DEPARTAMENTO DE CIENCIAS DE LA COMPUTACION

PROGRAMACION ORIENTADA A OBJETOS

CONTROL DE LECTURA N° 2

TEMA: MODELO VISTA CONTROLADOR (MVC) Y PATRONES DE DISEÑO

REALIZADO POR:

Pablo Guber Camacho Bravo

Estalyn Daniel Licuy Mecías

Nayeli Scarleth Loachamin Tipan

Deisy Abigail Quillupangui Tupe

DOCENTE:

Luis Enrique Jaramillo Montaño

Sangolquí – Ecuador

Modelo Vista Controlador (MVC) y Patrones de Diseño

1. Introducción

En el desarrollo de software, la arquitectura y los patrones de diseño son esenciales para crear aplicaciones escalables, mantenible y eficientes. El Modelo Vista Controlador (MVC) es uno de los patrones arquitectónicos más utilizados en el diseño de aplicaciones, especialmente en aplicaciones web y móviles. Este patrón se enfoca en separar la lógica de la aplicación en tres componentes principales: el Modelo, la Vista y el Controlador. Al emplear el MVC, los desarrolladores pueden facilitar la reutilización del código, mejorar la organización y facilitar el mantenimiento.

Junto con el MVC, existen diversos patrones de diseño que ayudan a estructurar mejor las aplicaciones, cada uno con su propósito específico para resolver problemas comunes durante el desarrollo de software, como la creación de interfaces de usuario, la gestión de la persistencia de datos, o la comunicación entre componentes.

2. Objetivos

2.1 Objetivo General

Analizar el Modelo Vista Controlador (MVC) y los Patrones de Diseño para comprender su impacto en el desarrollo de software y su aplicabilidad en distintos contextos.

2.2 *Objetivo Específico*

a. Explicar la estructura y funcionamiento del Modelo Vista Controlador (MVC), detallando sus componentes principales y su rol en la arquitectura de software.
b. Identificar y describir los principales Patrones de Diseño utilizados en el desarrollo de software, resaltando su importancia y aplicabilidad.
c. Comparar las ventajas y desventajas del uso de MVC y los Patrones de Diseño, analizando su impacto en la escalabilidad y mantenimiento de sistemas.
d. Proporcionar recomendaciones y buenas prácticas para la implementación de MVC y Patrones de Diseño en proyectos de software, basadas en estudios y experiencias de la industria.

3. Marco Teórico

El patrón Modelo Vista Controlador (MVC) es un enfoque arquitectónico que divide una aplicación en tres componentes principales:

- **Modelo:** Maneja la lógica de negocio y los datos.
- **Vista:** Se encarga de la presentación de la información.
- **Controlador:** Actúa como intermediario entre el modelo y la vista, gestionando la interacción del usuario.

Estos patrones mejoran la modularidad, reutilización de código y escalabilidad en el desarrollo de software.

3.1 Análisis del proceso

El proceso de implementación de patrones de diseño implica varias etapas clave:

1. **Identificación de problemas en el diseño:** Se analizan los requerimientos del software y se identifican problemas recurrentes en la arquitectura.
2. **Selección del patrón adecuado:** Se elige el patrón de diseño más adecuado para resolver el problema identificado.
3. **Implementación del patrón:** Se incorpora el patrón en el código fuente de la aplicación, asegurando su correcta integración.
4. **Evaluación y optimización:** Se realizan pruebas para verificar la eficiencia del patrón y se optimiza en caso de ser necesario.

3.2 Análisis de requisitos

Requisitos Funcionales:

- Separación clara de responsabilidades en la aplicación.
- Implementación de una arquitectura flexible y modular.
- Facilitar la reutilización de componentes del software.
- Mejorar la interacción entre los diferentes módulos del sistema.

Requisitos No Funcionales:

- Mejorar la mantenibilidad del código.
- Garantizar la escalabilidad y adaptabilidad del software.
- Facilitar la depuración y pruebas del sistema.
- Optimizar el rendimiento del software mediante estructuras bien definidas.

4. ***Recomendaciones***

a. Uso adecuado de la separación de responsabilidades: Es crucial mantener una clara separación entre el Modelo, la Vista y el Controlador, ya que esto facilita la escalabilidad y el mantenimiento de la aplicación. Asegúrese de que el Modelo no dependa de la Vista y viceversa.
b. Modularización del código: Los patrones de diseño deben ser implementados de manera modular, donde cada componente se pueda desarrollar, probar y mantener de forma independiente.
c. Adaptación a nuevas tecnologías: Al implementar MVC, es importante estar al tanto de las nuevas herramientas y marcos de trabajo que apoyan este patrón, como React, Angular o Laravel, para mejorar la productividad y las buenas prácticas.
d. Pruebas unitarias y de integración: Realizar pruebas exhaustivas, especialmente a nivel de controladores, ya que son los que gestionan la interacción entre el Modelo y la Vista. Esto garantizará una aplicación robusta y libre de errores

5. ***Conclusiones***

a. Facilita la mantenibilidad: La separación clara entre el Modelo, la Vista y el Controlador en MVC facilita el mantenimiento y la modificación del software. Esto es especialmente útil en aplicaciones a largo plazo donde los requisitos cambian frecuentemente.
b. Mejora la escalabilidad: Al dividir la aplicación en tres componentes, el patrón MVC hace que sea más fácil escalar y añadir nuevas funcionalidades sin comprometer la estructura general del sistema.
c. Fomenta el desarrollo colaborativo: MVC permite que equipos de desarrollo trabajen de forma más eficiente. Los desarrolladores de la Vista, por ejemplo, pueden trabajar sin interferir con el Modelo o el Controlador, y viceversa.
d. Compatible con múltiples plataformas: El patrón MVC es muy flexible y puede adaptarse a diversas plataformas y lenguajes de programación, lo que lo convierte en una opción popular en el desarrollo web y móvil.

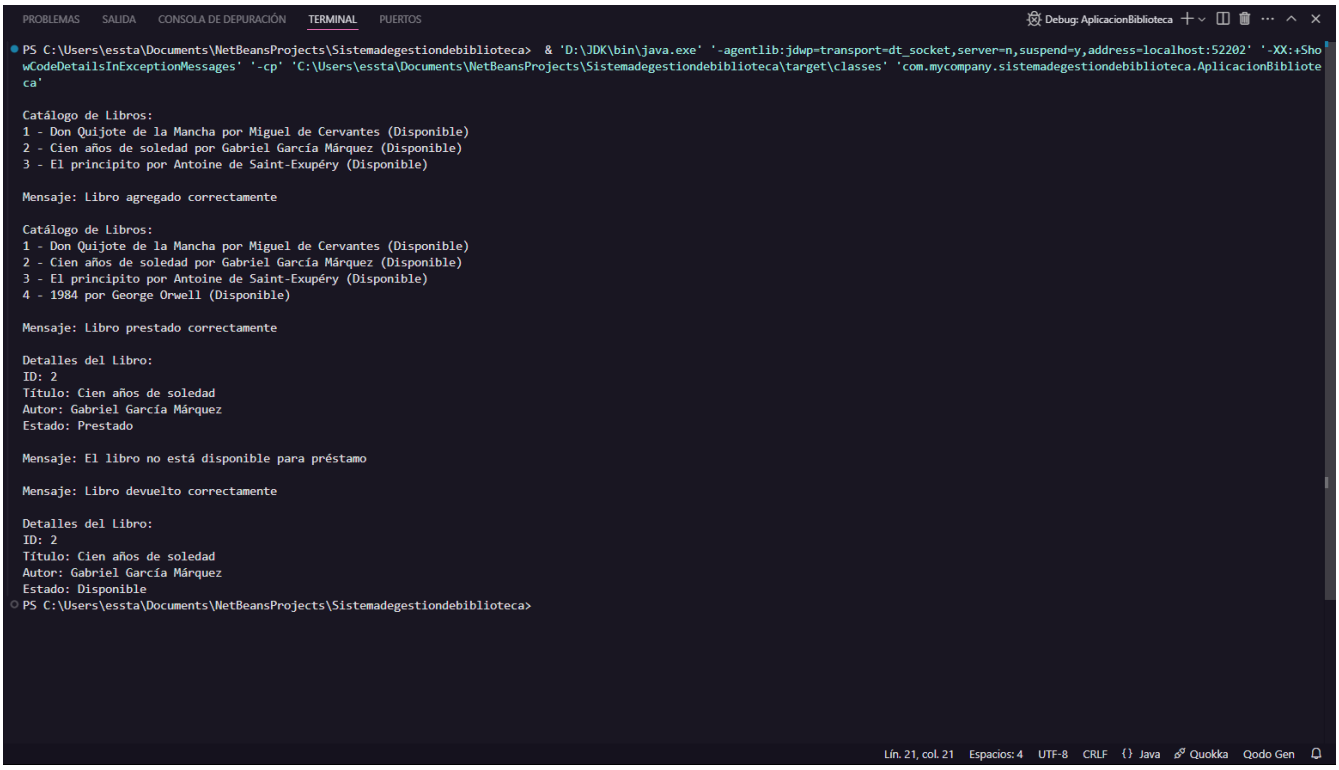
6. ***Resultados***

Aquí se presenta las capturas del funcionamiento de los cogidos

Sistemadegestiondebiblioteca

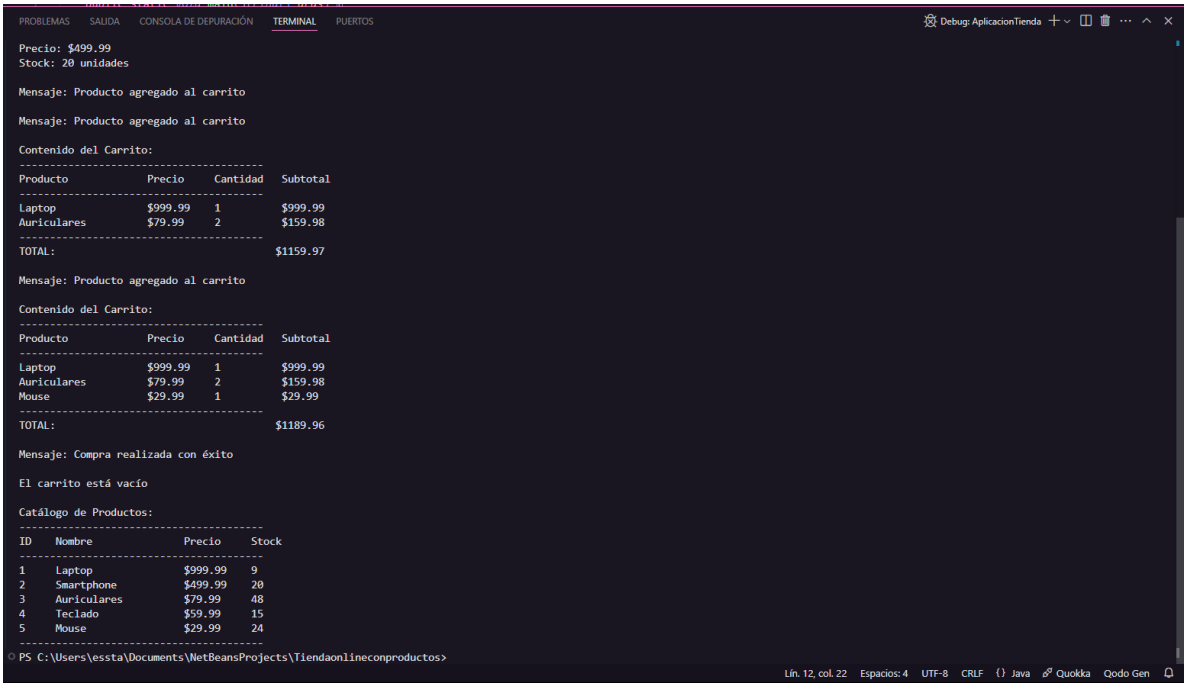
ruta 1:

Sistemadegestiondebiblioteca\src\main\java\com\mycompany\sistemadegestiondebiblioteca

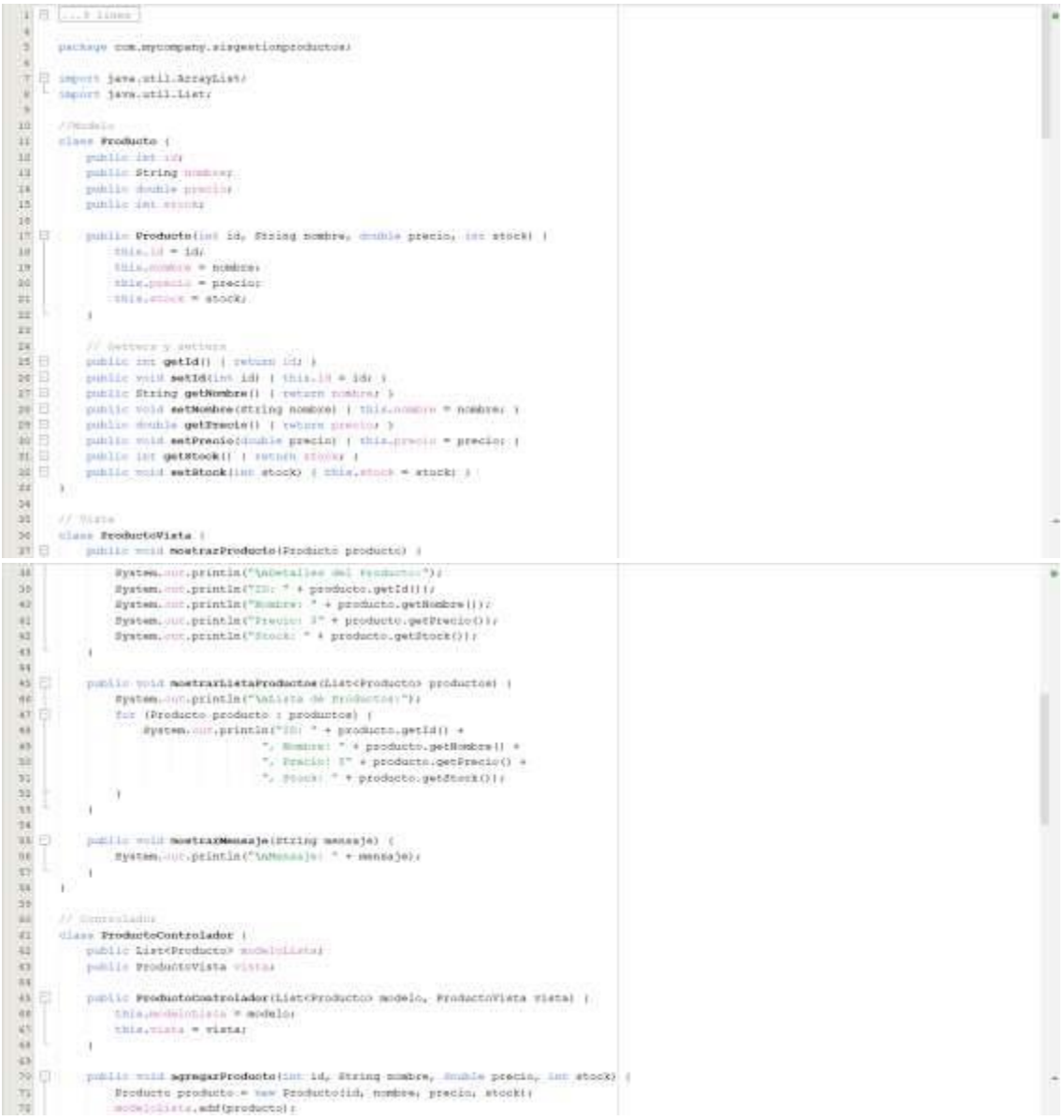


Tiendaonlineconproductos\src\main\java\com\mycompany\tiendaonlineconproductos

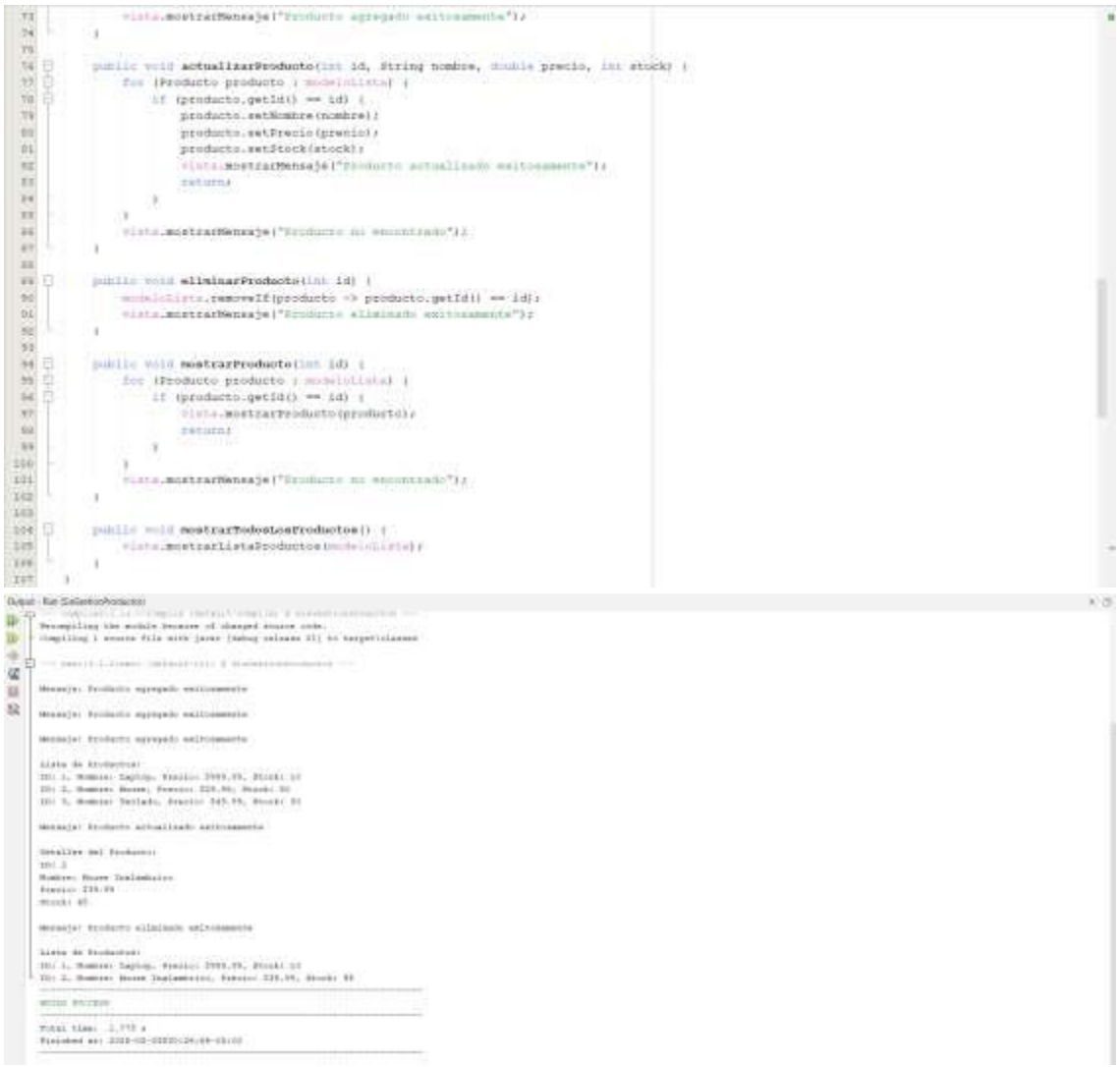
inedaonlineconproductos



Sistema Gestión de Productos



```
...9 lines

package com.mycompany.sistemagestionproductos;

import java.util.ArrayList;
import java.util.List;

//Modelo
class Producto {
    public int id;
    public String nombre;
    public double precio;
    public int stock;

    public Producto(int id, String nombre, double precio, int stock) {
        this.id = id;
        this.nombre = nombre;
        this.precio = precio;
        this.stock = stock;
    }

    // Getters y setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getNombre() { return nombre; }
    public void setNombre(String nombre) { this.nombre = nombre; }
    public double getPrecio() { return precio; }
    public void setPrecio(double precio) { this.precio = precio; }
    public int getStock() { return stock; }
    public void setStock(int stock) { this.stock = stock; }
}

// Vista
class ProductoVista {
    public void mostrarProducto(Producto producto) {
        System.out.println("\nDetalles del Producto:");
        System.out.println("ID: " + producto.getId());
        System.out.println("Nombre: " + producto.getNombre());
        System.out.println("Precio: $" + producto.getPrecio());
        System.out.println("Stock: " + producto.getStock());
    }

    public void mostrarListaProductos(List<Producto> productos) {
        System.out.println("\nLista de Productos:");
        for (Producto producto : productos) {
            System.out.println("ID: " + producto.getId() +
                               ", Nombre: " + producto.getNombre() +
                               ", Precio: $" + producto.getPrecio() +
                               ", Stock: " + producto.getStock());
        }
    }

    public void mostrarMensaje(String mensaje) {
        System.out.println("\nMensaje: " + mensaje);
    }
}

// Controlador
class ProductoControlador {
    public List<Producto> modeloLista;
    public ProductoVista vista;

    public ProductoControlador(List<Producto> modelo, ProductoVista vista) {
        this.modeloLista = modelo;
        this.vista = vista;
    }

    public void agregarProducto(int id, String nombre, double precio, int stock) {
        Producto producto = new Producto(id, nombre, precio, stock);
        modeloLista.add(producto);
```



Sistema Gestión de Estudiantes

```java
... 5 lines

package com.mycompany.sistemaestudiantes;

// Modelo
class Estudiante {
    public String nombre;
    public int id;
    public double promedio;

    public Estudiante(String nombre, int id, double promedio) {
        this.nombre = nombre;
        this.id = id;
        this.promedio = promedio;
    }

    public String getNombre() { return nombre; }
    public void setNombre(String nombre) { this.nombre = nombre; }
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public double getPromedio() { return promedio; }
    public void setPromedio(double promedio) { this.promedio = promedio; }
}

// Vista
class EstudianteVista {
    public void mostrarDetallesEstudiante(String nombre, int id, double promedio) {
        System.out.println("Estudiante:");
        System.out.println("Nombre: " + nombre);
        System.out.println("ID: " + id);
        System.out.println("Promedio: " + promedio);
    }
}

// Controlador
class EstudianteControlador {
    public Estudiante modelo;
    public EstudianteVista vista;

    public EstudianteControlador(Estudiante modelo, EstudianteVista vista) {
        this.modelo = modelo;
        this.vista = vista;
    }

    public void setEstudianteNombre(String nombre) {
        modelo.setNombre(nombre);
    }

    public String getEstudianteNombre() {
        return modelo.getNombre();
    }

    public void setEstudianteId(int id) {
        modelo.setId(id);
    }

    public int getEstudianteId() {
        return modelo.getId();
    }

    public void setEstudiantePromedio(double promedio) {
        modelo.setPromedio(promedio);
    }

    public double getEstudiantePromedio() {
        return modelo.getPromedio();
    }
```

```
    public void actualizarVista() {
        vista.mostrarDetallesEstudiante(modelo.getNombre(),
                                        modelo.getId(),
                                        modelo.getPromedio());
    }
}

public class DisGestionEstudiantes {

    public static void main(String[] args) {
        // Obtener datos del estudiante de la base de datos
        Estudiante modelo = new Estudiante("Juan Sanchez", 1, 9.5);

        // Crear la vista para mostrar los datos del estudiante
        EstudianteVista vista = new EstudianteVista();

        // Crear el controlador
        EstudianteController controlador = new EstudianteController(modelo, vista);

        // Actualizar vista
        controlador.actualizarVista();

        // Actualizar datos del modelo a través del controlador
        controlador.setEstudianteNombre("Juan Carlos Sanchez");
        controlador.setEstudiantePromedio(9.0);

        // Mostrar datos actualizados
        controlador.actualizarVista();

    }
}
```

```
Running fix projects...

---------------< com.mycompany:DisGestionEstudiantes >---------------
Building DisGestionEstudiantes 1.0-SNAPSHOT
  from pom.xml
--------------------------------[ jar ]---------------------------------

--- resources:3.3.1:resources (default-resources) @ DisGestionEstudiantes ---
skip non existing resourceDirectory C:\Users\ASUS\Desktop\Documentos\NetBeansProjects\DisGestionEstudiantes\src\main\resources

--- compiler:3.13.0:compile (default-compile) @ DisGestionEstudiantes ---
Nothing to compile - all classes are up to date.

--- exec:3.1.0:exec (default-cli) @ DisGestionEstudiantes ---
Estudiante:
Nombre: Juan Sanchez
ID: 1
Promedio: 9.5
Estudiante:
Nombre: Juan Carlos Sanchez
ID: 1
Promedio: 9.0
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.958 s
Finished at: 2025-02-25T21:31:15-05:00
------------------------------------------------------------------------
```