

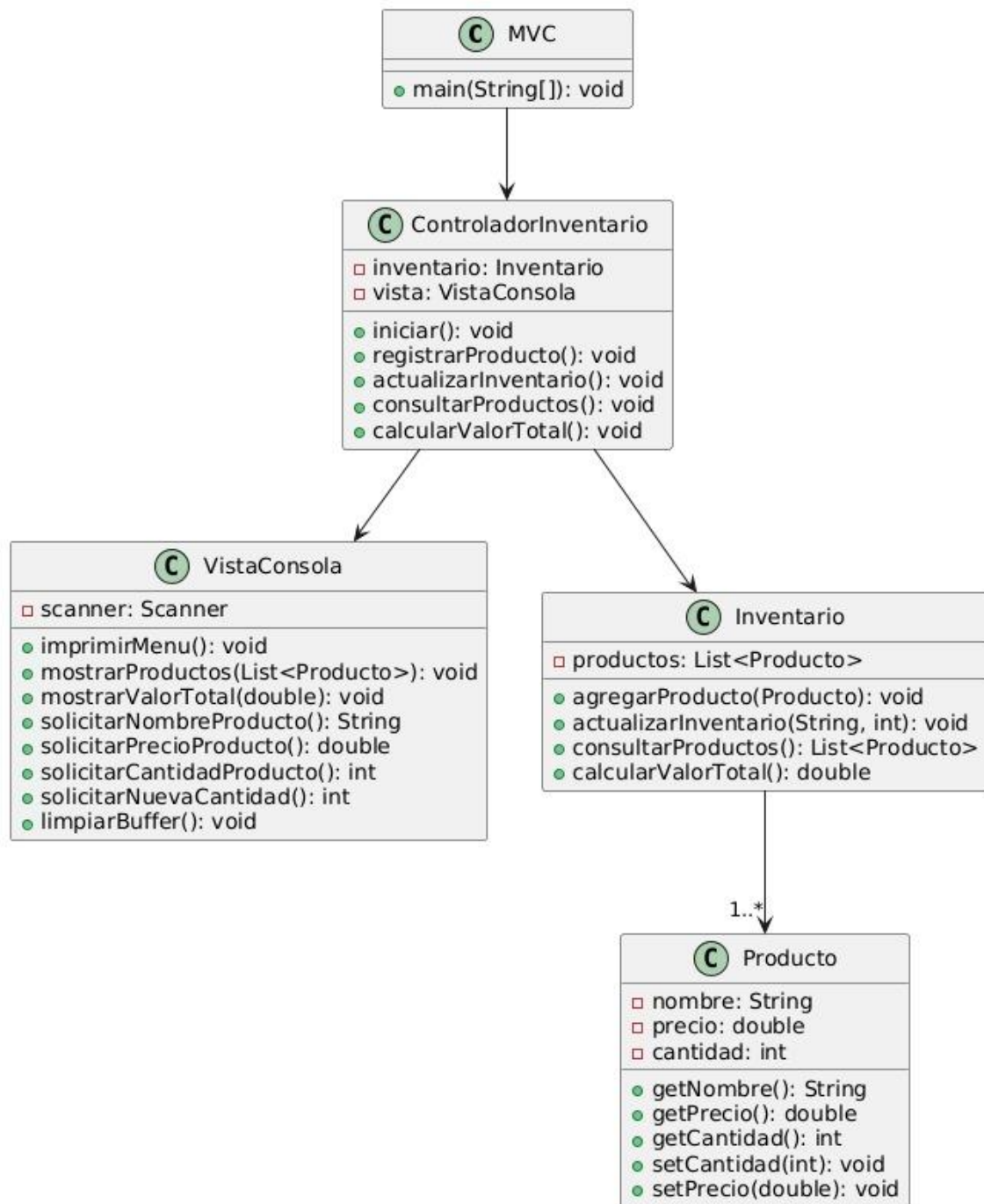
<b>INTEGRANTES:</b> Esteban Maila, Robinson Romero, Jefferson Tipantiza	<b>TEMA:</b> Sistema de Gestión de Inventario con MVC.
<b>NRC:</b> 1322	<b>FECHA:</b> 3/2/2025

## Informe de Desarrollo del Sistema de Gestión de Inventario utilizando MVC

### Objetivo de la Actividad

Desarrollar un sistema de gestión de inventario para una tienda, implementando el patrón de diseño Modelo-Vista-Controlador (MVC). El sistema debía permitir registrar productos, actualizar el inventario, consultar productos y calcular el valor total del inventario.

#### ➤ DIAGRAMA UML



## ➤ CÓDIGO

Enlace del repositorio de Git Hub:

<https://github.com/JEFFERSON712/ControladorInventario/tree/2e9bf4344cd4fb9d89cb0cf346f8624b94bd977b/src>

## ➤ Informe

## 1. Diseño del Sistema

Se definió el uso del patrón **Modelo-Vista-Controlador (MVC)** para organizar el código en tres componentes bien definidos:

- **Modelo (Model):** Encargado de la gestión de datos y la lógica de negocio.
  - **Vista (View):** Responsable de la interacción con el usuario, mostrando información y capturando entradas.
  - **Controlador (Controller):** Actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de datos y la lógica de control.
- 

## 2. Estructura del Código

El sistema se dividió en las siguientes clases, cada una cumpliendo una función específica:

### a) Modelo (Modelo.java)

Incluye las clases **Producto** e **Inventario**, responsables de la lógica de negocio y la manipulación de datos:

- **Clase Producto:** Define las propiedades de un producto (nombre, precio, cantidad) con sus respectivos métodos de acceso (getters y setters).
- **Clase Inventario:** Gestiona una lista de productos, permitiendo agregar productos, actualizar cantidades, consultar la lista y calcular el valor total del inventario.

```
J Modelo[1].java X
C: > Users > jeffe > AppData > Local > Microsoft > Windows > INetCache > IE > 84ZT2TXO > J Modelo[1].java > ...
1  import java.util.ArrayList;
2  import java.util.List;
3
4  class Producto {
5      private String nombre;
6      private double precio;
7      private int cantidad;
8
9      public Producto(String nombre, double precio, int cantidad) {
10         this.nombre = nombre;
11         this.precio = precio;
12         this.cantidad = cantidad;
13     }
14
15     public String getNombre() {
16         return nombre;
17     }
18
19     public double getPrecio() {
20         return precio;
21     }
22
23     public int getCantidad() {
24         return cantidad;
25     }
26
27     public void setCantidad(int cantidad) {
28         this.cantidad = cantidad;
29     }
30 }
31
32 class Inventario {
33     private List<Producto> productos;
```

## b) Vista (VistaConsola.java)

Se encargó de la interacción con el usuario mediante la consola:

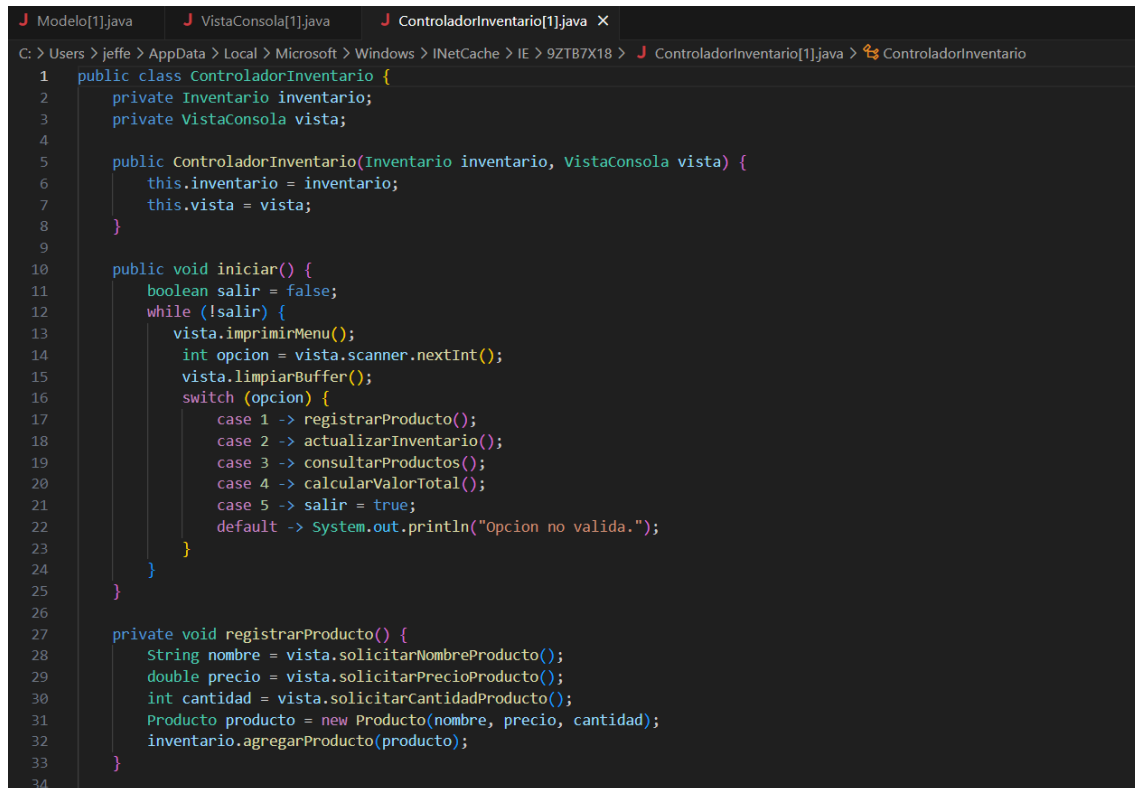
- Mostraba un menú de opciones.
- Solicitaba información al usuario para registrar o actualizar productos.
- Mostraba la lista de productos y el valor total del inventario.

```
J Modelo[1].java J VistaConsola[1].java X
C: > Users > jeffe > AppData > Local > Microsoft > Windows > INetCache > IE > UIK49K1M > J VistaConsola[1].java > ...
1  import java.util.List;
2  import java.util.Scanner;
3
4  public class VistaConsola {
5      public Scanner scanner;
6
7      public VistaConsola() {
8          scanner = new Scanner(System.in);
9      }
10
11     public void imprimirMenu(){
12         System.out.println("1. Registrar Producto");
13         System.out.println("2. Actualizar Inventario");
14         System.out.println("3. Consultar Productos");
15         System.out.println("4. Calcular Valor Total");
16         System.out.println("5. Salir");
17         System.out.print("Seleccione una opcion: ");
18     }
19     public void mostrarProductos(List<Producto> productos) {
20         System.out.println("Lista de productos:");
21         for (Producto producto : productos) {
22             System.out.println("Nombre: " + producto.getNombre() + ", Precio: " + producto.getPrecio() + ", Cantidad: " + producto.getCantidad());
23         }
24     }
25
26     public void mostrarValorTotal(double valorTotal) {
27         System.out.println("Valor total del inventario: " + valorTotal);
28     }
29
30     public String solicitarNombreProducto() {
31         System.out.print("Ingrese el nombre del producto: ");
32         return scanner.nextLine();
33     }
34 }
```

### c) Controlador (ControladorInventario.java)

Gestionó la lógica de control, capturando las acciones del usuario y llamando a los métodos correspondientes del Modelo y la Vista:

- Controlaba el flujo del menú y las operaciones del sistema.
- Implementó métodos para registrar productos, actualizar el inventario, consultar productos y calcular el valor total.



```
1 public class ControladorInventario {
2     private Inventario inventario;
3     private VistaConsola vista;
4
5     public ControladorInventario(Inventario inventario, VistaConsola vista) {
6         this.inventario = inventario;
7         this.vista = vista;
8     }
9
10    public void iniciar() {
11        boolean salir = false;
12        while (!salir) {
13            vista.imprimirMenu();
14            int opcion = vista.scanner.nextInt();
15            vista.limpiarBuffer();
16            switch (opcion) {
17                case 1 -> registrarProducto();
18                case 2 -> actualizarInventario();
19                case 3 -> consultarProductos();
20                case 4 -> calcularValorTotal();
21                case 5 -> salir = true;
22                default -> System.out.println("Opcion no valida.");
23            }
24        }
25    }
26
27    private void registrarProducto() {
28        String nombre = vista.solicitarNombreProducto();
29        double precio = vista.solicitarPrecioProducto();
30        int cantidad = vista.solicitarCantidadProducto();
31        Producto producto = new Producto(nombre, precio, cantidad);
32        inventario.agregarProducto(producto);
33    }
34}
```

### 3. Implementación del Código

Se siguieron los siguientes pasos para el desarrollo del sistema:

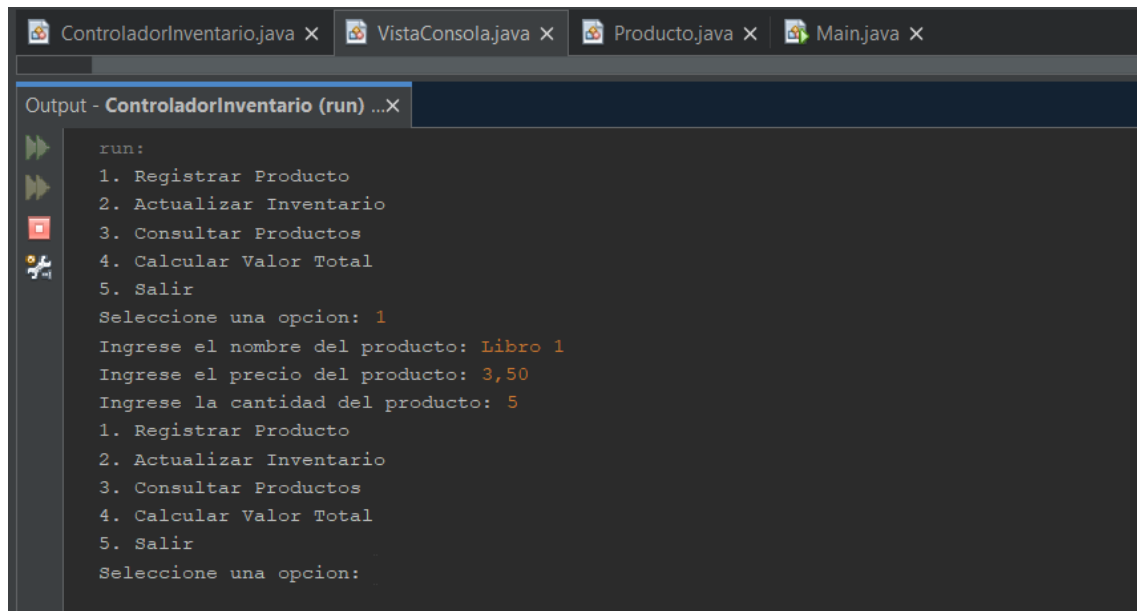
1. **Creación del Modelo:** Se definieron las clases Producto e Inventario con sus métodos necesarios para la gestión de productos.
2. **Desarrollo de la Vista:** Se implementó la clase VistaConsola para mostrar mensajes en la consola y capturar la entrada del usuario.
3. **Implementación del Controlador:** Se desarrolló ControladorInventario, que gestionaba la interacción entre el Modelo y la Vista.
4. **Integración del sistema:** La clase principal MVC.java inicializaba el sistema e invocaba el controlador para iniciar la ejecución.

### 5. Pruebas Realizadas

```
ControladorInventario.java × VistaConsola.java × Producto.java × Main.java ×  
Output - ControladorInventario (run) ...×  
run:  
1. Registrar Producto  
2. Actualizar Inventario  
3. Consultar Productos  
4. Calcular Valor Total  
5. Salir  
Seleccione una opcion:
```

```
ControladorInventario.java × VistaConsola.java × Producto.java × Main.java ×  
Output - ControladorInventario (run) ...×  
run:  
1. Registrar Producto  
2. Actualizar Inventario  
3. Consultar Productos  
4. Calcular Valor Total  
5. Salir  
Seleccione una opcion: 1  
Ingrese el nombre del producto: ||
```

```
ControladorInventario.java × VistaConsola.java × Producto.java × Main.java ×  
Output - ControladorInventario (run) ...×  
run:  
1. Registrar Producto  
2. Actualizar Inventario  
3. Consultar Productos  
4. Calcular Valor Total  
5. Salir  
Seleccione una opcion: 1  
Ingrese el nombre del producto: Libro 1  
Ingrese el precio del producto:
```



The screenshot shows an IDE window with four tabs: `ControladorInventario.java`, `VistaConsola.java`, `Producto.java`, and `Main.java`. The `Output - ControladorInventario (run) ...X` tab is active, displaying the following text:

```
run:
1. Registrar Producto
2. Actualizar Inventario
3. Consultar Productos
4. Calcular Valor Total
5. Salir
Seleccione una opcion: 1
Ingrese el nombre del producto: Libro 1
Ingrese el precio del producto: 3,50
Ingrese la cantidad del producto: 5
1. Registrar Producto
2. Actualizar Inventario
3. Consultar Productos
4. Calcular Valor Total
5. Salir
Seleccione una opcion:
```

```
Output - ControladorInventario (run) ...X
3. Consultar Productos
4. Calcular Valor Total
5. Salir
Seleccione una opcion: 1
Ingrese el nombre del producto: Libro 1
Ingrese el precio del producto: 3,50
Ingrese la cantidad del producto: 5
1. Registrar Producto
2. Actualizar Inventario
3. Consultar Productos
4. Calcular Valor Total
5. Salir
Seleccione una opcion: 1
Ingrese el nombre del producto: Cuaderno 1
Ingrese el precio del producto: 1,50
Ingrese la cantidad del producto: 2
1. Registrar Producto
2. Actualizar Inventario
3. Consultar Productos
4. Calcular Valor Total
5. Salir
Seleccione una opcion: 3
Lista de productos:
Nombre: Libro 1, Precio: 3.5, Cantidad: 5
Nombre: Cuaderno 1 , Precio: 1.5, Cantidad: 2
1. Registrar Producto
2. Actualizar Inventario
3. Consultar Productos
4. Calcular Valor Total
5. Salir
Seleccione una opcion: 4
Valor total del inventario: 20.5
1. Registrar Producto
2. Actualizar Inventario
3. Consultar Productos
4. Calcular Valor Total
5. Salir
Seleccione una opcion:
```

Se realizaron diversas pruebas para validar el funcionamiento correcto del sistema:



- **Registro de productos:** Se comprobó que los productos se agregaran correctamente al inventario.
  - **Actualización del inventario:** Se verificó la modificación de cantidades de productos existentes.
  - **Consulta de productos:** Se validó que la lista de productos se mostrara adecuadamente.
  - **Cálculo del valor total:** Se comprobó la correcta suma de los valores de los productos en el inventario.
- 

## 5. Retos y Soluciones

- **Problema con el buffer del Scanner:** Durante las pruebas, se detectó que el Scanner no capturaba correctamente algunas entradas después de números. Se resolvió utilizando el método limpiarBuffer() en la clase VistaConsola.
  - **Validación de entradas:** Se mejoró la robustez del sistema asegurando que las entradas del usuario fueran capturadas y convertidas correctamente.
- 

## 6. Conclusión

El sistema desarrollado cumple con los requisitos propuestos, utilizando el patrón MVC para una organización clara y estructurada del código. La separación de responsabilidades permitió una implementación limpia y fácil de mantener.