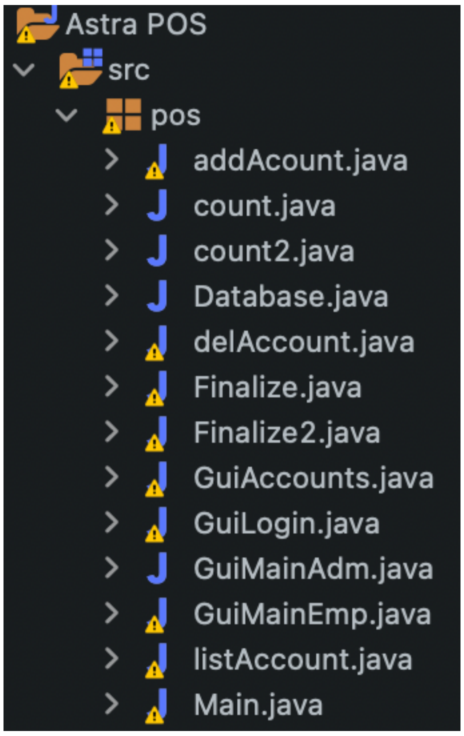
Criterion C: Development

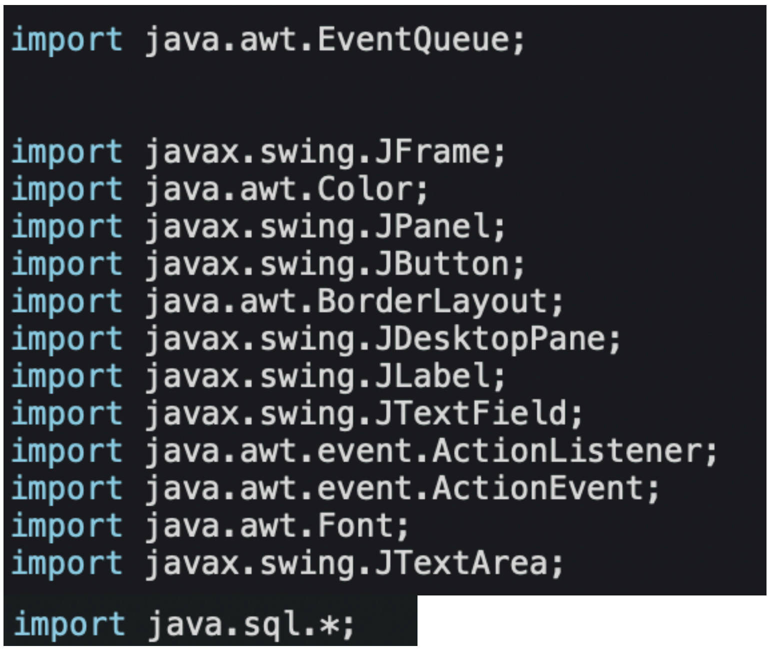
<u>All Classes</u>:



```
Astra POS
  src
    pos
      addAcount.java
      count.java
      count2.java
      Database.java
      delAccount.java
      Finalize.java
      Finalize2.java
      GuiAccounts.java
      GuiLogin.java
      GuiMainAdm.java
      GuiMainEmp.java
      listAccount.java
      Main.java
```

<u>Techniques used:</u>

Libraries Imported



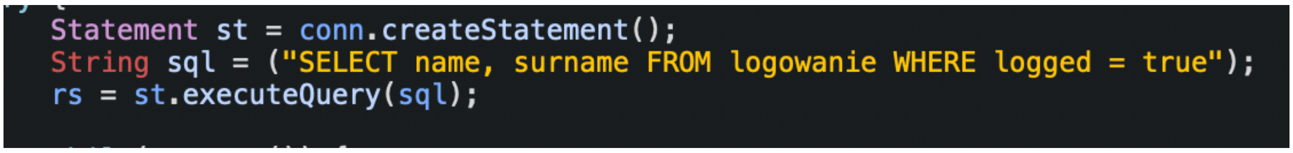
```java
import java.awt.EventQueue;


import javax.swing.JFrame;
import java.awt.Color;
import javax.swing.JPanel;
import javax.swing.JButton;
import java.awt.BorderLayout;
import javax.swing.JDesktopPane;
import javax.swing.JLabel;
import javax.swing.JTextField;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.Font;
import javax.swing.JTextArea;

import java.sql.*;
```

Figure 1 - Imported Libraries

awt and swing are used for graphical components

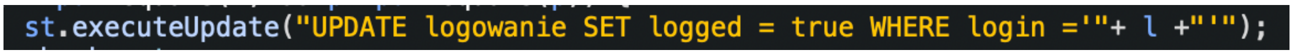
SQL(Structured Query Language) is used for the program to connect to the PostrgreSQL database, which is used to hold the data.

SQL commands used



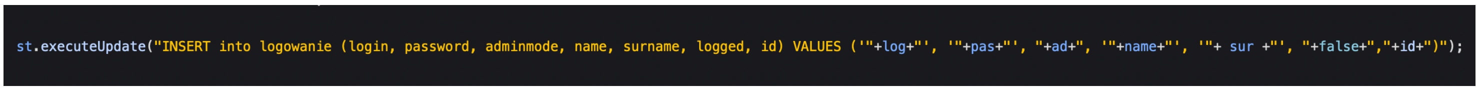
```
Statement st = conn.createStatement();
String sql = ("SELECT name, surname FROM logowanie WHERE logged = true");
rs = st.executeQuery(sql);
```

Figure 2 - Updating rows



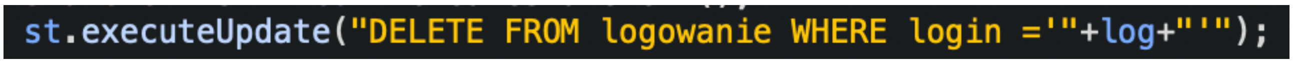
```
st.executeUpdate("UPDATE logowanie SET logged = true WHERE login ='"+ l +"'");
```

Figure 3 - Inserting rows into database



```
st.executeUpdate("INSERT into logowanie (login, password, adminmode, name, surname, logged, id) VALUES ('"+log+"', '"+pas+"', "+ad+", '"+name+"', '"+ sur +"', "+false+","+id+")");
```

Figure 4 - Deleting rows



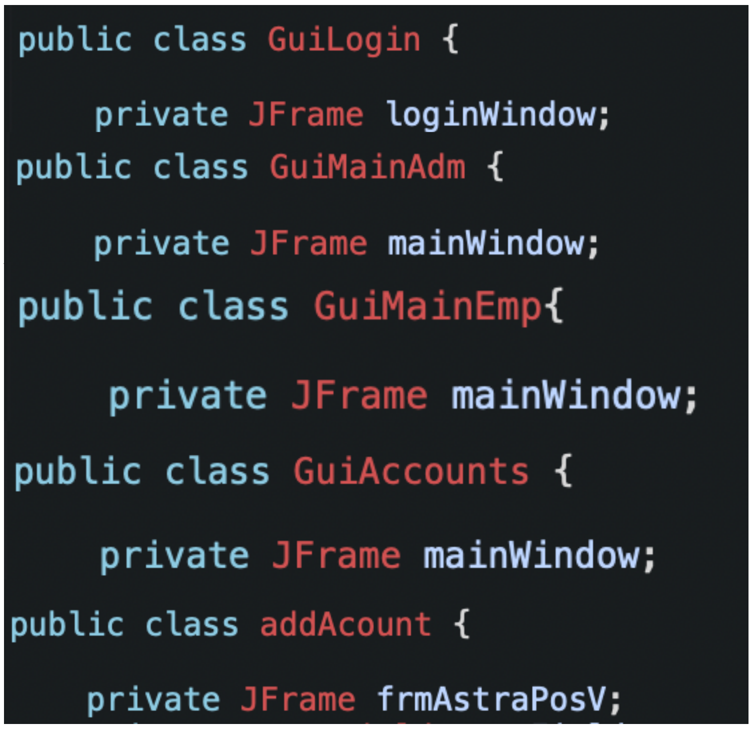
```
st.executeUpdate("DELETE FROM logowanie WHERE login ='"+log+"'");
```

Figure 5 - Reading fields from table

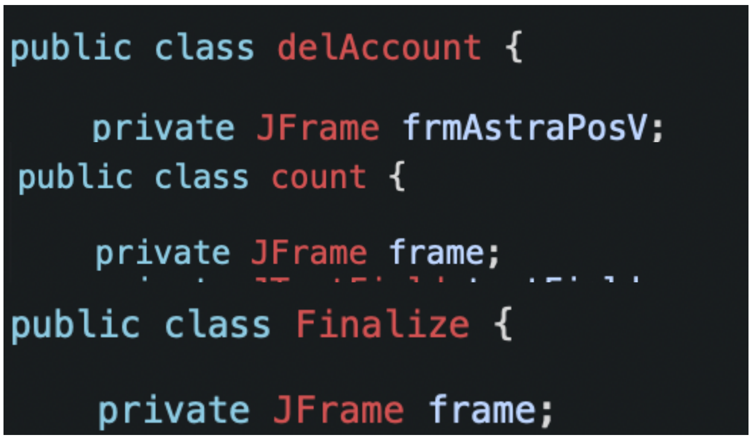
Production of the GUI

GUI was practically spread over various classes

Using JFrames



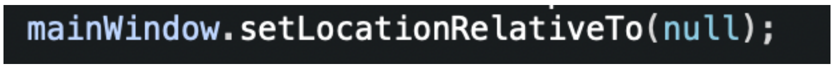
```
public class GuiLogin {

    private JFrame loginWindow;
public class GuiMainAdm {

    private JFrame mainWindow;
public class GuiMainEmp{

    private JFrame mainWindow;
public class GuiAccounts {

    private JFrame mainWindow;
public class addAcount {

    private JFrame frmAstraPosV;
```



```
public class delAccount {

    private JFrame frmAstraPosV;
public class count {

    private JFrame frame;
public class Finalize {

    private JFrame frame;
```

Every class created new JFrame, so that I could easily see the distinction between each application window

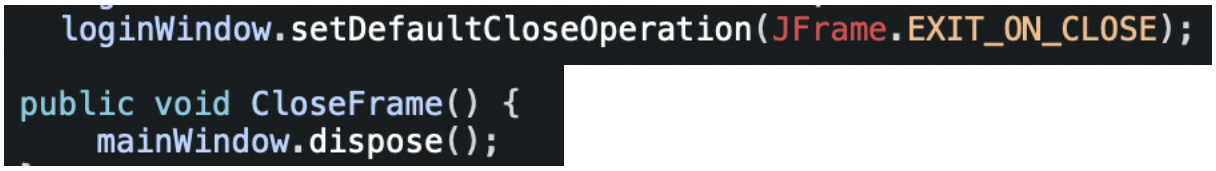
Centering JFrames



```
mainWindow.setLocationRelativeTo(null);
```

To center JFrame on the screen, I used the setLocationRelativeTo(null) method, which was used throughout program.

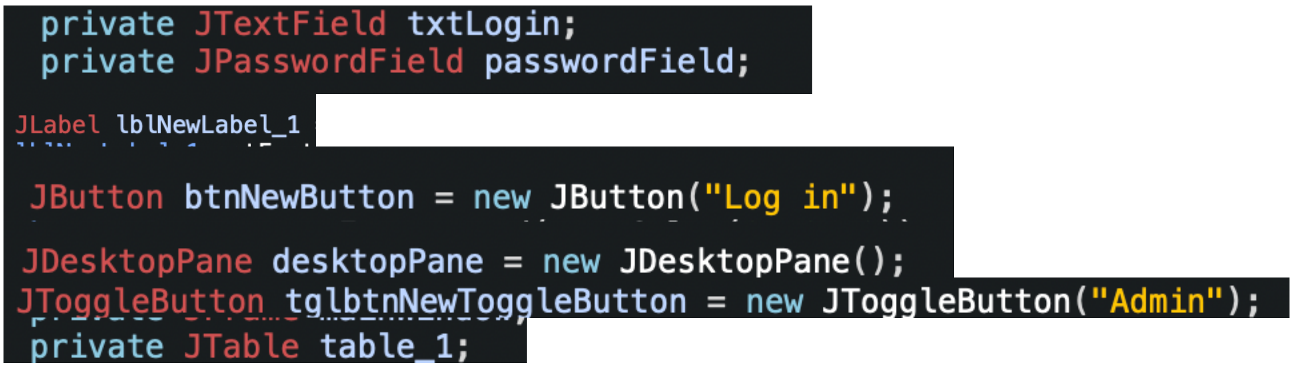
Closing Frames



```
  loginWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

public void CloseFrame() {
    mainWindow.dispose();
```

The only class that can close the program is the loginWindow, which is the starting window of the application. Other windows are disposing, using the CloseFrame() method

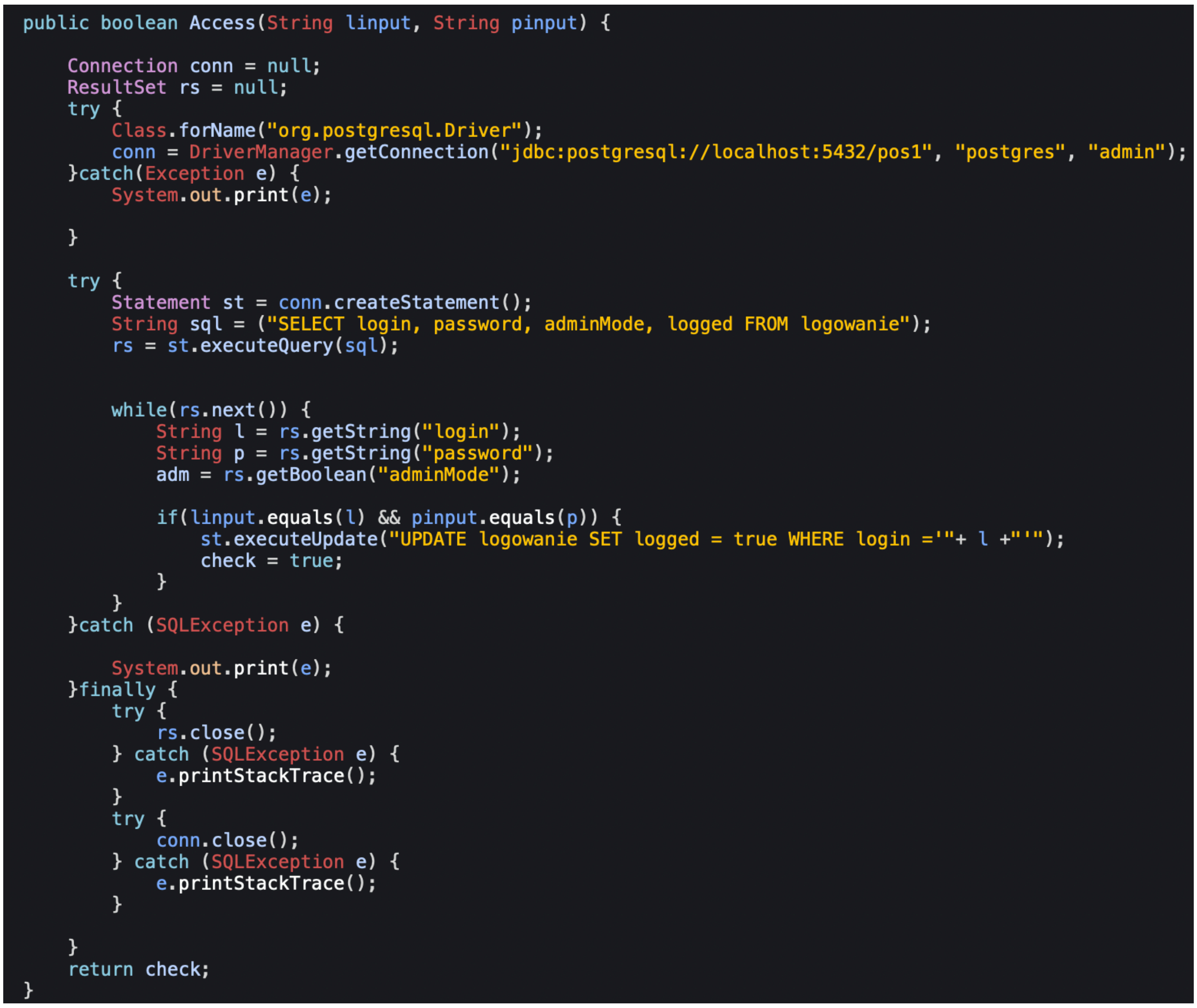
Adding other swing components



```
private JTextField txtLogin;
private JPasswordField passwordField;

JLabel lblNewLabel_1

JButton btnNewButton = new JButton("Log in");

JDesktopPane desktopPane = new JDesktopPane();
JToggleButton tglbtnNewToggleButton = new JToggleButton("Admin");
private JTable table_1;
```

To position components in content panes manually I used an absolute layout and components were positioned using setBounds method which allowed me to enter the x-coordinate and y-coordinate of the components. JTextField were used to scan the user input, JPasswordField were used to scan and read the input password in the login window. JButtons were used to run different methods and move between application windows. JDesktopPane were used as a content pane for GuiMainAdm and GuiMainEmp window for better look of the UI. JToggleButton was used to get the boolean for type of account added by user. JTable were used to display the list of accounts with the limit of 4 accounts.

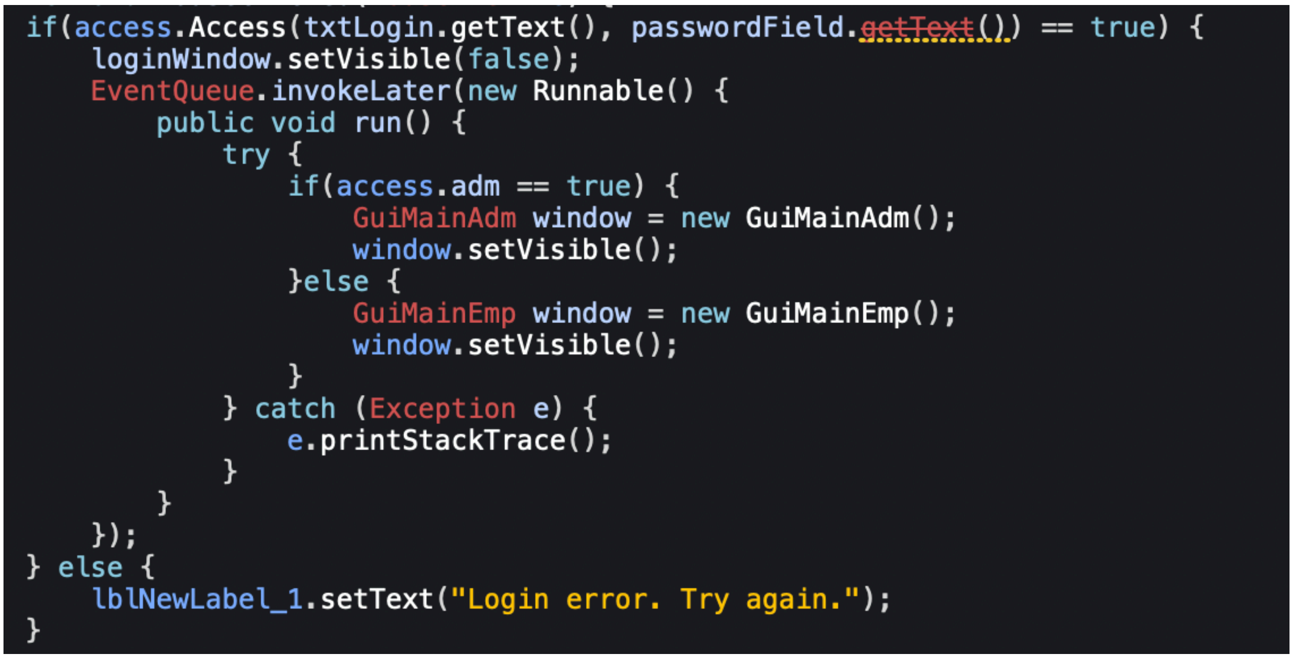
Getting login and password from the database and comparing it with the input



```java
public boolean Access(String linput, String pinput) {

    Connection conn = null;
    ResultSet rs = null;
    try {
        Class.forName("org.postgresql.Driver");
        conn = DriverManager.getConnection("jdbc:postgresql://localhost:5432/pos1", "postgres", "admin");
    }catch(Exception e) {
        System.out.print(e);

    }

    try {
        Statement st = conn.createStatement();
        String sql = ("SELECT login, password, adminMode, logged FROM logowanie");
        rs = st.executeQuery(sql);


        while(rs.next()) {
            String l = rs.getString("login");
            String p = rs.getString("password");
            adm = rs.getBoolean("adminMode");

            if(linput.equals(l) && pinput.equals(p)) {
                st.executeUpdate("UPDATE logowanie SET logged = true WHERE login ='"+ l +"'");
                check = true;
            }
        }
    }catch (SQLException e) {

        System.out.print(e);
    }finally {
        try {
            rs.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }

    }
    return check;
}
```

Figure 8 - method Access(String linput, String pinput) in Database.java

The method Access() searches the database logowanie and compares the input values of linput and pinput to the values of login and password in the database. The method returns the boolean check, which if the input values equal database values is true, and when the function does not find any comparison it return false.

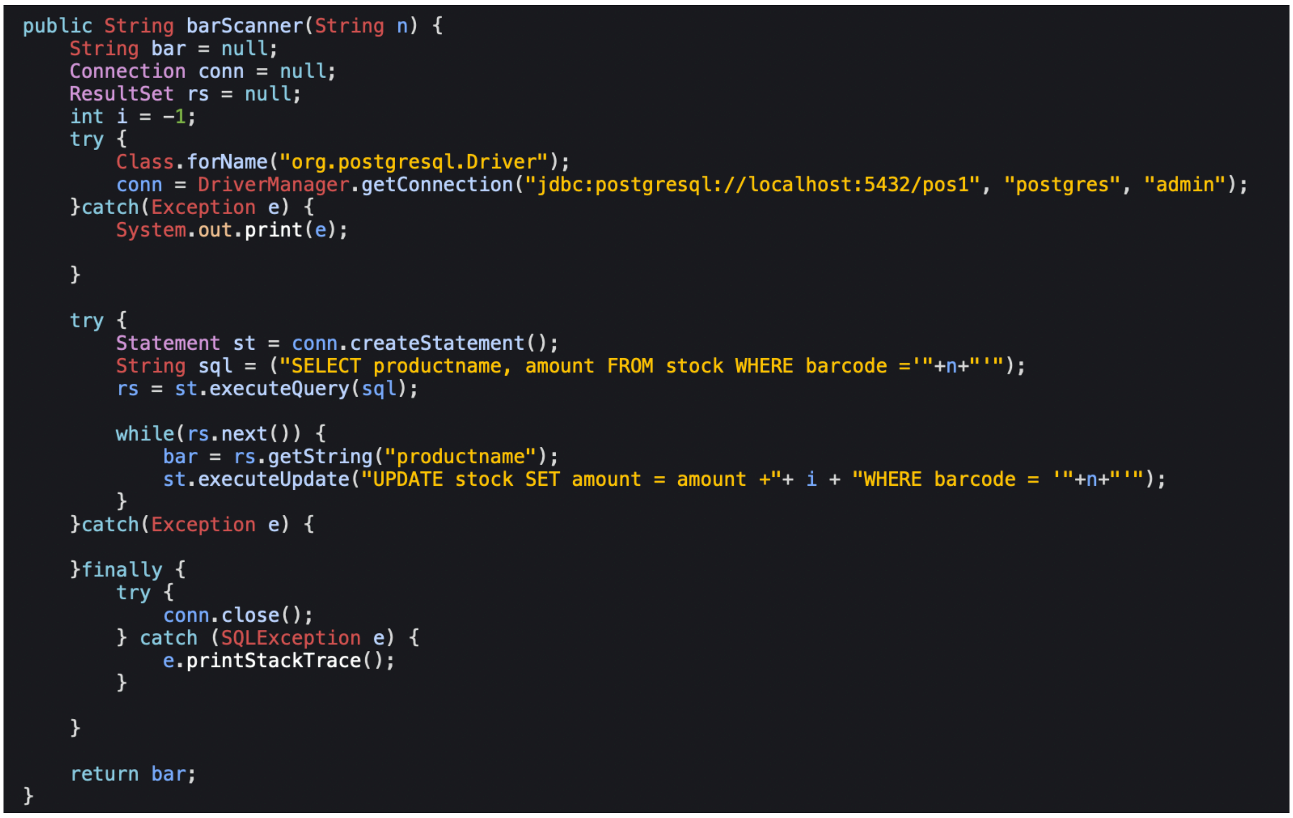


```java
if(access.Access(txtLogin.getText(), passwordField.getText()) == true) {
    loginWindow.setVisible(false);
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                if(access.adm == true) {
                    GuiMainAdm window = new GuiMainAdm();
                    window.setVisible();
                }else {
                    GuiMainEmp window = new GuiMainEmp();
                    window.setVisible();
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
} else {
    lblNewLabel_1.setText("Login error. Try again.");
}
```

Figure 9 - appliance of the Access() method in the guiLogin.java

In the guiLogin class, the input values of login and password are put into the access method and compared with database. If returned boolean is true, the method checks the type of account boolean and displays proper window, either GuiMainAdm or GuiMainEmp. If the returned boolean is false, the method displays the error message.

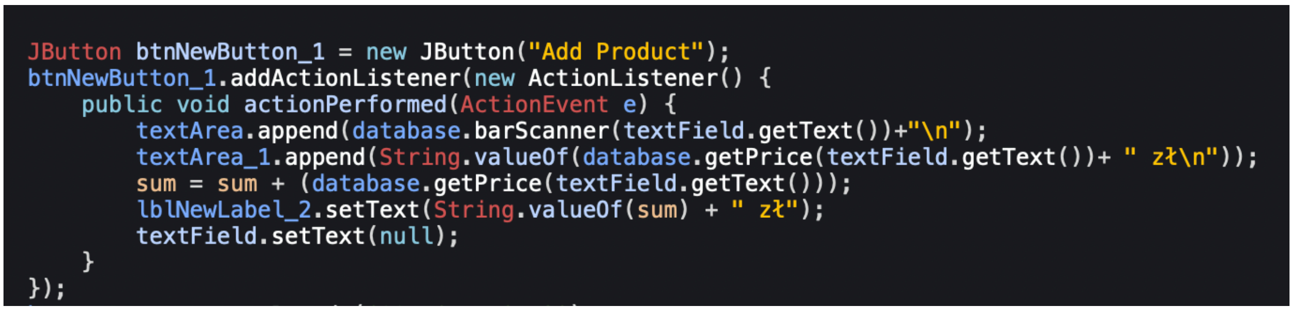
Scanning the barcode and displaying name and price of the product



```java
public String barScanner(String n) {
    String bar = null;
    Connection conn = null;
    ResultSet rs = null;
    int i = -1;
    try {
        Class.forName("org.postgresql.Driver");
        conn = DriverManager.getConnection("jdbc:postgresql://localhost:5432/pos1", "postgres", "admin");
    }catch(Exception e) {
        System.out.print(e);

    }

    try {
        Statement st = conn.createStatement();
        String sql = ("SELECT productname, amount FROM stock WHERE barcode ='"+n+"'");
        rs = st.executeQuery(sql);

        while(rs.next()) {
            bar = rs.getString("productname");
            st.executeUpdate("UPDATE stock SET amount = amount +"+ i + "WHERE barcode = '"+n+"'");
        }
    }catch(Exception e) {

    }finally {
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }

    }

    return bar;
}
```

Figure 10 - method barScanner(String n) in Database.java

Method barScanner() searches in the database for the productname and price, which have a barcode equal the input n.

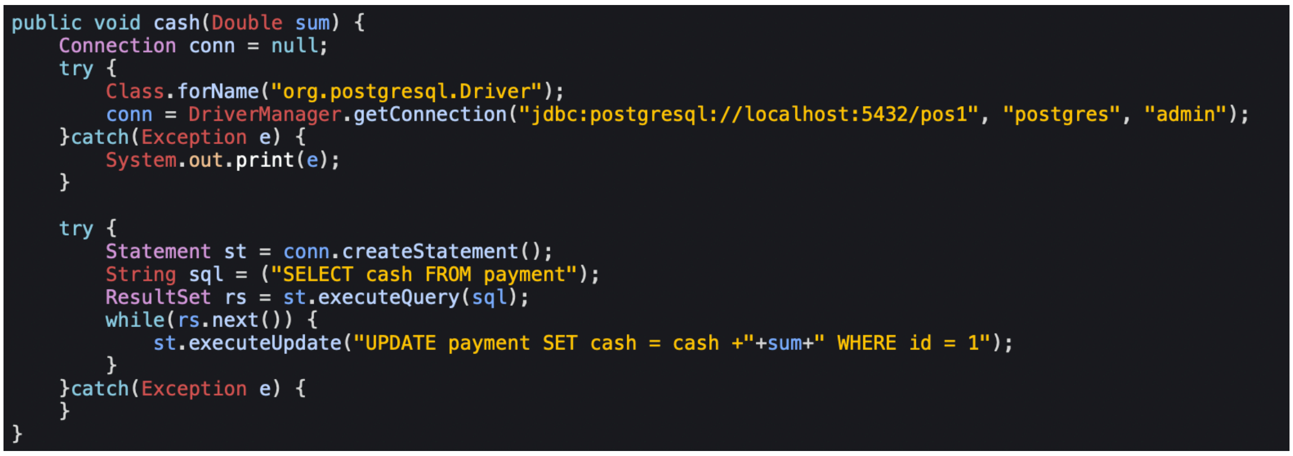


```java
JButton btnNewButton_1 = new JButton("Add Product");
btnNewButton_1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        textArea.append(database.barScanner(textField.getText())+"\n");
        textArea_1.append(String.valueOf(database.getPrice(textField.getText())+ " zł\n"));
        sum = sum + (database.getPrice(textField.getText()));
        lblNewLabel_2.setText(String.valueOf(sum) + " zł");
        textField.setText(null);
    }
});
```

Figure 11 - appliance of the barScanner() method in GuiMainAdm.java

In GuiMainAdm class, the input from the JTextField is compared to the barcode from stock database. The barScanner() method returns the productname that has a barcode equal to the JTextField input and displays it in the textArea. The other function getPrice(), which works the same as barScanner(), is responsible for returning the price value for the barcode equal to JTextField input, which is later displayed is TextArea_1. The sum is a static double that counts the sum of the transaction.

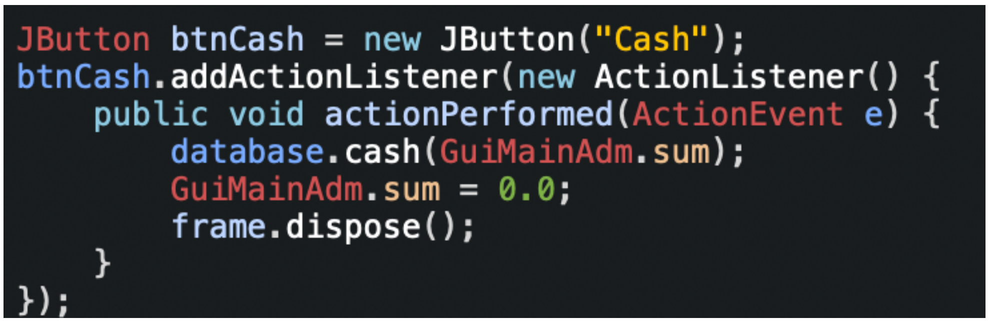
Updating the amount of money from transaction into database



```java
public void cash(Double sum) {
    Connection conn = null;
    try {
        Class.forName("org.postgresql.Driver");
        conn = DriverManager.getConnection("jdbc:postgresql://localhost:5432/pos1", "postgres", "admin");
    }catch(Exception e) {
        System.out.print(e);
    }

    try {
        Statement st = conn.createStatement();
        String sql = ("SELECT cash FROM payment");
        ResultSet rs = st.executeQuery(sql);
        while(rs.next()) {
            st.executeUpdate("UPDATE payment SET cash = cash +"+sum+" WHERE id = 1");
        }
    }catch(Exception e) {
    }
}
```

Figure 12 - method cash(Double sum) in Database.java

The method cash(Double sum) is used to update the value of cash in the database by adding the value of static double sum.

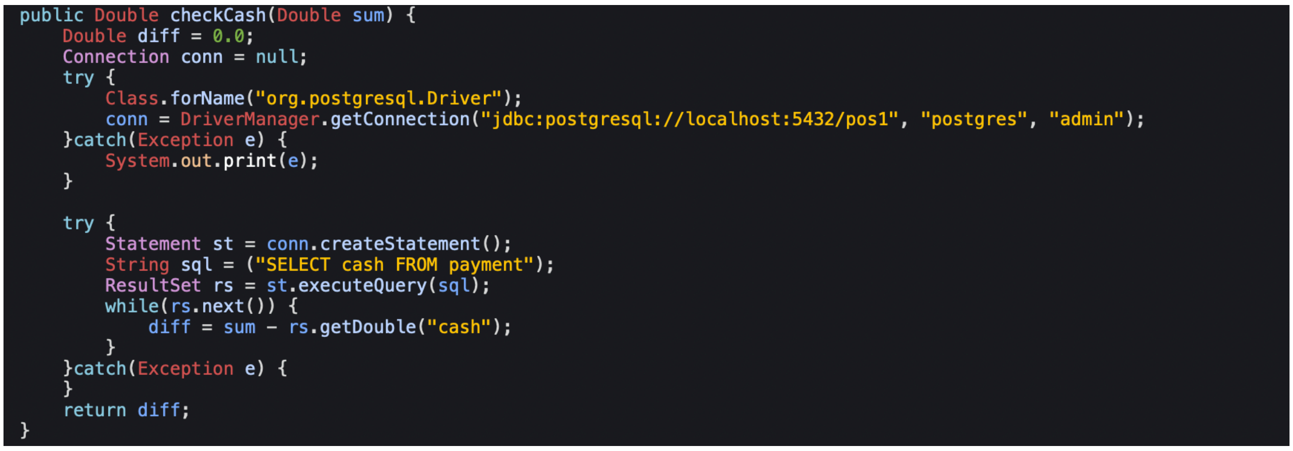


```java
JButton btnCash = new JButton("Cash");
btnCash.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        database.cash(GuiMainAdm.sum);
        GuiMainAdm.sum = 0.0;
        frame.dispose();
    }
});
```

Figure 13 - appliance of the cash() method in Finalize.java

In Finalize class, the static double sum from GuiMainAdm and puts it into the cash() method. Then the static double sum is zeroed for the next transaction. There is an equivalent of Finalize class, Finalize2, which is used for GuiMainEmp and uses its static double sum. There is also an equivalent of cash() method, card(), which is used for the same purpose as cash() method, but it counts the money from credit card transactions.

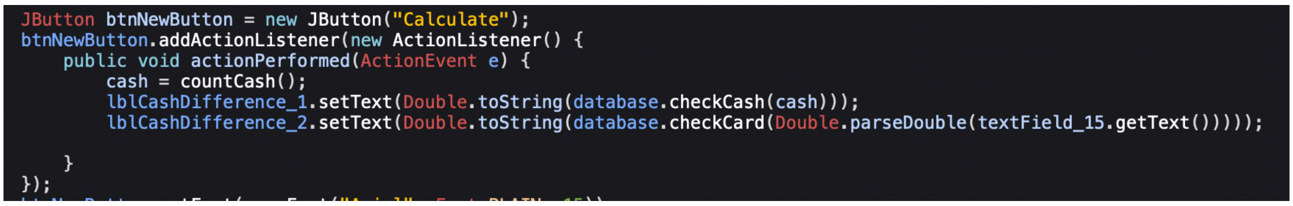
Counting the difference in money in the cash register to the estimated money from transactions



```java
public Double checkCash(Double sum) {
    Double diff = 0.0;
    Connection conn = null;
    try {
        Class.forName("org.postgresql.Driver");
        conn = DriverManager.getConnection("jdbc:postgresql://localhost:5432/pos1", "postgres", "admin");
    }catch(Exception e) {
        System.out.print(e);
    }

    try {
        Statement st = conn.createStatement();
        String sql = ("SELECT cash FROM payment");
        ResultSet rs = st.executeQuery(sql);
        while(rs.next()) {
            diff = sum - rs.getDouble("cash");
        }
    }catch(Exception e) {
    }
    return diff;
}
```

Figure 14 - method checkCash(Double sum) in the Database.java

This method is used to compare the input from counted bills to the amount of cash in the database that should be in the cash register. The function return the difference of those values. There is an equivalent of this function, checkCard(Double sum), which works the same as checkCash(), but return difference in money from credit card transactions

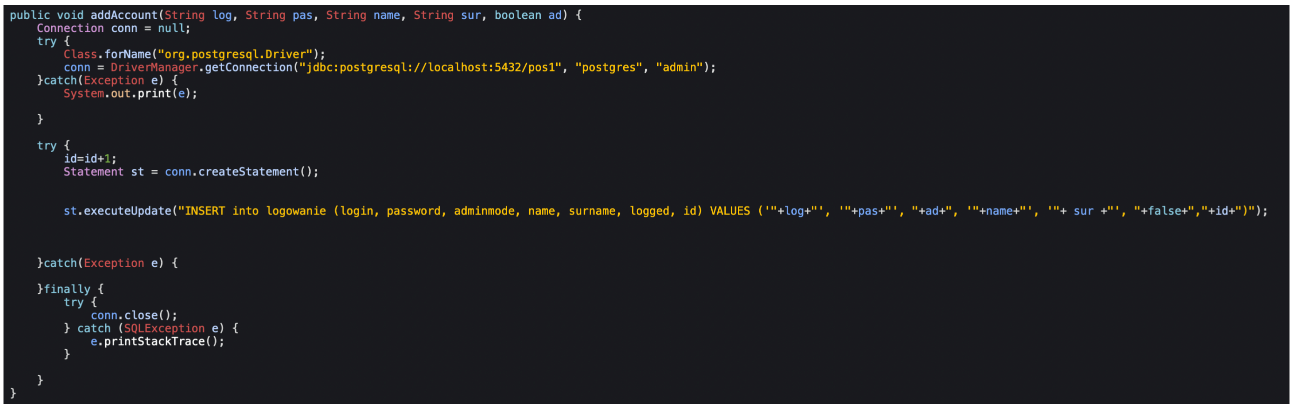


```java
JButton btnNewButton = new JButton("Calculate");
btnNewButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        cash = countCash();
        lblCashDifference_1.setText(Double.toString(database.checkCash(cash)));
        lblCashDifference_2.setText(Double.toString(database.checkCard(Double.parseDouble(textField_15.getText()))));

    }
});
```

Figure 15 - implementation of checkCard() and checkCash() in the count.java

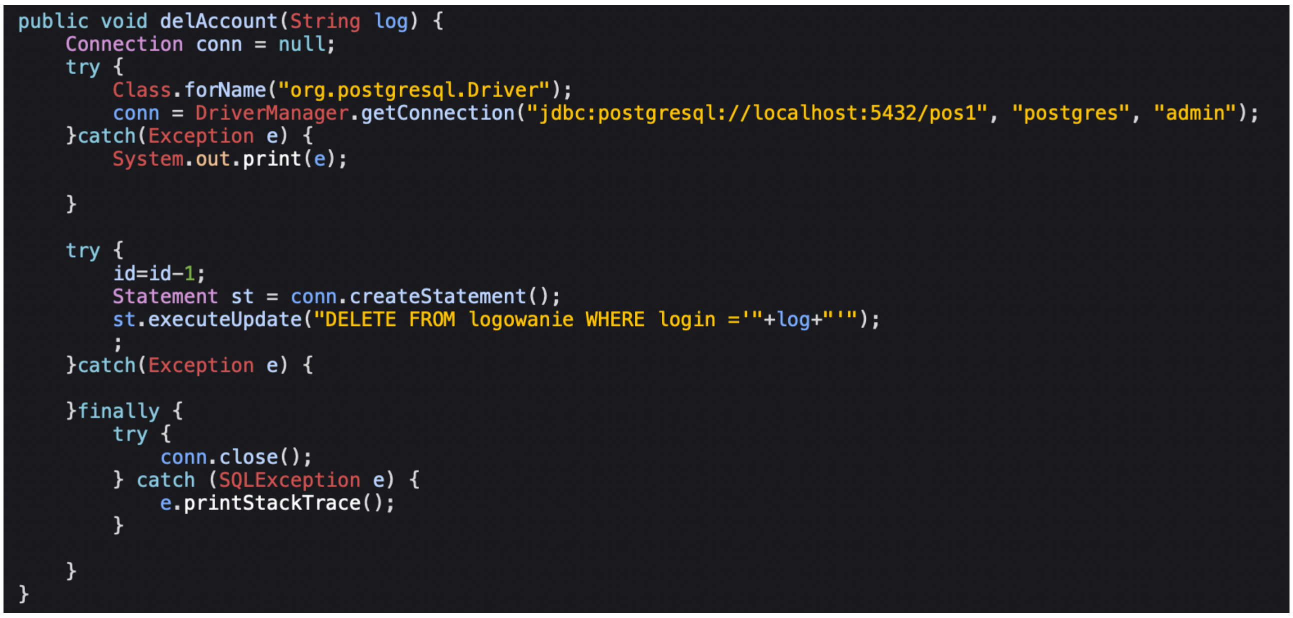
There is an equivalent of count.java, count2.java, which is used for the user type account in comparison to count class, which is used for admin type account.

Other methods worth mentioning



```java
public void addAccount(String log, String pas, String name, String sur, boolean ad) {
    Connection conn = null;
    try {
        Class.forName("org.postgresql.Driver");
        conn = DriverManager.getConnection("jdbc:postgresql://localhost:5432/pos1", "postgres", "admin");
    }catch(Exception e) {
        System.out.print(e);

    }

    try {
        id=id+1;
        Statement st = conn.createStatement();


        st.executeUpdate("INSERT into logowanie (login, password, adminmode, name, surname, logged, id) VALUES ('"+log+"', '"+pas+"', "+ad+", '"+name+"', '"+ sur +"', "+false+","+id+")");



    }catch(Exception e) {

    }finally {
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }

    }
}
```

Figure 16 - method addAccount() in Database class, used to insert new account into a database



```java
public void delAccount(String log) {
    Connection conn = null;
    try {
        Class.forName("org.postgresql.Driver");
        conn = DriverManager.getConnection("jdbc:postgresql://localhost:5432/pos1", "postgres", "admin");
    }catch(Exception e) {
        System.out.print(e);

    }

    try {
        id=id-1;
        Statement st = conn.createStatement();
        st.executeUpdate("DELETE FROM logowanie WHERE login ='"+log+"'");
        ;
    }catch(Exception e) {

    }finally {
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }

    }
}
```

Figure 17 - method delAccount() in Database class, used to delete account from a database by login.

Word Count: 823