

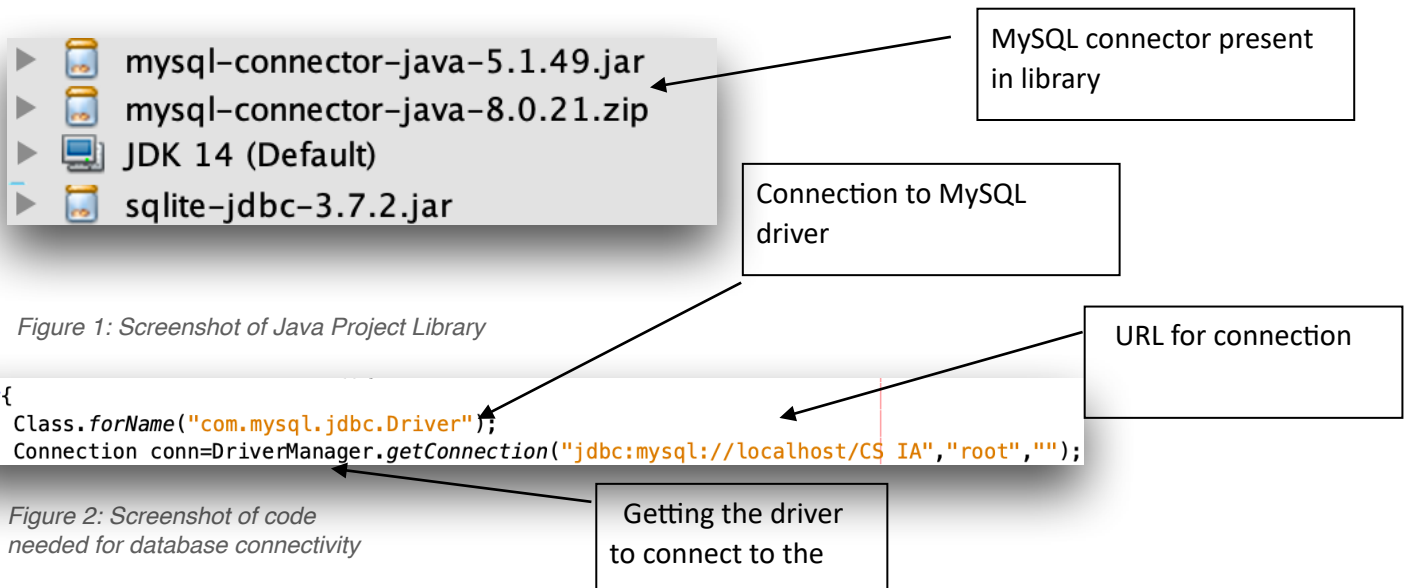
Criterion C: Development

List of techniques:

- NetBeans connectivity to database
- Graphical User Interface using Java Swing Library
- Authentication and Security
- Algorithms and Complex Queries
- Encapsulation
- Inheritance
- Event Listener Interface
- Loops
- Modular Code
- Data Abstraction
- Exception Handling
- Exporting data to PDF
- Auto-incrementing the ID number
- Records getting automatically entered in relation to ID number

Netbeans connectivity to database

We need a connector for the project to connect the Java NetBeans Project with the database so that they can communicate with each other.



The above code is required for establishing a connection between the Java NetBeans Project and the Database.

Graphical User Interface using Java Swing Library:

I used the Java Swing Library to create the GUI of my product as it offers a wide variety of options for customising the GUI. I used the components, coloured yellow in the diagram below, in my code.

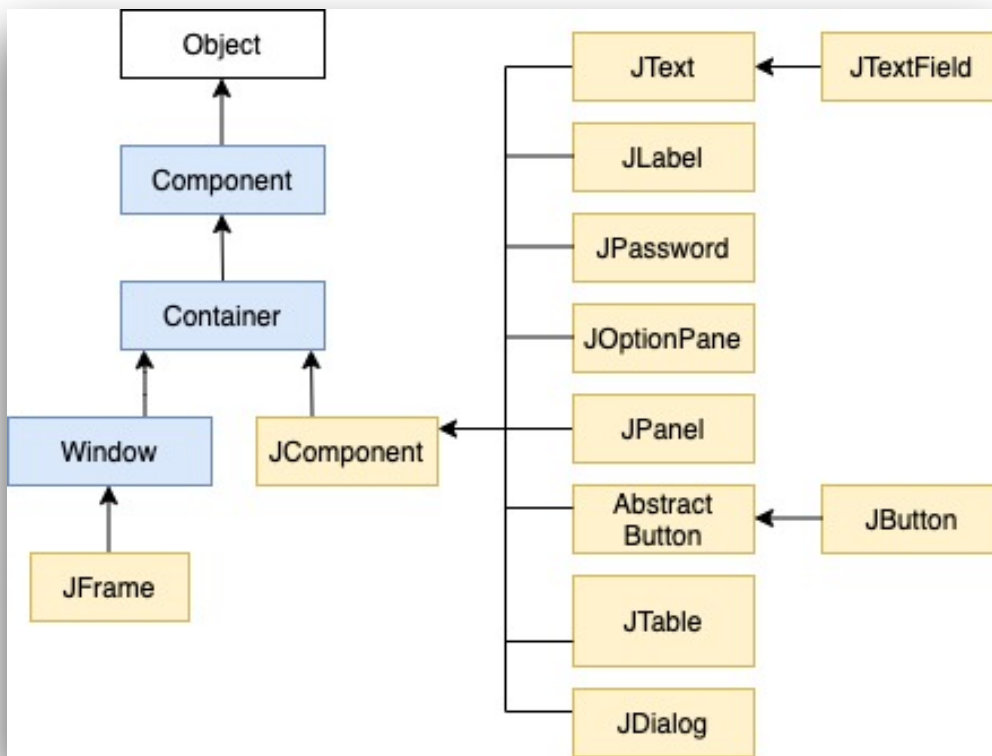


Figure 3: Screenshot of components used from Java Swing Library

By using the Java Swing Library, I have several tools at my disposal such as JFrame, JButtons, JPassword, JOptionPane etc, which allowed me to further improve the GUI functionality of the product.

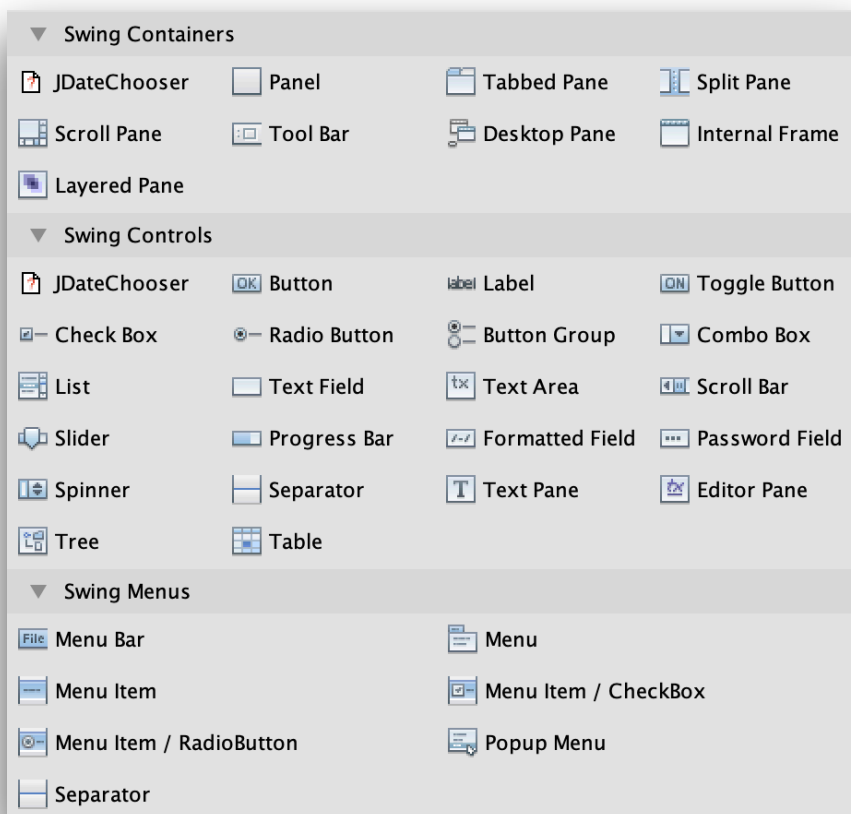


Figure 4: Screenshot of Java Swing Library with jDateChooser

Moreover, as I wanted one more functionality which was not available in the Java Swing Library I had to download it from the internet and add it to my project library and to the Java Swing Library also.

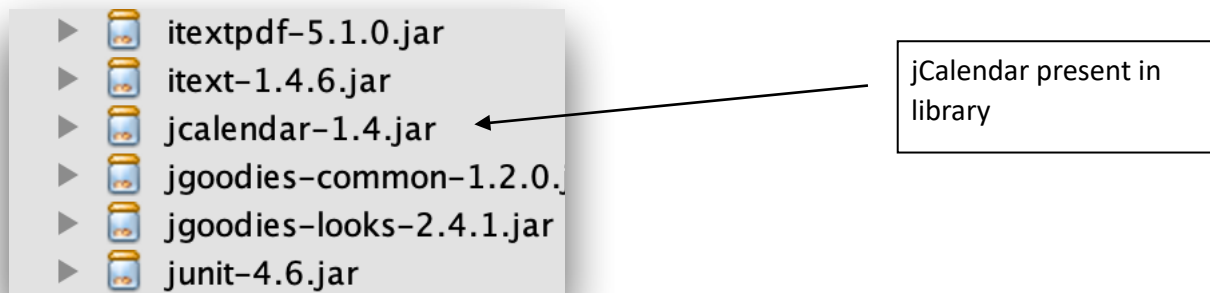


Figure 5: Screenshot of jCalendar in Project Library

After adding jCalendar to the project library, as I wanted the jCalendar component I added it to the Java Swing Library. jCalendar allows me to choose a date instead of manually typing the date.

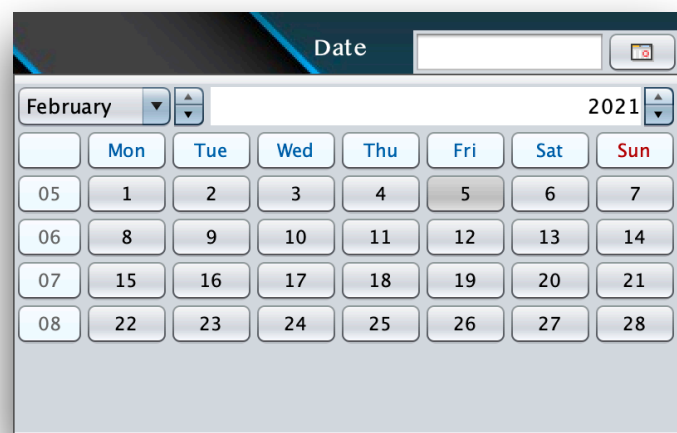


Figure 6: Screenshot of jCalendar

Authentication and Security

This is an important method to ensure data privacy for the client in order to prevent unauthorised access. The client wanted separate main menus for himself and for his employees to use. I wrote the following code to check if the username and password matches to either the Admin or Employee functionality, and if it does not match either, then it would display an error saying incorrect username and password entered.

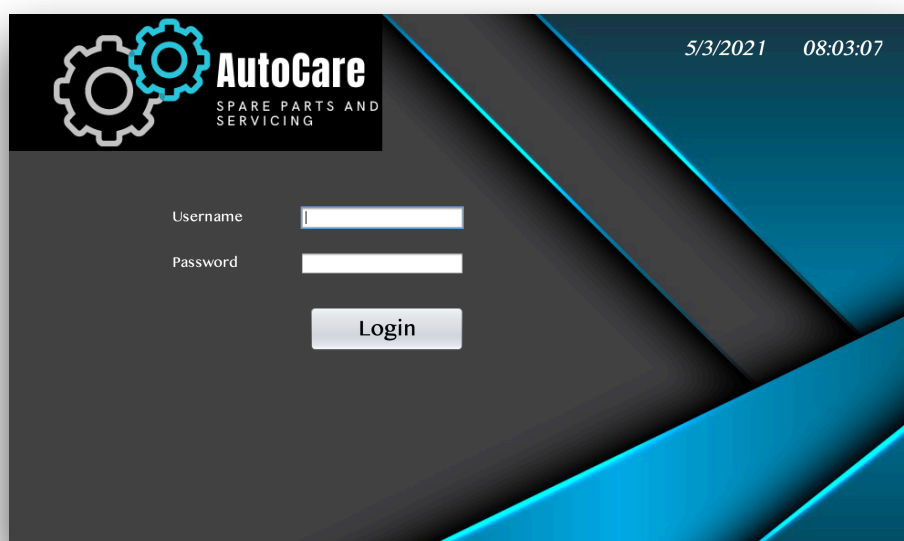


Figure 7: Screenshot of Login Form

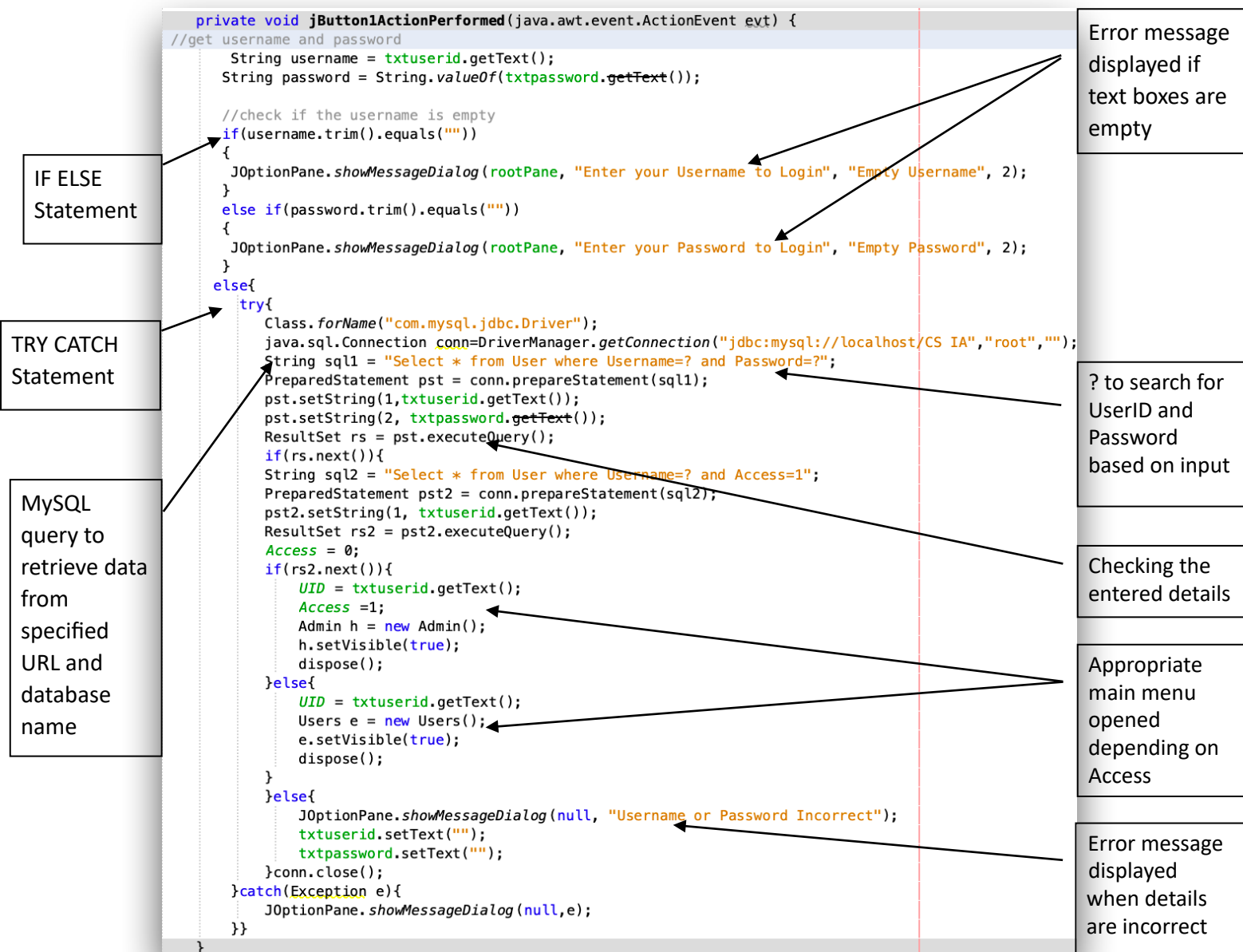


Figure 8: Screenshot of Login Button code

If the Username or Password aren't entered the error message 'Username or Password is empty' is displayed. If the Username or Password is incorrect the error message 'Incorrect Username or Password' is displayed. The Username and Password for each user is stored in the 'User' table along with the 'Access' rights. If the 'Access' right is 0 then the user is redirected to the 'EmployeeMainMenu', if the 'Access' right is 1 then the user is redirected to the 'AdminMainMenu' if the login is successful.

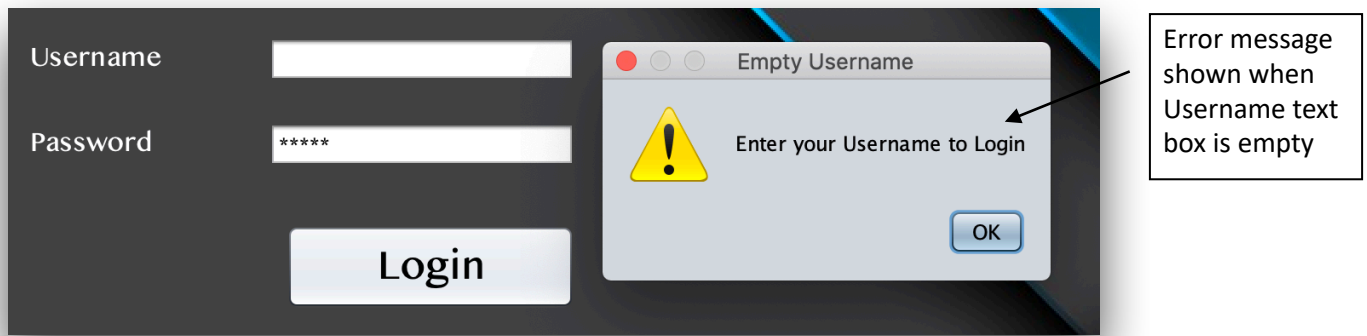


Figure 9 (a): Screenshot of Login Form when Username is not entered

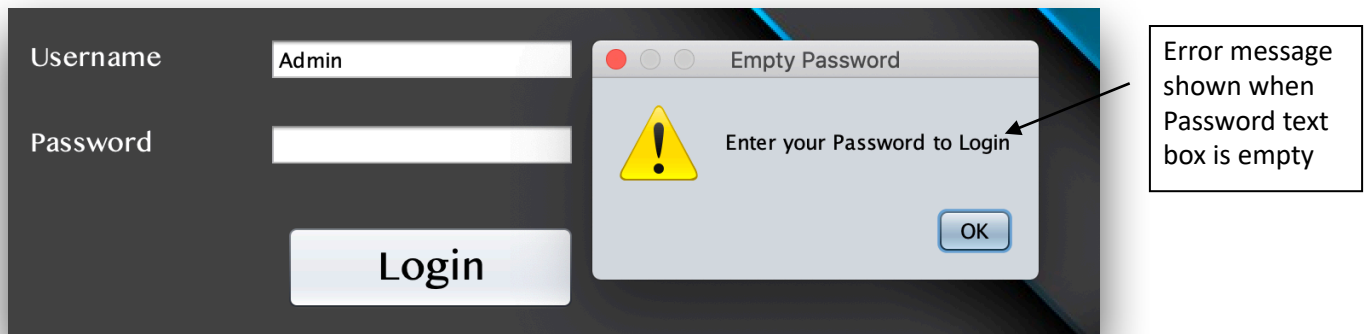


Figure 9 (b): Screenshot of Login Form when Password is not entered

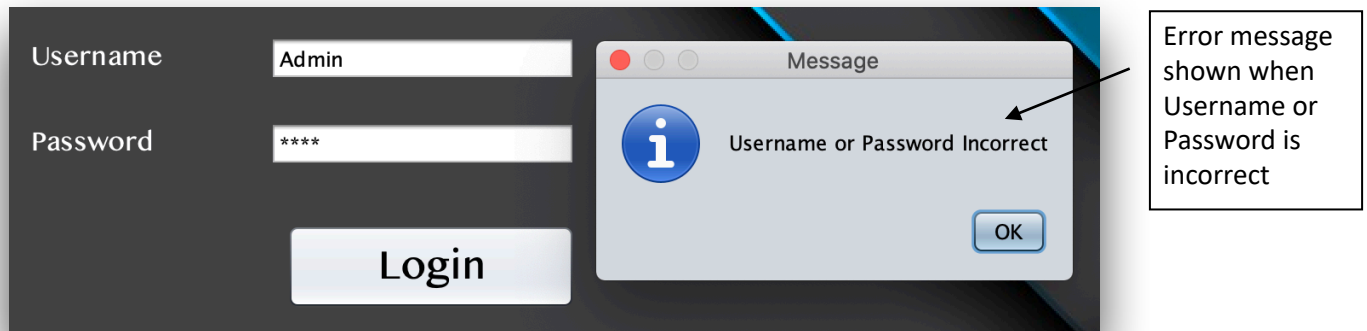


Figure 10: Screenshot of Login Form when Username or Password is incorrect

Users, Search and Bill Buttons
shown when user is an admin

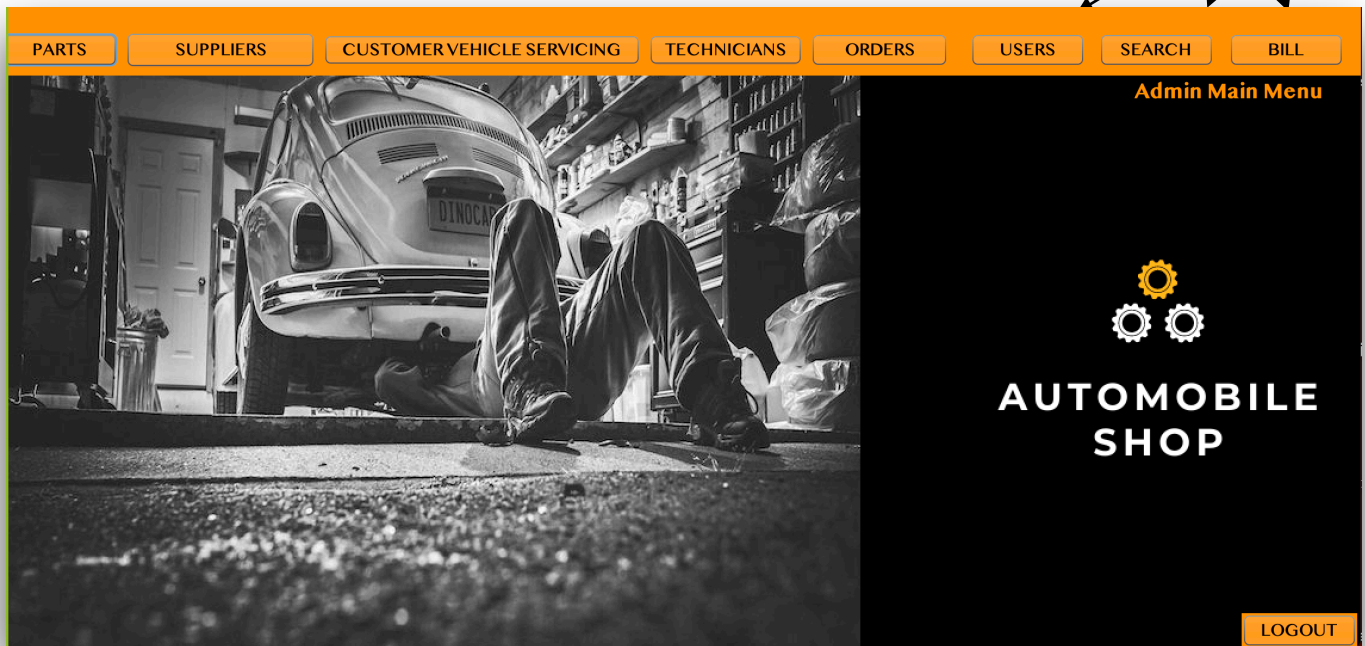


Figure 11: Screenshot of AdminMainMenu

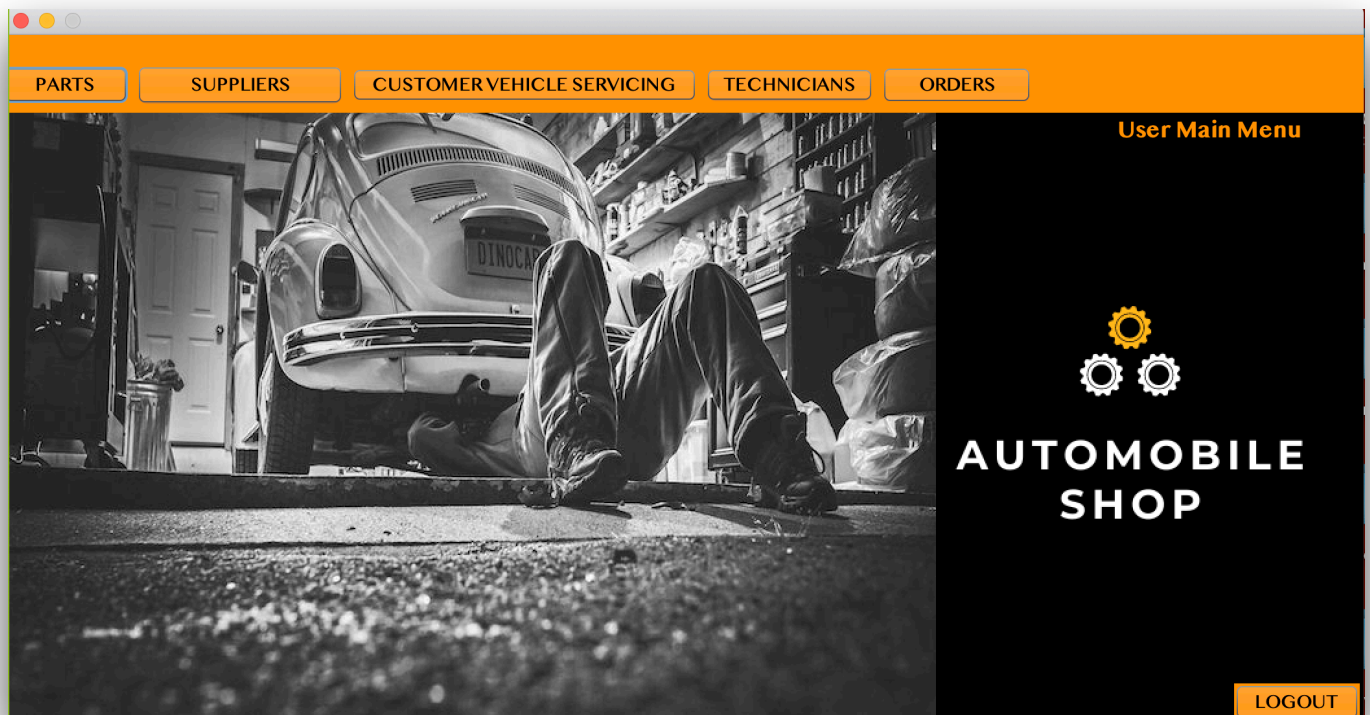


Figure 12: Screenshot of EmployeeMainMenu

Users, Search and Bill Buttons
are not shown when user is
an employee

Algorithms / Complex Queries

Having written the Pseudocode earlier in the Design phase, I was able to define and understand the important functions that I will be using in this project.

Database Connectivity Method: This is the first method that I used that is common to all classes as this involves connecting to the database, which is required for almost all of the classes. Thus, this has to be a public method so it can be called upon by any class to access the database.

```
package cs.ia;
import java.sql.*;
import javax.swing.*;

public class MySQLConnect {
    Connection conn=null;
    public static Connection ConnectDB(){
        try{
            Class.forName("com.mysql.jdbc.Driver");
            Connection conn=DriverManager.getConnection("jdbc:mysql://localhost/CS IA","root","");
            return conn;
        }catch(Exception e){
            JOptionPane.showMessageDialog(null, e);
            return null;
        }
    }
}
```

Figure 13: Screenshot of Database Connectivity Method

Add/Edit/Delete Data: These functions were the most important ones for the application. They can be called when they are needed for the adding, deleting and/or editing the data.

```
//Create a function to add a user
public boolean AddUser(String id, String name, String password)
{
    PreparedStatement st;
    ResultSet rs;
    String addQuery = "INSERT INTO `User`(`ID`, `Username`, `Password`) VALUES (?, ?, ?)";

    try {
        st = myconnection.createConnection().prepareStatement(addQuery);

        st.setString(1, id);
        st.setString(2, name);
        st.setString(3, password);

        return (st.executeUpdate()>0);
    } catch (SQLException ex) {
        Logger.getLogger(UserLoginDetailsClass.class.getName()).log(Level.SEVERE, null, ex);
        return false;
    }
}
```

Figure 14: Screenshot of method to add data

```

public boolean EditUser(String id, String name, String password)
{
    {

        PreparedStatement st;
        ResultSet rs;
        String editQuery = "UPDATE `User` SET `ID`=?,`UserName`=?,`Password`=? WHERE `ID`=?";

        try {
            st = myconnection.createConnection().prepareStatement(editQuery);

            st.setString(1, id);
            st.setString(2, name);
            st.setString(3, password);
            st.setString(4, id);

            return (st.executeUpdate()>0);

        } catch (SQLException ex) {
            Logger.getLogger(UserLoginDetailsClass.class.getName()).log(Level.SEVERE, null, ex);
            return false;
        }
    }
}

```

Figure 15: Screenshot of method to edit data

```

//Create a function to delete a user
public boolean DeleteUser(String id)
{
    {

        PreparedStatement st;
        ResultSet rs;
        String deleteQuery = "DELETE FROM `User` WHERE `ID`=?";

        try {
            st = myconnection.createConnection().prepareStatement(deleteQuery);

            st.setString(1, id);

            return (st.executeUpdate()>0);

        } catch (SQLException ex) {
            Logger.getLogger(PartsClass.class.getName()).log(Level.SEVERE, null, ex);
            return false;
        }
    }
}

```

Figure 16: Screenshot of method to delete data

Fill the Tables: This was created so that the admin could see the data present in all the forms. Each class has its own function and this function is beneficial for the viewing of all the records of different categories.

```
public void FillTable(JTable table)
{
    PreparedStatement ps;
    ResultSet rs;
    String selectQuery = "SELECT * FROM `User`";

    try {
        ps = myconnection.createConnection().prepareStatement(selectQuery);
        rs=ps.executeQuery();

        DefaultTableModel tableModel = (DefaultTableModel)table.getModel();

        Object[] row;

        while(rs.next())
        {
            row= new Object[3];
            row[0]=rs.getString(1);
            row[1]=rs.getString(2);
            row[2]=rs.getString(3);

            tableModel.addRow(row);
        }

    } catch (SQLException ex) {
        Logger.getLogger(PartsClass.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

WHILE Loop

Figure 17: Screenshot of function to fill tables

```
// display the selected row

DefaultTableModel model = (DefaultTableModel)jTable1.getModel();
int rIndex= jTable1.getSelectedRow();

//display data
aa.setText(model.getValueAt(rIndex,0).toString());
ab.setText(model.getValueAt(rIndex,1).toString());
ac.setText(model.getValueAt(rIndex,2).toString());
ad.setText(model.getValueAt(rIndex,3).toString());
ae.setText(model.getValueAt(rIndex,4).toString());
af.setText(model.getValueAt(rIndex,5).toString());
ag.setText(model.getValueAt(rIndex,6).toString());

}
```

Figure 18: Screenshot of code for data automatically shown in fields when a row is selected.

Inheritance:

I have used inheritance in my program so I can easily access methods from various classes in different aspects of my project. So to do this, I initially made one class with the code of the Database connectivity, called MyConnection, which acts as my Singleton class. By using inheritance, I made objects of this class all through the code, which allowed me to make the connection to the database without having to repeat it every time I need to access the Database. Furthermore, actions such as extracting entries from the database are common to many classes,

thus I have created classes of all the forms for the AdminMainMenu where I have defined the functions regarding adding a new object, editing object, deleting object etc.

All of the classes that have GUI's linked to them extend the JFrame, as follows:

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

If any other classes need to be extended, the object of the class has to be created as follows:

```
MY_CONNECTION myconnection = new MY_CONNECTION();  
String selectQuery = "SELECT * FROM `User` WHERE `Username`=? AND `Password`=?";
```

By using inheritance, I was able to save space, and increase time and memory efficiency.

Encapsulation:

As most variables are private, we need to use getter or setter methods in order to access or change the methods. The void set methods change their values, however, the get methods return the values of the variables. These are important methods as they allow us to access and change variables, without calling them directly.

```
try {  
    st = myconnection.createConnection().prepareStatement(editQuery);  
  
    st.setString(1, id);  
    st.setString(2, name);  
    st.setString(3, password);  
    st.setString(4, id);  
  
    return (st.executeUpdate()>0);  
  
} catch (SQLException ex) {  
    Logger.getLogger(UserLoginDetailsClass.class.getName()).log(Level.SEVERE, null, ex);  
    return false;  
}  
  
}
```

```

try {
    ps = myconnection.createConnection().prepareStatement(selectQuery);

    rs=ps.executeQuery();

    DefaultTableModel tableModel = (DefaultTableModel)table.getModel();

    Object[] row;

    while(rs.next())
    {
        row= new Object[3];
        row[0]=rs.getString(1);
        row[1]=rs.getString(2);
        row[2]=rs.getString(3);

        tableModel.addRow(row);
    }

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

```

Event Listener Interface:

To make the project user- friendly I've used to get the Event Listener Interface to show output on user's actions. In order to do this, I've used the ActionPerformed function in my buttons on the forms which makes it easier for the user to use the project, as it tells them if they have successfully added, updated or deleted data.

```

String id = jTextField1.getText();
String name= jTextField2.getText();
String password= jTextField3.getText();
String access= jTextField5.getText();

if( id.trim().equals("") || name.trim().equals("") || password.trim().equals("") || access.trim().equals(""))
{
    JOptionPane.showMessageDialog(rootPane, "Please Enter All Details ", "Empty Fields", JOptionPane.WARNING_MESSAGE);
}
else
{
    if( adduser.AddUser(id, name, password, access))
    {
        JOptionPane.showMessageDialog(rootPane, "New User Added Successfully", "Add User", JOptionPane.INFORMATION_MESSAGE);
        jTable1.setModel(new DefaultTableModel(null,new Object[]{"ID","Username","Password","Access"}));

        //populate the table
        adduser.FillTable(jTable1);
    }
    else
    {
        JOptionPane.showMessageDialog(rootPane, "New User Not Added. ", "Add User Error", JOptionPane.ERROR_MESSAGE);
    }
}
}
}

```

Figure 19: Screenshot of code to add data

```

    {
        //int id= Integer.valueOf(jTextFieldID.getText());
        String id = jTextField1.getText();
        String name= jTextField2.getText();
        String password= jTextField3.getText();
        String access= jTextField5.getText();

        {
            if( adduser.EditUser(id, name, password, access))
            {
                JOptionPane.showMessageDialog(rootPane, "User Updated Successfully", "Edit Parts", JOptionPane.INFORMATION_MESSAGE);
                jTable1.setModel(new DefaultTableModel(null,new Object[]{"ID","Username","Password","Access"}));

                //populate the table
                adduser.FillTable(jTable1);
            }
        }
    else
    {
        JOptionPane.showMessageDialog(rootPane, "User Not Updated, Please Enter All Details", "Edit User Error", JOptionPane.ERROR_MESSAGE);
    }
}
}
}

```

Figure 20: Screenshot of code to edit data

```

        int row = jTable1.getSelectedRow();
        if(row == -1)
    {
        JOptionPane.showMessageDialog(rootPane, "Please select a row to delete", "", JOptionPane.INFORMATION_MESSAGE);
    }

    else
    {
        int p = JOptionPane.showConfirmDialog(null, "Do you really want to delete the user?", "Delete",JOptionPane.YES_NO_OPTION);
        if (p==0) {
            String id= jTextField1.getText();

            if( adduser.DeleteUser(id))
            {
                JOptionPane.showMessageDialog(rootPane, "User Deleted Successfully", "Remove User", JOptionPane.INFORMATION_MESSAGE);
                jTable1.setModel(new DefaultTableModel(null,new Object[]{"ID","Username","Password","Access"}));

                //populate the table
                adduser.FillTable(jTable1);
            }
            else
            {
                JOptionPane.showMessageDialog(rootPane, "User Deleted", "", JOptionPane.INFORMATION_MESSAGE);
            }
        }
    }
    else
    {
        JOptionPane.showMessageDialog(rootPane, "User Not Deleted", "", JOptionPane.INFORMATION_MESSAGE);
    }
}

{
    jTextField1.setText("");
    jTextField2.setText("");
    jTextField3.setText("");
    jTextField5.setText("");
}
}
}

```

Figure 21: Screenshot of code to delete data

Loops:

I have used loops in my program to extract item entries from the database. Using loops, I have minimised the need to rewrite the same program multiple times, thus making my program efficient, which is the need of my client. **(Refer to Appendix Section A.1 and A.2)**

```
Document d=new Document();
PdfWriter.getInstance(d, new FileOutputStream("report.pdf"));
PdfPTable pt=new PdfPTable(colno);
d.open();
d.add(new Paragraph("Stock Report\n\n\n"));
for(int i=0;i<rowno;i++)
{
    pt.addCell(""+rs.getString(1));
    pt.addCell(""+rs.getString(2));
    pt.addCell(""+rs.getString(3));
    pt.addCell(""+rs.getString(4));
    pt.addCell(""+rs.getString(5));
    pt.addCell(""+rs.getString(6));
    pt.addCell(""+rs.getString(7));
    rs.next();
}
```

FOR Loop

Figure 22(a): Screenshot of FOR loop

```
public void FillTable(JTable table)
{
    PreparedStatement ps;
    ResultSet rs;
    String selectQuery = "SELECT * FROM `User`";

    try {
        ps = myconnection.createConnection().prepareStatement(selectQuery);
        rs=ps.executeQuery();

        DefaultTableModel tableModel = (DefaultTableModel)table.getModel();

        Object[] row;
        while(rs.next())
        {
            row= new Object[3];
            row[0]=rs.getString(1);
            row[1]=rs.getString(2);
            row[2]=rs.getString(3);

            tableModel.addRow(row);
        }

    } catch (SQLException ex) {
        Logger.getLogger(PartsClass.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

WHILE Loop

Figure 22(b): Screenshot of WHILE loop

Modular Code:

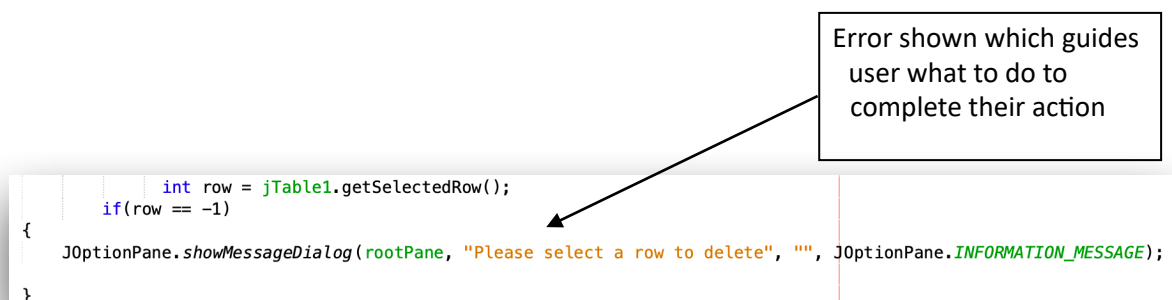
I have used techniques of Modular Code in order to keep the code reusable and maintainable. This technique is used to divide the code into separate manageable sub-functions, so that the same functions do not have to be defined in various parts of the code, making the code more efficient. This allows me more flexibility in my code as I am able to make changes throughout my code by simply changing small aspects in the code, which makes the code manageable and reusable. I have used this type of code in the classes of all the forms AdminMainMenu where I have created functions such as adding, editing, deleting entries in the database, taking the client back to the main menu without having to close each and every form, and I have used these functions throughout the project to be able to easily manage the code by changing only the functions in the classes and not having to define or change them throughout the rest of the program.

Data Abstraction:

Using Data Abstraction, I have been able to create restricted access for the employees, in which they cannot change information related to any of the forms whose information can be changed by the admin. As the employee is not allowed to edit information about any of the forms, the security that of the application is high as no external source can change the data, so they can keep track of all of the information without worrying that their data might be edited or deleted.

Exception Handling:

Exception handling is an essential part in my application. When the Client has to delete a record from any of the forms there is a chance that the desired record in the forms may not be selected, to prevent any crash to the program the IF statement here checks if the selected row count is less than 0 (which it won't be if a row is selected) and throws an error dialogue box with the response "Please select a row to delete", which also helps the user to understand the reason the record wasn't deleted.



Exporting Data to PDF:

I have used the iText.jar file in my program which has allowed me to export data from Parts into a PDF file. This functionality is extremely helpful and allows the Admin to easily manage data as well as back up their data into a safe location. I created a PDF file with the following method:

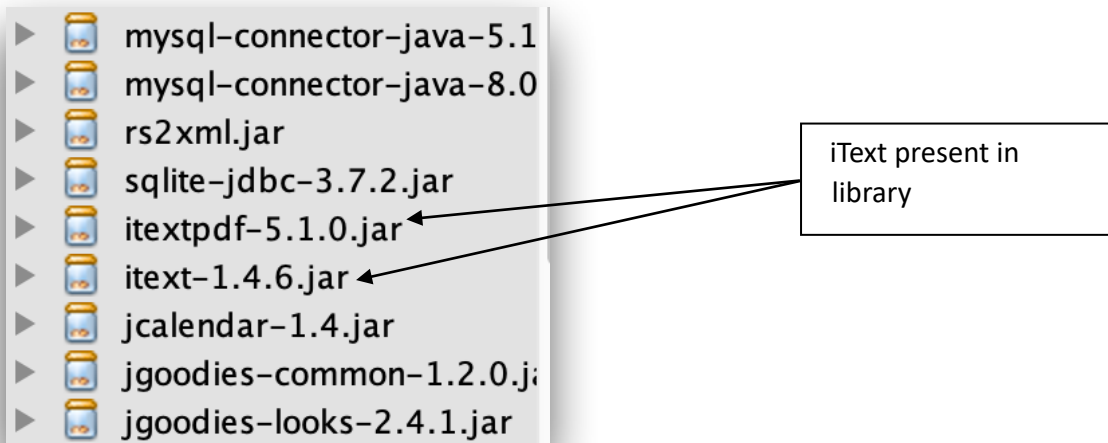


Figure 23: Screenshot of iText in Project Library

```
int rowno=0;
conn=MySqlConnection.ConnectDB();
//Statement st=conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
//ResultSet rs=st.executeQuery("select * from datadetails");
try{
    Statement st=conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
    ResultSet rs=st.executeQuery("select * from Parts");
    ResultSetMetaData rsmd=rs.getMetaData();
    int colno=rsmd.getColumnCount();

    while(rs.next())
    {
        rowno=rowno+1;
    }
    rs.first();

    //System.out.print(""+rowno);
    Document d=new Document();
    PdfWriter.getInstance(d, new FileOutputStream("report.pdf"));
    PdfPTable pt=new PdfPTable(colno);
    d.open();
    d.add(new Paragraph("Stock Report"));
    for(int i=0;i<rowno;i++)
    {
        pt.addCell(""+rs.getString(1));
        pt.addCell(""+rs.getString(2));
        pt.addCell(""+rs.getString(3));
        pt.addCell(""+rs.getString(4));
        pt.addCell(""+rs.getString(5));
        pt.addCell(""+rs.getString(6));
        pt.addCell(""+rs.getString(7));
        rs.next();
    }

    d.add(pt);
    d.close(); }catch(Exception e){
        JOptionPane.showMessageDialog(null,e);
    }
```

Figure 24: Screenshot of code to generate PDF

Auto-incrementing the ID number:

I have used this code to auto-increment ID in my application which has allowed me to not have to enter a new ID every time. This particular functionality is extremely helpful and allows the Admin to easily manage data. I created a this functionality with the following method:

```
String Sql="select MAX(ID) from Billing";
pst=conn.prepareStatement(Sql);
ResultSet rs = pst.executeQuery();
rs.next();

rs.getString("MAX(ID)");

if(rs.getString("MAX(ID)")== null)
{
    jTextField12.setText("E-0000001");
}
else {
    long id = Long.parseLong(rs.getString("MAX(ID)").substring(2,rs.getString("MAX(ID)").length()));
    id++;
    jTextField12.setText("E-"+String.format("%07d", id));
}
}
```

Figure 25: Screenshot of code to auto-increment ID

Records getting automatically entered in relation to ID number:

I have used this code so that records get automatically entered in relation to the ID which has allowed me to not have to enter the all data every time. This particular functionality is extremely helpful and allows the Admin to easily enter and manage data. I created this functionality with the following method:

```
String id = jTextField1.getText();
try
{
    conn=MySQLConnect.ConnectDB();
    Statement st = conn.createStatement();
    ResultSet rs = st.executeQuery("select * from Customers where ID like '"+id+"%");
    if (rs.next())
    {
        jTextField1.setText(rs.getString(1));
        jTextField2.setText(rs.getString(2));
        jTextField3.setText(rs.getString(3));
        jTextField4.setText(rs.getString(4));
    }
    else
    {
        jTextField2.setText("");
        jTextField3.setText("");
        jTextField4.setText("");
    }
}
catch (Exception e)
{
    JOptionPane.showMessageDialog(null, e);
}
}
```

Figure 26: Screenshot of code to get records automatically entered in relation to ID

Word count – 991 (excluding labels, screenshots and heading)

Bibliography – sources used for development

“Java Class, Methods, Instance Variables.” *w3resource*, www.w3resource.com/java-tutorial/java-class-methods-instance-variables.php.

“Java Database Connectivity with MySQL - Javatpoint.” *Www.javatpoint.com*, www.javatpoint.com/example-to-connect-to-the-mysql-database.

“Java Swing Tutorial - Javatpoint.” *Www.javatpoint.com*, www.javatpoint.com/java-swing.

“MySQL 8.0 Reference Manual :: 3.3.2 Creating a Table.” *MySQL*, dev.mysql.com/doc/refman/8.0/en/creating-tables.html.

NetBeans, Apache. *Downloading Apache NetBeans 12.2*, netbeans.apache.org/download/nb122/nb122.html.

“Java SE 15 - Downloads.” *Java SE Development Kit 15 - Downloads | Oracle India*, www.oracle.com/in/java/technologies/javase-jdk15-downloads.html.

“Apache Friends.” *Apache Friends RSS*, www.apachefriends.org/download.html.

“Connector/J 8.0.23.” *MySQL*, dev.mysql.com/downloads/connector/j/.

“Java Login Form Using NetBeans With Source Code - Part 2.” *YouTube*, YouTube, 28 Aug. 2016, www.youtube.com/watch?v=Uc7TXDG48Z4.

“How to Use JDateChooser or JCalendar in Java.” *YouTube*, YouTube, 9 Mar. 2018, www.youtube.com/watch?v=C0ZZpkliwVk.

“JCalendar.” *Toedtercom*, toedter.com/jcalendar/.