Contenido

[Requisitos de Harward 3](#_Toc154590540)

[Requisitos de software 3](#_Toc154590541)

[Estructura de la base de datos 3](#_Toc154590542)

[Tabla de usuario 4](#_Toc154590543)

[Tabla ventas 4](#_Toc154590544)

[Tabla inventario 4](#_Toc154590545)

[Procedimientos almacenados 5](#_Toc154590546)

[Registrar Venta 5](#_Toc154590547)

[5](#_Toc154590548)

[Obtener inventario 5](#_Toc154590549)

[obtener-rendimiento-diario 5](#_Toc154590550)

[obtener-rendimiento-mes 6](#_Toc154590551)

[reporte-xproducto-mensual 7](#_Toc154590552)

[Pantalla de loging 8](#_Toc154590553)

[rbOcultarActionPerformed 9](#_Toc154590554)

[Botón de Loging 9](#_Toc154590555)

[Pantalla geUsuarios 10](#_Toc154590556)

[Botón limpiar 10](#_Toc154590557)

[Botón Registrar 11](#_Toc154590558)

[Botón Actualizar 12](#_Toc154590559)

[Botón Eliminar 13](#_Toc154590560)

[Clase producto 14](#_Toc154590561)

[Void borrarProducto 14](#_Toc154590562)

[void actualizar 15](#_Toc154590563)

[void agregarProducto 16](#_Toc154590564)

[productos buscarPorId 17](#_Toc154590565)

[void buscarPorId 19](#_Toc154590566)

[buscarPorId 20](#_Toc154590567)

[ArrayList<Integer> buscarPorMarca 20](#_Toc154590568)

[ArrayList<Integer> buscarPorNombre 22](#_Toc154590569)

[Pantalla gesProductos 22](#_Toc154590570)

[Botón buscar 23](#_Toc154590571)

[cmbResultItemStateChanged 23](#_Toc154590572)

[Botón agregar 24](#_Toc154590573)

[Botón borrar 25](#_Toc154590574)

[Botón editar 26](#_Toc154590575)

[Botón limpiar 26](#_Toc154590576)

[Clase Inventario 27](#_Toc154590577)

[void ingresoInvetario 27](#_Toc154590578)

[Void obtenerRestantes 28](#_Toc154590579)

[void egresoInvetario 29](#_Toc154590580)

[Pantalla gestInventario 30](#_Toc154590581)

[Botón guardar 31](#_Toc154590582)

[Botón revisarDatos 32](#_Toc154590583)

[32](#_Toc154590584)

[Clase Reporte 33](#_Toc154590585)

[String[] reporteGenera 33](#_Toc154590586)

[void reporteProProducto() 34](#_Toc154590587)

[Clase factura 35](#_Toc154590588)

[void addProductoInTable 35](#_Toc154590589)

[void RemoveroductoInTable 36](#_Toc154590590)

[void registrarVenta 36](#_Toc154590591)

[String obtenerDirectorioSeleccionado() 37](#_Toc154590592)

[void generarFactura 37](#_Toc154590593)

[Pantalla facturar 38](#_Toc154590594)

[Botón Agregar 39](#_Toc154590595)

[Botón Eliminar 39](#_Toc154590596)

[Botón generar 39](#_Toc154590597)

[Validación de campos general 39](#_Toc154590598)

[Validar parámetro id 40](#_Toc154590599)

# Requisitos de Harward

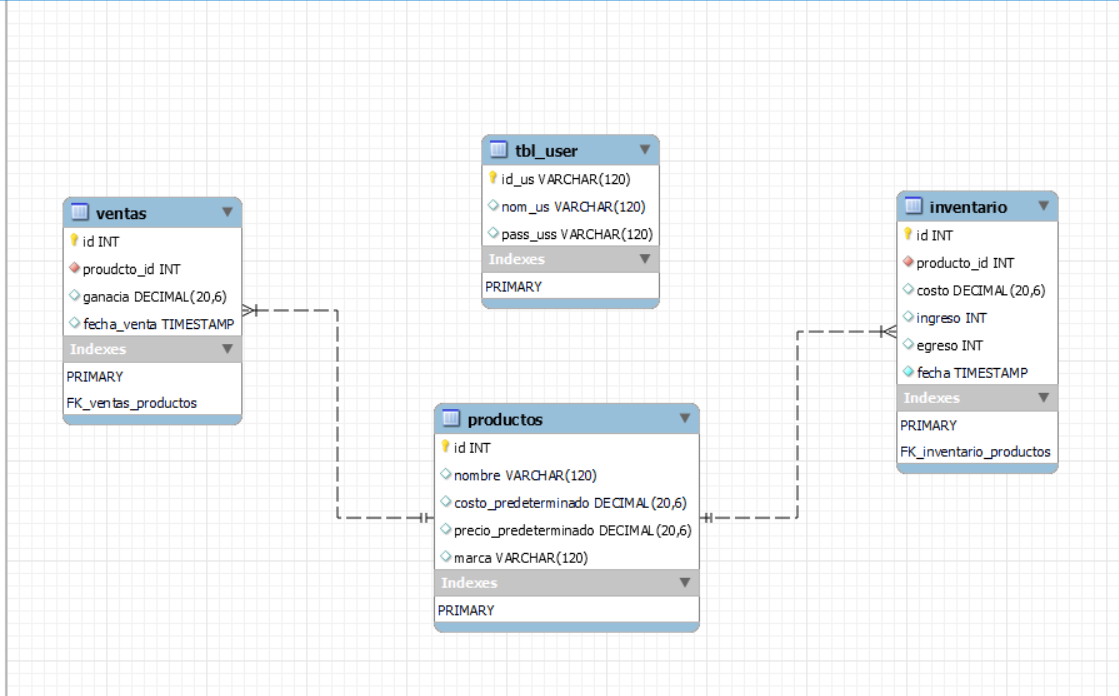
En la presente sección se detallan los requisitos de hardware recomendados para el desarrollo del sistema

|  |  |  |
| --- | --- | --- |
| Nombre | Descripción | Especificación |
| Procesador |  | I3, ryzen 3(mínimo)  I5 o ryzen 5(recomendado) |
| RAM |  | 8GB (mínimo)  12GB (recomendada) |
| Disco |  | 120GB (mínimo) |
| Sistema operativo |  | Windows 10 o 11 |

# Requisitos de software

|  |  |  |
| --- | --- | --- |
| Nombre | Descripción | Especificación |
| JDK | Versión del JDK | JDK 20 |
| Ide | Herramienta para escribir el código de java | Apache NetBeans |
| DB Administrador | Administrador de base de datos | Heidi SQL |
| DB motor | Motor de base de datos | MySQL |
| DB Administrador 2 | Administrador secundario  De base de datos | MySQL worbreanch |
| itext5-itextpdf | Librería para generar pdf | Versión 5.5.12 |
| mysql-connector | Librería para gestionar la conexión a mySql | Version 8.1.0 |

# Estructura de la base de datos

En esta sección se muestra y explica el diagrama de entidad relación de la base de datos

### Tabla de usuario

Esta tabla almacena la información de los usuarios del sistema. Los campos de la tabla son los siguientes:

id\_us: Identificador único del usuario.

nom\_us: Nombre del usuario.

pass\_uss: Contraseña del usuario.

### Tabla ventas

Esta tabla almacena la información de las ventas realizadas. Los campos de la tabla son los siguientes:

id: Identificador único de la venta. (con auto incremento)

proudcto\_id: Identificador del producto

ganacia: Ganancia obtenida en la venta.

fecha venta: Fecha de la venta.

### Tabla inventario

Esta tabla almacena la información del inventario de productos. Los campos de la tabla son los siguientes:

id: Identificador único de la operación del inventario(autoincremento)

producto id: Identificador del producto

costo: Costo del producto.

ingreso: Cantidad de productos ingresados al inventario.

egreso: Cantidad de productos egresados del inventario.

Cuando se ingresa nuevos productos se crea un registro en la tabla con

El id del producto ingresado ingreso: la cantidad de producto que se ingreso

Y costo el costo total del producto ingresado

Y cuando se retira productos se crea un registro con la id del producto a retirar y

Egreso la cantidad retirada

## Procedimientos almacenados

### Registrar Venta

### 

Este procedimiento almacenado registra una venta registrando el id del producto

Vendido y la ganancia de la venta y también registra un egreso del inventario registrando el producto id y la cantidad de producto que se vendió

### Obtener inventario

**BEGIN**

**SELECT**

**SUM**(costo) **AS** total\_costo,

**SUM**(ingreso - egreso) **AS** total\_restante,

producto\_id

**FROM**

inventario

**WHERE**

producto\_id = pro

**AND** fecha >= **DATE\_SUB**(**CURDATE**(), **INTERVAL** 365 **DAY**)

**GROUP** **BY**

producto\_id;

**END**

Este procedimiento obtiene información del inventario de un producto específico dentro del último año. Calcula el costo total y el total restante (considerando entradas y salidas) para el producto identificado por pro en la tabla inventario.

Función:

Consulta la tabla inventario para el producto especificado y registros dentro del último año.

Calcula el costo total y el total restante basado en las operaciones de entradas y salidas.

Agrupa los resultados por el producto.

### obtener-rendimiento-diario

BEGIN

SELECT

(SELECT SUM(costo) AS GastosTotales

FROM inventario

WHERE DATE(fecha) = CURDATE()) AS GastosTotales,

(SELECT SUM(ganacia) AS GananciaVentas

FROM ventas

WHERE DATE(fecha\_venta) = CURDATE()) AS GananciaVentas;

END

Este procedimiento realiza dos consultas separadas para obtener información específica de la base de datos relacionada con los gastos totales y las ganancias provenientes de las ventas del día actual (CURDATE()).

Lo que hace:

Realiza dos subconsultas para calcular:

GastosTotales: La suma de los costos (costo) del inventario para el día actual.

GananciaVentas: La suma de las ganancias (ganancia) de las ventas para el día actual.

Funcionamiento técnico:

Utiliza subconsultas para obtener valores sumados de las tablas inventario y ventas.

Filtra registros basados en la fecha actual utilizando CURDATE() y la función DATE() para comparar solo la parte de la fecha, no la hora.

Este procedimiento proporciona una manera rápida de obtener los gastos totales y las ganancias de las ventas para el día actual, permitiendo una visión instantánea de los datos financieros diarios.

### obtener-rendimiento-mes

**BEGIN**

**SELECT**

(**SELECT** **SUM**(costo) **AS** GastosTotales

**FROM** inventario

**WHERE** **MONTH**(fecha) = **MONTH**(**CURDATE**()) **AND** **YEAR**(fecha) = **YEAR**(**CURDATE**())) **AS** GastosTotales,

(**SELECT** **SUM**(ganacia) **AS** GananciaVentas

**FROM** ventas

**WHERE** **MONTH**(fecha\_venta) = **MONTH**(**CURDATE**()) **AND** **YEAR**(fecha\_venta) = **YEAR**(**CURDATE**())) **AS** GananciaVentas;

**END**

Lo mismo que el procedimiento anterior, pero filtra por el mes actual y no por el día actual

### reporte-xproducto-mensual

BEGIN

SELECT

p.nombre AS NombreProducto,

COALESCE(SUM(v.ganancia), 0) AS GananciaVentas,

COALESCE(SUM(i.costo), 0) AS CostoTotal,

COALESCE(SUM(i.ingreso - i.egreso), 0) AS CantidadRestante

FROM productos p

LEFT JOIN (

SELECT

producto\_id,

SUM(costo) AS costo,

SUM(ingreso) AS ingreso,

SUM(egreso) AS egreso

FROM inventario

WHERE YEAR(fecha) = YEAR(CURRENT\_DATE()) AND MONTH(fecha) = MONTH(CURRENT\_DATE())

GROUP BY producto\_id

) i ON p.id = i.producto\_id

LEFT JOIN (

SELECT

proudcto\_id,

SUM(ganacia) AS ganancia

FROM ventas

WHERE YEAR(fecha\_venta) = YEAR(CURRENT\_DATE()) AND MONTH(fecha\_venta) = MONTH(CURRENT\_DATE())

GROUP BY proudcto\_id

) v ON p.id = v.proudcto\_id

GROUP BY p.id, p.nombre;

END

Este procedimiento se utiliza para obtener datos que formarán parte de un reporte mensual de ventas. Realiza consultas complejas para recopilar información relacionada con los productos, ganancias de ventas, costos totales e inventario restante para el mes actual.

Lo que hace:

Realiza múltiples operaciones:

Obtiene el nombre del producto (NombreProducto), la ganancia de ventas (GananciaVentas), el costo total (CostoTotal) y la cantidad restante en inventario (CantidadRestante).

Utiliza subconsultas para calcular la suma de costos, ingresos y egresos del inventario, así como la suma de las ganancias de ventas, agrupadas por producto para el mes actual.

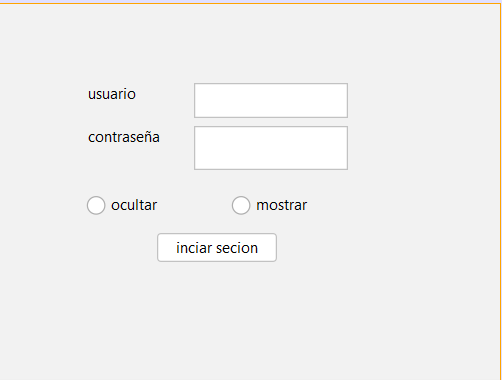
Funcionamiento técnico:

Usa operaciones JOIN (LEFT JOIN) para unir tablas (productos, inventario y ventas) y obtener información específica relacionada con productos, costos, ventas e inventario.

Filtra registros basados en el año y mes actuales utilizando funciones como YEAR(), MONTH() y CURRENT\_DATE().

Este procedimiento es útil para generar un informe detallado de ventas mensuales, proporcionando información clave sobre ganancias, costos y estado del inventario para cada producto durante el mes actual.

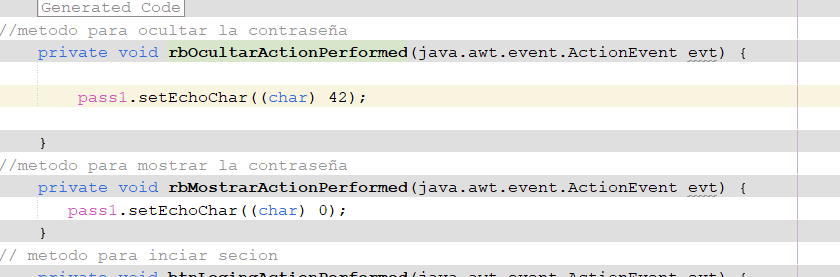
# Pantalla de loging



Componentes de la interfaz grafica

|  |  |
| --- | --- |
| Nombre | Tipo |
| buttonGroup1 | buttonGroup |
| btnLoging | JButton |
| rbMostrar | JRadioButton |
| rbOcultar | JRadioButton |
| jtxtus | JTextField |
| pass1 | JPasswordField |

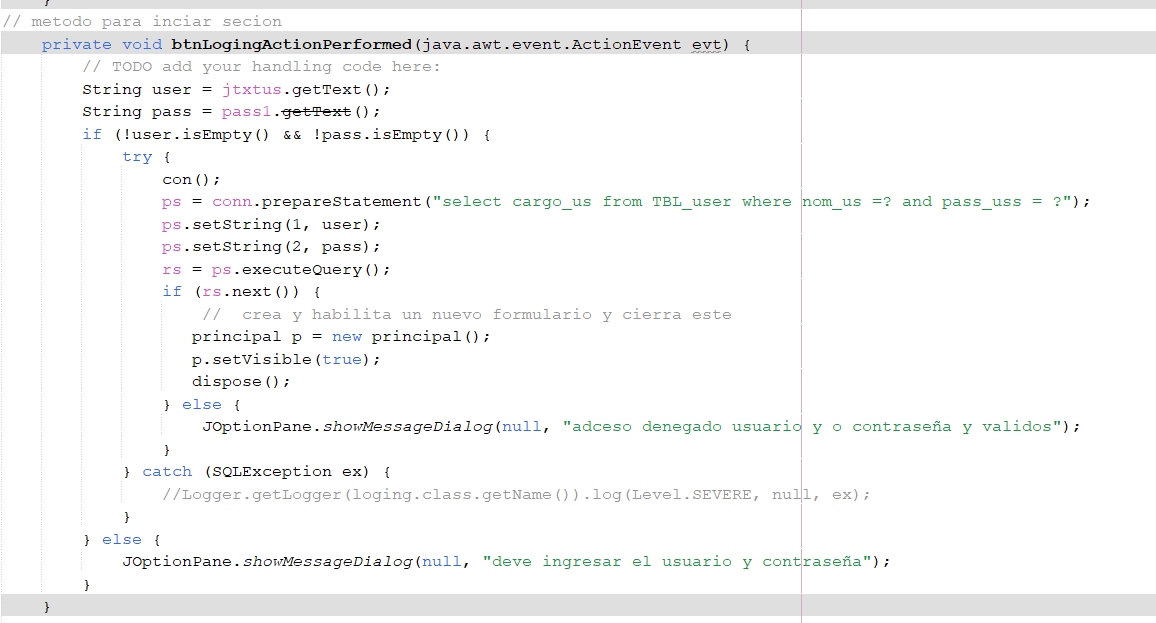
### rbOcultarActionPerformed



Estos eventos se ejecutan cuando se oprimen los radio botón de ocultara y mostrar

El de ocultar pone el carácter especial del JPasswordField con el símbolo asic correspondiste en al número 42 y el evento del botón mostrar lo coloca en 0 para que no se coloque ningún símbolo especial

### Botón de Loging

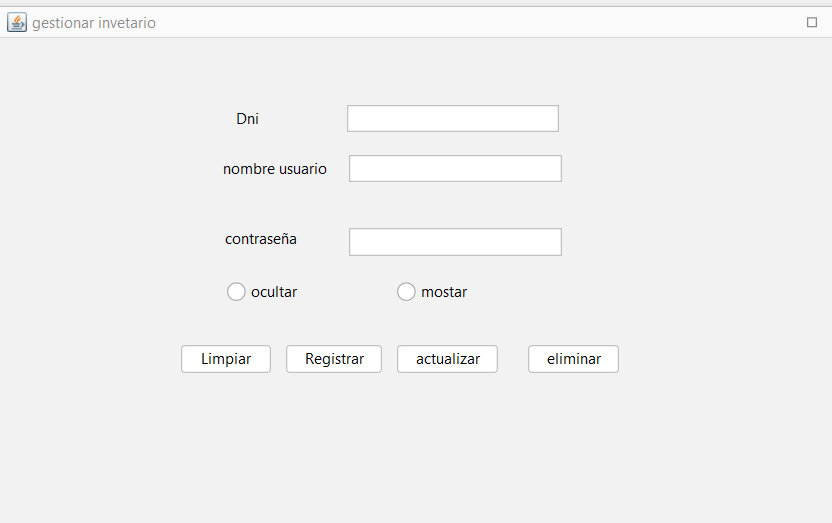


Este código es una parte del sistema de inicio de sesión. Toma el nombre de usuario y la contraseña ingresados por el usuario en dos campos de texto. Luego, verifica si ambos campos no están vacíos.

Después, se conecta a la base de datos para buscar un usuario que coincida con el nombre de usuario y la contraseña proporcionados. Si encuentra una coincidencia en la base de datos, abre una nueva ventana o formulario llamado "principal", mostrando el contenido que tiene. Si no encuentra una coincidencia, muestra un mensaje indicando que el acceso ha sido denegado debido a un usuario o contraseña incorrectos.

En resumen, este código comprueba si el nombre de usuario y la contraseña ingresados son correctos en la base de datos. Si lo son, permite el acceso al sistema; de lo contrario, muestra un mensaje de error.

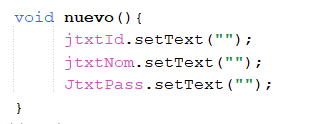
# Pantalla geUsuarios



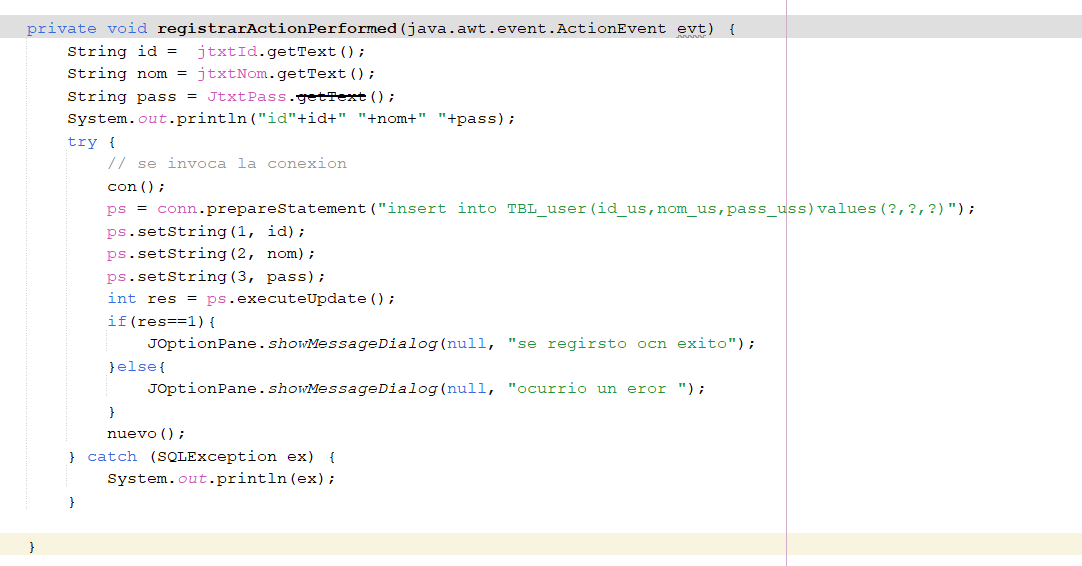
Esta ventana es básicamente un crud con la tabla de usuarios los radio botón de ocultar y mostrar funciona igual que en la pantalla pasada y los campos están nombrados de igual forma

### Botón limpiar

Llama a la función nuevo() la cual limpia todos los campos de la pantalla colocando su texto como una cadena vacia



### Botón Registrar



Este fragmento de código se encarga de registrar un nuevo usuario en la base de datos. Captura tres datos ingresados por el usuario: un ID, un nombre y una contraseña. Luego, realiza un intento de inserción en la tabla TBL\_user de la base de datos.

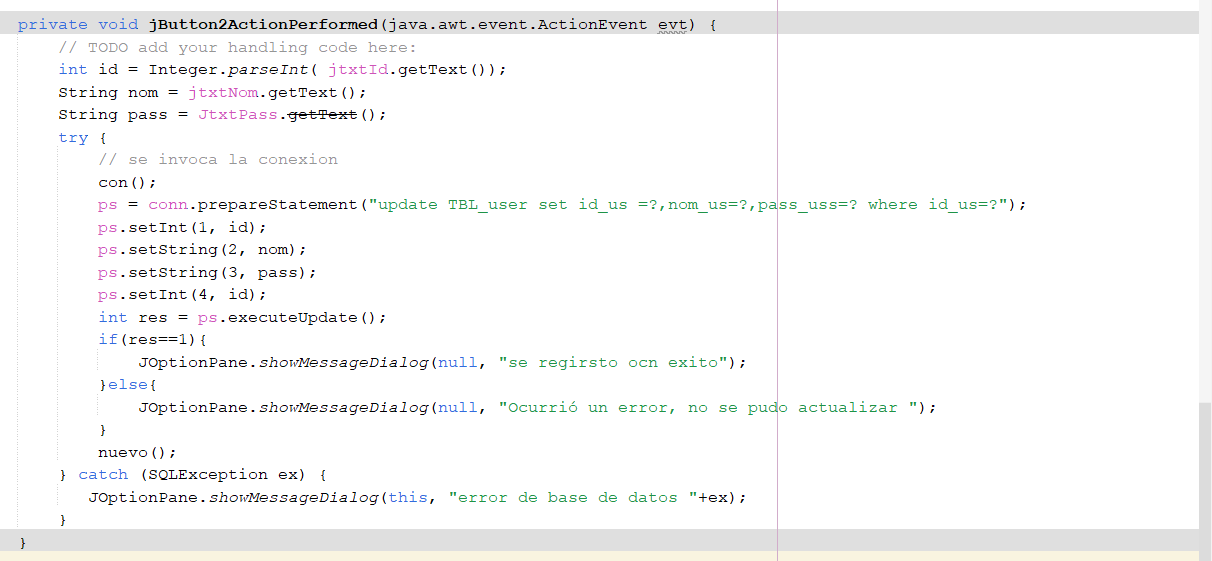
Primero, se establece la conexión con la base de datos. Después, se prepara una consulta SQL para insertar los valores del ID, nombre y contraseña en la tabla TBL\_user. Una vez preparada la consulta, se ejecuta con los valores capturados.

Si la inserción en la base de datos tiene éxito (res == 1), muestra un mensaje confirmando que el registro se realizó correctamente. Si hay algún error en la inserción, muestra un mensaje indicando que ocurrió un error.

Al final, se llama a la función nuevo() (para limpiar los campos)

En resumen, este código registra un nuevo usuario en la base de datos con un ID, nombre y contraseña proporcionados por el usuario y muestra un mensaje de confirmación o de error según el resultado de la inserción.

### Botón Actualizar



Este fragmento de código se encarga de actualizar un registro de usuario en la base de datos. Captura tres datos ingresados por el usuario: un ID (como entero), un nombre y una contraseña. Luego, intenta actualizar un registro existente en la tabla TBL\_user de la base de datos.

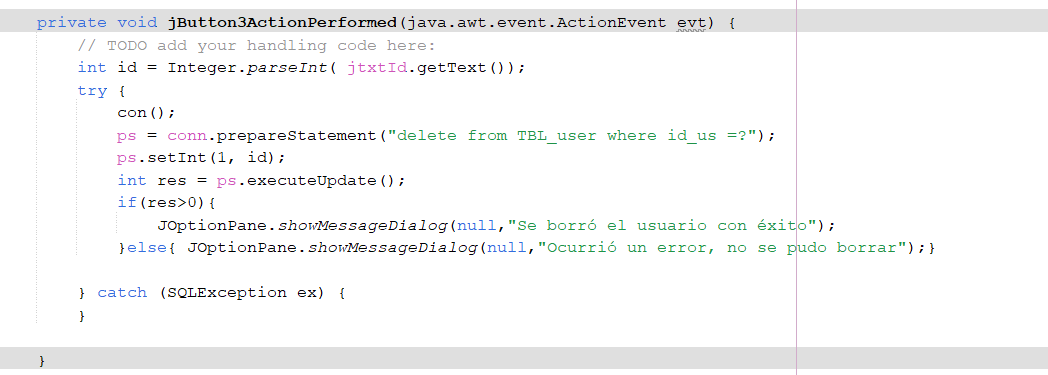
Primero, convierte el ID ingresado en un número entero (int) utilizando Integer.parseInt() para poder realizar operaciones con él. Luego, establece la conexión con la base de datos y prepara una consulta SQL para actualizar los valores del ID, nombre y contraseña en la tabla TBL\_user.

La consulta se diseña para actualizar un registro existente que coincida con el mismo ID. Los valores del ID, nombre y contraseña se establecen en la consulta SQL. Después de ejecutar la consulta, verifica si se actualizó correctamente (res == 1). Si la actualización tiene éxito, muestra un mensaje confirmando la actualización; si no, muestra un mensaje indicando que ocurrió un error.

Al final, se llama a la función nuevo() (limpia los campos)

En resumen, este código actualiza un registro de usuario existente en la base de datos con un nuevo ID, nombre y contraseña proporcionados por el usuario, mostrando un mensaje de confirmación o de error según el resultado de la operación de actualización.

### Botón Eliminar



Este fragmento de código se encarga de eliminar un registro de usuario de la base de datos. Captura un dato ingresado por el usuario: un ID (como entero), y luego intenta eliminar un registro existente en la tabla TBL\_user de la base de datos.

Primero, convierte el ID ingresado en un número entero (int) utilizando Integer.parseInt() para poder realizar operaciones con él. Luego, establece la conexión con la base de datos y prepara una consulta SQL para eliminar el registro que coincida con el ID proporcionado.

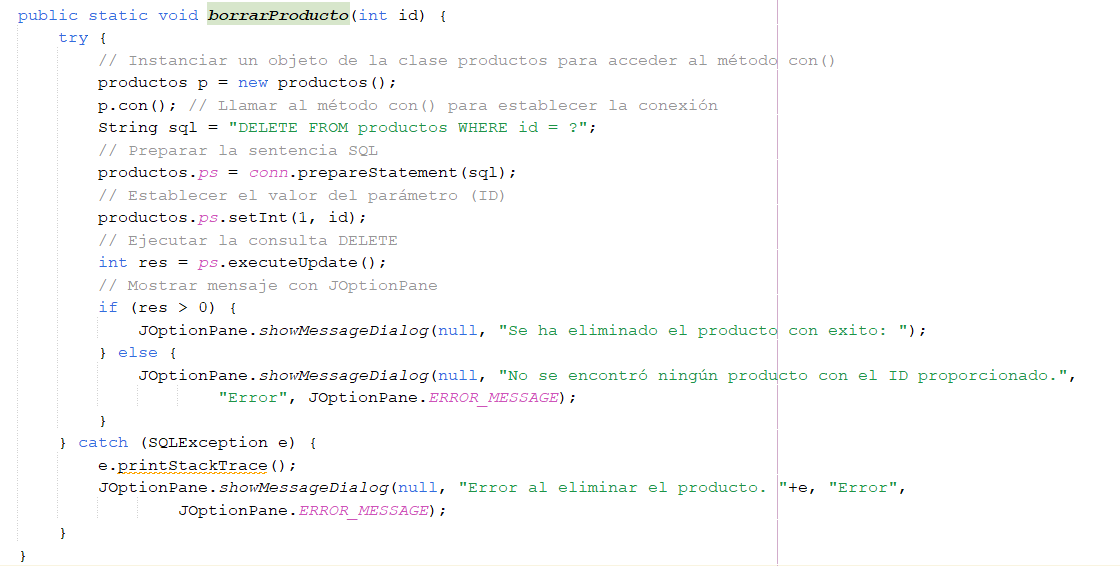
La consulta está diseñada para eliminar el registro que tenga el ID especificado en la tabla TBL\_user. Después de ejecutar la consulta, verifica si se eliminó correctamente (res > 0). Si la eliminación tiene éxito, muestra un mensaje confirmando que el usuario se eliminó; si no, muestra un mensaje indicando que ocurrió un error.

El bloque catch captura cualquier excepción SQL que pueda ocurrir durante el proceso, aunque en este caso, no se hace nada con la excepción capturada.

En resumen, este código elimina un registro de usuario existente en la base de datos, mostrando un mensaje de confirmación o de error según el resultado de la operación de eliminación.

# Clase producto

### Void borrarProducto



Este código está diseñado para eliminar un producto de una base de datos. Primero, crea una conexión a la base de datos utilizando un objeto de la clase productos. Luego, utiliza una consulta SQL DELETE para borrar el producto que coincida con el ID proporcionado.

El proceso implica los siguientes pasos:

Establecer conexión: Se establece una conexión con la base de datos.

Construir la consulta: Se prepara una consulta SQL DELETE que elimina un registro de la tabla de productos utilizando el ID proporcionado.

Asignar parámetros: Se asigna el valor del ID al parámetro de la consulta SQL.

Ejecutar la consulta: La consulta DELETE se ejecuta en la base de datos.

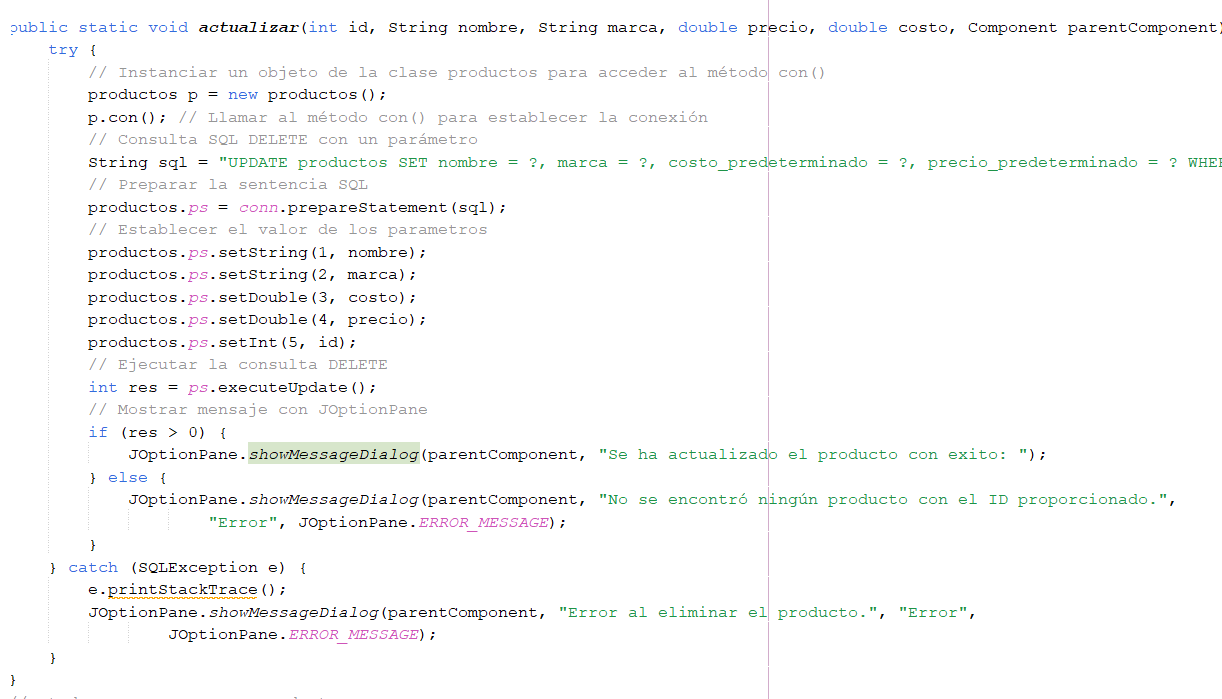
Verificar el resultado: Se verifica si el producto se eliminó correctamente o si no se encontró ningún producto con el ID proporcionado.

Mostrar mensajes: Dependiendo del resultado, se muestra un mensaje informando sobre la eliminación exitosa del producto o se muestra un mensaje de error si no se encuentra ningún producto con ese ID o si hay algún problema durante la eliminación.

En caso de que ocurra un error durante el proceso de eliminación, se muestra un mensaje de error indicando el problema encontrado.

Este código es una función que encapsula el proceso de eliminación de un producto de la base de datos, maneja posibles errores que puedan ocurrir durante la operación y muestra mensajes para informar al usuario sobre el resultado de la eliminación del producto.

### void actualizar



El método actualizar se encarga de modificar la información de un producto existente en la base de datos. Requiere varios parámetros:

id: Identificador único del producto que se desea actualizar.

nombre: Nuevo nombre que se asignará al producto.

marca: Nueva marca que se asignará al producto.

precio: Nuevo precio del producto.

costo: Nuevo costo del producto.

parentComponent: Componente de la interfaz de usuario utilizado para mostrar mensajes de resultado.

El funcionamiento del método es el siguiente:

Establece conexión: Conecta con la base de datos para realizar la actualización del producto.

Prepara consulta SQL: Construye una consulta SQL de tipo UPDATE para cambiar los datos del producto en la base de datos.

Asigna valores a los parámetros: Se asignan los nuevos valores (nombre, marca, costo, precio) a la consulta SQL para actualizar el producto con el ID proporcionado.

Ejecuta la consulta de actualización: Ejecuta la consulta SQL preparada para actualizar el registro del producto.

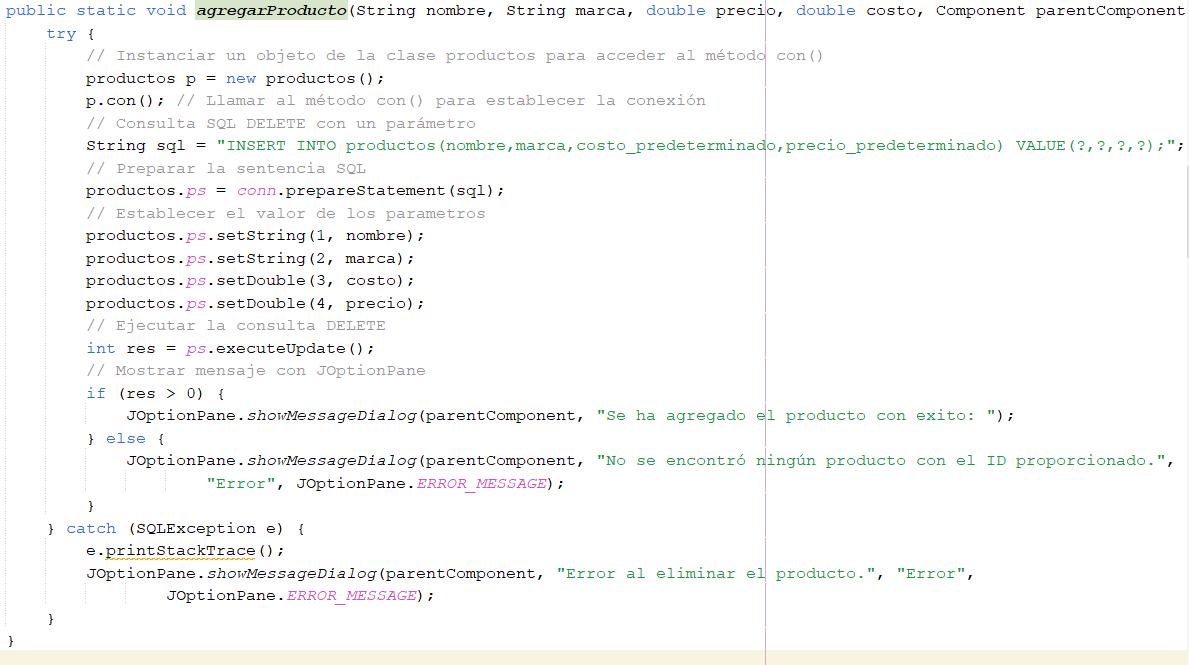
Verifica el resultado: Comprueba la cantidad de registros afectados por la actualización para determinar si la operación fue exitosa.

Muestra mensajes de resultado: Utiliza el componente parentComponent para mostrar mensajes al usuario informando sobre el éxito o fallo de la actualización del producto.

Manejo de errores: Captura y maneja posibles errores que puedan ocurrir durante la actualización del producto en la base de datos, mostrando mensajes detallados en caso de fallo.

Esta función está diseñada para permitir la modificación de la información de un producto en la base de datos y proporcionar retroalimentación al usuario sobre el resultado de la operación de actualización.

### void agregarProducto



El método agregarProducto está diseñado para añadir un nuevo producto a la base de datos. Requiere los siguientes parámetros:

nombre: Nombre del nuevo producto a ser agregado.

marca: Marca del nuevo producto.

precio: Precio del nuevo producto.

costo: Costo del nuevo producto.

parentComponent: Componente de la interfaz de usuario utilizado para mostrar mensajes de resultado.

El funcionamiento del método es el siguiente:

Establece conexión: Conecta con la base de datos para realizar la inserción del nuevo producto.

Prepara consulta SQL: Construye una consulta SQL de tipo INSERT para agregar un nuevo registro a la tabla de productos en la base de datos.

Asigna valores a los parámetros: Los valores proporcionados (nombre, marca, costo, precio) se asignan a la consulta SQL para insertar el nuevo producto en la base de datos.

Ejecuta la consulta de inserción: Ejecuta la consulta SQL preparada para añadir el nuevo producto a la base de datos.

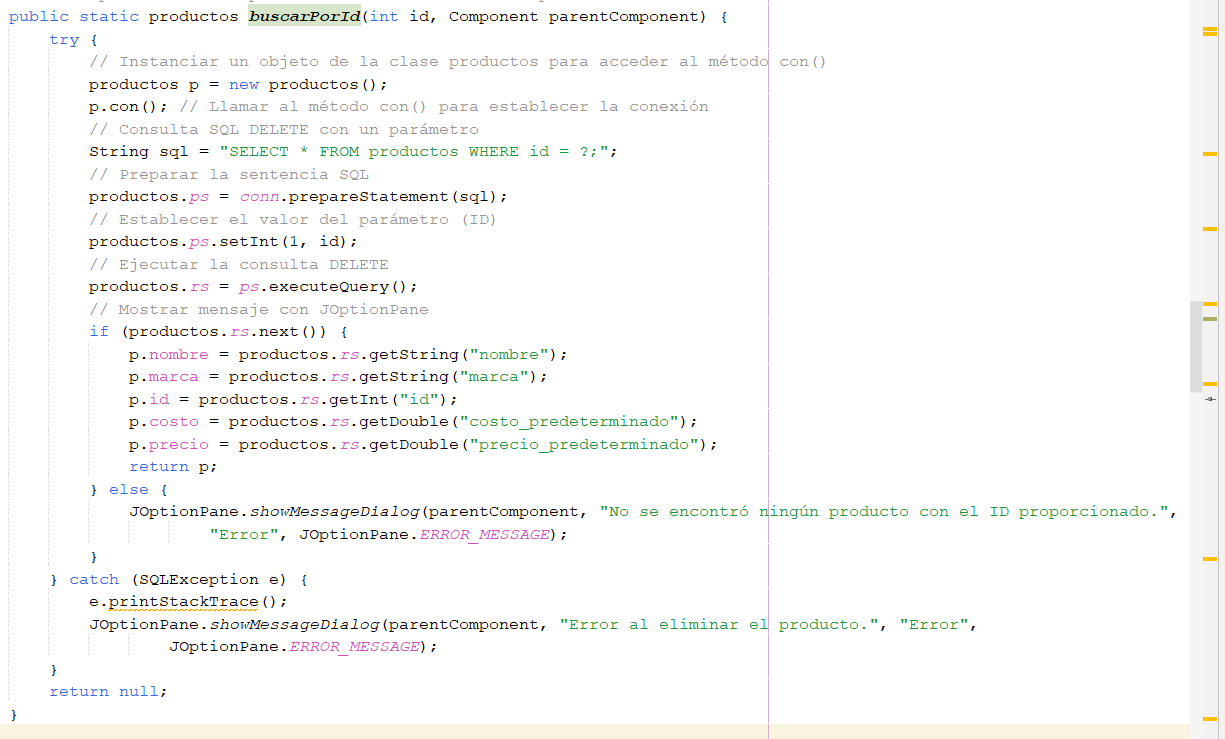
Verifica el resultado: Verifica la cantidad de registros afectados por la inserción para determinar si la operación fue exitosa.

Muestra mensajes de resultado: Utiliza el componente parentComponent para mostrar mensajes al usuario indicando si se agregó correctamente el nuevo producto o si hubo algún problema durante la operación.

Manejo de errores: Captura y maneja posibles errores que puedan ocurrir durante la inserción del nuevo producto, mostrando mensajes detallados en caso de fallo.

En resumen, este método proporciona la funcionalidad para agregar un nuevo producto a la base de datos y notificar al usuario sobre el resultado de la operación de inserción.

### productos buscarPorId



El método buscarPorId tiene como finalidad buscar un producto en la base de datos según su ID. Requiere dos parámetros:

id: Identificador único del producto que se desea buscar.

parentComponent: Componente de la interfaz de usuario utilizado para mostrar mensajes de resultado.

El funcionamiento del método es el siguiente:

Establece conexión: Conecta con la base de datos para realizar la búsqueda del producto por su ID.

Prepara consulta SQL: Construye una consulta SQL de tipo SELECT para buscar un registro en la tabla de productos que coincida con el ID proporcionado.

Asigna valor al parámetro: El ID proporcionado se asigna a la consulta SQL para filtrar la búsqueda por ese ID específico.

Ejecuta la consulta de búsqueda: Ejecuta la consulta SQL preparada para obtener el producto con el ID proporcionado.

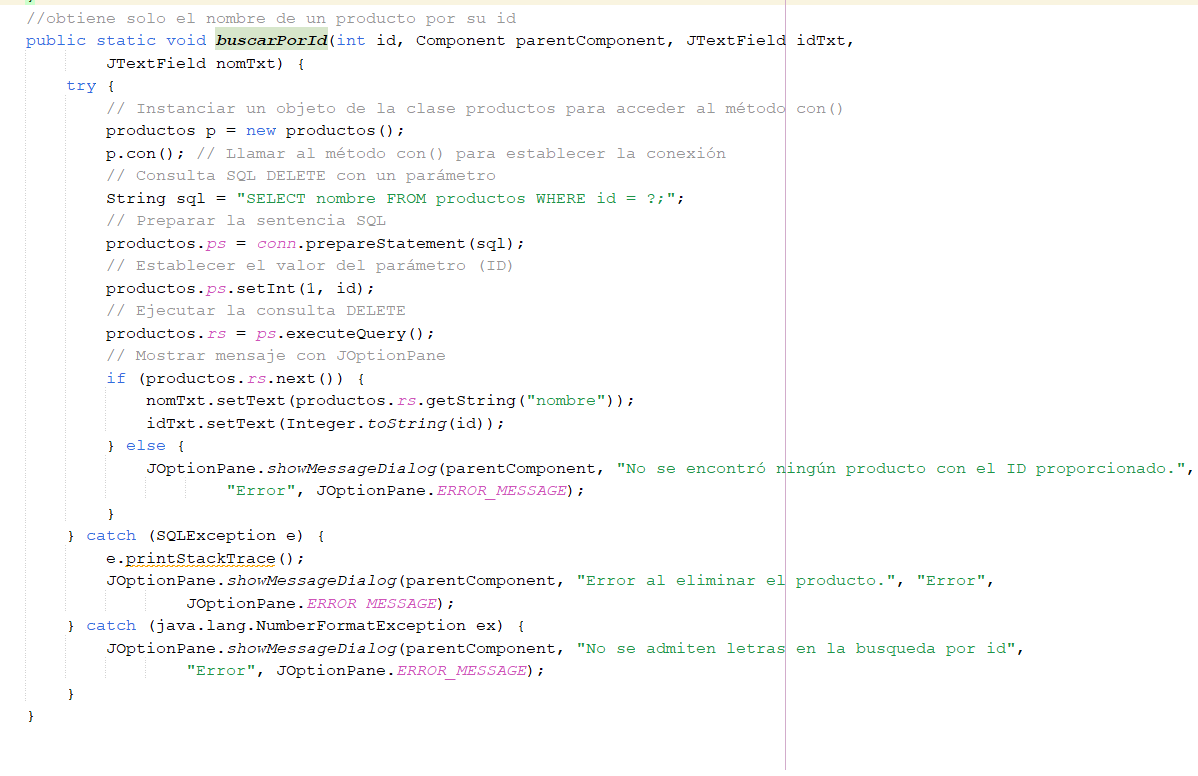
Verifica el resultado: Verifica si se encontró algún producto con el ID proporcionado en el resultado de la consulta.

Procesamiento del resultado: Si se encuentra un producto con el ID dado, se crea un objeto productos con la información obtenida (nombre, marca, id, costo, precio) y se devuelve ese objeto. Si no se encuentra ningún producto, se muestra un mensaje indicando la falta de coincidencias.

Manejo de errores: Captura y maneja posibles errores que puedan ocurrir durante la búsqueda del producto por ID, mostrando mensajes detallados en caso de fallo.

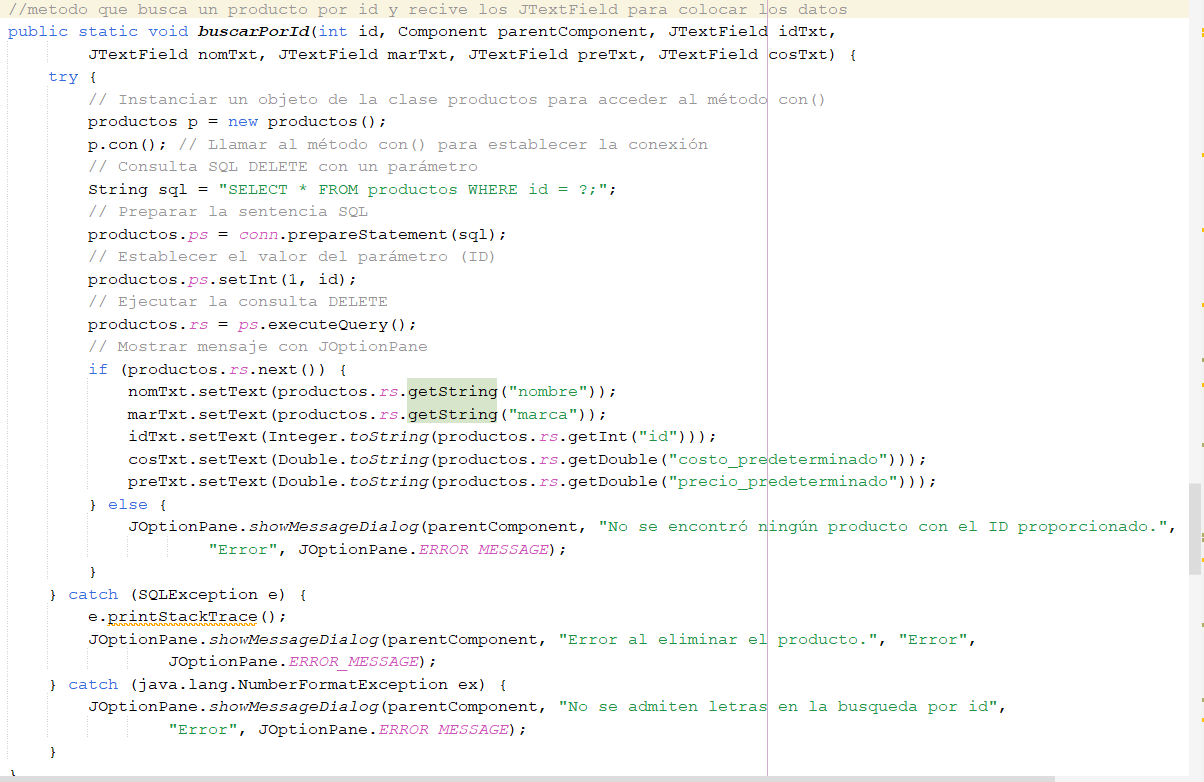
En resumen, este método permite buscar un producto por su ID en la base de datos y retorna el objeto productos correspondiente si se encuentra, de lo contrario, notifica al usuario sobre la ausencia de un producto con el ID proporcionado.

### void buscarPorId



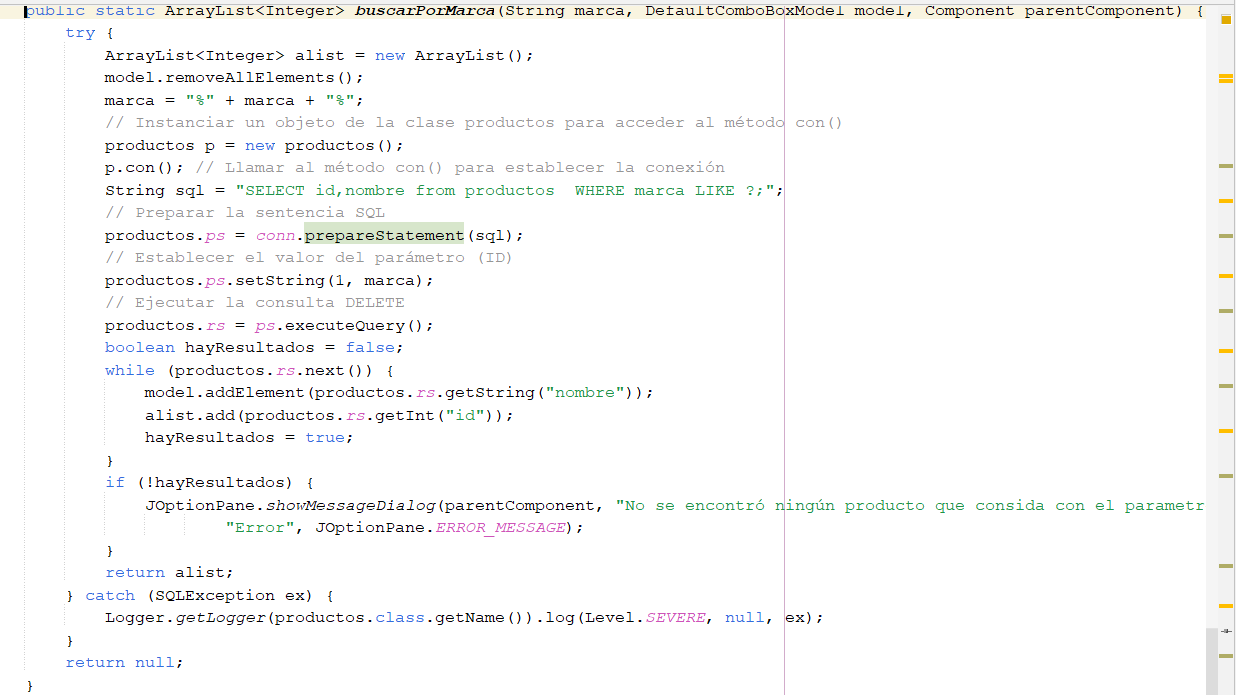
Este método es similar al anterior aprovecha la sobreescritura de métodos para crear otro método con el mismo nombre pero con diferentes para metros que son componentes JTextField recibe los JTextField de la id y el nombre del producto y busca en la base de datos por id un producto al igual que el método anterior pero en ves de devolver un objecto con los datos los coloca directamente en los JTextField

### buscarPorId



Igual que el método anterior, pero recibe los JTextField de todos los datos de la clase productos y coloca los resultados del producto encontrado directamente en los JTextField

### ArrayList<Integer> buscarPorMarca



El método buscarPorMarca tiene como finalidad buscar productos en la base de datos qu cuya marca sea similar a un parámetro especificado específica. Requiere tres parámetros:

marca: Marca de los productos que se desean buscar.

model: Objeto DefaultComboBoxModel utilizado para almacenar los nombres de los productos encontrados.

parentComponent: Componente de la interfaz de usuario utilizado para mostrar mensajes de resultado.

El funcionamiento del método es el siguiente:

Inicialización de variables: Se crea un ArrayList llamado alist para almacenar los IDs de los productos encontrados y se limpia el modelo (model) que contiene los nombres de los productos para una nueva búsqueda.

Formateo de la marca: La marca proporcionada se transforma para que pueda ser utilizada como un patrón en la consulta SQL. Se agrega % al principio y al final de la marca para buscar coincidencias parciales.

Establece conexión: Se conecta con la base de datos para realizar la búsqueda de productos por la marca.

Prepara consulta SQL: Construye una consulta SQL de tipo SELECT para buscar registros en la tabla de productos que coincidan con la marca proporcionada.

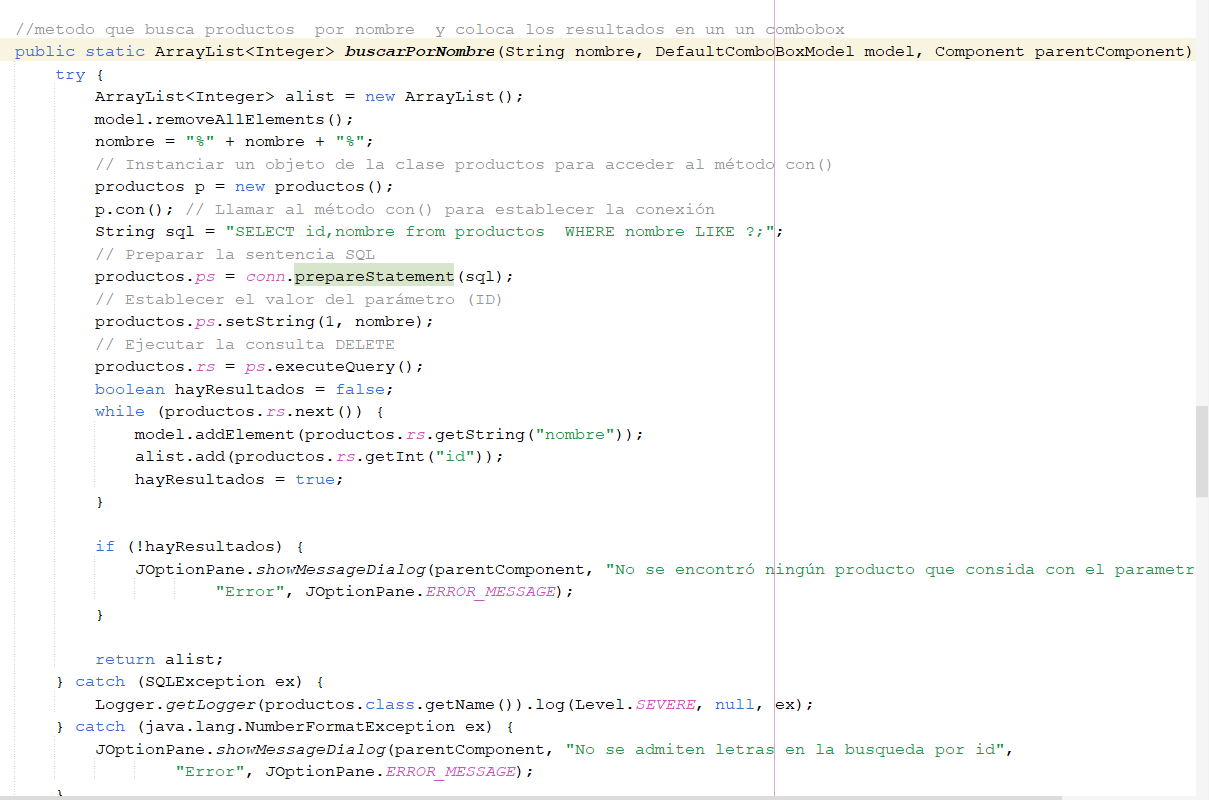
Asigna valor al parámetro: La marca formateada se asigna a la consulta SQL para filtrar la búsqueda por esa marca.

Ejecuta la consulta de búsqueda: Se ejecuta la consulta SQL preparada para obtener los productos que coinciden con la marca especificada.

Procesamiento del resultado: Por cada resultado obtenido de la consulta, se agrega el nombre del producto al modelo (model) y se guarda el ID del producto en el ArrayList alist. Si no se encuentran resultados, se muestra un mensaje indicando la falta de productos que coincidan con la marca.

Manejo de errores: Captura y maneja posibles errores que puedan ocurrir durante la búsqueda de productos por marca en la base de datos, mostrando mensajes detallados en caso de fallo.

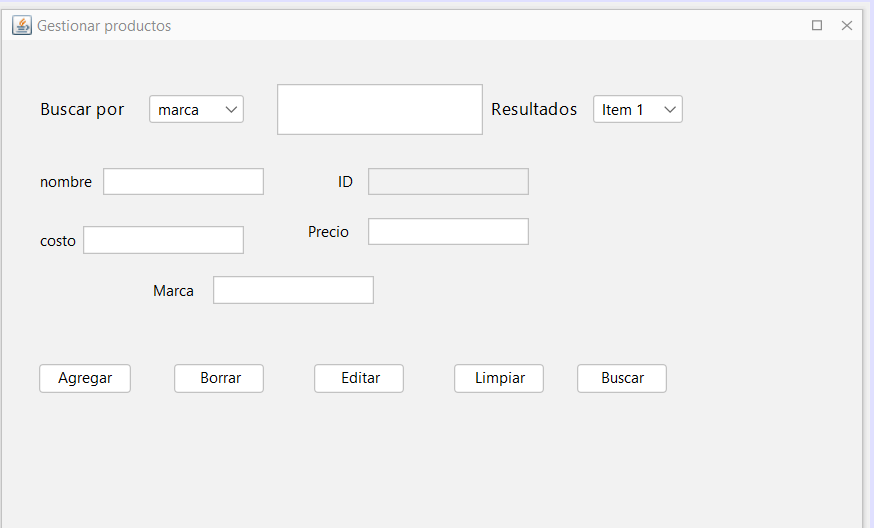
En resumen, este método permite buscar productos por marca en la base de datos, actualizar el modelo con los nombres de los productos encontrados y devuelve una lista de IDs correspondientes a los productos encontrados, o muestra un mensaje si no se encuentra ningún producto que coincida con la marca proporcionada.



### ArrayList<Integer> buscarPorNombre

Este método es similar al anterior, pero en ves de buscar por marca busca productos cuyo nombre censido con el nombre especificado

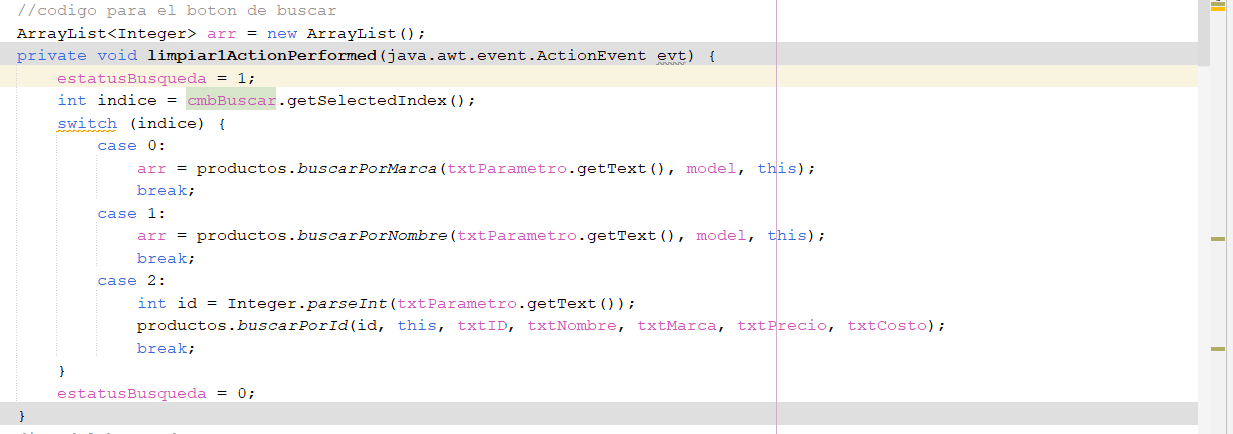
# Pantalla gesProductos



|  |  |
| --- | --- |
| Nombre | Tipo |
| cmbBuscar | JComboBox<String> |
| cmbResult | JComboBox<String> |
| JTextField | txtCosto |
| JTextField | txtID |
| JTextField | txtMarca |
| JTextField | txtNombre |
| JTextField | txtParametro |
| JTextField | txtPrecio |

### Botón buscar

A continuación se presenta una lógica para buscar productos que se reutiliza en varias pantallas por lo que es de suma importancia su comprensión



Se inicializa un ArrayList que servirá para almacenar las identificaciones (IDs) de los productos encontrados. Se establece el valor 1 a una variable previamente creada para gestionar el estado de la búsqueda.

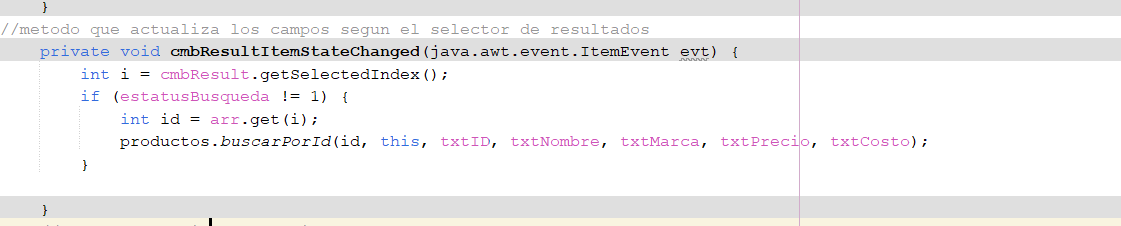
Se recupera el índice seleccionado en el componente cmbBuscar. Dependiendo del índice obtenido:

Si el índice es 0, se invoca el método buscarPorMarca de la clase Producto. Este método recibe como parámetro la marca ingresada y el modelo del ComboBox para almacenar los resultados. Se utiliza 'this' para referirse a la instancia actual. Los resultados de la búsqueda por marca se guardan en el ArrayList llamado 'arr'.

Si el índice es 1, se realiza una operación similar al caso anterior, pero se llama al método buscarPorNombre.

Si el índice es 2, se invoca el método buscarPorId. Este método toma como parám

### cmbResultItemStateChanged



Este método corresponde al cambio de estado de un componente cmbResult. Cuando se selecciona un elemento en este componente, se ejecuta el siguiente procedimiento:

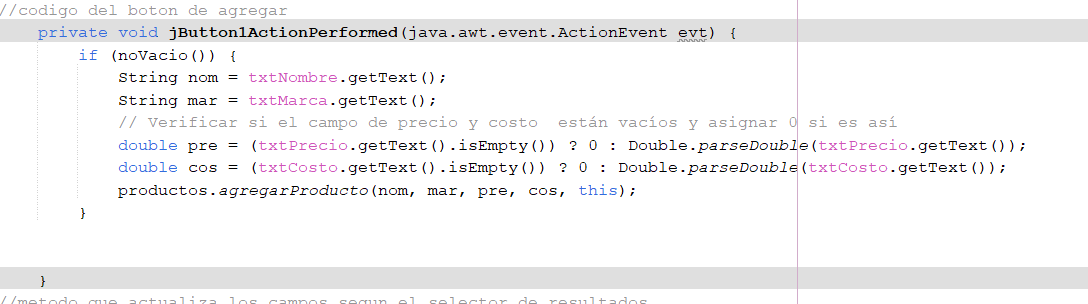
Se obtiene el índice del elemento seleccionado en el componente cmbResult.

Si la variable estatusBusqueda es diferente de 1, lo que indica que no hay una búsqueda activa en curso ya que si se actualizan los campos mientras aun se están agregando elementos en el comboBox esto impide que los elementos restantes se agreguen :

a. Se recupera el identificador (ID) del elemento en la posición 'i' desde la lista arr.

b. Se invoca el método buscarPorId de la clase Producto. Este método utiliza el ID recuperado para buscar y mostrar los detalles correspondientes al producto, actualizando los JTextField txtID, txtNombre, txtMarca, txtPrecio, y txtCosto con la información recuperada.

### Botón agregar



Este método corresponde a la acción realizada al hacer clic en el botón jButton1. Cuando se activa este botón, se realiza lo siguiente:

Se verifica si los campos de texto no están vacíos a través de la función 'noVacio()'.

Si los campos no están vacíos:

a. Se obtiene el texto ingresado en los campos de texto 'txtNombre' y 'txtMarca', asignándolos a las variables 'nom' y 'mar' respectivamente.

b. Se verifica si los campos de 'txtPrecio' y 'txtCosto' están vacíos. En caso afirmativo, se asigna el valor 0 a las variables 'pre' (precio) y 'cos' (costo). En caso contrario, se convierte el texto ingresado a valores numéricos (doble) y se asignan a las respectivas variables.

c. Se invoca el método 'agregarProducto' de la clase Producto, pasándole como parámetros el nombre, marca, precio, costo y una referencia al componente actual (this). Este método se encarga de añadir un nuevo producto a la base de datos con la información proporcionada.

### Botón borrar



Este método se ejecuta al hacer clic en el botón jButton2. Cuando se activa este botón, se realiza lo siguiente:

Se muestra un mensaje personalizado con dos opciones: 'Continuar' y 'Cerrar'. Esta ventana emergente solicita confirmación para eliminar un producto y almacena la respuesta del usuario en la variable 'res'.

Si el campo de texto 'txtID' no está vacío y se selecciona la opción 'Continuar' en el mensaje de confirmación:

a. Se intenta convertir el texto ingresado en 'txtID' a un número entero (ID del producto).

b. Se invoca el método 'borrarProducto' de la clase Producto, pasándole el ID del producto a eliminar.

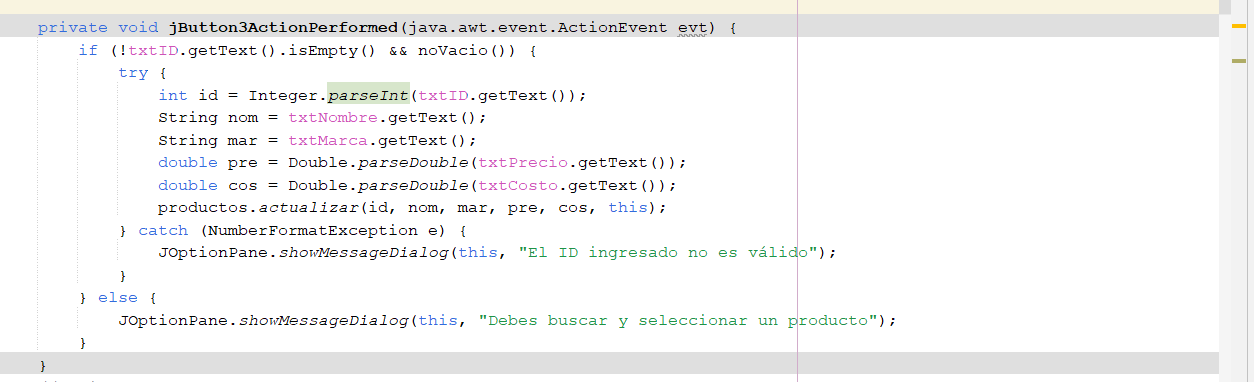
Si el campo de texto 'txtID' no está vacío pero no se selecciona la opción 'Continuar':

a. Se muestra un mensaje indicando que se debe buscar y seleccionar un producto.

Si el campo de texto 'txtID' está vacío:

1. Se muestra un mensaje indicando que es necesario buscar y seleccionar un producto

### Botón editar



Este método responde al evento de clic en el botón jButton3. Al activarse, se realiza lo siguiente:

Se verifica si el campo de texto 'txtID' no está vacío y si todos los campos de texto requeridos no están vacíos utilizando la función 'noVacio()' para validar.

Si 'txtID' no está vacío y todos los campos necesarios tienen datos:

a. Se intenta convertir el texto ingresado en 'txtID' a un número entero (ID del producto).

b. Se obtienen los valores de los campos de texto 'txtNombre', 'txtMarca', 'txtPrecio' y 'txtCosto' para obtener el nombre, marca, precio y costo del producto a actualizar.

c. Se invoca el método 'actualizar' de la clase Producto, pasando el ID del producto y los nuevos valores de nombre, marca, precio, costo, y una referencia a la interfaz actual (this).

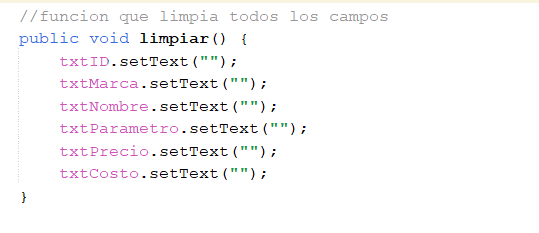
Si 'txtID' está vacío o falta información en algún otro campo:

a. Se muestra un mensaje indicando que es necesario buscar y seleccionar un producto válido.

b. En caso de que el ID ingresado no sea válido (no sea un número entero), se muestra un mensaje informando sobre este error.

### Botón limpiar

Llama a la función limpiar la cual limpia todos los campos



# Clase Inventario

### void ingresoInvetario



este método, llamado 'ingresoInventario', tiene como objetivo registrar el ingreso de productos en el inventario. Al ser activado, realiza lo siguiente:

Establece conexión con la base de datos para trabajar con la tabla 'inventario'.

Prepara una consulta SQL de tipo INSERT para agregar una entrada al inventario, incluyendo el ID del producto, el costo y la cantidad ingresada.

Asigna los valores correspondientes a los parámetros de la consulta SQL: el ID del producto, el costo y la cantidad ingresada.

Ejecuta la consulta preparada para agregar la entrada al inventario.

Verifica si la inserción se realizó correctamente verificando el número de registros afectados.

Muestra un mensaje utilizando JOptionPane, indicando si se agregaron los productos al inventario con éxito o si no se encontró un producto con el ID proporcionado.

En caso de que ocurra un error durante la ejecución de la consulta SQL, se maneja la excepción SQLException mostrando un mensaje de error detallado en JOptionPane.

### Void obtenerRestantes



"Este método, llamado 'obtenerRestantes', busca obtener información sobre los valores restantes en el inventario para un producto específico. Al ejecutarse, realiza lo siguiente:

Establece conexión con la base de datos para acceder a la información del inventario.

Prepara una consulta SQL que llama a un procedimiento almacenado utilizando el ID del producto como parámetro.

Asigna el valor del ID del producto al parámetro de la consulta preparada.

Ejecuta la consulta SQL para obtener los datos del inventario asociados al producto.

Verifica si hay resultados en el conjunto de resultados (ResultSet). Si se encuentra un resultado:

Obtiene los valores del total de costos y de la cantidad restante en el inventario.

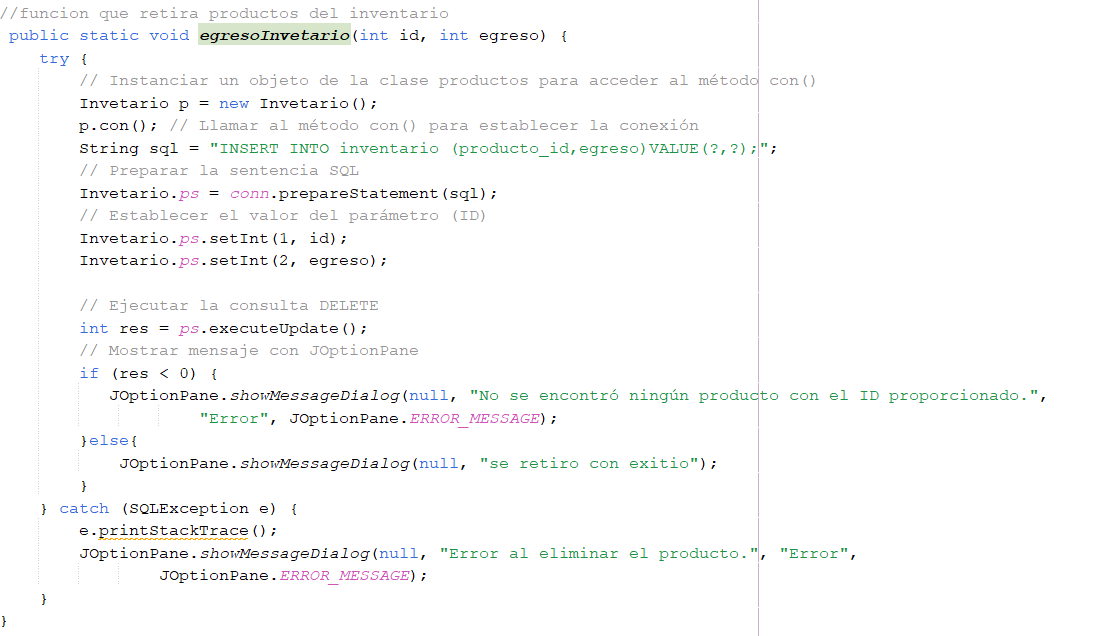
Almacena estos valores en un array de etiquetas ('etiquetas'), donde la posición 0 corresponde a la cantidad restante ('restantes') y la posición 1 al total de costos ('totalCosto').

En caso de no encontrar datos en el inventario para el producto especificado, muestra un mensaje de error utilizando JOptionPane.

Maneja excepciones de tipo SQLException, mostrando un mensaje detallado de error si ocurre algún problema al obtener los valores del inventario.

Devuelve el array 'etiquetas' que contiene la información recuperada del inventario (la cantidad restante y el total de costos) como un arreglo de cadenas (String[]).

### void egresoInvetario



"Este método, llamado 'egresoInventario', se encarga de registrar el retiro de productos del inventario. Al ser activado, realiza lo siguiente:

Establece conexión con la base de datos para acceder a la tabla 'inventario'.

Prepara una consulta SQL de tipo INSERT para agregar una entrada al inventario, indicando el ID del producto y la cantidad retirada ('egreso').

Asigna los valores correspondientes a los parámetros de la consulta SQL: el ID del producto y la cantidad retirada.

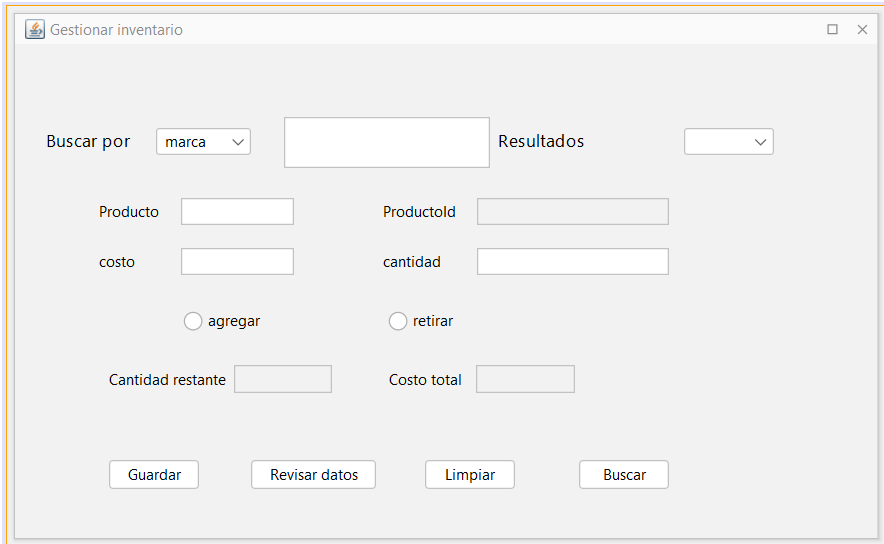
Ejecuta la consulta preparada para registrar la salida de productos del inventario.

Verifica si la operación se realizó correctamente revisando el número de registros afectados.

Muestra un mensaje utilizando JOptionPane indicando si se retiraron los productos con éxito o si no se encontró un producto con el ID proporcionado.

Maneja excepciones de tipo SQLException, mostrando un mensaje detallado de error en caso de surgir problemas al registrar el retiro de productos del inventario

# Pantalla gestInventario

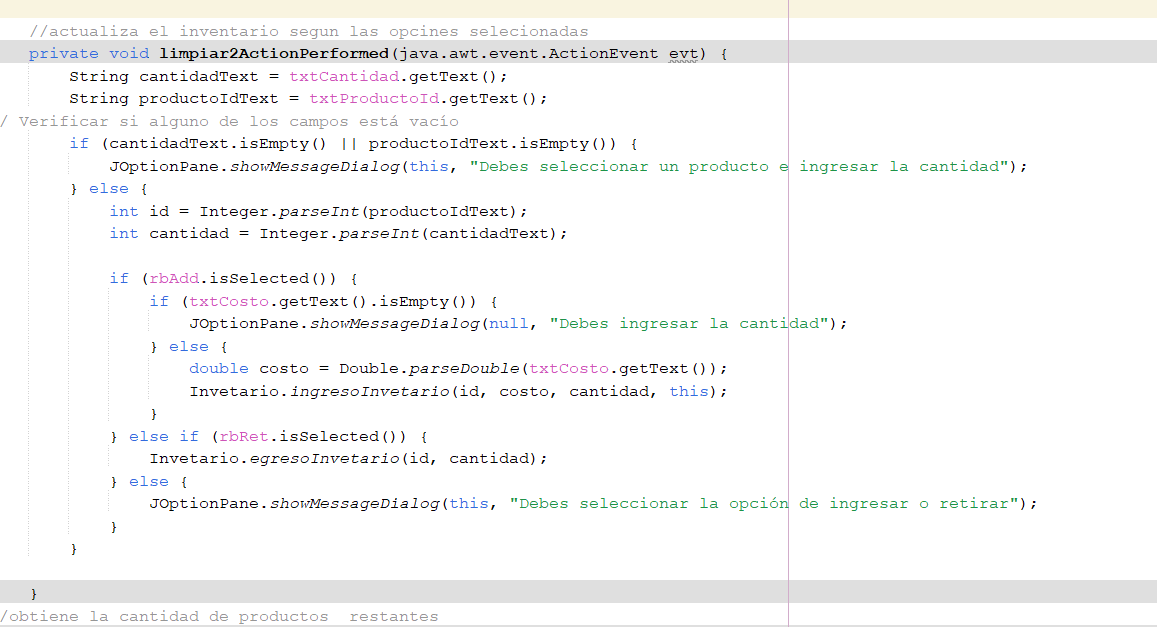


La lógica del botón buscar y los combobox funcionan igual que en la pantalla de gesProductos

|  |  |
| --- | --- |
| Nombre | Tipo |
| txtTotal | JTextField |
| txtProductoId | JTextField |
| txtProducto | JTextField |
| txtParametro | JTextField |
| txtCosto | JTextField |
| txtCantidad | JTextField |
| txtCan | JTextField |
| rbRet | JRadioButton |
| rbAdd | JRadioButton |
| cmbResult | JComboBox<String> |
| cmbBuscar | JComboBox<String> |

.

### Botón guardar



Este método, asociado a la acción de limpiar2, se encarga de actualizar el inventario según las opciones seleccionadas por el usuario. Al ejecutarse, realiza lo siguiente:

Obtiene el texto ingresado en los campos 'txtCantidad' y 'txtProductoId' para verificar si alguno de ellos está vacío.

Si alguno de los campos está vacío, muestra un mensaje utilizando JOptionPane indicando que se debe seleccionar un producto e ingresar la cantidad.

Si ambos campos tienen valores:

a. Convierte el texto del campo 'txtProductoId' a un número entero (ID del producto) y el texto del campo 'txtCantidad' a un número entero (cantidad).

b. Verifica qué opción ha sido seleccionada: si se ha marcado 'rbAdd' (opción de agregar al inventario) o 'rbRet' (opción de retirar del inventario).

c. Si 'rbAdd' está seleccionado, verifica si el campo 'txtCosto' está vacío. Si no lo está, convierte el texto del campo 'txtCosto' a un número decimal (costo) y llama al método 'ingresoInventario' de la clase Inventario para agregar productos con el ID, costo y cantidad proporcionados.

d. Si 'rbRet' está seleccionado, llama al método 'egresoInventario' de la clase Inventario para retirar productos del inventario con el ID y cantidad proporcionados.

e. Si ninguna opción está seleccionada, muestra un mensaje indicando que se debe seleccionar la opción de ingresar o retirar productos."

En resumen, este método gestiona la lógica para agregar o retirar productos del inventario según las selecciones del usuario, validando los campos ingresados y actuando de acuerdo con la opción elegida

### Botón revisarDatos

### 

El método 'limpiar3ActionPerformed' está diseñado para obtener la cantidad de productos restantes en el inventario. Al activarse, realiza lo siguiente:

Verifica si el campo de texto 'txtProductoId' no está vacío.

Si 'txtProductoId' contiene un valor:

a. Convierte el texto del campo 'txtProductoId' a un número entero (ID del producto).

b. Llama al método 'obtenerRestantes' de la clase Inventario, pasando el ID del producto como parámetro. Este método devuelve un array de Strings ('etiquetas') con la cantidad restante y el total de productos.

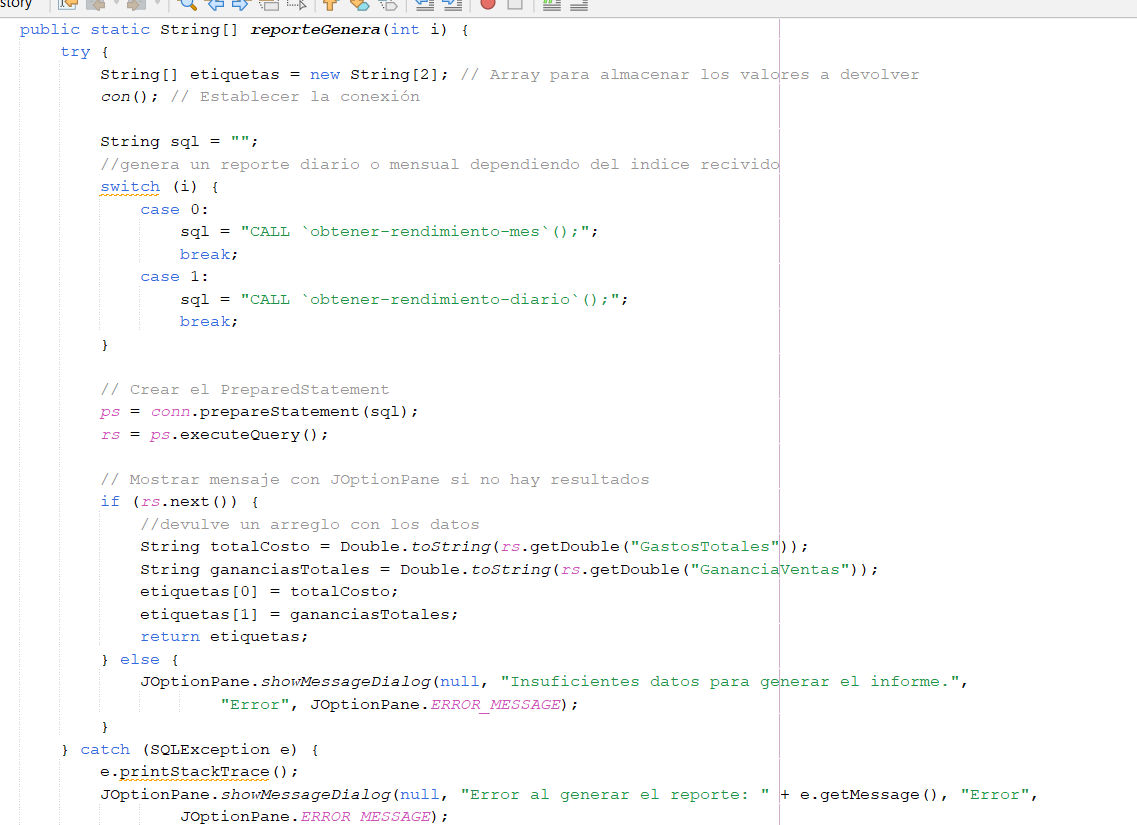
c. Asigna los valores obtenidos del array ('etiquetas') a los campos de texto 'txtCan' y 'txtTotal' para mostrar la cantidad restante y el total de productos, respectivamente.

Si 'txtProductoId' está vacío, muestra un mensaje utilizando JOptionPane indicando que se debe buscar y seleccionar un producto.

En resumen, este método recupera información del inventario para un producto específico y la muestra en campos de texto correspondientes, garantizando que el usuario seleccione un producto antes de realizar esta operación.

# Clase Reporte

### String[] reporteGenera



Este método, llamado 'reporteGenera', tiene la función de obtener datos para generar un informe general, ya sea diario o mensual, según el índice recibido como parámetro.

Al ejecutarse, lleva a cabo los siguientes pasos:

Establece la conexión con la base de datos.

Genera una consulta SQL dependiendo del índice recibido: 'obtener-rendimiento-mes' si el índice es 0 o 'obtener-rendimiento-diario' si es 1.

Prepara y ejecuta un PreparedStatement con la consulta SQL definida.

Verifica si hay resultados en el conjunto de resultados (ResultSet). Si se encuentra un resultado:

a. Obtiene los valores del total de costos y ganancias totales del informe.

b. Almacena estos valores en un array de Strings ('etiquetas'), donde la posición 0 corresponde al total de costos ('totalCosto') y la posición 1 a las ganancias totales ('gananciasTotales').

c. Retorna este array con los datos recuperados del informe.

Si no se encuentran datos en el informe, muestra un mensaje de error utilizando JOptionPane indicando que hay insuficientes datos para generar el informe.

Maneja excepciones de tipo SQLException, mostrando un mensaje detallado en caso de que ocurra algún error durante la generación del informe.

### void reporteProProducto()

El método 'reporteProProducto' está encargado de generar un informe por producto en un archivo PDF. Al ejecutarse, realiza los siguientes pasos:

Obtiene la ruta del directorio seleccionado para guardar el archivo de informe en formato PDF utilizando el método 'obtenerDirectorioSeleccionado' de la clase 'factura'.

Muestra un mensaje de advertencia indicando al usuario que la generación del informe podría llevar tiempo.

Establece la conexión con la base de datos.

Ejecuta un procedimiento almacenado ('reporte-xproducto-mensual') mediante un PreparedStatement.

Crea un documento PDF usando la librería 'iText'.

Agrega un título al documento con el texto 'Reporte por producto', alineado al centro y con un espacio después.

Crea una tabla en el documento con cuatro columnas: 'producto', 'ventas', 'gastos' y 'restantes'.

Recorre los resultados obtenidos del procedimiento almacenado y los agrega como filas a la tabla en el documento PDF.

Verifica si hay resultados en el conjunto de resultados (ResultSet). Si no hay resultados, muestra un mensaje informativo indicando la ausencia de datos para generar el reporte.

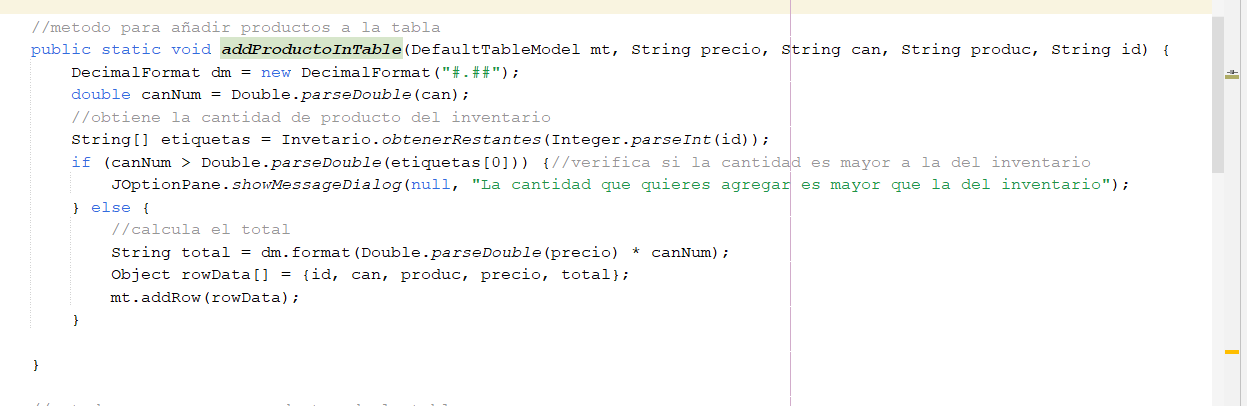
Agrega la tabla al documento y cierra el archivo PDF para finalizar la generación del informe.

Maneja excepciones de tipo SQLException y FileNotFoundException, mostrando mensajes detallados en caso de errores durante la generación del reporte."

En resumen, este método se encarga de generar un informe en formato PDF, mostrando información detallada por producto en forma de tabla dentro del documento. Además, realiza manejo de excepciones y notifica al usuario sobre el progreso y posibles errores durante la generación del reporte.

# Clase factura

### void addProductoInTable

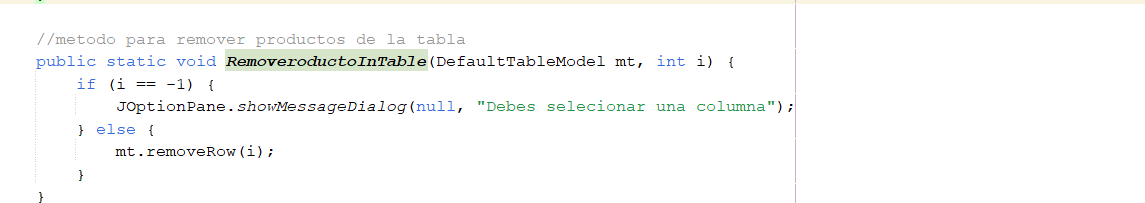


Este método, llamado 'addProductoInTable', se encarga de añadir productos a una tabla mediante un DefaultTableModel en una interfaz gráfica. Al ejecutarse, realiza lo siguiente:

1. Crea un objeto DecimalFormat ('dm') para formatear números con dos decimales.
2. Convierte la cantidad proporcionada ('can') a un número decimal ('canNum').
3. Utiliza el método 'obtenerRestantes' de la clase 'Invetario' para obtener las etiquetas (restantes en inventario y total) de un producto con un ID específico.
4. Verifica si la cantidad ingresada es mayor que la cantidad disponible en el inventario. Si es mayor, muestra un mensaje con JOptionPane indicando que la cantidad a agregar supera la cantidad disponible en el inventario.
5. Si la cantidad ingresada es válida (es menor o igual a la cantidad en inventario), calcula el total multiplicando el precio por la cantidad y formatea este valor.
6. Crea un arreglo de objetos 'rowData' con los datos del ID del producto, cantidad, nombre del producto, precio unitario y total.
7. Añade estos datos como una nueva fila al DefaultTableModel proporcionado ('mt') para que sean mostrados en la tabla."

En resumen, este método verifica si la cantidad de productos que se intenta agregar a la tabla no excede la cantidad disponible en el inventario y, en caso afirmativo, añade una nueva fila con los detalles del producto al modelo de tabla proporcionado para su visualización en la interfaz gráfica

### void RemoveroductoInTable

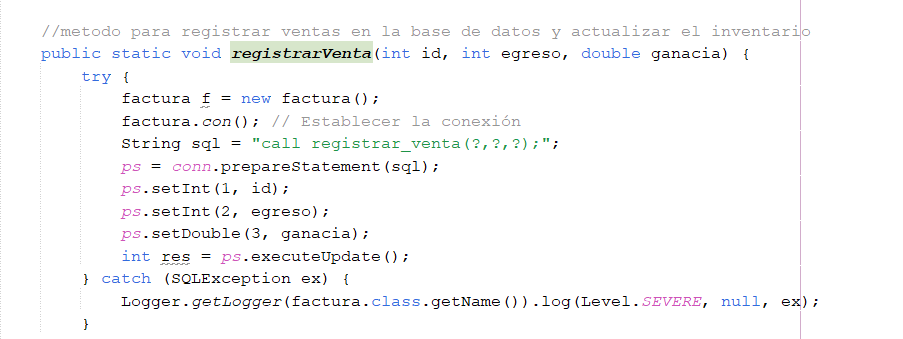


El método 'RemoveroductoInTable' se encarga de eliminar productos de una tabla representada por un DefaultTableModel en una interfaz gráfica. Al ejecutarse, realiza lo siguiente:

1. Verifica si el índice de la columna seleccionada ('i') es igual a -1. Si es así, muestra un mensaje con JOptionPane indicando que se debe seleccionar una columna para eliminar.
2. Si el índice es diferente de -1, lo que significa que se ha seleccionado una columna válida, elimina la fila correspondiente a ese índice del DefaultTableModel ('mt')."

En resumen, este método permite eliminar una fila de la tabla si se ha seleccionado una columna válida, y muestra un mensaje de advertencia si no se ha seleccionado ninguna columna.

### void registrarVenta



El método 'registrarVenta' se encarga de registrar una venta en la base de datos y actualizar el inventario correspondiente. Al ejecutarse, realiza lo siguiente:

Crea una instancia de la clase 'factura' para utilizar sus métodos.

Establece la conexión a la base de datos utilizando el método 'con' de la clase 'factura'.

Prepara la sentencia SQL para llamar al procedimiento almacenado 'registrar\_venta' con tres parámetros: el ID del producto vendido ('id'), la cantidad vendida ('egreso'), y la ganancia generada ('ganacia').

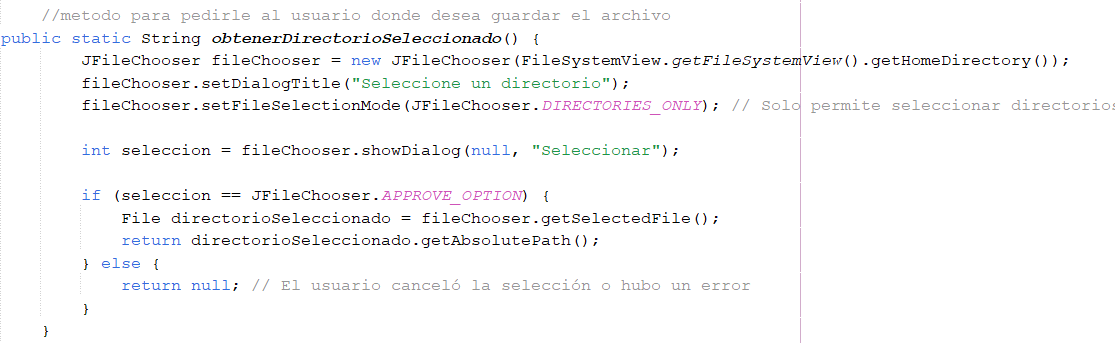
Asigna los valores a los parámetros en la sentencia preparada.

Ejecuta la consulta SQL utilizando 'executeUpdate' y almacena el resultado en la variable 'res'.

En caso de ocurrir una excepción de tipo SQLException, captura y registra la excepción en el log para su posterior manejo."

En resumen, este método facilita el registro de una venta en la base de datos y la actualización del inventario correspondiente con los datos proporcionados

### String obtenerDirectorioSeleccionado()



El método 'obtenerDirectorioSeleccionado' se encarga de solicitar al usuario que elija un directorio donde desea guardar un archivo. Al ejecutarse, realiza lo siguiente:

Crea un objeto 'JFileChooser', un componente de interfaz gráfica que permite al usuario seleccionar un directorio.

Establece el título del selector de archivos como 'Seleccione un directorio' y configura que solo se puedan seleccionar directorios.

Muestra el diálogo de selección de archivos ('showDialog') para que el usuario elija un directorio.

Si el usuario selecciona un directorio y presiona el botón 'Seleccionar', obtiene el directorio seleccionado a través del método 'getSelectedFile()' del 'JFileChooser' y devuelve su ruta absoluta como un String.

Si el usuario cancela la selección o hay un error en la operación, el método devuelve 'null'.

En resumen, este método interactúa con un componente de selección de archivos ('JFileChooser') para permitir al usuario elegir un directorio específico donde guardar archivos.

### void generarFactura

El método 'generarFactura' se encarga de generar facturas a partir de los datos presentes en una tabla. Al ejecutarse, realiza lo siguiente:

Obtiene el número de filas de la tabla ('mt') para verificar si hay productos agregados.

Si la tabla está vacía, muestra un mensaje usando JOptionPane indicando que se deben agregar productos antes de facturar y retorna, interrumpiendo la operación.

Declara una variable 'totalNeto' para almacenar el monto total de la factura.

Crea un documento PDF para la factura y establece su título como 'Inversiones Ponce'.

Crea una tabla en el documento PDF con 4 columnas ('cantidad', 'producto', 'precio', 'total').

Recorre las filas de la tabla de productos ('mt'), obteniendo la cantidad, producto, precio y total de cada artículo. Añade estas celdas a la tabla del PDF.

Para cada producto en la factura, actualiza la base de datos registrando la venta utilizando el método 'registrarVenta'.

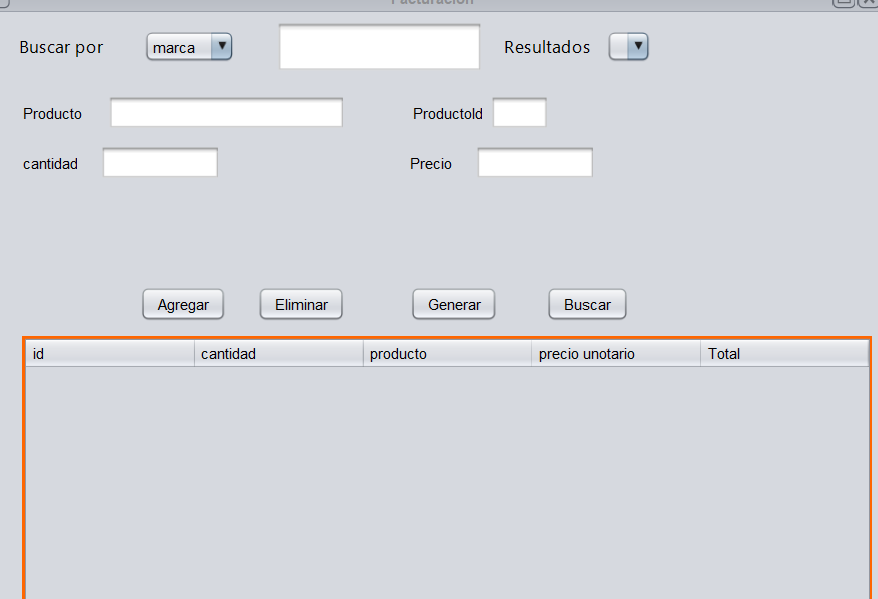
Calcula y muestra el total a pagar en el documento PDF.

Solicita al usuario ingresar la cantidad pagada y calcula el cambio.

Finalmente, cierra el documento PDF y muestra un mensaje con el total a pagar y el cambio.

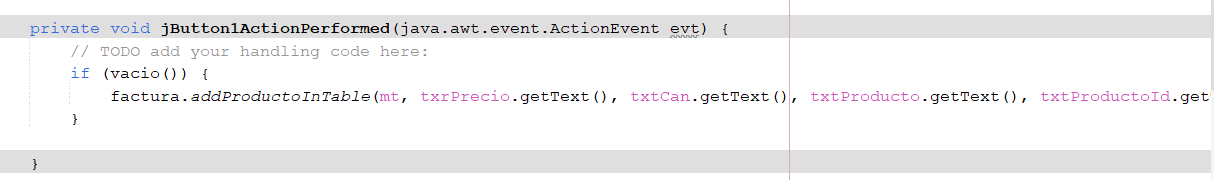
En resumen, este método genera una factura en formato PDF a partir de los datos de una tabla, calcula el monto total a pagar, actualiza la base de datos con las ventas y gestiona la interacción con el usuario para el pago y el cambio.

# Pantalla facturar



La lógica del botón buscar es igual que en las pantallas anteriores

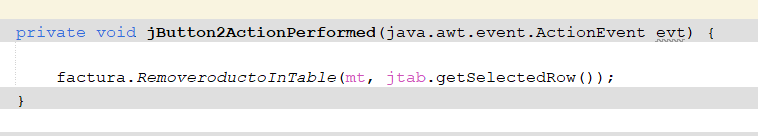
### Botón Agregar



Verifica si hay campos vacíos invocando al método 'vacio'.

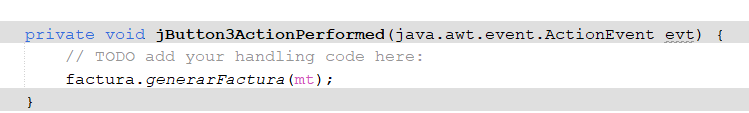
Si no hay campos vacíos (es decir, si 'vacio' retorna falso), llama al método 'addProductoInTable' de la clase 'factura'.

### Botón Eliminar



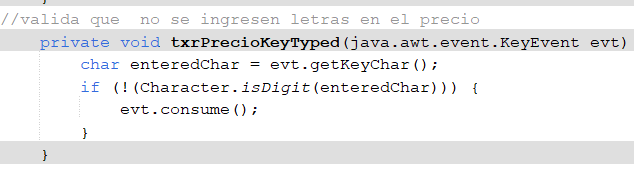
Llama al método pasándole el modelo y la fila seleccionada

### Botón generar



# Validación de campos general

Em todas las pantallas del sistema se usa un evento para evitar que el usuario ingrese letras en los campos numéricos como precio costo y cantidad



El método 'txrPrecioKeyTyped' está asociado al evento de tecla tipeada ('KeyTyped') del componente 'txrPrecio'. Cuando se ejecuta este método:

Captura el carácter ingresado por el usuario a través del evento 'getKeyChar()' de la clase 'KeyEvent'.

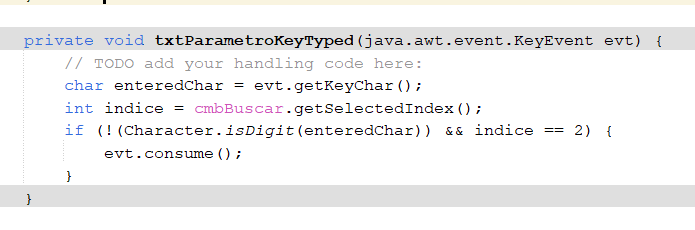
Verifica si el carácter ingresado no es un dígito utilizando 'Character.isDigit(enteredChar)'.

Si el carácter ingresado no es un dígito, utiliza 'evt.consume()' para consumir ese evento, lo que evita que el carácter se muestre en el campo de texto 'txrPrecio'.

En resumen, este método impide que se ingresen caracteres que no sean dígitos en el campo de texto 'txrPrecio', asegurando que solo se ingresen números

Y todos los campos numéricos tienen asociado un evento iguala a este

### Validar parámetro id



"El método 'txtParametroKeyTyped' está asociado al evento de tecla tipeada ('KeyTyped') del campo de texto 'txtParametro'. Al ejecutarse este método:

Captura el carácter ingresado por el usuario a través del evento 'getKeyChar()' de la clase 'KeyEvent'.

Obtiene el índice seleccionado en el JComboBox 'cmbBuscar' usando 'cmbBuscar.getSelectedIndex()'.

Verifica si el carácter ingresado no es un dígito y si el índice seleccionado en 'cmbBuscar' es igual a 2 (lo que posiblemente indica una búsqueda por ID).

Si el carácter ingresado no es un dígito y se está buscando por ID (índice 2 en 'cmbBuscar'), utiliza 'evt.consume()' para consumir ese evento, lo que evita que el carácter se muestre en el campo de texto 'txtParametro'.

En resumen, este método impide que se ingresen caracteres que no sean dígitos en el campo de texto 'txtParametro' cuando se está buscando por ID, asegurando que solo se ingresen números en ese caso