleton, cada new EstudianteRepository() crearía una instancia separada con su propia colección de datos, provocando que los estudiantes

agregados en un servicio no sean visibles en otro, fragmentando la información, lo cual es fundamental en donde múltiples controladores y vistas interactúan con el mismo modelo de datos.

Ejecución



- Diagrama NVC mostrando separación de responsabilidades.



• Evidencia de persistencia compartida.

```
╔═══════════════════════════════════════════════════════╗
║        DEMOSTRACIÓN PATRÓN SINGLETON + MVC + CONTROLADOR        ║
╚═══════════════════════════════════════════════════════╝

--- Prueba de Singleton ---
[Singleton] Nueva instancia de EstudianteRepository creada.
repo1 == repo2: true
HashCode repo1: 883049899
HashCode repo2: 883049899

Estudiante agregado vía repo1: Estudiante{id='TEST1', nombres='Estudiante Prueba', edad=20}
Estudiante obtenido vía repo2: Estudiante{id='TEST1', nombres='Estudiante Prueba', edad=20}
¿Se encuentra el mismo estudiante? true

--- Singleton verificado correctamente ---
```

3. CONCLUSIONES

La implementación del patrón **Singleton** en `EstudianteRepository` garantizó efectivamente una única instancia compartida entre todos los componentes de la aplicación, evitando la fragmentación de datos y asegurando consistencia en las operaciones CRUD. Las pruebas con múltiples controladores demostraron que los cambios realizados por un controlador son inmediatamente visibles para los demás, validando que la persistencia en memoria es compartida correctamente gracias al patrón Singleton.

La arquitectura implementada permitió una clara separación de responsabilidades: el modelo (`Estudiante`) representa los datos, el repositorio gestiona la persistencia Singleton, el servicio (`EstudianteService`) aplica las reglas de negocio y validaciones, el controlador (`EstudianteControlador`) coordina las acciones entre vista y servicio, y la vista (`EstudianteUI`) maneja la interfaz gráfica. Esta separación facilita el mantenimiento, testing y escalabilidad del código.

4. RECOMENDACIONES

Se recomienda migrar del almacenamiento en memoria a **persistencia permanente** utilizando bases de datos relacionales (MySQL, PostgreSQL) o NoSQL (MongoDB), manteniendo el patrón Singleton pero con datos que persistan entre ejecuciones. Integrar frameworks como **Spring Boot** con **JPA/Hibernate** facilitaría esta transición sin afectar la arquitectura existente.

Implementar **pruebas unitarias** para validar cada capa de la aplicación de forma independiente. Agregar testing de integración para verificar el flujo completo Vista-Controlador-Servicio-Repositorio, incluyendo escenarios edge cases como validaciones de datos inválidos, operaciones concurrentes y manejo de excepciones.

5. REFERENCIAS

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

Bloch, J. (2018). *Effective Java* (3rd ed.). Addison-Wesley Professional.

Martin, R. C. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Desig*n. Prentice Hall.