Universidad de las Fuerzas Armadas-ESPE

Asignatura de Programación Orientada a Objetos

SEGUNDO PARCIAL

TRABAJO PRÁCTICO

Nombres de los estudiantes: Juliette Estefanía Flores Tanicuchi.

Alejandra Mariela Sánchez Delgado

Enma Maritza Enríquez Romero

Luis Eduardo Erazo Vallejo

Fecha: 31-01-2025

TRABAJO PRÁCTICO

Tema: Sistema de Gestión de Inventario con MVC**.**

Introducción

Actualmente, la gestión eficiente de un inventario es primordial en los negocios que manejan productos físicos para un funcionamiento exitoso.

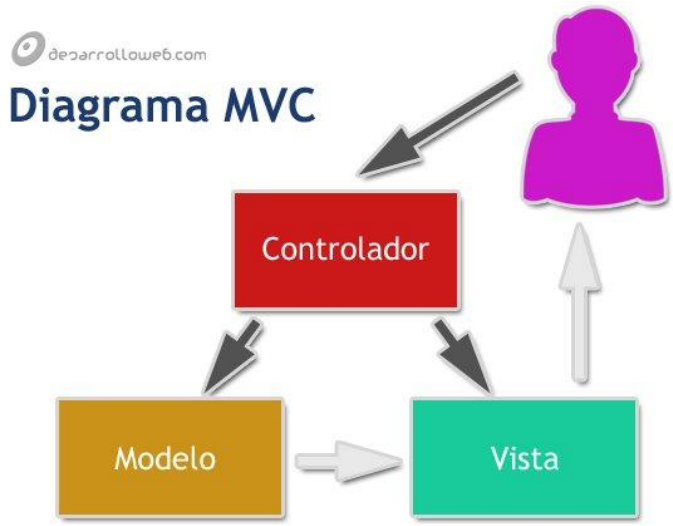
Un sistema de inventario permite controlar la disponibilidad, el valor y la cantidad de productos logrando una optimización de operaciones**.**

Objetivo General

- Desarrollar e implementar un sistema de gestión de inventario para una tienda utilizando el patrón de diseño Modelo-Vista-Controlador (MVC), permitiendo una administración eficiente y estructurada de los productos.

Marco Teórico

MVC es la abreviatura de Modelo, Vista y Controlador esto es una forma muy común para la organización de códigos en donde su función principal es que cada sección del código tenga un determinado propósito.



Un sistema de gestión de inventario con MVC (Modelo-Vista-Controlador) es una arquitectura de software que separa la lógica de negocio de la presentación y la interacción con el usuario.

- **Modelo (Model)**

El modelo representa la lógica de negocio y los datos del sistema. En un sistema de gestión de inventario, el modelo puede incluir:

- Clases para representar los productos, categorías, proveedores, etc.

- Métodos para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en los datos.

- Validaciones y reglas de negocio para asegurar la consistencia de los datos.

- **Vista (View)**

La vista es responsable de presentar la información al usuario. En un sistema de gestión de inventario, la vista puede incluir:

- Páginas web para mostrar la lista de productos, detalles de un producto, formulario para agregar/editar productos, etc.

- Plantillas para renderizar los datos del modelo en la página web.

- Elementos de la interfaz de usuario, como botones, formularios, tablas, etc.

- **Controlador (Controller)**

El controlador actúa como intermediario entre el modelo y la vista. En un sistema de gestión de inventario, el controlador puede incluir:

- Métodos para manejar las solicitudes HTTP (GET, POST, PUT, DELETE, etc.) y redirigir al usuario a la vista correspondiente.

- Lógica para validar y procesar los datos enviados por el usuario.

- Interacción con el modelo para realizar operaciones CRUD y obtener datos.

Beneficios del patrón MVC

El patrón MVC ofrece varios beneficios, como:

- **Separación de preocupaciones:** Cada componente tiene una responsabilidad específica, lo que facilita el mantenimiento y la escalabilidad del sistema.

- **Reutilización de código:** El modelo y el controlador pueden ser reutilizados en diferentes vistas y contextos.

- **Flexibilidad y personalización:** La vista puede ser personalizada y modificada sin afectar la lógica de negocio del modelo y el controlador.

Características

- **Separación de preocupaciones:** El patrón MVC separa las preocupaciones de la aplicación en tres componentes independientes.

- **Reutilización de código:** El patrón MVC permite reutilizar código en diferentes partes de la aplicación.

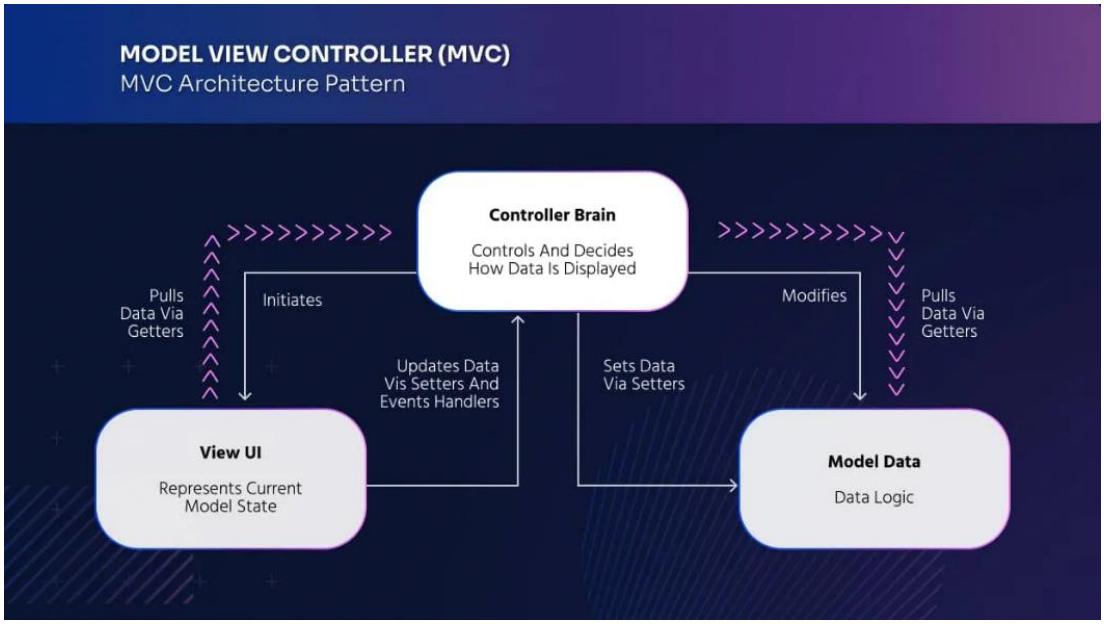
- **Flexibilidad y escalabilidad:** El patrón MVC permite cambiar o agregar componentes sin afectar la estructura general de la aplicación.

- **Independencia entre componentes:** Los componentes del patrón MVC (Modelo, Vista y Controlador) son independientes entre sí.

- **Comunicación entre componentes:** Los componentes del patrón MVC se comunican entre sí para realizar las operaciones necesarias.

- **Presentación de información:** El patrón MVC permite presentar información al usuario de manera clara y organizada.

- **Manejo de solicitudes:** El patrón MVC permite manejar solicitudes del usuario y realizar operaciones necesarias.



Patrón arquitectónico de un MVC

Dentro de las ventajas mas comunes de este patrón son:

- El aprovechamiento de ciertos componentes de la interfaz de usuario reutilizable
- Optimización del procesamiento de los datos, validación y conversación
- Permite el modularidad
- Sencillez para poder crear diferentes representaciones con los mismos datos

El patrón de diseño MVC es apto para:

- **Desarrolladores**

- **Desarrolladores web:** Es especialmente útil para crear aplicaciones web complejas.
- **Desarrolladores de software:** Es aplicable a la creación de software de escritorio y móvil.
- **Ingenieros de software:** Es útil para diseñar y desarrollar sistemas de software a gran escala.

- **Proyectos**

- **Aplicaciones web complejas:** Como plataformas de comercio electrónico, redes sociales, etc.

- **Sistemas de gestión:** Como sistemas de gestión de inventario, sistemas de gestión de proyectos, etc.
- **Aplicaciones móviles:** Como aplicaciones de banca móvil, aplicaciones de entrega de comida, etc.

Funcionamiento

- **Solicitud del usuario:** El usuario realiza una solicitud al sistema, como hacer clic en un botón o enviar un formulario.
- **Controlador:** El controlador recibe la solicitud y la procesa.
- **Modelo:** El controlador utiliza el modelo para realizar operaciones de negocio, como obtener datos de la base de datos.
- **Vista:** El controlador utiliza la vista para presentar los resultados al usuario.
- **Respuesta:** La vista envía la respuesta al usuario, que puede ser una página web, un mensaje de error, etc.

El patrón MVC es una arquitectura de software que separa las preocupaciones de la aplicación en tres componentes independientes, lo que facilita la mantenibilidad, escalabilidad y reutilización de código.

El patrón MVC aplicado a un sistema de gestión de inventario permite:

- Organización estructurada de la lógica de negocio, interfaz y control
- Mantenimiento eficiente del código
- Escalabilidad para añadir nuevas funcionalidades
- Testing independiente de cada componente

2. Componentes Principales

Modelo:

- Gestión de datos de productos

- Operaciones CRUD

- Reglas de negocio

- Validaciones

- Persistencia de datos

Vista:

- Interfaz de usuario

- Formularios de entrada

- Reportes

- Visualización de datos

- Mensajes al usuario

Controlador:

- Coordinación entre Modelo y Vista

- Procesamiento de solicitudes

- Gestión de flujo de datos

- Manejo de eventos

- Validaciones de entrada

- **Casos de Uso Típicos**

1. Gestión de Productos:

- Registro de nuevos productos

- Actualización de existencias

- Consulta de inventario

- Eliminación de productos

2. Control de Stock:

- Alertas de bajo inventario

- Registro de movimientos
- Control de pérdidas
- Gestión de ubicaciones

3. Reportes:

- Inventario actual
- Movimientos históricos
- Valoración del inventario
- Productos críticos

4. Consideraciones de Diseño

Seguridad:

- Autenticación de usuarios
- Autorización de operaciones
- Registro de auditoría
- Backup de datos

Usabilidad:

- Interfaz intuitiva
- Flujos de trabajo eficientes
- Mensajes claros
- Ayuda contextual

Rendimiento:

- Optimización de consultas
- Caché de datos
- Paginación de resultados
- Procesamiento asíncrono

- **Tecnologías y Herramientas Comunes**

Backend:

- Python (Django, Flask)

- Java (Spring)

- PHP (Laravel)

- .NET (ASP.NET MVC)

Frontend:

- HTML5/CSS3

- JavaScript

- Frameworks (React, Angular, Vue.js)

- Bootstrap/Material-UI

Base de Datos:

- MySQL

- PostgreSQL

- MongoDB

- SQLite

METODOLOGÍA

Para el desarrollo del sistema de gestión de inventario, se llevó a cabo un análisis detallado de los requerimientos funcionales y no funcionales. Los requerimientos funcionales incluyen la capacidad de agregar, modificar, eliminar y consultar productos en el inventario. Los requerimientos no funcionales abarcan la usabilidad, rendimiento y escalabilidad del sistema. Además, se identificaron los actores principales del sistema, como administradores y empleados de la tienda, quienes interactuarán con la aplicación.

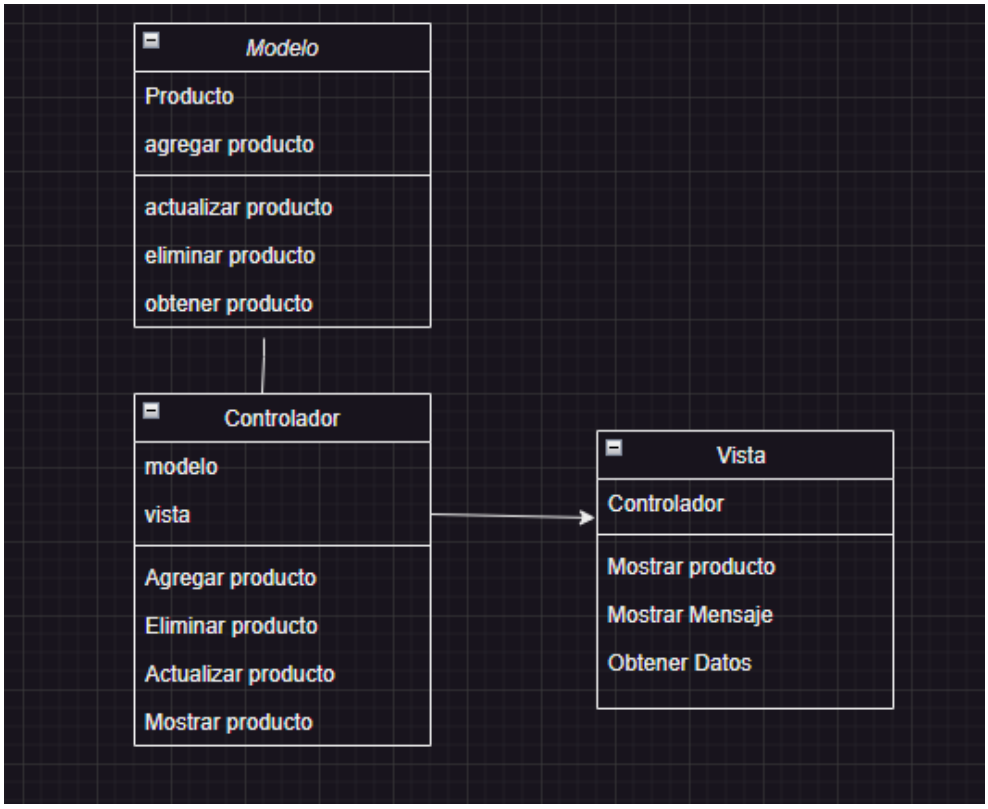
Diseño de la Arquitectura

El diseño del sistema se basa en el patrón MVC, permitiendo la separación de la lógica de negocio, la interfaz de usuario y la gestión de control. Se definieron los siguientes componentes clave:

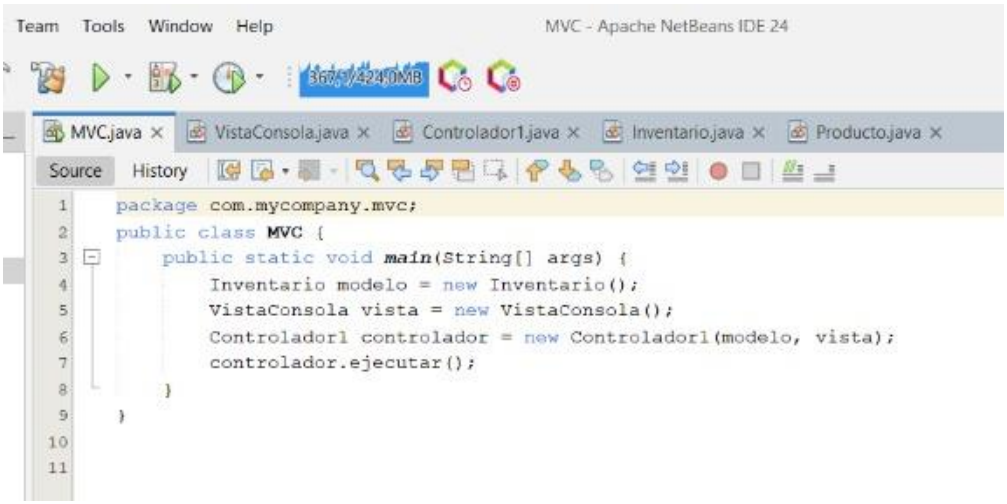
Modelo: Encargado del manejo de datos, consultas y actualizaciones en la base de datos.

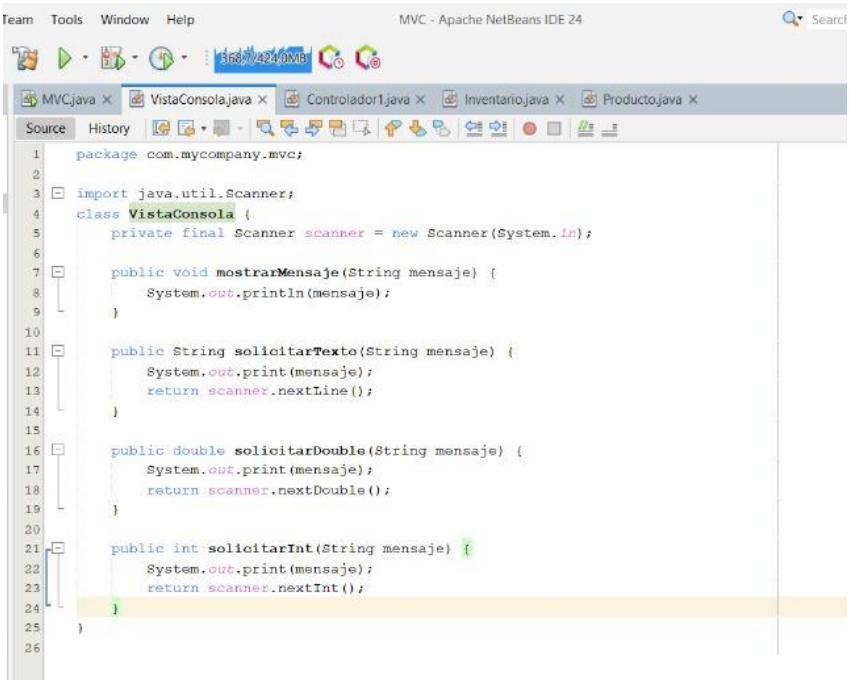
Vista: Responsable de la presentación de la información y la interacción con el usuario.

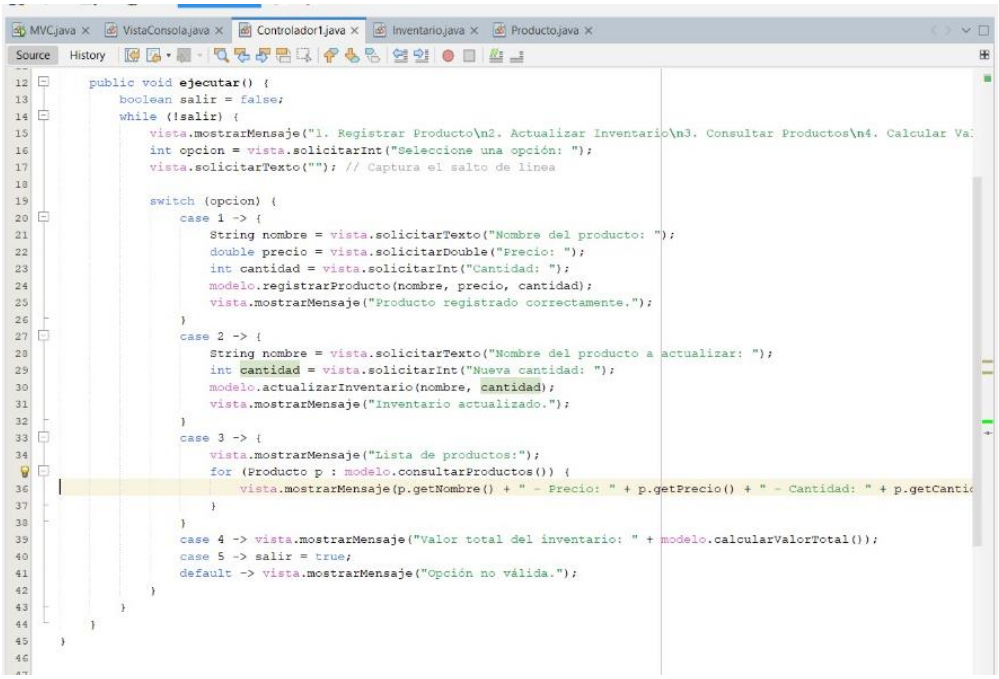
Controlador: Gestiona la lógica de aplicación, procesando las entradas del usuario y actualizando el modelo y la vista según corresponda.



EJEMPLO

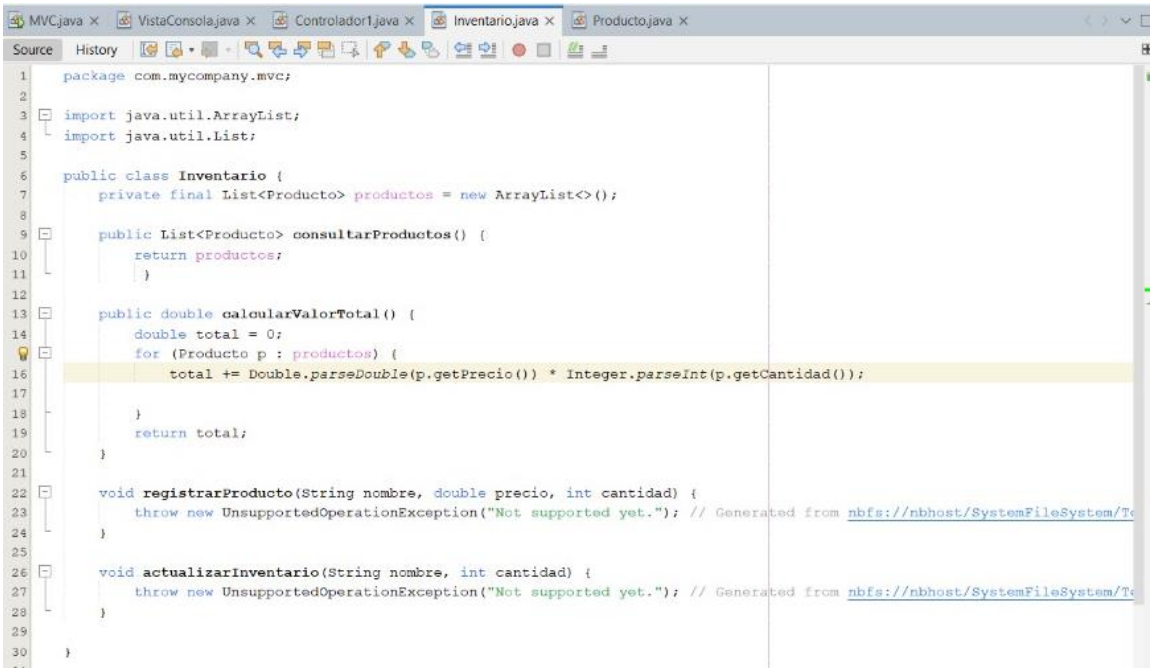

Team Tools Window Help
MVC - Apache NetBeans IDE 24
MVC.java
VistaConsola.java
Controlador1.java
Inventario.java
Producto.java
Source History
package com.mycompany.mvc;
public class MVC {
public static void main(String[] args) {
Inventario modelo = new Inventario();
VistaConsola vista = new VistaConsola();
Controlador1 controlador = new Controlador1(modelo, vista);
controlador.ejecutar();
}
}


Team Tools Window Help
MVC - Apache NetBeans IDE 24
MVC.java
VistaConsola.java
Controlador1.java
Inventario.java
Producto.java
Source History
package com.mycompany.mvc;
import java.util.Scanner;
class VistaConsola {
private final Scanner scanner = new Scanner(System.in);
public void mostrarMensaje(String mensaje) {
System.out.println(mensaje);
}
public String solicitarTexto(String mensaje) {
System.out.print(mensaje);
return scanner.nextLine();
}
public double solicitarDouble(String mensaje) {
System.out.print(mensaje);
return scanner.nextDouble();
}
public int solicitarInt(String mensaje) {
System.out.print(mensaje);
return scanner.nextInt();
}
}




```java
    public void ejecutar() {
        boolean salir = false;
        while (!salir) {
            vista.mostrarMensaje("1. Registrar Producto\n2. Actualizar Inventario\n3. Consultar Productos\n4. Calcular Va
            int opcion = vista.solicitarInt("Seleccione una opción: ");
            vista.solicitarTexto(""); // Captura el salto de linea

            switch (opcion) {
                case 1 -> {
                    String nombre = vista.solicitarTexto("Nombre del producto: ");
                    double precio = vista.solicitarDouble("Precio: ");
                    int cantidad = vista.solicitarInt("Cantidad: ");
                    modelo.registrarProducto(nombre, precio, cantidad);
                    vista.mostrarMensaje("Producto registrado correctamente.");
                }
                case 2 -> {
                    String nombre = vista.solicitarTexto("Nombre del producto a actualizar: ");
                    int cantidad = vista.solicitarInt("Nueva cantidad: ");
                    modelo.actualizarInventario(nombre, cantidad);
                    vista.mostrarMensaje("Inventario actualizado.");
                }
                case 3 -> {
                    vista.mostrarMensaje("Lista de productos:");
                    for (Producto p : modelo.consultarProductos()) {
                        vista.mostrarMensaje(p.getNombre() + " - Precio: " + p.getPrecio() + " - Cantidad: " + p.getCanti
                    }
                }
                case 4 -> vista.mostrarMensaje("Valor total del inventario: " + modelo.calcularValorTotal());
                case 5 -> salir = true;
                default -> vista.mostrarMensaje("Opción no válida.");
            }
        }
    }
}
```




```java
package com.mycompany.mvc;

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

public class Inventario {
    private final List<Producto> productos = new ArrayList<>();

    public List<Producto> consultarProductos() {
        return productos;
        }

    public double calcularValorTotal() {
        double total = 0;
        for (Producto p : productos) {
            total += Double.parseDouble(p.getPrecio()) * Integer.parseInt(p.getCantidad());

        }
        return total;
    }

    void registrarProducto(String nombre, double precio, int cantidad) {
        throw new UnsupportedOperationException("Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/T
    }

    void actualizarInventario(String nombre, int cantidad) {
        throw new UnsupportedOperationException("Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/T
    }

}
```

MVC.java | VistaConsola.java | Controlador1.java | Inventario.java | Producto.java

Source | History

```java
public class Producto {

    private final String nombre;
    private final double precio;
    private int cantidad;

    // Constructor con parámetros
    public Producto(String nombre, double precio, int cantidad) {
        this.nombre = nombre;
        this.precio = precio;
        this.cantidad = cantidad;
    }

    // Métodos Getters y Setters
    public String getNombre() { return nombre; }
    public double getPrecio() { return precio; }
    public int getCantidad() { return cantidad; }
    public void setCantidad(int cantidad) { this.cantidad = cantidad;
    }
}
```

Output - Run (MVC)

```
Changes detected - recompiling the module! :source
Compiling 7 source files with javac [debug release 23] to target\classes

--- exec:3.1.0:exec (default-cli) @ MVC ---
1. Registrar Producto
2. Actualizar Inventario
3. Consultar Productos
4. Calcular Valor Total
5. Salir
Seleccione una opci?n:
```

Run (MVC)

CONCLUSIÓN

- Este SGI con MVC en C es una implementación básica, pero se puede mejorar con:

Interfaz gráfica: GTK o Qt.

API REST: Para acceder desde aplicaciones web o móviles.

Notificaciones: Alertas por correo o SMS cuando el stock sea bajo.

Multiusuario: Control de accesos y roles de usuarios.

- La arquitectura MVC dentro del sistema de gestión de inventario permite una separación clara entre la lógica del negocio,la presentación y la interacción con el usuario facilitando la comprensión del sistema.

- La implementación del sistema basado en la arquitectura MVC asegura un diseño limpio y organizado ofreciendo ventajas en los equipos lo cual lo hace ideal para un proyecto a largo plazo y con potencial de crecimiento.

Bibliografía

- Aguilar, J. M., & JMAguilar. (s/f). *¿Qué es el patrón MVC en programación y por qué es útil?* campusMVP.es. Recuperado el 3 de febrero de 2025, de https://www.campusmvp.es/recursos/post/que-es-el-patron-mvc-en-programacion-y-por-que-es-util.aspx
- *MVC: Model, view, controller*. (s/f). Codecademy. https://www.codecademy.com/article/mvc
- Bustamante, E., & Nittala, P. (2024, agosto 20). *Understanding the MVC pattern in software design*. Oshyn.com. https://www.oshyn.com/blog/mvc-pattern-software-design

Repositorio de Github

https://github.com/Esteff593/AE-Juliette-Flores-Alejandra-Sanchez-Enma-Enriquez-Luis-Erazo