Funds Manager

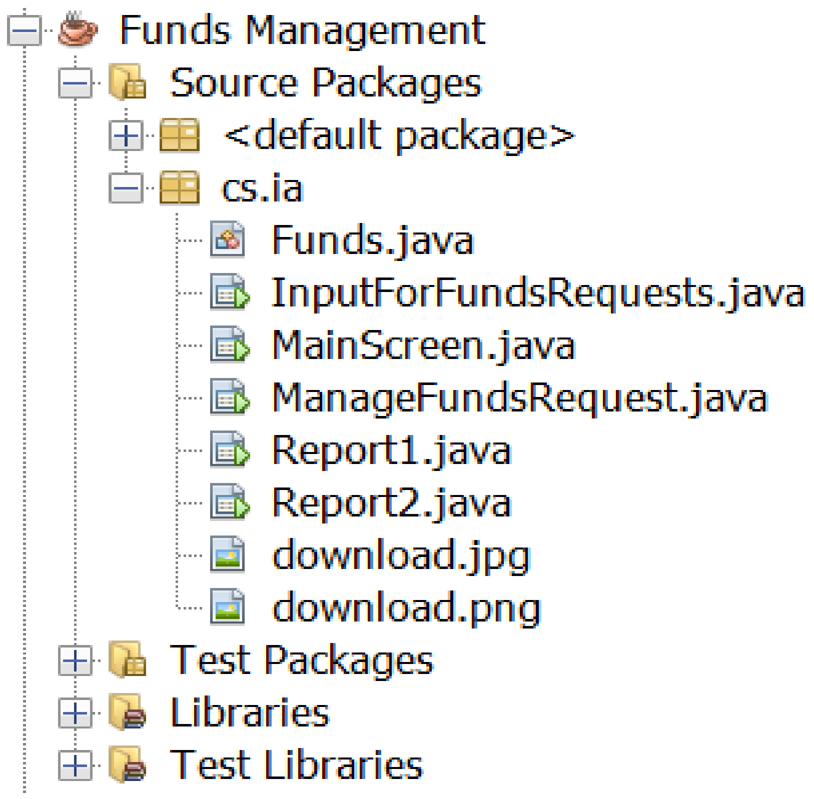
Criterion C: Development

Executable JAR File

Funds_Manager.jar

Figure 1: Executable JAR File

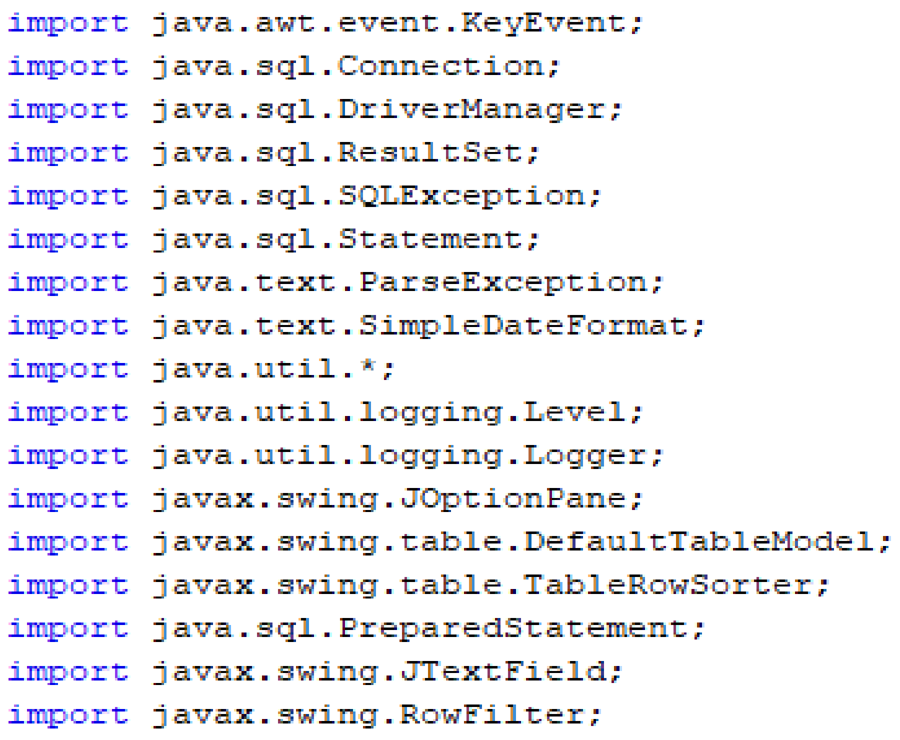
Classes and Folder Structure



```
Funds Management
  Source Packages
    <default package>
    cs.ia
      Funds.java
      InputForFundsRequests.java
      MainScreen.java
      ManageFundsRequest.java
      Report1.java
      Report2.java
      download.jpg
      download.png
  Test Packages
  Libraries
  Test Libraries
```

Figure 2: Program Classes and Folder Structure

Imported Libraries



```
import java.awt.event.KeyEvent;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableRowSorter;
import java.sql.PreparedStatement;
import javax.swing.JTextField;
import javax.swing.RowFilter;
```

Figure 3: Imported Libraries

The input/output (io) imports are used in order to input images such as the company's logo.

Swing and awt are used in order to enhance the user interface by using certain graphical components available, such as the calendar and buttons.

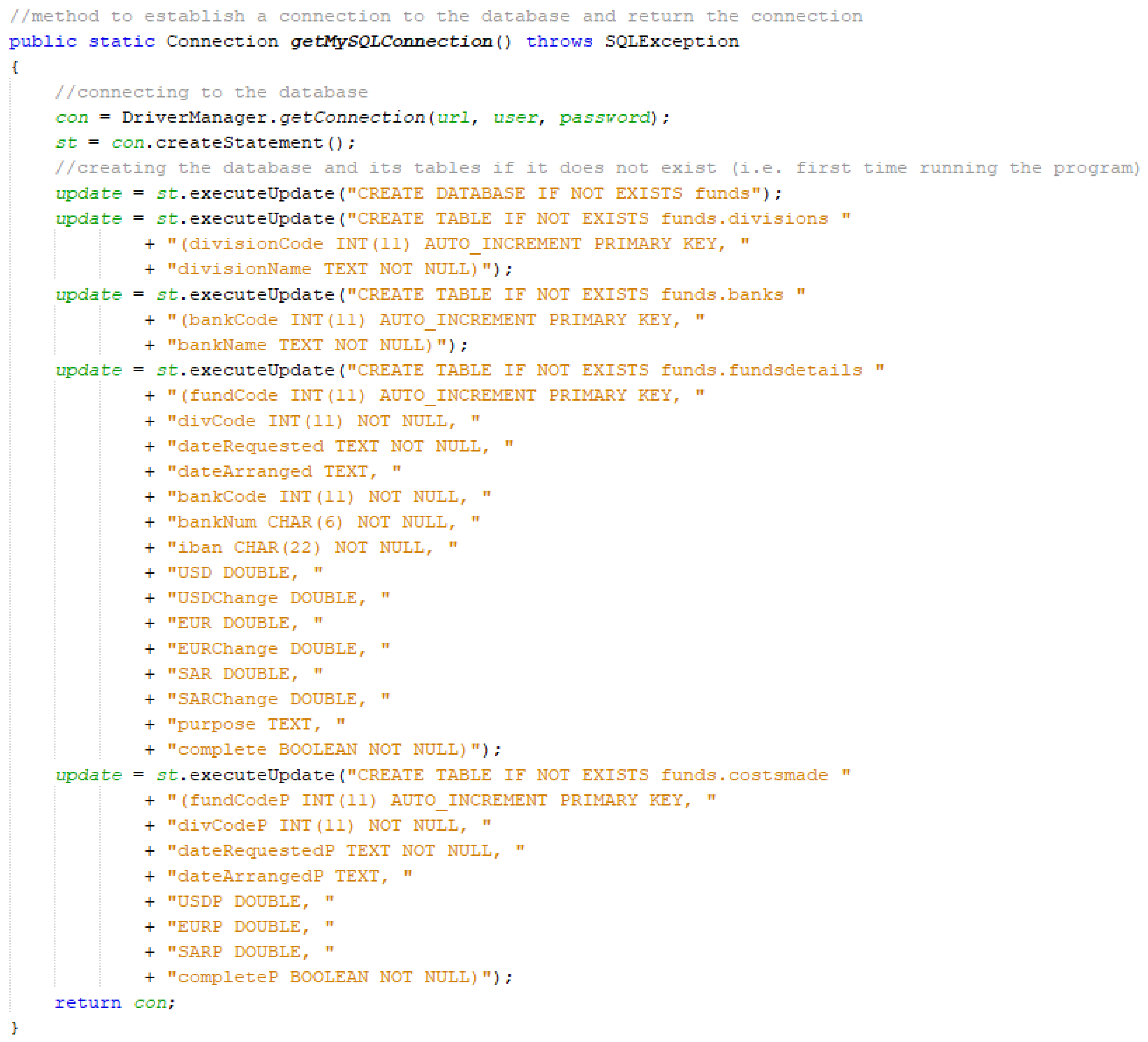
The Connection, Driver Manager, SQLException, Statement, and PreparedStatement classes were used as they were essential in order to transfer data to and from SQL. I am using MySQL as my server to retrieve, update or delete data.

List of Techniques

1. Object arrays
2. Static Polymorphism
3. Structured Query Language (SQL)
4. Static methods and variables
5. Encapsulation
6. Swing GUI

7. Use of additional libraries

Connecting to MySQL Database



```
//method to establish a connection to the database and return the connection
public static Connection getMySQLConnection() throws SQLException
{
    //connecting to the database
    con = DriverManager.getConnection(url, user, password);
    st = con.createStatement();
    //creating the database and its tables if it does not exist (i.e. first time running the program)
    update = st.executeUpdate("CREATE DATABASE IF NOT EXISTS funds");
    update = st.executeUpdate("CREATE TABLE IF NOT EXISTS funds.divisions "
            + "(divisionCode INT(11) AUTO_INCREMENT PRIMARY KEY, "
            + "divisionName TEXT NOT NULL)");
    update = st.executeUpdate("CREATE TABLE IF NOT EXISTS funds.banks "
            + "(bankCode INT(11) AUTO_INCREMENT PRIMARY KEY, "
            + "bankName TEXT NOT NULL)");
    update = st.executeUpdate("CREATE TABLE IF NOT EXISTS funds.fundsdetails "
            + "(fundCode INT(11) AUTO_INCREMENT PRIMARY KEY, "
            + "divCode INT(11) NOT NULL, "
            + "dateRequested TEXT NOT NULL, "
            + "dateArranged TEXT, "
            + "bankCode INT(11) NOT NULL, "
            + "bankNum CHAR(6) NOT NULL, "
            + "iban CHAR(22) NOT NULL, "
            + "USD DOUBLE, "
            + "USDChange DOUBLE, "
            + "EUR DOUBLE, "
            + "EURChange DOUBLE, "
            + "SAR DOUBLE, "
            + "SARChange DOUBLE, "
            + "purpose TEXT, "
            + "complete BOOLEAN NOT NULL)");
    update = st.executeUpdate("CREATE TABLE IF NOT EXISTS funds.costsmade "
            + "(fundCodeP INT(11) AUTO_INCREMENT PRIMARY KEY, "
            + "divCodeP INT(11) NOT NULL, "
            + "dateRequestedP TEXT NOT NULL, "
            + "dateArrangedP TEXT, "
            + "USDP DOUBLE, "
            + "EURP DOUBLE, "
            + "SARP DOUBLE, "
            + "completeP BOOLEAN NOT NULL)");
    return con;
}
```

Figure 4: Method to connect to the MySQL localhost, then create the database and tables in case they do not exist

Panel 1 (Main Screen/Home Page)

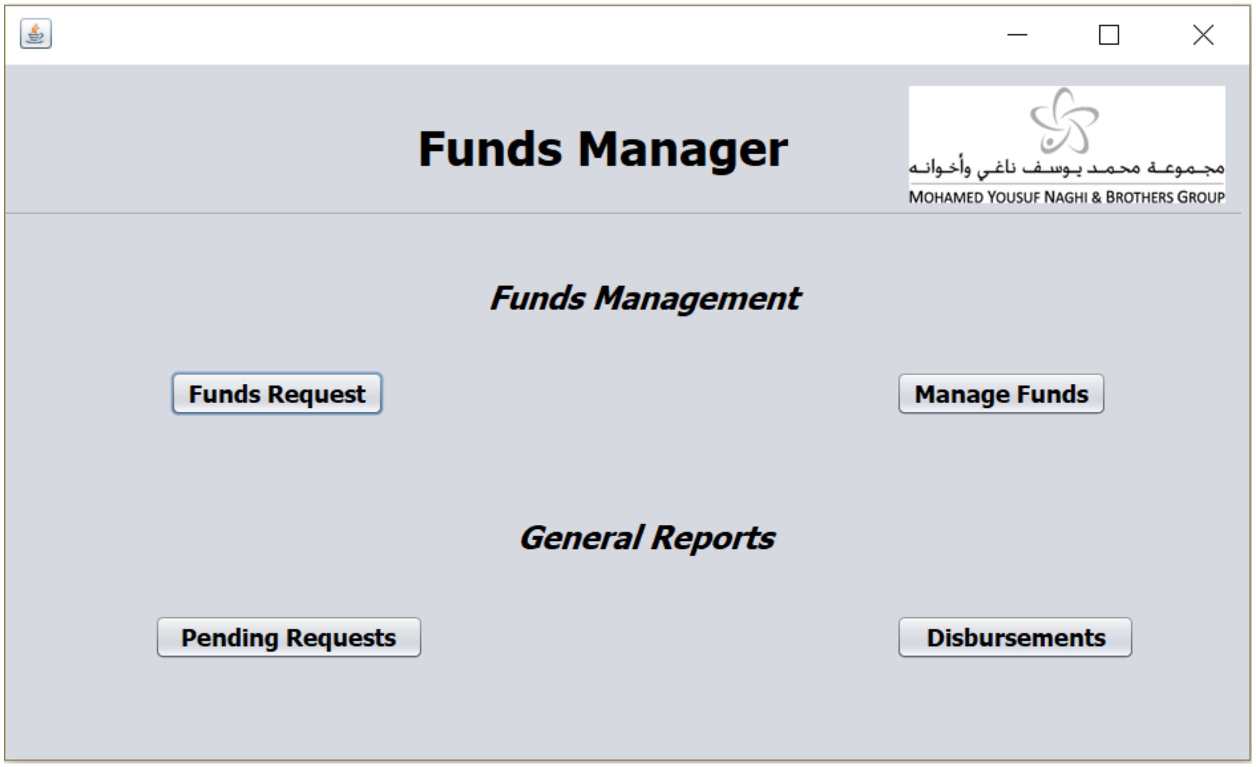
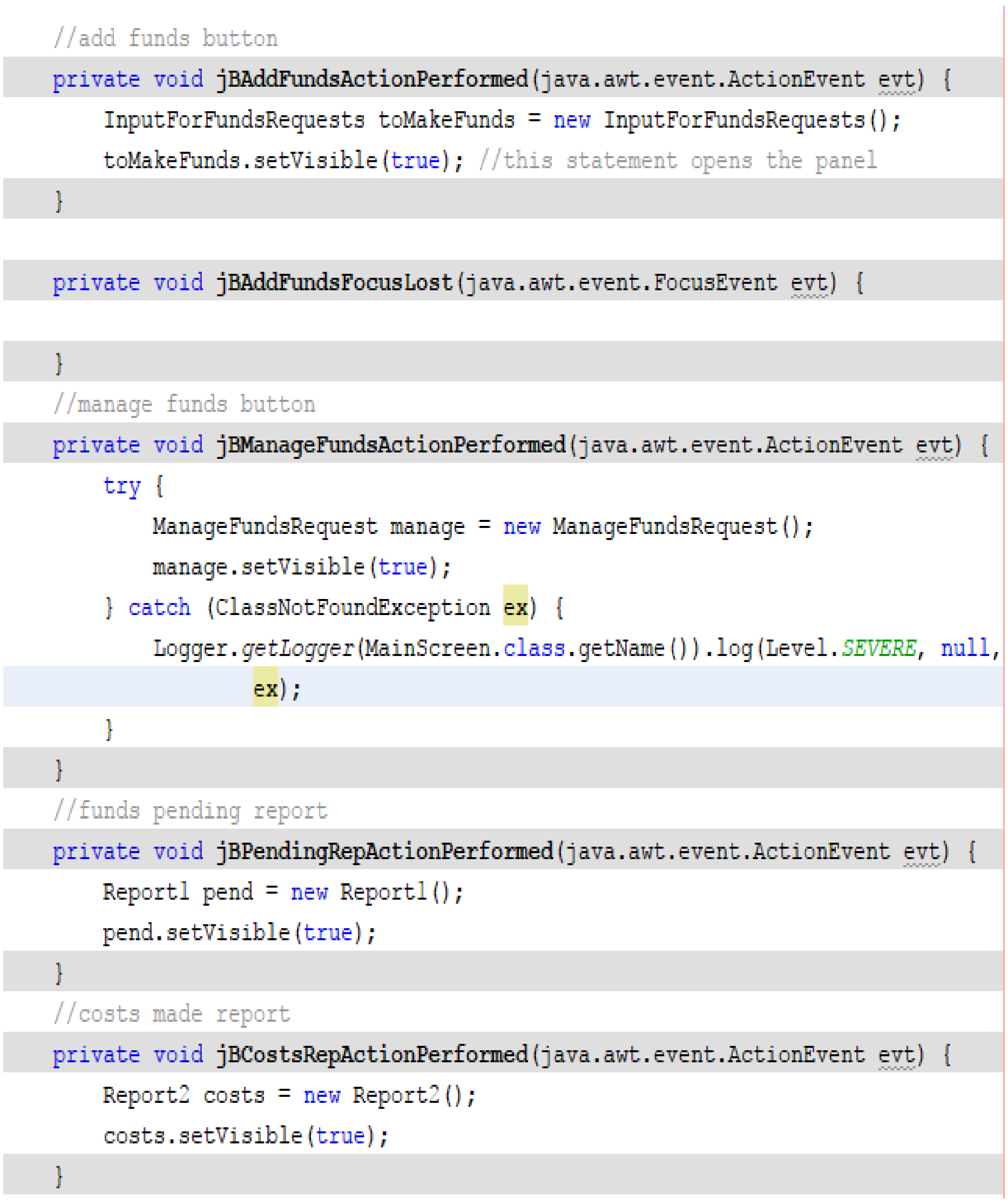


Figure 5: Main screen showing 4 buttons which links to 4 other JFrame that will be used in this project. Two of the buttons under the 'Manage' heading as shown lead to panels that manage funds that come to the business. The other 2 present reports to the user of the program

The following actions (Figure 6) are performed when either of the buttons are clicked on (Figure 5)



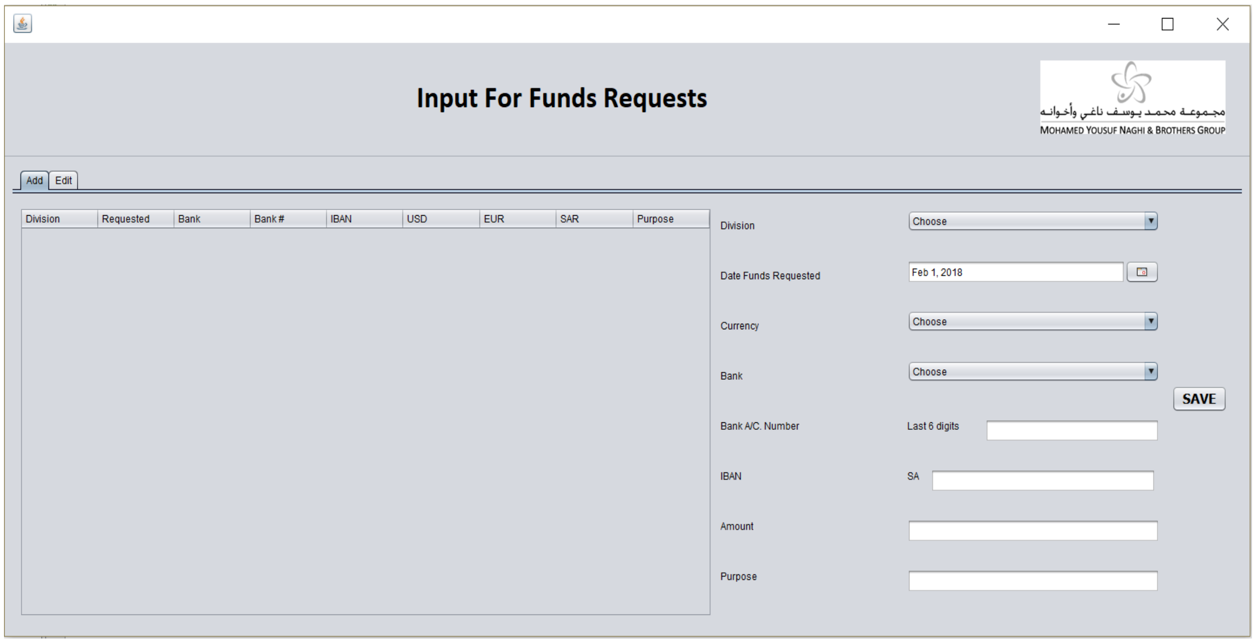
```
    //add funds button
    private void jBAddFundsActionPerformed(java.awt.event.ActionEvent evt) {
        InputForFundsRequests toMakeFunds = new InputForFundsRequests();
        toMakeFunds.setVisible(true); //this statement opens the panel
    }

    private void jBAddFundsFocusLost(java.awt.event.FocusEvent evt) {

    }
    //manage funds button
    private void jBManageFundsActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            ManageFundsRequest manage = new ManageFundsRequest();
            manage.setVisible(true);
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(MainScreen.class.getName()).log(Level.SEVERE, null,
                    ex);
        }
    }
    //funds pending report
    private void jBPendingRepActionPerformed(java.awt.event.ActionEvent evt) {
        Report1 pend = new Report1();
        pend.setVisible(true);
    }
    //costs made report
    private void jBCostsRepActionPerformed(java.awt.event.ActionEvent evt) {
        Report2 costs = new Report2();
        costs.setVisible(true);
    }
```

Figure 6: Code created to open the different panel based on whichever button the user clicks on

Panel 2 (Funds Request)



Division	Requested	Bank	Bank #	IBAN	USD	EUR	SAR	Purpose

Division	Choose
Date Funds Requested	Feb 1, 2018
Currency	Choose
Bank	Choose
Bank A/C. Number	Last 6 digits
IBAN	SA
Amount	
Purpose	

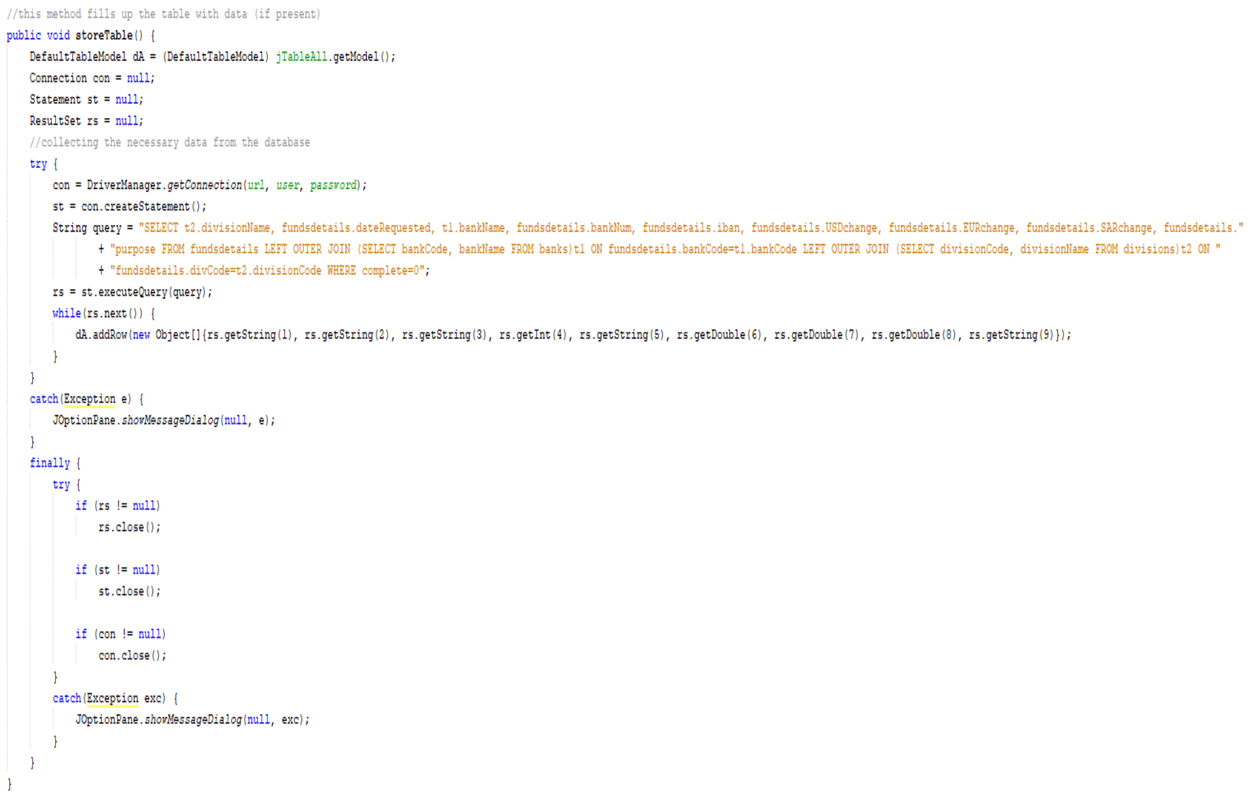
Figure 7: When the user clicks on the Funds Request button (Figure 5), they will be sent to this panel, where the user is then able to add funds

To fetch data from the database and then load onto the table, I used different Table class methods in order to complete this task. I also used multiple SQL methods in one single query to complete the required search (Figure 9). I also used the methods to sort the table (Figure 8).

```
public void sortTable()
{
    //using the DefaultTableModel methods to sort
    DefaultTableModel dE = (DefaultTableModel) jTableEdit.getModel();
    TableRowSorter<DefaultTableModel> so = new TableRowSorter<DefaultTableModel> (dE);
    jTableEdit.setRowSorter(so);

    DefaultTableModel dA = (DefaultTableModel) jTableAll.getModel();
    TableRowSorter<DefaultTableModel> sort = new TableRowSorter<DefaultTableModel> (dA);
    jTableAll.setRowSorter(sort);
}
```

Figure 8: Method to sort the table by selecting the column headings



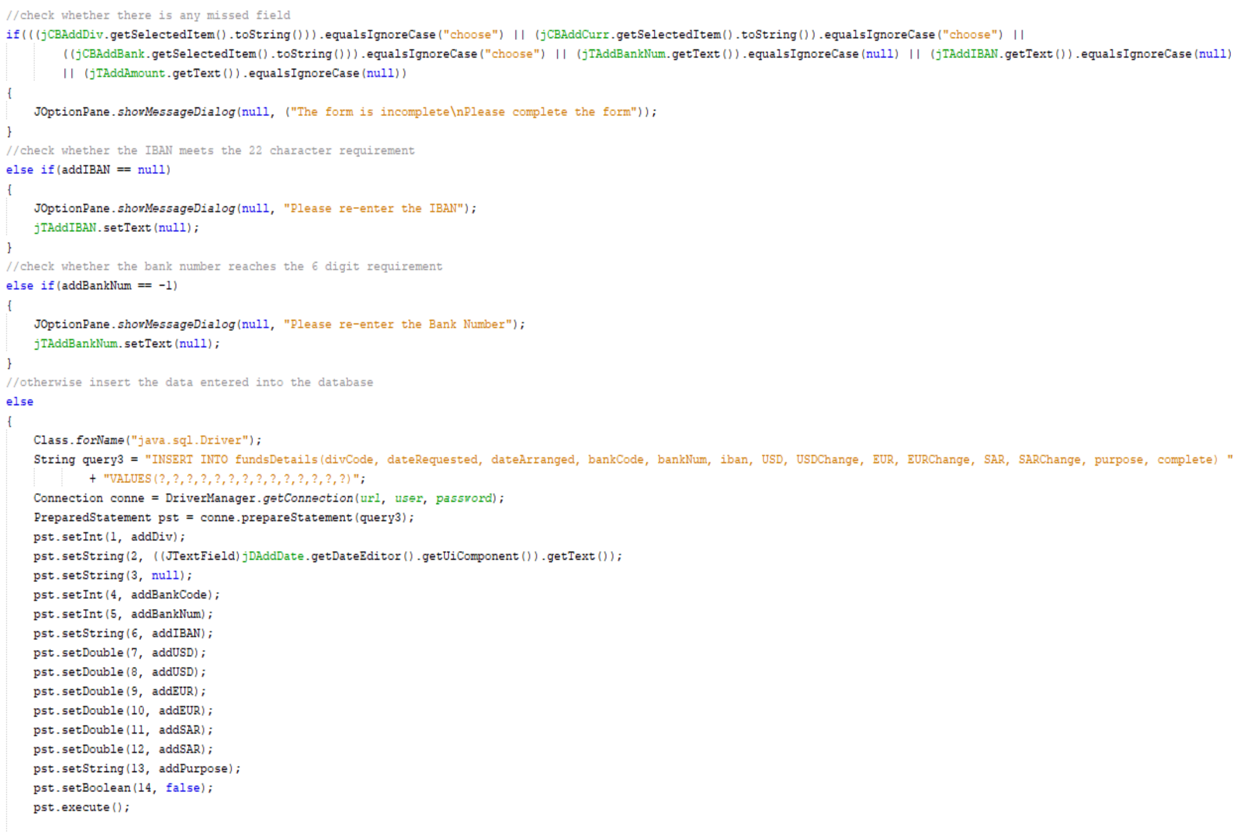
```
//this method fills up the table with data (if present)
public void storeTable() {
    DefaultTableModel dA = (DefaultTableModel) jTableAll.getModel();
    Connection con = null;
    Statement st = null;
    ResultSet rs = null;
    //collecting the necessary data from the database
    try {
        con = DriverManager.getConnection(url, user, password);
        st = con.createStatement();
        String query = "SELECT t2.divisionName, fundsdetails.dateRequested, t1.bankName, fundsdetails.bankNum, fundsdetails.iban, fundsdetails.USDchange, fundsdetails.EURchange, fundsdetails.SARchange, fundsdetails."
                + "purpose FROM fundsdetails LEFT OUTER JOIN (SELECT bankCode, bankName FROM banks)t1 ON fundsdetails.bankCode=t1.bankCode LEFT OUTER JOIN (SELECT divisionCode, divisionName FROM divisions)t2 ON "
                + "fundsdetails.divCode=t2.divisionCode WHERE complete=0";
        rs = st.executeQuery(query);
        while(rs.next()) {
            dA.addRow(new Object[]{rs.getString(1), rs.getString(2), rs.getString(3), rs.getInt(4), rs.getString(5), rs.getDouble(6), rs.getDouble(7), rs.getDouble(8), rs.getString(9)});
        }
    }
    catch(Exception e) {
        JOptionPane.showMessageDialog(null, e);
    }
    finally {
        try {
            if (rs != null)
                rs.close();

            if (st != null)
                st.close();

            if (con != null)
                con.close();
        }
        catch(Exception exc) {
            JOptionPane.showMessageDialog(null, exc);
        }
    }
}
```

Figure 9: The code to store the table with data from the database.

The 'Save' button in Figure 7 saves the entry by the user. Considerable error checks have been made in order to ensure that the user is entering valid data (Figure 10). The code checks whether or not the IBAN, and Bank Number have been input validly, and it also checks whether or not all the necessary fields have been filled.



```java
//check whether there is any missed field
if(((jCBAddDiv.getSelectedItem().toString())).equalsIgnoreCase("choose") || (jCBAddCurr.getSelectedItem().toString()).equalsIgnoreCase("choose") ||
        ((jCBAddBank.getSelectedItem().toString())).equalsIgnoreCase("choose") || (jTAddBankNum.getText()).equalsIgnoreCase(null) || (jTAddIBAN.getText()).equalsIgnoreCase(null)
        || (jTAddAmount.getText()).equalsIgnoreCase(null))
{
    JOptionPane.showMessageDialog(null, ("The form is incomplete\nPlease complete the form"));
}
//check whether the IBAN meets the 22 character requirement
else if(addIBAN == null)
{
    JOptionPane.showMessageDialog(null, "Please re-enter the IBAN");
    jTAddIBAN.setText(null);
}
//check whether the bank number reaches the 6 digit requirement
else if(addBankNum == -1)
{
    JOptionPane.showMessageDialog(null, "Please re-enter the Bank Number");
    jTAddBankNum.setText(null);
}
//otherwise insert the data entered into the database
else
{
    Class.forName("java.sql.Driver");
    String query3 = "INSERT INTO fundsDetails(divCode, dateRequested, dateArranged, bankCode, bankNum, iban, USD, USDChange, EUR, EURChange, SAR, SARChange, purpose, complete) "
            + "VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?)";
    Connection conne = DriverManager.getConnection(url, user, password);
    PreparedStatement pst = conne.prepareStatement(query3);
    pst.setInt(1, addDiv);
    pst.setString(2, ((JTextField)jDAddDate.getDateEditor().getUiComponent()).getText());
    pst.setString(3, null);
    pst.setInt(4, addBankCode);
    pst.setInt(5, addBankNum);
    pst.setString(6, addIBAN);
    pst.setDouble(7, addUSD);
    pst.setDouble(8, addUSD);
    pst.setDouble(9, addEUR);
    pst.setDouble(10, addEUR);
    pst.setDouble(11, addSAR);
    pst.setDouble(12, addSAR);
    pst.setString(13, addPurpose);
    pst.setBoolean(14, false);
    pst.execute();
```

Figure 10: Error checking and the method to save the new fund request input by the user

Panel 3 (Edit Panel)

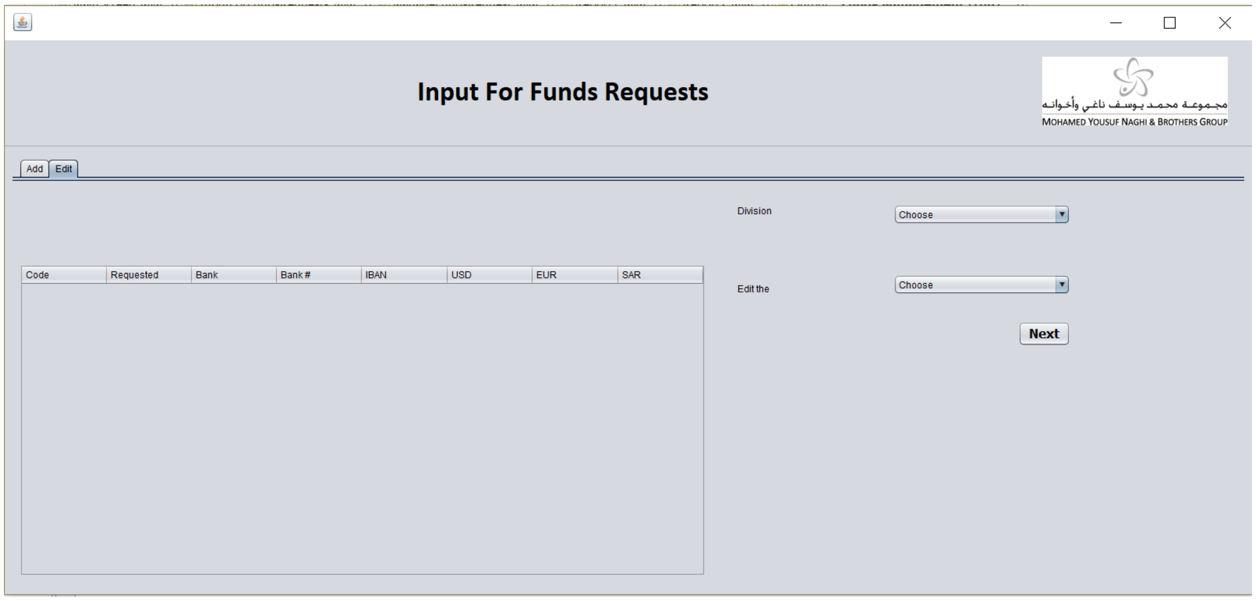
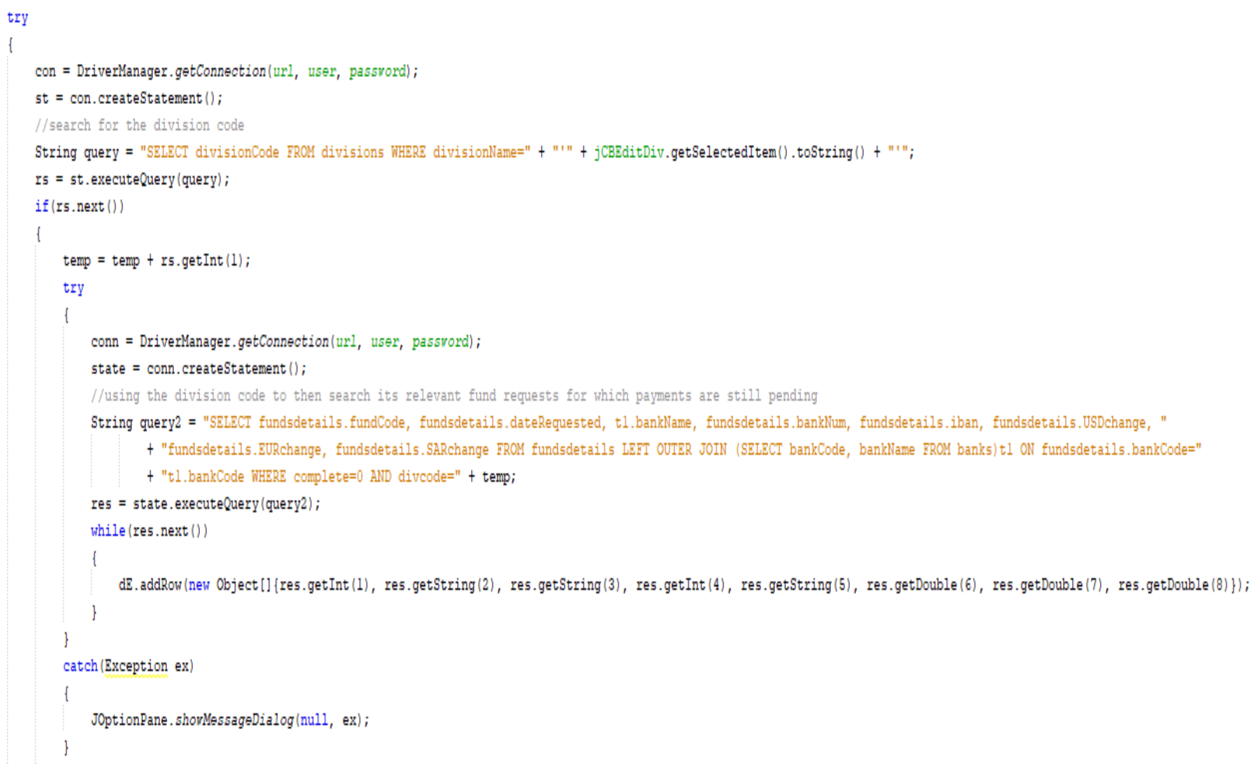


Figure 11: The edit panel. This is where the user is given the option to edit any fund request they wish. The search is made simpler by searching based on the division

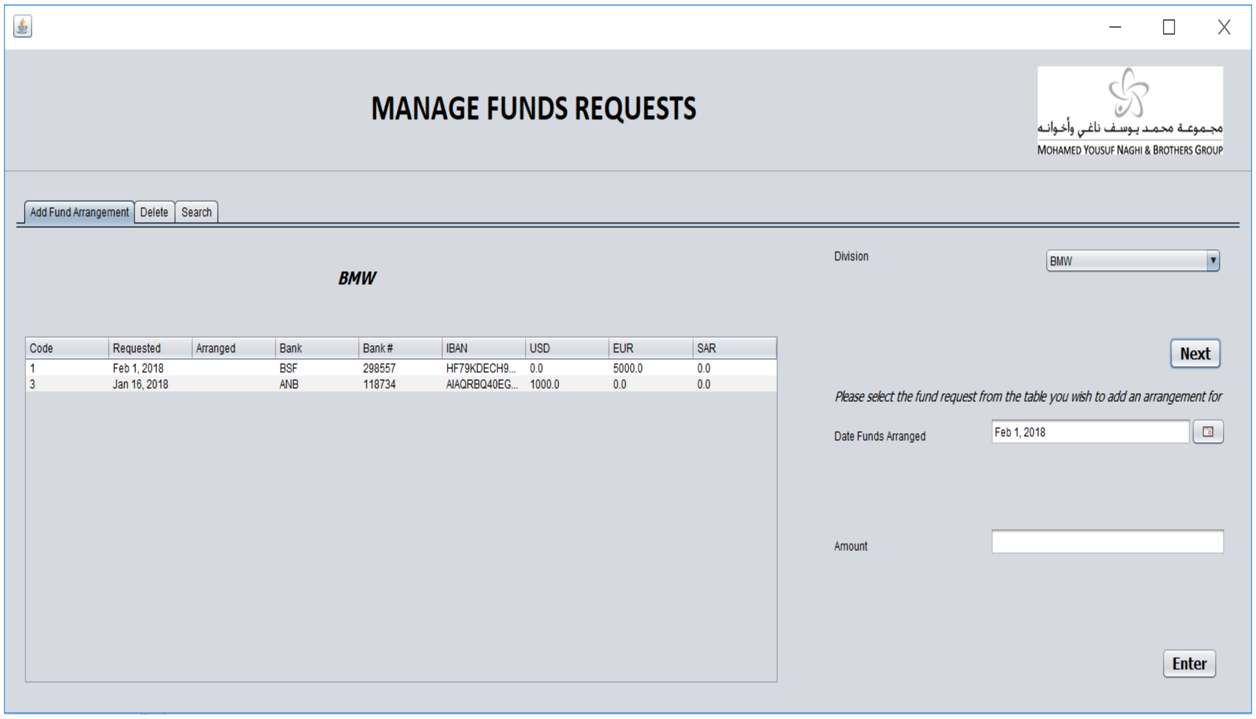
When loading the table, 2 searches need to be made in the database. The first is searching what the index of the division chosen is in the divisions table, and the next query is searching for all the fund requests from the database for that particular division. The following code (Figure 12) shows how this is done.



```
try
{
    con = DriverManager.getConnection(url, user, password);
    st = con.createStatement();
    //search for the division code
    String query = "SELECT divisionCode FROM divisions WHERE divisionName=" + "'" + jCBEditDiv.getSelectedItem().toString() + "'";
    rs = st.executeQuery(query);
    if(rs.next())
    {
        temp = temp + rs.getInt(1);
        try
        {
            conn = DriverManager.getConnection(url, user, password);
            state = conn.createStatement();
            //using the division code to then search its relevant fund requests for which payments are still pending
            String query2 = "SELECT fundsdetails.fundCode, fundsdetails.dateRequested, t1.bankName, fundsdetails.bankNum, fundsdetails.iban, fundsdetails.USDchange, "
                    + "fundsdetails.EURchange, fundsdetails.SARchange FROM fundsdetails LEFT OUTER JOIN (SELECT bankCode, bankName FROM banks)t1 ON fundsdetails.bankCode="
                    + "t1.bankCode WHERE complete=0 AND divcode=" + temp;
            res = state.executeQuery(query2);
            while(res.next())
            {
                dE.addRow(new Object[]{res.getInt(1), res.getString(2), res.getString(3), res.getInt(4), res.getString(5), res.getDouble(6), res.getDouble(7), res.getDouble(8)});
            }
        }
        catch(Exception ex)
        {
            JOptionPane.showMessageDialog(null, ex);
        }
```

Figure 12: Method used to search the relevant data to fill the table

Panel 4 (Add Funds Arrangement)


MANAGE FUNDS REQUESTS
مجموعة محمد يوسف ناغي وأخوانه
MOHAMED YOUSUF NAGHI & BROTHERS GROUP
Add Fund Arrangement
Delete
Search
Next
Enter

BMW

Code	Requested	Arranged	Bank	Bank #	IBAN	USD	EUR	SAR
1	Feb 1, 2018		BSF	298557	HF79KDECH9...	0.0	5000.0	0.0
3	Jan 16, 2018		ANB	118734	AIAQRBQ40EG...	1000.0	0.0	0.0

Division: BMW

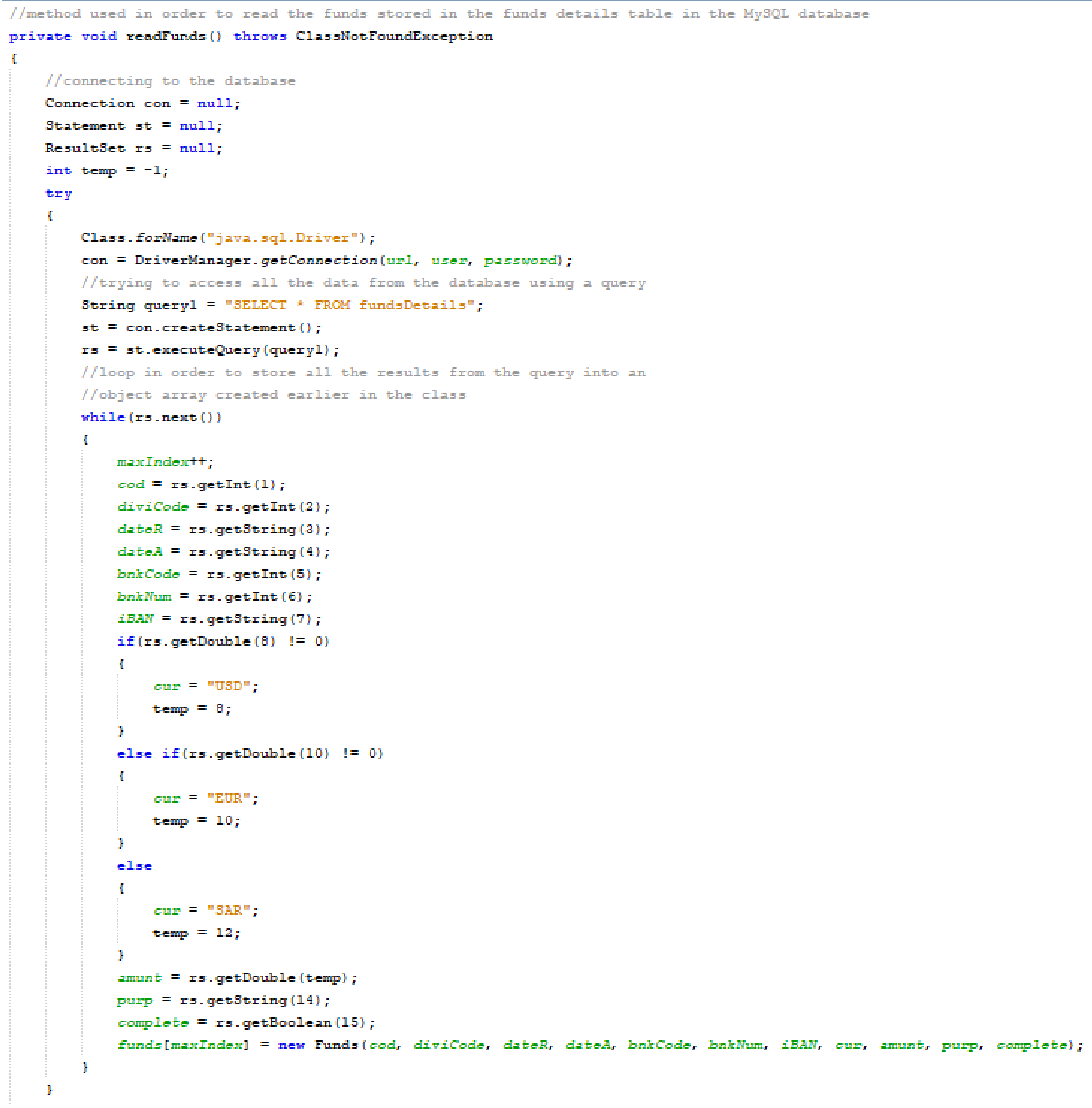
Please select the fund request from the table you wish to add an arrangement for

Date Funds Arranged: Feb 1, 2018

Amount:

Figure 13: Panel used to enter a fund arrangement

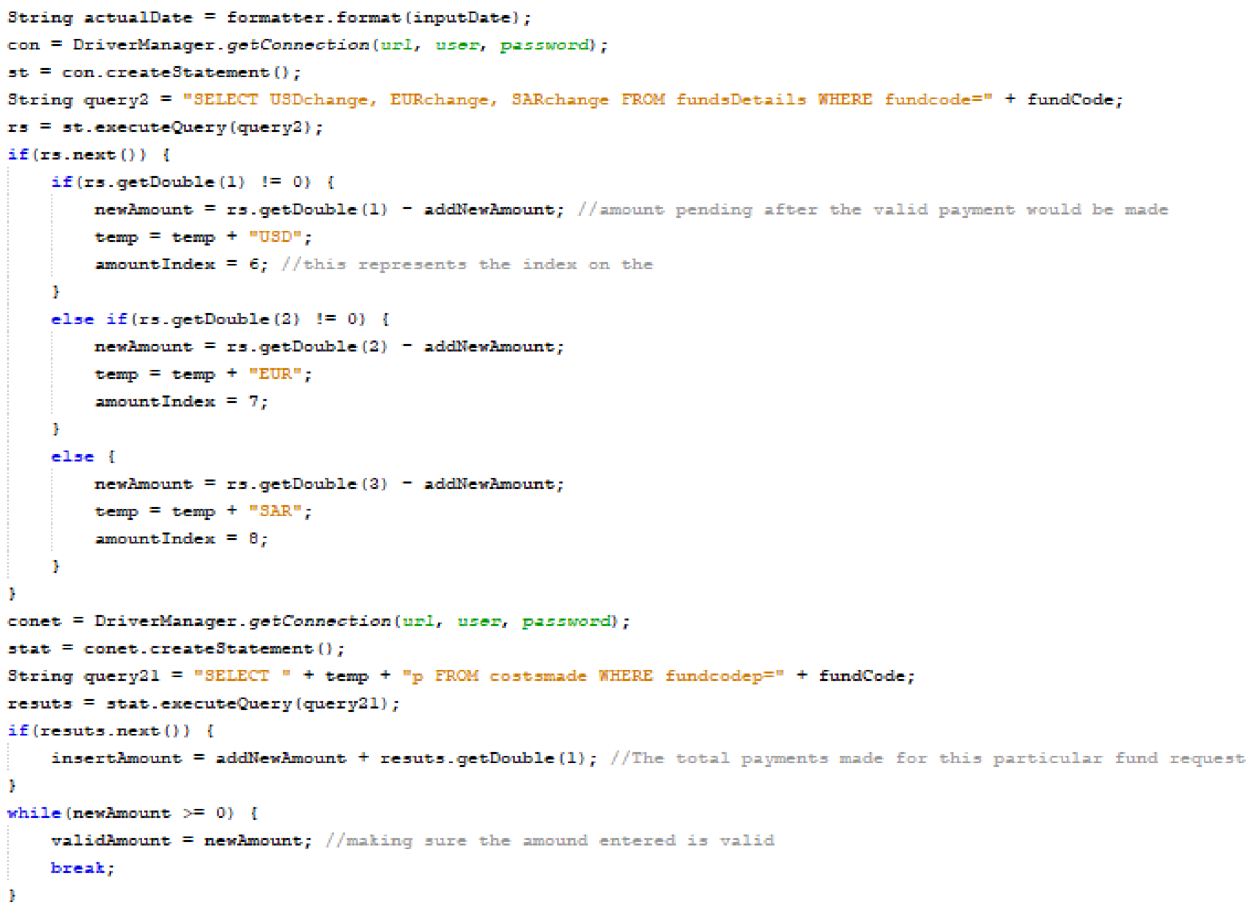
In this panel, I decided I would make an object array of all the fund requests entered by the user in order to make the coding a lot simpler. The code I used to create the object is shown below (Figure 14).



```
//method used in order to read the funds stored in the funds details table in the MySQL database
private void readFunds() throws ClassNotFoundException
{
    //connecting to the database
    Connection con = null;
    Statement st = null;
    ResultSet rs = null;
    int temp = -1;
    try
    {
        Class.forName("java.sql.Driver");
        con = DriverManager.getConnection(url, user, password);
        //trying to access all the data from the database using a query
        String query1 = "SELECT * FROM fundsDetails";
        st = con.createStatement();
        rs = st.executeQuery(query1);
        //loop in order to store all the results from the query into an
        //object array created earlier in the class
        while(rs.next())
        {
            maxIndex++;
            cod = rs.getInt(1);
            diviCode = rs.getInt(2);
            dateR = rs.getString(3);
            dateA = rs.getString(4);
            bnkCode = rs.getInt(5);
            bnkNum = rs.getInt(6);
            iBAN = rs.getString(7);
            if(rs.getDouble(8) != 0)
            {
                cur = "USD";
                temp = 8;
            }
            else if(rs.getDouble(10) != 0)
            {
                cur = "EUR";
                temp = 10;
            }
            else
            {
                cur = "SAR";
                temp = 12;
            }
            amunt = rs.getDouble(temp);
            purp = rs.getString(14);
            complete = rs.getBoolean(15);
            funds[maxIndex] = new Funds(cod, diviCode, dateR, dateA, bnkCode, bnkNum, iBAN, cur, amunt, purp, complete);
        }
    }
```

Figure 14: Method to create a Funds object

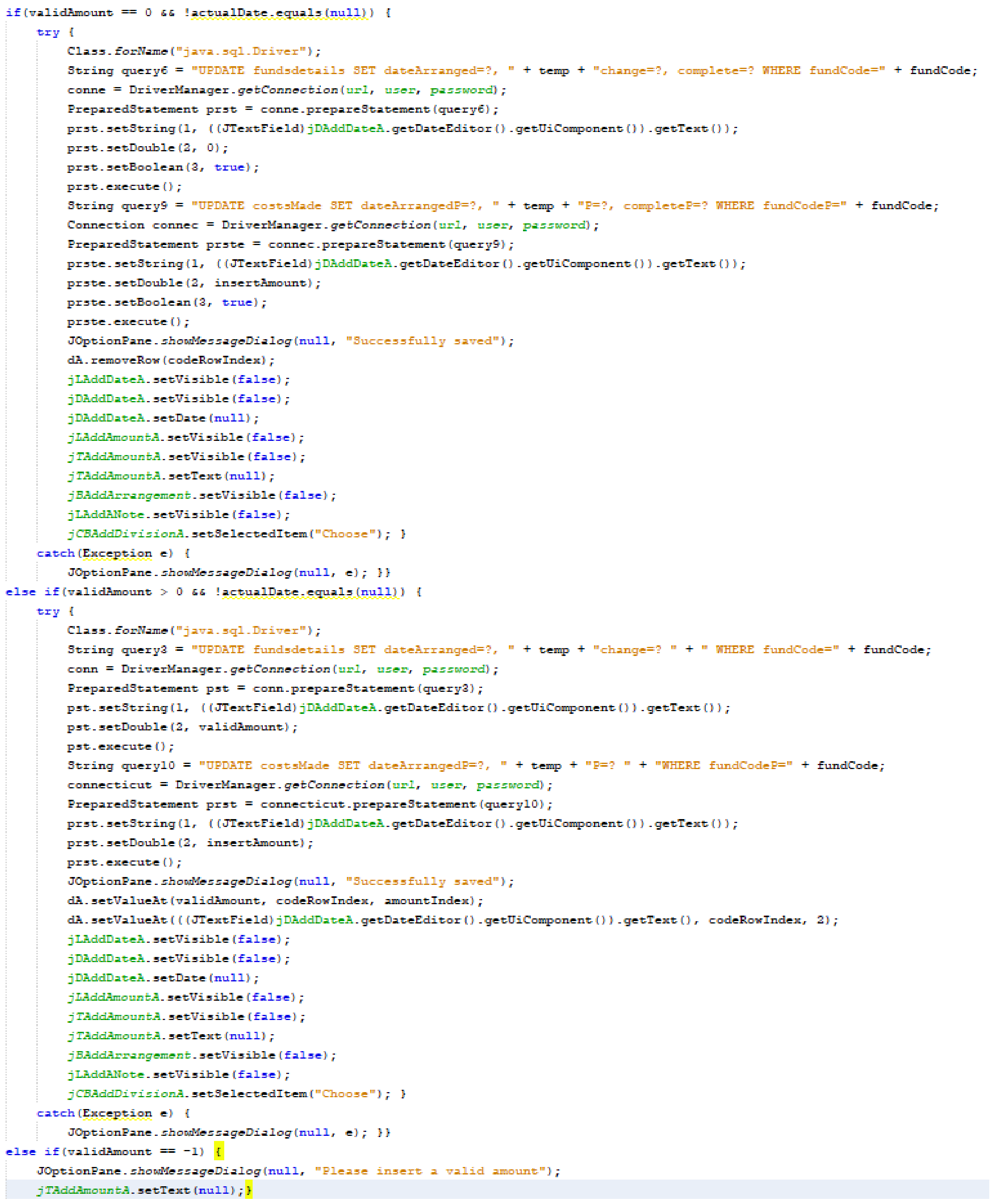
When entering an amount in Figure 13, there need to be certain checks which make sure that the data entered is not invalid.



```
String actualDate = formatter.format(inputDate);
con = DriverManager.getConnection(url, user, password);
st = con.createStatement();
String query2 = "SELECT USDchange, EURchange, SARchange FROM fundsDetails WHERE fundcode=" + fundCode;
rs = st.executeQuery(query2);
if(rs.next()) {
    if(rs.getDouble(1) != 0) {
        newAmount = rs.getDouble(1) - addNewAmount; //amount pending after the valid payment would be made
        temp = temp + "USD";
        amountIndex = 6; //this represents the index on the
    }
    else if(rs.getDouble(2) != 0) {
        newAmount = rs.getDouble(2) - addNewAmount;
        temp = temp + "EUR";
        amountIndex = 7;
    }
    else {
        newAmount = rs.getDouble(3) - addNewAmount;
        temp = temp + "SAR";
        amountIndex = 8;
    }
}
conet = DriverManager.getConnection(url, user, password);
stat = conet.createStatement();
String query21 = "SELECT " + temp + "p FROM costsmade WHERE fundcodep=" + fundCode;
resuts = stat.executeQuery(query21);
if(resuts.next()) {
    insertAmount = addNewAmount + resuts.getDouble(1); //The total payments made for this particular fund request
}
while(newAmount >= 0) {
    validAmount = newAmount; //making sure the amound entered is valid
    break;
}
```

Figure 15: The code which checks whether the amount entered to pay for a fund request is valid and can be saved in the database

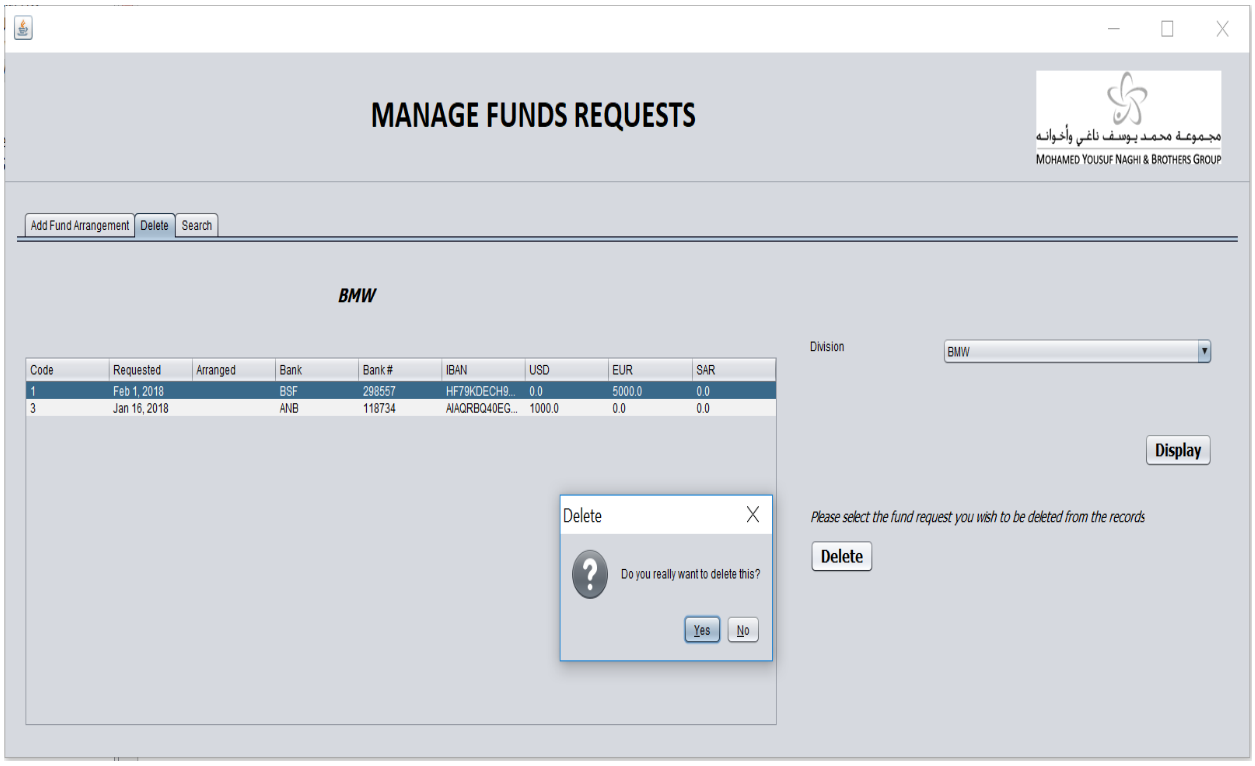
I wrote a series of if-statements to make sure that there was no chance of any invalid data getting through to the database (Figure 16), otherwise, that would mean that the program would not have much use as the same thing would have been happening with the previous system used at the business.



```
if(validAmount == 0 && !actualDate.equals(null)) {
    try {
        Class.forName("java.sql.Driver");
        String query6 = "UPDATE fundsdetails SET dateArranged=?, " + temp + "change=?, complete=? WHERE fundCode=" + fundCode;
        conne = DriverManager.getConnection(url, user, password);
        PreparedStatement prst = conne.prepareStatement(query6);
        prst.setString(1, ((JTextField)jDAddDateA.getDateEditor().getUiComponent()).getText());
        prst.setDouble(2, 0);
        prst.setBoolean(3, true);
        prst.execute();
        String query9 = "UPDATE costsMade SET dateArrangedP=?, " + temp + "P=?, completeP=? WHERE fundCodeP=" + fundCode;
        Connection connec = DriverManager.getConnection(url, user, password);
        PreparedStatement prste = connec.prepareStatement(query9);
        prste.setString(1, ((JTextField)jDAddDateA.getDateEditor().getUiComponent()).getText());
        prste.setDouble(2, insertAmount);
        prste.setBoolean(3, true);
        prste.execute();
        JOptionPane.showMessageDialog(null, "Successfully saved");
        dA.removeRow(codeRowIndex);
        jLAddDateA.setVisible(false);
        jDAddDateA.setVisible(false);
        jDAddDateA.setDate(null);
        jLAddAmountA.setVisible(false);
        jTAddAmountA.setVisible(false);
        jTAddAmountA.setText(null);
        jBAddArrangement.setVisible(false);
        jLAddANote.setVisible(false);
        jCBAddDivisionA.setSelectedItem("Choose"); }
    catch(Exception e) {
        JOptionPane.showMessageDialog(null, e); }}
else if(validAmount > 0 && !actualDate.equals(null)) {
    try {
        Class.forName("java.sql.Driver");
        String query3 = "UPDATE fundsdetails SET dateArranged=?, " + temp + "change=? " + " WHERE fundCode=" + fundCode;
        conn = DriverManager.getConnection(url, user, password);
        PreparedStatement pst = conn.prepareStatement(query3);
        pst.setString(1, ((JTextField)jDAddDateA.getDateEditor().getUiComponent()).getText());
        pst.setDouble(2, validAmount);
        pst.execute();
        String query10 = "UPDATE costsMade SET dateArrangedP=?, " + temp + "P=? " + "WHERE fundCodeP=" + fundCode;
        connecticut = DriverManager.getConnection(url, user, password);
        PreparedStatement prst = connecticut.prepareStatement(query10);
        prst.setString(1, ((JTextField)jDAddDateA.getDateEditor().getUiComponent()).getText());
        prst.setDouble(2, insertAmount);
        prst.execute();
        JOptionPane.showMessageDialog(null, "Successfully saved");
        dA.setValueAt(validAmount, codeRowIndex, amountIndex);
        dA.setValueAt(((JTextField)jDAddDateA.getDateEditor().getUiComponent()).getText(), codeRowIndex, 2);
        jLAddDateA.setVisible(false);
        jDAddDateA.setVisible(false);
        jDAddDateA.setDate(null);
        jLAddAmountA.setVisible(false);
        jTAddAmountA.setVisible(false);
        jTAddAmountA.setText(null);
        jBAddArrangement.setVisible(false);
        jLAddANote.setVisible(false);
        jCBAddDivisionA.setSelectedItem("Choose"); }
    catch(Exception e) {
        JOptionPane.showMessageDialog(null, e); }}
else if(validAmount == -1) {
    JOptionPane.showMessageDialog(null, "Please insert a valid amount");
    jTAddAmountA.setText(null);}
```

Figure 16: Making sure the correct data is sent to the database and it works for repeated errors

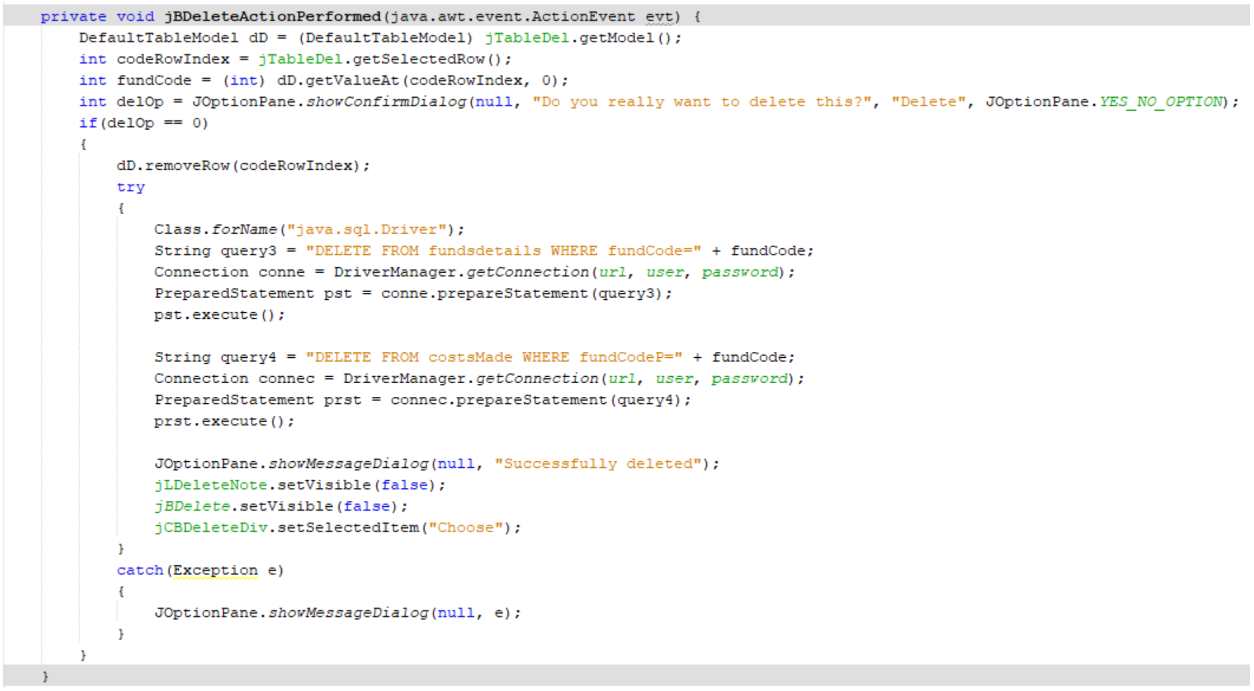
Panel 5 (Delete Fund Request)


MANAGE FUNDS REQUESTS
مجموعة محمد يوسف ناغي وأخوانه
MOHAMED YOUSUF NAGHI & BROTHERS GROUP
Add Fund Arrangement
Delete
Search
BMW
Division
BMW
Display
Please select the fund request you wish to be deleted from the records
Delete
Delete
Do you really want to delete this?
Yes
No

Code	Requested	Arranged	Bank	Bank #	IBAN	USD	EUR	SAR
1	Feb 1, 2018		BSF	298557	HF79KDECH9...	0.0	5000.0	0.0
3	Jan 16, 2018		ANB	118734	AIAQRBQ40EG...	1000.0	0.0	0.0

Figure 17: The delete panel of the manage funds module of the program.

When deleting a fund request, the program displays a confirmation box asking for confirmation on whether the user wishes to delete that particular highlighted fund request. When the user selects yes, the fund requests deletes from the table, and also, it is deleted in the database as well. The following (Figure 18) is the code for the 'Delete' button (Figure 17).



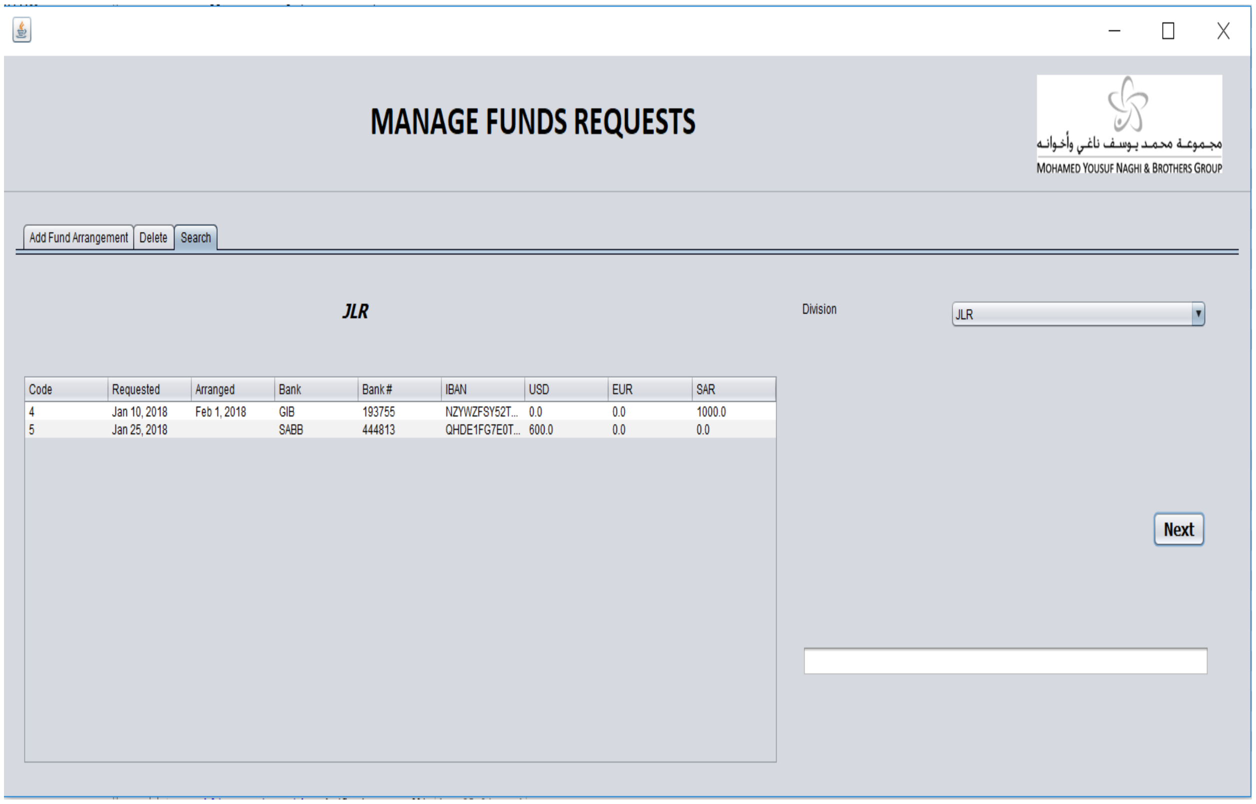
```
private void jBDeleteActionPerformed(java.awt.event.ActionEvent evt) {
    DefaultTableModel dD = (DefaultTableModel) jTableDel.getModel();
    int codeRowIndex = jTableDel.getSelectedRow();
    int fundCode = (int) dD.getValueAt(codeRowIndex, 0);
    int delOp = JOptionPane.showConfirmDialog(null, "Do you really want to delete this?", "Delete", JOptionPane.YES_NO_OPTION);
    if(delOp == 0)
    {
        dD.removeRow(codeRowIndex);
        try
        {
            Class.forName("java.sql.Driver");
            String query3 = "DELETE FROM fundsdetails WHERE fundCode=" + fundCode;
            Connection conne = DriverManager.getConnection(url, user, password);
            PreparedStatement pst = conne.prepareStatement(query3);
            pst.execute();

            String query4 = "DELETE FROM costsMade WHERE fundCodeP=" + fundCode;
            Connection connec = DriverManager.getConnection(url, user, password);
            PreparedStatement prst = connec.prepareStatement(query4);
            prst.execute();

            JOptionPane.showMessageDialog(null, "Successfully deleted");
            jLDeleteNote.setVisible(false);
            jBDelete.setVisible(false);
            jCBDeleteDiv.setSelectedItem("Choose");
        }
        catch(Exception e)
        {
            JOptionPane.showMessageDialog(null, e);
        }
    }
}
```

Figure 18: The Delete Fund Request code

Panel 6 (Search Fund Request)



Code	Requested	Arranged	Bank	Bank#	IBAN	USD	EUR	SAR
4	Jan 10, 2018	Feb 1, 2018	GIB	193755	NZYWZFSY52T...	0.0	0.0	1000.0
5	Jan 25, 2018		SABB	444813	QHDE1FG7E0T...	600.0	0.0	0.0

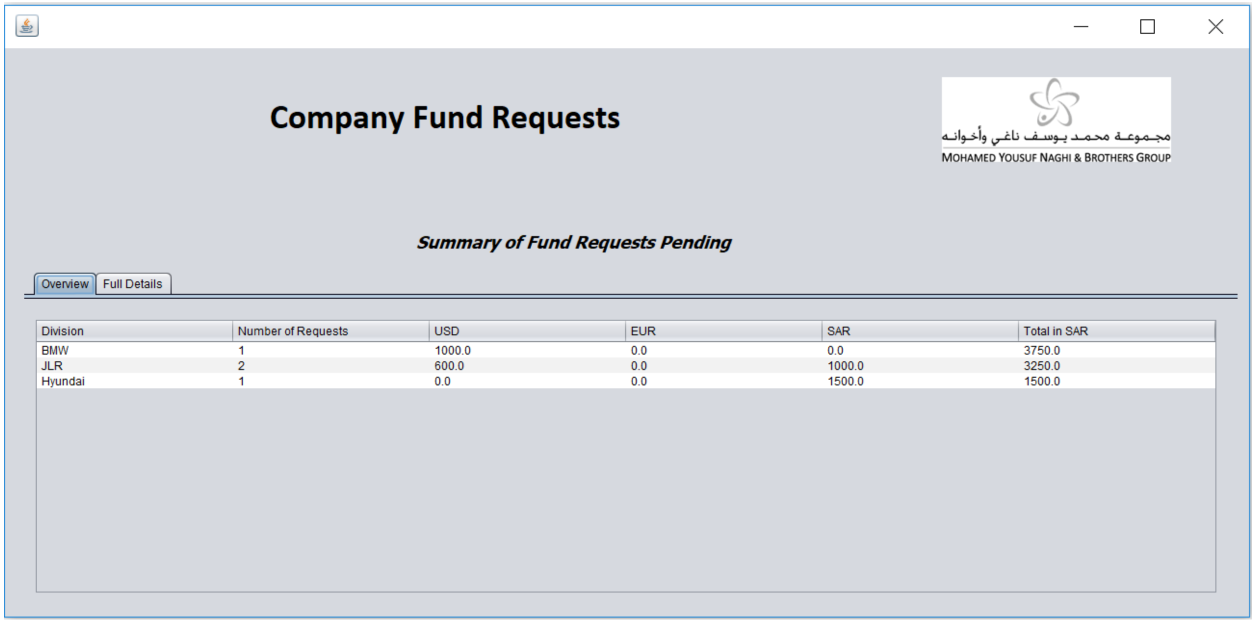
Figure 19: The Search Panel. This panel is used to search for a fund request. The user wishes for this feature as he would want to note down certain details about the fund request he is searching for

In this section of the program, there is no second button, this is because while typing, the program is short-listing the possible records that would have the search data. Below is the code (Figure 20) for the events that would happen when typing into the text field.

```
//search method using a key release event
private void jTSearchNonDateKeyReleased(java.awt.event.KeyEvent evt) {
    DefaultTableModel dS = (DefaultTableModel) jTableSearch.getModel(); //create the default table model
    String search = jTSearchNonDate.getText().toUpperCase();
    TableRowSorter<DefaultTableModel> tS = new TableRowSorter<DefaultTableModel> (dS);
    jTableSearch.setRowSorter(tS);
    tS.setRowFilter(RowFilter.regexFilter(search));
}
```

Figure 20: Search method

Panel 7 (Overview of Pending Requests)


Company Fund Requests
مجموعة محمد يوسف ناغي وأخوانه
MOHAMED YOUSUF NAGHI & BROTHERS GROUP
Overview
Full Details

Summary of Fund Requests Pending

Division	Number of Requests	USD	EUR	SAR	Total in SAR
BMW	1	1000.0	0.0	0.0	3750.0
JLR	2	600.0	0.0	1000.0	3250.0
Hyundai	1	0.0	0.0	1500.0	1500.0

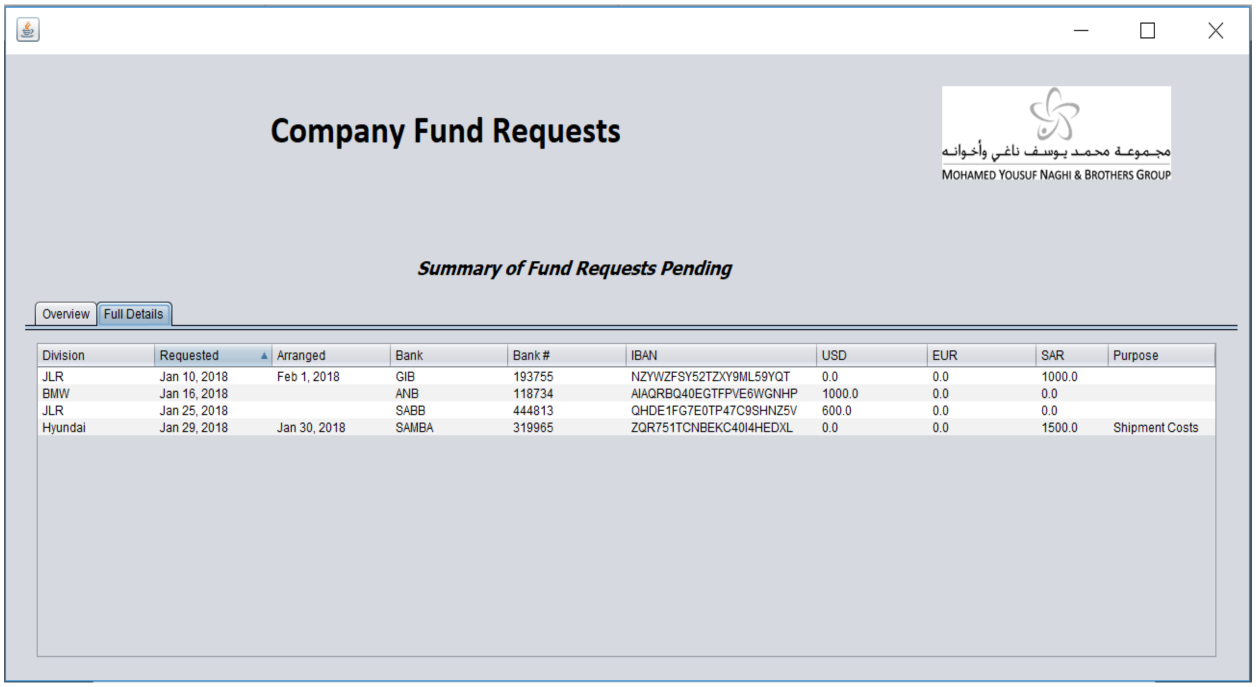
Figure 21: An overview of all the pending funds requests. The client needs this for the sake of analysis. They may want to examine how much the business needs to pay a certain division, and this is made very simple as the last column shows the value in Saudi Arabian Riyals

The following is the algorithm to store this table

```java
DefaultTableModel dGen = (DefaultTableModel) jTableReport1.getModel();
DefaultTableModel dTot = (DefaultTableModel) jTableReport1Total.getModel();
try
{
    con = DriverManager.getConnection(url, user, password);
    st = con.createStatement();
    String query = "SELECT divisions.divisionName, t1.numRequests, t1.tot_us, t1.tot_eu, t1.tot_sar FROM divisions LEFT OUTER JOIN (SELECT divCode, COUNT(*) AS numRequests, "
            + "SUM(USDchange) AS tot_us, SUM(EURchange) AS tot_eu, SUM(SARchange) AS tot_sar FROM fundsdetails WHERE complete=0 GROUP BY divCode)t1 ON divisions.divisionCode="
            + "t1.divCode";
    rs = st.executeQuery(query);
    while(rs.next())
    {
        double total = (rs.getDouble(3)*3.75) + (rs.getDouble(4)*4.35) + rs.getDouble(5); //get the total amount in SAR using the currency conversions researched online
        dGen.addRow(new Object[]{rs.getString(1), rs.getInt(2), rs.getDouble(3), rs.getDouble(4), rs.getDouble(5), df.format(total)});
    }
```

Figure 21: Store table for overview of funds pending

Panel 8 (Detailed Report on Pending Requests)



Company Fund Requests

مجموعة محمد يوسف ناغي وأخوانه
MOHAMED YOUSUF NAGHI & BROTHERS GROUP

Summary of Fund Requests Pending

Division	Requested	Arranged	Bank	Bank #	IBAN	USD	EUR	SAR	Purpose
JLR	Jan 10, 2018	Feb 1, 2018	GIB	193755	NZYWZFSY52TZXY9ML59YQT	0.0	0.0	1000.0	
BMW	Jan 16, 2018		ANB	118734	AIAQRBQ40EGTFPVE6WGNHP	1000.0	0.0	0.0	
JLR	Jan 25, 2018		SABB	444813	QHDE1FG7E0TP47C9SHNZ5V	600.0	0.0	0.0	
Hyundai	Jan 29, 2018	Jan 30, 2018	SAMBA	319965	ZQR751TCNBEKC40I4HEDXL	0.0	0.0	1500.0	Shipment Costs

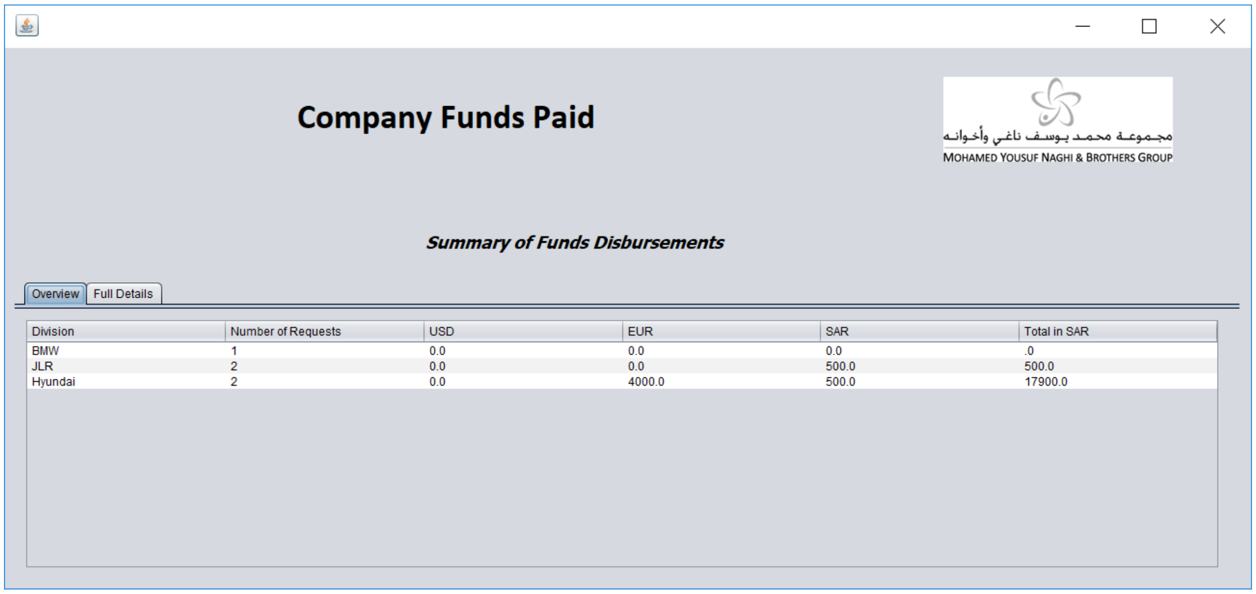
Figure 22: This shows all the details for each fund request for which payments are pending. The user would need this to note down which fund request needs to be paid with most urgency by sorting the table by date the fund was requested, the earliest one would be the most urgent

Setting up a query for this is a lot simpler than the overview section, as I only needed to select all the funds requests and did not require any calculations for this.

```java
conn = DriverManager.getConnection(url, user, password);
sta = conn.createStatement();
String query2 = "SELECT t2.divisionName, fundsdetails.dateRequested, fundsdetails.dateArranged, t1.bankName, fundsdetails.bankNum, fundsdetails.iban, fundsdetails.USDchange, "
        + "fundsdetails.EURchange, fundsdetails.SARchange, fundsdetails.purpose FROM fundsdetails LEFT OUTER JOIN (SELECT bankCode, bankName FROM banks)t1 ON fundsdetails."
        + "bankCode=t1.bankCode LEFT OUTER JOIN (SELECT divisionCode, divisionName FROM divisions)t2 ON fundsdetails.divCode=t2.divisionCode WHERE complete=0";
res = sta.executeQuery(query2);
while(res.next())
{
    dTot.addRow(new Object[]{res.getString(1), res.getString(2), res.getString(3), res.getString(4), res.getInt(5), res.getString(6), res.getDouble(7), res.getDouble(8),
        res.getDouble(9), res.getString(10)});
}
```

Figure 23: Code for storing data onto the detailed report table

Panel 9 (Payments Overview)



Company Funds Paid

مجموعة محمد يوسف ناغي وأخوانه
MOHAMED YOUSUF NAGHI & BROTHERS GROUP

Summary of Funds Disbursements

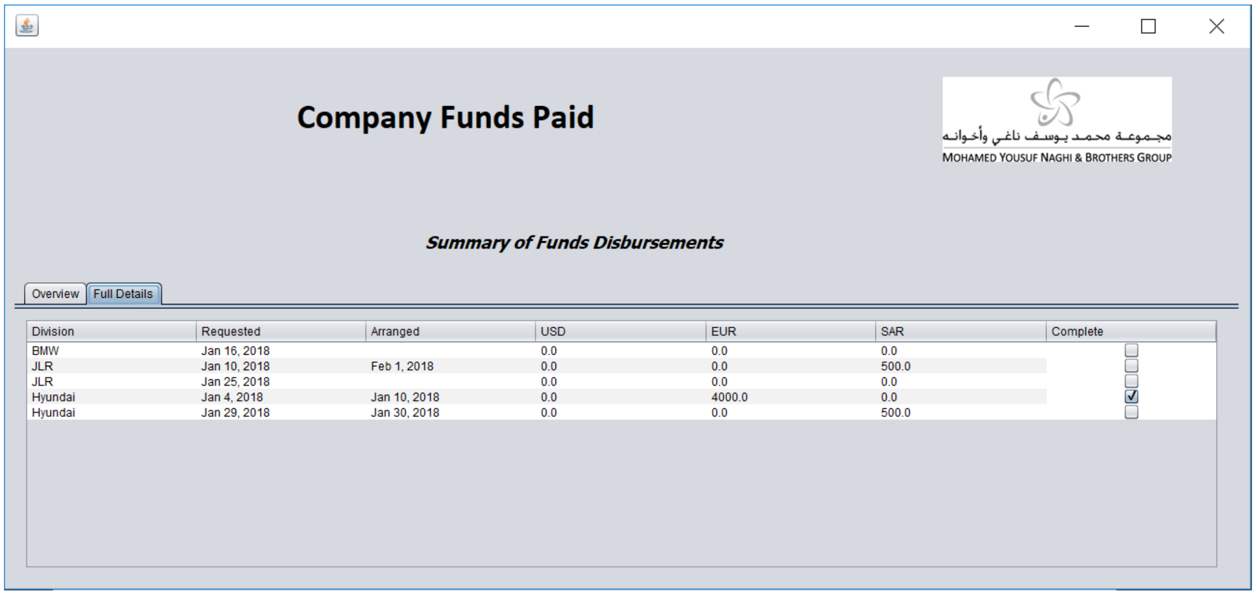
Division	Number of Requests	USD	EUR	SAR	Total in SAR
BMW	1	0.0	0.0	0.0	.0
JLR	2	0.0	0.0	500.0	500.0
Hyundai	2	0.0	4000.0	500.0	17900.0

Figure 24: An overview for all the payments made by the business on each division. This is important as it ensures the business that there is a balance and no amount is missing. This feature makes this program unique as it is not possible to keep track of both pending and payments in a spreadsheet

```java
DefaultTableModel dSum = (DefaultTableModel) jTableReport2.getModel();
DefaultTableModel dTot = (DefaultTableModel) jTableReport2Total.getModel();
try
{
    con = DriverManager.getConnection(url, user, password);
    st = con.createStatement();
    String query = "SELECT divisions.divisionName, t1.numRequests, t1.tot_us, t1.tot_eu, t1.tot_sar FROM divisions LEFT OUTER JOIN (SELECT divCodeP, COUNT(*) AS numRequests, "
            + "SUM(USDP) AS tot_us, SUM(EURP) AS tot_eu, SUM(SARP) AS tot_sar FROM costsMade GROUP BY divCodeP)t1 ON divisions.divisionCode=t1.divCodeP";
    rs = st.executeQuery(query);
    while(rs.next())
    {
        double total = (rs.getDouble(3)*3.75) + (rs.getDouble(4)*4.35) + rs.getDouble(5);
        dSum.addRow(new Object[]{rs.getString(1), rs.getInt(2), rs.getDouble(3), rs.getDouble(4), rs.getDouble(5), dc.format(total)});
    }
```

Figure 25: Store table for overview of all payments grouped by funds

Panel 10 (Detailed Report of Payments)


Company Funds Paid
مجموعة محمد يوسف ناغي وأخوانه
MOHAMED YOUSUF NAGHI & BROTHERS GROUP
Summary of Funds Disbursements
Overview
Full Details

Division	Requested	Arranged	USD	EUR	SAR	Complete
BMW	Jan 16, 2018		0.0	0.0	0.0	☐
JLR	Jan 10, 2018	Feb 1, 2018	0.0	0.0	500.0	☐
JLR	Jan 25, 2018		0.0	0.0	0.0	☐
Hyundai	Jan 4, 2018	Jan 10, 2018	0.0	4000.0	0.0	✓
Hyundai	Jan 29, 2018	Jan 30, 2018	0.0	0.0	500.0	☐

Figure 26: A full detailed report of all the payments made by the business to each division. The table format is different to the pending report as this shows the completed requests as well. A check box is present to tell the user whether the business has completely paid for a certain fund request

```
conn = DriverManager.getConnection(url, user, password);
sta = conn.createStatement();
String query2 = "SELECT t2.divisionName, costsMade.dateRequestedP, costsMade.dateArrangedP, costsMade.USDP, costsMade.EURP, costsMade.SARP, costsMade.completeP FROM costsMade "
        + "LEFT OUTER JOIN (SELECT divisionCode, divisionName FROM divisions)t2 ON costsMade.divCodeP=t2.divisionCode";
res = sta.executeQuery(query2);
while(res.next())
{
    dTot.addRow(new Object[]{res.getString(1), res.getString(2), res.getString(3), res.getDouble(4), res.getDouble(5), res.getDouble(6), res.getBoolean(7)});
}
```

Figure 27: Code for storing data onto the detailed report table for payments made by the business

559 words