Criterion C: Development

Executable Jar File

Classes Used and Project Folder Structure

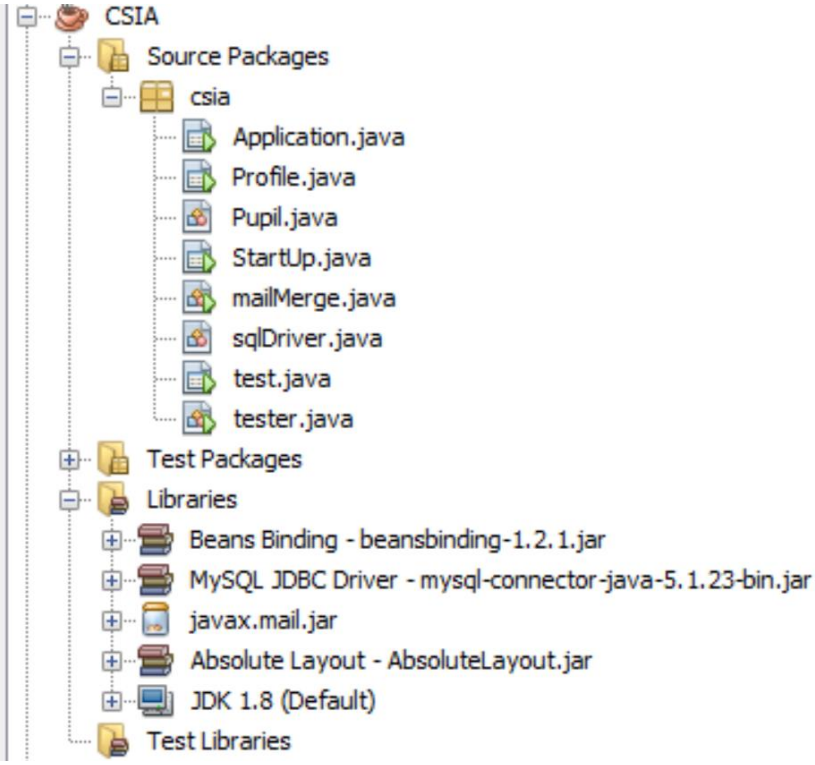

Figure 1. A visual display of all the classes and Add-ons being used in the project

List of Imports used throughout all classes

```
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.sql.*;
import java.util.logging.*;
import java.util.Locale;
import java.util.Date;
import javax.swing.table.DefaultTableModel;
import javax.swing.event.*;
import java.text.*;
import javax.mail.*;
import javax.mail.internet.*;
import java.util.*;
```

Figure 2. Imported Libraries

The first 2 imports are used to place "KeyListeners", these are used to minimalize invalid input. If there is a field where only integer values are meant to be added, yet the user incorrectly types text, these will catch those out. The code needs to be manually written into the JOptionPane accessories, such as text fields. Here is a sample code for a KeyListener for a payment field:

```
addPaySessField.addKeyListener(new KeyAdapter() {
    public void keyTyped(KeyEvent ev) {
        char c = ev.getKeyChar();
        if ( ((c < '0') || (c > '9')) && (c != KeyEvent.VK_BACK_SPACE)) {
            ev.consume();  // ignore event
        }
    }
});
```

Figure 3. KeyListener for "addPaySessField"

The next import for SQL is used to be able to integrate the features of the database with Java and to be able to write code and changes freely, this will be looked at in more depth. The "Util" and "Text" related imports are for used as utilities, either in the form of a more unusual Date variable, or to log and aid "try catch" methods in catching errors. The "Swing" imports play a role in the making of the GUI and finally the "Mail" imports are used to send automated emails as part of the program, this will also be covered later.

List of Techniques:

1. Arrays
2. Object Arrays
3. SQL
4. Static Methods and Variables
5. Encapsulation
6. Polymorphism
7. Using Imported Libraries
8. Swing GUI
9. MailMerge
10. Validation/Error Checking
11. Security Question
12. Opening and uploading documents

Integrated Development Environment(IDE)

The IDE used to develop the solution was Netbeans. Since Netbeans enables drag and drop creation of UI components, it made the solution efficient and less time consuming. However, at some points, such as for the KeyListeners, I did have to program components individually.

Connecting to MySQL database

This is done to store data for my program and calling the necessary update and save methods for the database is necessary for this. This was done using a separate sqlDriver class, to implement a further layer of abstraction. The accessing and saving of majority of the data is done through these methods:

```
public void access() throws ParseException {
    index = 0;
    String temp[] = new String[15];
    boolean tempe;
    Connection con = null;
    Statement st = null;
    ResultSet rs = null;
    ResultSet rt = null;
    String url = "jdbc:mysql://localhost:3306/csia";
    String user = "root";
    String password = "swords";
    int rowCount = 0;
    try {
        con = DriverManager.getConnection(url, user, password);
        st = con.createStatement();
        rt = st.executeQuery("select count(*) from Pupils");
        if (rt.next()) {
            rowCount = (rt.getInt(1));
        }
        rs = st.executeQuery("select * from Pupils");
        for (int count = 1; count <= rowCount; count++) {
            if (rs.absolute(count)) {
                for (int i = 2; i <= 16; i++) {
                    temp[i - 2] = rs.getString(i);
                }
            }
            if (temp[3].equals("Male")) {
                tempe = true;

            } else {
                tempe = false;
            }
            DateFormat format = new SimpleDateFormat("yyyy-MM-dd", Locale.ENGLISH);
            String datt = temp[2];
            java.util.Date dat = format.parse(datt);
            people[count - 1] = new Pupil(temp[0], Integer.parseInt(temp[1]), dat, tempe, temp[4], temp[5]);
            if (temp[6] != null) {
                people[count - 1].setEmerNum(temp[6]);
            }
            if (temp[7] != null) {
                if (temp[7].equals("Right")) {
                    tempe = true;
                } else {
                    tempe = false;
                }
                people[count - 1].setHand(tempe);
            }
            if (temp[8] != null) {
                datt = temp[8];
                dat = format.parse(datt);
                people[count - 1].setSessDate(dat);
            }
            if (temp[9] != null) {
                people[count - 1].setSessTime(Integer.parseInt(temp[9]));
            }
            if (temp[10] != null) {
                people[count - 1].setSessInfo(temp[10]);
            }
```

Code continued on next page

```
            if (temp[11] != null) {
                people[count - 1].setExp(temp[11]);
            }
            if (temp[12] != null) {
                people[count - 1].setMedical(temp[12]);
            }
            if (temp[13] != null) {
                people[count - 1].setPay(temp[13]);
            }
            if (temp[14] != null) {
                people[count - 1].setPaySess(Integer.parseInt(temp[14]));
            }
            index++;
        }
    } catch (SQLException ex) {
        Logger lgr = Logger.getLogger(tester.class.getName());
        lgr.log(Level.SEVERE, ex.getMessage(), ex);
    } finally {
        try {
            if (rs != null) {
                rs.close();
            }
            if (st != null) {
                st.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (SQLException ex) {
            Logger lgr = Logger.getLogger(tester.class.getName());
            lgr.log(Level.WARNING, ex.getMessage(), ex);
```

Figure 4. Access Method used to call data from SQL Database

```
public void reWrite() throws ParseException {
    Connection con = null;
    Statement st = null;
    ResultSet rs = null;
    String url = "jdbc:mysql://localhost:3306/csia";
    String user = "root";
    String password = "swords";
    try {
        con = DriverManager.getConnection(url, user, password);
        st = con.createStatement();
        st.executeUpdate("truncate Pupils");
        for (int count = 0; count < index; count++) {
            String gen = "", hand = "";
            java.util.Date utilDate = new SimpleDateFormat("EEE MMM d HH:mm:ss zzz yyyy").parse(people[count].getDOB() + "");
            java.sql.Date dateB = new java.sql.Date(utilDate.getTime());
            java.sql.Date dateS = null;
            if (people[count].getSessDate() != null) {
                java.util.Date utilDatea = new SimpleDateFormat("EEE MMM d HH:mm:ss zzz yyyy").parse(people[count].getSessDate() + "");
                dateS = new java.sql.Date(utilDatea.getTime());
            }
            if (people[count].getGender() == true) {
                gen = "Male";
            } else {
                gen = "Female";
            }
            if (people[count].getHand() == true) {
                hand = "Right";
            } else if (people[count].getHand() == false) {
                hand = "Left";
            }
            st.executeUpdate("insert into Pupils values(" + (count + 1) + ",'" + people[count].getName() + "'," + people[count].getAge() + ",'" + dateB + "','"
                    + gen + "','" + people[count].getEmail() + "','" + people[count].getPhoNum() + "','" + people[count].getEmerNum() + "','"
                    + hand + "','" + dateS + "'," + people[count].getSessTime() + ",'" + people[count].getSessInfo() + "','"
                    + people[count].getExp() + "','" + people[count].getMedical() + "','" + people[count].getPay() + "'," + people[count].getPaySess()
                    + ")");
        }
```

Code continued on next page

```
    } catch (SQLException ex) {
        Logger lgr = Logger.getLogger(tester.class.getName());
        lgr.log(Level.SEVERE, ex.getMessage(), ex);
    } finally {
        try {
            if (rs != null) {
                rs.close();
            }
            if (st != null) {
                st.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (SQLException ex) {
            Logger lgr = Logger.getLogger(tester.class.getName());
            lgr.log(Level.WARNING, ex.getMessage(), ex);
        }
    }
}
```

Figure 5. ReWrite Method used to update the database every time an action is done

Log in Screen

Figure 6. Shows the overall Login Screen for the Program

```
private void enterButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if (userField.getText().equals("")) {
        output("You must enter a username");
    } else if (passField.getText().equals("")) {
        output("You must enter a password");
    } else if (databaseCheck()) {
        appStart();
        if (safeCondition == true) {
            output("Welcome");
        } else {
            output("Welcome. Please set up a security option in the Personal Information tab as this will better "
                    + "secure your account");
        }
        this.dispose();
    } else {
        output("Invalid username or password");
    }
}
```

Figure 7. Actions taken to ensure that data entered is correct when the Submit Button is clicked. The method called to do this is in the following figure.

```
public boolean databaseCheck() {
    Connection con = null;
    Statement st = null;
    ResultSet rs = null;
    String url = "jdbc:mysql://localhost:3306/csia";
    String user = "root";
    String password = "swords";
    boolean check = false;
    try {
        con = DriverManager.getConnection(url, user, password);
        st = con.createStatement();
        rs = st.executeQuery("select * from Login");
        rs.absolute(1);
        if (rs.getString(1).equals(userField.getText()) && rs.getString(2).equals(passField.getText())) {
            check = true;
        }
    } catch (SQLException ex) {
        Logger lgr = Logger.getLogger(tester.class.getName());
        lgr.log(Level.SEVERE, ex.getMessage(), ex);
    } finally {
        try {
            if (rs != null) {
                rs.close();
            }
            if (st != null) {
                st.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (SQLException ex) {
            Logger lgr = Logger.getLogger(tester.class.getName());
            lgr.log(Level.WARNING, ex.getMessage(), ex);
        }
    }
    return check;
}
```

Figure 8. Shows the process of checking the database for the Login parameters to ensure that they are correct

```
String answer = "", email = "", sub = "", body = "";
Connection con = null;
Statement st = null;
ResultSet rs = null;
String url = "jdbc:mysql://localhost:3306/csia";
String user = "root";
String password = "swords";
try {
    con = DriverManager.getConnection(url, user, password);
    st = con.createStatement();
    rs = st.executeQuery("select * from Forgot");
    rs.absolute(1);
    safeQuestion = rs.getString(2);
    safeAnswer = rs.getString(3);
    answer = input("We'll get you your username and password immediately. Firstly, please answer your security question\n"
            + safeQuestion);
    if (answer.equalsIgnoreCase(safeAnswer)) {
        email = input("Correct. Now can you enter the email to which you wish to send your login details. Be careful to enter your own email.");
        sub = "Masterclass Database Login Details";
        body = "Your login credentials are as follows:\n ";
        rs = st.executeQuery("select * from Login");
        rs.absolute(1);
        body = body + "Username: " + rs.getString(1) + "\nPassword: " + rs.getString(2) + "\nThank you.";
        mailMerge.sendMail(email, sub, body);
    } else {
        output("This isn't the right answer. Please try again.");
    }
```

Figure 9. Shows the process that takes place when the Forgot My Password Button is pressed. A security question is asked, and if the answer is correct, then an email containing Login details is sent. I did not deem it necessary to show the SQL error logging code once again, as I have already shown it before.

We can see that even the Login class uses the mailMerge class in the case that the user has forgotten their password. Since the mailMerge class is used later as well, I thought it fit to display it here.

```
public class mailMerge {
    //Initialise some global variables which will be used throughout the program for finctions
    private static String email_my = "mcadatabase.noreply@gmail.com",
            password_my = "MasterClass",
            host_at = "smtp.gmail.com",
            port_num = "465";
    //Actual method used to send emails given the necessary parameters
    public static boolean sendMail(String strTo, String strSubject, String strBody) {
        Properties property = new Properties();
        property.put("mail.smtp.user", email_my);
        property.put("mail.smtp.host", host_at);
        property.put("mail.smtp.port", port_num);
        property.put("mail.smtp.starttls.enable", "true");
        property.put("mail.smtp.auth", "true");
        property.put("mail.smtp.socketFactory.port", port_num);
        property.put("mail.smtp.socketFactory.class", "javax.net.ssl.SSLSocketFactory");
        property.put("mail.smtp.socketFactory.fallback", "false");
        try {
            Session sess = Session.getDefaultInstance(property, null);
            try (Transport trans = sess.getTransport("smtp")) {
                MimeMessage message = new MimeMessage(sess);
                message.setText(strBody);
                message.setSubject(strSubject);
                message.setFrom(new InternetAddress(email_my));
                message.addRecipient(Message.RecipientType.TO, new InternetAddress(strTo));
                message.saveChanges();
                trans.connect(host_at, email_my, password_my);
                trans.sendMessage(message, message.getAllRecipients());
            }
            return true;
        } catch (Exception ex) {
            ex.printStackTrace();
```

Figure 10. shows the necessary parameters and methods that need to be called and used to send an email to any valid email.

This same mailMerge code is used throughout the program to send information.

How the Login details, such as the security question and Username and Password are set up, will be addressed later in the main program.

```
public void reboot() {
    try {
        access();
    } catch (ParseException ex) {
        Logger.getLogger(Application.class.getName()).log(Level.SEVERE, null, ex);
    }
    DefaultTableModel model = (DefaultTableModel) mainTable.getModel();
    DefaultTableModel mode = (DefaultTableModel) sessTable.getModel();
    Object[] row = new String[6];
    Object[] ro = new String[5];
    String temp;
    model.setNumRows(0);
    mode.setNumRows(0);
    int income = 0;
    for (int count = 0; count < index; count++) {
        row[0] = people[count].getName();
        row[1] = people[count].getAge() + "";
        if (people[count].getGender() == true) {
            temp = "Male";
        } else {
            temp = "Female";
        }
        row[2] = temp;
        SimpleDateFormat fDate = new SimpleDateFormat("dd/MM/yyyy");
        row[3] = (fDate.format(people[count].getSessDate()));
        row[4] = people[count].getEmail();
        row[5] = people[count].getPhoNum();
        model.addRow(row);
        ro[0] = people[count].getName();
        ro[1] = (fDate.format(people[count].getSessDate()));
        ro[2] = people[count].getSessTime() + "";
        ro[3] = people[count].getPaySess() + "";
        income = income + people[count].getPaySess();
        ro[4] = people[count].getPhoNum();
        mode.addRow(ro);
    }
    searchTextField.setText("");
    infoIncomeLabel.setText("Company Esitmated Income:      " + income);
```

Figure 11. Shows the code used to update the table itself rather than the objects or the SQL database. This uses methods from the Table library.

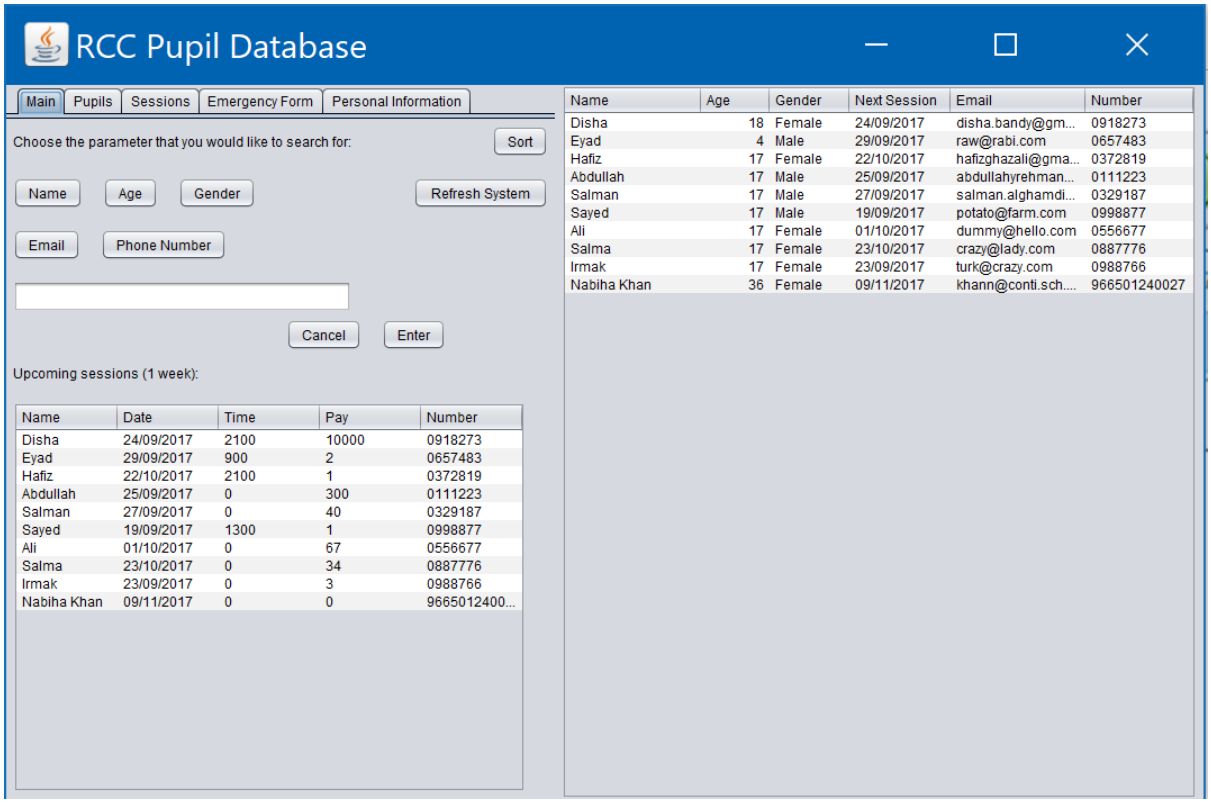

Figure 12. The main program starts up as follows, automatically loaded data into the two different tables, one general one and one for sessions. The "Sort" button only has one purpose, to show a pop up message explaining how the column headers of the tables have been programmed to sort it in ascending order. The search method requires you to choose a parameter and click Enter.

```
public void profileStart() {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Profile().setVisible(true);
        }
    });
}
```

```
mainTable.getSelectionModel().addListSelectionListener(new ListSelectionListener() {
    @Override
    public void valueChanged(ListSelectionEvent event) {
        Profile.param = mainTable.getValueAt(mainTable.getSelectedRow(), 0).toString();
        profileStart();
        reboot();
    }
});
```

Figure 13, 14. Used to make the table events clickable, this opens a second GUI for the corresponding object where the user can edit and save all the distinct fields. The same code is used to open files from the Documents tab then, using the ReWrite code displayed previously, this data can be updated and saved in the database. Moreover, here I implement polymorphism as I have to override the parent class function for the "valueChanged" method.

```
private void searchButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Connection con = null;
    Statement st = null;
    ResultSet rs = null;
    ResultSet rt = null;
    String url = "jdbc:mysql://localhost:3306/csia";
    String user = "root";
    String password = "swords";
    int rowCount = 0;
    Object[] row = new String[6];
    String temp[] = new String[15];
    DefaultTableModel model = (DefaultTableModel) mainTable.getModel();
    if (search.equals("")) {
        output("Please choose one of the above fields");
        nameToggleSearch.setVisible(true);
        ageToggleSearch.setVisible(true);
        genToggleSearch.setVisible(true);
        emailToggleSearch.setVisible(true);
        numToggleSearch.setVisible(true);
        nameToggleSearch.setSelected(false);
        ageToggleSearch.setSelected(false);
        genToggleSearch.setSelected(false);
        emailToggleSearch.setSelected(false);
        numToggleSearch.setSelected(false);
        searchTextField.setText("");
    }
    if (searchTextField.getText().equals("")) {
        output("You must enter a valid input");
        nameToggleSearch.setVisible(true);
        ageToggleSearch.setVisible(true);
        genToggleSearch.setVisible(true);
        emailToggleSearch.setVisible(true);
        numToggleSearch.setVisible(true);
        nameToggleSearch.setSelected(false);
        ageToggleSearch.setSelected(false);
        genToggleSearch.setSelected(false);
        emailToggleSearch.setSelected(false);
```

Code continued on next page

```
} else {
    model.setNumRows(0);
    try {
        con = DriverManager.getConnection(url, user, password);
        st = con.createStatement();
        rt = st.executeQuery("select count(*) from Pupils where " + search + "='" + searchTextField.getText() + "'");
        if (rt.next()) {
            rowCount = (rt.getInt(1));
        }
        if (rowCount < 1) {
            output("No entries match your search");
            nameToggleSearch.setVisible(true);
            ageToggleSearch.setVisible(true);
            genToggleSearch.setVisible(true);
            emailToggleSearch.setVisible(true);
            numToggleSearch.setVisible(true);
            nameToggleSearch.setSelected(false);
            ageToggleSearch.setSelected(false);
            genToggleSearch.setSelected(false);
            emailToggleSearch.setSelected(false);
            numToggleSearch.setSelected(false);
            searchTextField.setText("");
        } else {
            rs = st.executeQuery("select  * from Pupils where " + search + "='" + searchTextField.getText() + "'");
            for (int count = 1; count <= rowCount; count++) {
                if (rs.absolute(count)) {
                    for (int i = 2; i <= 16; i++) {
                        temp[i - 2] = rs.getString(i);
                    }
                }
                row[0] = temp[0];
                row[1] = temp[1];
                row[2] = temp[3];
                row[3] = temp[8];
                row[4] = temp[4];
                row[5] = temp[5];
                mainTable.removeAll();

                model.addRow(row);
            }
            output("Search successful. Click on any entries to view and edit the full pupil profile. To revert back to the "
                    + "full table, reboot the system");
            nameToggleSearch.setVisible(true);
            ageToggleSearch.setVisible(true);
            genToggleSearch.setVisible(true);
            emailToggleSearch.setVisible(true);
            numToggleSearch.setVisible(true);
            nameToggleSearch.setSelected(false);
            ageToggleSearch.setSelected(false);
            genToggleSearch.setSelected(false);
            emailToggleSearch.setSelected(false);
            numToggleSearch.setSelected(false);
            searchTextField.setText("");
    }
```

Figure 15. Code used for the search engine feature. Acknowledges which parameter is being searched for and then uses the database to do a search and find it, while also making sure the user hasn't made any errors in data entry.

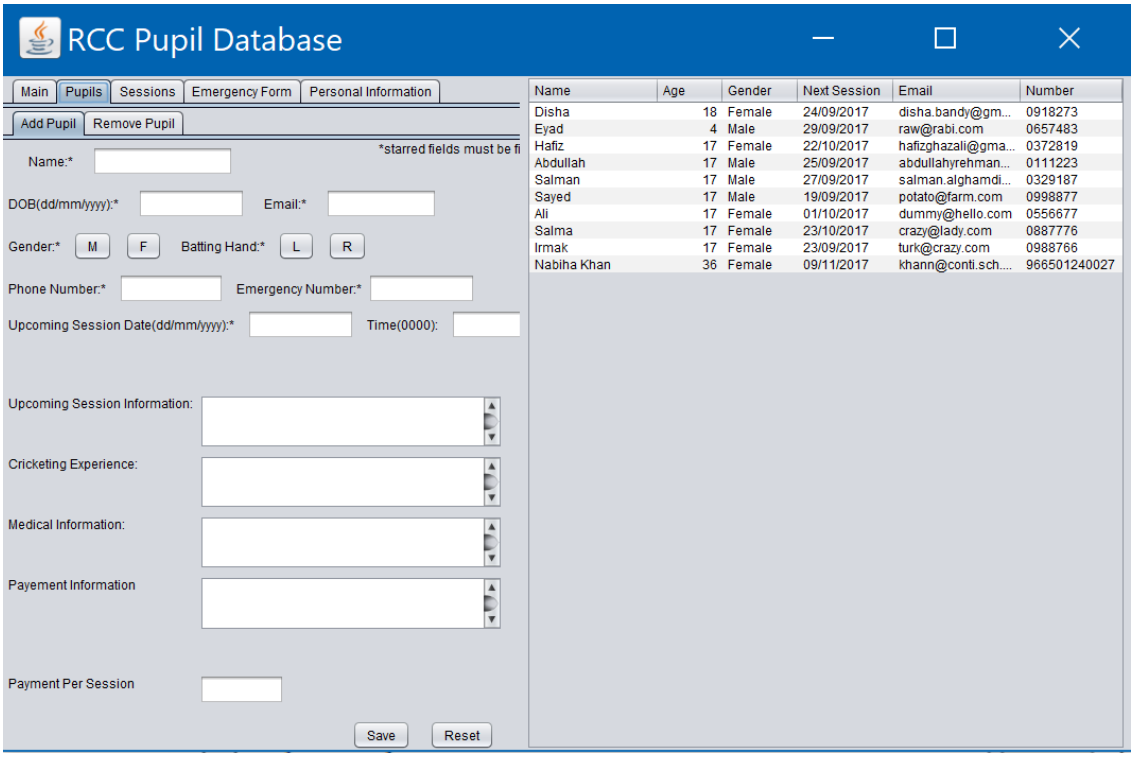

Figure 16. The second tab in the main program is to do with the addition and removal of pupils from the database.

```
private void addSaveButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    int minFlg = 0;
    Date date = new Date();
    DateFormat fDate = new SimpleDateFormat("dd/MM/yyyy", Locale.ENGLISH);
    if (addNameField.getText().equals("") || addDOBField.getText().equals("") || flag == 0
            || addEmailField.getText().equals("") || addPhoNumField.getText().equals("") || addSessDateField.getText().equals("")) {
        output("All starred fields must be filled");
        minFlg = 2;
    }
    for (int count = 0; count < index; count++) {
        if (people[count].getEmail().equals(addEmailField.getText()) || people[count].getPhoNum().equals(addPhoNumField.getText())) {
            minFlg = 1;
        }
    }
    if (minFlg == 1) {
        output("This pupil already exists. Please enter the correct values");
    }
    else if (minFlg == 0) {
        try {
            verifyDate(fDate.parse(addDOBField.getText()));
            long diff = Math.abs(date.getTime() - fDate.parse(addDOBField.getText()).getTime());
            long age = (diff / (86400000))/365;
            String agee = age + "";
            if(age>0){
                people[index] = new Pupil(addNameField.getText(), Integer.parseInt(agee), fDate.parse(addDOBField.getText()), gen,
                    addEmailField.getText(), addPhoNumField.getText());
            }
            else{
                output("The date value entered is incorrect. Please enter correct values");
            }
        } catch (ParseException ex) {
            Logger.getLogger(Application.class.getName()).log(Level.SEVERE, null, ex);
        }
        if (!addEmerNumField.getText().equals("")) {
            people[index].setEmerNum(addEmerNumField.getText());
        }
        if (flg == 0) {
```

```
            people[index].setHand(hand);
        }
        if (!addSessDateField.getText().equals("")) {
            try {
                verifyDate(fDate.parse(addSessDateField.getText()));
                people[index].setSessDate(fDate.parse(addSessDateField.getText()));
            } catch (ParseException ex) {
                Logger.getLogger(Profile.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
        if (!addTimeField.getText().equals("")) {
            people[index].setSessTime(Integer.parseInt(addTimeField.getText()));
        }
        if (!addSessInfoField.getText().equals("")) {
            people[index].setSessInfo(addSessInfoField.getText());
        }
        if (!addExpField.getText().equals("")) {
            people[index].setExp(addExpField.getText());
        }
        if (!addMedField.getText().equals("")) {
            people[index].setMedical(addMedField.getText());
        }
        if (!addPayField.getText().equals("")) {
            people[index].setPay(addPayField.getText());
        }
        if (!addPaySessField.getText().equals("")) {
            people[index].setPaySess(Integer.parseInt(addPaySessField.getText()));
        }
        index++;
        try {
            reWrite();
        } catch (ParseException ex) {
            Logger.getLogger(Application.class.getName()).log(Level.SEVERE, null, ex);
        }
        reboot();
        output("The Pupil has been added");
        addNameField.setText("");
```

Figure 17. The code used to check all parameters necessary are provided, and if so, the pupil object is made and added to the database, added to the object array, and to the table. Many verification checks are in place to make sure the data is valid and will not disrupt the workings of the program. One example is the verifyDate method. Methods from the Maths library are also used to auto calculate age.

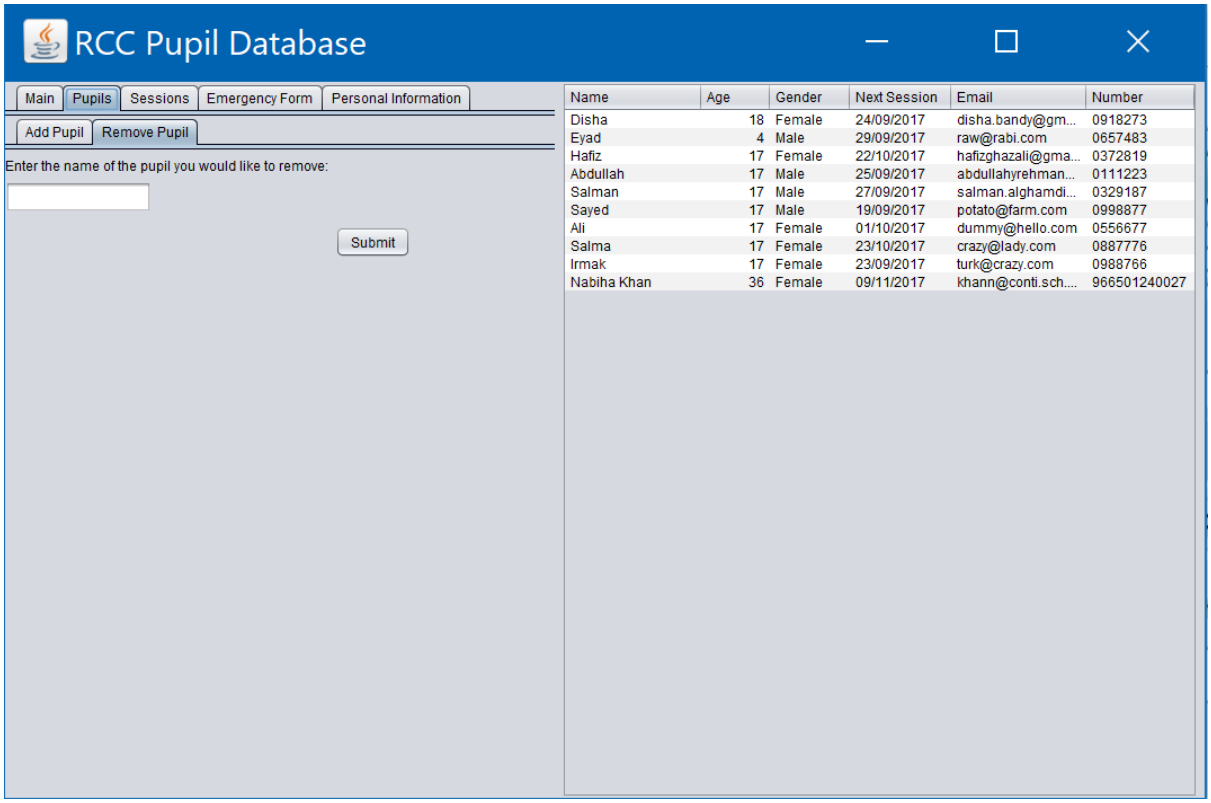

Figure 18. The second tab of the Pupils tab, used to remove pupils from the database. The user simply enters the name and the name gets deleted.

```
private void removeButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    int flag = 0;
    if (removeField.getText().equals("")) {
        output("Please enter a name");
        flag = 2;
    }
    String list = "";
    Connection con = null;
    Statement st = null;
    ResultSet rs = null;
    String url = "jdbc:mysql://localhost:3306/csia";
    String user = "root";
    String password = "swords";
    for (int count = 0; count < index; count++) {
        if (people[count].getName().equals(removeField.getText())) {
            flag = 1;
            try {
                con = DriverManager.getConnection(url, user, password);
                st = con.createStatement();
                st.executeUpdate("delete from Pupils where Name='" + removeField.getText() + "'");
                output("The pupil has been deleted");
                removeField.setText("");
                reboot();
            } catch (SQLException ex) {
                Logger lgr = Logger.getLogger(tester.class.getName());
                lgr.log(Level.SEVERE, ex.getMessage(), ex);
            } finally {
                try {
                    if (rs != null) {
                        rs.close();
                    }
                    if (st != null) {
                        st.close();
                    }
                    if (con != null) {
                        con.close();
                    }

                } catch (SQLException ex) {
                    Logger lgr = Logger.getLogger(tester.class.getName());
                    lgr.log(Level.WARNING, ex.getMessage(), ex);
                }
            }
        }

    }
    if (flag == 0) {
        output("This person does not exist in the database. Please enter another name");
        reboot();
        removeField.setText("");
    }
}
```

Figure 19. The code used to check if the name is in the database through the use of the object array, as is done throughout the program. If it is, then the database is updated and the pupil is deleted.

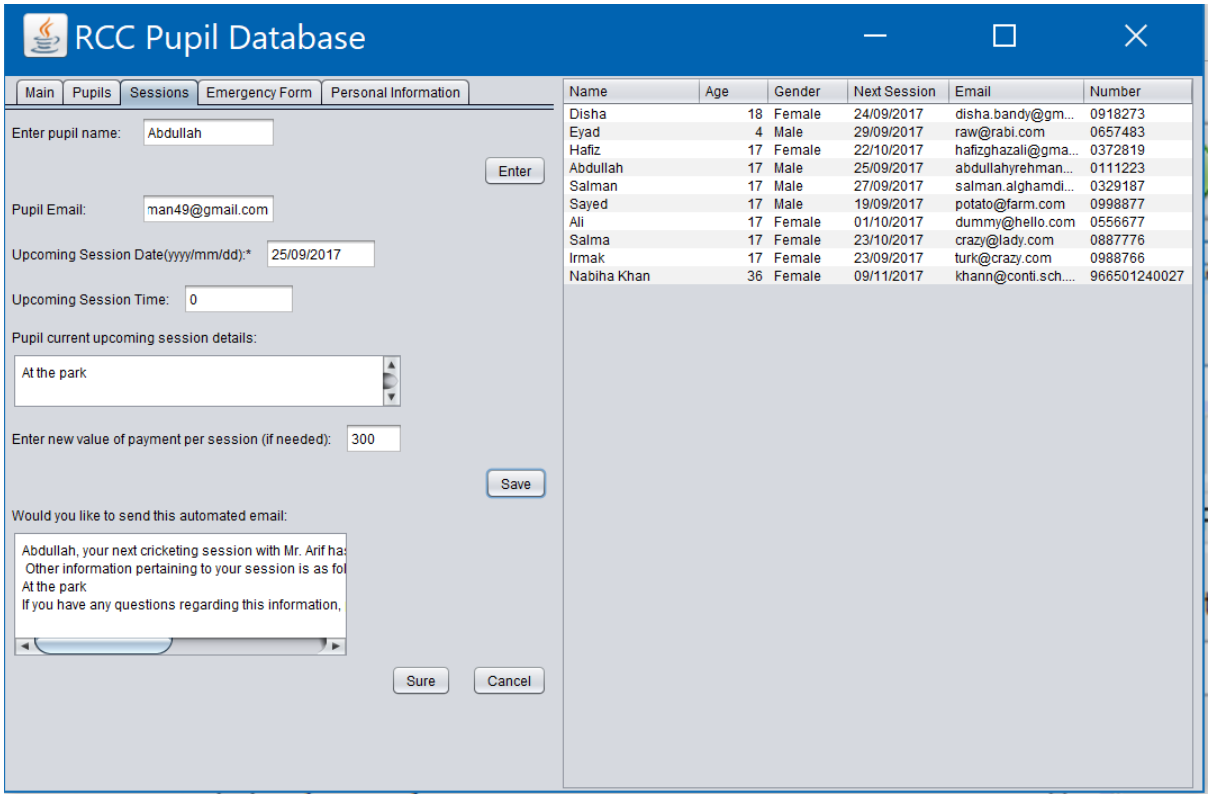

Figure 20. This is the sessions tab, used to manage the sessions for every pupil, the user enters the name and then the data is loaded automatically. The data can then be changed and updated by pressing the save button. Upon this change, the user will get an automated email, which he can then send to the corresponding pupil.

```
private void sessEnterButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    int flag = 0;
    for (int count = 0; count < index; count++) {
        if (people[count].getName().equals(sessNameField.getText())) {
            sessEmailField.setText(people[count].getEmail());
            SimpleDateFormat fDate = new SimpleDateFormat("dd/MM/yyyy");
            sessDateField.setText(fDate.format(people[count].getSessDate()));
            sessTimeField.setText(people[count].getSessTime() + "");
            if (people[count].getSessInfo().equals(null)) {
                sessInfoField.setText("");
            } else {
                sessInfoField.setText(people[count].getSessInfo());
            }
            sessPayField.setText(people[count].getPaySess() + "");
            flag = 1;
            break;
        }
    }
    if (flag == 0) {
        output("This pupil does not exist. Please enter another name.");
        sessNameField.setText("");
        sessEmailField.setText("");
        sessDateField.setText("");
        sessTimeField.setText("");
        sessInfoField.setText("");
        sessPayField.setText("");
        sessSendField.setText("");
    }

}
```

Figure 21. The first button in the process that loads the data into the following fields, again using the objects and object array.

```
private void sessSaveButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    int flag = 0;
    DateFormat fDate = new SimpleDateFormat("dd/MM/yyyy", Locale.ENGLISH);
    if (sessDateField.getText().equals("")) {
        output("All starred field must be filled.");
    }
    for (int count = 0; count < index; count++) {
        if (people[count].getName().equals(sessNameField.getText())) {
            try {
                verifyDate(fDate.parse(sessDateField.getText()));
                people[count].setSessDate(fDate.parse(sessDateField.getText()));
            } catch (ParseException ex) {
                Logger.getLogger(Application.class.getName()).log(Level.SEVERE, null, ex);
            }
            people[count].setSessTime(Integer.parseInt(sessTimeField.getText()));
            people[count].setSessInfo(sessInfoField.getText());
            people[count].setPaySess(Integer.parseInt(sessPayField.getText()));
            flag = 1;
            break;
        }
    }
    if (flag == 0) {
        output("This pupil does not exist. Please enter another name.");
    }
    sessSendField.setVisible(true);
    sessSureButton.setVisible(true);
    sessCancelButton.setVisible(true);
    sessSendField.setText(sessNameField.getText() + ", your next cricketing session with Mr. Arif has been set for " + sessDateField.getText()
            + " at " + sessTimeField.getText() + ".\n Other information pertaining to your session is as follows:\n" + sessInfoField.getText()
            + "\nIf you have any questions regarding this information, please contact Mr. Arif through his private email. Thank you.");
    try {
        reWrite();
    } catch (ParseException ex) {
        Logger.getLogger(Application.class.getName()).log(Level.SEVERE, null, ex);
    }
    reboot();
}
```

Figure 22. This saves the updated data and also keeps some checks in place to make sure that it is correct and relevant. Then it loads a message into the email field so that the user can make any changes and then send the email.

```
private void sessSureButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String email = sessEmailField.getText();
    String sub = "Master-Class Cricket Academy Session Update";
    String body = sessSendField.getText();
    mailMerge.sendMail(email, sub, body);
    output("The email has been sent");
}
```

Figure 23. Uses the mailMerge class to send the email from the host to the recipient.

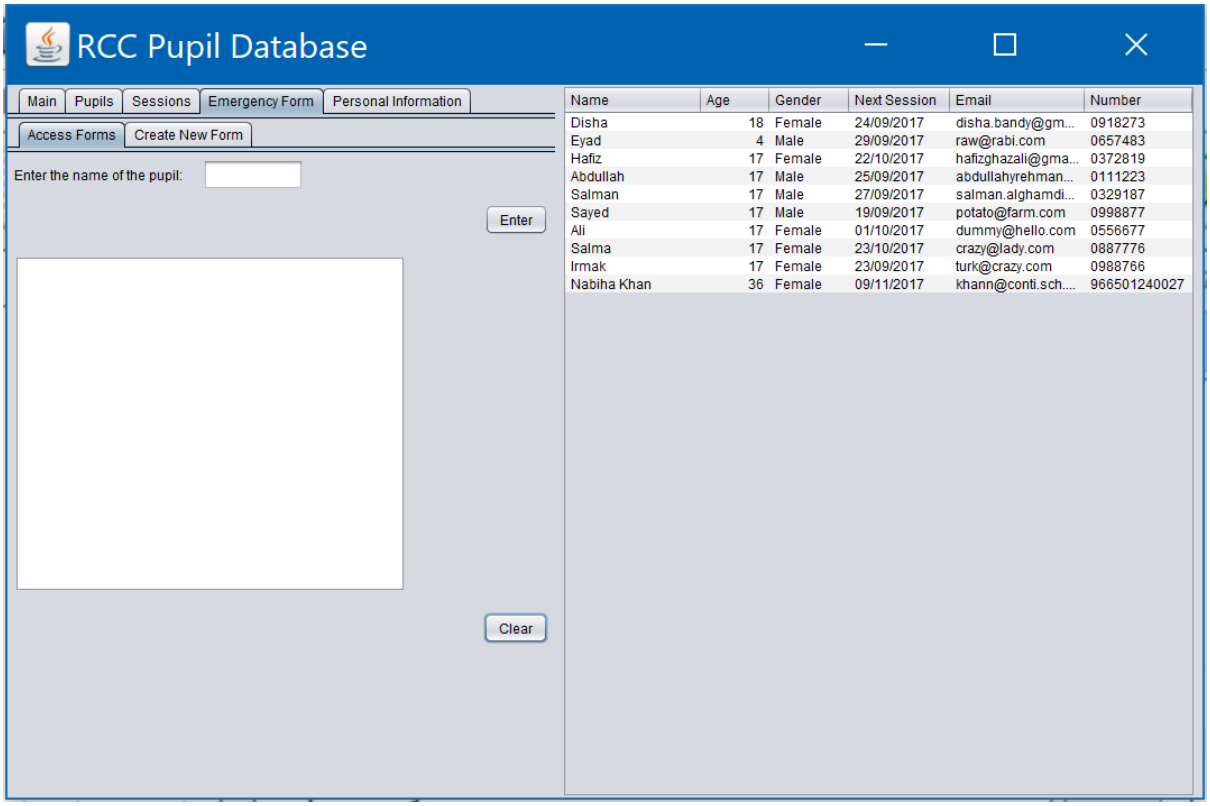

Figure 24. This is the 4th tab in the program, it enables the user to access previously written emergency forms. The user enters a name and all previous forms are automatically loaded.

```
private void formsEnterButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    int flag = 0;
    if (accessNameField.getText().equals("")) {
        output("Please enter a name in the field");
    }
    for (int count = 0; count < index; count++) {
        if (people[count].getName().equals(accessNameField.getText())) {
            flag = 1;
        }
    }
    if (flag == 1) {
        Connection con = null;
        Statement st = null;
        ResultSet rs = null;
        String url = "jdbc:mysql://localhost:3306/csia";
        String user = "root";
        String password = "swords";
        try {
            con = DriverManager.getConnection(url, user, password);
            st = con.createStatement();
            rs = st.executeQuery("select * from forms where name='" + accessNameField.getText() + "'");
            if (rs.absolute(1)) {
                accessFormField.setText(rs.getString(2));
            }
            if (accessFormField.getText().equals("")) {
                output("This pupil does not have any previous forms");
            }
        } catch (SQLException ex) {
            Logger lgr = Logger.getLogger(tester.class.getName());
            lgr.log(Level.SEVERE, ex.getMessage(), ex);
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (st != null) {
                    st.close();
                }
                if (con != null) {
                    con.close();
                }
            } catch (SQLException ex) {
                Logger lgr = Logger.getLogger(tester.class.getName());
                lgr.log(Level.WARNING, ex.getMessage(), ex);
            }
        }
    } else {
        output("This name was not found in the database. Please enter a valid name");
        accessNameField.setText("");
    }
}
```

Figure 25. The code used to load all the forms from the database, given a relevant correct name.

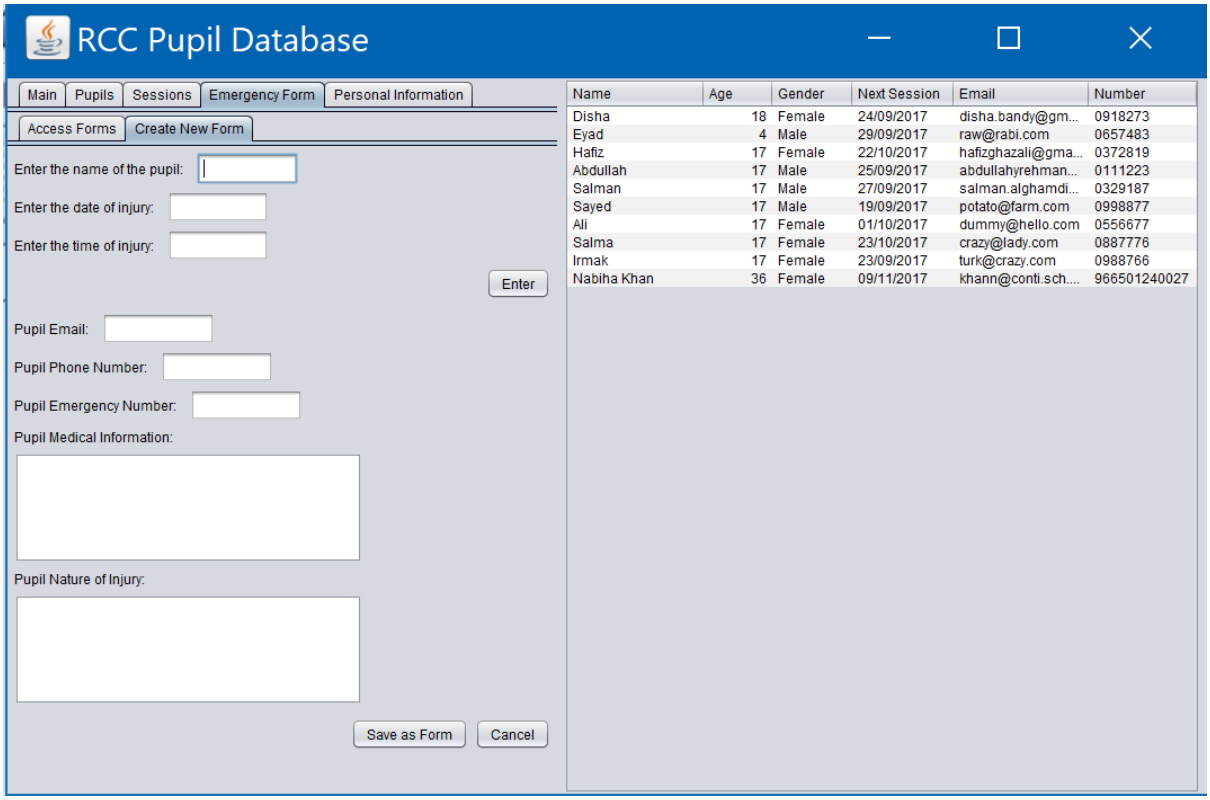

Figure 26. The user enters relevant details to the injury itself and subsequent data is loaded and can then be altered. Followingly the data can be saved as one big emergency form.

```
private void formNameButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    int flag = 0;
    for (int count = 0; count < index; count++) {
        if (people[count].getName().equals(formNameField.getText())) {
            formEmailField.setText(people[count].getEmail());
            formPhoField.setText(people[count].getPhoNum());
            formEmerField.setText(people[count].getEmerNum());
            formMedField.setText(people[count].getMedical());
            flag = 1;
            break;
        }
    }
    if (flag == 0) {
        output("This pupil does not exist. Please enter another name");
    }
}
```

Figure 27. This code is used to verify the name and then load the data in the following fields.

```
private void formSaveButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    DateFormat fDate = new SimpleDateFormat("dd/MM/yyyy", Locale.ENGLISH);
    try {
        verifyDate(fDate.parse(formDateField.getText()));
    } catch (ParseException ex) {
        Logger.getLogger(Application.class.getName()).log(Level.SEVERE, null, ex);
    }
    String form = "Date: " + formDateField.getText() + "\n" + formNameField.getText() + " recieved an injury at " + formTimeField.getText() + "\n."
            + "Contact Information:\nPupil Email: " + formEmailField.getText() + "\nPupil Phone Number: " + formPhoField.getText()
            + "\nPupil Emergency Number: " + formEmerField.getText() + "\nMedical Information:\n" + formMedField.getText() + "\nInjury Details:"
            + formInjuryField.getText() + "\n";
    Connection con = null;
    Statement st = null;
    ResultSet rs = null;
    String url = "jdbc:mysql://localhost:3306/csia";
    String user = "root";
    String password = "swords";
    try {
        con = DriverManager.getConnection(url, user, password);
        st = con.createStatement();
        rs = st.executeQuery("select * from forms where name='" + formNameField.getText() + "'");
        if (rs.absolute(1)) {
            form = form + rs.getString(2);
        }
        st.executeUpdate("insert into forms values('" + formNameField.getText() + "','" + form + "')");
        formNameField.setText("");
        formDateField.setText("");
        formTimeField.setText("");
        formEmailField.setText("");
        formPhoField.setText("");
        formEmerField.setText("");
        formMedField.setText("");
        formInjuryField.setText("");
        output("The Emergency Injury Form has been added to the database");
    } catch (SQLException ex) {
        Logger lgr = Logger.getLogger(tester.class.getName());
        lgr.log(Level.SEVERE, ex.getMessage(), ex);
```

Figure 28. This code is used to check and update all the fields, while also simultaneously saving the emergency form. Previous forms for the same person are also loaded so that they can all be saved as one chronological form.

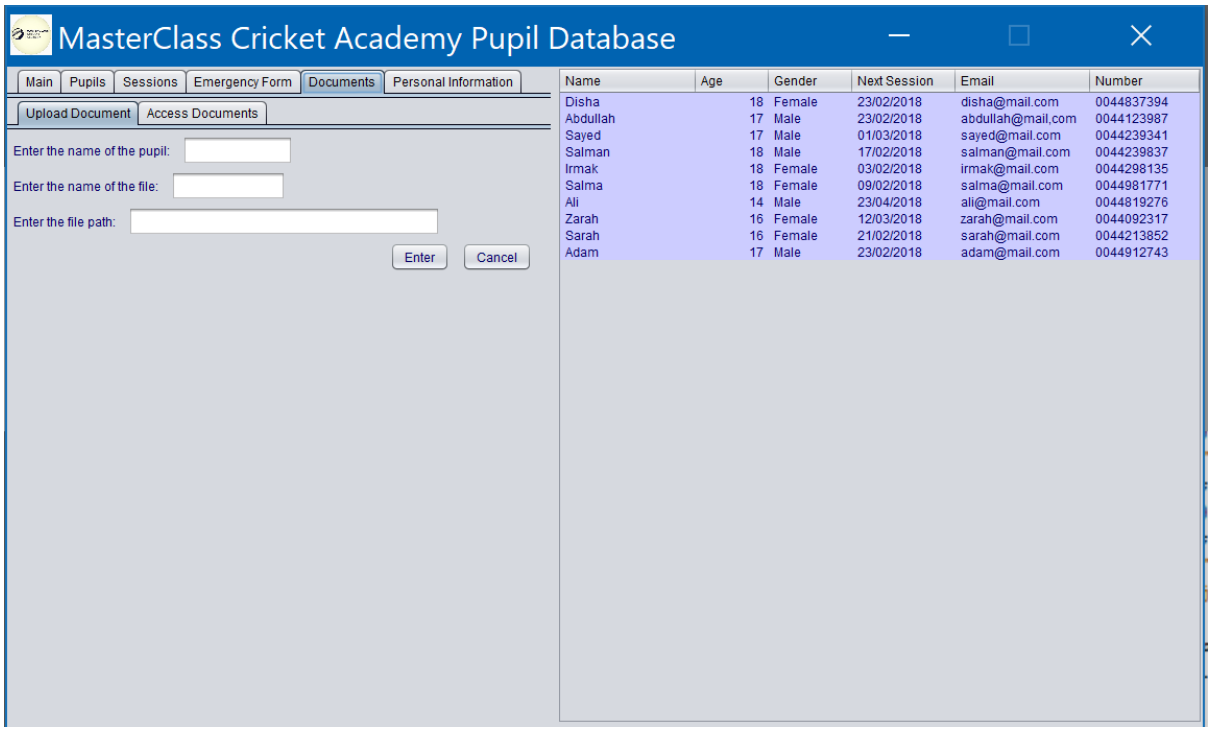

Figure 29. The user enters the name of the pupil and enters a file name with a file path. This is uploaded into the database

```
int flag = 0;
if (docNameField.getText().equals("") || docFileField.getText().equals("") || docPathField.getText().equals("")) {
    output("You must enter all the information");
} else {
    File doc = new File(docPathField.getText());
    if (!doc.exists()) {
        output("FilePath is invalid. Please enter a correct file path.");
    } else {
        for (int count = 0; count < index; count++) {
            if (people[count].getName().equals(docNameField.getText())) {
                Connection con = null;
                Statement st = null;
                ResultSet rs = null;
                String url = "jdbc:mysql://localhost:3306/CricketDatabase";
                String user = "root";
                String password = "";
                try {
                    con = DriverManager.getConnection(url, user, password);
                    st = con.createStatement();
                    String temp = docPathField.getText().replace("\\", "##");
                    st.executeUpdate("insert into Documents values('" + docNameField.getText() +
                            "','" + docFileField.getText() + "','" + temp + "')");
                    flag = 1;
                    output("The file has been added to the database");
                    docNameField.setText("");
                    docFileField.setText("");
                    docPathField.setText("");
```

Figure 30. The file path is uploaded into the database. Checks are in place to make sure the name exists and that the file path exists. To upload the file paths, I used an escape character so that the string could be added.

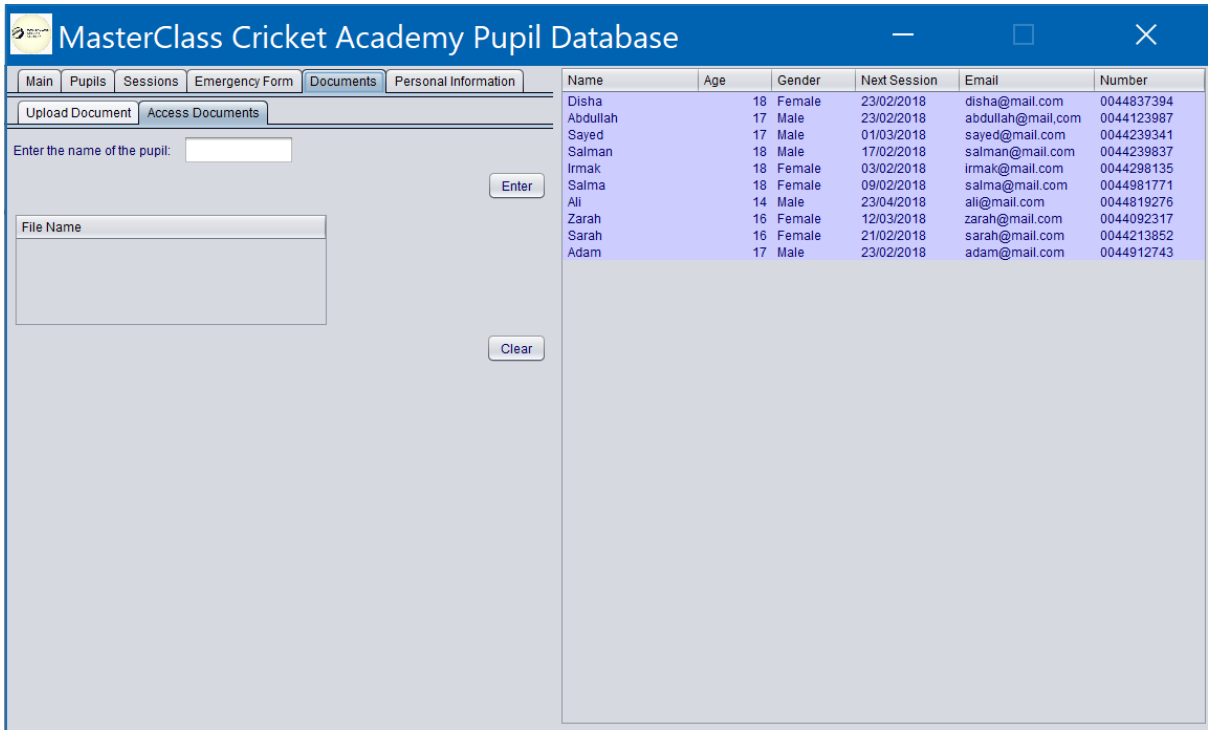

Figure 31. The user enters the name of the pupil and the files are loaded in the table. The user can click on these files to open them. This is enabled through the use of libraries which allow me to access Desktop related methods.

```
int flag = 0;
if (docAccessField.getText().equals("")) {
    output("You need to enter a name");
} else {
    for (int count = 0; count < index; count++) {
        if (people[count].getName().equals(docAccessField.getText())) {
            DefaultTableModel model = (DefaultTableModel) docAccessTable.getModel();
            Object[] row = new Object[1];
            Connection con = null;
            Statement st = null;
            ResultSet rs = null;
            String url = "jdbc:mysql://localhost:3306/CricketDatabase";
            String user = "root";
            String password = "";
            try {
                con = DriverManager.getConnection(url, user, password);
                st = con.createStatement();
                rs = st.executeQuery("select * from Documents where Name ='" + docAccessField.getText() + "'");
                rs.last();
                if(rs.last()==false){
                    output("There are no documents saved for this pupil");
                }
                for (int i = 1; i < rs.getRow() + 1; i++) {
                    rs.absolute(i);
                    row[0] = rs.getString(2);
                    model.addRow(row);
                }
                flag = 1;
```

32. The file names are loaded into the table using a table model.

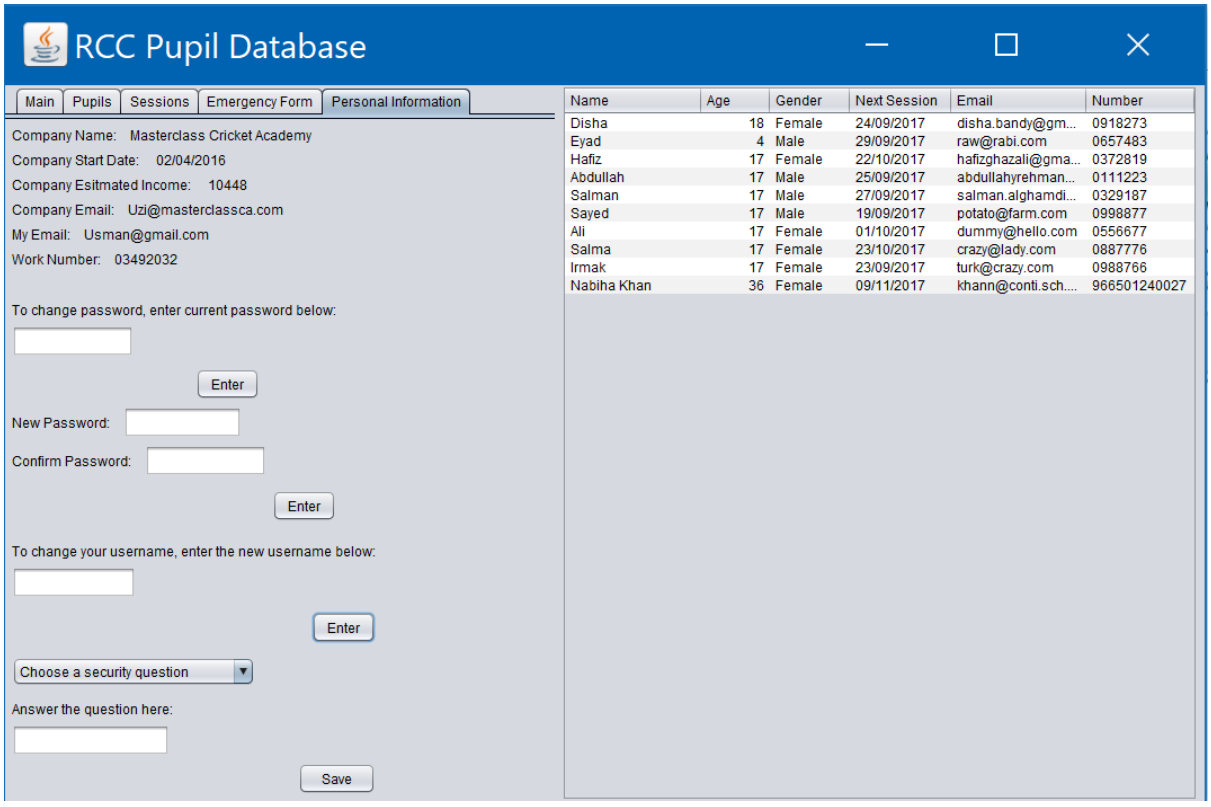

Figure 33. Shows the final tab of the program which is for the user himself, to store data about the company and themselves. Here passwords and usernames can be changed and updated in the database. There is also an option to get a security question so that if you forget your password, it can later be retrieved.

```
private void infoPassButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Connection con = null;
    Statement st = null;
    ResultSet rs = null;
    String url = "jdbc:mysql://localhost:3306/csia";
    String user = "root";
    String password = "swords";
    try {
        con = DriverManager.getConnection(url, user, password);
        st = con.createStatement();
        rs = st.executeQuery("select * from Login");
        rs.absolute(1);
        if (rs.getString(2).equals(infoPassField.getText())) {
            infoNPassField.setVisible(true);
            infoCPassField.setVisible(true);
            infoNPassLabel.setVisible(true);
            infoCPassLabel.setVisible(true);
            infoNPassButton.setVisible(true);
            infoPassField.setText("");
        } else {
            output("This is not the current password");
            infoPassField.setText("");
        }
    } catch (SQLException ex) {
        Logger lgr = Logger.getLogger(tester.class.getName());
        lgr.log(Level.SEVERE, ex.getMessage(), ex);
```

Figure 34. This verifies that the user remembers their old password so that they can be allowed to change it. It compares the data from input and the database and then checks if they match.

```
private void infoNPassButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Connection con = null;
    Statement st = null;
    ResultSet rs = null;
    String url = "jdbc:mysql://localhost:3306/csia";
    String user = "root";
    String password = "swords";
    try {
        con = DriverManager.getConnection(url, user, password);
        st = con.createStatement();
        rs = st.executeQuery("select Username from Login");
        if (rs.absolute(1)) {
            String userN = rs.getString(1);
            if (infoNPassField.getText().equals(infoCPassField.getText())) {
                st.executeUpdate("truncate Login");
                st.executeUpdate("insert into Login values('" + userN + "','" + infoCPassField.getText() + "')");
                output("Your password has been saved");
                infoNPassField.setText("");
                infoCPassField.setText("");
                infoNPassField.setVisible(false);
                infoCPassField.setVisible(false);
                infoNPassLabel.setVisible(false);
                infoCPassLabel.setVisible(false);
                infoNPassButton.setVisible(false);
            } else {
                output("The two passwords you have entered do not match. Please confirm the same password.");
                infoNPassField.setText("");
                infoCPassField.setText("");
            }
        }
    } catch (SQLException ex) {
        Logger lgr = Logger.getLogger(tester.class.getName());
        lgr.log(Level.SEVERE, ex.getMessage(), ex);
```

Figure 35. This looks at the two passwords entered, and if they match then the database is updated to match them.

```
private void infoUserButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Connection con = null;
    Statement st = null;
    ResultSet rs = null;
    String url = "jdbc:mysql://localhost:3306/csia";
    String user = "root";
    String password = "swords";
    boolean check = false;
    try {
        con = DriverManager.getConnection(url, user, password);
        st = con.createStatement();
        rs = st.executeQuery("select Passcode from Login");
        if (rs.absolute(1)) {
            String passN = rs.getString(1);
            st.executeUpdate("truncate Login");
            st.executeUpdate("insert into Login values('" + infoUserField.getText() + "','" + passN + "')");
            output("The username has been changed");
            infoUserField.setText("");
        }
    } catch (SQLException ex) {
        Logger lgr = Logger.getLogger(tester.class.getName());
        lgr.log(Level.SEVERE, ex.getMessage(), ex);
    } finally {
        try {
            if (rs != null) {
                rs.close();
            }
            if (st != null) {
                st.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (SQLException ex) {
            Logger lgr = Logger.getLogger(tester.class.getName());
            lgr.log(Level.WARNING, ex.getMessage(), ex);
        }
    }
```

Figure 36. To update the Username the user does not have to input as much data and does not have as many checks as the data is not as much at risk. The new username is simply updated in the database.

```
private void infoSecurityButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if (infoSecurityCombo.getSelectedIndex() == 0) {
        output("Please select a question for security");
    } else if (infoSecurityField.getText().equals("")) {
        output("Please enter an answer to the chosen question");
    } else {
        output("Thank you, your response has been saved");
        Connection con = null;
        Statement st = null;
        ResultSet rs = null;
        String url = "jdbc:mysql://localhost:3306/csia";
        String user = "root";
        String password = "swords";
        try {
            con = DriverManager.getConnection(url, user, password);
            st = con.createStatement();
            st.executeUpdate("truncate Forgot");
            st.executeUpdate("insert into Forgot values('true', '"+infoSecurityCombo.getSelectedItem()+"', '"
                    +infoSecurityField.getText()+"')");
        } catch (SQLException ex) {
            Logger lgr = Logger.getLogger(tester.class.getName());
            lgr.log(Level.SEVERE, ex.getMessage(), ex);
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (st != null) {
                    st.close();
                }
                if (con != null) {
                    con.close();
                }
            } catch (SQLException ex) {
                Logger lgr = Logger.getLogger(tester.class.getName());
                lgr.log(Level.WARNING, ex.getMessage(), ex);
            }
        }
```

Figure 37. This code is used to set up a security question for the user, which is asked before they can have their password emailed to them. It takes in a question from a provided list and an answer, and updates them in a database.

It is evident in the screenshots that throughout the program I make use of the Object Array and Objects. The “people[ ]” array is used extensively to access different pupils in the database and change different parameters for them. The pupil objects are encapsulated, meaning that their fields are made private so that error checks can be used easily and to ensure safe extensibility to the program. This is why Accessor and Mutator Methods are used throughout the program. Other OOP features like polymorphism have also been implemented as discussed earlier in *Figure 14.*

This shows the main functions and features of my program. There are many smaller error checks and validation checks I could not fully show as it would take too long. However, this can be seen in the full code in Appendix 3.

Number of words: 555