

## Table of Contents

|  |                  |
|--|------------------|
| <b><i>Libraries imported .....</i></b>                                   | <b><i>2</i></b>  |
| <b><i>Connection to MySQL database .....</i></b>                         | <b><i>3</i></b>  |
| <b><i>Adding items to the database .....</i></b>                         | <b><i>3</i></b>  |
| <b><i>Updating items present in the database .....</i></b>               | <b><i>4</i></b>  |
| <b><i>Deleting items from the database.....</i></b>                      | <b><i>4</i></b>  |
| <b><i>Searching.....</i></b>   | <b><i>4</i></b>  |
| <b><i>Exception handling .....</i></b>                                   | <b><i>5</i></b>  |
| <b><i>Encapsulation .....</i></b>  | <b><i>6</i></b>  |
| <b><i>Inheritance .....</i></b>  | <b><i>6</i></b>  |
| <b><i>Polymorphism .....</i></b>   | <b><i>6</i></b>  |
| <b><i>Graphing .....</i></b>   | <b><i>7</i></b>  |
| <b><i>Printing.....</i></b>  | <b><i>7</i></b>  |
| <b><i>Complex conditional statements .....</i></b>                       | <b><i>8</i></b>  |
| <b><i>Populating tables with information from the database .....</i></b> | <b><i>9</i></b>  |
| <b><i>Double dimensional arrays.....</i></b>                             | <b><i>9</i></b>  |
| <b><i>Dynamic data structures .....</i></b>                              | <b><i>10</i></b> |
| <b><i>Graphical User Interface.....</i></b>                              | <b><i>11</i></b> |
| <b><i>Notification popup .....</i></b>                                   | <b><i>12</i></b> |
| <b><i>Uses relationship .....</i></b>                                    | <b><i>13</i></b> |

## Libraries imported

```
import java.awt.HeadlessException;
import java.awt.print.PrinterException;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.table.DefaultTableModel;
import net.proteanit.sql.DbUtils;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;
```

- SQL was used for the program to communicate with the database used to hold the data.
- Swing was used in order to create all the graphical components of the software.
- JFreeChart was used in order to create a dataset and a bar graph
- Input/output package used for printing purpose

## Connection to MySQL database<sup>1</sup>

```
public Connection getConnection() {  
    try {  
        Connection con=null;  
        con=DriverManager.getConnection("jdbc:mysql://localhost:8889/COMP IA","root","root");  
        return con;  
    } catch (Exception obj) {  
        JOptionPane.showMessageDialog(null,"Not connected");  
        return null;  
    }  
}
```

getConnection() uses the Connection class function getConnection(). Here, the address for the database is one parameter and the other two are the username and password of the database. The connection is made which other functions in the class can use for their prepared statements. Because the classes had to connect to the database regularly, it made sense to write a function which connects the program to the database as it allows for reusability of code.

## Adding items to the database<sup>2</sup>

```
String sql="INSERT INTO Orders(Customer_Name, Product_Name, Quantity, Price, Delivery_Date)" + " VALUES (?, ?, ?, ?, ?)";
```

Orders is the name of the database table. The values in the brackets are the column names and the question marks signify that the value comes from the prepared statement and not with the query itself. In the query, the INSERT function is clearly mentioned.

```
pst=con.prepareStatement(sql);  
pst.setString(1, CustomerName.getText());  
pst.setString(2, (String) ProductName.getSelectedItemAt());  
pst.setString(3, quantity.getText());  
pst.setString(4, Price.getText());  
DateFormat df = new SimpleDateFormat("yyyy-MM-dd");  
pst.setString(5, df.format(DeliveryDate.getDate()));  
pst.setString(6, srno.getText());  
pst.executeUpdate();
```

The '?' values are provided by the code on the left. Values from the text field are taken and inserted into the database table. After adding the values, the update is executed, after which the values, can be

seen in the database table.

This links to success criterion c<sup>3</sup>.

---

<sup>1</sup> Refer to Appendix 5: Code; Menu page class

<sup>2</sup> Refer to Appendix 5: Code; Orders for the day class

<sup>3</sup> Refer to Criterion A: Planning for success criteria

### Updating items present in the database<sup>4</sup>

In this query, the function UPDATE is stated explicitly, highlighting the purpose of the function

```
String sql="UPDATE Orders SET Customer_Name=?,Product_Name=?,Quantity=?, Price=?,Delivery_Date=? WHERE Serial=?";
```

clearly. The query states to update the values of the different columns WHERE the serial number is '?'. Like in the insert function, the '?' values are specified later in the function.

```
pst.setString(1, name.getText());
pst.setString(2, price.getText());
pst.setString(3, quantity.getText());
pst.setString(4, serial.getText());
pst.executeUpdate();
```

This links to success criterion d.

### Deleting items from the database<sup>5</sup>

```
String sql="DELETE FROM Orders WHERE Serial=?";
```

Here, the functions DELETE is again stated explicitly. The query tells the program to delete the entire row in the Orders table where serial = '?'. The serial number is then provided and the row is deleted.

```
pst.setString(1, srno.getText());
pst.executeUpdate();
```

This links to success criterion e.

### Searching<sup>6</sup>

```
private void filterKeyReleased(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    try {
        String selection=(String)select.getSelectedItemAt();
        Class.forName("com.mysql.cj.jdbc.Driver");
        con=DriverManager.getConnection("jdbc:mysql://localhost:8889/COMP IA","root","root");
        String sql="SELECT * from Lifetime_orders where "+selection+"=?";
        pst=con.prepareStatement(sql);
        pst.setString(1, filter.getText());
        rs=pst.executeQuery();
        OrdersTable.setModel(DbUtils.resultSetToTableModel(rs));
    }
    catch(Exception obj) {
        JOptionPane.showMessageDialog(null, obj);
    }
}
```

- 1- 'selection' holds the parameter for search and initially, the program selects everything from the database table.
- 2- 'filter' is the text field where the user enters the search keywords.
- 3- rs is the resultset which holds every value where the value entered by the user is present. This is then transferred to the table using the DbUtils class.


<sup>4</sup> Refer to Appendix 5: Code; Orders for the day class

<sup>5</sup> Refer to Appendix 5: Code; Orders for the day class

<sup>6</sup> Refer to Appendix 5: Code; Lifetime orders class

Another type of searching is between two date ranges. This is done using the following code:

```
private void sortDateActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
  
    DateFormat df = new SimpleDateFormat("yyyy-MM-dd");  
    String val1=df.format(datechoose1.getDate());  
    String val2=df.format(datechoose2.getDate());  
    try{  
        Class.forName("com.mysql.cj.jdbc.Driver");  
        Connection con=DriverManager.getConnection("jdbc:mysql://localhost:8889/COMP_IA","root","root");  
        String sql="SELECT * from Lifetime_orders WHERE Delivery_Date BETWEEN '"+val1+"' and '"+val2+"'";  
        PreparedStatement pst=con.prepareStatement(sql);  
        ResultSet rs=pst.executeQuery();  
        OrdersTable.setModel(DbUtils.resultSetToTableModel(rs));  
    }  
    catch(Exception e){  
        JOptionPane.showMessageDialog(null,e);  
    }  
}
```



- 1- The dates chosen by the user get stored in two strings: val1 and val2 which matches the format of the dates stored in the database.
- 2- The query clearly mentions SELECT every item where the delivery date is BETWEEN val1 and val2. Then the table is populated.

This links to success criterion *g*.

### Exception handling

Try-catch block has been used to ensure that the program does not crash while it is running.

An example is:

```
// TODO add your handling code here:  
try {  
    boolean complete=receipt.print();  
    if(complete) {  
        JOptionPane.showMessageDialog(null,"Done printing","Information",JOptionPane.INFORMATION_MESSAGE);  
    }  
    else {  
        JOptionPane.showMessageDialog(null,"Printing","Printer",JOptionPane.ERROR_MESSAGE);  
    }  
}  
catch(HeadlessException | PrinterException obj) {  
    JOptionPane.showMessageDialog(null, obj);  
  
    receipt.setText("");  
}  
}
```

In the above example, specific exceptions have been written because they are the only two exceptions that the program can face in the given function. This increases the program's efficiency.

## Encapsulation

```
// Variables declaration - do not modify
private javax.swing.JButton exit;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JPanel jPanel1;
private javax.swing.JButton login;
private javax.swing.JPasswordField password;
private javax.swing.JButton reset;
private javax.swing.JTextField username;
// End of variables declaration
```

The access specified to the data members is private and hence, accessor and mutator functions will have to be used to access/change values of variables. An example is:

```
srno.setText("");
CustomerName.setText("");
ProductName.setSelectedItem("-");
Price.setText("");
quantity.setText("");

, CustomerName.getText();
String)ProductName.getSelectedSelectedItem();
, (String) ProductName.getSelectedSelectedItem();
, quantity.getText();
eger.parseInt(quantity.getText());
, Price.getText();
```

## Inheritance

Each class inherits the JFrame class, hence, they all receive the different properties of the JFrame class, like buttons, combo boxes and panels. Hence, these items don't need to be imported to the class. The following is an example of inheritance.

```
public class EditingForm extends javax.swing.JFrame {
```

## Polymorphism

```
public Connection getConnection() {
    try {
        Connection con=null;
        con=DriverManager.getConnection("jdbc:mysql://localhost:8889/COMP IA","root","root");
        return con;
    } catch (Exception obj) {
        JOptionPane.showMessageDialog(null,"Not connected");
        return null;
    }
}
```

The above function's name is getConnection() and it uses the library function getConnection(). However, due to different function signatures, this allowed for reusability of the above code.

## Graphing<sup>7</sup>

```
try {
    DefaultCategoryDataset ds=new DefaultCategoryDataset();
    ds.setValue(Integer.valueOf(jan.getText()),"Sales","January");
    ds.setValue(Integer.valueOf(feb.getText()),"Sales","February");
    ds.setValue(Integer.valueOf(mar.getText()),"Sales","March");
    ds.setValue(Integer.valueOf(apr.getText()),"Sales","April");
    ds.setValue(Integer.valueOf(may.getText()),"Sales","May");
    ds.setValue(Integer.valueOf(jun.getText()),"Sales","June");
    ds.setValue(Integer.valueOf(july.getText()),"Sales","July");
    ds.setValue(Integer.valueOf(aug.getText()),"Sales","August");
    ds.setValue(Integer.valueOf(sept.getText()),"Sales","September");
    ds.setValue(Integer.valueOf(oct.getText()),"Sales","October");
    ds.setValue(Integer.valueOf(nov.getText()),"Sales","November");
    ds.setValue(Integer.valueOf(dec.getText()),"Sales","December");

    JFreeChart barchart=ChartFactory.createBarChart("Monthly sales", "Months", "Sales", ds, PlotOrientation.VERTICAL, false, true, false);
    CategoryPlot bar=barchart.getCategoryPlot();
    bar.setRangeGridlinePaint(Color.BLACK);

    ChartFrame frame;
    frame = new ChartFrame("BAR CHART FOR SALES",barchart);
    frame.setVisible(true);
    frame.setSize(1000,600);
    jan.setEditable(false);
    feb.setEditable(false);
    mar.setEditable(false);
    apr.setEditable(false);
    may.setEditable(false);
    jun.setEditable(false);
    july.setEditable(false);
    aug.setEditable(false);
    sept.setEditable(false);
    oct.setEditable(false);
    nov.setEditable(false);
    dec.setEditable(false);
}
catch(Exception obj) {
    JOptionPane.showMessageDialog(null,"Click on the Monthly Sales Button");
}
```

- 1- A dataset is being created in order to store the values of the monthly sales.
- 2- A new frame is being created in order to display the chart. Labels for the X and Y axis are added to make the graph understandable.

This fulfills success criterion *h*.

## Printing

Code:

```
private void PrintActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    try {
        boolean complete=receipt.print();
        if(complete) {
            JOptionPane.showMessageDialog(null,"Done printing","Information",JOptionPane.INFORMATION_MESSAGE);
        }
        else {
            JOptionPane.showMessageDialog(null,"Printing","Printer",JOptionPane.ERROR_MESSAGE);
        }
    }
    catch(HeadlessException | PrinterException obj) {
        JOptionPane.showMessageDialog(null, obj);

        receipt.setText("");
    }
}
```

<sup>7</sup> Refer to Appendix 5: Code; total sales class

The library function `.print()` is being used in order to perform the printing. The user gets a message 'completed' if the printing has been completed, else he/she gets a 'printing' message.

This fulfills success criterion *i*.

### Complex conditional statements<sup>8</sup>

The program checks for whether the user is an admin or an employee. This is achieved using the following through if statements and the following code is used.

```
if(rs.next()) {
    if(username.getText().equalsIgnoreCase("admin")) {
        Admin obj1=new Admin();
        obj1.setVisible(true);
        setVisible(false);
    }
    else {
        EmployeePage obj2=new EmployeePage();
        obj2.setVisible(true);
        setVisible(false);
    }
}
```

This links to success criterion *a*.

Another example is when an employee is being added.

```
while(flag) {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        con=DriverManager.getConnection("jdbc:mysql://localhost:8889/COMP IA","root","root");
        String sql="SELECT Name from Employees WHERE Name='"+name.getText()+"'";
        Statement stat=con.createStatement();
        rs=stat.executeQuery(sql);
        if(rs.next()) {
            JOptionPane.showMessageDialog(null, "User already registered!");
            break;
        }

        else {
            sql="INSERT INTO Employees(Name, Username, Password, Status) VALUES (?, ?, ?, ?)";
            con=DriverManager.getConnection("jdbc:mysql://localhost:8889/COMP IA","root","root");
            pst=con.prepareStatement(sql);
            pst.setString(1, name.getText());
            pst.setString(2, username.getText());
            pst.setString(3, password.getText());
            pst.setString(4, Status.getSelectedItem().toString());
            pst.executeUpdate();
            JOptionPane.showMessageDialog(null, "Inserted successfully");
            flag=false;
        }
        con.close();
    } catch (Exception obj) {
        JOptionPane.showMessageDialog(null, obj);
    }
    showTableData();
}
```

Here, the program checks whether the same user is already present in the table. If not, the user is added, else the program gives an error.

---

<sup>8</sup> Refer to Appendix 5: Code; login class



## Populating tables with information from the database

```
public void showTableData() {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        con=DriverManager.getConnection("jdbc:mysql://localhost:8889/COMP IA","root","root");
        String s="Delivered";
        String sql="SELECT * from Lifetime_Orders";
        pst=con.prepareStatement(sql);
        rs=pst.executeQuery();

        OrdersTable.setModel(DbUtils.resultSetToTableModel(rs));

        con.close();
    }
    catch(Exception obj) {
        JOptionPane.showMessageDialog(null,obj);
    }
}
```

Connection to the database is being made and all the data is being stored in the prepared statement.

Using the DbUtils class, the table is populated with data from the database.

## Double dimensional arrays

Used in order to initialize the jTable. Code:

```
OrdersTable.setModel(new javax.swing.table.DefaultTableModel(
    new Object [][] {
        {null, null, null, null, null, null},
        {null, null, null, null, null, null},
        {null, null, null, null, null, null},
        {null, null, null, null, null, null}
    },
    new String [] {
        "ID", "NAME", "PRICE", "QUANTITY", "TOTAL", "STATUS"
    }
));
```

The values were provided to the table using the DbUtils class which provides library functions

to get the values from the database to the table.

## Dynamic data structures

In order to store the names of the stock items which keep on changing, an ArrayList was used. Implementation:

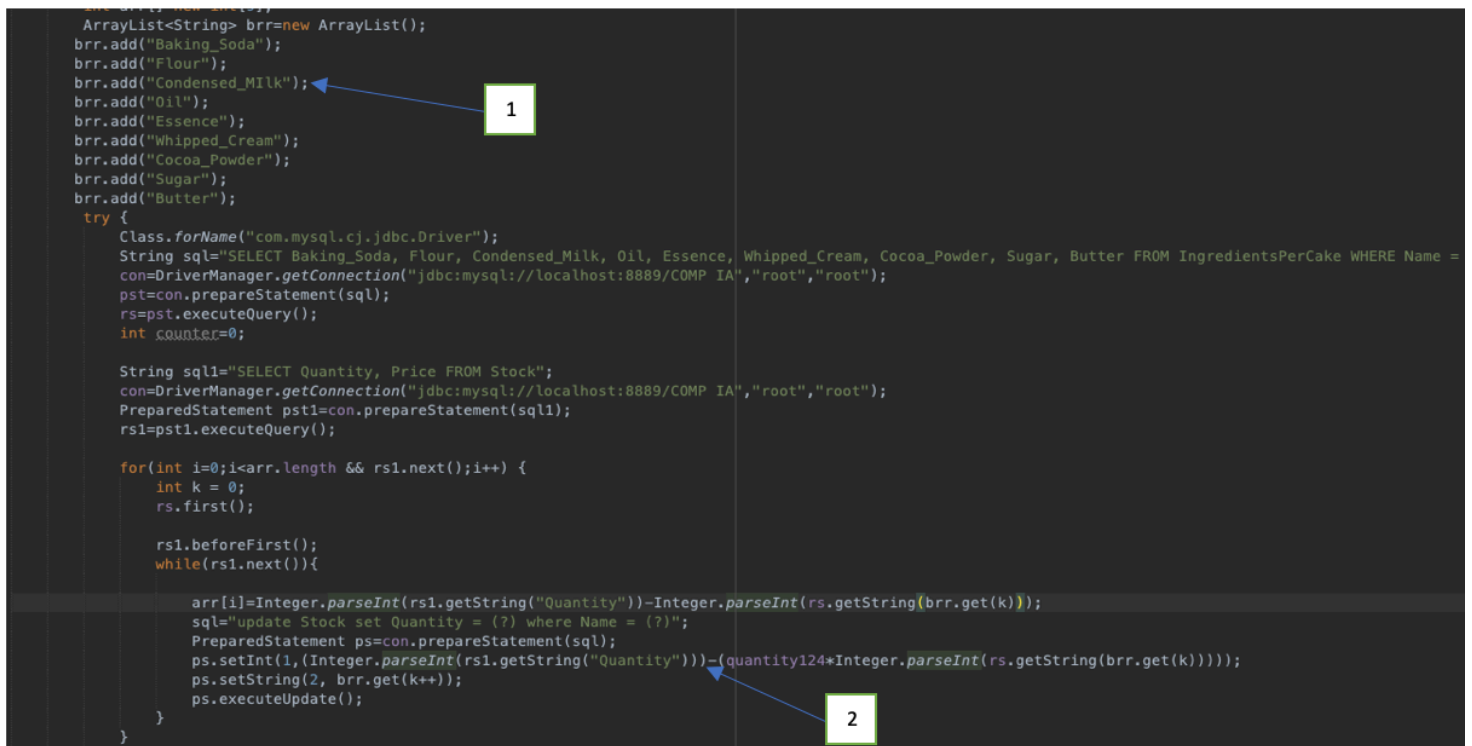
```
ArrayList<String> brr=new ArrayList();
brr.add("Baking_Soda");
brr.add("Flour");
brr.add("Condensed_Milk");
brr.add("Oil");
brr.add("Essence");
brr.add("Whipped_Cream");
brr.add("Cocoa_Powder");
brr.add("Sugar");
brr.add("Butter");
try {
    Class.forName("com.mysql.cj.jdbc.Driver");
    String sql="SELECT Baking_Soda, Flour, Condensed_Milk, Oil, Essence, Whipped_Cream, Cocoa_Powder, Sugar, Butter FROM IngredientsPerCake WHERE Name = ?";
    con=DriverManager.getConnection("jdbc:mysql://localhost:8889/COMP_IA","root","root");
    pst=con.prepareStatement(sql);
    rs=pst.executeQuery();
    int counter=0;

    String sql1="SELECT Quantity, Price FROM Stock";
    con=DriverManager.getConnection("jdbc:mysql://localhost:8889/COMP_IA","root","root");
    PreparedStatement pst1=con.prepareStatement(sql1);
    rs1=pst1.executeQuery();

    for(int i=0;i<brr.length && rs1.next();i++) {
        int k = 0;
        rs.first();

        rs1.beforeFirst();
        while(rs1.next()){

            arr[i]=Integer.parseInt(rs1.getString("Quantity"))-Integer.parseInt(rs.getString(brr.get(k)));
            sql="update Stock set Quantity = (?) where Name = (?)";
            PreparedStatement ps=con.prepareStatement(sql);
            ps.setInt(1,Integer.parseInt(rs1.getString("Quantity"))-(Integer.parseInt(rs.getString(brr.get(k)))));
            ps.setString(2, brr.get(k++));
            ps.executeUpdate();
        }
    }
}
```



- 1- Items are added to the list
- 2- The quantity required to make an item a given quantity of stock is removed from the total amount of stock in the database. Here, the use of an ArrayList is better than an array because it allows for future expandability if more stock items are added.

## Graphical User Interface

```
Price.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        PriceActionPerformed(evt);
    }
});

back.setText("Back");
back.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        backActionPerformed(evt);
    }
});

ProductName.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent evt) {
        ProductNameMousePressed(evt);
    }
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        ProductNameMouseClicked(evt);
    }
    public void mouseExited(java.awt.event.MouseEvent evt) {
        ProductNameMouseExited(evt);
    }
});
ProductName.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        ProductNameActionPerformed(evt);
    }
});
```

Calling the respective functions  
when certain actions are performed.  
For example, clicking on a button

```
JMenuItem1 = new javax.swing.JMenuItem();
JMenu1 = new javax.swing.JMenu();
JPanel1 = new javax.swing.JPanel();
JPanel2 = new javax.swing.JPanel();
JLabel1 = new javax.swing.JLabel();
JLabel2 = new javax.swing.JLabel();
JLabel3 = new javax.swing.JLabel();
JLabel4 = new javax.swing.JLabel();
srno = new javax.swing.JTextField();
Insert = new javax.swing.JButton();
Update = new javax.swing.JButton();
Delete = new javax.swing.JButton();
JLabel5 = new javax.swing.JLabel();
CustomerName = new javax.swing.JTextField();
Price = new javax.swing.JTextField();
back = new javax.swing.JButton();
ProductName = new javax.swing.JComboBox();
DeliveryDate = new com.toedter.calendar.JDateChooser();
date = new javax.swing.JLabel();
JLabel8 = new javax.swing.JLabel();
quantity = new javax.swing.JTextField();
getprice = new javax.swing.JButton();
JPanel3 = new javax.swing.JPanel();
JScrollPane1 = new javax.swing.JScrollPane();
OrdersTable = new javax.swing.JTable();
filter = new javax.swing.JTextField();
JLabel7 = new javax.swing.JLabel();
select = new javax.swing.JComboBox();
JScrollPane3 = new javax.swing.JScrollPane();
Receipt = new javax.swing.JTextArea();
Print = new javax.swing.JButton();
generateReceipt = new javax.swing.JButton();
JButton2 = new javax.swing.JButton();
JButton4 = new javax.swing.JButton();
JButton5 = new javax.swing.JButton();
JButton6 = new javax.swing.JButton();
JButton7 = new javax.swing.JButton();
JButton8 = new javax.swing.JButton();
JButton9 = new javax.swing.JButton();
```

Initializing  
the GUI  
components

```
javax.swing.GroupLayout jPanel2Layout = new javax.swing.GroupLayout(jPanel2);
jPanel2.setLayout(jPanel2Layout);
jPanel2Layout.setHorizontalGroup(
    jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(jPanel2Layout.createSequentialGroup()
            .addGap(27, 27, 27)
            .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(jPanel2Layout.createSequentialGroup()
                    .addComponent(date, javax.swing.GroupLayout.PREFERRED_SIZE, 335, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(back)
                    .addContainerGap()
                )
                .addGroup(jPanel2Layout.createSequentialGroup()
                    .addComponent(insert)
                    .addGap(64, 64, 64)
                    .addComponent(update)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 39, Short.MAX_VALUE)
                    .addComponent(delete)
                    .addGap(33, 33, 33)
                )
            )
        )
        .addGroup(jPanel2Layout.createSequentialGroup()
            .addComponent(srno)
            .addComponent(customerName)
            .addComponent(price)
            .addComponent(productName, 0, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addComponent(deliveryDate, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addComponent(quantity, javax.swing.GroupLayout.PREFERRED_SIZE, 235, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(getprice, javax.swing.GroupLayout.PREFERRED_SIZE, 99, javax.swing.GroupLayout.PREFERRED_SIZE)
        )
);
```

Setting the layout of  
the different  
components on the  
JFrame.

```

    jan.setEditable(false);
    feb.setEditable(false);
    mar.setEditable(false);
    apr.setEditable(false);
    may.setEditable(false);
    jun.setEditable(false);
    july.setEditable(false);
    aug.setEditable(false);
    sept.setEditable(false);
    oct.setEditable(false);
    nov.setEditable(false);
    dec.setEditable(false);

```

Here, the text fields are not editable because calculations provide the values for the fields and not any user input.

This links to success criterion *b* and *k* as the admin and the employee are able to view the required pages because of the GUI.

```

JFreeChart barchart=ChartFactory.createBarChart("Monthly sales", "Months", "Sales", ds, PlotOrientation.VERTICAL, false, true, false);
CategoryPlot bar=barchart.getCategoryPlot();
bar.setRangeGridlinePaint(Color.BLACK);

ChartFrame frame;
frame = new ChartFrame("BAR CHART FOR SALES",barchart);
frame.setVisible(true);
frame.setSize(1000,600);
Toolkit toolkit=getToolkit();
Dimension size=toolkit.getScreenSize();
frame.setLocation(size.width/2-getWidth()/2,size.height/2-getHeight()/2);

```

Creating a new frame for the graph, initializing it by giving it a heading and setting its size.

### Notification popup<sup>9</sup>

The admin gets a notification if the stock for any item is low upon login. Code:

```

public void checkStock() {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection con=DriverManager.getConnection("jdbc:mysql://localhost:8889/COMP IA","root","root");
        String sql="Select Name, Quantity from Stock";
        PreparedStatement pst=con.prepareStatement(sql);
        ResultSet rs=pst.executeQuery();
        while(rs.next()) {
            String s=rs.getString("Name");
            if(!s.equalsIgnoreCase("Essence")) {
                int quantity= Integer.parseInt(rs.getString("Quantity"));
                if(quantity<500)
                    JOptionPane.showMessageDialog(null, "Stock of "+s+" is low! Please buy new stock and update the system!");
            }
        }
    }
    catch(Exception obj) {
        JOptionPane.showMessageDialog(null, obj);
    }
}

```

Connection to database

Checks if quantity is lesser than amount told by client.

If yes, the program gives a popup upon login saying that the stock of item 's' is low.

This links to success criterion *j*.

<sup>9</sup> Refer to Appendix 5: Code; Admin Homepage class

## Uses relationship

Many classes *use* each other's data members in order to achieve multiple functions. An example is:

```
private void jButton8ActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    setVisible(false);  
    Admin obj1=new Admin();  
    obj1.setVisible(true);  
}
```

Here, the function *uses* the Admin class' frame and makes it visible through the library function setVisible().

**Word count: 1082**