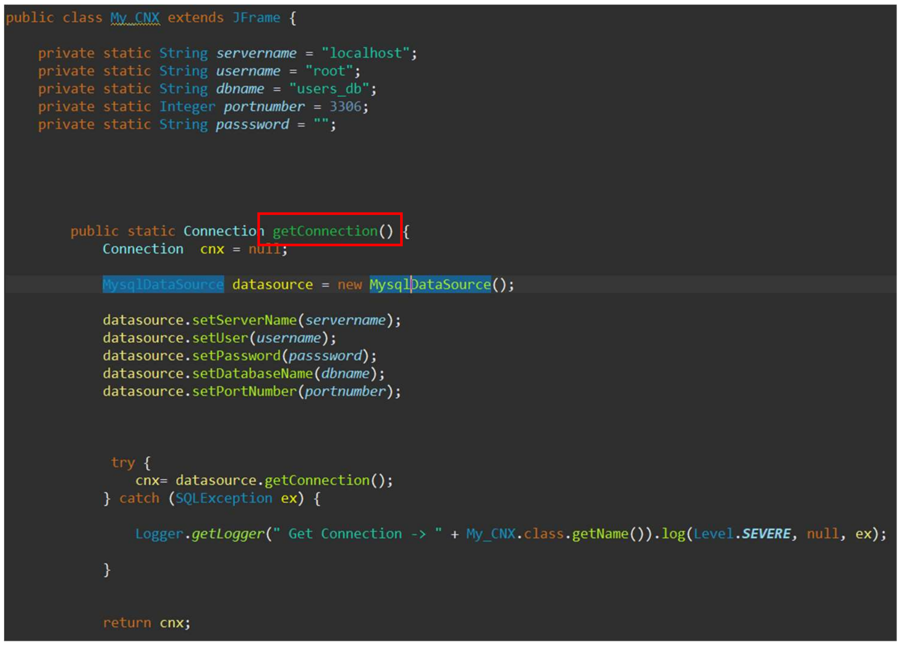
Criterion C - Development

List of Techniques Used

1. Encapsulation
2. MySQL Database Connectivity
3. Printing
4. Conditional Insertion of data into a 2D Array from database table
5. Use of External GUI Libraries (JavaSwing and Java AWT) and Predefined methods
6. Use of Flags and Validation
7. Complex Selection
8. User-Defined methods
9. Event Handling
10. Complex MySQL Queries
11. Exception Handling
12. Multi-level Inheritance
13. Parsing

1. Encapsulation

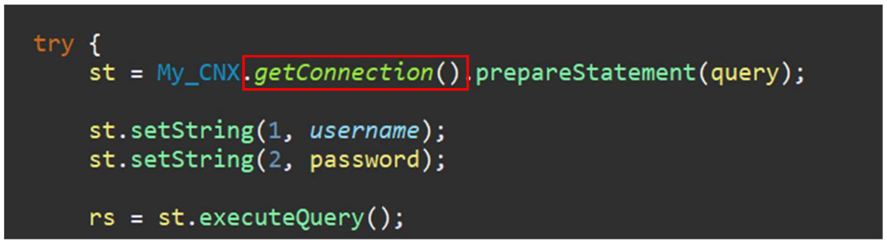
The use of encapsulation allows for the reusability of the code. It also allows the programmer to hide the inner implementation of the class from the user. Visibility of all variables within the class is set to "private" and the visibility of the methods is set to "public", so that they can be accessed by any of the other classes. This is especially helpful because the method "getConnection()" of "*My_CNX"* class is used in multiple classes to establish a connection to the MySQL database to insert queries. This way the code does not have to be re-written every time a query needs to be made to the database. As all of the users and donations are logged in the database, this helps reduce redundancy and repetition of code.



```
public class My_CNX extends JFrame {

    private static String servername = "localhost";
    private static String username = "root";
    private static String dbname = "users_db";
    private static Integer portnumber = 3306;
    private static String passsword = "";



        public static Connection getConnection() {
            Connection  cnx = null;

            MysqlDataSource datasource = new MysqlDataSource();

            datasource.setServerName(servername);
            datasource.setUser(username);
            datasource.setPassword(passsword);
            datasource.setDatabaseName(dbname);
            datasource.setPortNumber(portnumber);


             try {
                cnx= datasource.getConnection();
            } catch (SQLException ex) {

                Logger.getLogger(" Get Connection -> " + My_CNX.class.getName()).log(Level.SEVERE, null, ex);

            }


            return cnx;
```

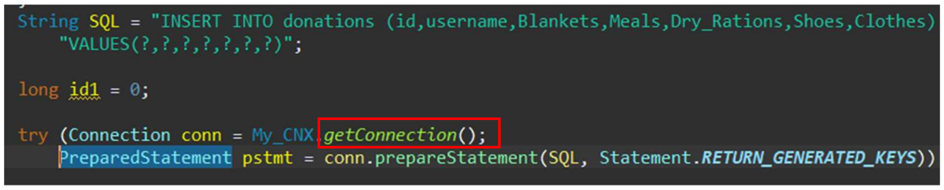
Examples of usage in other classes is shown below:

1. In Login pages (for account sign in and registration)



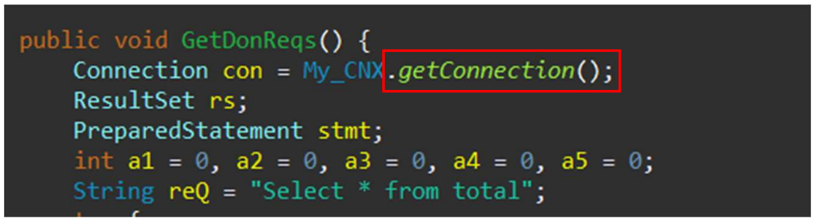
```
try {
    st = My_CNX.getConnection().prepareStatement(query);

    st.setString(1, username);
    st.setString(2, password);

    rs = st.executeQuery();
```

2. In receiver and donor pages (for updating records and fetching records)

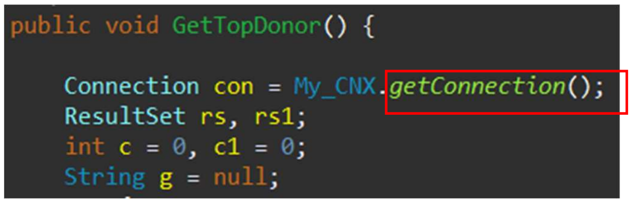


```
String SQL = "INSERT INTO donations (id,username,Blankets,Meals,Dry_Rations,Shoes,Clothes)
    "VALUES(?,?,?,?,?,?,?)";

long id1 = 0;

try (Connection conn = My_CNX.getConnection();
    PreparedStatement pstmt = conn.prepareStatement(SQL, Statement.RETURN_GENERATED_KEYS))
```

3. Top Donor and Donation Requests functionality



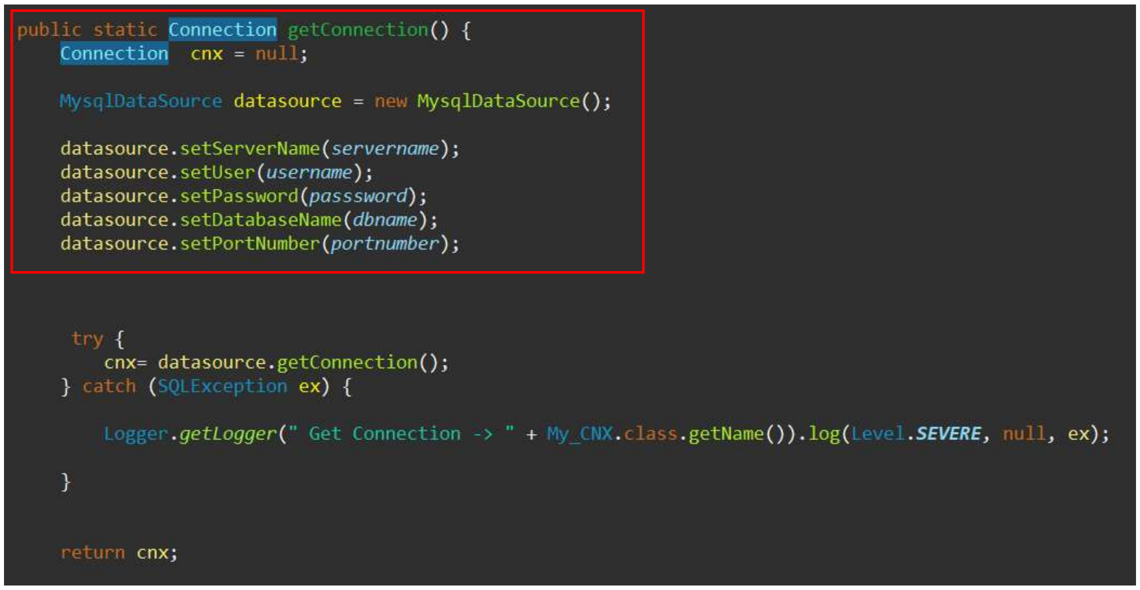
```
public void GetDonReqs() {
    Connection con = My_CNX.getConnection();
    ResultSet rs;
    PreparedStatement stmt;
    int a1 = 0, a2 = 0, a3 = 0, a4 = 0, a5 = 0;
    String reQ = "Select * from total";
```



```
public void GetTopDonor() {

    Connection con = My_CNX.getConnection();
    ResultSet rs, rs1;
    int c = 0, c1 = 0;
    String g = null;
```

2. **MySQL database connectivity**

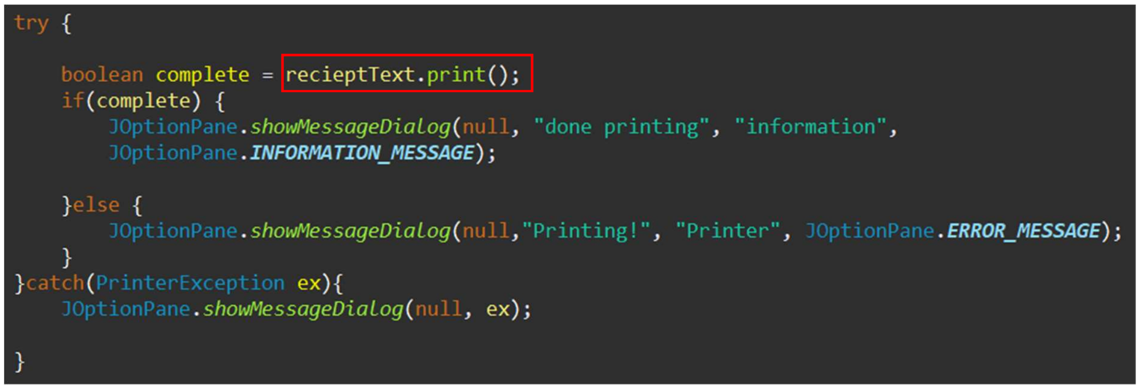
MySQL database and Java connectivity is established through the MySQL JDBC (Java Database connectivity) driver. This makes it possible to communicate with the database server and therefore execute queries on the MySQL database tables from within the java code. Every time the application needs to communicate with the database, a connection needs to be established with the database, the getConnection() method can be evoked to do this. This allows the application to communicate and/or update the database in real time which is crucial in making sure that all records are consistent all the time which is a requirement of the criteria 2e and 3d.



```java
public static Connection getConnection() {
    Connection  cnx = null;

    MysqlDataSource datasource = new MysqlDataSource();

    datasource.setServerName(servername);
    datasource.setUser(username);
    datasource.setPassword(passsword);
    datasource.setDatabaseName(dbname);
    datasource.setPortNumber(portnumber);


     try {
        cnx= datasource.getConnection();
    } catch (SQLException ex) {

        Logger.getLogger(" Get Connection -> " + My_CNX.class.getName()).log(Level.SEVERE, null, ex);

    }


    return cnx;
```

3. Printing

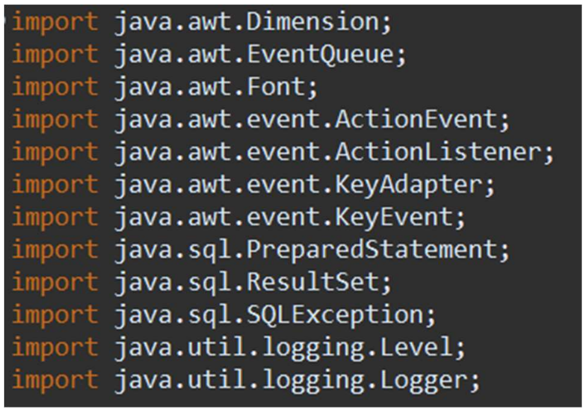
When the user presses the "print" button, the application prints a receipt of the user's carts, which is linked to success criteria 3e. The print procedure is written within a try-catch block to ensure a "graceful failure". Furthermore, appropriate messages are also shown while the printing is in process. These add to the user-friendliness of the application which is a requirement of the Criteria 1a.



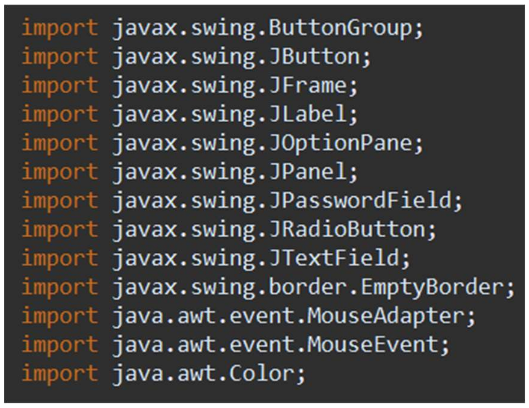
```java
try {

    boolean complete = recieptText.print();
    if(complete) {
        JOptionPane.showMessageDialog(null, "done printing", "information",
        JOptionPane.INFORMATION_MESSAGE);

    }else {
        JOptionPane.showMessageDialog(null,"Printing!", "Printer", JOptionPane.ERROR_MESSAGE);
    }
}catch(PrinterException ex){
    JOptionPane.showMessageDialog(null, ex);

}
```

4. Use of External GUI Libraries (JavaSwing, Java AWT, Java MySQL) and Predefined methods

The following GUI libraries are used to develop a visually appealing, user-friendly and powerful interface. This is linked to success criteria 1a and b as well as Criteria 3e. The javax.swing library is a powerful GUI creation tool which allows the addition of GUI elements e.g. radio buttons, text fields. Next, java.awt enables the incorporation of event handling and a vast library of predefined methods, which is integral to the working of a multiple-panel based GUI. "java.awt" is also used to incorporate colours and different fonts in the application.

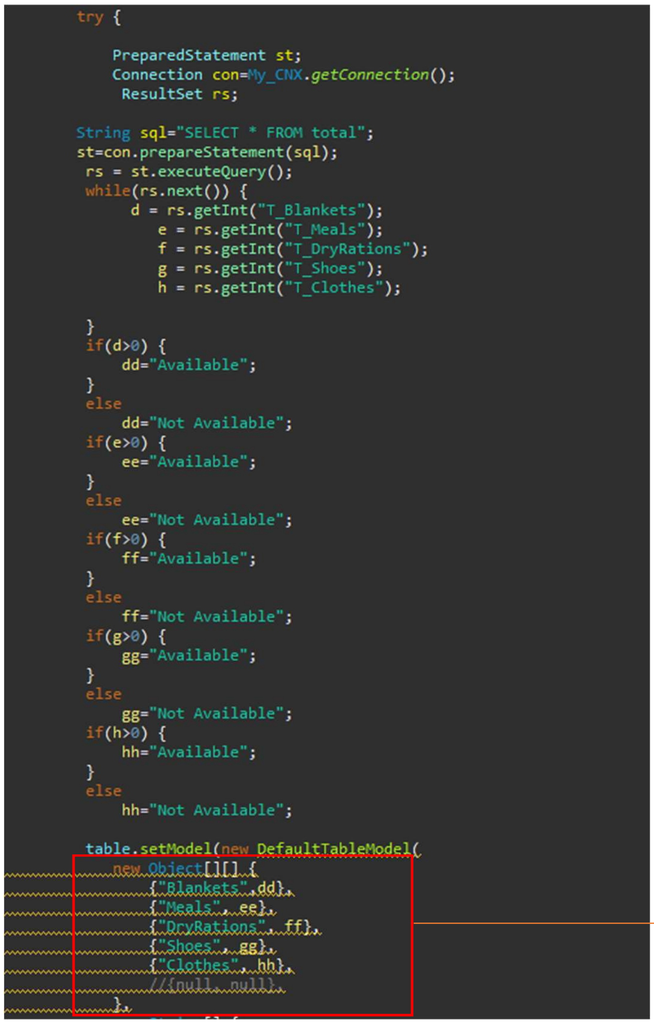


```
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;
```



```
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JRadioButton;
import javax.swing.JTextField;
import javax.swing.border.EmptyBorder;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.Color;
```

5. Conditional Insertion of Data into a 2D Array from Database Table



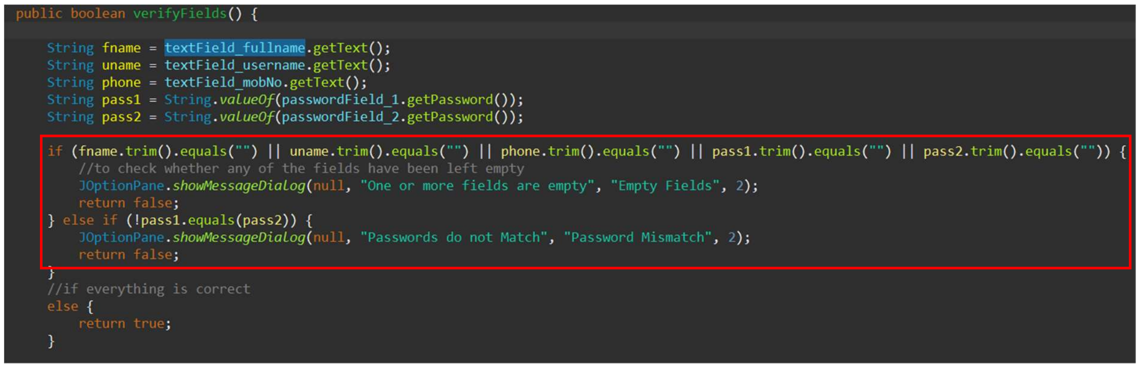
```
try {

    PreparedStatement st;
    Connection con=My_CNX.getConnection();
     ResultSet rs;

String sql="SELECT * FROM total";
st=con.prepareStatement(sql);
 rs = st.executeQuery();
 while(rs.next()) {
      d = rs.getInt("T_Blankets");
         e = rs.getInt("T_Meals");
         f = rs.getInt("T_DryRations");
         g = rs.getInt("T_Shoes");
         h = rs.getInt("T_Clothes");

 }
 if(d>0) {
     dd="Available";
 }
 else
     dd="Not Available";
 if(e>0) {
     ee="Available";
 }
 else
     ee="Not Available";
 if(f>0) {
     ff="Available";
 }
 else
     ff="Not Available";
 if(g>0) {
     gg="Available";
 }
 else
     gg="Not Available";
 if(h>0) {
     hh="Available";
 }
 else
     hh="Not Available";

 table.setModel(new DefaultTableModel(
    new Object[][] {
        {"Blankets",dd},
        {"Meals", ee},
        {"DryRations", ff},
        {"Shoes", gg},
        {"Clothes", hh},
        //{null, null},
    },
```

This method displays the availability of items to the receivers which is linked to success criteria 3a. An SQL statement is used to retrieve column data from the table of total donations. Then, an if-else statement is used to determine whether any of the columns of the table are found to be zero (which would signify that the item is out of stock). If an item is found to be zero the 2D array table model is updated to show that the item is “Not Available”. If the quantity of an Item is found to be more than 0 then the table is updated to show that the item is “available”.

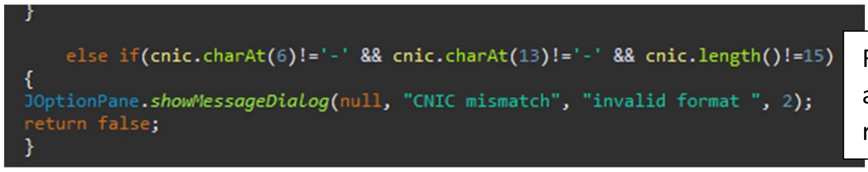
Declaration of 2D array named “Object[][]” and consequently insertion using predefined variables that get values dynamically from the “donations” database table at runtime

6. Use of Flags and Validation

Flags are used to prevent extreme and/or abnormal situations from occurring which is an implicit requirement of Criteria 3f. They are a sort of validation check which ensure that certain conditions have been met. This is to make sure that the GUI works properly and to avoid null fields in the database. In the example of the code below, a Boolean method checks whether the fields have been left empty and whether the “password” and “confirm passwords” have the same text. If the user leaves any of the fields empty or if the passwords do not match, an error message pops up for each case.



```java
public boolean verifyFields() {

    String fname = textField_fullname.getText();
    String uname = textField_username.getText();
    String phone = textField_mobNo.getText();
    String pass1 = String.valueOf(passwordField_1.getPassword());
    String pass2 = String.valueOf(passwordField_2.getPassword());

    if (fname.trim().equals("") || uname.trim().equals("") || phone.trim().equals("") || pass1.trim().equals("") || pass2.trim().equals("")) {
        //to check whether any of the fields have been left empty
        JOptionPane.showMessageDialog(null, "One or more fields are empty", "Empty Fields", 2);
        return false;
    } else if (!pass1.equals(pass2)) {
        JOptionPane.showMessageDialog(null, "Passwords do not Match", "Password Mismatch", 2);
        return false;
    }
    //if everything is correct
    else {
        return true;
    }
```

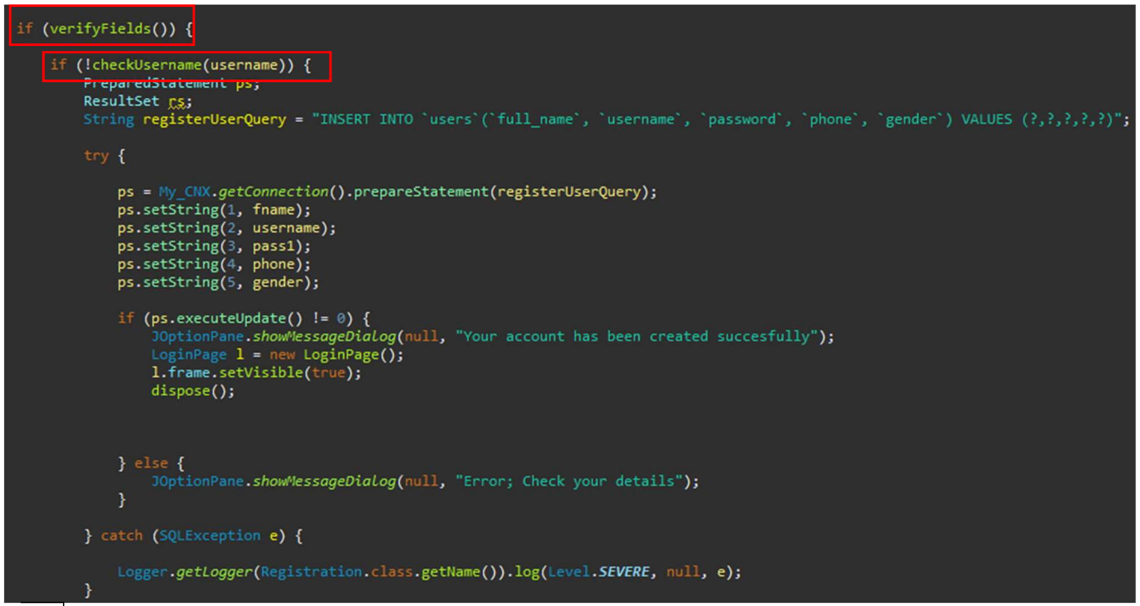


```java
}

    else if(cnic.charAt(6)!='-' && cnic.charAt(13)!='-' && cnic.length()!=15)
{
JOptionPane.showMessageDialog(null, "CNIC mismatch", "invalid format ", 2);
return false;
}
```

Fulfils criteria 3a. CNIC in a correct format can be registered only

7. Complex Selection

Complex selection (nested if-else statements) has been used throughout the application especially for validation as well as in code blocks where certain conditions need to be met in order to ensure smooth working of the GUI. If-else statements also allow for decision making such that there are different actions performed based on the input parameters



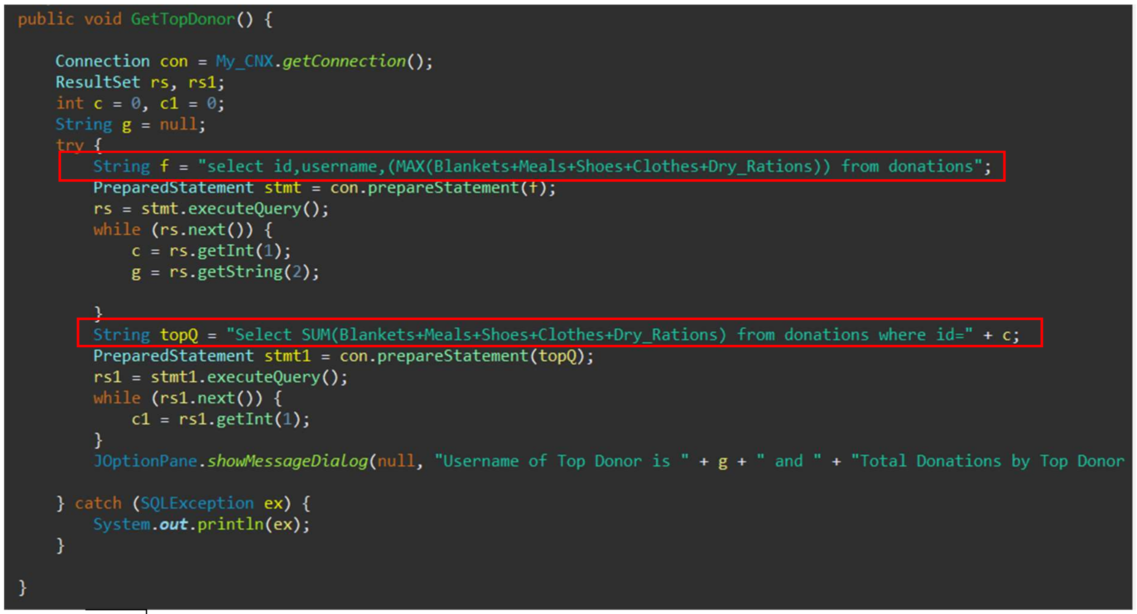
```java
if (verifyFields()) {

    if (!checkUsername(username)) {
        PreparedStatement ps;
        ResultSet rs;
        String registerUserQuery = "INSERT INTO `users`(`full_name`, `username`, `password`, `phone`, `gender`) VALUES (?,?,?,?,?)";

        try {

            ps = My_CNX.getConnection().prepareStatement(registerUserQuery);
            ps.setString(1, fname);
            ps.setString(2, username);
            ps.setString(3, pass1);
            ps.setString(4, phone);
            ps.setString(5, gender);

            if (ps.executeUpdate() != 0) {
                JOptionPane.showMessageDialog(null, "Your account has been created succesfully");
                LoginPage l = new LoginPage();
                l.frame.setVisible(true);
                dispose();


            } else {
                JOptionPane.showMessageDialog(null, "Error; Check your details");
            }

        } catch (SQLException e) {

            Logger.getLogger(Registration.class.getName()).log(Level.SEVERE, null, e);
        }
```

This code first verifies whether the verifyfields() boolean method returns true. If so then checks whether the username that the person has chosen does not already exist in the database. If it does the user is told to pick a new name. If it doesn’t the new username is registered in a new account

8. User-Defined methods

Used defined methods were used in parts of the code where the code was hard to understand. So, to improve readability most of the complex algorithms were defined as functions and called whenever they were required. This was required as large parts of the code would be hard to understand within ActionListeners. The more complex and larger parts of code were defined separately as functions and called into the ActionListener and where ever they were required. These help us in the overall development of the program

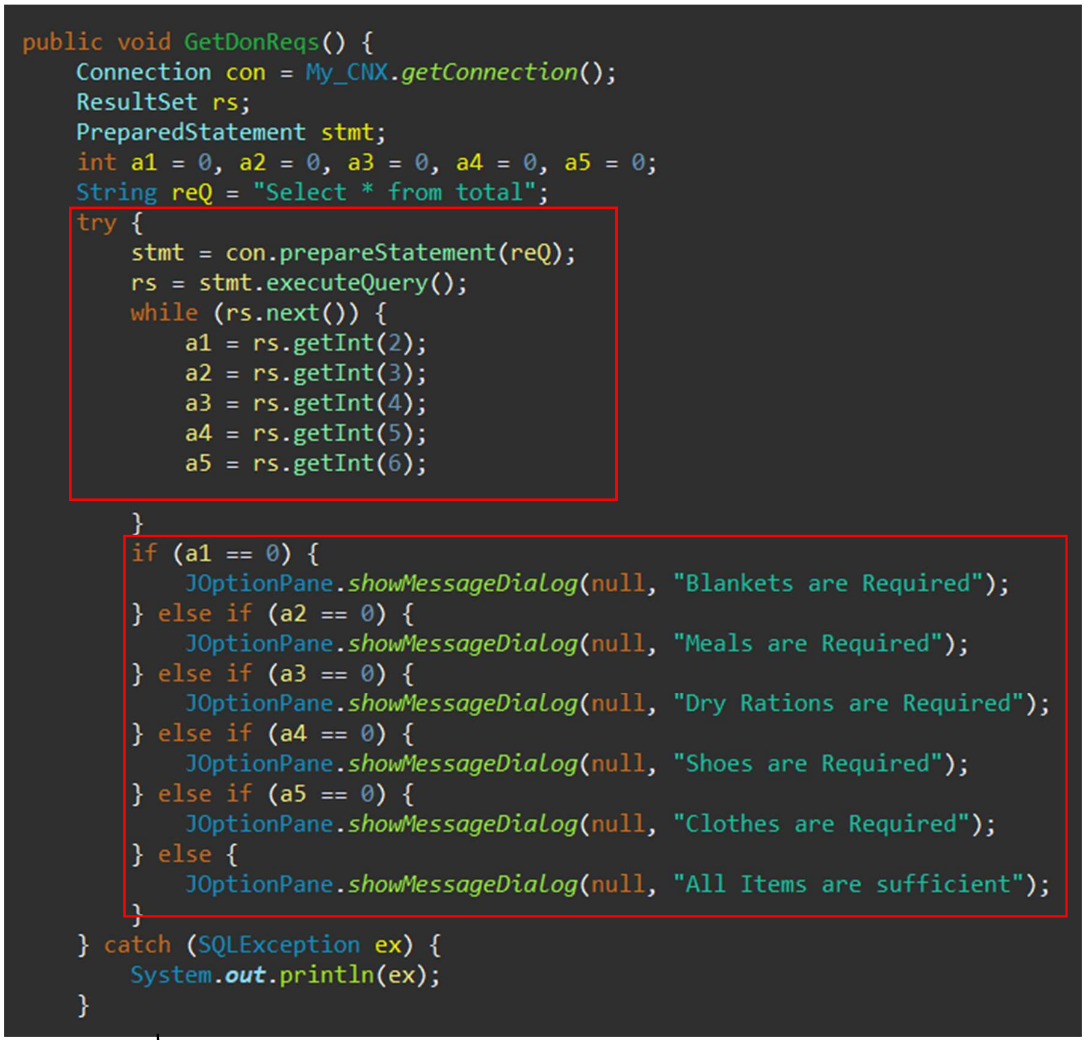
Examples of User defined methods:



```
public void GetTopDonor() {

    Connection con = My_CNX.getConnection();
    ResultSet rs, rs1;
    int c = 0, c1 = 0;
    String g = null;
    try {
        String f = "select id,username,(MAX(Blankets+Meals+Shoes+Clothes+Dry_Rations)) from donations";
        PreparedStatement stmt = con.prepareStatement(f);
        rs = stmt.executeQuery();
        while (rs.next()) {
            c = rs.getInt(1);
            g = rs.getString(2);

        }
        String topQ = "Select SUM(Blankets+Meals+Shoes+Clothes+Dry_Rations) from donations where id=" + c;
        PreparedStatement stmt1 = con.prepareStatement(topQ);
        rs1 = stmt1.executeQuery();
        while (rs1.next()) {
            c1 = rs1.getInt(1);
        }
        JOptionPane.showMessageDialog(null, "Username of Top Donor is " + g + " and " + "Total Donations by Top Donor

    } catch (SQLException ex) {
        System.out.println(ex);
    }

}
```

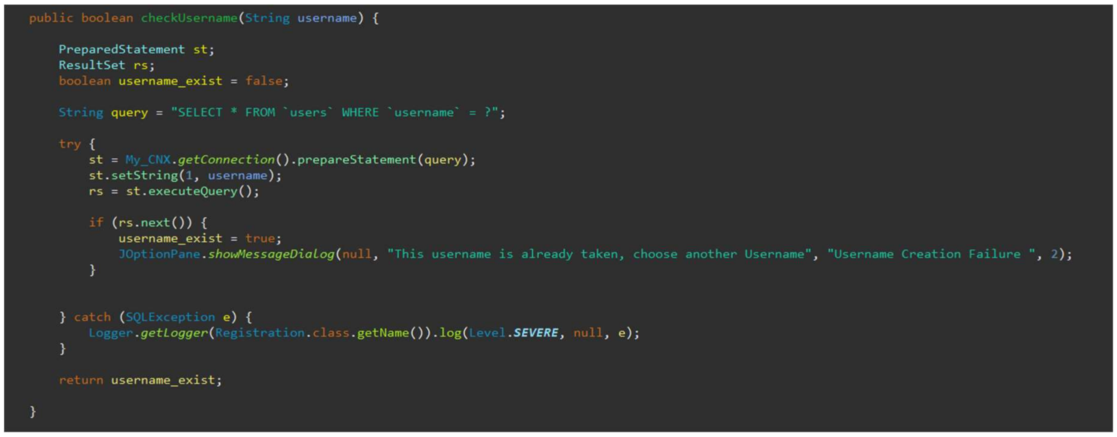
This first query “f” finds the user with the user with the most amount of donations.

The second query “topQ” finds the number of donations that this top donor (that was found in query f) has and displays this information in a JOptionPane



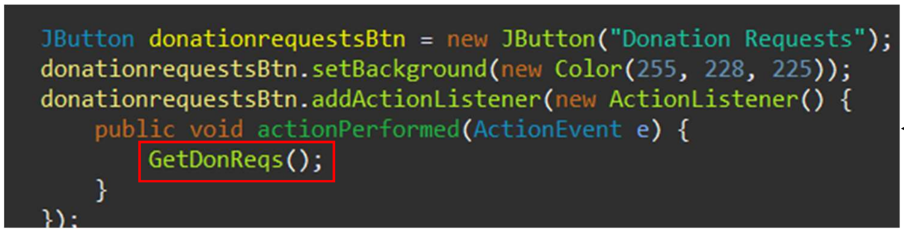
```
public void GetDonReqs() {
    Connection con = My_CNX.getConnection();
    ResultSet rs;
    PreparedStatement stmt;
    int a1 = 0, a2 = 0, a3 = 0, a4 = 0, a5 = 0;
    String reQ = "Select * from total";
    try {
        stmt = con.prepareStatement(reQ);
        rs = stmt.executeQuery();
        while (rs.next()) {
            a1 = rs.getInt(2);
            a2 = rs.getInt(3);
            a3 = rs.getInt(4);
            a4 = rs.getInt(5);
            a5 = rs.getInt(6);

        }
        if (a1 == 0) {
            JOptionPane.showMessageDialog(null, "Blankets are Required");
        } else if (a2 == 0) {
            JOptionPane.showMessageDialog(null, "Meals are Required");
        } else if (a3 == 0) {
            JOptionPane.showMessageDialog(null, "Dry Rations are Required");
        } else if (a4 == 0) {
            JOptionPane.showMessageDialog(null, "Shoes are Required");
        } else if (a5 == 0) {
            JOptionPane.showMessageDialog(null, "Clothes are Required");
        } else {
            JOptionPane.showMessageDialog(null, "All Items are sufficient");
        }
    } catch (SQLException ex) {
        System.out.println(ex);
    }
```

This method gets values for each of the type of donation and check whether any of them are 0. If any is found to be 0, a prompt shows up. If none are 0 another prompt shows up

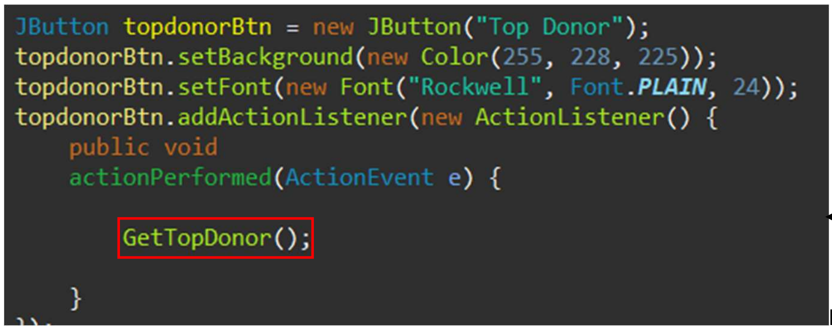


```
public boolean checkUsername(String username) {

    PreparedStatement st;
    ResultSet rs;
    boolean username_exist = false;

    String query = "SELECT * FROM `users` WHERE `username` = ?";

    try {
        st = My_CNX.getConnection().prepareStatement(query);
        st.setString(1, username);
        rs = st.executeQuery();

        if (rs.next()) {
            username_exist = true;
            JOptionPane.showMessageDialog(null, "This username is already taken, choose another Username", "Username Creation Failure ", 2);
        }


    } catch (SQLException e) {
        Logger.getLogger(Registration.class.getName()).log(Level.SEVERE, null, e);
    }

    return username_exist;

}
```

Subsequently, these methods are called where complexity of codes needs to be hidden. i.e under actionlisteners where only functionality is required and also under already complex and large parts of code where the actual implementation of the class is not necessarily required.

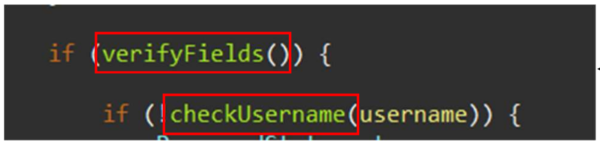


```
JButton donationrequestsBtn = new JButton("Donation Requests");
donationrequestsBtn.setBackground(new Color(255, 228, 225));
donationrequestsBtn.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        GetDonReqs();
    }
});
```



```
JButton topdonorBtn = new JButton("Top Donor");
topdonorBtn.setBackground(new Color(255, 228, 225));
topdonorBtn.setFont(new Font("Rockwell", Font.PLAIN, 24));
topdonorBtn.addActionListener(new ActionListener() {
    public void
    actionPerformed(ActionEvent e) {

        GetTopDonor();

    }
```

n

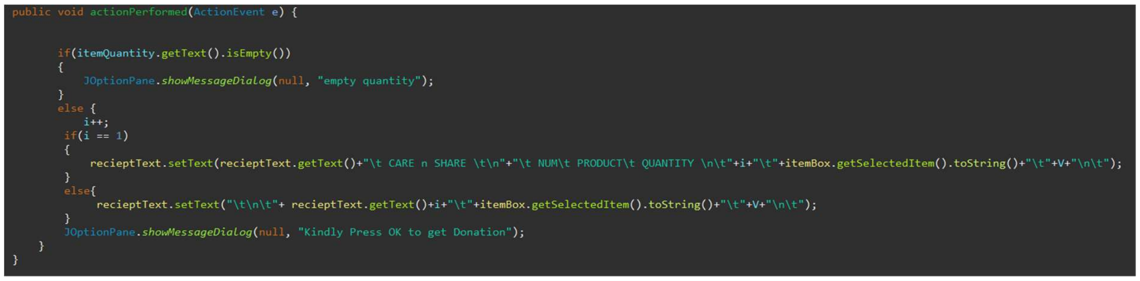


```
  if (verifyFields()) {

      if (!checkUsername(username)) {
```

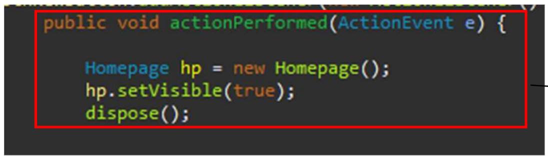
9. Event Handling

As a heavily GUI based application, interactable GUI elements are made possible through the use of action listeners associated with every interactable element. These allow for an action to be performed once user interacts with the interface/element. These are used extensively in this application.

ActionListener's are used to establish communication and communicate with the MySQL database. They are also used to dispose JFrames and open new ones once the user navigates to a new page



```
public void actionPerformed(ActionEvent e) {


       if(itemQuantity.getText().isEmpty())
       {
           JOptionPane.showMessageDialog(null, "empty quantity");
       }
       else {
           i++;
        if(i == 1)
        {
            recieptText.setText(recieptText.getText()+"\t CARE n SHARE \t\n"+"\t NUM\t PRODUCT\t QUANTITY \n\t"+i+"\t"+itemBox.getSelectedItem().toString()+"\t"+V+"\n\t");
        }
        else{
             recieptText.setText("\t\n\t"+ recieptText.getText()+i+"\t"+itemBox.getSelectedItem().toString()+"\t"+V+"\n\t");
        }
        JOptionPane.showMessageDialog(null, "Kindly Press OK to get Donation");
    }
}
```

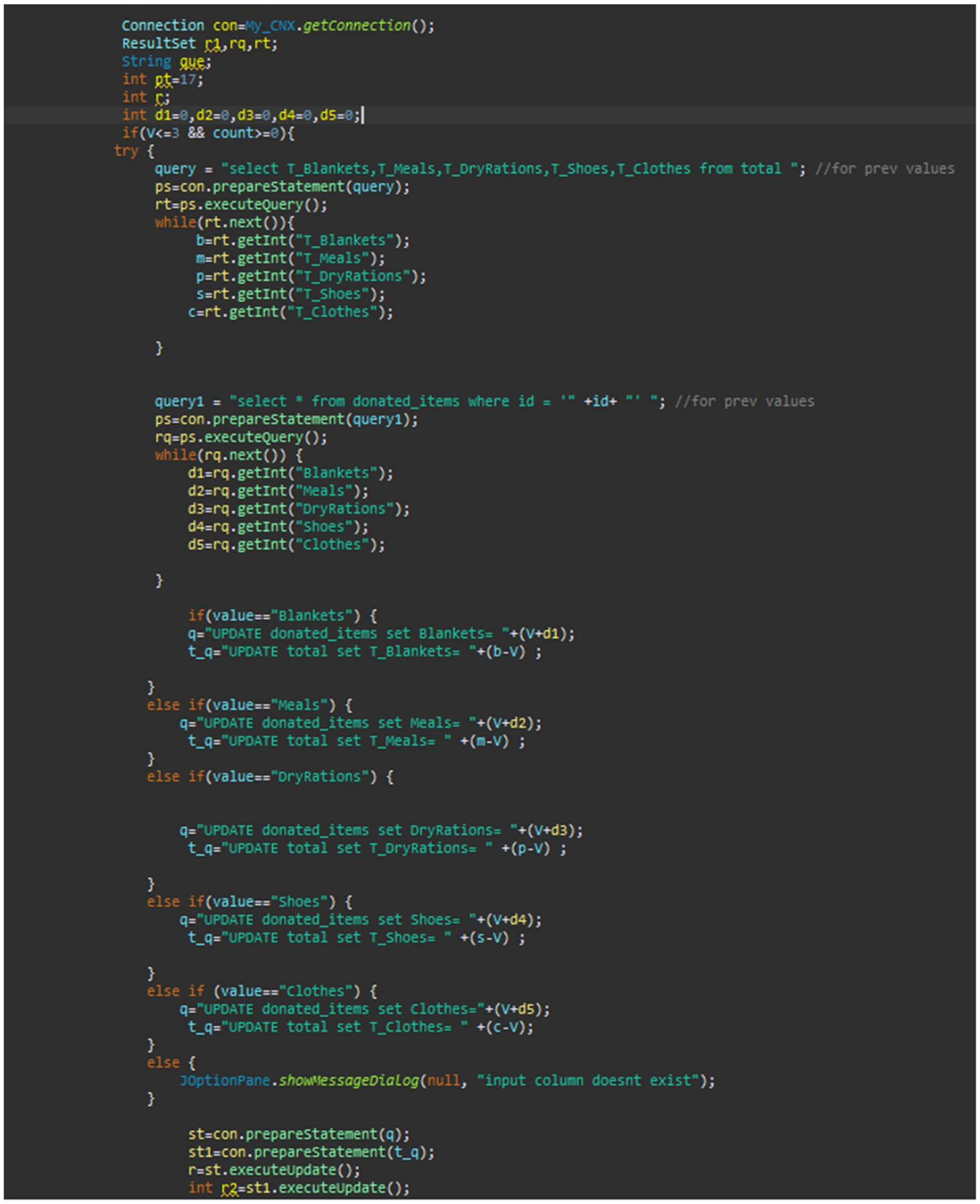


```
public void actionPerformed(ActionEvent e) {

    Homepage hp = new Homepage();
    hp.setVisible(true);
    dispose();
```

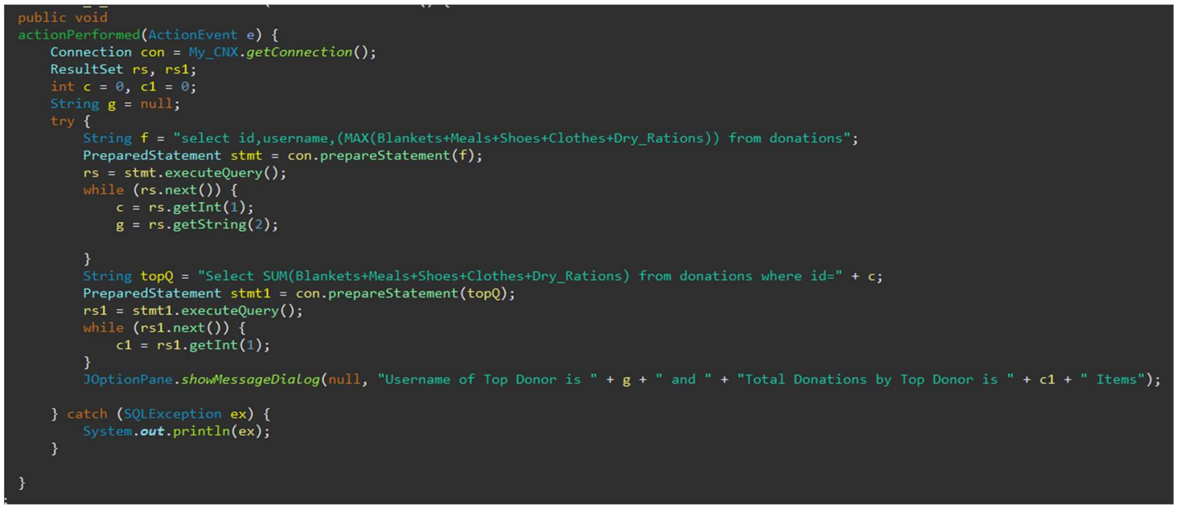
Current Jframe gets closed and new one is opened.

10. Complex MySQL queries

The application is connected to a MySQL Database. Query Language is used throughout the backend to communicate with the MySQL database. To make update the tables, fetch data and perform other functions on the tables. MySQL was used as there needed to be a way to store the accounts and donation records as well as accurately maintain donation details. This helps us fulfil Criteria 2 a, c d, e as well as 3 a, b, c, d

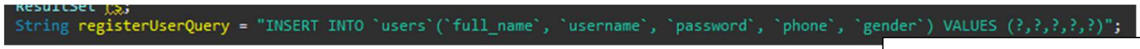


```
            Connection con=My_CNX.getConnection();
            ResultSet r1,rq,rt;
            String que;
            int pt=17;
            int r;
            int d1=0,d2=0,d3=0,d4=0,d5=0;
            if(V<=3 && count>=0){
           try {
                query = "select T_Blankets,T_Meals,T_DryRations,T_Shoes,T_Clothes from total "; //for prev values
                ps=con.prepareStatement(query);
                rt=ps.executeQuery();
                while(rt.next()){
                     b=rt.getInt("T_Blankets");
                     m=rt.getInt("T_Meals");
                     p=rt.getInt("T_DryRations");
                     s=rt.getInt("T_Shoes");
                    c=rt.getInt("T_Clothes");

                }


                query1 = "select * from donated_items where id = '" +id+ "' "; //for prev values
                ps=con.prepareStatement(query1);
                rq=ps.executeQuery();
                while(rq.next()) {
                    d1=rq.getInt("Blankets");
                    d2=rq.getInt("Meals");
                    d3=rq.getInt("DryRations");
                    d4=rq.getInt("Shoes");
                    d5=rq.getInt("Clothes");

                }

                    if(value=="Blankets") {
                    q="UPDATE donated_items set Blankets= "+(V+d1);
                    t_q="UPDATE total set T_Blankets= "+(b-V) ;

               }
               else if(value=="Meals") {
                   q="UPDATE donated_items set Meals= "+(V+d2);
                    t_q="UPDATE total set T_Meals= " +(m-V) ;
               }
               else if(value=="DryRations") {


                   q="UPDATE donated_items set DryRations= "+(V+d3);
                    t_q="UPDATE total set T_DryRations= " +(p-V) ;

               }
               else if(value=="Shoes") {
                   q="UPDATE donated_items set Shoes= "+(V+d4);
                    t_q="UPDATE total set T_Shoes= " +(s-V) ;

               }
               else if (value=="Clothes") {
                   q="UPDATE donated_items set Clothes="+(V+d5);
                    t_q="UPDATE total set T_Clothes= " +(c-V);
               }
               else {
                   JOptionPane.showMessageDialog(null, "input column doesnt exist");
               }

                    st=con.prepareStatement(q);
                    st1=con.prepareStatement(t_q);
                    r=st.executeUpdate();
                    int r2=st1.executeUpdate();
```



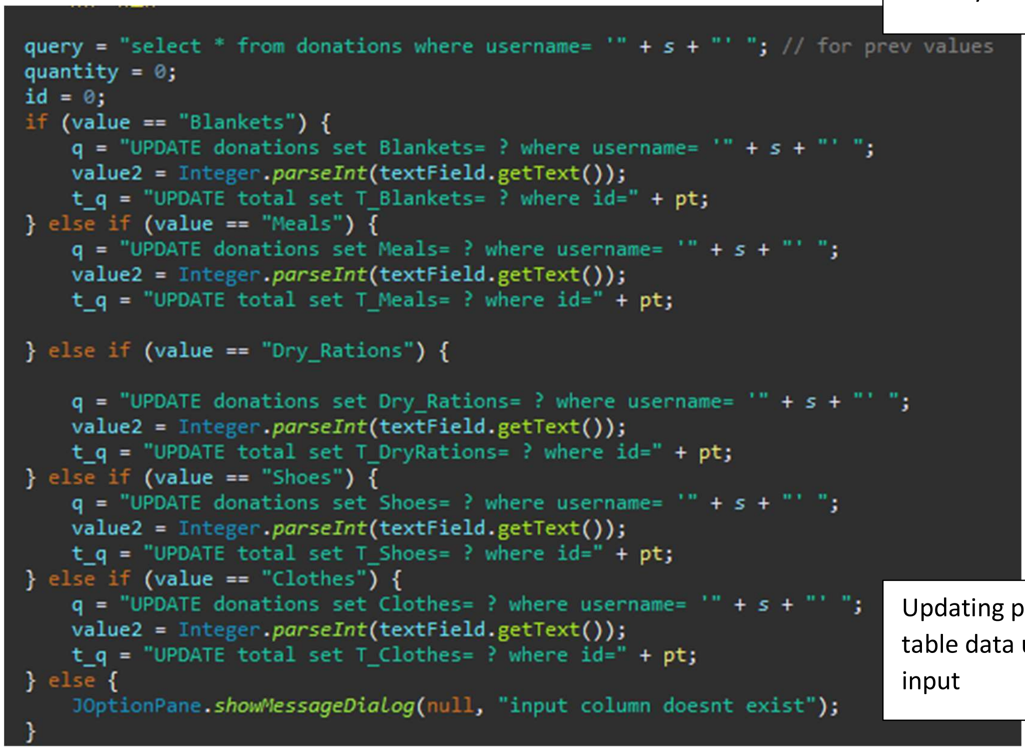
```
public void
actionPerformed(ActionEvent e) {
    Connection con = My_CNX.getConnection();
    ResultSet rs, rs1;
    int c = 0, c1 = 0;
    String g = null;
    try {
        String f = "select id,username,(MAX(Blankets+Meals+Shoes+Clothes+Dry_Rations)) from donations";
        PreparedStatement stmt = con.prepareStatement(f);
        rs = stmt.executeQuery();
        while (rs.next()) {
            c = rs.getInt(1);
            g = rs.getString(2);

        }
        String topQ = "Select SUM(Blankets+Meals+Shoes+Clothes+Dry_Rations) from donations where id=" + c;
        PreparedStatement stmt1 = con.prepareStatement(topQ);
        rs1 = stmt1.executeQuery();
        while (rs1.next()) {
            c1 = rs1.getInt(1);
        }
        JOptionPane.showMessageDialog(null, "Username of Top Donor is " + g + " and " + "Total Donations by Top Donor is " + c1 + " Items");

    } catch (SQLException ex) {
        System.out.println(ex);
    }

}
```

For explanation of the aforementioned queries check “8. User defined methods”



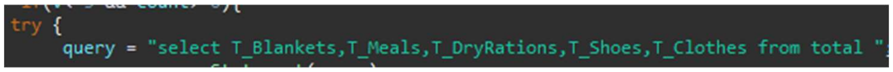
```
String registerUserQuery = "INSERT INTO `users`(`full_name`, `username`, `password`, `phone`, `gender`) VALUES (?,?,?,?,?)";
```

Insertion of records directly into tables.



```
query = "select * from donations where username= '" + s + "' "; // for prev values
quantity = 0;
id = 0;
if (value == "Blankets") {
    q = "UPDATE donations set Blankets= ? where username= '" + s + "' ";
    value2 = Integer.parseInt(textField.getText());
    t_q = "UPDATE total set T_Blankets= ? where id=" + pt;
} else if (value == "Meals") {
    q = "UPDATE donations set Meals= ? where username= '" + s + "' ";
    value2 = Integer.parseInt(textField.getText());
    t_q = "UPDATE total set T_Meals= ? where id=" + pt;

} else if (value == "Dry_Rations") {

    q = "UPDATE donations set Dry_Rations= ? where username= '" + s + "' ";
    value2 = Integer.parseInt(textField.getText());
    t_q = "UPDATE total set T_DryRations= ? where id=" + pt;
} else if (value == "Shoes") {
    q = "UPDATE donations set Shoes= ? where username= '" + s + "' ";
    value2 = Integer.parseInt(textField.getText());
    t_q = "UPDATE total set T_Shoes= ? where id=" + pt;
} else if (value == "Clothes") {
    q = "UPDATE donations set Clothes= ? where username= '" + s + "' ";
    value2 = Integer.parseInt(textField.getText());
    t_q = "UPDATE total set T_Clothes= ? where id=" + pt;
} else {
    JOptionPane.showMessageDialog(null, "input column doesnt exist");
}
```

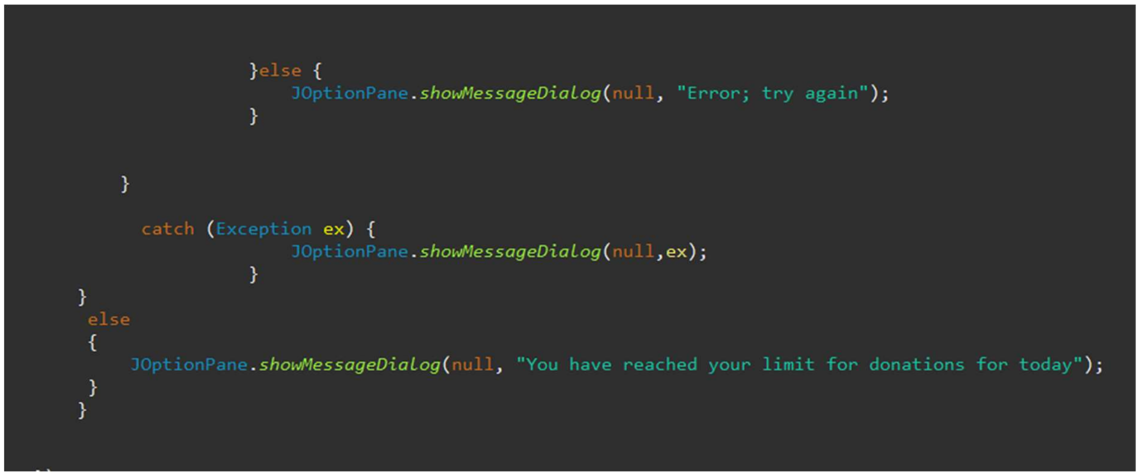
Updating preexisting table data using user input



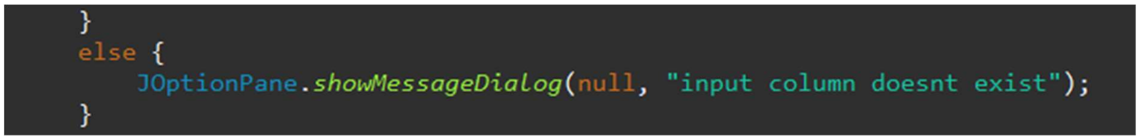
```
try {
    query = "select T_Blankets,T_Meals,T_DryRations,T_Shoes,T_Clothes from total "
```

11. Exception Handling

Exception handling is used throughout the application to ensure the smooth functioning of the application even when extreme conditions occur. If exceptions are not handled the program may crash and so this helps in fulfilling criteria 1a is it makes the program more user friendly



```
                    }else {
                        JOptionPane.showMessageDialog(null, "Error; try again");
                    }


            }

            catch (Exception ex) {
                        JOptionPane.showMessageDialog(null,ex);
                    }
    }
     else
     {
         JOptionPane.showMessageDialog(null, "You have reached your limit for donations for today");
     }
    }
```

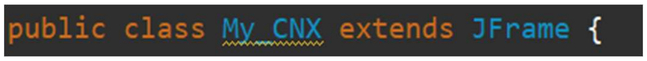


```
    }
    else {
        JOptionPane.showMessageDialog(null, "input column doesnt exist");
    }
```

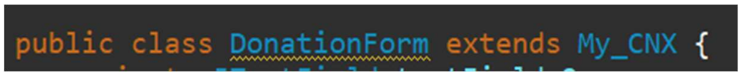
12. Multi-level inheritance

Multi level inheritance is used as each GUI class represents a JFrame and to define a JFrame, a class needs to extend “Jframe”. For example as “DonationForm” is a GUI class it needs to extend the Jframe class. This is because predefined methods of the JFrame class are necessary to develop the GUI properly.

However, we also need to inherit the methods in the My_CNX class as methods from that class are required as well. In simple words to get we need the JFrame class to be the superclass and to do that we can make “My_CNX” inherit the methods of “JFrame” class and then make all classes that require methods from both classes inherit methods of the My_CNX class. i.e



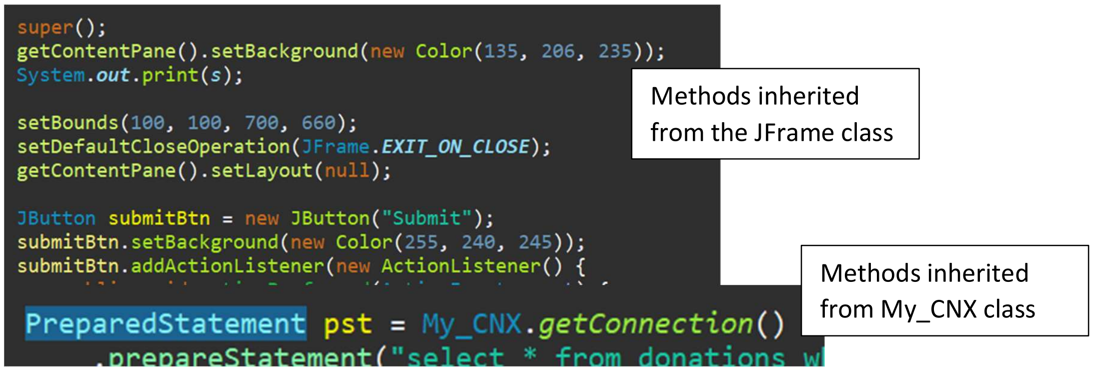
```
public class My_CNX extends JFrame {
```



```
public class DonationForm extends My_CNX {
```

Now Donation form can call methods from the JFrame class as well as the My_CNX class.

Examples



```
super();
getContentPane().setBackground(new Color(135, 206, 235));
System.out.print(s);

setBounds(100, 100, 700, 660);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
getContentPane().setLayout(null);

JButton submitBtn = new JButton("Submit");
submitBtn.setBackground(new Color(255, 240, 245));
submitBtn.addActionListener(new ActionListener() {
```

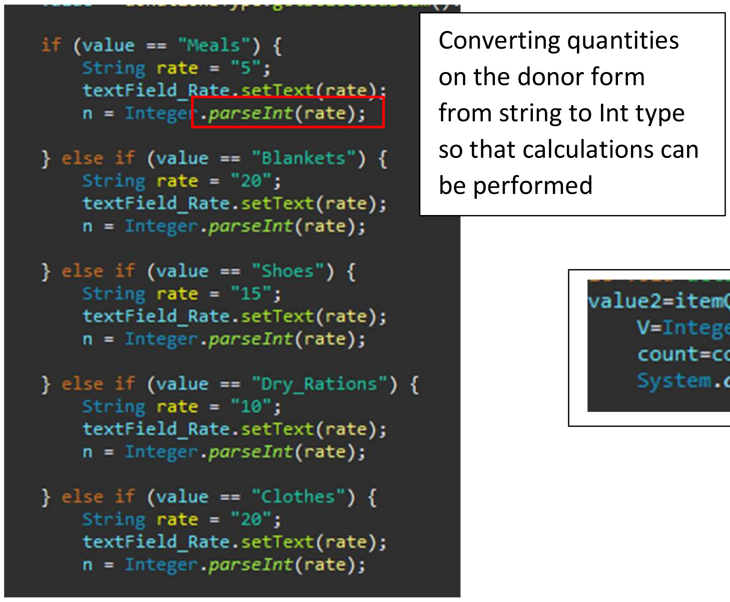
```
PreparedStatement pst = My_CNX.getConnection()
    .prepareStatement("select * from donations w
```

Methods inherited from the JFrame class

Methods inherited from My_CNX class

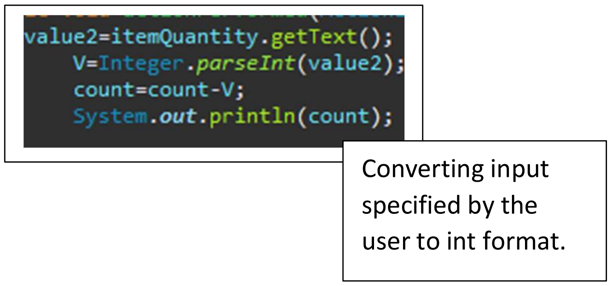
13. Parsing

User input is taken in lots of areas in this application. As user input is normally in string format the .parseInt() method is used to convert string numbers into integer format so that they can be prepared for insertion into database. This indirectly helps us achieve a requirement of criteria 2b, e and 3d, e



```
if (value == "Meals") {
    String rate = "5";
    textField_Rate.setText(rate);
    n = Integer.parseInt(rate);

} else if (value == "Blankets") {
    String rate = "20";
    textField_Rate.setText(rate);
    n = Integer.parseInt(rate);

} else if (value == "Shoes") {
    String rate = "15";
    textField_Rate.setText(rate);
    n = Integer.parseInt(rate);

} else if (value == "Dry_Rations") {
    String rate = "10";
    textField_Rate.setText(rate);
    n = Integer.parseInt(rate);

} else if (value == "Clothes") {
    String rate = "20";
    textField_Rate.setText(rate);
    n = Integer.parseInt(rate);
```

Converting quantities on the donor form from string to Int type so that calculations can be performed



```
value2=itemQuantity.getText();
    V=Integer.parseInt(value2);
    count=count-V;
    System.out.println(count);
```

Converting input specified by the user to int format.

Bibliography

GeeksforGeeks. 2021. *Inheritance in Java - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/inheritance-in-java/> [Accessed 4 October 2021].

Braunschweig, D., 2021. *Encapsulation*. [online] Press.rebus.community. Available at: <https://press.rebus.community/programmingfundamentals/chapter/encapsulation> [Accessed 4 October 2021].

Oracle Help Center. 2021. *Java Documentation - Get Started*. [online] Available at: <https://docs.oracle.com/en/java/> [Accessed 4 October 2021].

GeeksforGeeks. 2021. *Multidimensional Arrays in Java - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/multidimensional-arrays-in-java/> [Accessed 4 October 2021].

Docs.oracle.com. 2021. *javax.swing (Java Platform SE 7)*. [online] Available at: <https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html> [Accessed 4 October 2021].