**UNIVERSIDAD DE LAS FUERZAS ARMADAS (ESPE)**

**SEGUNDO SEMESTRE**

**CARRERAS TECNICAS**

**DEPARTAMENTO DE CIENCIAS EXACTAS**

**SEGUNDO PARCIAL**

**“SISTEMA DE GESTION DE INVENTARIO CON MVC”**

**AUTORES:**

• Guerrero Steven. • Sislema Gabriel. • Shaffik Samir.

**NRC:**

1322

**DOCENTE:**

Luis Enrique Jaramillo Monaño

# SANGOLQUI – ECUADOR

**Introducción**

En la actualidad, la gestión eficiente del inventario es un aspecto clave para el éxito de cualquier negocio. Contar con un sistema que permita registrar, actualizar y consultar productos de manera estructurada facilita la administración de los recursos y optimiza los procesos internos.

En esta actividad experimental, se desarrollará un Sistema de Gestión de Inventario utilizando el patrón Modelo-Vista-Controlador (MVC), el cual permite separar la lógica del negocio, la presentación de los datos y la interacción con el usuario. La implementación de este sistema incluirá:

* Un Modelo, encargado de manejar los datos del inventario y las operaciones sobre los productos.
* Una Vista, responsable de presentar la información al usuario y recibir sus solicitudes.
* Un Controlador, que coordina la comunicación entre el modelo y la vista, garantizando una interacción fluida.

A través de esta actividad, se fortalecerán los conocimientos en arquitectura de software y programación estructurada, aplicando principios de diseño que mejoran la organización y el mantenimiento del código.

1. **OBJETIVOS** 
   1. **OBJETIVO GENERAL**

Desarrollar la capa modelo del sistema, implementando clases como Producto e Inventario con métodos que permitan registrar, actualizar, consultar productos y calcular el valor total del inventario.

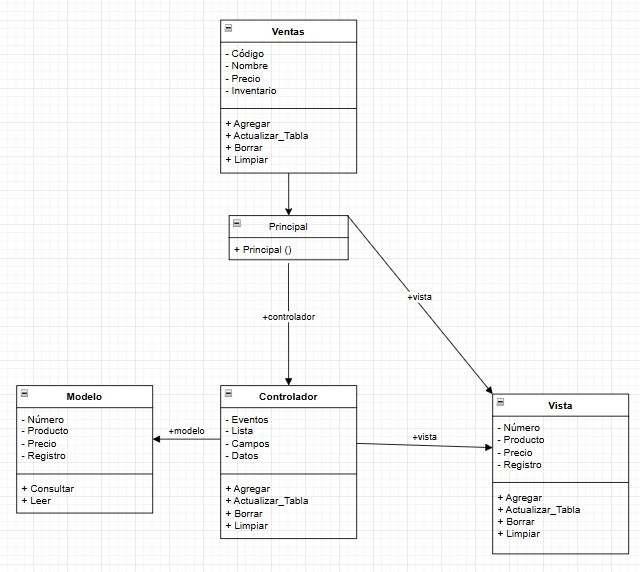
* 1. **OBJETIVOS ESPECÍFICO**

Elaborar diagramas UML que representen la estructura del sistema, incluyendo el diseño de clases, interacciones entre ellas y el flujo de datos bajo el patrón MVC.

Optimizar la lógica del sistema de inventario, garantizando que las operaciones de registro, consulta y actualización sean rápidas, precisas y escalables.

1. **MARCO TEÓRICO**

Diagrama UML



**Ilustración 1**

***UML***

**Descripción de las clases**

1. **Ventas**
   * + Atributos: Código, Nombre, Precio, Inventario.
     + Métodos: Agregar, Actualizar Tabla, Borrar, Limpiar.
     + Representa la entidad de ventas que maneja la información de los productos.
2. **Principal**
   * + Método: Principal().
     + Actúa como el punto de entrada del sistema, inicializando la aplicación.
3. **Modelo**
   * + Atributos: Número, Producto, Precio, Registro.
     + Métodos: Consultar, Leer.
     + Representa la capa de datos del sistema, encargada de la gestión de la información.
4. **Controlador**

Atributos: Eventos, Lista, Campos, Datos.

* + - Métodos: Agregar, Actualizar Tabla, Borrar, Limpiar.
    - Actúa como intermediario entre la Vista y el Modelo, manejando eventos y lógica de negocio.

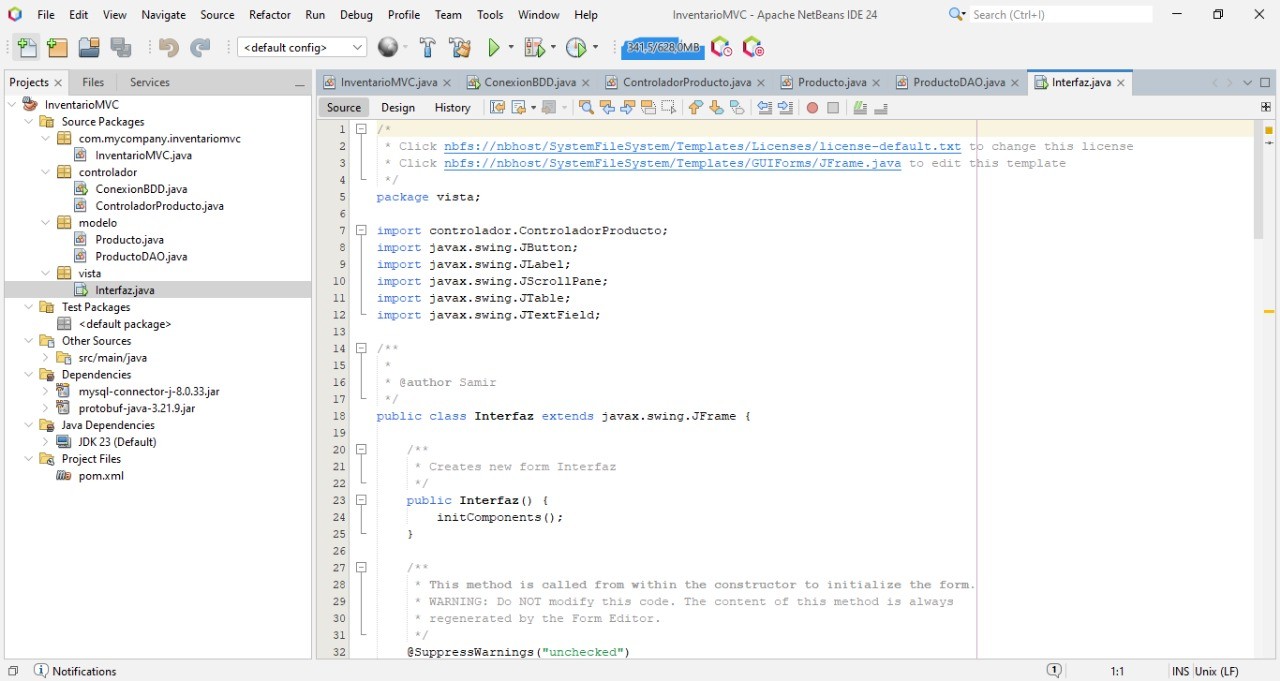
1. **Vista**
   * Atributos: Número, Producto, Precio, Registro.
   * Métodos: Agregar, Actualizar Tabla, Borrar, Limpiar.
   * Representa la interfaz gráfica o la capa de presentación del sistema.

**Relaciones entre las clases**

* + Principal se conecta con Controlador y Vista, indicando que inicializa la aplicación y gestiona la interacción.
  + Controlador interactúa con Modelo (gestión de datos) y Vista (actualización de la interfaz).
  + Ventas es una clase independiente que maneja las operaciones sobre los productos.

A continuación vamos a describir el código correspondiente.

**Código**



*Ilustración 2*

1. **controlador**: Maneja la lógica del negocio, incluyendo la clase

ControladorProducto.java.

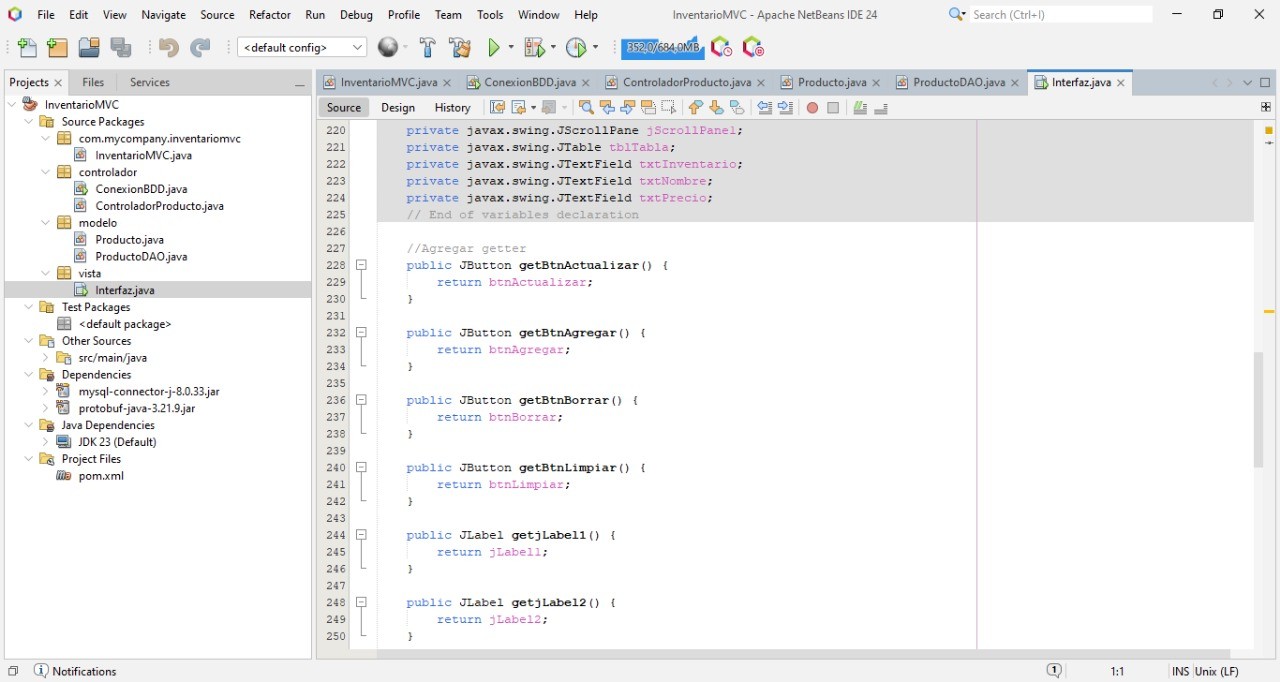
1. **modelo**: Contiene las clases relacionadas con los datos, como Producto.java y

ProductoDAO.java.

1. **vista**: Contiene la interfaz gráfica de usuario (Interfaz.java), que extiende JFrame y utiliza componentes de Swing (JButton, JLabel, JTable, etc.).

**Código Visible (Interfaz.java)**

* Define la clase Interfaz, que representa la GUI.
* Importa el controlador ControladorProducto.
* Usa componentes de Swing para construir la interfaz.
* Tiene un constructor Interfaz() que llama al método initComponents(), el cual es generado automáticamente por NetBeans.



*Ilustración 3*

**Actualizar (getBtnActualizar())**

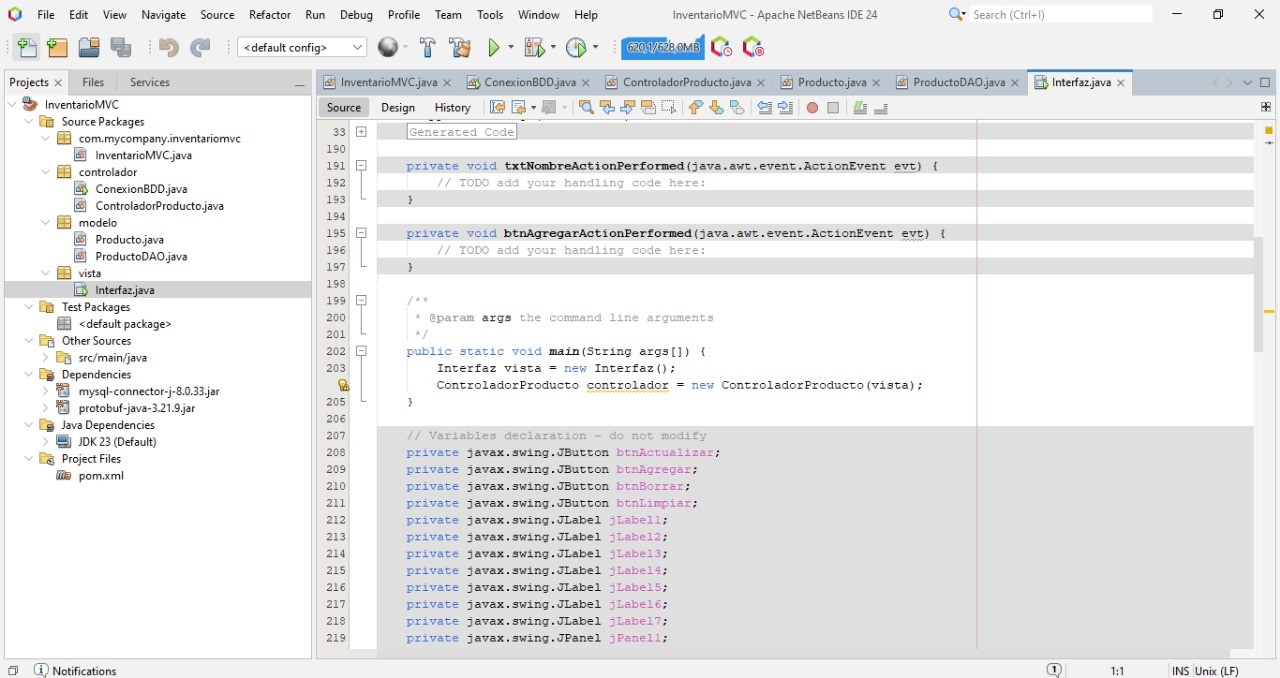
* Se encarga de modificar la información de un producto en el inventario.
* Puede actualizar datos como el nombre, la cantidad disponible o el precio del producto.
* Es probable que llame a una función en el Controlador para modificar un registro en la base de datos.

**Agregar (getBtnAgregar())**

* Permite insertar un nuevo producto en el inventario.
* Podría capturar datos desde los campos de texto (txtInventario, txtNombre, txtPrecio) y enviarlos al modelo, que luego los guarda en la base de datos.

**Borrar (getBtnBorrar())**

* Elimina un producto del inventario.
* Es posible que use el ID del producto para encontrarlo y eliminarlo de la base de datos.



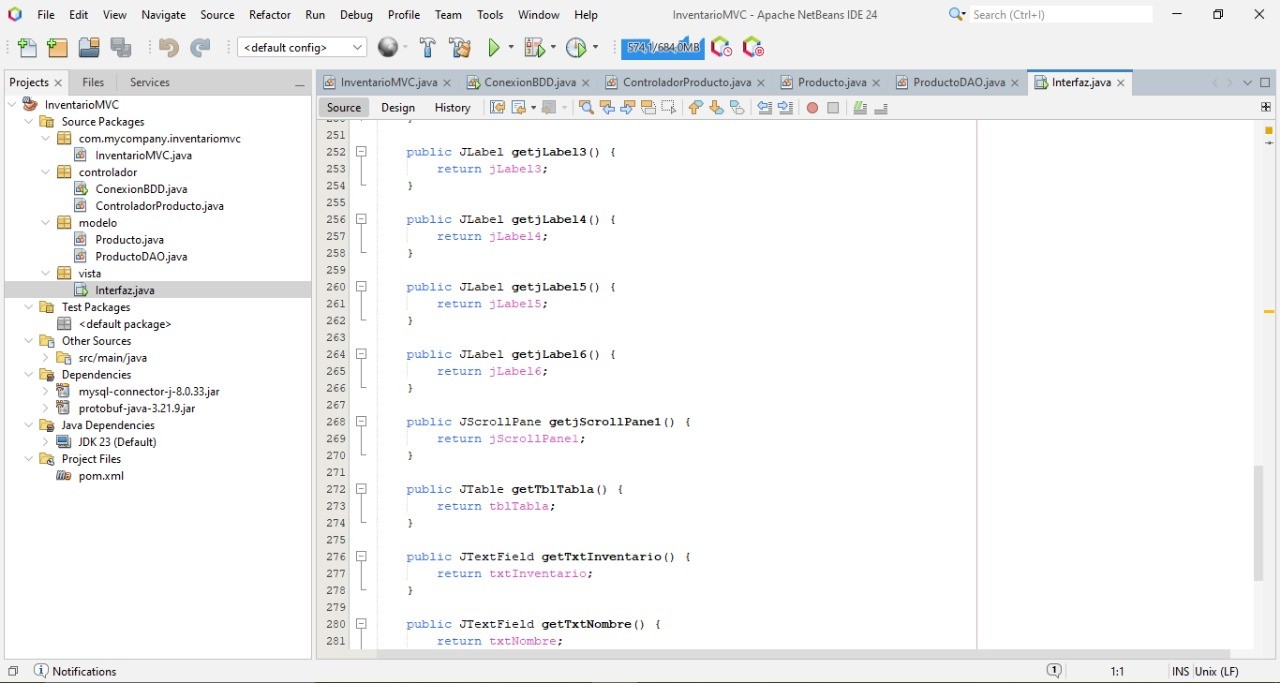
*Ilustración 4*

**Limpiar (getBtnLimpiar())**

* Borra el contenido de los campos de texto en la interfaz gráfica.
* Ayuda a los usuarios a ingresar nuevos productos sin necesidad de borrar manualmente la información anterior.

**Etiquetas (getJLabel1() y getJLabel2())**

* Se usan para mostrar información en la interfaz.
* Pueden representar títulos, descripciones o mensajes como "Inventario de productos"
* "Ingrese los datos del producto".



*Ilustración 5*

**Etiquetas (JLabel)**

* **jLabel3 - jLabel6** : Son etiquetas utilizadas para mostrar texto en la interfaz. Pueden representar campos como "Nombre", "Precio", "Cantidad", etc.

**Panel de desplazamiento (JScrollPane)**

* **jScrollPanel** : Es un contenedor que permite agregar una barra de desplazamiento a la tabla si los datos superan el tamaño visible.

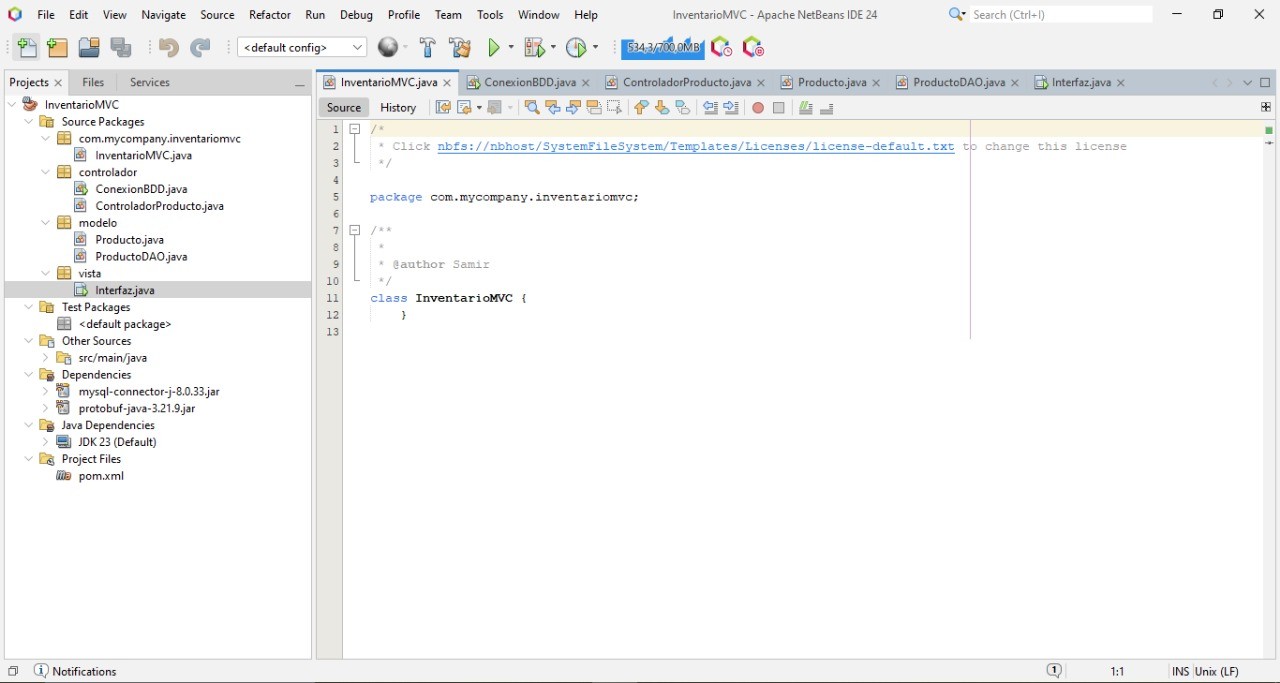
**Tabla (JTable)**

* **tblTabla** : Representa la tabla donde se muestran los productos del inventario.

Contiene columnas como nombre, precio y cantidad.

**Campos de texto (JTextField)**

* **txtInventario**: Campo donde el usuario ingresa o muestra el número de inventario de un producto.
* **txtNombre** : Campo donde se introduce o muestra el nombre del producto



*Ilustración 6*

**Estructura del Proyecto**

**Paquete com.mycompany.inventariomvc**

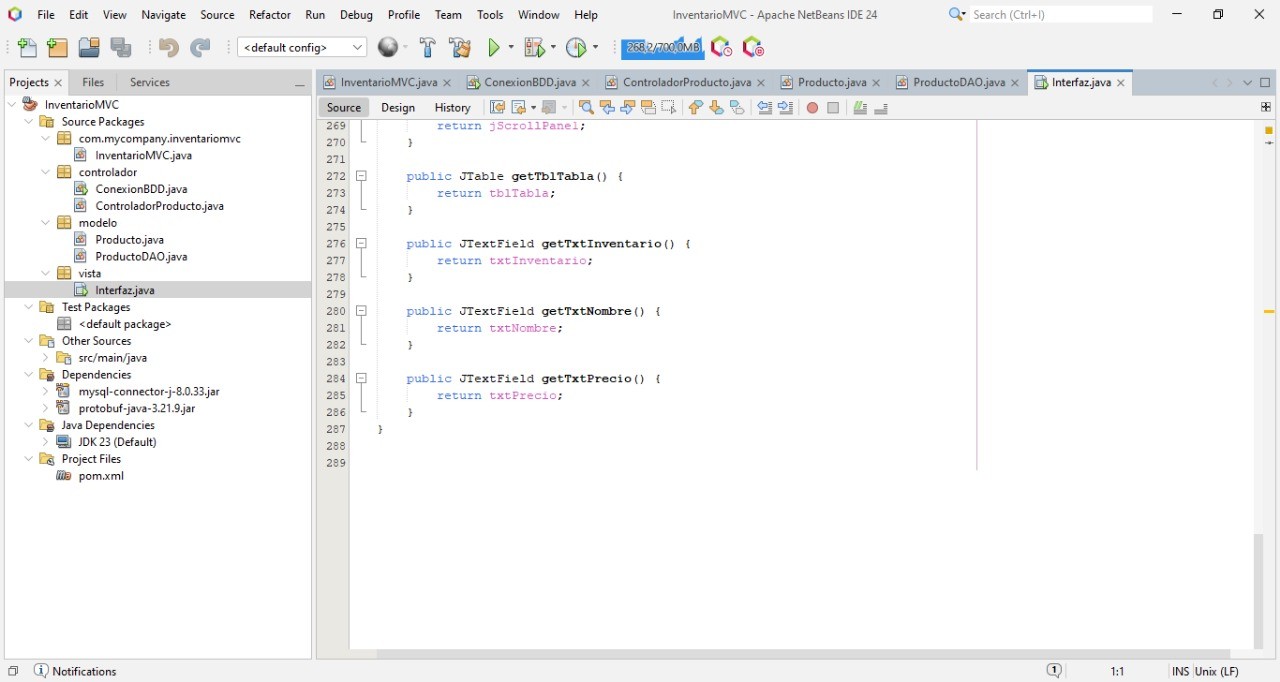
* **Controlador** (controlador): Contiene clases como ControladorProducto.java y ConexionBDD.java, encargadas de gestionar la lógica de negocio y la conexión a la base de datos.
* **Modelo** (modelo): Contiene clases como Producto.java (probablemente representa un objeto Producto) y ProductoDAO.java (posiblemente maneja la interacción con la base de datos).
* **Vista** (vista): Contiene Interfaz.java, que podría ser la clase responsable de la interfaz gráfica o consola para interactuar con el usuario.

**Archivo Abierto: InventarioMVC.java**

* Está dentro del paquete com.mycompany.inventariomvc.
* Define una clase llamada InventarioMVC.
* Actualmente, la clase está vacía.
* Incluye un comentario con el autor (@autor Samir).

**Dependencias**

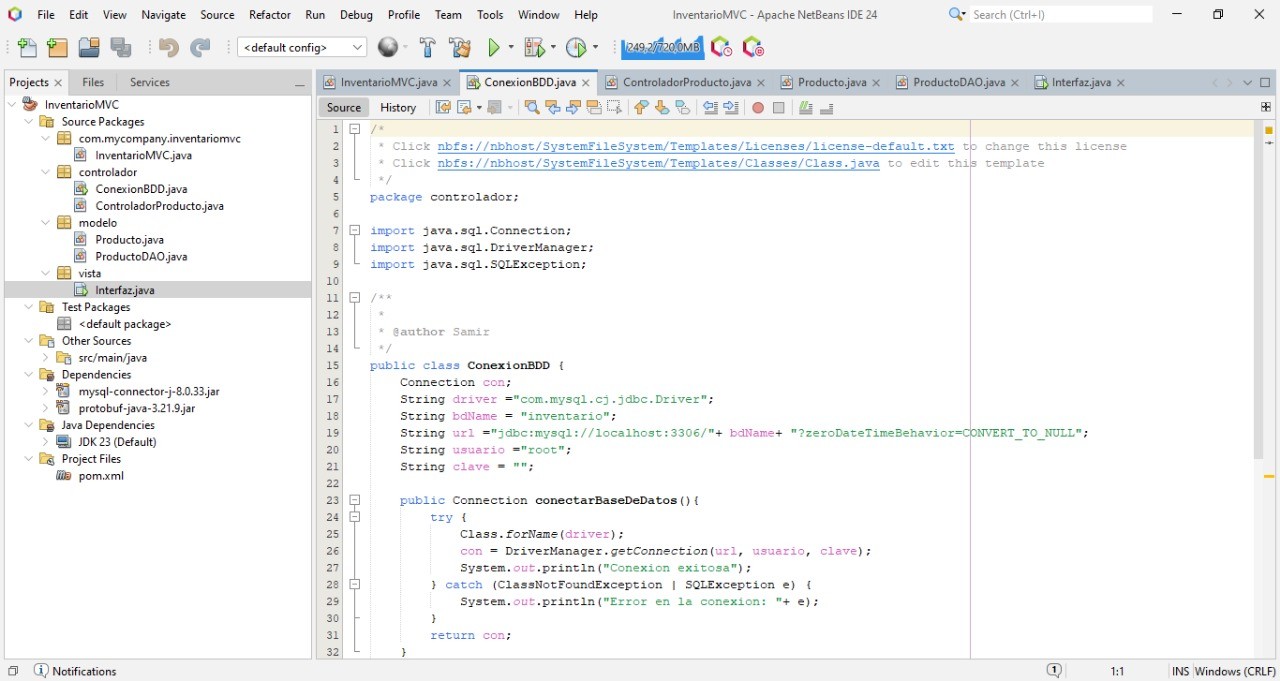
* **MySQL Connector (mysql-connector-j-8.0.33.jar)**: Indica que la aplicación usa MySQL como base de datos.
* **Protocol Buffers (protobuf-java-3.21.9.jar)**: Se usa para la serialización de datos eficiente.



*Ilustración 7*

**Métodos**

* **getTblTabla():** Devuelve un objeto JTable, muestra los productos en una tabla dentro de la interfaz.
* **getTxtInventario():** Devuelve un JTextField llamado txtInventario, que parece ser un campo de texto donde se introduce la cantidad de inventario de un producto.
* **getTxtNombre():** Devuelve un JTextField llamado txtNombre, utilizado para ingresar o mostrar el nombre del producto.
* **getTxtPrecio():** Devuelve un JTextField llamado txtPrecio.



*Ilustración 8*

**Connection con;**

* Es un objeto de tipo Connection de la librería JDBC.
* Sirve para establecer y manejar la conexión con la base de datos.
* Cuando se usa DriverManager.getConnection(), este objeto almacena la conexión activa.

**String driver = "com.mysql.cj.jdbc.Driver";**

* Especifica el driver JDBC de MySQL necesario para conectar la base de datos con

Java.

* "com.mysql.cj.jdbc.Driver" es el controlador de MySQL actualizado (el prefijo cj indica que es compatible con versiones modernas de MySQL).
* Importante: Sin este driver, Java no podría comunicarse con MySQL.

**String bdName = "inventario";**

* Contiene el nombre de la base de datos a la que se quiere conectar.
* En este caso, se llama "inventario", lo que significa que la base de datos debe existir en el servidor MySQL.

**String url = "jdbc:mysql://localhost:3306/" + bdName +**

**"?zeroDateTimeBehavior=CONVERT\_TO\_NULL";**

* Representa la URL de conexión a la base de datos. Se desglosa así:
* **"jdbc:mysql://":**Especifica que se usará JDBC con un servidor MySQL.
* **"localhost":** Indica que la base de datos está alojada en la misma máquina donde corre el programa.
* **":3306":** Es el puerto predeterminado de MySQL (puede cambiar si MySQL está configurado en otro puerto).
* **"/" + bdName:** Se concatena el nombre de la base de datos ("inventario") para conectarse a ella.
* **"?zeroDateTimeBehavior=CONVERT\_TO\_NULL":** Evita problemas con valores

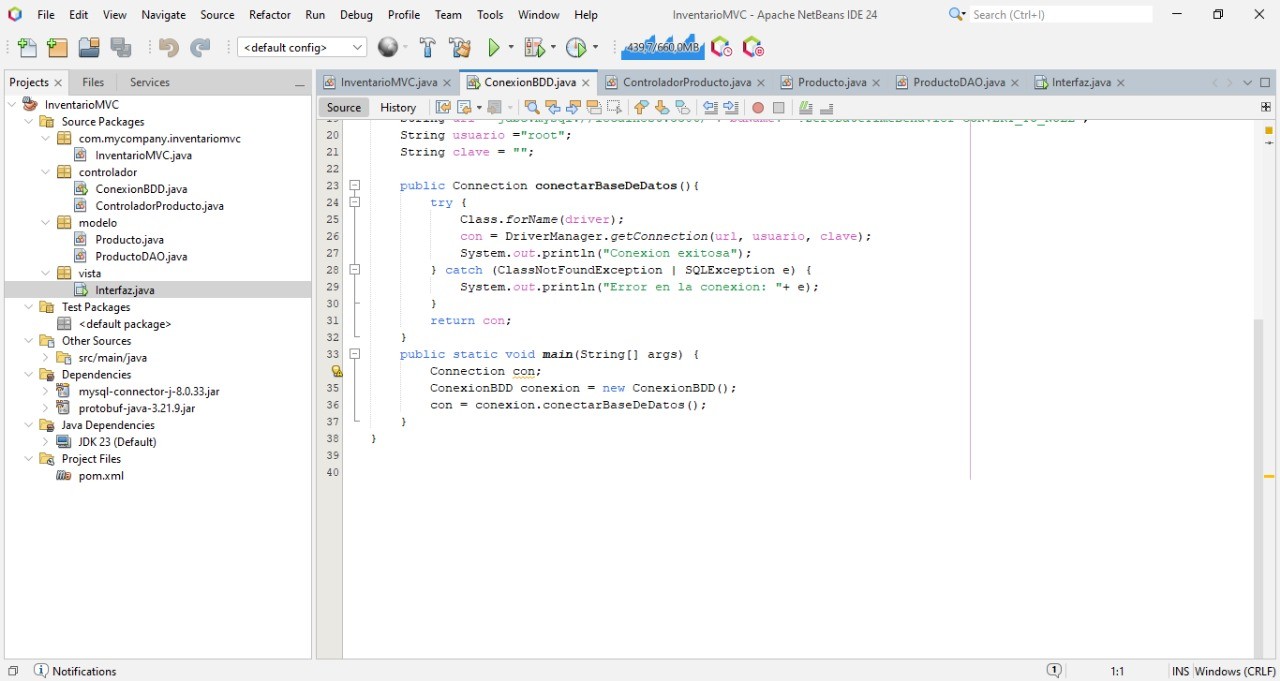
DateTime en MySQL cuando hay fechas con valor 0000-00-00.

**String usuario = "root";**

* Contiene el nombre de usuario con el que se accederá a MySQL.
* "root" es el usuario administrador predeterminado de MySQL

**String clave = "";**

* Almacena la contraseña del usuario "root".
* En este caso, está vacía (""), lo que significa que el usuario "root" no tiene contraseña configurada en MySQL.
* Nota de seguridad: No es recomendable dejar credenciales en código. Es mejor usar variables de entorno o un archivo de configuración.



*Ilustración 9*

**Atributos de la conexión**

* **String driver:** Define el controlador JDBC para la conexión.
* **String url:** Representa la URL de conexión a la base de datos.
* **String usuario:** Usuario con permisos para acceder a la base de datos.
* **String clave:** Contraseña correspondiente al usuario.

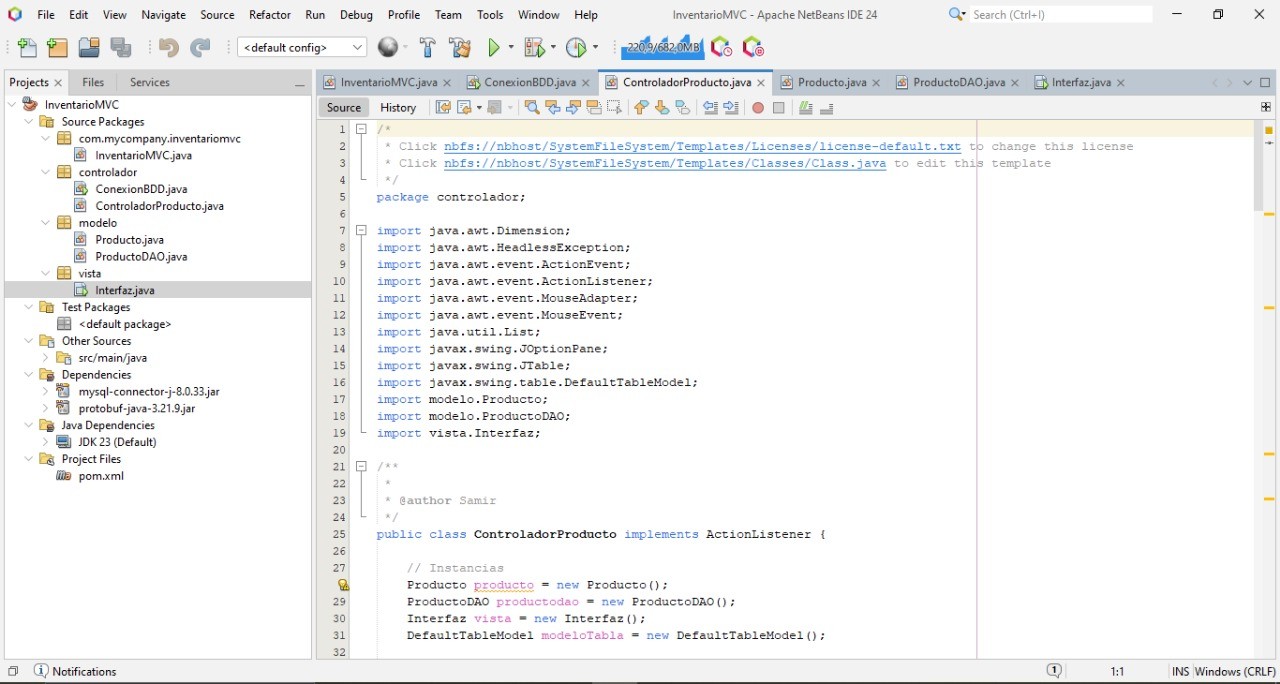
**Método conectarBaseDeDatos()**

* **Class.forName(driver)**: Carga dinámicamente el controlador JDBC especificado.
* **DriverManager.getConnection(url, usuario, clave):** Intenta establecer la conexión usando los parámetros definidos.
* Si la conexión es exitosa, imprime "Conexión exitosa".
* Si ocurre un error, captura la excepción (ClassNotFoundException o SQLException) e imprime un mensaje de error.

**Arquitectura MVC**

El proyecto está dividido en paquetes:

* **modelo**: Lógica de datos (ej., ProductoDAO.java).
* **controlador**: Control de la lógica (ej., ControladorProducto.java).
* **vista**: Interfaz gráfica o de usuario (ej., Interfaz.java).



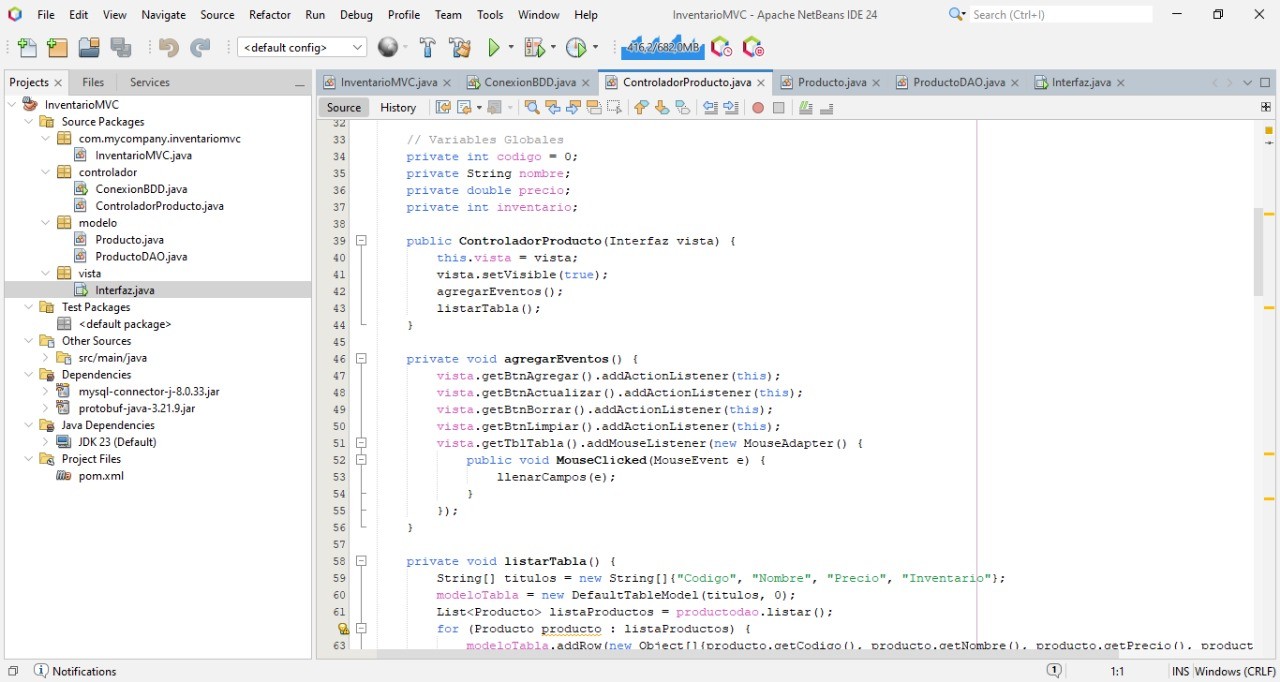
*Ilustración 10*

**Paquetes y librerías**

* **Paquete:** controlador. Agrupa las clases relacionadas con la lógica de control.
* **java.awt y javax.swing:** Para manejar eventos y elementos de la interfaz gráfica como tablas, paneles y cuadros de diálogo.
* **modelo:** Para interactuar con las clases Producto y ProductoDAO.
* **vista.Interfaz:** Para usar la clase que define la interfaz gráfica.

**Instancias Declaradas**

* **Producto producto**: Representa un producto específico.
* **ProductoDAO productodao:** Maneja las operaciones de base de datos para productos (como consultas y actualizaciones).
* **Interfaz vista:** Representa la ventana gráfica del usuario.
* **DefaultTableModel modeloTabla:** Modelo para manejar datos dentro de tablas en la interfaz gráfica.



*Ilustración 11*

**Variables Globales**

* **int codigo = 0:**

Representa el código único de cada producto en el inventario. Se inicializa en 0 y se actualiza al seleccionar o manipular datos de la tabla.

* **String nombre:**

Contiene el nombre del producto. Este dato se utiliza para mostrar y registrar información en la tabla.

* **double precio:**

Almacena el precio unitario del producto. Es importante para registrar el costo en la base de datos y mostrarlo en la interfaz.

* **int inventario:**

Representa la cantidad disponible del producto en stock. Se utiliza para controlar el nivel de inventario en la base de datos y en la interfaz gráfica.

**Constructor:** ControladorProducto(Interfaz vista)

* **this.vista = vista:**

Conecta el controlador con la interfaz gráfica para permitir que interactúen.

* **vista.setVisible(true):**

Hace visible la ventana de la interfaz gráfica al usuario.

* **agregarEventos():**

Configura los eventos para que los botones y la tabla puedan responder a las acciones del usuario, como agregar, actualizar, eliminar o seleccionar productos.

* **listarTabla():**

Llena la tabla en la interfaz gráfica con los datos actuales de los productos obtenidos desde la base de datos.

**Método:** agregarEventos()

* Configura los "listeners" para los botones y la tabla.

**Botones:**

* **Agregar:** Para registrar un nuevo producto.
* **Actualizar:** Para modificar los datos de un producto seleccionado.
* **Borrar:** Para eliminar un producto del inventario.
* **Limpiar:** Para limpiar los campos de texto de la interfaz gráfica.

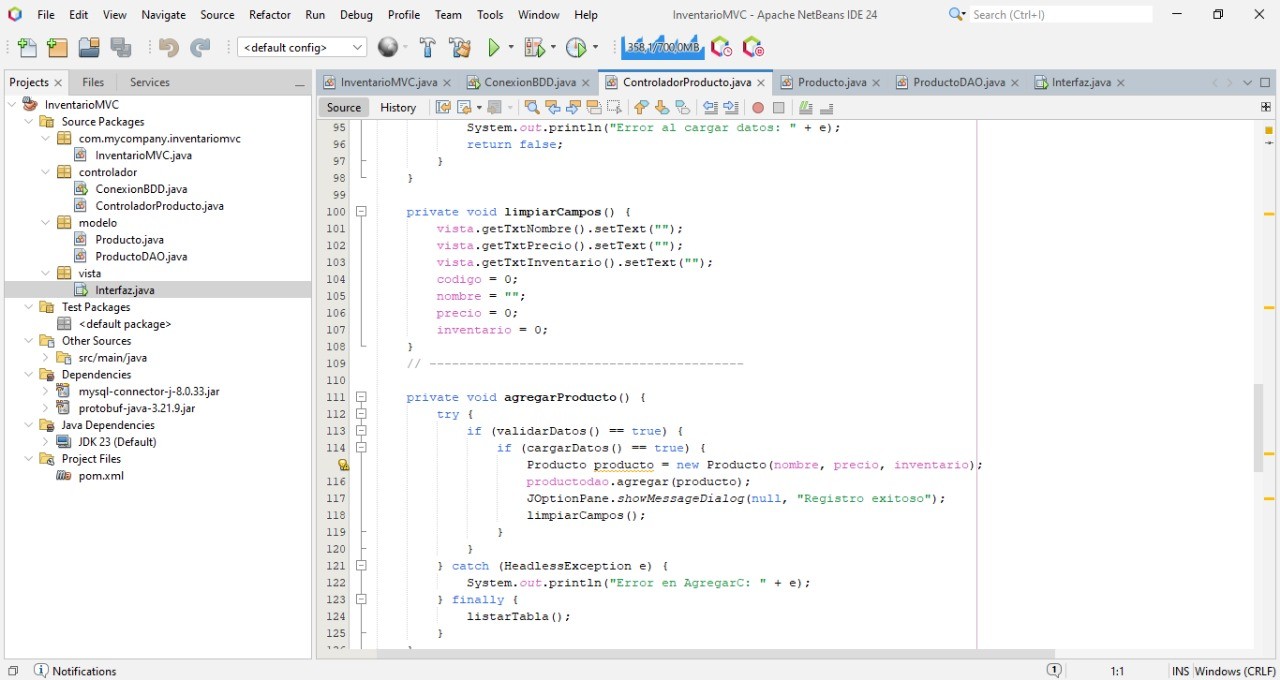
**Tabla:**

* Detecta clics en las filas para cargar los datos seleccionados en los campos de la interfaz.

**Método:** listarTabla()

Llena la tabla con los datos de productos obtenidos desde la base de datos:

* Define las columnas de la tabla: "Código", "Nombre", "Precio", "Inventario".
* Recupera la lista de productos desde el DAO.
* Itera sobre cada producto y lo agrega como una fila a la tabla.



*Ilustración 12*

**Variables dentro del método limpiarCampos()**

* **vista.getTxtNombre().setText(""):**

Limpia el campo de texto donde se ingresa el nombre del producto en la interfaz gráfica, dejándolo vacío.

* **vista.getTxtPrecio().setText(""):**

Limpia el campo de texto donde se ingresa el precio del producto, dejándolo vacío.

* **vista.getTxtInventario().setText(""):**

Limpia el campo de texto donde se ingresa la cantidad de inventario del producto, dejándolo vacío.

* **codigo:**

Representa el identificador único del producto. Se reinicia a 0 al limpiar los campos.

* **nombre:**

Contiene el nombre del producto. Se establece como una cadena vacía ("") al limpiar los campos.

* **precio:**

Variable que almacena el precio del producto. Se reinicia a 0.

* **inventario:**

Representa la cantidad disponible del producto en inventario. También se reinicia a 0.

**Variables dentro del método agregarProducto()**

* **Producto producto =** new Producto(nombre, precio, inventario):

Crea un nuevo objeto de la clase Producto utilizando los valores almacenados en las variables nombre, precio e inventario.

* **productodao.agregar(producto):**

Llama al método agregar() de la clase ProductoDAO para guardar el producto en la base de datos o en una colección.

* **JOptionPane.showMessageDialog(null, "Registro exitoso"):**

Muestra un cuadro de diálogo indicando que el registro del producto fue exitoso.

* **limpiarCampos():**

Se invoca para limpiar los campos de texto y reiniciar las variables después de agregar el producto.

* **listarTabla():**

Este método se ejecuta al final del bloque finally. Probablemente actualiza la tabla en la interfaz gráfica con los datos más recientes del inventario.

**Variables de control en el método**

* **validarDatos():**

Método que verifica si los datos ingresados son válidos antes de intentar agregar el producto.

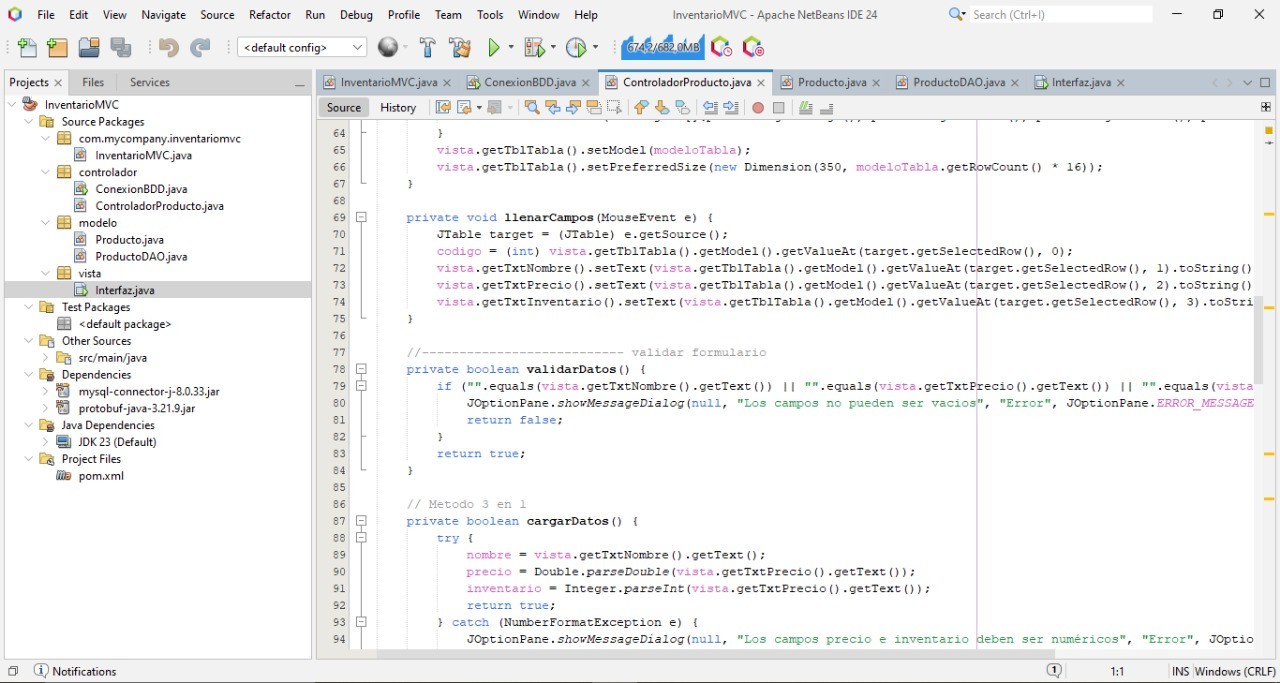
Retorna un booleano (true o false).

* **cargarDatos():**

Método que probablemente carga los datos desde los campos de texto a las variables nombre, precio e inventario. Retorna true si los datos fueron cargados exitosamente.

* **HeadlessException e:**

Variable que captura cualquier excepción que pueda ocurrir al interactuar con la interfaz gráfica en el bloque catch.



*Ilustración 13*

**1. Método llenarCampos(MouseEvent e) Variables:**

* **JTable target:**

Obtiene la tabla donde ocurrió el evento del clic. e.getSource() devuelve la fuente del evento.

* **codigo:**

Se asigna el valor de la primera columna (0) de la fila seleccionada en la tabla. Es convertido a un entero ((int)).

* **vista.getTxtNombre().setText(...):**

Llena el campo de texto para el nombre del producto con el valor de la segunda columna (1) de la fila seleccionada.

* **vista.getTxtPrecio().setText(...):**

Llena el campo de texto para el precio del producto con el valor de la tercera columna (2) de la fila seleccionada.

* **vista.getTxtInventario().setText(...):**

Llena el campo de texto para la cantidad en inventario con el valor de la cuarta columna (3) de la fila seleccionada. **2. Método validarDatos() Variables y lógica:**

* **if ("".equals(vista.getTxtNombre().getText())):**

Comprueba si el campo del nombre está vacío.

* **JOptionPane.showMessageDialog(...):**

Muestra un mensaje de error si algún campo está vacío.

* **return false:**

Detiene la ejecución del método si se encuentra algún campo vacío.

* **return true:**

Indica que los datos son válidos si ningún campo está vacío.

**3. Método cargarDatos() Variables y lógica:**

* **nombre:**

Se asigna el texto ingresado en el campo del nombre del producto.

* **precio:**

Convierte el texto del campo de precio a un valor de tipo Double con Double.parseDouble(...).

* **inventario:**

Convierte el texto del campo de inventario a un valor de tipo Integer con Integer.parseInt(...).

* **catch (NumberFormatException e):**

Captura cualquier error que ocurra si los valores en los campos de precio o inventario no son numéricos.

* **JOptionPane.showMessageDialog(...):**

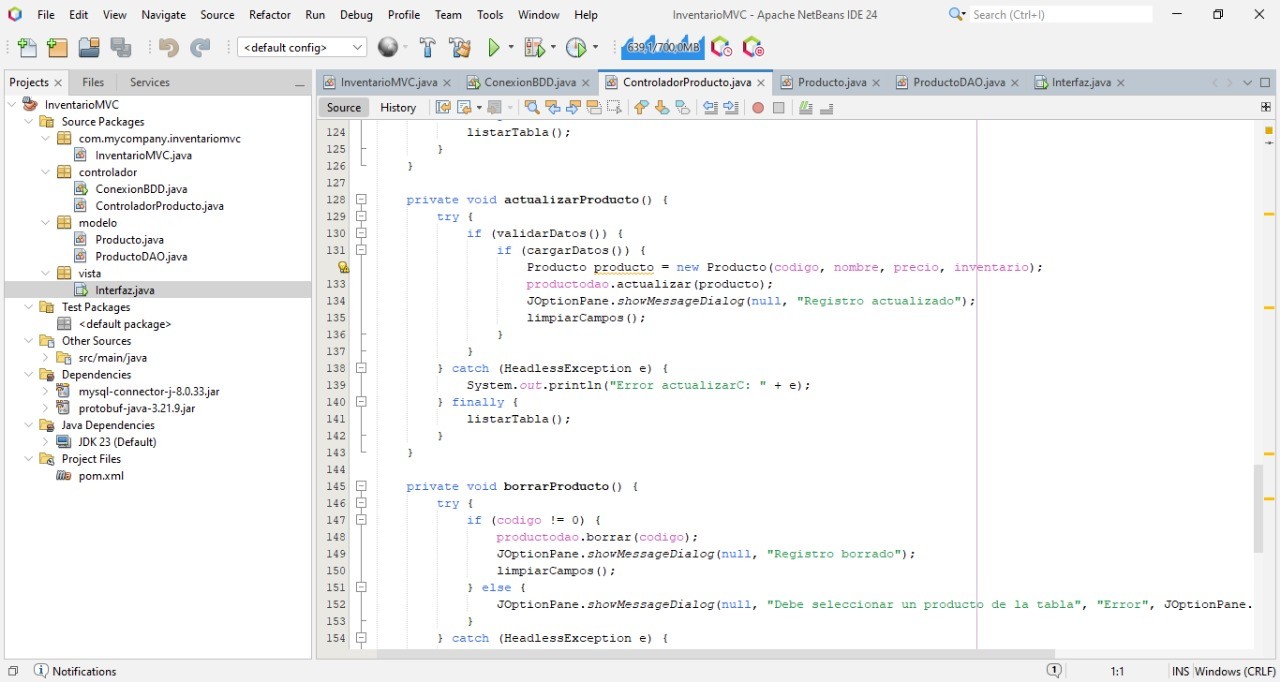
Muestra un mensaje de error si ocurre una excepción.

* **return true:**

Indica que los datos fueron cargados exitosamente.

* **return false:**

Indica que hubo un error al cargar los datos (por ejemplo, si los valores no eran numéricos).



*Ilustración 14*

1. **Método actualizarProducto()**
   * **if (validarDatos()):**

Verifica que los campos no estén vacíos usando el método validarDatos.

* + **if (cargarDatos()):**

Carga los datos ingresados en los campos de texto en las variables nombre, precio e inventario mediante el método cargarDatos.

* + **Producto producto = new Producto(codigo, nombre, precio, inventario):** Crea un objeto de tipo Producto con los datos cargados. Aquí, codigo es el identificador único del producto.
  + **productodao.actualizar(producto):**

Llama al método actualizar de la clase ProductoDAO para enviar el objeto actualizado a la base de datos.

* + **JOptionPane.showMessageDialog(...):**

Muestra un mensaje de éxito si el producto fue actualizado correctamente.

* + **limpiarCampos():**

Limpia los campos de texto después de actualizar.

* + **Excepciones:**

Si ocurre un error, lo captura y lo imprime con System.out.println(...).

Finalmente, llama al método listarTabla() para actualizar la tabla en la interfaz gráfica.

1. **Método borrarProducto()**

Este método elimina un producto del inventario basado en el código.

Flujo del método:

* + **if (codigo != 0):**

Verifica que el código del producto no sea cero. Esto asegura que un producto ha sido seleccionado para borrar.

* + **productodao.borrar(codigo):**

Llama al método borrar de la clase ProductoDAO, pasando el código del producto a eliminar.

* + **JOptionPane.showMessageDialog(...):**

Muestra un mensaje de éxito si el producto fue eliminado correctamente.

* + **limpiarCampos():**

Limpia los campos de texto después de borrar.

* + **Else (cuando codigo == 0):**

Si no se selecciona ningún producto, muestra un mensaje de error indicando que se debe seleccionar un producto de la tabla.

* + **Excepciones:**

Captura cualquier error y lo imprime con System.out.println(...).

**Variables involucradas**

* + **codigo:**

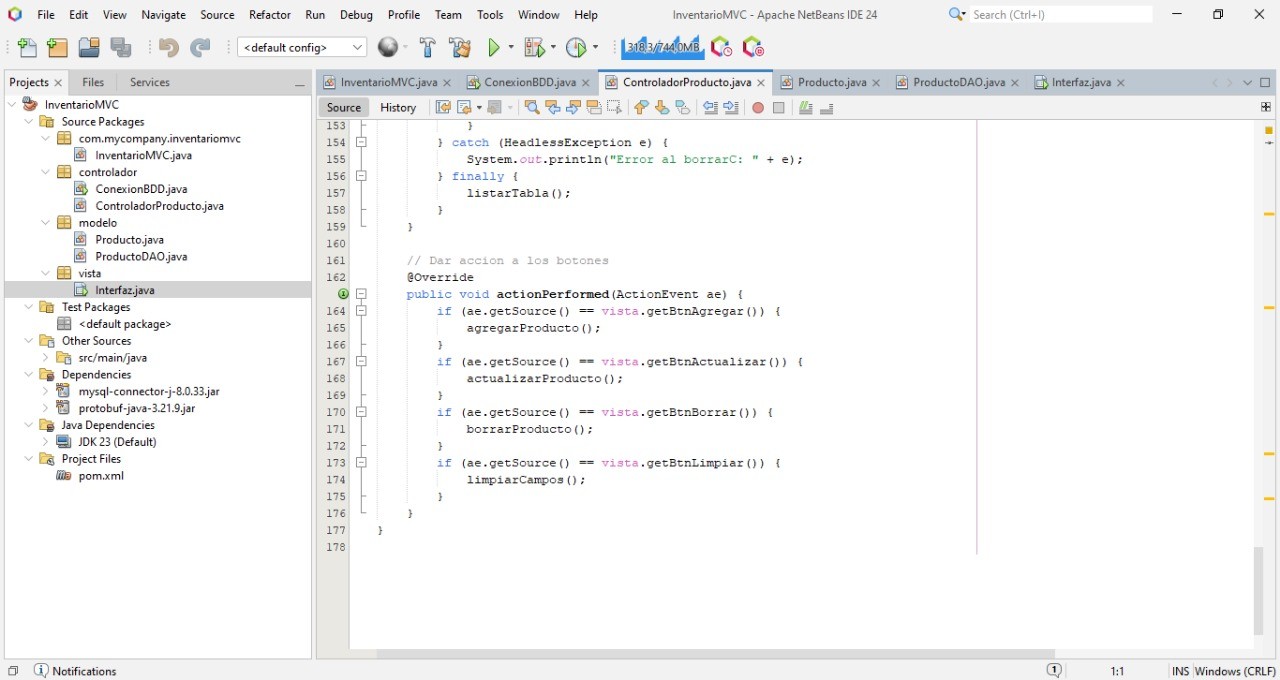
Identificador único del producto que se va a actualizar o borrar.

* + **nombre, precio, inventario:**

Datos del producto cargados desde los campos de texto o seleccionados de la tabla.

* + **productodao:**

Objeto de la clase ProductoDAO, encargado de realizar las operaciones con la base de datos.



*Ilustración 15*

1. **vista**
   * **Ubicación:** En el controlador (ControladorProducto.java).
   * **Función:** Representa la interfaz gráfica (Interfaz.java) y permite acceder a sus botones.
2. **ae.getSource()**
   * **Ubicación:** Dentro del método actionPerformed().
   * **Función**: Identifica qué botón de la GUI fue presionado.

**Botones de la interfaz y su función en el inventario**

Cada if verifica qué botón fue presionado y ejecuta la función correspondiente:

* + **vista.getBtnAgregar()**

**Acción:** Llama a agregarProducto().

**Función:** Agrega un nuevo producto al inventario.

* + **vista.getBtnActualizar()**

**Acción:** Llama a actualizarProducto().

**Función:** Modifica un producto existente en la base de datos.

* + **vista.getBtnBorrar()**

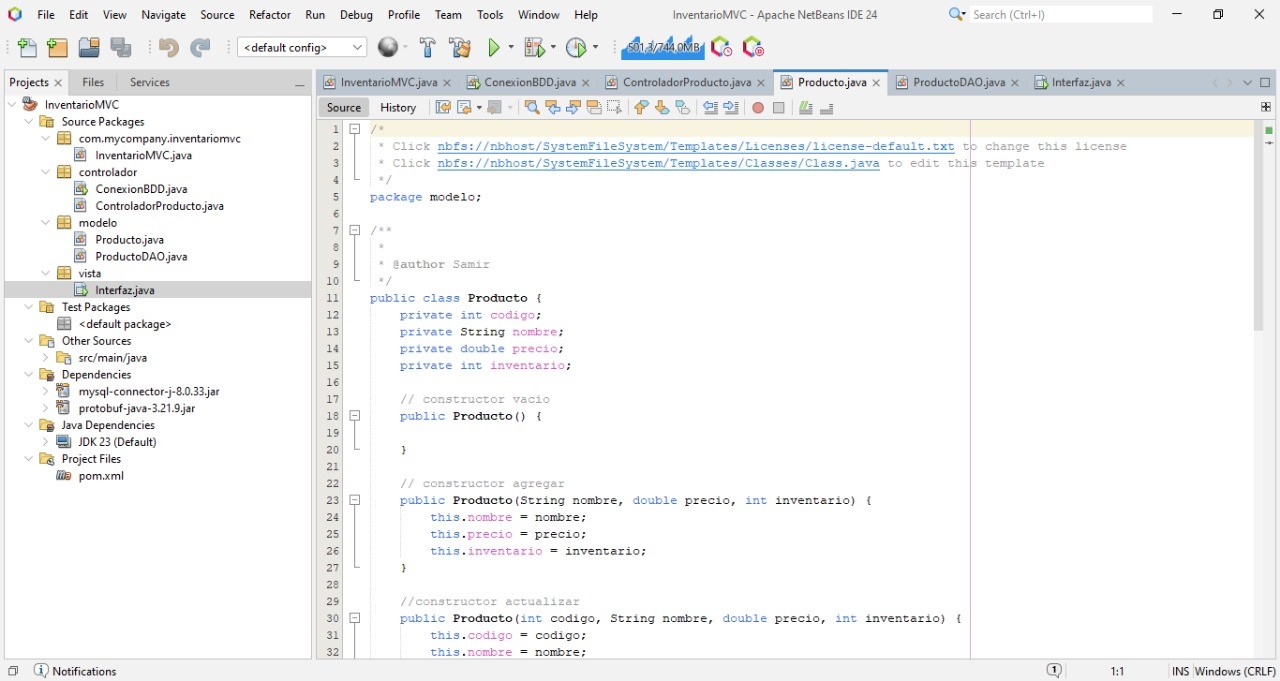
**Acción:** Llama a borrarProducto().

**Función:** Elimina un producto del inventario.

* + **vista.getBtnLimpiar()**

**Acción:** Llama a limpiarCampos().

**Función:** Borra los campos de entrada en la interfaz gráfica.



*Ilustración 16*

**Variables en Producto.java**

* + **private int codigo;**

**Función:** Identifica cada producto con un código único.

**Tipo:** int (número entero).

**Uso:** Se usa cuando se quiere modificar o eliminar un producto específico.

* + **private String nombre; Función:** Almacena el nombre del producto.

**Tipo:** String (texto).

**Ejemplo:** "Laptop Dell XPS 15".

* + **private double precio;**

**Función:** Guarda el precio del producto.

**Tipo:** double (número decimal).

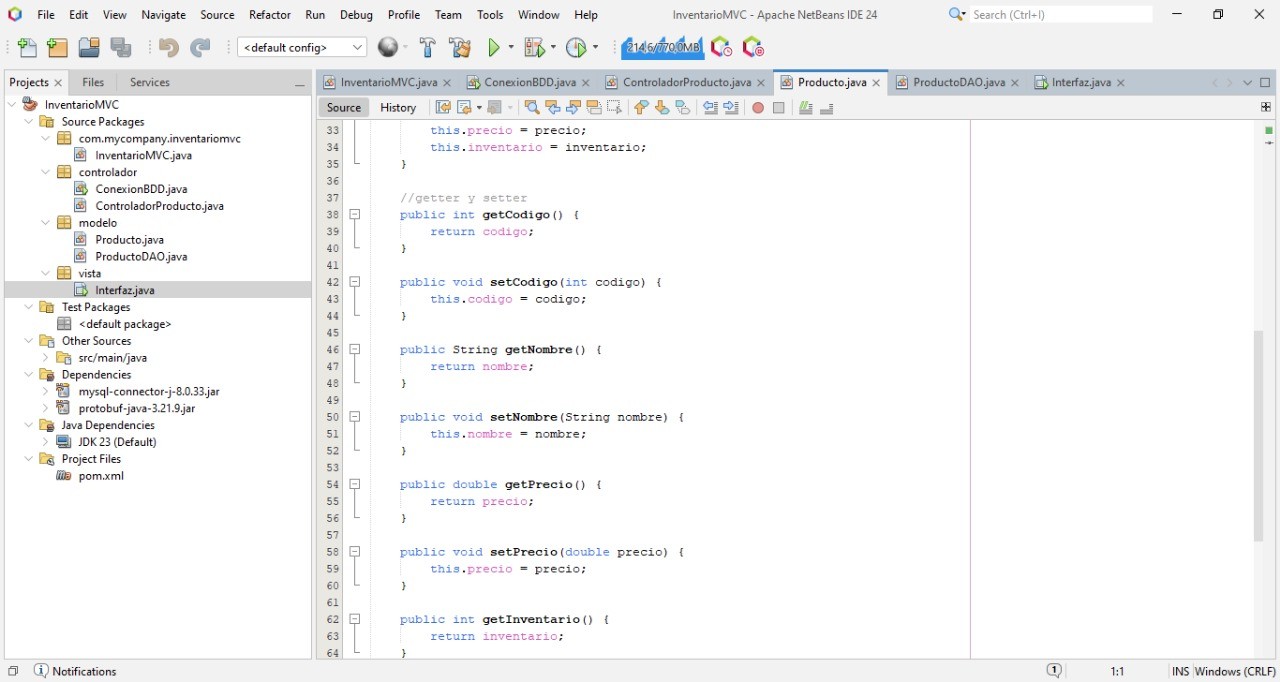
**Ejemplo:** 1599.99.

* + **private int inventario;**

**Función:** Representa la cantidad de unidades disponibles en el inventario.

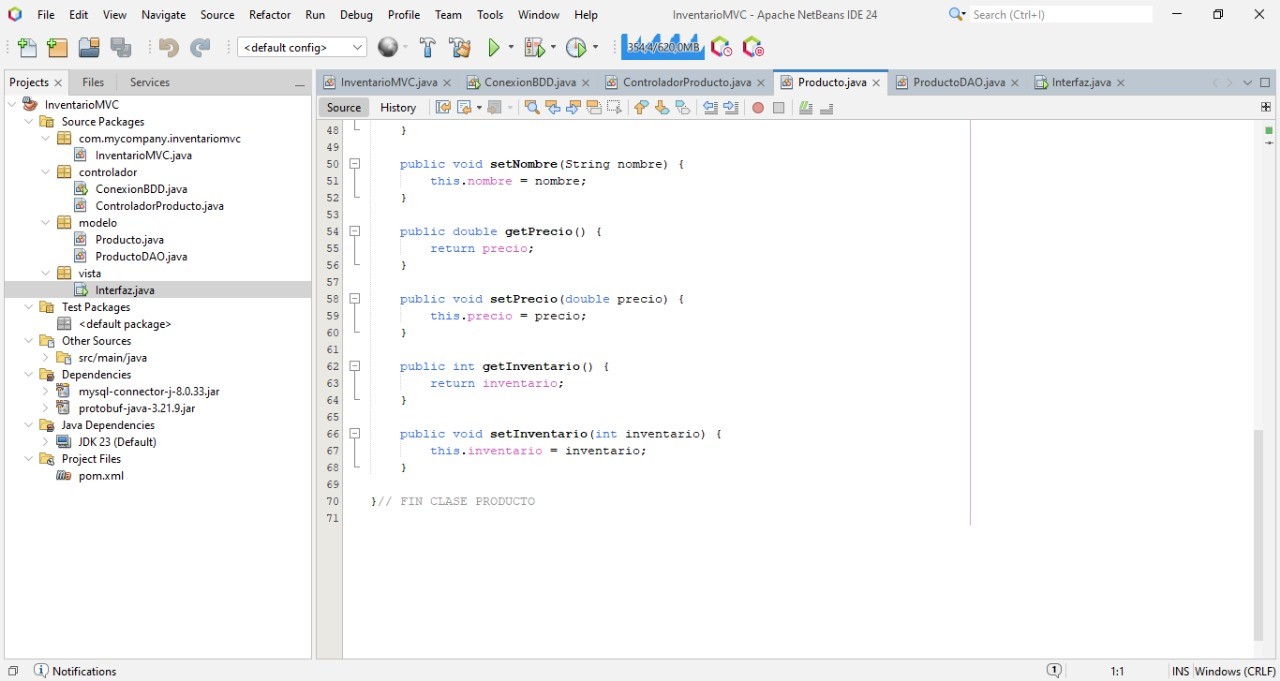
**Tipo:** int (número entero).

**Ejemplo:** 50.



*Ilustración 17*

* + **codigo:** Identificador único del producto en el inventario. Se usa para diferenciar productos y facilitar su búsqueda en la base de datos.
  + **nombre:** Nombre del producto. Permite identificarlo fácilmente en el sistema y mostrarlo en la interfaz de usuario.
  + **precio:** Valor monetario del producto. Se usa para cálculos de ventas, costos y facturación.
  + **inventario:** Cantidad disponible del producto en stock. Es clave para el control de existencias y la reposición de productos.



**Variables y su función en el inventario** • **nombre (String)**

**Función:** Representa el nombre del producto en el inventario.

**Métodos:**

**getNombre():** Devuelve el nombre del producto.

**setNombre(String nombre):** Permite asignar o cambiar el nombre del producto.

* + **precio (double)**

**Función:** Guarda el precio del producto en el inventario.

**Métodos:**

**getPrecio():** Retorna el precio del producto.

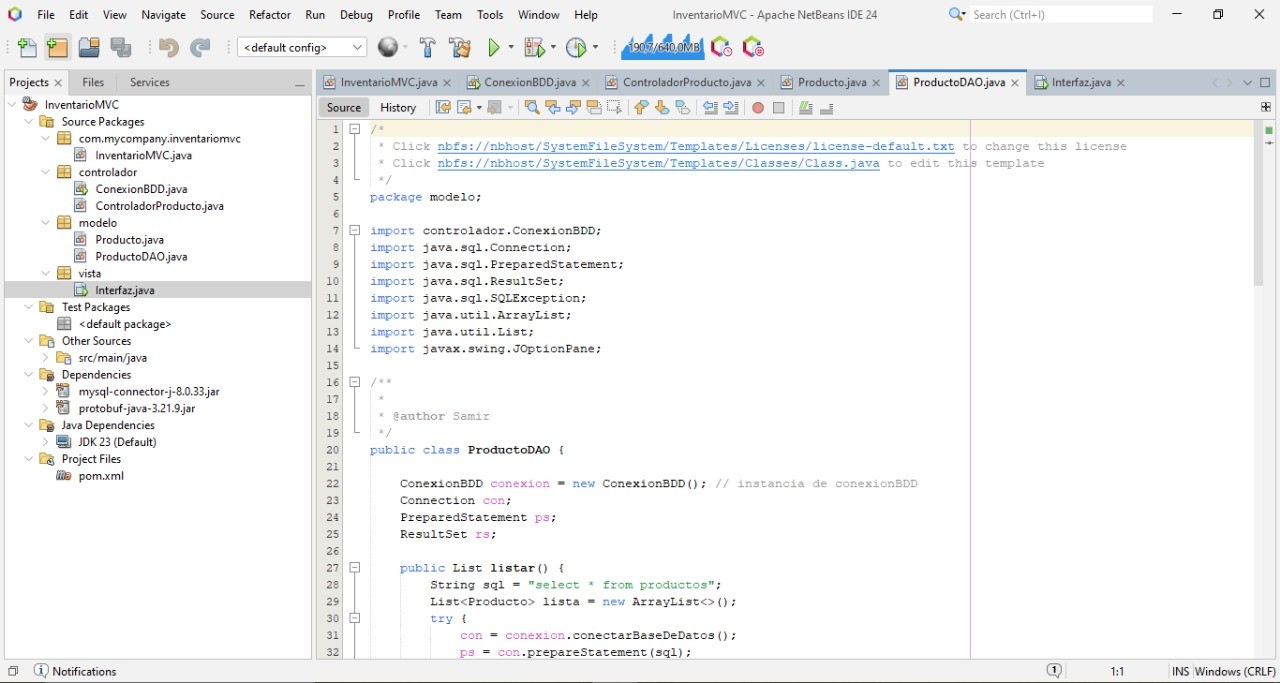
**setPrecio(double precio):** Establece un nuevo precio para el producto. • **inventario (int)**

**Función:** Indica la cantidad de unidades disponibles del producto en el inventario.

**Métodos:**

**getInventario():** Devuelve la cantidad actual en inventario.

**setInventario(int inventario):** Modifica la cantidad de productos disponibles en el inventario.



*Ilustración 18*

**Explicación de las variables y su función en el inventario**

* **ConexionBDD conexion = new ConexionBDD();**

Es una instancia de la clase ConexionBDD, que maneja la conexión con la base de datos.

Permite realizar consultas SQL, insertar, actualizar y eliminar datos de la tabla de productos.

* **Connection con;**

Variable que almacena la conexión activa con la base de datos.

Se obtiene llamando al método conexion.conectarBaseDeDatos().

* **PreparedStatement ps;**

Se usa para ejecutar consultas SQL de forma segura (evita inyecciones SQL).

Se prepara con con.prepareStatement(sql).

* **ResultSet rs;**

Guarda los resultados obtenidos de una consulta SQL.

Se usa para recuperar los datos de la base de datos.

**Método listar() y su función**

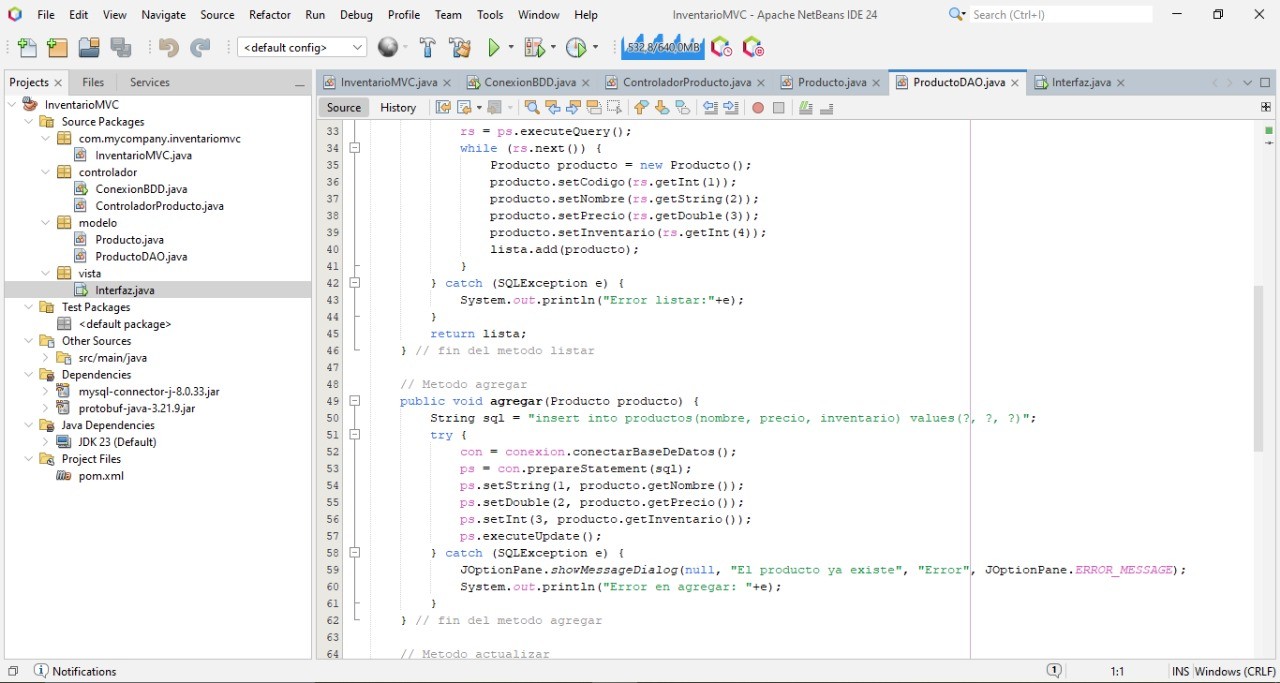
* **Consulta SQL:** "SELECT \* FROM productos" (Obtiene todos los productos de la base de datos).
* **Lista los productos:**

**Crea una lista ArrayList<Productos> para almacenar los productos recuperados.**

Establece la conexión con con = conexion.conectarBaseDeDatos();.

Prepara y ejecuta la consulta con ps.executeQuery();.

* **Retorno:** Devuelve una lista con todos los productos de la base de datos.



*Ilustración 19*

**1. Método listar()**

**Objetivo:**

* Recupera todos los productos almacenados en la base de datos y los devuelve en una

lista.

**Funcionamiento:**

**Se ejecuta la consulta SQL:** "SELECT \* FROM productos".

Se recorre el ResultSet (rs) con while(rs.next()), extrayendo la información de cada producto:

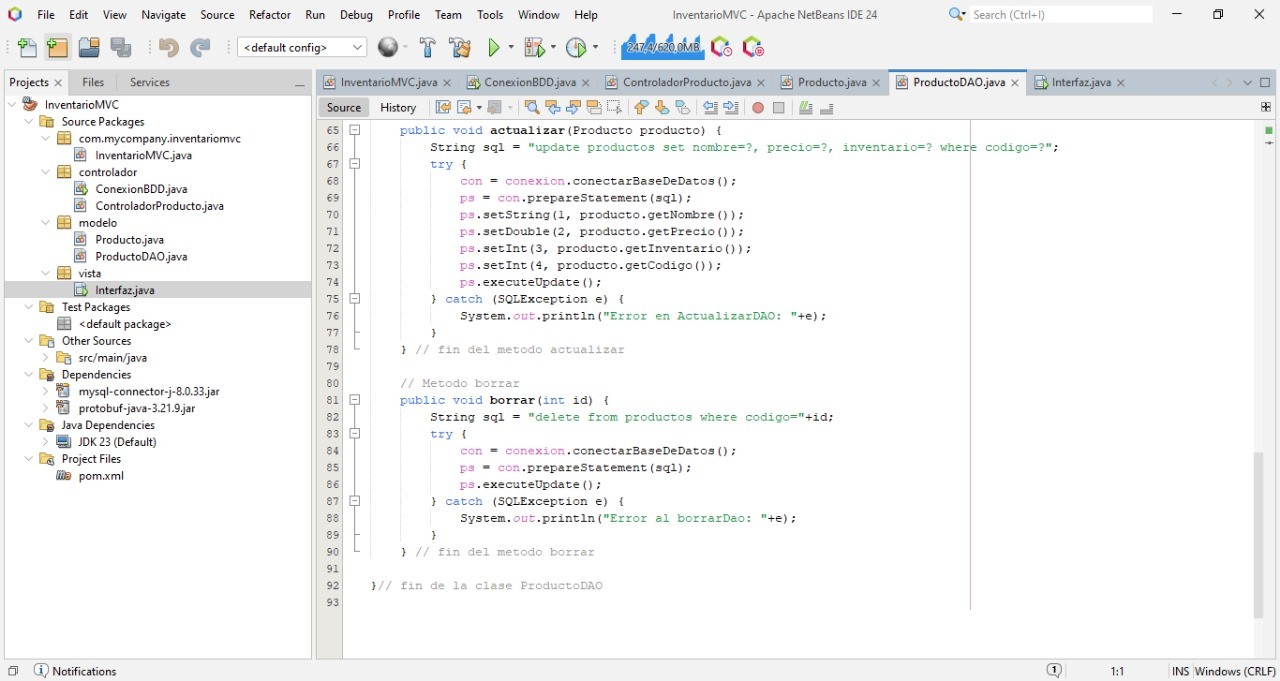
* **setCodigo(rs.getInt(1)):**Obtiene el código del producto.
* **setNombre(rs.getString(2)):** Obtiene el nombre.
* **setPrecio(rs.getDouble(3)):** Obtiene el precio.
* **setInventario(rs.getInt(4)) :** Obtiene la cantidad disponible en inventario.

Cada producto se almacena en un ArrayList<Productos>.

**Si ocurre un error, se muestra un mensaje en la consola:** System.out.println("Error listar:" + e).

**2. Método agregar(Producto producto)**

* **ps.setString(1, producto.getNombre()) :** Inserta el nombre del producto.
* **ps.setDouble(2, producto.getPrecio()) :** Inserta el precio del producto.
* **ps.setInt(3, producto.getInventario()) :**Inserta la cantidad en inventario.



*Ilustración 20*

**Parámetros del método actualizar (producto):**

Es una instancia de la clase Producto que representa un elemento del inventario.

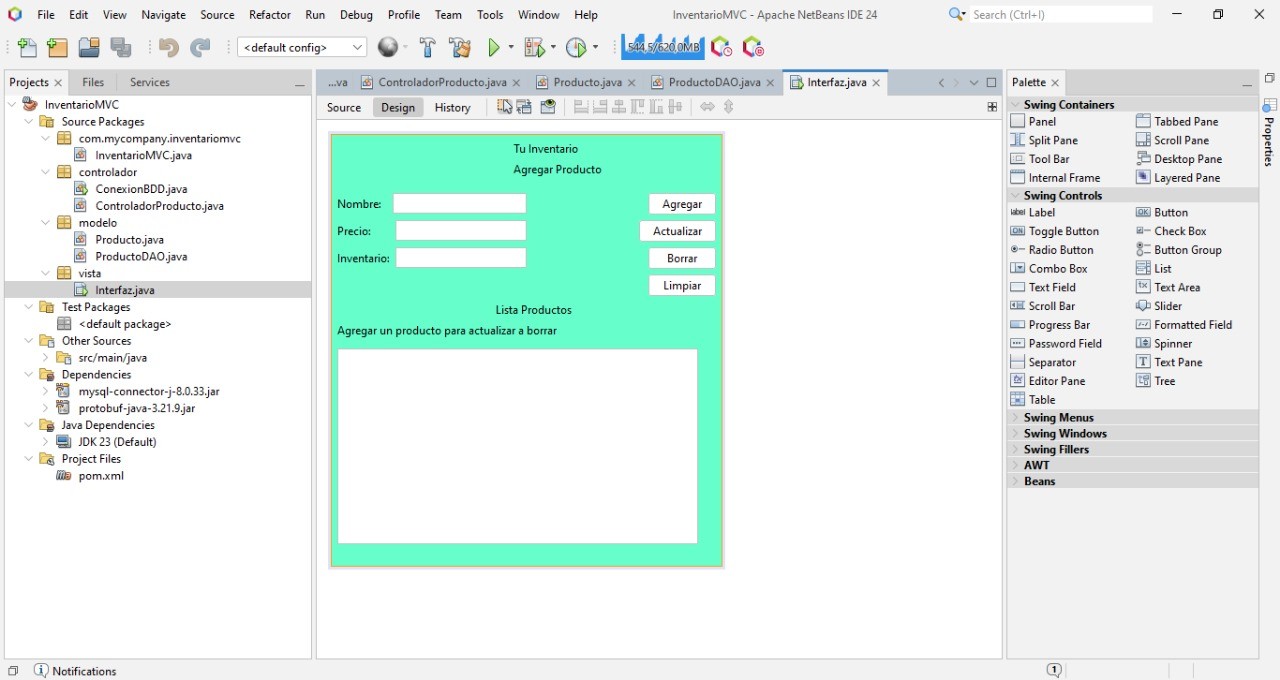
* **Contiene los atributos del producto como: producto.getNombre():** Nombre del producto. **producto.getPrecio():** Precio del producto. **producto.getInventario():** Cantidad en inventario. **producto.getCodigo():** Código único del producto.
* **Parámetro del método borrar (id):**

Representa el identificador único del producto que se desea eliminar de la base de datos.

* **Excepciones (catch (SQLException e)):**

Capturan errores que puedan ocurrir durante la ejecución de las consultas SQL.

**Ejemplo:** Si la base de datos está desconectada o la consulta tiene errores, se imprime un mensaje de error.



*Ilustración 21*

**Campos de entrada (TextFields)**

* **Nombre**: Campo de texto donde se introduce el nombre del producto a agregar, actualizar o buscar.
* **Precio**: Campo de texto para ingresar el precio del producto.
* **Inventario**: Campo de texto para indicar la cantidad de existencias disponibles del producto.

1. **Botones de acción**
   * **Agregar**:

Permite añadir un nuevo producto al inventario con los datos ingresados en los campos de texto (nombre, precio, inventario).

Al presionar este botón, se ejecuta un método del controlador que guarda los datos en la base de datos a través del modelo (DAO).

* + **Actualizar**:

Modifica los datos de un producto existente en función de su código o identificador.

Los campos se rellenan con los datos actuales del producto, que se pueden editar y guardar.

* + **Borrar**:

Elimina un producto del inventario según su identificador (normalmente seleccionado en la tabla o ingresado en un campo relacionado).

Manda una solicitud al modelo para ejecutar la operación DELETE en la base de datos.

* + **Limpiar**:

Borra los datos de los campos de entrada para permitir ingresar información nueva.

1. **Área de lista de productos (JTable o JTextArea)**
   * **"Lista Productos"**:

Muestra un listado de los productos actualmente almacenados en el inventario. Es probable que este componente sea una tabla dinámica (JTable) o un área de texto

(JTextArea).

Los productos listados podrían estar ordenados por nombre, precio o cantidad.

También podría permitir seleccionar un producto para realizar operaciones como actualizar o borrar.

1. **Instrucciones**
   * **Mensaje debajo de la lista**:

**"Agregar un producto para actualizar o borrar":** Indica que para actualizar o eliminar, es necesario que previamente exista un producto en el inventario.

**Diseño visual**

* + **Estilo minimalista**:

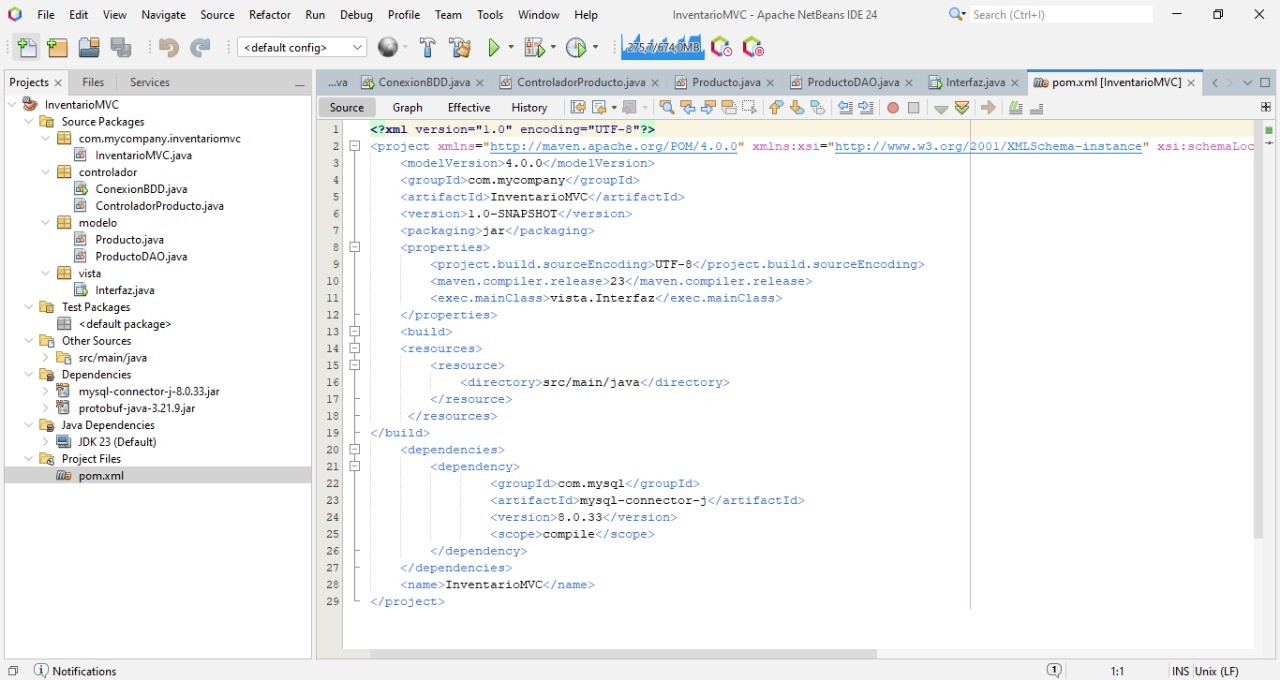
Fondo de color verde claro para destacar la sección principal.

Organización de elementos en una cuadrícula simple para facilitar el uso.

**Función en el sistema**

Esta interfaz actúa como la capa de Vista en el patrón MVC:

* + **Vista (Interfaz gráfica)**: Interactúa con el usuario para recibir entradas y mostrar resultados.
  + **Controlador**: Gestiona la lógica y los eventos, enviando las solicitudes al modelo.
  + **Modelo (DAO)**: Ejecuta las operaciones en la base de datos, como agregar, actualizar, borrar o listar productos.



**Variables en el archivo ProductoDAO.java (gestión de base de datos)**

1. **sql**
   * **Función:** Es una variable de tipo String que contiene las instrucciones SQL para interactuar con la base de datos.
2. **con**
   * **Función:** Representa la conexión activa con la base de datos.
   * **Uso:** Proviene del método conectarBaseDeDatos() definido en la clase

ConexionBDD. Permite ejecutar operaciones SQL.

1. **ps**
   * **Función:** Es un objeto de tipo PreparedStatement, que se utiliza para ejecutar las consultas SQL de manera segura.
   * **Uso:** Configura los parámetros de las consultas SQL (ps.setString, ps.setDouble, ps.setInt).
2. **producto**
   * **Función:** Es un objeto de la clase Producto que contiene los datos de un producto

(nombre, precio, inventario, código).

* + **Uso:** Al actualizar, se obtienen sus atributos con métodos como producto.getNombre(), producto.getPrecio().

1. **id**
   * **Función:** Identifica el producto de forma única en la tabla de la base de datos.
   * **Uso:** En la consulta SQL de eliminación (where codigo=" + id).

**Variables en la interfaz gráfica (Interfaz.java)**

1. **Nombre (Campo de texto)**
   * **Función:** Captura o muestra el nombre del producto.
   * **Relación:** Este dato se almacena en la columna nombre de la base de datos.
2. **Precio (Campo de texto)**
   * **Función:** Captura o muestra el precio del producto.
   * **Relación:** Este dato se almacena en la columna precio de la tabla.
3. **Inventario (Campo de texto)**
   * **Función:** Captura o muestra la cantidad disponible del producto.
   * **Relación:** Este dato se almacena en la columna inventario.
4. **Botones:**
   * **Agregar:** Llama al método en el controlador para insertar un nuevo producto en la base de datos.
   * **Actualizar:** Llama al método para modificar los datos de un producto existente.
   * **Borrar:** Llama al método para eliminar un producto.
   * **Limpiar:** Limpia los campos de texto de la interfaz gráfica. **Variables generales relacionadas con el inventario**

**nombre**

* + Representa el nombre del producto.
  + Almacenado como String.

**precio**

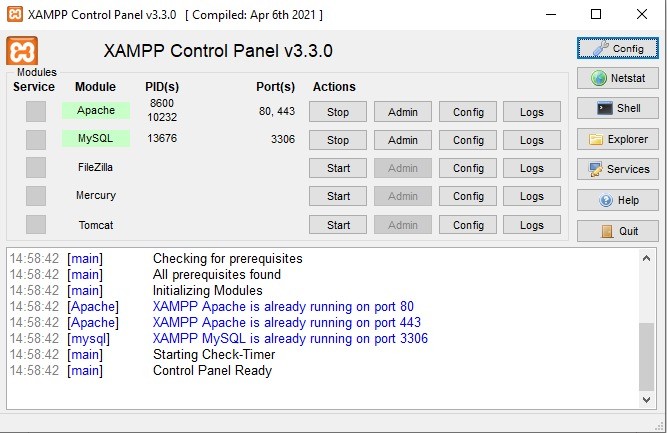
* + Representa el costo del producto.
  + Almacenado como double.

**inventario**

* + Representa la cantidad disponible del producto.
  + Almacenado como int.

**codigo**

* + Identificador único de cada producto.
  + Almacenado como int.



*Ilustración 22*

**Servicios activos:**

* + **Apache**: Está corriendo en los puertos 80 y 443.
  + **MySQL**: Está corriendo en el puerto 3306. Ambos servicios están marcados en color verde, indicando que están en funcionamiento.

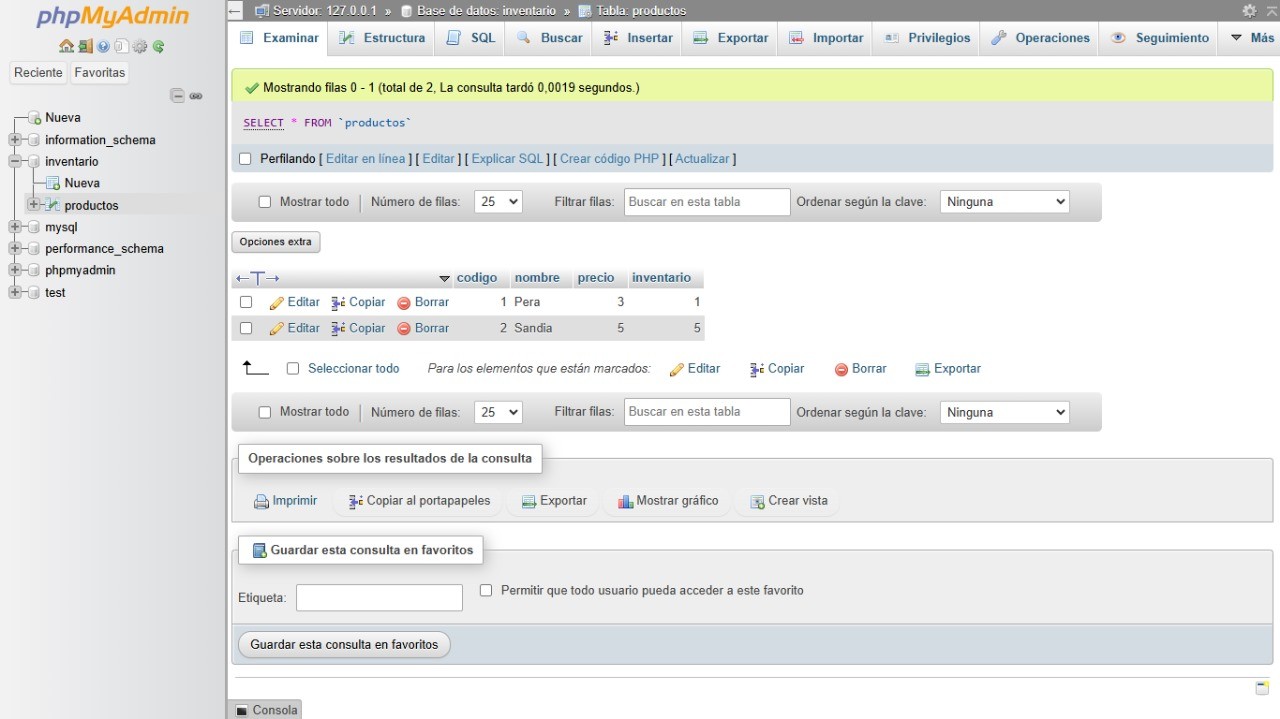
**Acciones disponibles**:

Para Apache y MySQL se pueden realizar acciones como:

* + **Stop**: Detener el servicio.
  + **Admin**: Acceder a las herramientas administrativas.
  + **Config** y **Logs**: Ver configuraciones y registros.

**Registros en la parte inferior**:

* + Confirma que ambos servicios están activos:
  + Apache ya está corriendo en los puertos 80 y 443.
  + MySQL está corriendo en el puerto 3306.
  + Mensaje final: "Control Panel Ready", indicando que todo está listo para usar.

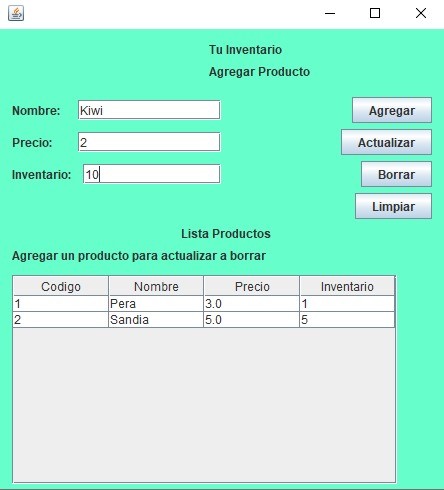


*Ilustración 23*

* + Se está trabajando en la base de datos "inventario", específicamente en la tabla

"productos".

* + La consulta ejecutada muestra todos los registros de la tabla.
  + Los datos de la tabla incluyen dos filas con información sobre productos: código, nombre, precio e inventario.
  + Existen opciones para editar, copiar, borrar o exportar cada registro, además de herramientas para buscar, ordenar y realizar otras acciones sobre los resultados de la consulta.



*Ilustración 24*

* + **Título**:

La aplicación se titula "Tu Inventario".

**Sección "Agregar Producto"**:

* + **Campos de entrada**:
  + **Nombre**: Aquí se ingresa el nombre del producto (en este caso, "Kiwi").
  + **Precio**: Para registrar el precio del producto (valor ingresado: 2).
  + **Inventario**: Para la cantidad de unidades disponibles (valor ingresado: 10).
  + **Botones disponibles**:
  + **Agregar**: Añade un nuevo producto con los datos ingresados.
  + **Actualizar**: Modifica un producto existente.
  + **Borrar**: Elimina el producto seleccionado.
  + **Limpiar**: Limpia los campos de entrada.
  + **Sección "Lista Productos"**:

Muestra una tabla con los productos del inventario actual.

Columnas:

* + **Código**: Identificador único.
  + **Nombre**: Nombre del producto.
  + **Precio**: Precio del producto.
  + **Inventario**: Cantidad disponible.

En la tabla aparecen dos productos:

* + Producto 1: Pera, con precio 3.0 y un inventario de 1 unidad.
  + Producto 2: Sandía, con precio 5.0 y un inventario de
    1. **CONCLUSIONES**

MVC divide el sistema en tres componentes (Modelo, Vista, Controlador), facilitando el mantenimiento y la escalabilidad del sistema de inventario.

La separación de las capas facilita la identificación y corrección de errores en el sistema de gestión de inventario.

La estructura MVC facilita actualizaciones y modificaciones sin interferir con el funcionamiento general del sistema de inventario.

* + 1. **RECOMENDACIONES**

Realizar el código de una manera ordenada para no tener ningún inconveniente mas adelante con la estructura para que sea más fácil de comprender,

Verificar que los controladores reciban las entradas del usuario desde la vista previa, para poder procesarlas y enviarlas al modelo, evitando que la lógica del negocio se mezcle con la interfaz usuario.

* + 1. **BIBLIOGRAFÍA**

Heizer, J., & Render, B. (2020). *Principles of Operations Management: Sustainability and Supply Chain Management* (12.a ed.). Pearson.

Elmasri, R., & Navathe, S. B. (2020). *Fundamentals of Database Systems* (7.a ed.).

Pearson.

Pressman, R. S., & Maxim, B. R. (2020). *Software Engineering: A Practitioner’s*

*Approach* (9.a ed.). McGraw-Hill.

Kroenke, D. M., & Auer, D. J. (2019). *Database Concepts* (9.a ed.). Pearson.

Laudon, K. C., & Laudon, J. P. (2021). *Management Information Systems: Managing the Digital Firm* (16.a ed.). Pearson.

Shelly, G. B., & Vermatt, R. E. (2018). *Discovering Computers: Fundamentals* (1.a ed.). Cengage Learning.

Ullman, J. D., & Widom, J. (2021). *A First Course in Database Systems* (4.a ed.).

P