Criterion C: Product Design

Table of Contents

Word Count: 1,071

1. Use of additional libraries

Several additional libraries were used in order to make this program. They include:

```
import java.awt.Color;
import java.awt.EventQueue;
import java.awt.Font;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.FileOutputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter
import java.util.ArrayList;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
```

```
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.SwingConstants;
import javax.swing.border.EmptyBorder;
import javax.swing.table.DefaultTableModel;
import com.itextpdf.text.BaseColor;
import com.itextpdf.text.Document;
import com.itextpdf.text.Element;
import com.itextpdf.text.FontFactory;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.Phrase;
import com.itextpdf.text.pdf.BaseFont;
import com.itextpdf.text.pdf.PdfPCell;
import com.itextpdf.text.pdf.PdfPTable;
import com.itextpdf.text.pdf.PdfWriter;
```

- java.awt.* - to change features of the containers (buttons, labels, etc)
- java.io.* - used for reading and writing data, and for this program, used to write to the pdf which is opened using java.io.File in DisplayTableInvoice class.
- java.sql.* - To establish connection to MySQL, retrieve, and interact with data from the database tables
- java.swing.* - Helps to provide and design the user interface (by adding components to the JFrame)
- java.util.* - Used for the collection framework (ArrayList) and date time facilities
- com.itextpdf.text.* - To make the PDF document and edit it

2. OOP Concepts

- Inheritance:

  In all classes, the main class extends/inherits the JFrame class, which is the inbuilt class. This means that all classes inherit the JFrame methods and variables, hence, we can use GUI features without any problems.

```
public class PrepareInvoice extends JFrame
```

The “extends” keyword after PrepareInvoice class shows that it inherits features of JFrame

- Polymorphism:

  Examples of polymorphism include Method Overriding, which is done in almost all classes wherever a mouse listener has been added. This is because of different parameters for the method addMouseListener().

```
toggleButton.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        setVisible(false);
        Clients c = new Clients();
        c.setVisible(true);
    }
    @Override
    public void mouseEntered(MouseEvent e) {
        toggleButton.setBackground(Color.GREEN);
    }
    @Override
    public void mouseExited(MouseEvent e) {
        toggleButton.setBackground(Color.WHITE);
    }
});
```

  Another example of polymorphism includes Method Overloading, where a method of a class has the same name, but different parameter types. Code for the same is shown below:

```
public class Function{
    Connection con = null;
    ResultSet rs = null;
    PreparedStatement pst = null;
    public ResultSet find(int s){
        try {
        con = DriverManager.getConnection("jdbc:mysql://localhost:3306/csiatables", "root", "mySQLpassword");
        pst = con.prepareStatement("SELECT * FROM productlist WHERE SerialNo = ?");
        pst.setInt(1, s);
        rs=pst.executeQuery();
        }
        catch(Exception e) {
            JOptionPane.showMessageDialog(null, e.getMessage());
        }
        return rs;
    }

    public ResultSet find(String s) {
        try {
        con = DriverManager.getConnection("jdbc:mysql://localhost:3306/csiatables", "root", "mySQLpassword");
        pst = con.prepareStatement("SELECT * FROM clientlist WHERE ClientID = ?");
        pst.setString(1, s);
        rs=pst.executeQuery();
        }
        catch(Exception e) {
            JOptionPane.showMessageDialog(null, e.getMessage());
        }
        return rs;
    }

}
```

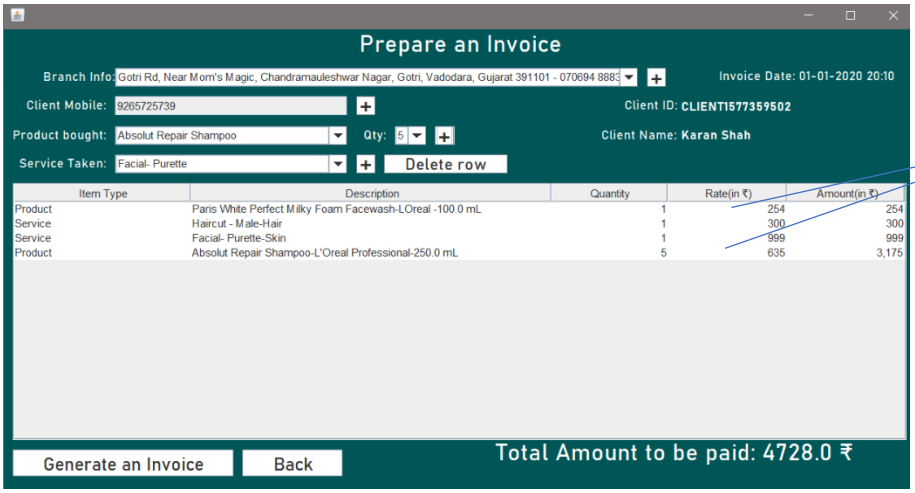
Same method name (find()) with different parameter types, i.e. string and integer

3. ArrayLists

- ArrayLists were used to store the information of all rows of JTable of preparing invoice menu into a list. They are dynamic in nature, and don't need an exact size, hence, it is the ideal data structure. This means that its size increases whenever more elements are added, and its size decreases as elements are removed.

```
ArrayList<RowDataService> sl = list;
Object rowData[] = new Object[5];
for (int i = 0; i < sl.size(); i ++) {
    rowData[0] = sl.get(i).itemtype;
    rowData[1] = sl.get(i).description;
    rowData[2] = sl.get(i).quantity;
    rowData[3] = sl.get(i).rate;
    rowData[4] = sl.get(i).amount;
    dtm.addRow(rowData);
}
```

```
ArrayList<RowDataProduct> sl = list;
Object rowData[] = new Object[5];
for (int i = 0; i < sl.size(); i ++) {
    rowData[0] = sl.get(i).itemtype;
    rowData[1] = sl.get(i).description;
    rowData[2] = sl.get(i).quantity;
    rowData[3] = sl.get(i).rate;
    rowData[4] = sl.get(i).amount;
    dtm.addRow(rowData);
}
```



These are rows in the list. They have been added using ArrayList.

4. File Handling

```
String path1="";
JFileChooser j = new JFileChooser();
j.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
int g = j.showSaveDialog(null);

if(g==JFileChooser.APPROVE_OPTION) {
    path1 = j.getSelectedFile().toString();
}
int time = (int) (System.currentTimeMillis()/1000);
path1 = path1.replace("\\", "\\\\");
String path = path1+"\\\\Invoice-"+cname.getText()+"-"+time+".pdf";
```

Name of the pdf file

- JFileChooser has been used to select location where the PDF has to be saved. The selected path is converted to string and is stored in the string variable “path1”. As this path has to be added into the database (for using this path to open the invoice from table), two backslashes in the path were important, and as backslash is an escape character, it is hidden when it is displayed. So, to have 2 backslashes in the output, 4 backslashes had to be added to the path. For this, “.replace(String x, String y) - where all x’s in the string are replaced with y” method was used. This replaced all the 2 backslashes in the path with 4 backslashes, and 2 of the backslashes weren’t displayed as they were escape characters, therefore, leaving the path with 2 backslashes.

```
Document doc = new Document();
try{
    if (table.getRowCount()!=0) {
        String FONT1 = "resources/fonts/PlayfairDisplay-Regular.ttf";
        com.itextpdf.text.Font f3 =FontFactory.getFont(FONT1, BaseFont.IDENTITY_H, BaseFont.EMBEDDED, 18);
        PdfWriter.getInstance(doc, new FileOutputStream(path));

        doc.open();
        Paragraph para = new Paragraph("BEAUTYMANNTRA INVOICE",
                FontFactory.getFont(FontFactory.HELVETICA_BOLD, 24,
                        Font.BOLD, BaseColor.CYAN.darker().darker().darker()));
        para.setAlignment(Element.ALIGN_CENTER);
        Paragraph para1 = new Paragraph("-----------------------------------------------------------"
                + "-------------------------------------------------------------------------");
        Paragraph para2 = new Paragraph("Date: " + formattedDate);
        para2.setAlignment(Element.ALIGN_RIGHT);
        Paragraph para3 = new Paragraph("-----------------------------------------------------------"
                + "-------------------------------------------------------------------------");
        Paragraph para4 = new Paragraph("Client ID: " + cnumber.getText());
        Paragraph para5 = new Paragraph("Client Name: " + cname.getText());
        Paragraph para6 = new Paragraph("-----------------------------------------------------------"
                + "-------------------------------------------------------------------------");
        doc.add(para); doc.add(para1); doc.add(para2);
        doc.add(para3); doc.add(para4); doc.add(para5); doc.add(para6);

        PdfPTable table1 = new PdfPTable(5);
        table1.setWidthPercentage(105);
        table1.setSpacingBefore(11f);
        table1.setSpacingAfter(11f);
```

FileOutputStream is used to write data to a file, and in this case, a new pdf with an instance of Document (doc) is created. Data will be written to this file, which will be created in the file path “path”, with the name “Invoice-…..pdf”

```
float[] colWidth= {2f, 2f, 2f, 2f, 2f};
table1.setWidths(colWidth);
PdfPCell c1 = new PdfPCell(new Phrase("Item Type"));
table1.addCell(c1);
PdfPCell c2 = new PdfPCell(new Phrase("Description"));
table1.addCell(c2);
PdfPCell c3 = new PdfPCell(new Phrase("Quantity"));
table1.addCell(c3);
PdfPCell c4 = new PdfPCell(new Phrase("Rate"));
table1.addCell(c4);
PdfPCell c5 = new PdfPCell(new Phrase("Amount"));
table1.addCell(c5);
for (int m = 0; m < table.getRowCount(); m ++) {
    String itype = table.getValueAt(m, 0).toString();
    String desc = table.getValueAt(m, 1).toString();
    String qty = table.getValueAt(m, 2).toString();
    String r = table.getValueAt(m, 3).toString();
    String amt = table.getValueAt(m, 4).toString();
    table1.addCell(itype); table1.addCell(desc); table1.addCell(qty);
    table1.addCell(r); table1.addCell(amt);
    }
doc.add(table1);
Paragraph para7 = new Paragraph("Total Amount: Rs. " + sum, f3);
para7.setAlignment(Element.ALIGN_RIGHT);
doc.add(para7);

Paragraph space = new Paragraph("------------------------------------------------------------"
        + "------------------------------------------------------------");
Paragraph branchinfo = new Paragraph(branchInfo.getSelectedItem().toString(),
        FontFactory.getFont(FontFactory.HELVETICA, 10, Font.BOLD, BaseColor.BLACK));
branchinfo.setAlignment(Element.ALIGN_CENTER);
doc.add(space);
doc.add(branchinfo);
doc.close();
```

Conversion of row data (objects) to String

5. Insertion, edit, deletion using MySQL queries

To insert data:

- First of all, the latest MySQL java connector had to be added to the class path. A connection to the database, which is secure as a local connection is created, has to be established. Then, a prepared statement has to be created. The first 2 steps are common for all SQL related functions in java. After that, an SQL query for INSERT INTO has to be declared.

```
try {
    Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/csiatables", "root", "mySQLpassword");
    String query = "INSERT INTO `csiatables`.`clientlist` (`ClientID`, `Name`, `MobileNumber`, `Email`, `Gender`, `Birthday`)"
            + " VALUES (?, ?, ?, ?, ?, ?)";
    pst = con.prepareStatement(query);
    pst.setString(1, id);
    if (clientName.getText().isEmpty() == false && clientContact.getText().isEmpty() == false) {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd");
        String date = sdf.format(bdayChoose.getDate());
        pst.setString(2, clientName.getText());
        pst.setString(3, clientContact.getText());
        pst.setString(4, clientEmail.getText().toLowerCase());
        pst.setString(5, gender);
        pst.setString(6, date);
    }
    pst.executeUpdate();
    JOptionPane.showMessageDialog(null, "Added Client Successfully");
    id = client + (System.currentTimeMillis()/1000);
    clientID.setText(id);
    clearContents();
}
catch(Exception e1) {
    JOptionPane.showMessageDialog(null, e1);
}
```

Changes any upper-case letter in the string to a lower-case letter as emails have no upper-case letters

This executes the query and adds data to table in the database

If a client has been successfully added, ID for the next client will automatically be generated and set to ID text field, which cannot be edited as the client ID is unique and shouldn't be added manually

To edit data:

```
String id1 = clientID.getText();
String name1 = clientName.getText();
String number1 = clientContact.getText();
String email1 = clientEmail.getText().toLowerCase();
char gender1 = 'M';
SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd");
String bday = sdf.format(bdayChoose.getDate());
if (radioButton.isSelected()) {
    gender1 = 'M';
}
else if (radioButton_1.isSelected()){
    gender1 = 'F';
}
String query = "update clientlist set Name = '" + name1 +"', MobileNumber = '" + number1 +"',"
        + " Email = '" + email1 +"', Gender = '" + gender1 + "', Birthday = '" + bday +"' "
                + "where ClientID = '" + id1 +"'";
try {
    con1 = DriverManager.getConnection("jdbc:mysql://localhost:3306/csiatables", "root", "mySQLpassword");
    PreparedStatement pst1 = con1.prepareStatement(query);
    if (name1.isEmpty()==false && number1.isEmpty()==false && email1.isEmpty()==false && bday.isEmpty()==false
            && (radioButton.isSelected()==true || radioButton_1.isSelected()==true)){
        pst1.execute();
        JOptionPane.showMessageDialog(null, "Client Edited Successfully!");
        textFieldEditable(false);
        clientID.setEditable(true);
        clearContents();
    }
    else {
        JOptionPane.showMessageDialog(null, "One or more fields are empty, please fill all the fields!");
    }
}
catch(Exception e1) {
    JOptionPane.showMessageDialog(null, e1);
}
```

MySQL update query

The UPDATE query will only be executed if this condition is satisfied, i.e. if no text fields are left empty

To delete data:

```
btnDeleteProduct.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String query = "DELETE FROM `csiatables`.`productlist` WHERE (`SerialNo` = ? )";
        try {
            Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/csiatables", "root", "mySQLpassword");
            pst = con.prepareStatement(query);
            pst.setInt(1, Integer.parseInt(serialNo.getText()));
            pst.executeUpdate();
            JOptionPane.showMessageDialog(null, "Product deleted successfully!");
            clearContents();
        }
        catch(Exception e1) {
            JOptionPane.showMessageDialog(null, e1);
        }
    }
});
```

MySQL DELETE query

Text from the text field is initially of string type, and serial number, in the database is of type integer. For this, data entered in the text field has to be converted to an integer, hence, Integer.parseInt() is used.

6. Selection and Sorting using SQL statements

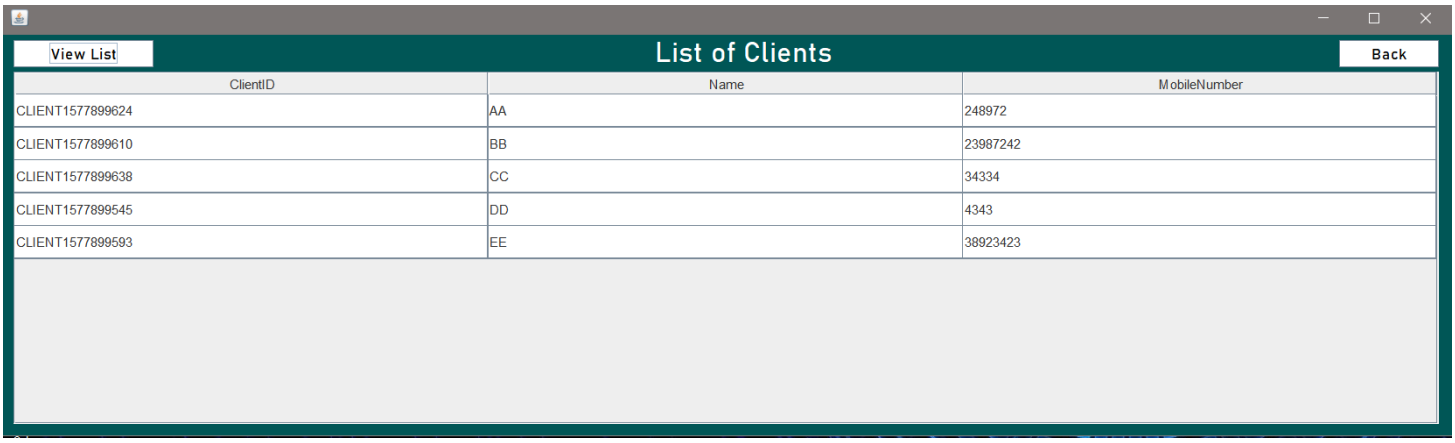
- The SELECT query was used to display list of clients, products, invoices, and services in JTable, using the external jar rs2xml.jar. To sort the displayed data, a query was written in the following way:

```
try {
    Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/csiatables", "root", "mySQLpassword");
    String query = "select * from clientlist order by Name ASC";
    pst = conn.prepareStatement(query);
    rs = pst.executeQuery();
    table.setModel(DbUtils.resultSetToTableModel(rs));
    tcm = table.getColumnModel();
    tcm.removeColumn(tcm.getColumn(5));
    tcm.removeColumn(tcm.getColumn(4));
    tcm.removeColumn(tcm.getColumn(3));
} catch (SQLException e1) {
    // TODO Auto-generated catch block
    e1.printStackTrace();
}
```

This sorts data from the client in the database alphabetically according to the client name (ascending order), and also displays the information correctly.

Using rs2xml.jar

Output after adding clients DD, EE, BB, AA, CC in this order is shown below:



ClientID	Name	MobileNumber
CLIENT1577899624	AA	248972
CLIENT1577899610	BB	23987242
CLIENT1577899638	CC	34334
CLIENT1577899545	DD	4343
CLIENT1577899593	EE	38923423

As it can be noticed, data has been sorted alphabetically by names. However, in the database, data is in the order it was entered.

ClientID	Name	MobileNumber	Email	Gender	Birthday
CLIENT1577899545	DD	4343	dd@gmail.com	M	2020/01/03
CLIENT1577899593	EE	38923423	ee@gmail.com	M	2020/01/03
CLIENT1577899610	BB	23987242	bb@gmail.com	M	2020/01/04
CLIENT1577899624	AA	248972	aa@gmail.com	M	2020/01/10
CLIENT1577899638	CC	34334	cc@gmail.com	M	2020/01/17
NULL	NULL	NULL	NULL	NULL	NULL

7. Exception Handling

- Exceptions were handled using the try and catch block and were used several times. It was used most frequently when establishing a connection to the MySQL database. For establishing this connection, a try and catch block had to be added. It was also used when the pdf had to be generated. Also, to display the exception message, JOptionPane.showMessageDialog() was used. The parameters for this method were (null, Exception e). This displayed the exception message in a message dialog. For example, to catch SQL exceptions:

```
} catch (SQLException e1) {
    JOptionPane.showMessageDialog(null, e1);
}
```

- Or to catch normal exceptions:

```
catch(Exception e1) {
    JOptionPane.showMessageDialog(null, e1);
}
```

8. Parameter passing

Throughout this program, different methods through which parameters were passed are:

- textFieldEditable(boolean)

```
public void textFieldEditable(boolean n) {
    serviceCat.setEditable(n);
    serviceSubCat.setEditable(n);
    serviceRate.setEditable(n);
}
```

- find (int/String) – depending on service/product/clients – returns result set. Code for the same is provided below:

```
public ResultSet find(int s) {
    try {
    con = DriverManager.getConnection("jdbc:mysql://localhost:3306/csiatables", "root", "mySQLpassword");
    pst = con.prepareStatement("SELECT * FROM servicelist WHERE SerialNo = ?");
    pst.setInt(1, s);
    rs=pst.executeQuery();
    }
    catch(Exception e) {
        JOptionPane.showMessageDialog(null, e.getMessage());
    }
    return rs;
}
```

- setComboButtonEditable(boolean) – in PrepareInvoice.java

```
public void setComboButtonEditable(boolean n) {
    serviceComboBox.setEnabled(n);
    prodComboBox.setEnabled(n);
    prodButton.setEnabled(n);
    servButton.setEnabled(n);
    comboBox_2.setEnabled(n);
    btnNewButton_1.setEnabled(n);
}
```

- addServiceRowToTable (ArrayList) and addProductRowToTable (ArrayList)

```
public void addServiceRowToTable(ArrayList<RowDataService> list) {
    DefaultTableModel dtm = (DefaultTableModel) table.getModel();
    ArrayList<RowDataService> sl = list;
    Object rowData[] = new Object[5];
    for (int i = 0; i < sl.size(); i ++) {
        rowData[0] = sl.get(i).itemtype;
        rowData[1] = sl.get(i).description;
        rowData[2] = sl.get(i).quantity;
        rowData[3] = sl.get(i).rate;
        rowData[4] = sl.get(i).amount;
        dtm.addRow(rowData);
    }
}

public void addProductRowToTable(ArrayList<RowDataProduct> list) {
    DefaultTableModel dtm = (DefaultTableModel) table.getModel();
    ArrayList<RowDataProduct> sl = list;
    Object rowData[] = new Object[5];
    for (int i = 0; i < sl.size(); i ++) {
        rowData[0] = sl.get(i).itemtype;
        rowData[1] = sl.get(i).description;
        rowData[2] = sl.get(i).quantity;
        rowData[3] = sl.get(i).rate;
        rowData[4] = sl.get(i).amount;
        dtm.addRow(rowData);
    }
}
```

9. Re-use of methods

Re-using of methods in a class decreases several lines of code. Screenshot of the code for same is shown below:

Example #1: setComboButton method

```
if(rs1.next()) {
    cname.setText(rs1.getString("Name"));    cnumber.setText(rs1.getString("ClientID"));
    clientNumber.setEditable(false);
    setComboButtonEditable(true);
}
```

```
btnNewButton_1.setFont(new Font("Bahnschrift", Font.PLAIN, 20));
btnNewButton_1.setBounds(10, 462, 243, 30);
contentPane.add(btnNewButton_1);

setComboButtonEditable(false);

JLabel lblQty = new JLabel("Qty:");
lblQty.setHorizontalAlignment(SwingConstants.RIGHT);
```

Used twice in the same class; cut several lines of code. The method "setComboButton" has been defined and the screenshot is shown on page 12

Example #2: textFieldEditable method

```
202         bdayChoose = new JDateChooser();
203         bdayChoose.setBounds(371, 151, 250, 23);
204         panel_1.add(bdayChoose);
205
206         textFieldEditable(false);
```

Used thrice in the same class, screenshot shown on page 9

```
319         textFieldEditable(true);
320         clientName.setText(rs.getString(name));
321         clientContact.setText(rs.getString(number));
322         clientEmail.setText(rs.getString(email));

23              textFieldEditable(false);
23              clientID.setEditable(true);
```

10. Use of 2-D Arrays

```
String[][] info ={  {"owner", "ownerpassword"},
                    {"manager", "managerpassword"}
                 };
    public void actionPerformed(ActionEvent e) {
        String user = textField.getText();
        String pass = passwordField.getText();
        for (int i = 0; i<2; i ++) {
            if (user.equals(info[i][0]) && pass.equals(info[i][1])) {
                if (info[i][0] == "owner") {
                    dispose();
                    GenRep mm = new GenRep();
                    mm.setVisible(true);
                }
                else if (info[i][0] == "manager") {
                    JOptionPane.showMessageDialog(null, "Reports can't be accessed by managers");
                }
                else {
                    JOptionPane.showMessageDialog(null, "Invalid username or password!");
                }
            }
            else if (user.length()==0 && pass.length() == 0){
                JOptionPane.showMessageDialog(null, "User fields have been left blank");
            }
            else if (user.length()==0) {
                JOptionPane.showMessageDialog(null, "Username field has been left blank!");
            }
            else if (pass.length() == 0) {
                JOptionPane.showMessageDialog(null, "Password field has been left blank!");
            }
        }
    }
});
```

2-D Array to store the username and password for the owner and manager.

Method performed when the login button is clicked. Scanning through the 2-D array to check if entered username and password is correct.

Message is displayed if either username or password is incorrect, or has been left blank.