

Criterion C: Development

Database

For my program in Netbeans, I have used Derby database, an in-built database of Netbeans. I have created four tables from it, where one of it is used to store the login details (username and password) and the other three tables were structured so that they could store the item name and its stock in number for the three fields - hand, face and misc. The three tables have two fields: Name of item which is in String, and the stock which is an integer value. For the login details, the two fields are username and password, both being Strings.

Libraries

In order to make my program, I had to import various libraries throughout the program. They are:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import net.proteanit.sql.DbUtils;
import javax.swing.JOptionPane;
```



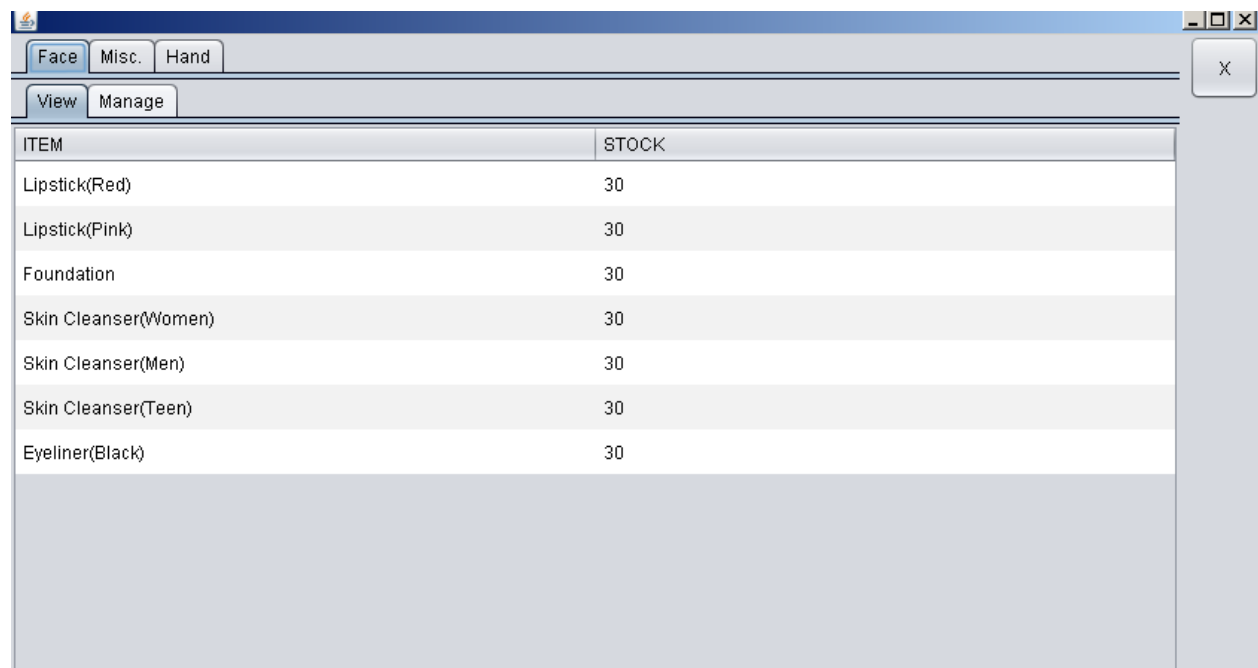
The external library, rs2xlm.jar is a library that helped me to display the products and the stock level in the JTable.

Stock Frame

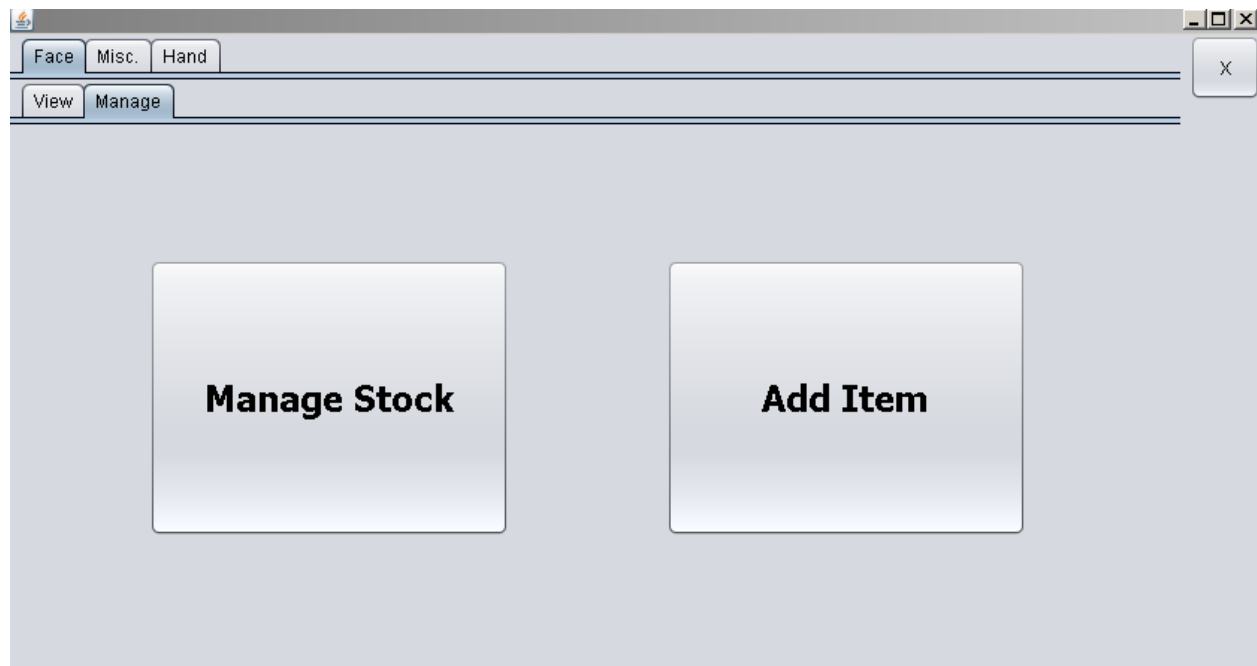
The main and the most important frame/class is the Stock frame where you can see the items and stocks of the corresponding item. The java swing components used are given below:

```
// Variables declaration - do not modify
private javax.swing.JButton jButton1;
private javax.swing.JButton jButton2;
private javax.swing.JButton jButton3;
private javax.swing.JButton jButton4;
private javax.swing.JButton jButton5;
private javax.swing.JButton jButton6;
private javax.swing.JButton jButton7;
private javax.swing.JButton jButton8;
private javax.swing.JButton jButton9;
private javax.swing.JPanel jPanel1;
private javax.swing.JPanel jPanel2;
private javax.swing.JPanel jPanel3;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JScrollPane jScrollPane2;
private javax.swing.JScrollPane jScrollPane3;
private javax.swing.JTabbedPane jTabbedPane1;
private javax.swing.JTabbedPane jTabbedPane2;
private javax.swing.JTabbedPane jTabbedPane3;
private javax.swing.JTabbedPane jTabbedPane4;
private javax.swing.JTable jTable1;
private javax.swing.JTable jTable2;
private javax.swing.JTable jTable3;
// End of variables declaration
}
```

The design of the Stock class is also given below:



ITEM	STOCK
Lipstick(Red)	30
Lipstick(Pink)	30
Foundation	30
Skin Cleanser(Women)	30
Skin Cleanser(Men)	30
Skin Cleanser(Teen)	30
Eyeliners(Black)	30



The code used to display the data from the database to the table is given below:

```
public stock() {  
    initComponents();  
    face();  
    hand();  
    misc();  
}  
private void face(){  
    try{  
        Connection conn = DriverManager.getConnection("jdbc:derby://localhost:1527/storeface","cc","cc");  
        System.out.println("Connected");  
        PreparedStatement st = conn.prepareStatement("Select * from face");  
        ResultSet rs = st.executeQuery();  
  
        jTable1.setModel(DbUtils.resultSetToTableModel(rs));  
  
    }  
    catch(Exception e){  
        System.out.println(e);  
    }  
}
```

The code given is used to display the item and the stock of the cosmetic items related to the face.

The SQL query 'Select' was used to retrieve the item and the stock. Likewise, this code is used almost identically in the other two fields 'Hand' and 'Misc.'

Exit Button

For each and every frame except the login and the change password class, there is a small button with an 'X' on the top right corner. This is a button that will allow the user to exit the program immediately.

```
private void jButton9ActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    System.exit(0);  
}
```

SQL commands

Since my program had to be linked with the database and should be able to carry out the functions such as adding new items and altering the stock, I used sql commands while making the software.

Adding new items

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    try {  
  
        Connection conn = DriverManager.getConnection("jdbc:derby://localhost:1527/storeface","cc","cc");  
        System.out.println("Connected");  
        String itemx = jTextField2.getText();  
        String stockx = jTextField1.getText();  
        int stockxx = Integer.parseInt(stockx);  
  
        PreparedStatement st = conn.prepareStatement("Insert into face (item, stock) values (?,?)");  
  
        st.setString(1, itemx);  
        st.setInt(2, stockxx);  
  
        int a = st.executeUpdate();  
  
        if (a>0) {  
            JOptionPane.showMessageDialog(null, "Data inserted to database.");  
            jTextField2.setText("");  
            jTextField1.setText("");  
  
        }  
        conn.close();  
    }  
}
```

The above code is an extract from the field Addface. The code gets the name and the initial stock of that item from the textfield and converts the string value of the stock into an integer value using `parseInt()`. Then, it inputs the records into the database using the sql command. If it is successfully executed, a pop-up message will inform the user that the data is inserted into the database. Here, the SQL command 'Insert' was used to store the new items in the database.

Updating Stock

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {  
    try {  
        Connection conn = DriverManager.getConnection("jdbc:derby://localhost:1527/storeface","cc","cc");  
        conn.setAutoCommit(false);  
        System.out.println("Connected");  
  
        String sql = "select * from face";  
        PreparedStatement st = conn.prepareStatement(sql);  
        ResultSet rs = st.executeQuery();  
        String sat = numtxt.getText();  
        int stt = Integer.parseInt(sat);  
        String rv = (String)jComboBox1.getSelectedItem();  
        while (rs.next()){  
            String name = rs.getString("item");  
            int num = rs.getInt("stock");  
            int finall = num + stt;  
            if (name.equals(rv)){  
                String spl = "update face set stock = ? where item = ?";  
                PreparedStatement pst = conn.prepareStatement(spl);  
                pst.setInt(1,finall);  
                pst.setString(2, rv);  
  
                int g = pst.executeUpdate();  
                JOptionPane.showMessageDialog(null,"Stock has been updated!");  
                numtxt.setText("");  
                pst.close();  
            }  
        }  
    }  
}
```

The above code uses an sql command which updates the stock for the item that was selected from the combobox. The user can click on the combobox and choose which item to update the stock. This is the code from the first button which is the 'Add Stock' button that adds the number in the textfield with the initial stock of the corresponding item when clicked.

The code for the button that removes stock from the initial stock is quite similar to the 'Add Stock' button. The major difference is that before subtracting the stock, it checks if the removing stock is greater than the initial stock as stocks cannot go below zero.


```

String sql = "select * from face";
PreparedStatement st = conn.prepareStatement(sql);
ResultSet rs = st.executeQuery();

String sat = numtxt.getText();
int stt = Integer.parseInt(sat);
String rv = (String)jComboBox1.getSelectedItem();
while (rs.next()){
    String name = rs.getString("item");
    int num = rs.getInt("stock");
    int finall = num - stt;
    if (name.equals(rv) && (finall>0)){
        String spl = "update face set stock = ? where item = ?";
        PreparedStatement pst = conn.prepareStatement(spl);
        pst.setInt(1,finall);
        pst.setString(2, rv);

        int a = pst.executeUpdate();

        JOptionPane.showMessageDialog(null,"Stock has been updated!");
        numtxt.setText("");
        pst.close();
    }
}

```

As seen from the code above, I have used an if-statement at the last to check if the finall(the integer variable that stores the updated stock after subtracting) is above 0.

The sql command 'update' was used for both adding and subtracting stocks as the item name remains unchanged and only the stock number gets manipulated.

The screenshot shows a software window with a light gray background. At the top left is a small icon. At the top right are standard window controls (minimize, maximize, close) and a close button labeled 'X'. On the left side, there is a dropdown menu currently showing 'Lipstick(Red)'. The dropdown list is open, showing the following items: 'Lipstick(Red)', 'Lipstick(Pink)' (highlighted), 'Foundation', 'Skin Cleanser(Women)', 'Skin Cleanser(Men)', 'Skin Cleanser(Teen)', and 'Eyeliner(Black)'. To the right of the dropdown is a white rectangular input field. Below the input field is a button labeled 'Remove Stock'. In the bottom right corner, there is a button labeled 'Back'. In the bottom left corner, the word 'Remarks' is displayed in a bold, black font.

This is how the frame looks while updating the stock. From the combobox the user can choose the item, enter the number that is to be either added or subtracted, and press the button accordingly.

The combobox is linked with the database and displays all the items within that specific field(hand/face/misc.). The code given below shows how the link between the combobox and the database is created.

```

private void FillCombo(){
    try{
        Connection conn = DriverManager.getConnection("jdbc:derby://localhost:1527/storeface","cc","cc");

        System.out.println("Connected");
        String sql = "select * from face";
        PreparedStatement st = conn.prepareStatement(sql);
        ResultSet rs = st.executeQuery();

        while (rs.next()){
            String name = rs.getString("item");
            jComboBox1.addItem(name);
        }

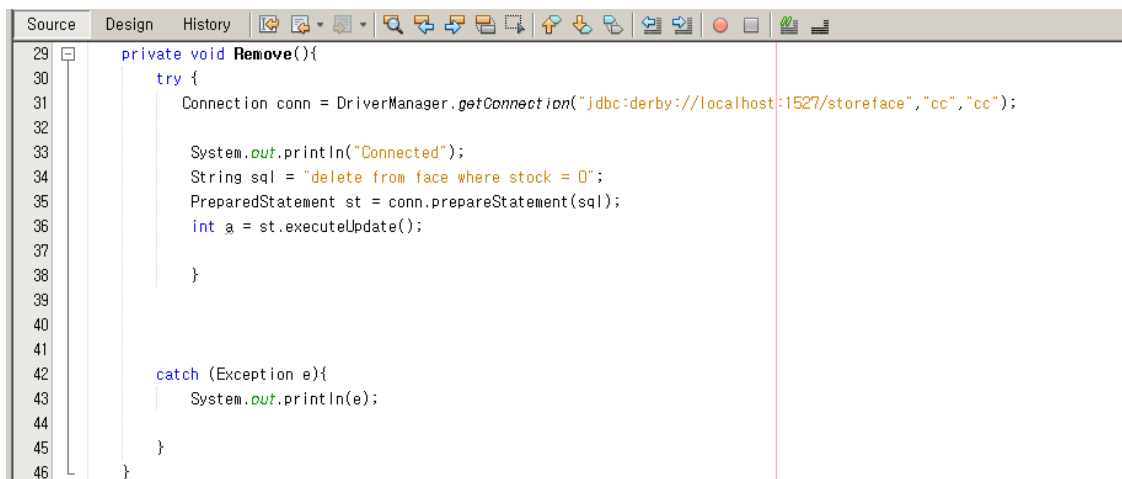
    }
    catch (Exception e){
        System.out.println(e);
    }
}
}

```

As seen from the code, I have used `jCombobox.addItem(name)` to add every item in the database into the combobox. This was done by first establishing a connection with the database, then taking the item name from the database using `executeQuery()`.

Deleting Items

As the client asked for this feature, I have used the SQL statement `DELETE` in order to delete the items that have reached 0 stock from the program. Since extreme values are not allowed by keeping a condition which is that the subtracting value cannot exceed the current stock level, the minimum stock an item can have is 0. When this occurs, the program searches for the item with the stock level 0, and removes it from the database. The code used is given below.



```

Source  Design  History
29  private void Remove(){
30      try {
31          Connection conn = DriverManager.getConnection("jdbc:derby://localhost:1527/storeface","cc","cc");
32
33          System.out.println("Connected");
34          String sql = "delete from face where stock = 0";
35          PreparedStatement st = conn.prepareStatement(sql);
36          int a = st.executeUpdate();
37
38      }
39
40
41
42      catch (Exception e){
43          System.out.println(e);
44      }
45
46  }

```

Alert

When the stock of a certain item goes below 10, the program warns the user that the specific item needs restock. The message alarm is displayed in a textbox that is found in the frames that updates the stock.

This function was added in order to meet the needs of my client. Since he wanted the program to be able to warn him whenever items should be restocked, this feature was added.

```
private void Remarks(){
    String alert = "";
    try {
        Connection conn = DriverManager.getConnection("jdbc:derby://localhost:1527/storeface","cc","cc");
        conn.setAutoCommit(false);
        System.out.println("Connected");
        String sql = "select * from face";
        PreparedStatement st = conn.prepareStatement(sql);
        ResultSet rs = st.executeQuery();
        while (rs.next()){
            String name = rs.getString("item");
            int num = rs.getInt("stock");
            if (num<11){
                alert = alert + "\n" + name + " needs RESTOCK!";
            }
        }
        jLabel2.setText(alert);
    }

    catch (Exception e){
        System.out.println(e);
    }
}
```

If this occurs, it looks something like this.

Lipstick(Red)

Add Stock

Remove Stock

Remarks

Lipstick(Red) needs RESTOCK! Only 8 left.

Back

Error Handling

```
try {  
  
    Connection conn = DriverManager.getConnection("jdbc:derby://localhost:1527/storeface","cc","cc");  
    System.out.println("Connected");  
    String itemx = jTextField2.getText();  
    String stockx = jTextField1.getText();  
    int stockxx = Integer.parseInt(stockx);  
  
    PreparedStatement st = conn.prepareStatement("Insert into misc (item, stock) values (?,?)");  
  
    st.setString(1, itemx);  
    st.setInt(2, stockxx);  
  
    int a = st.executeUpdate();  
  
    if (a>0) {  
        JOptionPane.showMessageDialog(null, "Data inserted to database.");  
        jTextField2.setText("");  
        jTextField1.setText("");  
    }  
}  
  
catch (Exception e) {  
    System.out.println(e);  
}
```

To detect errors throughout the program, Try Catch block was used frequently. This helps not only to detect errors but also helps to handle the error according to different error types.

Bibliography

Most of the codes I used in the program were taught by my computer science teacher during and out of class.

Some parts such as SQL codes, filling the combo box, etc. were obtained from:

- "Geeksforgeeks | A Computer Science Portal For Geeks". *Geeksforgeeks*, <https://www.geeksforgeeks.org/>.