Clase Conexion

```java
package controlador;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

/**
 *
 * @author ediso
 */
public class Conexion {

    // Conexión local con SSL deshabilitado
    public static Connection conectar() {
        try {
            Connection cn = DriverManager.getConnection(
                "jdbc:mysql://localhost/bd_sistema_concesionario?useSSL=false", "root", "25112004"
            );
            return cn;
        } catch (SQLException e) {
            System.out.println("Error en la conexion local: " + e);
        }
        return null;
    }
}
```

Importaciones

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
```

- **`Connection`**: Representa la conexión con la base de datos.
- **`DriverManager`**: Gestiona los controladores JDBC y facilita la obtención de conexiones con la base de datos.
- **`SQLException`**: Maneja las excepciones relacionadas con operaciones de bases de datos.

1. Definición del método

```java
public static Connection conectar() {
    try {
        Connection cn = DriverManager.getConnection(
            "jdbc:mysql://localhost/bd_sistema_concesionario?useSSL=false", "root", "angelsebas2004"
        );
        return cn;
    } catch (SQLException e) {
        System.out.println("Error en la conexion local: " + e);
    }
    return null;
}
```

- El método **`conectar()`** es **`static`**, lo que significa que se puede llamar sin necesidad de crear una instancia de la clase `Conexion`.
- **Tipo de retorno** : Devuelve un objeto de tipo **`Connection`**si la conexión es exitosa; de lo contrario, devuelve **`null`**

2. Configuración de la conexión

```
Connection cn = DriverManager.getConnection(
    "jdbc:mysql://localhost/bd_sistema_concesionario?useSSL=false", "root", "angelsebas2004"
);
```

- **`DriverManager.getConnection()`**: Método estático que establece la conexión con la base de datos.
- **Cadena de conexión** :
  - **`jdbc:mysql://`**: Especifica el protocolo JDBC para una base de datos MySQL.
  - **`localhost`**: Indica que la base de datos está alojada en la misma máquina (servidor local).
  - **`bd_sistema_concesionario`**: Nombre de la base de datos.
  - **`useSSL=false`**: Desactiva el uso de SSL (Secure Sockets Layer) para conexiones locales. Esto es común en entornos de desarrollo.
- **`"root"`**: Usuario de la base de datos.
- **`"angelsebas2004"`**: Contraseña del usuario **root** .

3. Excepciones

```
    } catch (SQLException e) {
        System.out.println("Error en la conexion local: " + e);
    }
    return null;
```

- Si hay algún problema al establecer la conexión, como un error en la cadena de conexión, usuario o contraseña incorrectos, o la base de datos no está disponible, se lanza una **`SQLException`**.
- El mensaje de error se imprime en la consola.

4.Return

- Si la conexión se establece correctamente, el objeto **`Connection`**(`cn`) se devuelve al llamador.
- Si ocurre un error, el método devuelve **`null`**.

Clase Ctrl_Categoria

Importaciones

- **`Connection`**: Representa la conexión con la base de datos.
- **`PreparedStatement`**: Objeto para ejecutar consultas SQL parametrizadas.
- **`ResultSet`**: Almacena resultados de consultas SQL `SELECT`.
- **`SQLException`**: Maneja errores relacionados con la base de datos.
- **`Statement`**: Ejecuta consultas SQL no parametrizadas.
- **`Categoria`**: Clase modelo que almacena los atributos de una categoría.

Método guardar

```
public boolean guardar(Categoria objeto) {
    boolean respuesta = false;
    Connection cn = controlador.Conexion.conectar();
    try {

        PreparedStatement consulta = cn.prepareStatement("insert into tb_categoria values(?,?,?)");
        consulta.setInt(1, 0);
        consulta.setString(2, objeto.getDescripcion());
        consulta.setInt(3, objeto.getEstado());

        if (consulta.executeUpdate() > 0) {
            respuesta = true;
        }

        cn.close();

    } catch (SQLException e) {
        System.out.println("Error al guardar cartegoria: " + e);
    }

    return respuesta;
}
```

Descripción

1. **Objetivo** : Inserta una nueva categoría en la tabla **`tb_categoria`**.
2. **Parámetros** : Recibe un objeto `Categoria`, que contiene la descripción y estado de la categoría.
3. **Lógica** :
    - Se conecta a la base de datos usando `Conexion.conectar()`.
    - Utilice un **`PreparedStatement`**para la consulta parametrizada `INSERT`.
    - **`setInt`**y**`setString`** : Asigna valores a los parámetros de la consulta.
        - **`1`**: Valor `0`(se supone que el ID es autoincremental).
        - **`2`**: Descripción de la categoría.
        - **`3`**: Estado de la categoría.
    - **`executeUpdate`**: Ejecuta la consulta y devuelve el número de filas afectadas.
4. **Cierre** : Cierra la conexión al final.

Método existe Categoría

```
public boolean existeCategoria(String categoria) {
    boolean respuesta = false;
    String sql = "select descripcion from tb_categoria where descripcion = '" + categoria + "';";
    Statement st;

    try {
        Connection cn = Conexion.conectar();
        st = cn.createStatement();
        ResultSet rs = st.executeQuery(sql);
        while (rs.next()) {
            respuesta = true;
        }

    } catch (SQLException e) {
        System.out.println("Error al consultar cartegoria: " + e);
    }
    return respuesta;
}
```

Descripción

1. **Objetivo** : Comprueba si una categoría ya existe en la base de datos.
2. **Parámetros** : Recibe un `String`con el nombre de la categoría.
3. **Lógica** :
    - Se conecta a la base de datos.
    - Se ejecuta una consulta SQL `SELECT`mediante un **`Statement`**.
    - Si el **`ResultSet`**devuelve alguna fila, se establece `respuesta = true`(categoría existente).

Método actualizar

```
public boolean actualizar(Categoria objeto, int idCategoria) {
    boolean respuesta = false;
    Connection cn = controlador.Conexion.conectar();
    try {

        PreparedStatement consulta = cn.prepareStatement("update tb_categoria set descripcion=? where idCategoria ='" +
        consulta.setString(1, objeto.getDescripcion());

        if (consulta.executeUpdate() > 0) {
            respuesta = true;
        }

        cn.close();

    } catch (SQLException e) {
        System.out.println("Error al actualizar cartegoria: " + e);
    }

    return respuesta;
}
```

Descripción

1. **Objetivo** : Actualiza la descripción de una categoría existente.
2. **Parámetros** :
    - **`objeto`**: Contiene la nueva descripción.
    - **`idCategoria`**: ID de la categoría que se desea actualizar.
3. **Lógica** :
    - Utilice uno **`PreparedStatement`**para la consulta `UPDATE`.
    - **`setString`**asigna la nueva descripción.
    - La condición **`where idCategoria = ?`**asegura que solo se actualizará el registro correcto.

Método eliminar

```java
public boolean eliminar(int idColumna) {
    boolean respuesta = false;
    Connection cn = Conexion.conectar();
    try {

        PreparedStatement consulta = cn.prepareStatement(
                "delete from tb_categoria where idColumna ='" + idColumna + "'");
        consulta.executeUpdate();

        if (consulta.executeUpdate() > 0) {
            respuesta = true;
        }

        cn.close();

    } catch (SQLException e) {
        System.out.println("Error al eliminar cartegoria: " + e);
    }

    return respuesta;
}
```

Descripción

1. **Objetivo** : Eliminar una categoría de la tabla `tb_categoria`.
2. **Parámetros** : Recibe el **`idColumna`**que identifica la fila a eliminar.
3. **Lógica** :
    - Utilice uno **`PreparedStatement`**con una consulta `DELETE`.
    - La condición **`where idColumna = ?`**identifica qué fila eliminar.
    - **Error** : El método se ejecuta `executeUpdate`dos veces. Solo una llamada es necesaria.

Clase Ctrl_Cliente

Importaciones

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import javax.swing.JOptionPane;
import modelo.Cliente;
import modelo.Empleado;
```

Estas importaciones son necesarias para trabajar con JDBC (Java Database Connectivity) y realizar operaciones con la base de datos. También se importan clases para mostrar cuadros de diálogo y las clases `Cliente` y `Empleado`.

Método loginUser

```
public boolean loginUser(String usuario, String password) {
    boolean autenticado = false;
    Connection cn = Conexion.conectar();

    if (cn != null) {
        Statement stmt = null;
        ResultSet rs = null;
        try {
            stmt = cn.createStatement();
            String sql = "SELECT * FROM tb_usuario WHERE usuario = '" + usuario + "' AND password = '" + password + "'
            rs = stmt.executeQuery(sql);

            if (rs.next()) {
                autenticado = true;
            }
        } catch (SQLException e) {
            System.out.println("Error al llenar la tabla usuarios: " + e.getMessage());
        } finally {
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (cn != null) cn.close();
            } catch (SQLException e) {
                System.out.println("Error al cerrar la conexión: " + e.getMessage());
            }
        }
    } else {
        System.out.println("La conexión a la base de datos no se pudo establecer.");
    }

    return autenticado;
}
```

- **Objetivo**: Verificar las credenciales de un usuario.
- **Detalles**:
    - Conecta a la base de datos y verifica si el nombre de usuario y la contraseña coinciden con algún registro en la tabla `tb_usuario`.
    - Devuelve `true` si se encuentra un usuario que coincida, de lo contrario, devuelve `false`.

Método guardar

```java
/**
 * ************************************************
 * metodo para guardar un nuevo usuario
 * ************************************************
 */
public boolean guardar(Empleado objeto) {
    boolean respuesta = false;
    Connection cn = Conexion.conectar();
    try {
        PreparedStatement consulta = cn.prepareStatement("insert into tb_usuario values(?,?,?,?,?,?,?)")
        consulta.setInt(1, 0);//id
        consulta.setString(2, objeto.getNombre());
        consulta.setString(3, objeto.getApellido());
        consulta.setString(4, objeto.getUsuario());
        consulta.setString(5, objeto.getPassword());
        consulta.setString(6, objeto.getTelefono());
        consulta.setString(7, objeto.getEstado());
        if (consulta.executeUpdate() > 0) {
            respuesta = true;
        }
        cn.close();
    } catch (SQLException e) {
        System.out.println("Error al guardar usuario: " + e);
    }
    return respuesta;
}
```

- **Objetivo**: Guardar un nuevo usuario en la base de datos.
- **Detalles**:
    - Inserta un nuevo registro en la tabla `tb_usuario` con los datos del objeto `Empleado`.
    - Devuelve `true` si la inserción es exitosa.

Método existeUsuario

```java
/**
 * ************************************************************************
 * metodo para consultar si el producto ya esta registrado en la BBDD
 * ************************************************************************
 */
public boolean existeUsuario(String usuario) {
    boolean respuesta = false;
    String sql = "select usuario from tb_usuario where usuario = '" + usuario + "';";
    Statement st;
    try {
        Connection cn = Conexion.conectar();
        st = cn.createStatement();
        ResultSet rs = st.executeQuery(sql);
        while (rs.next()) {
            respuesta = true;
        }
    } catch (SQLException e) {
        System.out.println("Error al consultar usuario: " + e);
    }
    return respuesta;
}
```

- **Objetivo**: Verificar si un nombre de usuario ya está registrado en la base de datos.

- **Detalles**:

    - Consulta la tabla `tb_usuario` para ver si existe un registro con el nombre de usuario proporcionado.
    - Devuelve `true` si encuentra el usuario, de lo contrario, devuelve `false`.

Otro Método loginUser

```
/**
 * ************************************************
 * metodo para Iniciar Sesion
 * ************************************************
 */
public boolean loginUser(Empleado objeto) {
    boolean respuesta = false;
    Connection cn = Conexion.conectar();
    String sql = "select  usuario, password from tb_usuario where usuario = '" + objeto.getUsuario() + "' and password
    Statement st;
    try {
        st = cn.createStatement();
        ResultSet rs = st.executeQuery(sql);
        while (rs.next()) {
            respuesta = true;
        }
    } catch (SQLException e) {
        System.out.println("Error al Iniciar Sesion");
        JOptionPane.showMessageDialog(null, "Error al Iniciar Sesion");
    }
    return respuesta;
}
```

- **Objetivo**: Verificar las credenciales de un usuario (otra versión del método `loginUser`).

- **Detalles**:

    - Similar al primer método `loginUser`, pero recibe un objeto `Empleado` en lugar de dos parámetros separados.
    - Devuelve `true` si las credenciales coinciden.

Método eliminar

```
/**
 * ************************************************
 * metodo para eliminar un usuario
 * ************************************************
 */
public boolean eliminar(int idUsuario) {
    boolean respuesta = false;
    Connection cn = Conexion.conectar();
    try {
        PreparedStatement consulta = cn.prepareStatement(
                "delete from tb_usuario where idUsuario ='" + idUsuario + "'");
        consulta.executeUpdate();

        if (consulta.executeUpdate() > 0) {
            respuesta = true;
        }
        cn.close();
    } catch (SQLException e) {
        System.out.println("Error al eliminar usuario: " + e);
    }
    return respuesta;
}
//metodo para actualizar
```

- **Objetivo**: Eliminar un usuario de la base de datos.

- **Detalles**:

  - Elimina un registro de la tabla `tb_usuario` basado en el `idUsuario` proporcionado.
  - Devuelve `true` si la eliminación es exitosa.

Método actualizar

```
public boolean actualizar(Empleado empleado, int idEmpleado) {
    boolean respuesta = false;
    Connection cn = Conexion.conectar();
    try {
        String sql = "UPDATE tb_empleado SET nombre = ?, apellido = ?, usuario = ?, password = ?, telefono = ?, estado
        PreparedStatement pst = cn.prepareStatement(sql);

        pst.setString(1, empleado.getNombre());
        pst.setString(2, empleado.getApellido());
        pst.setString(3, empleado.getUsuario());
        pst.setString(4, empleado.getPassword());
        pst.setString(5, empleado.getTelefono());
        pst.setString(6, empleado.getEstado());
        pst.setInt(7, idEmpleado);

        if (pst.executeUpdate() > 0) {
            respuesta = true;
        }
        cn.close();
    } catch (SQLException e) {
        System.out.println("Error al actualizar empleado: " + e);
    }
    return respuesta;
}
```

- **Objetivo**: Actualizar la información de un empleado en la base de datos.

- **Detalles**:

  - Actualiza un registro en la tabla `tb_empleado` con los datos proporcionados.
  - Devuelve `true` si la actualización es exitosa.

Clase Ctrl_Cliente

Importaciones

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import javax.swing.table.DefaultTableModel;
import modelo.Producto;
import vista.AdministrarInventarioAdministrador;
```

- **`Connection`**: Representa la conexión con la base de datos.

- **`PreparedStatement`**: Permite ejecutar consultas SQL parametrizadas (más seguras).
- **`ResultSet`**: Almacena los resultados devueltos por consultas SQL `SELECT`.

- **`SQLException`**: Captura y manejo de errores relacionados con la base de datos.
- **`Statement`**: Ejecuta consultas SQL no parametrizadas.
- **`Cliente`**: Clase modelo que representa un cliente, con atributos como `nombre`, `apellido`, `cedula`, etc.

Método guardar

```
public boolean guardar(Cliente objeto) {
    boolean respuesta = false;
    Connection cn = Conexion.conectar();
    try {
        PreparedStatement consulta = cn.prepareStatement("insert into tb_cliente values(?,?,?,?,?,?,?)");
        consulta.setInt(1, 0);//id
        consulta.setString(2, objeto.getNombre());
        consulta.setString(3, objeto.getApellido());
        consulta.setString(4, objeto.getCedula());
        consulta.setString(5, objeto.getTelefono());
        consulta.setString(6, objeto.getDireccion());

        if (consulta.executeUpdate() > 0) {
            respuesta = true;
        }
        cn.close();
    } catch (SQLException e) {
        System.out.println("Error al guardar cliente: " + e);
    }
    return respuesta;
}
```

Descripción :

1. **Objetivo** : Insertar un nuevo cliente en la tabla **`tb_cliente`**.
2. **Parámetros** :
    - Recibe un objeto **`Cliente`**que contenga datos como `nombre`, `apellido`, `cedula`, etc.
3. **Lógica** :
    - Se conecta a la base de datos.
    - Prepare una consulta SQL parametrizada para insertar un registro.
    - **`setInt`**y**`setString`** : Asigna valores a los campos en el orden indicado.
    - Ejecuta la consulta con **`executeUpdate()`** , que devuelve el número de filas afectadas.
    - Si es mayor que 0, la operación fue exitosa.

Método existe Cliente

```
public boolean existeCliente(String cedula) {
    boolean respuesta = false;
    String sql = "select cedula from tb_cliente where cedula = '" + cedula + "';";
    Statement st;
    try {
        Connection cn = Conexion.conectar();
        st = cn.createStatement();
        ResultSet rs = st.executeQuery(sql);
        while (rs.next()) {
            respuesta = true;
        }
    } catch (SQLException e) {
        System.out.println("Error al consultar cliente: " + e);
    }
    return respuesta;
}
```

Descripción :

1. **Objetivo** : Verificar si un cliente con una cédula específica ya está registrado.
2. **Parámetros** :
    - Recibe un **`String`**que contiene la **cédula** del cliente.
3. **Lógica** :
    - Construya una consulta SQL para buscar la cédula en la tabla **`tb_cliente`**.
    - Si **`ResultSet`**devuelve algún registro, se establece **`respuesta = true`**.

Método actualizar

```
    public boolean actualizar(Cliente cliente, int idCliente) {
        boolean respuesta = false;
        Connection cn = Conexion.conectar();
        try {
            String sql = "UPDATE tb_cliente SET nombre = ?, apellido = ?, cedula = ?, telefono = ?, direccion = ? WHERE idC
            PreparedStatement pst = cn.prepareStatement(sql);

            pst.setString(1, cliente.getNombre());
            pst.setString(2, cliente.getApellido());
            pst.setString(3, cliente.getCedula());
            pst.setString(4, cliente.getTelefono());
            pst.setString(5, cliente.getDireccion());
            pst.setInt(6, idCliente);

            if (pst.executeUpdate() > 0) {
                respuesta = true;
            }
            cn.close();
        } catch (SQLException e) {
            System.out.println("Error al actualizar cliente: " + e);
        }
        return respuesta;

}
```

Descripción :

1. **Objetivo** : Actualizar la información de un cliente existente.
2. **Parámetros** :
    - **`Cliente cliente`**: Contiene los nuevos valores para el cliente.
    - **`int idCliente`**: El ID del cliente a actualizar.
3. **Lógica** :
    - Se prepara una consulta `UPDATE` con parámetros `?`.
    - Asigna los valores actualizados usando **`setString`**y **`setInt`**.
    - Ejecuta la consulta con **`executeUpdate()`**.
    - Si la cantidad de filas afectadas es mayor que 0, la operación fue exitosa.

Método eliminar

```java
public boolean eliminar(int idCliente) {
    boolean respuesta = false;
    Connection cn = Conexion.conectar();
    try {
        PreparedStatement consulta = cn.prepareStatement(
                "delete from tb_cliente where idCliente ='" + idCliente + "'");
        consulta.executeUpdate();

        if (consulta.executeUpdate() > 0) {
            respuesta = true;
        }
        cn.close();
    } catch (SQLException e) {
        System.out.println("Error al eliminar cliente: " + e);
    }
    return respuesta;
}
```

Descripción :

1. **Objetivo** : Eliminar un cliente de la tabla `tb_cliente`.
2. **Parámetros** :
    - **`int idCliente`**: El ID del cliente que se desea eliminar.
3. **Lógica** :
    - Construya una consulta SQL `DELETE`para eliminar un registro basado en el **ID** .
    - **Error** : Se llama **`executeUpdate()`** dos veces. Esto no es necesario y es incorrecto.
4. **Problema de seguridad** : La concatenación del parámetro `idCliente`en la consulta introduce **la inyección SQL** .

Clase Ctrl_Producto

Importaciones

```java
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import javax.swing.table.DefaultTableModel;
import modelo.Producto;
import vista.AdministrarInventarioAdministrador;
```

- **`Connection`**: Representa la conexión con la base de datos
- **`PreparedStatement`**: Permite ejecutar consultas SQL parametrizadas (seguras y eficientes).
- **`ResultSet`**: Contiene los resultados devueltos por una consulta SQL.
- **`SQLException`**: Captura de errores relacionados
- **`Statement`**: Ejecuta consultas SQL sin parámetros (menos seguro).
- **`Producto`**: Clase modelo que representa un producto con sus atr
- **`DefaultTableModel`**: Se usa para manejar datos en ta

- **`AdministrarInventarioAdministrador`**: Posiblemente una interfaz grafica

```
public boolean guardar(Producto objeto) {
    boolean respuesta = false;
    Connection cn = Conexion.conectar();
    try {

        PreparedStatement consulta = cn.prepareStatement("insert into tb_producto values(?,?,?,?,?,?,?)");
        consulta.setInt(1, 0);//id
        consulta.setString(2, objeto.getNombre());
        consulta.setInt(3, objeto.getCantidad());
        consulta.setDouble(4, objeto.getPrecio());
        consulta.setString(5, objeto.getDescripcion());
        consulta.setInt(6, objeto.getPorcentajeIva());
        consulta.setInt(7, objeto.getIdCategoria());


        if (consulta.executeUpdate() > 0) {
            respuesta = true;
        }

        cn.close();

    } catch (SQLException e) {
        System.out.println("Error al guardar producto: " + e);
    }

    return respuesta;
}
```

Descripción :

1. **Objetivo** : Insertar un nuevo producto en la tabla `tb_producto`.
2. **Parámetros** : Un objeto `Producto`que contiene los datos del producto (nomb
3. **Lógica** :
    - Se utiliza una consulta SQL parametrizada para insertar valores.
    - Se ejecuta la consulta y se verifica si **`executeUpdate()`** devuelve un número mayor a 0, lo que indica éxito.

Método existeProducto

```
/**
 * ********************************************************************
 * metodo para consultar si el producto ya esta registrado en la BBDD
 * ********************************************************************
 */
public boolean existeProducto(String producto) {
    boolean respuesta = false;
    String sql = "select nombre from tb_producto where nombre = '" + producto + "';";
    Statement st;

    try {
        Connection cn = Conexion.conectar();
        st = cn.createStatement();
        ResultSet rs = st.executeQuery(sql);
        while (rs.next()) {
            respuesta = true;
        }

    } catch (SQLException e) {
        System.out.println("Error al consultar producto: " + e);
    }
    return respuesta;
```

Descripción :

1. **Objetivo** : Verificar si un producto ya existe en la base de datos.
2. **Parámetros** : El nombre del producto como **`String`**.
3. **Lógica** :
    - Se ejecuta una consulta usando SQL `Statement`(lo cual **no es seguro** por la posibilidad de **inyección SQL**).
    - Si `ResultSet`devuelve algún registro, se establece la variable `respuesta`como `true`.

Método actualizar

```java
/**
 * ************************************************
 * metodo para actualizar un producto
 * ************************************************
 */
public boolean actualizar(Producto objeto, int idProducto) {
    boolean respuesta = false;
    Connection cn = Conexion.conectar();
    try {

        PreparedStatement consulta = cn.prepareStatement("update tb_producto set nombre=?, cantidad = ?, precio = ?, d
        consulta.setString(1, objeto.getNombre());
        consulta.setInt(2, objeto.getCantidad());
        consulta.setDouble(3, objeto.getPrecio());
        consulta.setString(4, objeto.getDescripcion());
        consulta.setInt(5, objeto.getPorcentajeIva());
        consulta.setInt(6, objeto.getIdCategoria());


        if (consulta.executeUpdate() > 0) {
            respuesta = true;
        }
        cn.close();
    } catch (SQLException e) {
        System.out.println("Error al actualizar producto: " + e);
    }
    return respuesta;
}
```

Descripción :

1. **Objetivo** : Actualizar los datos de un producto específico.
2. **Parámetros** :
    - **`Producto objeto`**: Contiene los nuevos datos del producto.
    - **`int idProducto`**: Identificador del producto a actualizar.
3. **Lógica** :
    - Se utiliza una consulta `UPDATE`parametrizada.
    - Se asignan los nuevos valores con **`setString`**, **`setInt`**, etc.

Método eliminar

```java
/**
 * ************************************************
 * metodo para eliminar un producto
 * ************************************************
 */
public boolean eliminar(int idProducto) {
    boolean respuesta = false;
    Connection cn = Conexion.conectar();
    try {
        PreparedStatement consulta = cn.prepareStatement(
                "delete from tb_producto where idProducto ='" + idProducto + "'");
        consulta.executeUpdate();

        if (consulta.executeUpdate() > 0) {
            respuesta = true;
        }
        cn.close();
    } catch (SQLException e) {
        System.out.println("Error al eliminar producto: " + e);
    }
    return respuesta;
}
```

Descripción :

1. **Objetivo** : Eliminar un producto de la base de datos.
2. **Parámetros** : **`int idProducto`**, el ID del producto a eliminar.

Método actualizarStock

```java
public boolean actualizarStock(Producto object, int idProducto) {
    boolean respuesta = false;
    Connection cn = Conexion.conectar();
    try {
        PreparedStatement consulta = cn.prepareStatement("update tb_producto set cantidad=? where idProducto ='" + idPr
        consulta.setInt(1, object.getCantidad());

        if (consulta.executeUpdate() > 0) {
            respuesta = true;
        }
        cn.close();
    } catch (SQLException e) {
        System.out.println("Error al actualizar stock del producto: " + e);
    }
    return respuesta;
}
```

Descripción :

1. **Objetivo** : Actualizar únicamente la cantidad (stock) de un producto.
2. **Parámetros** :
    - **`Producto object`**: Contiene el nuevo valor de la cantidad.
    - **`int idProducto`**: Identificador del producto.

Clase Ctrl_RegistrarVenta

import controlador.Conexion; // Clase que gestiona la conexión a la base de datos.

import java.sql.Connection; // Representa la conexión con la base de datos.

import java.sql.PreparedStatement; // Permite consultas SQL parametrizadas.

import java.sql.ResultSet; // Contiene los resultados de una consulta SQL.

import java.sql.SQLException; // Captura errores relacionados con la base de datos.

import java.sql.Statement; // Ejecuta consultas SQL sin parámetros.

import modelo.CabeceraVenta; // Clase modelo que representa la cabecera de una venta.

import modelo.DetalleVenta; // Clase modelo que representa el detalle de una venta.

Método guardar

```
/**
 * ************************************************
 * metodo para guardar la cabecera de venta
 * ************************************************
 */
public boolean guardar(CabeceraVenta objeto) {
    boolean respuesta = false;
    Connection cn = Conexion.conectar();
    try {
        PreparedStatement consulta = cn.prepareStatement("insert into tb_cabecera_venta values(?,?,?,?,?)",
                Statement.RETURN_GENERATED_KEYS);
        consulta.setInt(1, 0);//id
        consulta.setInt(2, objeto.getIdCliente());
        consulta.setDouble(3, objeto.getValorPagar());
        consulta.setString(4, objeto.getFechaVenta());
        consulta.setInt(5, objeto.getEstado());

        if (consulta.executeUpdate() > 0) {
            respuesta = true;
        }

        ResultSet rs = consulta.getGeneratedKeys();
        while(rs.next()){
            iDColVar = rs.getBigDecimal(1);
            idCabeceraRegistrada = iDColVar.intValue();
        }

        cn.close();
    } catch (SQLException e) {
        System.out.println("Error al guardar cabecera de venta: " + e);
    }
    return respuesta;
}
```

Descripción :

1. **Objetivo** : Guardar un registro en la tabla `tb_cabecera_venta`(cabecera de una venta).
2. **Lógica** :
    - Se inserta un nuevo registro y se devuelve el ID autogenerado usando **`Statement.RETURN_GENERATED_KEYS`**.

- El ID generado se almacena `idCabeceraRegistrada`para poder asociar los detalles de la venta.

Método guardarDetalle

```java
/**
 * ************************************************
 * metodo para guardar el detalle de venta
 * ************************************************
 */
public boolean guardarDetalle(DetalleVenta objeto) {
    boolean respuesta = false;
    Connection cn = Conexion.conectar();
    try {
        PreparedStatement consulta = cn.prepareStatement("insert into tb_detalle_venta values(?,?,?,?,?,?,?,?,?,?)");
        consulta.setInt(1, 0);//id
        consulta.setInt(2, idCabeceraRegistrada);
        consulta.setInt(3, objeto.getIdProducto());
        consulta.setInt(4, objeto.getCantidad());
        consulta.setDouble(5, objeto.getPrecioUnitario());
        consulta.setDouble(6, objeto.getSubTotal());
        consulta.setDouble(7, objeto.getDescuento());
        consulta.setDouble(8, objeto.getIva());
        consulta.setDouble(9, objeto.getTotalPagar());
        consulta.setInt(10, objeto.getEstado());

        if (consulta.executeUpdate() > 0) {
            respuesta = true;
        }
        cn.close();
    } catch (SQLException e) {
        System.out.println("Error al guardar detalle de venta: " + e);
    }
    return respuesta;
}
```

Descripción :

1. **Objetivo** : Guardar los detalles de la venta (productos comprados) en la tabla `tb_detalle_venta`.
2. **Lógica** :
    - Se enlaza cada detalle de venta con la cabecera utilizando el atributo `idCabeceraRegistrada`.
    - Se almacenan valores como ID del producto, cantidad, precio unitario, subtotal, descuento, IVA y total.

Método actualizar

```java
/**
 * ************************************************
 * metodo para actualizar un producto
 * ************************************************
 */
public boolean actualizar(CabeceraVenta objeto, int idCabeceraVenta) {
    boolean respuesta = false;
    Connection cn = Conexion.conectar();
    try {

        PreparedStatement consulta = cn.prepareStatement(
                "update tb_cabecera_venta set idCliente = ?, estado = ? "
                        + "where idCabeceraVenta ='" + idCabeceraVenta + "'");
        consulta.setInt(1, objeto.getIdCliente());
        consulta.setInt(2, objeto.getEstado());

        if (consulta.executeUpdate() > 0) {
            respuesta = true;
        }
        cn.close();
    } catch (SQLException e) {
        System.out.println("Error al actualizar cabecera de venta: " + e);
    }
    return respuesta;
}
```

Descripción :

1. **Objetivo** : Actualizar la cabecera de una venta existente.
2. **Lógica** :
    - Se actualizarán valores como el ID del cliente y el estado de la venta.
    - El ID de la cabecera se utiliza para identificar el registro y modificarlo.

Método llenarTablaVentas

```
public void llenarTablaVentas() {
    Connection cn = Conexion.conectar();
    String sql = "SELECT cv.idCabeceraventa, cv.idCliente, cv.ValorPagar, cv.fechaVenta, cv.estado, c.nombre, c.apell
                 "FROM tb_cabecera_venta cv " +
                 "INNER JOIN tb_cliente c ON cv.idCliente = c.idCliente";
    try {
        PreparedStatement pst = cn.prepareStatement(sql);
        ResultSet rs = pst.executeQuery();
        while (rs.next()) {
            // Procesar los resultados
            int idCabeceraventa = rs.getInt("idCabeceraventa");
            int idCliente = rs.getInt("idCliente");
            double ValorPagar = rs.getDouble("ValorPagar");
            String fechaVenta = rs.getString("fechaVenta");
            int estado = rs.getInt("estado");
            String nombre = rs.getString("nombre");
            String apellido = rs.getString("apellido");

            // Aquí puedes agregar código para llenar la tabla en la interfaz de usuario
            // ...
        }
        cn.close();
    } catch (SQLException e) {
        System.out.println("Error al llenar la tabla de ventas: " + e);
    }
}
```

Descripción :

1. **Objetivo** : Obtener los datos de las ventas (cabecera y cliente) y llenar una tabla en la interfaz gráfica.
2. **Lógica** :
    - Realice una consulta SQL con un **INNER JOIN** para combinar información de `tb_cabecera_venta`y `tb_cliente`.
    - Itera sobre los resultados para procesar los datos.

Importaciones

```
import com.itextpdf.text.BadElementException;
import com.itextpdf.text.BaseColor;
import com.itextpdf.text.Chunk;
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Font;
import com.itextpdf.text.FontFactory;
import com.itextpdf.text.Image;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.pdf.PdfPTable;
import com.itextpdf.text.pdf.PdfWriter;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.swing.JOptionPane;
```

1. **Librerías de iText** : Para trabajar con documentos PDF, tablas, imágenes, fuentes y colores.
2. **Librerías estándar** : Para manejo de excepciones, entradas/salidas de archivos, y ventanas emergentes.
3. **Librerías de JDBC** : Para la conexión y consultas a la base de datos.

1. Método ReportesClientes

```
public void ReportesClientes() {
    Document documento = new Document();
    try {
        String ruta = System.getProperty("user.home");
        PdfWriter.getInstance(documento, new FileOutputStream(ruta + "/Desktop/Reporte_Clientes.pdf"));
        Image header = Image.getInstance("src/img/header1.jpg");
        header.scaleToFit(650, 1000);
        header.setAlignment(Chunk.ALIGN_CENTER);
        //formato al texto
        Paragraph parrafo = new Paragraph();
        parrafo.setAlignment(Paragraph.ALIGN_CENTER);
        parrafo.add("Reporte creado por \nEdison Zambrano © Programador Fantasma\n\n");
        parrafo.setFont(FontFactory.getFont("Tahoma", 18, Font.BOLD, BaseColor.DARK_GRAY));
        parrafo.add("Reporte de Clientes \n\n");

        documento.open();
        //agregamos los datos
        documento.add(header);
        documento.add(parrafo);

        PdfPTable tabla = new PdfPTable(5);
        tabla.addCell("Codigo");
        tabla.addCell("Nombres");
        tabla.addCell("Cedula");
        tabla.addCell("Telefono");
        tabla.addCell("Direccion");

        try {
            Connection cn = Conexion.conectar();
            PreparedStatement pst = cn.prepareStatement(
                    "select idCliente, concat(nombre, ' ', apellido) as nombres, cedula, telefono, direccion from t
            ResultSet rs = pst.executeQuery();
            if (rs.next()) {
                do {
```

- **Propósito** : Genera un reporte PDF con la lista de clientes registrados en la base de datos.
- **Pasos principales** :
    1. **Preparar el documento** :
        - Crear un objeto `Document`.
        - Definir la ruta del archivo PDF como el escritorio del usuario.
        - Agregue una imagen (`header`) como encabezado al documento.
        - Agregue un párrafo que sirve como título del informe.
    2. **Crear la tabla** :
        - Se usa una tabla (`PdfPTable`) con 5 columnas: Código, Nombres, Cédula, Teléfono y Dirección.
        - Realice una consulta SQL para recuperar los datos desde la tabla `tb_cliente`.
        - Agrega los datos de la base de datos a las celdas de la tabla.
    3. **Finalizar el documento** :
        - Agregar la tabla al PDF.
        - Cerrar el documento.
        - Mostrar un mensaje de confirmación al usuario.

2. **Método** ReportesProductos

```
public void ReportesProductos() {
    Document documento = new Document();
    try {
        String ruta = System.getProperty("user.home");
        PdfWriter.getInstance(documento, new FileOutputStream(ruta + "/Desktop/Reporte_Productos.pdf"));
        Image header = Image.getInstance("src/img/header1.jpg");
        header.scaleToFit(650, 1000);
        header.setAlignment(Chunk.ALIGN_CENTER);
        //formato al texto
        Paragraph parrafo = new Paragraph();
        parrafo.setAlignment(Paragraph.ALIGN_CENTER);
        parrafo.add("Reporte creado por \nEdison Zambrano © Programador Fantasma\n\n");
        parrafo.setFont(FontFactory.getFont("Tahoma", 18, Font.BOLD, BaseColor.DARK_GRAY));
        parrafo.add("Reporte de Productos \n\n");

        documento.open();
        //agregamos los datos
        documento.add(header);
        documento.add(parrafo);

        float[] columnsWidths = {3, 5, 4, 5, 7, 5, 6};

        PdfPTable tabla = new PdfPTable(columnsWidths);
        tabla.addCell("Codigo");
        tabla.addCell("Nombre");
        tabla.addCell("Cant.");
        tabla.addCell("Precio");
        tabla.addCell("Descripcion");
        tabla.addCell("Por. Iva");
        tabla.addCell("Categoria");
```

- **Propósito** : Genera un reporte PDF con los productos registrados.
- **Diferencias con el método anterior** :
    - La tabla tiene 7 columnas y anchos personalizados.
    - Se consulta la tabla `tb_productoy` se unen datos de `tb_categoria`para mostrar la categoría de cada producto.
    - Las columnas incluyen: Código, Nombre, Cantidad, Precio, Descripción, Porcentaje IVA y Categoría.

3. **Método** ReportesCategorias

```
public void ReportesCategorias() {
    Document documento = new Document();
    try {
        String ruta = System.getProperty("user.home");
        PdfWriter.getInstance(documento, new FileOutputStream(ruta + "/Desktop/Reporte_Categorias.pdf"));
        Image header = Image.getInstance("src/img/header1.jpg");
        header.scaleToFit(650, 1000);
        header.setAlignment(Chunk.ALIGN_CENTER);
        //formato al texto
        Paragraph parrafo = new Paragraph();
        parrafo.setAlignment(Paragraph.ALIGN_CENTER);
        parrafo.add("Reporte creado por \nEdison Zambrano © Programador Fantasma\n\n");
        parrafo.setFont(FontFactory.getFont("Tahoma", 18, Font.BOLD, BaseColor.DARK_GRAY));
        parrafo.add("Reporte de Categorias \n\n");

        documento.open();
        //agregamos los datos
        documento.add(header);
        documento.add(parrafo);

        PdfPTable tabla = new PdfPTable(3);
        tabla.addCell("Codigo");
        tabla.addCell("Descripcion");
        tabla.addCell("Estado");

        try {
            Connection cn = Conexion.conectar();
            PreparedStatement pst = cn.prepareStatement(
                    "select * from tb_categoria");
            ResultSet rs = pst.executeQuery();
            if (rs.next()) {
                do {
                    tabla.addCell(rs.getString(1));
                    tabla.addCell(rs.getString(2));
                    tabla.addCell(rs.getString(3));
```

- **Propósito** : Genera un reporte PDF de categorías de productos.
- **Diferencias** :
    - La tabla tiene 3 columnas: Código, Descripción y Estado.
    - Se consulta la tabla `tb_categoria`.

4. **Método** ReportesVentas

```
public void ReportesVentas() {
    Document documento = new Document();
    try {
        String ruta = System.getProperty("user.home");
        PdfWriter.getInstance(documento, new FileOutputStream(ruta + "/Desktop/Reporte_Ventas.pdf"));
        Image header = Image.getInstance("src/img/header1.jpg");
        header.scaleToFit(650, 1000);
        header.setAlignment(Chunk.ALIGN_CENTER);
        //formato al texto
        Paragraph parrafo = new Paragraph();
        parrafo.setAlignment(Paragraph.ALIGN_CENTER);
        parrafo.add("Reporte creado por \nEdison Zambrano © Programador Fantasma\n\n");
        parrafo.setFont(FontFactory.getFont("Tahoma", 18, Font.BOLD, BaseColor.DARK_GRAY));
        parrafo.add("Reporte de Ventas \n\n");

        documento.open();
        //agregamos los datos
        documento.add(header);
        documento.add(parrafo);

        float[] columnsWidths = {3, 9, 4, 5, 3};

        PdfPTable tabla = new PdfPTable(columnsWidths);
        tabla.addCell("Codigo");
        tabla.addCell("Cliente");
        tabla.addCell("Tot. Pagar");
        tabla.addCell("Fecha Venta");
        tabla.addCell("Estado");

        try {
            Connection cn = Conexion.conectar();
            PreparedStatement pst = cn.prepareStatement(
                    "select cv.idCabeceraVenta as id, concat(c.nombre, ' ', c.apellido) as cliente, "
                            + "cv.valorPagar as total, cv.fechaVenta as fecha, cv.estado "
                            + "from tb_cabecera_venta as cv, tb_cliente as c "
```

- **Propósito** : Genera un reporte PDF con las ventas registradas.
- **Diferencias** :
    - La tabla tiene 5 columnas: Código, Cliente, Total a Pagar, Fecha Venta y Estado.
    - Se consulta la tabla `tb_cabecera_venta`y se unen datos de `tb_cliente`.

Librerías utilizadas

```
import com.itextpdf.text.BaseColor;
import com.itextpdf.text.Chunk;
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Element;
import com.itextpdf.text.Font;
import com.itextpdf.text.Image;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.Phrase;
import com.itextpdf.text.pdf.PdfPCell;
import com.itextpdf.text.pdf.PdfPTable;
import com.itextpdf.text.pdf.PdfWriter;
import controlador.Conexion;
import java.awt.Desktop;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.text.SimpleDateFormat;
import java.util.Date;
import vista.CrearFactura;
```

- **iTextPDF** : Biblioteca para crear documentos PDF.
- **`com.itextpdf.text`**: Contiene las clases principales para crear y manipular texto, tablas, imágenes y más.
- **`java.awt.Desktop`**: Para abrir el archivo PDF automáticamente después de generarlo.

Método DatosCliente

```
public void DatosCliente(int idCliente) {
    Connection cn = Conexion.conectar();
    String sql = "select * from tb_cliente where idCliente = '" + idCliente + "'";
    Statement st;
    try {

        st = cn.createStatement();
        ResultSet rs = st.executeQuery(sql);
        while (rs.next()) {
            nombreCliente = rs.getString("nombre") + " " + rs.getString("apellido");
            cedulaCliente = rs.getString("cedula");
            telefonoCliente = rs.getString("telefono");
            direccionCliente = rs.getString("direccion");
        }
        cn.close();
    } catch (SQLException e) {
        System.out.println("Error al obtener datos del cliente: " + e);
    }
}
```

Este método extrae los datos de un cliente específico desde la base de datos utilizando un `idCliente`.

- Se conecta a la base de datos usando la clase `Conexion`.
- Ejecute una consulta SQL para obtener los datos del cliente.
- Asigna los valores al cliente (`nombreCliente`, `cedulaCliente`, etc.).

Este método extrae los datos de un cliente específico desde la base de datos utilizando un `idCliente`.

- Se conecta a la base de datos usando la clase `Conexion`.
- Ejecute una consulta SQL para obtener los datos del cliente.
- Asigna los valores al cliente (`nombreCliente`, `cedulaCliente`, etc.).

Método generarFacturaPDF

```
public void generarFacturaPDF() {
    try {

        //cargar la fecha actual
        Date date = new Date();
        fechaActual = new SimpleDateFormat("yyyy/MM/dd").format(date);
        //cambiar el formato de la fecha de / a _
        String fechaNueva = "";
        for (int i = 0; i < fechaActual.length(); i++) {
            if (fechaActual.charAt(i) == '/') {
                fechaNueva = fechaActual.replace("/", "_");
            }
        }

        nombreArchivoPDFVenta = "Venta_" + nombreCliente + "_" + fechaNueva + ".pdf";

        FileOutputStream archivo;
        File file = new File("src/pdf/" + nombreArchivoPDFVenta);
        archivo = new FileOutputStream(file);

        Document doc = new Document();
        PdfWriter.getInstance(doc, archivo);
        doc.open();

        Image img = Image.getInstance("src/img/emblema.jpeg");
        Paragraph fecha = new Paragraph();
        Font negrita = new Font(Font.FontFamily.TIMES_ROMAN, 12, Font.BOLD, BaseColor.BLUE);
        fecha.add(Chunk.NEWLINE); //agregar nueva linea
        fecha.add("Factura: 001" + "\nFecha: " + fechaActual + "\n\n");

        PdfPTable Encabezado = new PdfPTable(4);
        Encabezado.setWidthPercentage(100);
        Encabezado.getDefaultCell().setBorder(0);//quitar el borde de la tabla
```

Genera el archivo PDF con la factura de venta. Este es el núcleo del código.

a) Configuración inicial

- Obtiene la fecha actual y la formatea.
- Define el nombre del archivo PDF basado en el nombre del cliente y la fecha actual.
- Crea un archivo en la ruta `src/pdf/`.

b) Encabezado

- Incluye un logotipo (`emblema.jpeg`).
- Agrega datos de la empresa, como:
    - Nombre
    - RUC (número de identificación fiscal)
    - Dirección
    - Razón social
- Muestra la fecha de la factura.

c) Datos del cliente

Crea una tabla que contiene la información básica del cliente, como:

- Cédula/RUC
- Nombre
- Teléfono
- Dirección

d) Productos

Crea una tabla para listar los productos adquiridos. Incluye:

- Cantidad
- Descripción
- Precio unitario
- Precio total

Los datos se obtienen del componente `jTable_productos`(probablemente una tabla en la interfaz gráfica de la clase `CrearFactura`).

e) Total a pagar

Muestra el total a pagar extraído del campo de texto `CrearFactura.txt_total_pagar`.

f) Mensaje de agradecimiento

Incluye un mensaje de agradecimiento al cliente por su compra.

g) Generación y apertura del PDF

- Cierra el archivo y el documento.
- Abre automáticamente el PDF generado en el visor predeterminado.

Clase EnviarCorreo

```java
import java.util.Properties;
import javax.mail.*;
import javax.mail.internet.*;
import java.io.File;

public class EnviarCorreo {

    public static void enviarPDFPorCorreo(String destinatario, String asunto, String cuerpo, String rutaPDF) {
        final String correoEmisor = "tu_correo@gmail.com"; // Cambia esto
        final String claveCorreo = "tu_contraseña"; // Cambia esto

        // Configuración del servidor SMTP
        Properties propiedades = new Properties();
        propiedades.put("mail.smtp.host", "smtp.gmail.com");
        propiedades.put("mail.smtp.port", "587");
        propiedades.put("mail.smtp.auth", "true");
        propiedades.put("mail.smtp.starttls.enable", "true");

        // Crear la sesión
        Session sesion = Session.getInstance(propiedades, new Authenticator() {
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(correoEmisor, claveCorreo);
            }
        });

        try {
            // Crear el mensaje
            MimeMessage mensaje = new MimeMessage(sesion);
            mensaje.setFrom(new InternetAddress(correoEmisor));
            mensaje.addRecipient(Message.RecipientType.TO, new InternetAddress(destinatario));
            mensaje.setSubject(asunto);

            // Crear el cuerpo del mensaje
            MimeBodyPart cuerpoMensaje = new MimeBodyPart();
            cuerpoMensaje.setText(cuerpo);
```

Importación de Librerías

- `Properties`: Permite configurar las propiedades del servidor de correo (como SMTP).
- `javax.mail.*`: Proporciona las clases para enviar y manejar correos electrónicos.
- `javax.mail.internet.*`: Proporciona las clases para construir el contenido del correo, como `MimeMessage`y `MimeBodyPart`.
- `java.io.File`: Se usa para manejar la ruta del archivo PDF a adjuntar.

Método enviarPDFPorCorreo

Parámetros:

1. **`destinatario`**: Dirección de correo electrónico a la que se envía el mensaje.
2. **`asunto`**:Asunto del correo.
3. **`cuerpo`**: Contenido del mensaje en formato texto.
4. **`rutaPDF`**: Ruta local del archivo PDF que se va a adjuntar.

Configuración del Servidor SMTP

El código utiliza **Gmail SMTP** como servidor para enviar correos electrónicos.

Propiedades del servidor SMTP:

Java
Copiar código

```
propiedades.put("mail.smtp.host", "smtp.gmail.com");   // Servidor SMTP de Gmail
```

```
propiedades.put("mail.smtp.port", "587");              // Puerto de salida para TLS
propiedades.put("mail.smtp.auth", "true");             // Habilita autenticación SMTP
propiedades.put("mail.smtp.starttls.enable", "true");  // Habilita STARTTLS
```

Creación de la sesión

```
Session sesion = Session.getInstance(propiedades, new Authenticator() {
    protected PasswordAuthentication getPasswordAuthentication() {
        return new PasswordAuthentication(correoEmisor, claveCorreo);
```

La sesión se crea con el método `Session.getInstance` pasando las propiedades y un **`Authenticator`**.
O `Authenticator` devuelve un **`PasswordAuthentication`**con el correo emisor y la contraseña.

Construcción del mensaje

El mensaje se compone usando las clases **`MimeMessage`**, **`MimeBodyPart`**y **`Multipart`**.

1. Creación del mensaje base:

```
MimeMessage mensaje = new MimeMessage(sesion);
mensaje.setFrom(new InternetAddress(correoEmisor));
mensaje.addRecipient(Message.RecipientType.TO, new InternetAddress(destinatario));
mensaje.setSubject(asunto);
```

- **`MimeMessage`**: Representa el correo completo.
- **`setFrom`**:Configurar el emisor.
- **`addRecipient`**:Configurar el destinatario.

- **`setSubject`**: Defina el asunto del correo.

Creación del cuerpo del mensaje:

```
// Crear el cuerpo del mensaje
MimeBodyPart cuerpoMensaje = new MimeBodyPart();
cuerpoMensaje.setText(cuerpo);
```

`MimeBodyPart`: Representa una parte del contenido del correo (texto).

Añadir archivo PDF:

```
MimeBodyPart adjunto = new MimeBodyPart();
adjunto.attachFile(new File(rutaPDF));
```

Utilice `attachFile()`para adjuntar el archivo PDF.

Combinar el cuerpo y el archivo adjunto:

```
Multipart multiparte = new MimeMultipart();
multiparte.addBodyPart(cuerpoMensaje);
multiparte.addBodyPart(adjunto);
```

public class ActualizarStock

Constructor:ActualizarStock()

- Inicializa la ventana y sus componentes con initComponents()(generado por el editor visual de NetBeans).
- Configura el título y tamaño de la ventana.
- Llama al método CargarComboProductos()para llenar el jComboBoxcon los nombres de los productos desde la base de datos.

CargarComboProductos()

Este método realiza lo siguiente:

1. **Conexión** a la base de datos usando Conexion.conectar().
2. Ejecuta la consulta SQL:

   sql
   Copiar código
   SELECT * FROM tb_producto

3. Llena el jComboBox_productocon los nombres de los productos obtenidos.
4. Maneja errores con un bloque try-catche imprime el error si ocurre.

MostrarStock()

Se ejecuta cuando el usuario selecciona un producto del jComboBox.

1. Consulta la base de datos para obtener:
   - El **ID** (idProducto) del producto.
   - La **cantidad real** del producto.
2. La cantidad se muestra en txt_cantidad_actual.
3. Si no se encuentra el producto, el campo txt_cantidad_actualqueda vacío.