

Taller 3

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

1. DESARROLLO

Parte A: Arquitectura con MVC

Objetivo: Aplicar la arquitectura Modelo–Vista–Controlador en un CRUD de estudiante (ID,Nombres, Edad) basado en los documentos U2T1 y U2T2

Instrucciones:

1. Analiza la estructura de 3 capas: Modelo, Repositorio y Servicio.

El proyecto implementa una arquitectura MVC (Modelo-Vista-Controlador) organizada en 3 capas principales:

Capa Modelo (Entidad del Dominio)

Ubicación

ec.edu.espe.datos.modelo

Componentes

- **Estudiante.java** - Clase POJO que representa la entidad del dominio

Estructura

```
public class Estudiante {  
    private String id;          // Cédula del estudiante  
    private String nombres;      // Nombres completos  
    private int edad;           // Edad del estudiante  
  
    // Constructor, getters y setters  
}
```

Capa Repositorio (Acceso a Datos)

Ubicación

ec.edu.espe.datos.repository

Componentes

- `EstudianteRepository.java` - Manejo de persistencia

Operaciones

- `crear(Estudiante)` - Agregar nuevo estudiante
- `listar()` - Obtener todos los estudiantes
- `buscarPorId(String)` - Buscar estudiante por cédula
- `actualizar(Estudiante)` - Modificar datos de estudiante
- `eliminar(String)` - Eliminar estudiante por cédula

Características

- Patrón Singleton: Garantiza una única instancia
- Persistencia en archivo TXT (formato CSV)
- Métodos privados: `guardarEnArchivo()`, `cargarDesdeArchivo()`

Capa Servicio (Lógica de Negocio)

Ubicación

`ec.edu.espe.logica_negocio`

Componentes

- `EstudianteService.java` - Capa de lógica de negocio

Funciones principales

- `guardarEstudiante()` - Valida y guarda nuevo estudiante
- `editarEstudiante()` - Valida y actualiza estudiante existente
- `eliminarEstudiante()` - Elimina estudiante
- `obtenerEstudiantes()` - Retorna lista de estudiantes
- `buscarPorCedula()` - Busca estudiante específico

Validaciones

- Validación de cédula ecuatoriana (algoritmo del módulo 10)
- Validación de campos obligatorios
- Validación de duplicados (cédula única)
- Validación de edad positiva
- Validación de formato de provincia (01-24)

2. Identifica responsabilidades de cada capa.

Capa Modelo (Entidad del Dominio)

Responsabilidad

Representar los datos del estudiante como un objeto Java simple (POJO - Plain Old Java Object).

¿Qué hace?

Almacena id (cédula), nombres y edad

¿Qué no hace?

No contiene lógica de negocio ni persistencia

Capa Repositorio (Acceso a Datos)

Responsabilidad

Gestionar el acceso y persistencia de datos en el archivo `estudiantes.txt`.

¿Qué hace?

- Crear, leer, actualizar y eliminar estudiantes
- Guardar en archivo
- Cargar desde archivo

¿Qué no hace?

No valida reglas de negocio (solo persiste)

Capa Servicio (Lógica de Negocio)

Responsabilidad

Implementar las reglas de negocio, validaciones y coordinar las operaciones entre la Vista y el Repositorio.

¿Qué hace?

- Validar cédula ecuatoriana
- Validar duplicados
- Aplicar reglas de negocio
- Manejar excepciones

¿Qué no hace?

No conoce detalles de la UI ni de persistencia

3. Ejecuta un-CRUD estudiantil con MVC.

READ

Gestión Estudiantes (Validaciones Co...)

Datos del Estudiante

Cédula (Solo números): Buscar

Nombres (Solo letras):

Edad (Numérica):

Guardar Editar Eliminar Limpiar

Cédula	Nombres	Edad
1756177935	Denise Rea	23
1756177901	Belen Rea	27

CREATE

Gestión Estudiantes (Validaciones Co...)

Datos del Estudiante

Cédula (Solo números): 0707071502 Buscar

Nombres (Solo letras): Pedro Fernández

Edad (Numérica): 30

Guardar Editar Eliminar Limpiar

Cédula	Nombres	Edad
1756177935	Denise Rea	23
1756177901	Belen Rea	27

Gestión Estudiantes (Validaciones Co...)

Datos del Estudiante

Cédula (Solo números): Buscar

Nombre:
Edad (Numérica):

Mensaje

Guardado en TXT correctamente.

Aceptar

1756177935	Denise Rea	23
1756177901	Belen Rea	27
0707071502	Pedro Fernández	30

UPDATE

Gestión Estudiantes (Validaciones Co...)

Datos del Estudiante

Cédula (Solo números): 0707071502 Buscar

Nombres (Solo letras): Pedro Fernández

Edad (Numérica): 30

Guardar Editar Eliminar Limpiar

Cédula	Nombres	Edad
1756177935	Denise Rea	23
1756177901	Belen Rea	27
0707071502	Pedro Fernández	30

Gestión Estudiantes (Validaciones Co...)

Datos del Estudiante

Cédula (Solo números):	0707071502	Buscar
Nombres (Solo letras):	Pedro Hernández	
Edad (Numérica):	35	

Guardar Editar Eliminar Limpiar

Cédula	Nombres	Edad
1756177935	Denise Rea	23
1756177901	Belen Rea	27
0707071502	Pedro Fernández	30

Gestión Estudiantes (Validaciones Co...)

Datos del Estudiante

Cédula (Solo números):	<input type="text"/>	Buscar
Nombre:	<input type="text"/>	<input type="button" value="X"/>
Edad (Número):	<input type="text"/>	<input type="button" value="X"/>

Mensaje

Editado y actualizado en TXT.

Aceptar

1756177935	Denise Rea	23
1756177901	Belen Rea	27
0707071502	Pedro Hernández	35

DELETE

Gestión Estudiantes (Validaciones Co...)

Datos del Estudiante

Cédula (Solo números):	0707071502	Buscar	
Nombres (Solo letras):	Pedro Hernández		
Edad (Numérica):	35		
<input type="button" value="Guardar"/>	<input type="button" value="Editar"/>	<input type="button" value="Eliminar"/>	<input type="button" value="Limpiar"/>

Cédula	Nombres	Edad
1756177935	Denise Rea	23
1756177901	Belen Rea	27
0707071502	Pedro Hernández	35

Gestión Estudiantes (Validaciones Co...)

Datos del Estudiante

Cédula (Solo números):	0707071502	Buscar
Nombre:	<input type="text" value="Pedro Hernández"/>	
Edad (Número):	<input type="text" value="35"/>	
<input type="button" value="Guardar"/>	<input type="button" value="Editar"/>	<input type="button" value="Eliminar"/>

Confirmar

¿ Eliminar cédula 0707071502?

Cédula	Nombres	Edad
1756177935	Denise Rea	23
1756177901	Belen Rea	27
0707071502	Pedro Hernández	35

Datos del Estudiante

Cédula	Nombres	Edad
1756177935	Denise Rea	23
1756177901	Belen Rea	27

4. Explica cómo MVC favorece la separación de responsabilidades.

La arquitectura MVC aplicada en este proyecto favorece la separación de responsabilidades al dividir la aplicación en tres capas: Modelo (Estudiante + Repository), Controlador (Service) y Vista (UI).

Esta separación permite que cada componente tenga una única responsabilidad definida, lo que resulta en bajo acoplamiento entre las capas:

- **El Modelo** se encarga de representar los datos y su persistencia, sin conocer nada sobre la lógica de negocio o la interfaz de usuario.
- **El Servicio (Controlador)** implementa todas las validaciones y reglas de negocio, actuando como intermediario entre la Vista y el Modelo, sin tener conocimiento de los detalles de persistencia ni de los componentes gráficos.
- **La Vista** maneja únicamente la presentación y captura de eventos del usuario, delegando toda la lógica al Servicio.

Este diseño permite que los cambios en una capa no afecten a las demás.

Por ejemplo, si se decide cambiar la persistencia de archivo TXT a una base de datos SQL, solo se modifica el Repository sin tocar el Service ni la UI. De igual manera, se puede reemplazar la interfaz Swing por una aplicación web sin alterar la lógica de negocio.

Además, esta separación facilita enormemente el testing, ya que cada capa puede probarse de forma independiente mediante pruebas unitarias. El resultado es un código más mantenable, escalable y reutilizable que cumple con los principios SOLID de diseño orientado a objetos.

Parte B: Arquitectura con Singleton

Objetivo: Implementar Singleton en la capa de datos para evitar múltiples instancias del repo

1. Revisar la clase EstudianteRepository transformada en Singleton.

```
public class EstudianteRepository {  
    // Patrón Singleton  
    private static EstudianteRepository instancia;  
  
    private List<Estudiante> estudiantes;  
    private final String FILE_NAME = "estudiantes.txt";  
  
    // Constructor privado para evitar instanciación directa  
    private EstudianteRepository() {  
        this.estudiantes = new ArrayList<>();  
        cargarDesdeArchivo(); // Carga datos al iniciar  
    }  
  
    // Método para obtener la única instancia (thread-safe)  
    public static synchronized EstudianteRepository getInstancia() {  
        if (instancia == null) {  
            instancia = new EstudianteRepository();  
        }  
        return instancia;  
    }
```

2. Ejecutar el CRUD garantizando una única lista compartida.

Cédula	Nombres	Edad
1756177935	Denise Rea	23
1756177901	Belen Rea	27
1714771654	Eduardo	48

Datos del Estudiante

Cédula (Solo números):

Nombres (Solo letras):

Edad (Numérica):

Buttons: Guardar, Editar, Eliminar, Limpiar

Cédula	Nombres	Edad
1756177935	Denise Rea	23
1756177901	Belen Rea	27
1714771654	Eduardo	48

Datos del Estudiante

Cédula (Solo números):

Nombres (Solo letras):

Edad (Numérica):

Buttons: Guardar, Editar, Eliminar, Limpiar

Cédula	Nombres	Edad
1756177935	Message	
1756177901	 Editado y actualizado en TXT.	
1714771654	<input type="button" value="OK"/>	

Cédula	Nombres	Edad
1756177935	Noemi Rea	23
1756177901	Belen Rea	27
1714771654	Eduardo	48

3. Comparar resultados con MVC sin Singleton.

Para realizar una comparativa directa con una implementación sin un patrón de diseño, se ha configurado un entorno de pruebas con la finalidad de analizar el rendimiento en base a métricas importantes:

=====

PRUEBA 1: Uso de Memoria - Creación de Instancias

=====

Singleton (100 refs → 1 instancia): 2.531.984 bytes (2.472,64 KB)

Sin Singleton (100 instancias): 6.815.312 bytes (6.655,58 KB)

Singleton ahorra: 4.283.328 bytes (4.182,94 KB)

Ahorro porcentual: 169,2%

Factor de ahorro: 2,7x

Singleton: 1 objeto + 100 referencias

Sin Singleton: 100 objetos completos

=====

PRUEBA 2: Rendimiento - Operación CREATE

=====

Singleton: 584,49 ms (584.488.800 ns)

Sin Singleton: 380,53 ms (380.526.600 ns)

Sin Singleton es 34,9% más rápido

=====

PRUEBA 3: Rendimiento - Operación READ (Buscar por ID)

=====

Singleton: 6,91 ms (6.914.000 ns)
Sin Singleton: 7,89 ms (7.892.100 ns)

Singleton es 12,4% más rápido

=====

PRUEBA 4: Rendimiento - Operación UPDATE

=====

Singleton: 344,95 ms (344.949.900 ns)
Sin Singleton: 482,22 ms (482.223.500 ns)

Singleton es 28,5% más rápido

=====

PRUEBA 5: Rendimiento - Operación DELETE

=====

Singleton: 420,59 ms (420.585.300 ns)
Sin Singleton: 225,90 ms (225.896.400 ns)

Sin Singleton es 46,3% más rápido

=====

PRUEBA 6: Rendimiento - Acceso Concurrente

=====

Configuración: 5 threads, 500 operaciones por thread

Singleton: 457,10 ms (457.099.700 ns)
Sin Singleton: 796,31 ms (796.306.500 ns)

Singleton es 42,6% más rápido

=====

PRUEBA 7: Throughput - Operaciones por Segundo

=====

Singleton: 1.632 ops/seg
Sin Singleton: 1.371 ops/seg

Singleton es 19,0% más rápido

Ventajas de usar Singleton:

- Ahorro significativo de memoria (una sola instancia)
- Mejor rendimiento en concurrencia
- Control centralizado del estado

- Evita creación redundante de objetos

Ventajas de usar MVC sin Singleton:

- Operaciones CRUD más rápidas (sin sincronización)
- Mejor para operaciones aisladas
- Más fácil de testear
- Sin estado compartido

4. Explicar el impacto en la persistencia de datos

En Singleton la persistencia de datos, se mantiene, y es más adecuada al trabajar con una sola instancia, ya que se evita la pérdida de datos, y la actualización de datos en tiempo real.

Realiza un cuadro comparativo entre MVC y Singleton respondiendo:

- ¿Qué problema resuelve cada uno?
- ¿En qué capa se utiliza?
- ¿Cómo influye en el mantenimiento?
- ¿Cómo evita fallas de diseño?

	MVC	Singleton
Problema que resuelve	Garantiza una única instancia compartida para evitar inconsistencias de datos, conflictos al acceder al archivo estudiantes.txt y desperdicio de memoria por objetos duplicados.	Consistencia de Datos Sincronización del archivo Problema de desperdicio de recursos
Capa en que se utiliza	Se implementa en la capa de Repository (EstudianteRepository) y en la capa de Service (EstudianteService) para centralizar el acceso a datos y la lógica de negocio.	Capa de Acceso de Datos, es la capa intermedia del modelo. Para ser más específicos en Repository.
Mantenimiento	Facilita el mantenimiento al centralizar los cambios, simplifica el debugging al tener una única instancia rastreable y permite evolucionar el código sin afectar múltiples objetos.	Se obtiene cambios centralizados, debugging simplificado, evolución del código.
Prevención de fallas	Elimina código duplicado de instanciación y protege contra pérdida o corrupción de datos al coordinar todas	Elimina la duplicación de código, previene condiciones de carrera, protege contra la pérdida

	las operaciones de escritura.	de datos.
--	-------------------------------	-----------

2. CONCLUSIONES

El Patrón Singleton demostró ser altamente efectivo en el consumo de memoria, garantizando consistencia de datos y proporcionando escalabilidad controlada al limitar la instancia a un solo objeto. No obstante, presenta desafíos significativos en el rendimiento debido a posibles cuellos de botella en entornos concurrentes y complejidad elevada en la implementación de pruebas unitarias por su naturaleza de estado global.

3. RECOMENDACIONES

Se recomienda implementar el patrón Singleton cuando múltiples hilos requieren acceso sincronizado al mismo recurso, especialmente en casos donde la creación del objeto consume recursos significativos como conexiones a bases de datos, sistemas de caché o configuraciones globales. Es particularmente útil cuando se necesita garantizar consistencia global y un punto único de verdad en toda la aplicación, así como para la gestión centralizada de configuraciones o servicios compartidos.

Sin embargo, debe evitarse su uso cuando se requiere testing extensivo con técnicas de mocking, el rendimiento concurrente es crítico para la aplicación, o cuando la aplicación necesita escalar horizontalmente. En aplicaciones .NET modernas, se recomienda considerar el uso de Dependency Injection con lifetime Singleton como alternativa, ya que proporciona beneficios similares de instancia única pero con mayor facilidad para realizar pruebas unitarias y mejor mantenibilidad del código.

4. REFERENCIAS

- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.
- Freeman, E., & Freeman, E. (2020). *Head First Design Patterns: Building Extensible and Maintainable Object-Oriented Software* (2nd ed.). O'Reilly Media. <https://www.oreilly.com/library/view/head-first-design/9781492077992/>
- Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley Professional. <https://martinfowler.com/books/eaa.html>