**Documentación de MiniMarketPro**

**Integrantes:**

* **Carlos Ushiña**
* **Anthony Haro**
* **Francis Aconda**
* **Cristian Ushiña**

Estas importaciones son importantes para el funcionamiento del programa y en su desarrollo, como:

**import javax.swing.\*;**

Este import incluye las clases y componentes del paquete javax.swing, que proporciona bibliotecas para construir interfaces gráficas de usuario (GUI) en Java. En este código, se utilizan clases como JPanel, JtextField, Jbutton, Jtable, y JoptionPane para crear y gestionar la interfaz gráfica.

**Import javax.swing.table.DefaultTableModel;**

Este import introduce la clase DefaultTableModel del paquete javax.swing.table. DefaultTableModel es una implementación estándar de la interfaz TableModel utilizada comúnmente para gestionar datos tabulares en una Jtable. En el código, se utiliza para configurar y actualizar la tabla que muestra los productos.

**Import java.awt.event.ActionEvent;**

Este import pertenece al paquete java.awt.event y trae la clase ActionEvent. ActionEvent es un evento que se genera cuando ocurre una acción en un componente de la interfaz de usuario, como hacer clic en un botón. En el código, se utiliza para manejar eventos de botones.

**Import java.awt.event.ActionListener;**

Similar al anterior, este import pertenece a java.awt.event e incluye la interfaz ActionListener. La interfaz ActionListener se utiliza para manejar eventos de acción, como los generados por los botones cuando se produce una interacción del usuario. En el código, se implementan escuchadores de acción para los botones.

**Import java.sql.Connection;**

Este import es parte del paquete java.sql y trae la clase Connection. La clase Connection se utiliza para establecer una conexión con una base de datos. En el código, se utiliza para conectarse a la base de datos que almacena la información sobre los productos.

**Import java.sql.PreparedStatement;**

Este import también es parte de java.sql e incluye la clase PreparedStatement. PreparedStatement se utiliza para ejecutar consultas SQL precompiladas y parametrizadas en una base de datos. En el código, se utiliza para ejecutar consultas preparadas para agregar, actualizar o buscar productos en la base de datos.

**import java.sql.ResultSet;**

Este import, perteneciente a java.sql, incluye la clase ResultSet. ResultSet es un conjunto de resultados que representa los datos obtenidos de una consulta SQL. En el código, se utiliza para procesar los resultados de las consultas SQL y obtener información de la base de datos.

**import java.sql.SQLException;**

Este import también es parte de java.sql y trae la clase SQLException. SQLException es una excepción que se lanza cuando ocurre un error relacionado con la base de datos durante la ejecución de una operación SQL. En el código, se utiliza para manejar posibles errores de conexión o consultas en la base de datos.

**Código:**

**Atributos y Componentes GUI:**

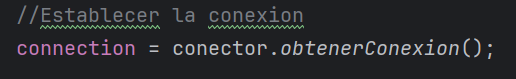
La clase contiene atributos que representan componentes de la interfaz de usuario, como JTextField, JButton y JTable.

Los componentes incluyen campos de texto para ingresar información sobre productos (ID, tipo, nombre, marca, cantidad, precio), botones para agregar, actualizar, buscar, eliminar y regresar, y una tabla para mostrar datos de productos.

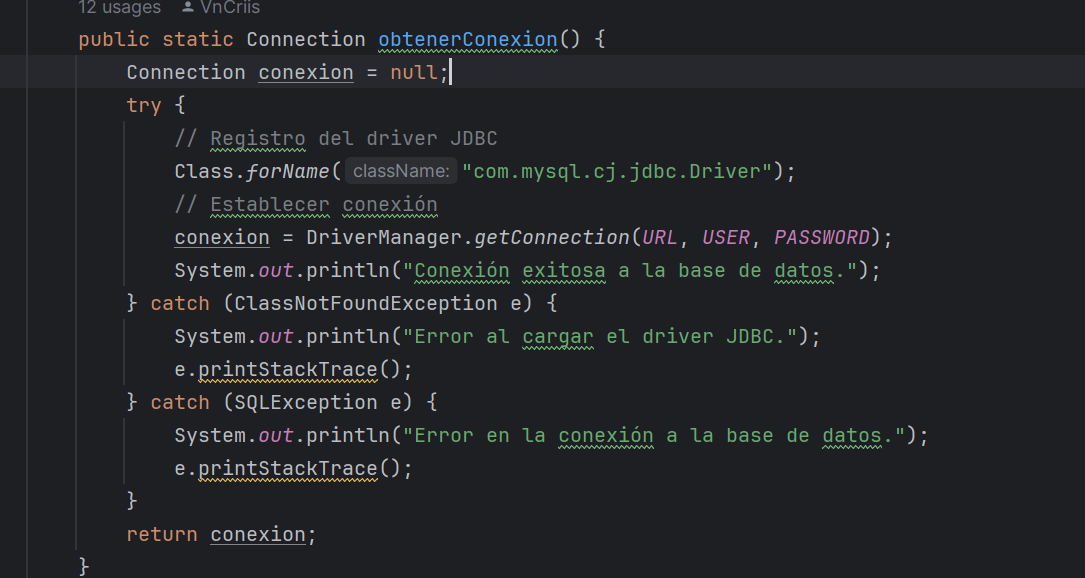


**Conexión a la Base de Datos:**

La clase establece una conexión a una base de datos mediante la instancia de un objeto Connection. La conexión es obtenida utilizando un objeto llamado conector, que probablemente sea una clase o método externo proporcionado para gestionar conexiones a la base de datos.



En donde se llama a la función “Obtener Conexión ” que se encuentren en otra clase para que la conexión sea mas optimizada en las clases necesarias.



**Explicación:**

**public static Connection obtenerConexion() {**

El método es público, estático (puede ser invocado sin crear una instancia de la clase) y devuelve un objeto Connection que representa la conexión a la base de datos.

**Connection conexion = null;**

Se declara una variable conexion de tipo Connection e inicializada como nula.

**try {**

Inicia un bloque try para manejar posibles excepciones.

**Class.forName("com.mysql.cj.jdbc.Driver");**

Registra el controlador JDBC de MySQL. En este caso, se utiliza el controlador com.mysql.cj.jdbc.Driver para la versión Connector/J de MySQL.

**conexion = DriverManager.getConnection(URL, USER, PASSWORD);**

Establece la conexión a la base de datos utilizando el método getConnection de DriverManager. Los parámetros URL, USER, y PASSWORD deben estar previamente definidos y contener la información necesaria para conectarse a la base de datos MySQL.

**System.out.println("Conexión exitosa a la base de datos.");**

Imprime un mensaje en la consola indicando que la conexión a la base de datos fue exitosa.

**} catch (ClassNotFoundException e) {**

Captura y maneja la excepción ClassNotFoundException, que puede ocurrir si el controlador JDBC no puede ser cargado.

**System.out.println("Error al cargar el driver JDBC.");**

Imprime un mensaje en la consola indicando que hubo un error al cargar el controlador JDBC.

**} catch (SQLException e) {**

Captura y maneja la excepción SQLException, que puede ocurrir si hay un error durante la conexión a la base de datos.

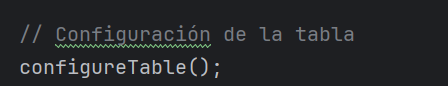
**System.out.println("Error en la conexión a la base de datos.");**

Imprime un mensaje en la consola indicando que hubo un error en la conexión a la base de datos.

**} return conexion;**

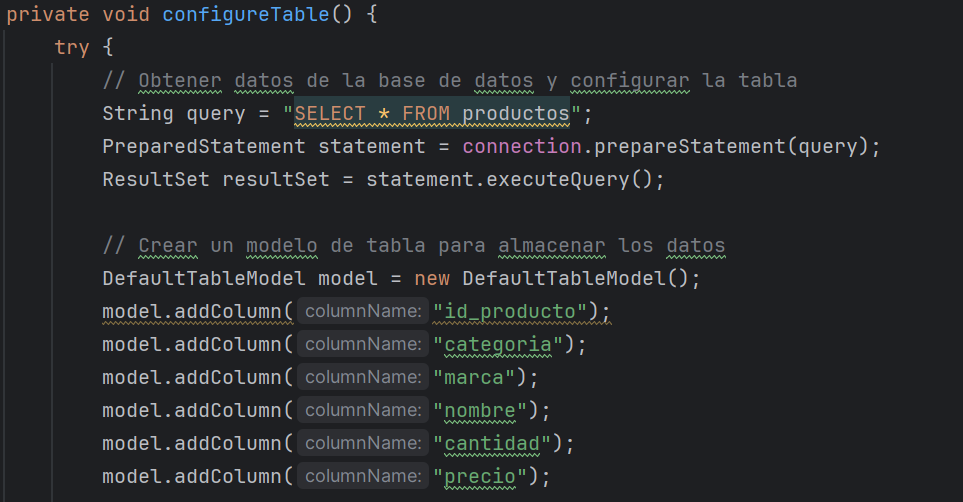
Retorna el objeto Connection. Si la conexión es exitosa, este objeto contendrá la conexión a la base de datos; de lo contrario, será nulo.

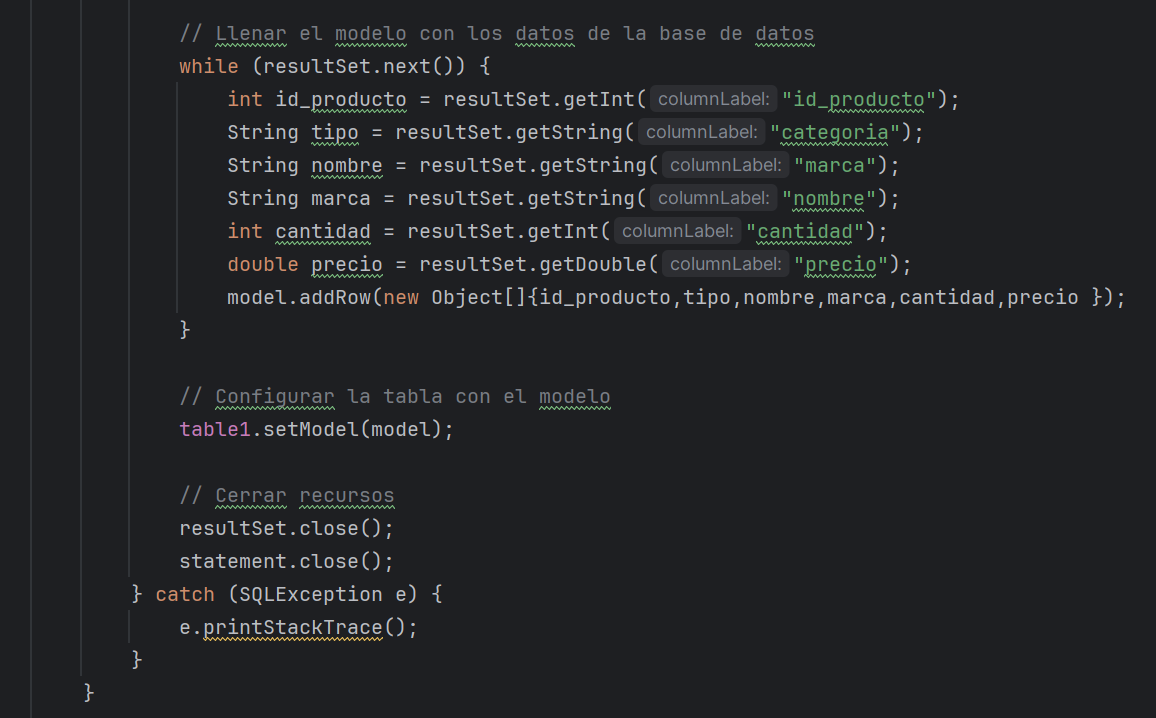
**Configuración de la Tabla:**

****

El método configureTable() se encarga de obtener datos de la tabla "productos" en la base de datos y llenar una tabla en la interfaz gráfica con esta información utilizando un DefaultTableModel.

Código de la función:





**Explicación:**

**Conexión a la Base de Datos:**

Se realiza una conexión a la base de datos utilizando la conexión almacenada en la variable connection.

**Consulta SQL y Ejecución:**

Se crea una consulta SQL para seleccionar todos los registros de la tabla "productos".

Se prepara y ejecuta la consulta SQL mediante un objeto PreparedStatement.

**Creación del Modelo de la Tabla:**

Se instancia un objeto DefaultTableModel que servirá como el modelo para la tabla.

Se agregan columnas al modelo, representando los campos de la tabla "productos" como "id\_producto", "categoria", "marca", "nombre", "cantidad" y "precio".

**Llenado del Modelo con Datos de la Base de Datos:**

Se recorre el conjunto de resultados (resultSet) obtenido de la ejecución de la consulta SQL.

Por cada fila, se extraen los valores de los campos "id\_producto", "categoria", "marca", "nombre", "cantidad" y "precio".

Se añade una nueva fila al modelo con estos valores.

**Configuración de la Tabla con el Modelo:**

Se configura la tabla (table1) con el modelo creado, lo que implica mostrar los datos en la interfaz gráfica.

**Cierre de Recursos:**

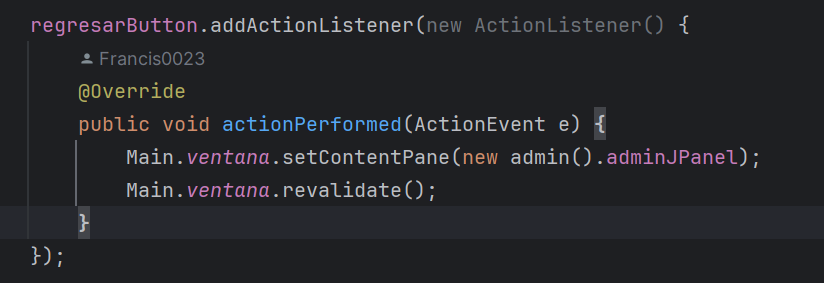
Se cierran los recursos, como el conjunto de resultados y el objeto PreparedStatement, para liberar memoria y evitar posibles problemas de recursos.

**Manejo de Excepciones:**

Se maneja la excepción SQLException, imprimiendo la traza de la pila en caso de que ocurra un error durante la ejecución de la consulta SQL.

**Manejo de Eventos:**

La clase define varios ActionListener para manejar eventos de los botones en la interfaz. Por ejemplo, agregarButton llama al método agregarDatos() cuando se presiona, buscarButton llama a buscarProducto(). Como:



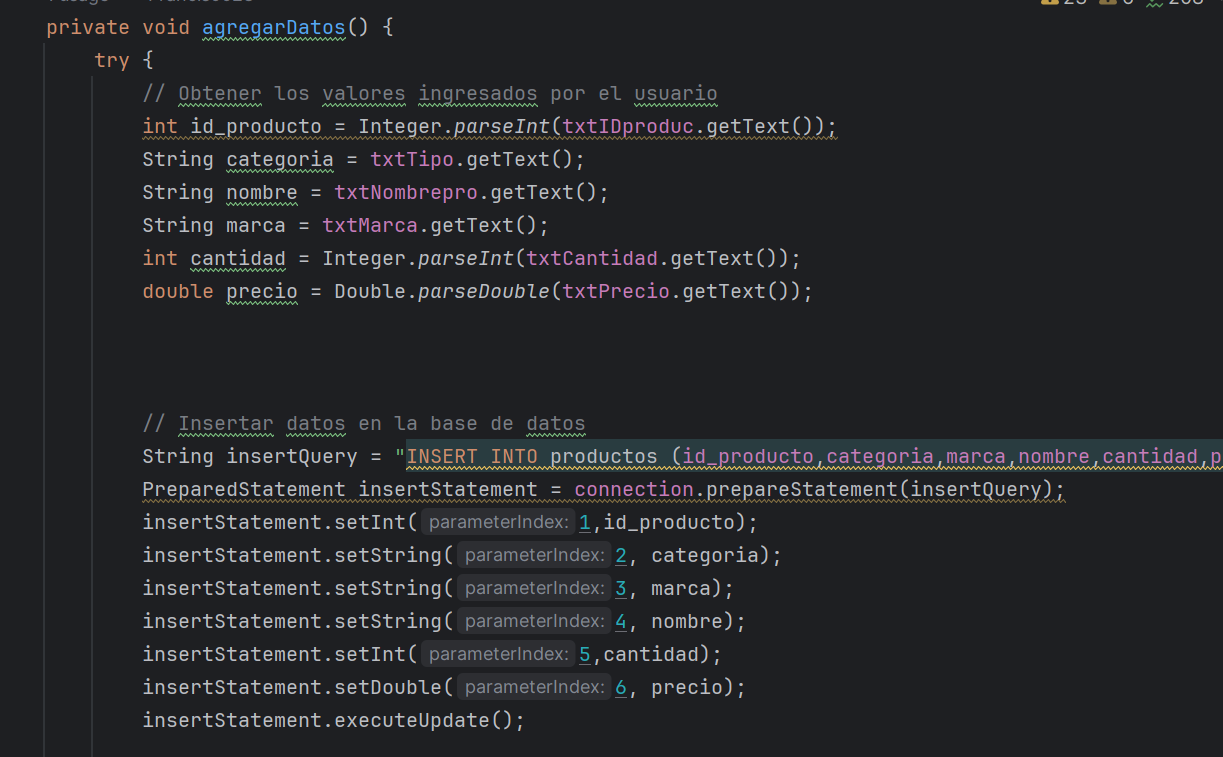
**Explicación:**

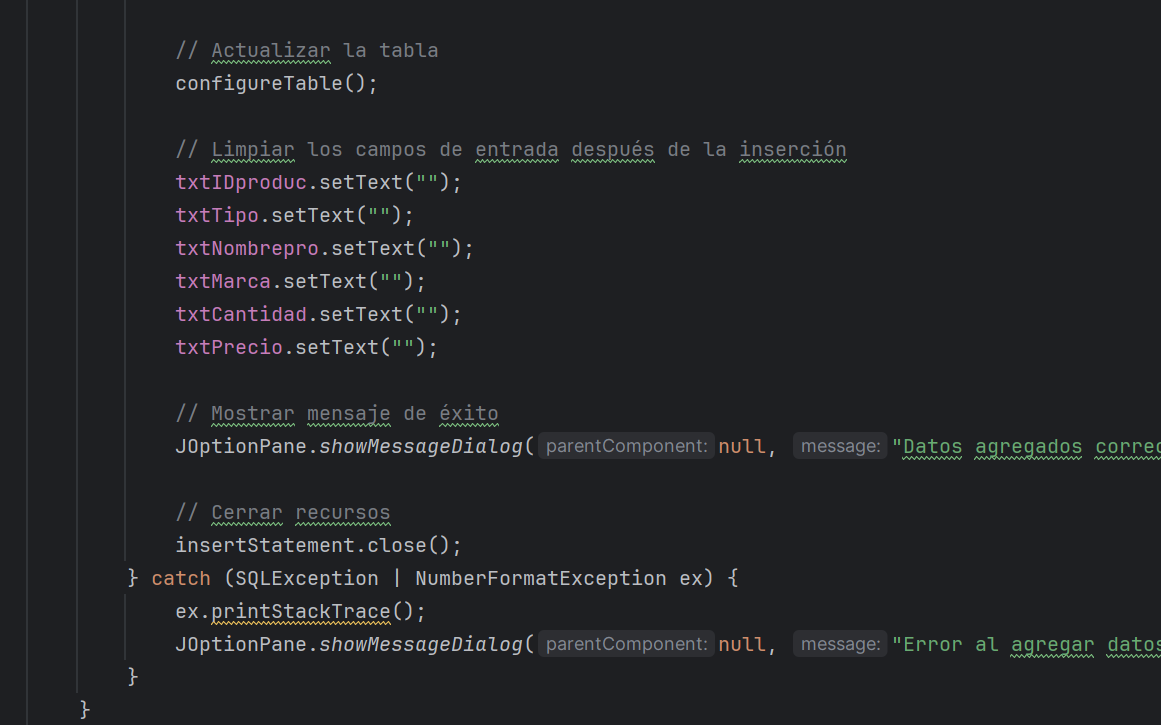
Este fragmento de código está asociado a un botón de regreso en una interfaz gráfica. Cuando el botón es presionado, se ejecuta el código dentro del bloque actionPerformed. En este caso, la acción consiste en cambiar el contenido principal de la ventana (Main.ventana) a un nuevo panel, específicamente al panel de administrador (admin().adminJPanel). Además, se utiliza el método revalidate() para actualizar y volver a validar la interfaz gráfica con el nuevo contenido.

**Operaciones CRUD:**

Métodos como agregarDatos(), buscarProducto(), actualizarProducto() y eliminarProducto() realizan operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en la base de datos según las interacciones del usuario. Estas operaciones incluyen inserción, búsqueda, actualización y eliminación de datos de productos.

**Opción de agregar:**





**Explicación:**

**Captura de Datos del Usuario:**

Se obtienen los valores ingresados por el usuario desde diferentes campos de entrada de texto en la interfaz gráfica.

**Preparación de la Inserción:**

Se prepara una consulta SQL parametrizada para insertar datos en la tabla "productos". La consulta utiliza placeholders (?) para representar los valores que se ingresarán más adelante.

**Creación de la Declaración Preparada:**

Se crea un objeto PreparedStatement a partir de la conexión a la base de datos y la consulta preparada.

**Asignación de Parámetros:**

Los valores obtenidos del usuario se asignan a los parámetros correspondientes en la consulta preparada.

**Ejecución de la Inserción:**

La consulta preparada se ejecuta utilizando executeUpdate(), lo que lleva a la inserción de los nuevos datos en la tabla.

**Actualización de la Tabla en la GUI:**

Se llama al método configureTable() para actualizar la tabla en la interfaz gráfica y reflejar los cambios realizados en la base de datos.

**Limpiar Campos de Entrada:**

Se eliminan los datos ingresados por el usuario en los campos de entrada de texto para preparar la interfaz para una nueva entrada.

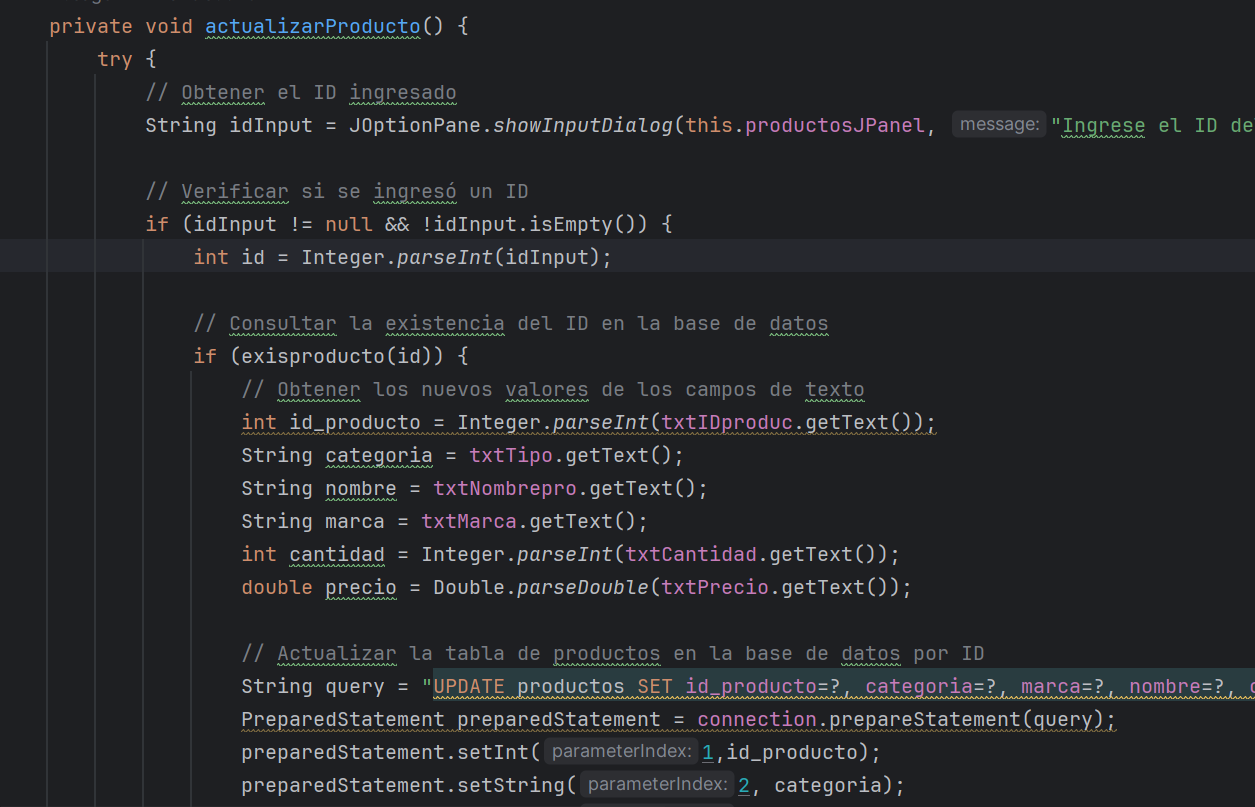
**Mensaje de Éxito:**

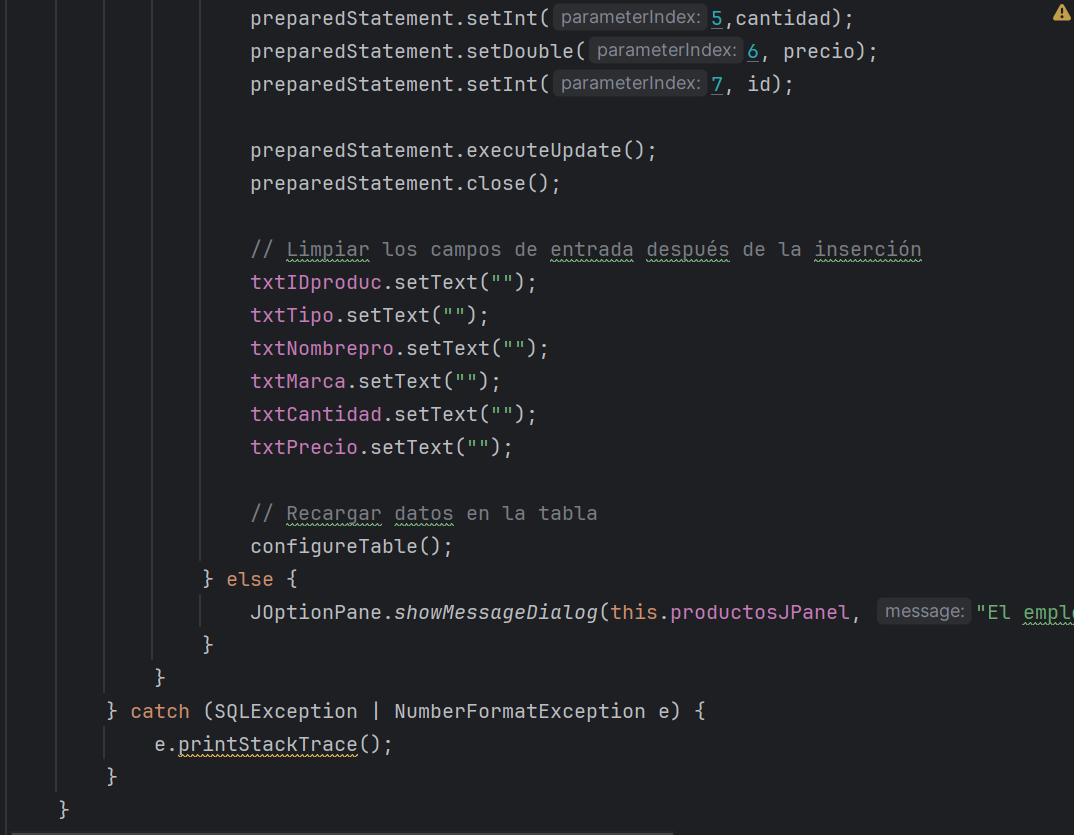
Se muestra un mensaje de éxito a través de un cuadro de diálogo JOptionPane indicando que los datos se han agregado correctamente.

**Manejo de Excepciones:**

Se manejan posibles excepciones, como errores de formato (NumberFormatException) o problemas con la base de datos (SQLException). En caso de error, se imprime la traza de la excepción y se muestra un mensaje de error.

**Actualización de productos**

****

****

**Solicitud de ID al Usuario:**

Muestra un cuadro de diálogo (JOptionPane.showInputDialog()) solicitando al usuario que ingrese el ID del producto que desea actualizar.

**Verificación de la Existencia del ID:**

Verifica si se ingresó un ID y si existe en la base de datos (exisproducto(id)). Si el ID no es nulo y existe, continúa con la actualización.

**Obtención de Nuevos Valores:**

Obtiene los nuevos valores de los campos de texto de la interfaz gráfica, como ID, categoría, nombre, marca, cantidad y precio.

**Actualización en la Base de Datos:**

Prepara y ejecuta una consulta SQL de actualización (UPDATE) en la base de datos para modificar el registro del producto con el ID proporcionado. Utiliza un objeto PreparedStatement para manejar consultas parametrizadas.

**Limpieza de Campos de Entrada:**

Limpia los campos de entrada en la interfaz gráfica después de la actualización para proporcionar una experiencia de usuario limpia.

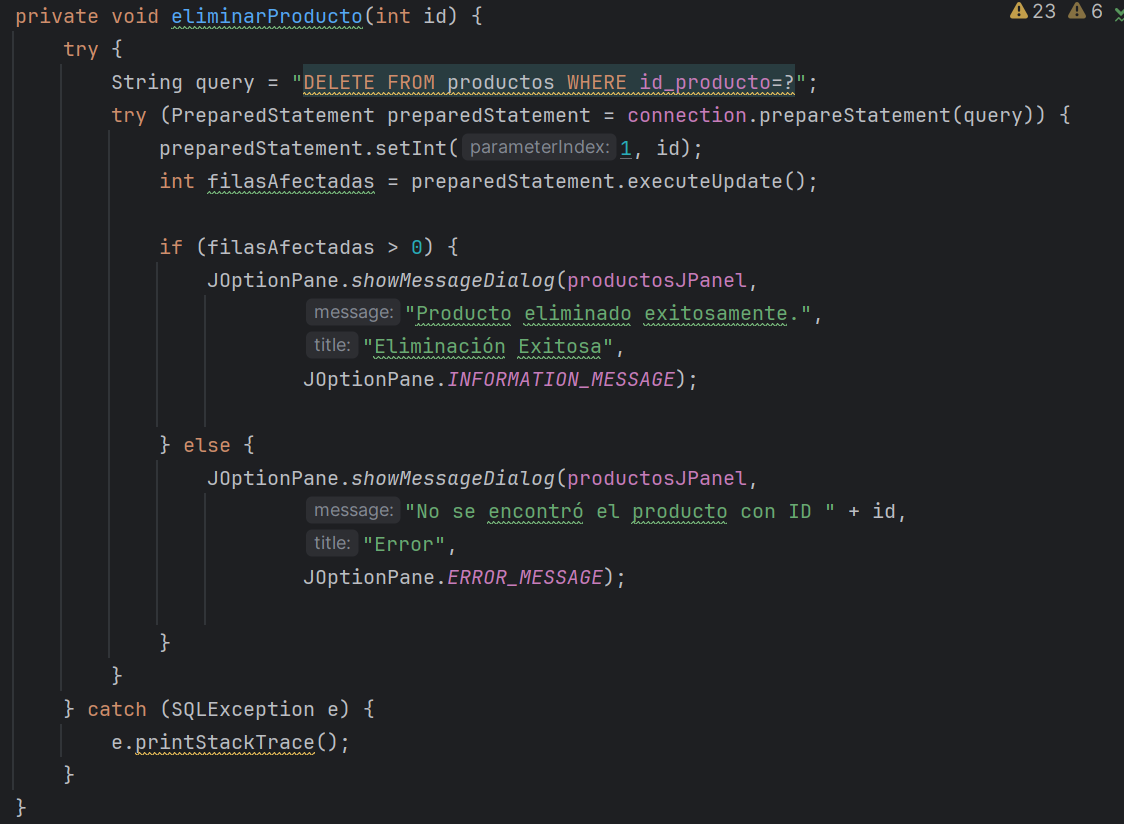
**Recarga de Datos en la Tabla:**

Vuelve a cargar los datos en la tabla (configureTable()) para reflejar los cambios actualizados en la interfaz gráfica.

**Manejo de Excepciones:**

Maneja excepciones como SQLException y NumberFormatException que podrían ocurrir durante la ejecución y las imprime en la consola.

**Borrar Productos**

****

**Explicación:**

**id\_producto.** Aquí tienes una explicación esencial del código:

**private void eliminarProducto(int id):** Este método es privado (private), lo que significa que solo puede ser accedido desde dentro de la misma clase. Toma un parámetro id, que representa el identificador único del producto que se va a eliminar.

**try { ... } catch (SQLException e) { e.printStackTrace(); }:** El código está envuelto en un bloque try-catch para manejar posibles excepciones de tipo SQLException. Las excepciones SQL generalmente ocurren cuando hay problemas de conexión con la base de datos o al ejecutar consultas SQL.

**String query = "DELETE FROM productos WHERE id\_producto**=?";: Se define una cadena de consulta SQL que borra el registro de la tabla productos donde el id\_producto es igual al parámetro proporcionado.

**try (PreparedStatement preparedStatement = connection.prepareStatement(query)) { ... }:** Se utiliza un PreparedStatement para ejecutar la consulta SQL preparada. Los PreparedStatement son útiles para evitar inyecciones de SQL y mejorar la seguridad de las consultas.

**preparedStatement.setInt(1, id**);: Se establece el valor del parámetro en la consulta preparada. En este caso, se asigna el valor del id proporcionado como argumento al PreparedStatement.

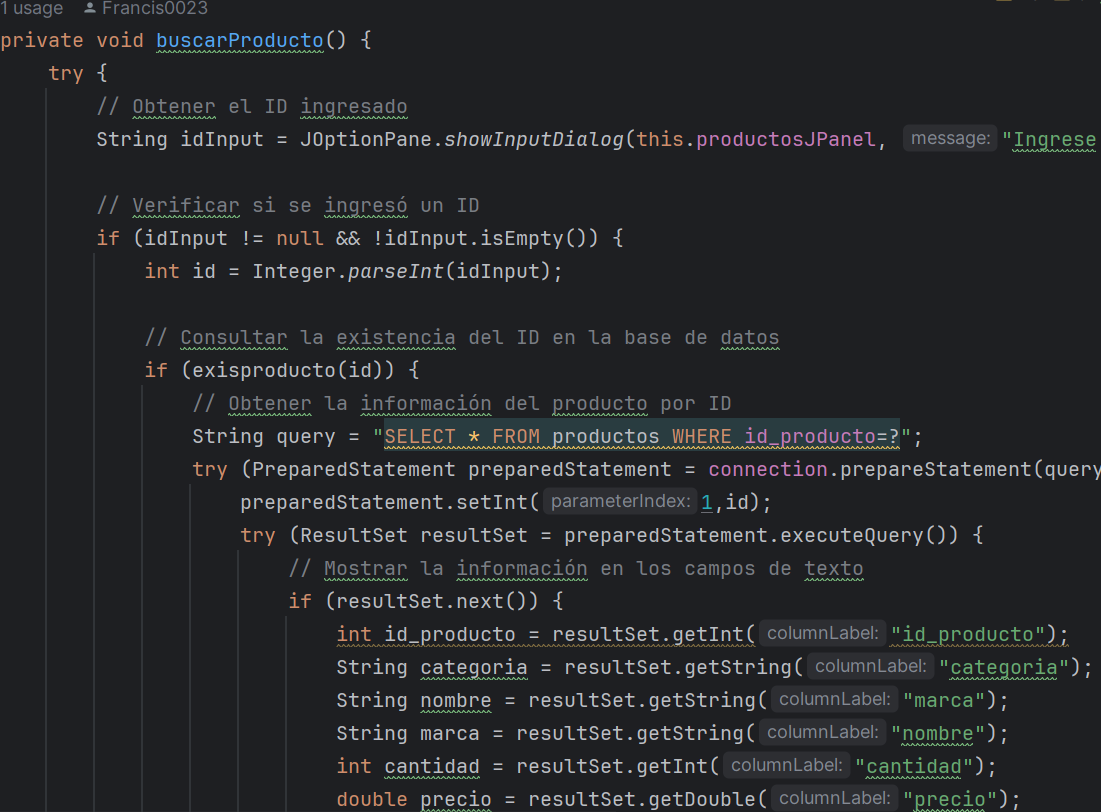
**int filasAfectadas = preparedStatement.executeUpdate();**: Se ejecuta la actualización de la base de datos y se obtiene el número de filas afectadas. Si el número de filas afectadas es mayor que cero, significa que se eliminó al menos un registro.

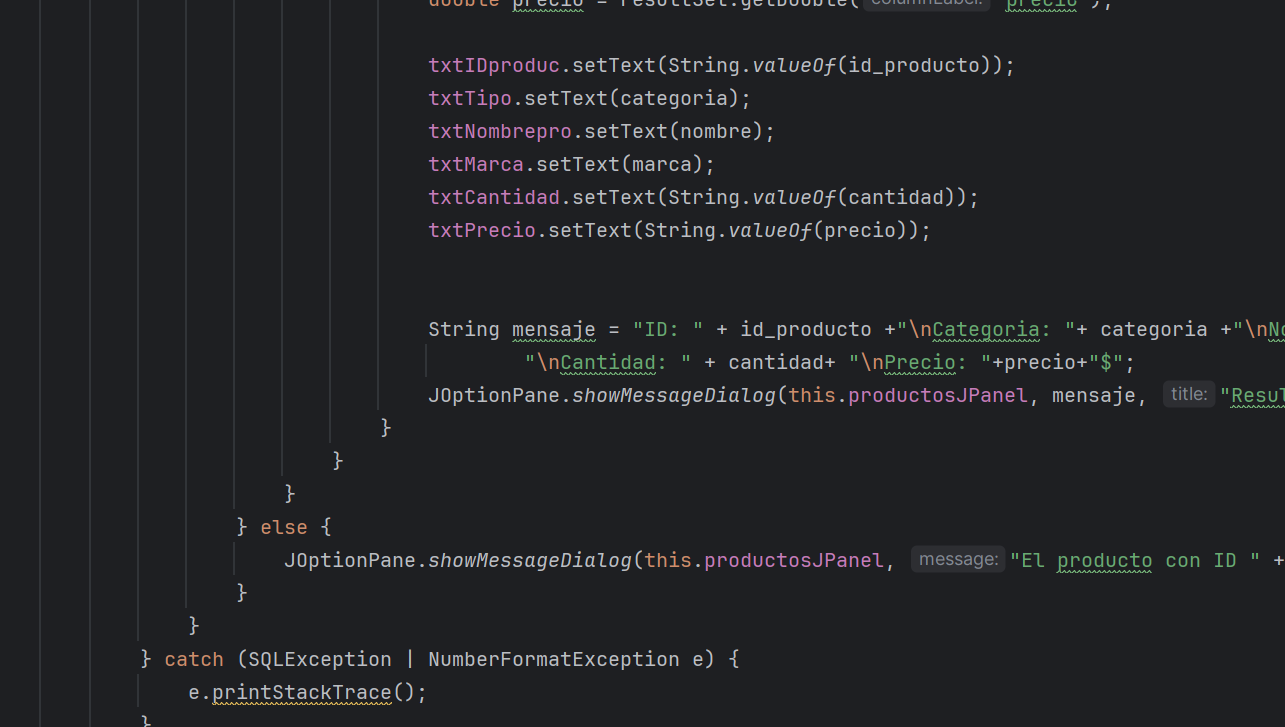
**if (filasAfectadas > 0) { ... } else { ... }:** Se realiza una verificación condicional para determinar si se eliminó con éxito algún producto. Si se eliminó al menos una fila, se muestra un mensaje de eliminación exitosa. De lo contrario, se muestra un mensaje indicando que no se encontró el producto con el id proporcionado.

**JOptionPane.showMessageDialog(...);:** Se utiliza JOptionPane para mostrar mensajes emergentes en la interfaz gráfica. En este caso, se muestra un mensaje informativo en caso de éxito y un mensaje de error en caso de que el producto no se encuentre en la base de datos.

**} catch (SQLException e) { e.printStackTrace(); }:** En caso de que se produzca una excepción SQL durante la ejecución de la consulta, se imprime la traza de la excepción. Esto puede ser útil para depurar problemas de conexión o consultas incorrectas.

**Opción de buscar**

****

****

**Explicación:**

**Solicitud de ID al Usuario:**

Se utiliza JOptionPane.showInputDialog para solicitar al usuario que ingrese un ID de producto. La ventana de entrada tiene un mensaje descriptivo y un campo para ingresar el ID.

**Validación del ID Ingresado:**

Se verifica si se ingresó un ID válido. Si el usuario presiona cancelar o no ingresa ningún valor, se evita realizar la búsqueda.

**Conversión del ID a Entero:**

El ID ingresado se convierte a un entero utilizando Integer.parseInt.

**Consulta de Existencia del Producto:**

Se llama a la función exisproducto(id) para verificar si el producto con el ID ingresado existe en la base de datos.

**Consulta SQL para Obtener Información:**

Si el producto existe, se ejecuta una consulta SQL preparada para obtener la información del producto desde la base de datos.

**Mostrar Información en los Campos de Texto:**

Si la consulta devuelve resultados, se extraen los datos del ResultSet y se muestran en campos de texto de la interfaz gráfica.

**Visualización del Resultado:**

Se crea un mensaje informativo con la información del producto encontrado y se muestra mediante un cuadro de diálogo JOptionPane.

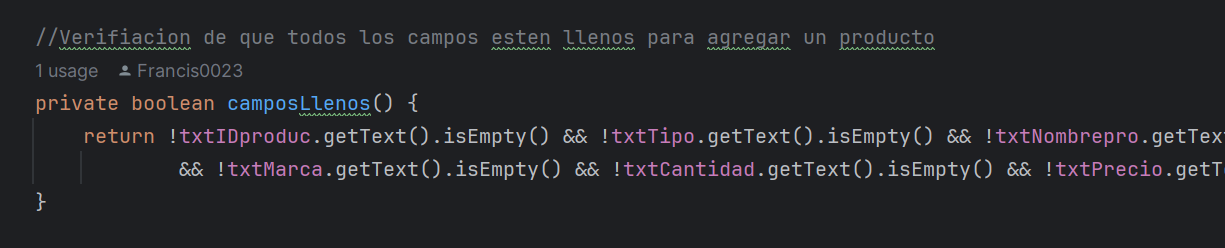
**Manejo de Errores:**

Se utiliza un bloque try-catch para manejar posibles excepciones. En particular, se capturan excepciones de tipo SQLException y NumberFormatException. Cualquier excepción se imprime en la consola.

**Aplicación de funciones adicionales**

**Validación de Campos:**

Se implementa una validación (camposLlenos()) para asegurarse de que todos los campos requeridos estén llenos antes de agregar un nuevo producto.



**Explicación**:

**txtIDproduc.getText().isEmpty():** Verifica si el campo de texto txtIDproduc está vacío. Si el campo está vacío, retorna false; de lo contrario, retorna true.

**txtTipo.getText().isEmpty():** Similar al anterior, verifica si el campo de texto txtTipo está vacío.

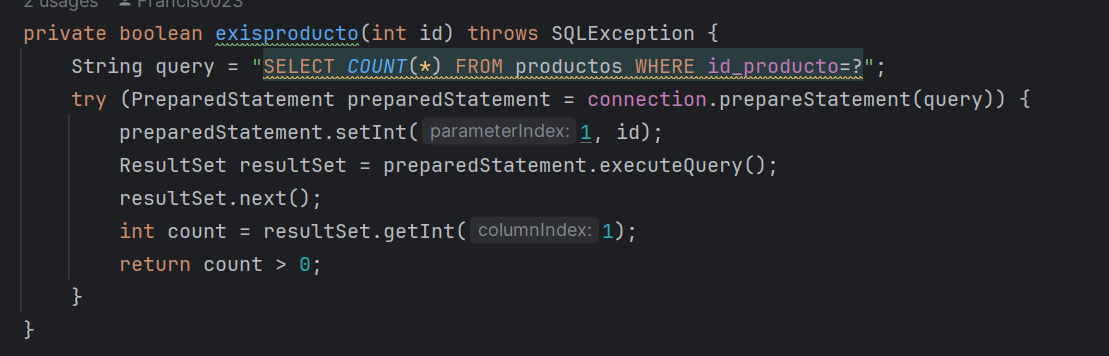
**txtNombrepro.getText().isEmpty():** Igual que antes, verifica si el campo de texto txtNombrepro está vacío.

**txtMarca.getText().isEmpty():** Verifica si el campo de texto txtMarca está vacío.

**txtCantidad.getText().isEmpty():** Comprueba si el campo de texto txtCantidad está vacío.

**txtPrecio.getText().isEmpty():** Verifica si el campo de texto txtPrecio está vacío.

**Función de si existe un producto en la base de datos:**



**Explicación:**

**private boolean exisproducto(int id) throws SQLException {**

Este método es privado, devuelve un valor booleano (true si el producto existe, false si no) y toma como parámetro un identificador de producto (id). Puede lanzar una excepción del tipo SQLException en caso de errores de conexión o consulta SQL.

**String query = "SELECT COUNT(\*) FROM productos WHERE id\_producto=?";**

Se define una cadena query que representa una consulta SQL. En este caso, se cuenta el número de filas en la tabla productos donde el valor de la columna id\_producto coincide con el parámetro proporcionado (id).

**try (PreparedStatement preparedStatement = connection.prepareStatement(query)) {**

Se inicia un bloque try-with-resources para garantizar que los recursos (como PreparedStatement) se cierren adecuadamente. Se prepara una declaración SQL con la conexión establecida (connection) y la consulta definida anteriormente.

**preparedStatement.setInt(1, id);**

Se establece el valor del parámetro en la consulta preparada. En este caso, se asigna el valor del identificador de producto proporcionado (id) al primer marcador de posición (?) en la consulta.

**ResultSet resultSet = preparedStatement.executeQuery();**

Se ejecuta la consulta y se obtiene un conjunto de resultados (ResultSet) que contiene el resultado de la operación.

**resultSet.next();**

Se mueve al primer (y en este caso, único) resultado en el conjunto de resultados. En este contexto, se espera que el resultado sea un valor entero que representa la cuenta de productos encontrados.

**int count = resultSet.getInt(1);**

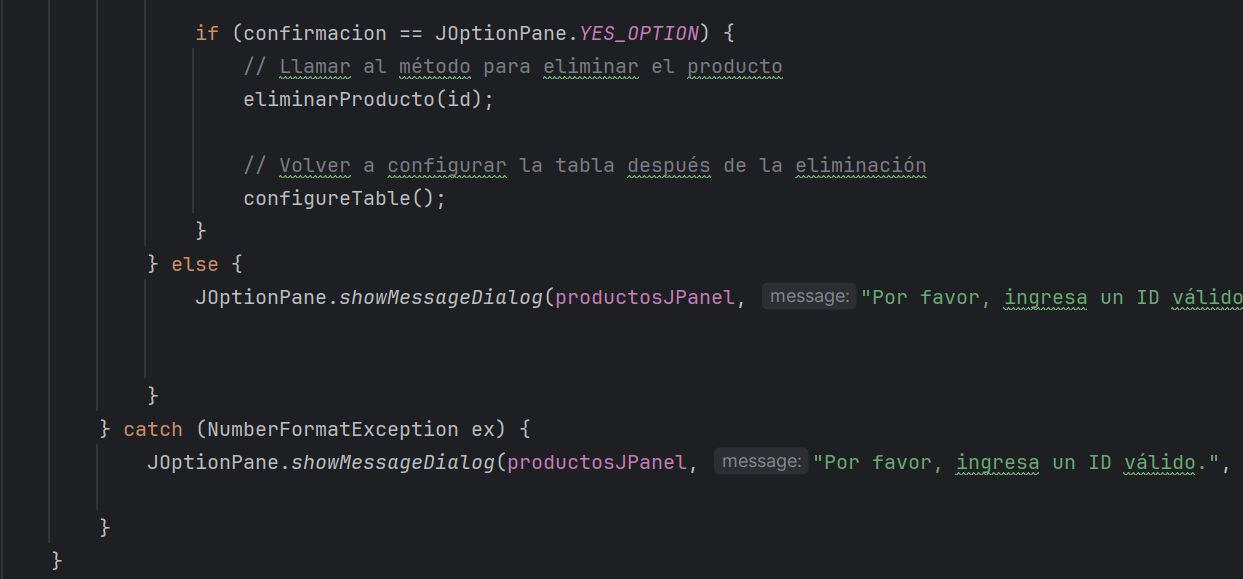
Se obtiene el valor entero de la primera columna (1) del resultado. Este valor representa el conteo de productos que cumplen con la condición de la consulta.

**return count > 0;**

El método devuelve true si el conteo es mayor que cero, lo que significa que al menos un producto con el identificador proporcionado existe en la base de datos. De lo contrario, devuelve false. Este resultado se utiliza para determinar si el producto existe o no en la base de datos.

**Función de confirmar eliminación:**

****

****

**Explicación:**

**Solicitar ID para Eliminación:**

Se utiliza JOptionPane.showInputDialog para solicitar al usuario que ingrese el ID del producto que desea eliminar. El diálogo muestra un mensaje, un campo de entrada y un ícono de pregunta.

**Validación del ID Ingresado:**

Se verifica si el usuario ingresó un ID válido. Si el campo de entrada no está vacío, se procede a convertir el ID a un valor entero usando Integer.parseInt.

**Confirmación de Eliminación:**

Se utiliza JOptionPane.showConfirmDialog para mostrar un cuadro de diálogo de confirmación. Este cuadro pregunta al usuario si está seguro de eliminar el producto con el ID proporcionado.

**Proceso de Eliminación:**

Si el usuario elige "Sí" en el cuadro de diálogo de confirmación (JOptionPane.YES\_OPTION), se llama al método eliminarProducto(id) para ejecutar la eliminación en la base de datos.

**Actualización de la Tabla:**

Después de la eliminación, se llama al método configureTable para volver a configurar la tabla que muestra los productos, actualizando así la interfaz gráfica con la base de datos actualizada.

**Manejo de Errores:**

Se maneja la excepción NumberFormatException para capturar posibles errores cuando se intenta convertir el ID ingresado a un entero. Si ocurre un error, se muestra un mensaje de error utilizando JOptionPane.showMessageDialog.

**Otros aspectos importantes serian lo siguiente:**

**Interfaz de Usuario Dinámica:**

La interfaz de usuario se actualiza dinámicamente después de realizar operaciones como agregar, actualizar o eliminar productos. La tabla se vuelve a configurar para reflejar los cambios en la base de datos.

**Integración con Otras Clases o Métodos:**

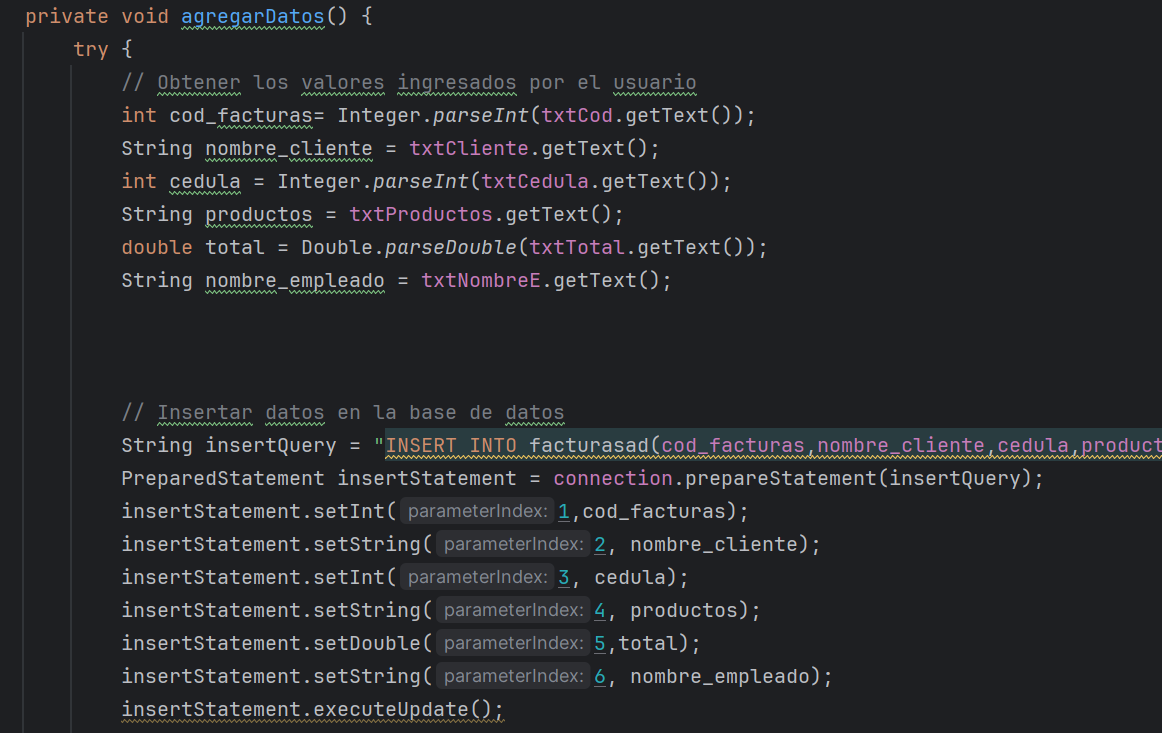
La clase hace referencia a otras clases o métodos externos como conector, Main, y admin. La integración con estas entidades no se proporciona en el código compartido.

**Documentación Clase Facturas AD**

Igualmente, que en la anterior clase se necesitan las diferentes importaciones y algo similar en la utilización de los datos, ya que en esta clase se va a ingresar y buscar los datos.

Tal como la anterior clase, se va a necesitar un conector para llamar a la clase que contiene la conexión y a la función de configuración de la tabla para mostrar la información que se esta guardando.

**Función de Agregar una factura:**

****



**Explicación:**

**Explicación:**

**Obtención de valores ingresados por el usuario:**

Se extraen los valores ingresados en componentes de interfaz de usuario, como cajas de texto (txtCod, txtCliente, etc.). Estos valores representan la información de una factura, como el código de factura, nombre del cliente, cédula, productos, total y nombre del empleado.

**Inserción de datos en la base de datos:**

Se utiliza una consulta preparada (PreparedStatement) para insertar los valores obtenidos en la base de datos. La consulta SQL está diseñada para insertar los valores en columnas específicas de la tabla facturasad.

**Actualización de la tabla en la interfaz gráfica:**

Se llama al método configureTable() para actualizar la tabla en la interfaz gráfica con los nuevos datos agregados. Este método se encarga de obtener la información actualizada de la base de datos y refrescar la tabla.

**Limpieza de los campos de entrada:**

Después de la inserción exitosa, se limpian los campos de entrada (txtCod, txtCliente, etc.) para preparar la interfaz para una nueva entrada de datos.

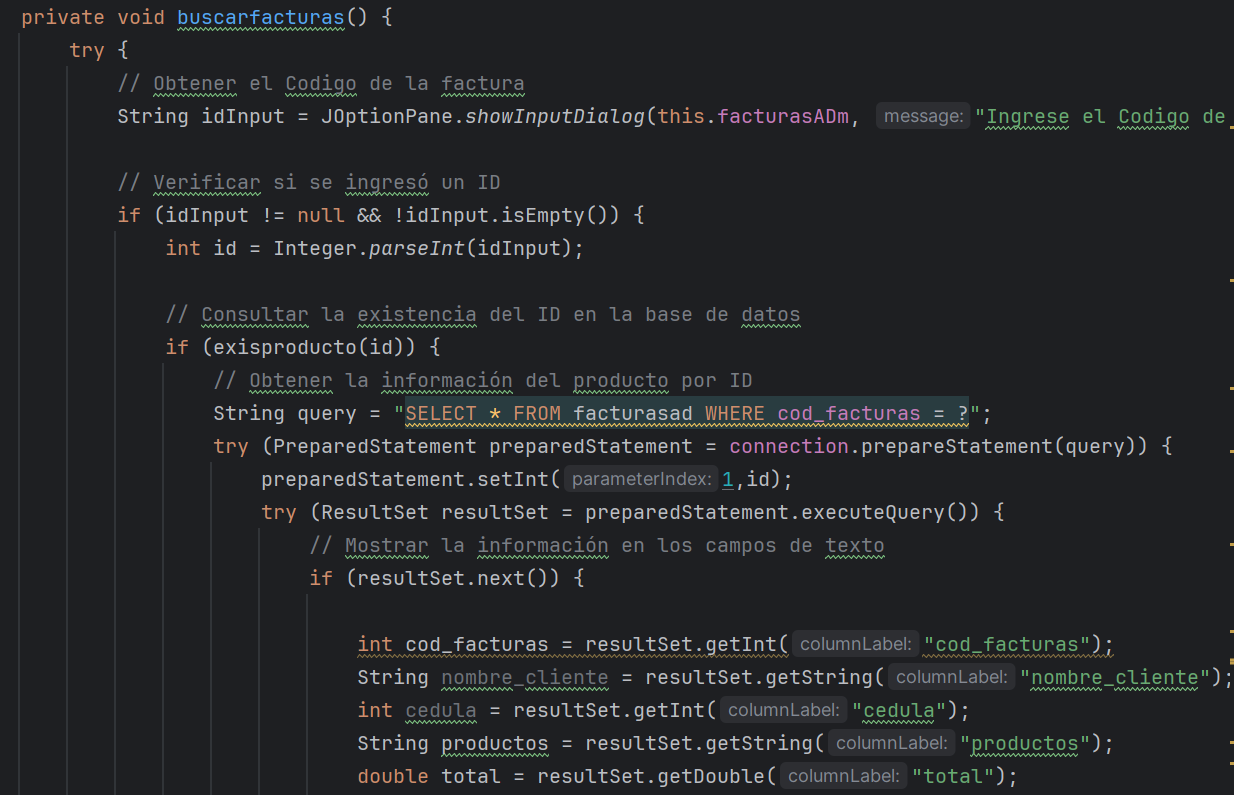
**Mensaje de éxito:**

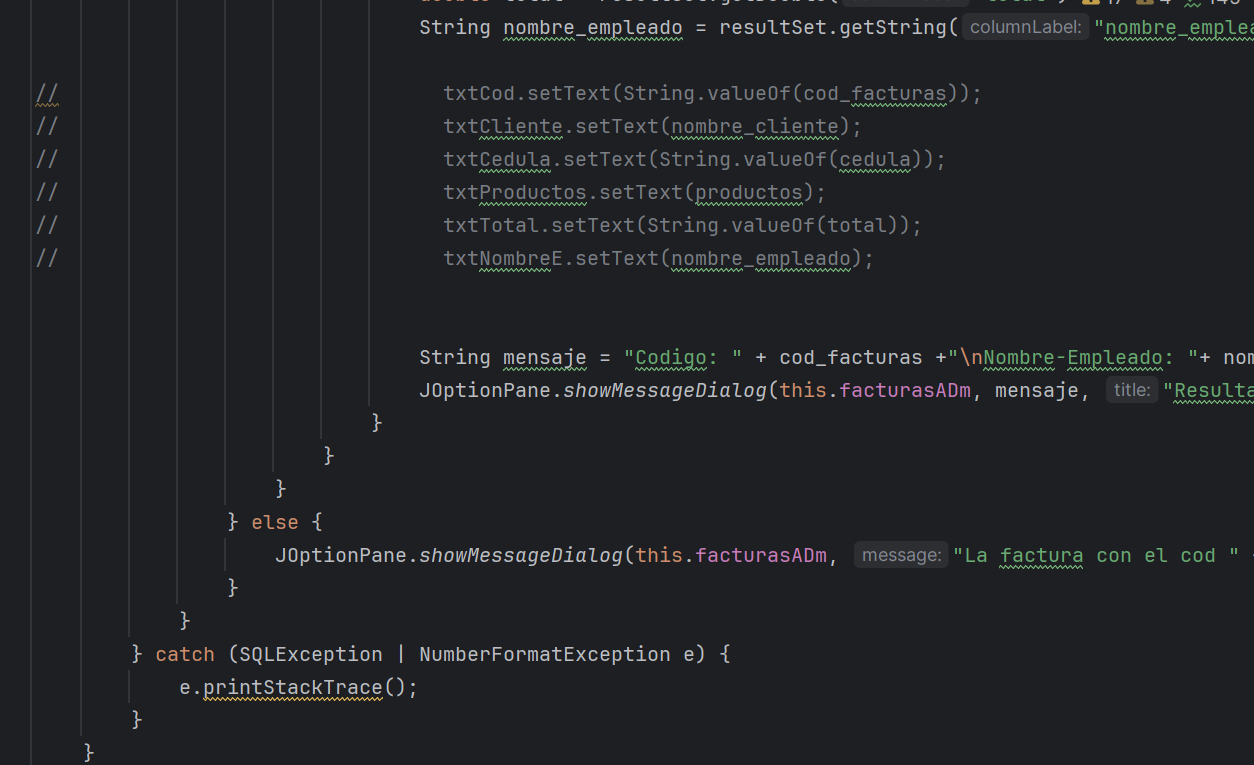
Se muestra un cuadro de diálogo (JOptionPane) indicando que los datos se han agregado correctamente en caso de éxito.

**Manejo de excepciones:**

Se utiliza un bloque try-catch para manejar excepciones. En caso de errores, se imprime la traza de la excepción (ex.printStackTrace()) y se muestra un cuadro de diálogo de error (JOptionPane.ERROR\_MESSAGE).

Función para buscar facturas:





**Explicación:**

**Solicitud de Código de Factura:**

La función comienza solicitando al usuario que ingrese el código de la factura mediante un cuadro de diálogo de entrada (JOptionPane.showInputDialog).

**Verificación del Código Ingresado:**

Se verifica si el usuario ingresó un código (no es nulo y no está vacío). Si no se ingresó un código, la búsqueda se detiene.

**Conversión del Código a Entero:**

El código ingresado se convierte de String a int utilizando Integer.parseInt.

**Consulta de la Existencia del Código en la Base de Datos:**

Se verifica si el código de factura ingresado existe en la base de datos utilizando la función exisproducto(id). Esta función no está visible en el código proporcionado, pero asumimos que verifica la existencia del código en la base de datos.

**Consulta de la Información de la Factura en la Base de Datos:**

Si el código existe, se realiza una consulta SQL para obtener la información de la factura desde la tabla facturasad. La consulta utiliza un PreparedStatement con un parámetro (cod\_facturas = ?) para evitar ataques de inyección SQL.

**Mostrar la Información de la Factura:**

Si se encuentra una factura con el código proporcionado, se extraen los datos (como el código, nombre del cliente, cédula, productos, total y nombre del empleado) del ResultSet y se construye un mensaje informativo.

**Cuadro de Diálogo de Resultado:**

Se muestra un cuadro de diálogo (JOptionPane.showMessageDialog) con la información de la factura obtenida de la base de datos.

**Manejo de Errores:**

Se capturan excepciones (SQLException y NumberFormatException) que podrían ocurrir durante la conversión del código a entero o la ejecución de consultas SQL. En caso de error, se imprime el rastreo de la pila.