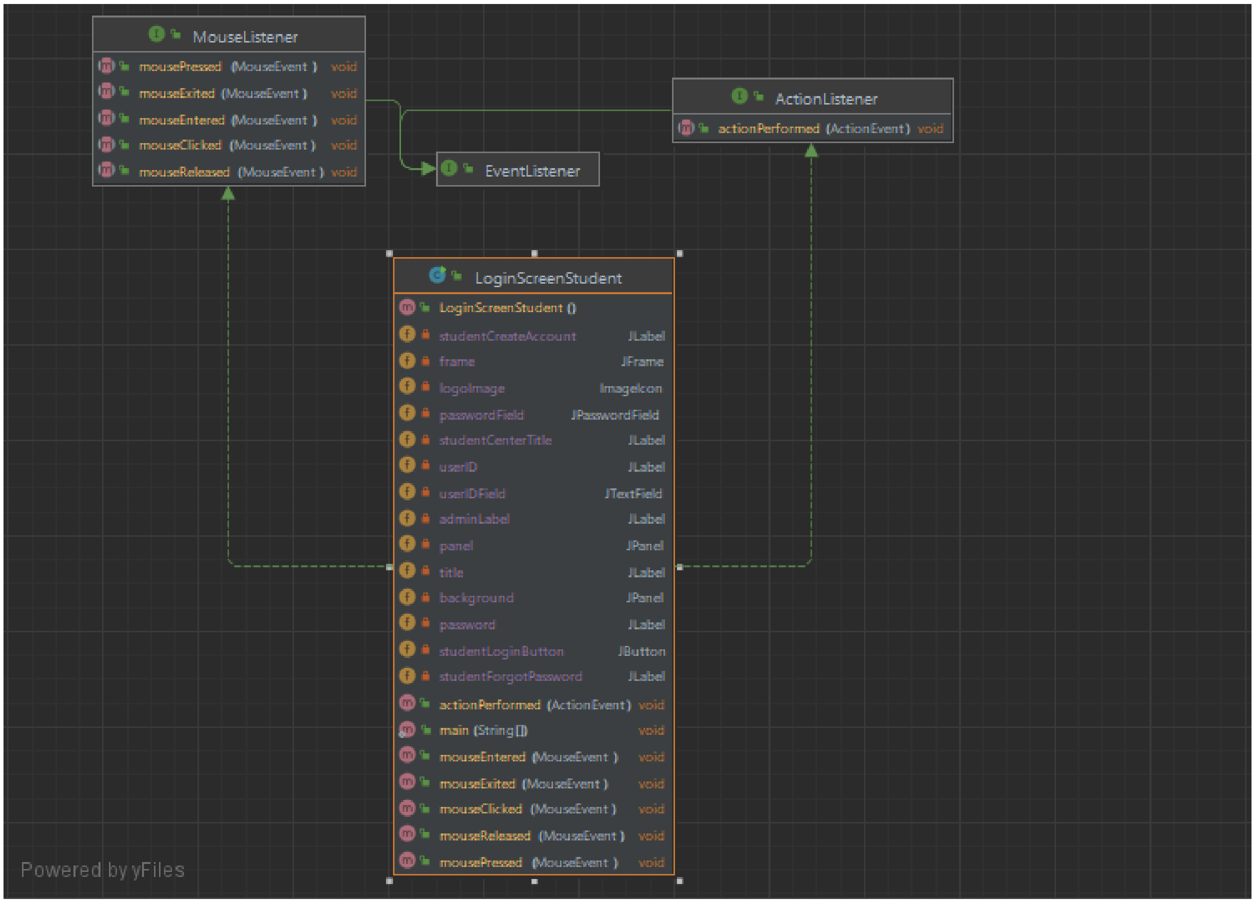
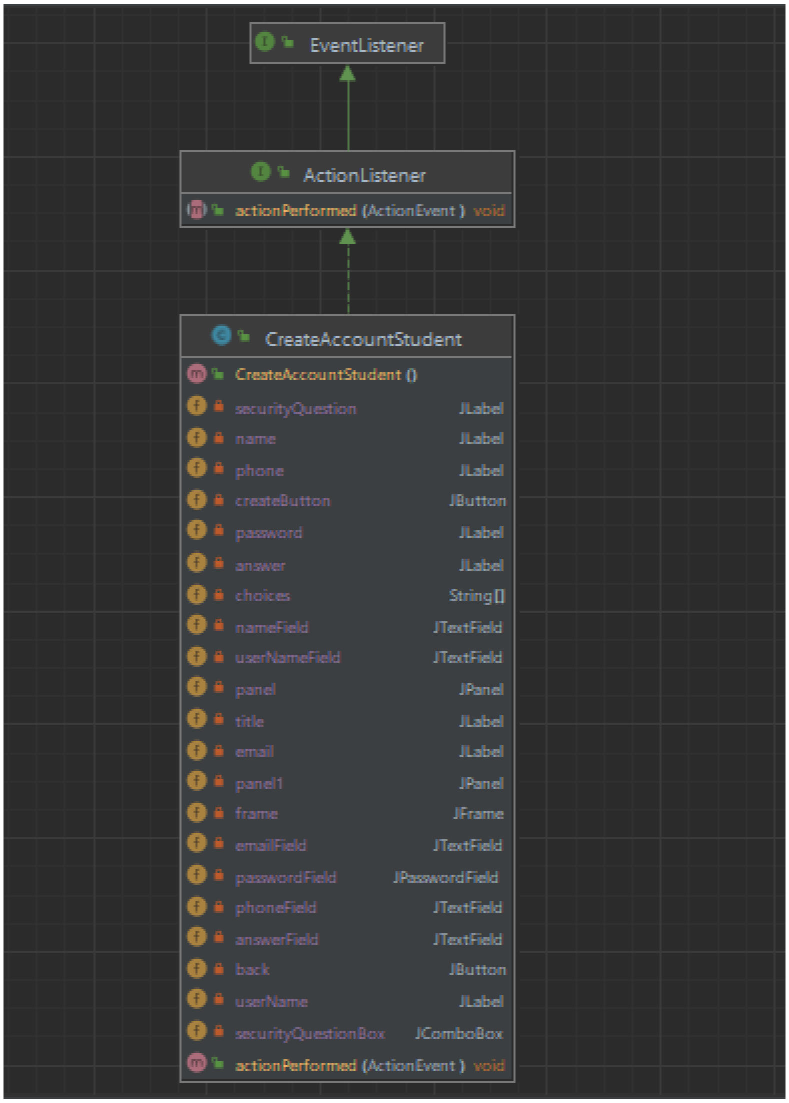
Criterion C

UML Diagrams:

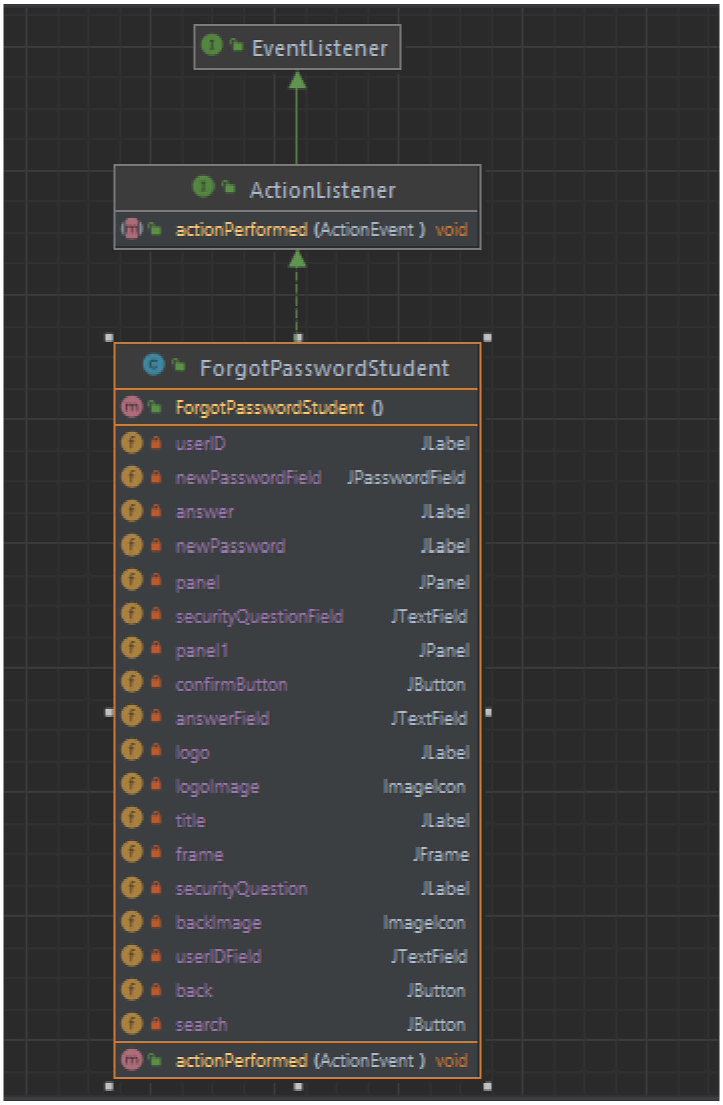
<u>LoginScreenStudent</u>


MouseListener
mousePressed (MouseEvent) void
mouseExited (MouseEvent) void
mouseEntered (MouseEvent) void
mouseClicked (MouseEvent) void
mouseReleased (MouseEvent) void
ActionListener
actionPerformed (ActionEvent) void
EventListener
LoginScreenStudent
LoginScreenStudent ()
studentCreateAccount JLabel
frame JFrame
logoImage ImageIcon
passwordField JPasswordField
studentCenterTitle JLabel
userID JLabel
userIDField JTextField
adminLabel JLabel
panel JPanel
title JLabel
background JPanel
password JLabel
studentLoginButton JButton
studentForgotPassword JLabel
actionPerformed (ActionEvent) void
main (String[]) void
mouseEntered (MouseEvent) void
mouseExited (MouseEvent) void
mouseClicked (MouseEvent) void
mouseReleased (MouseEvent) void
mousePressed (MouseEvent) void
Powered by yFiles

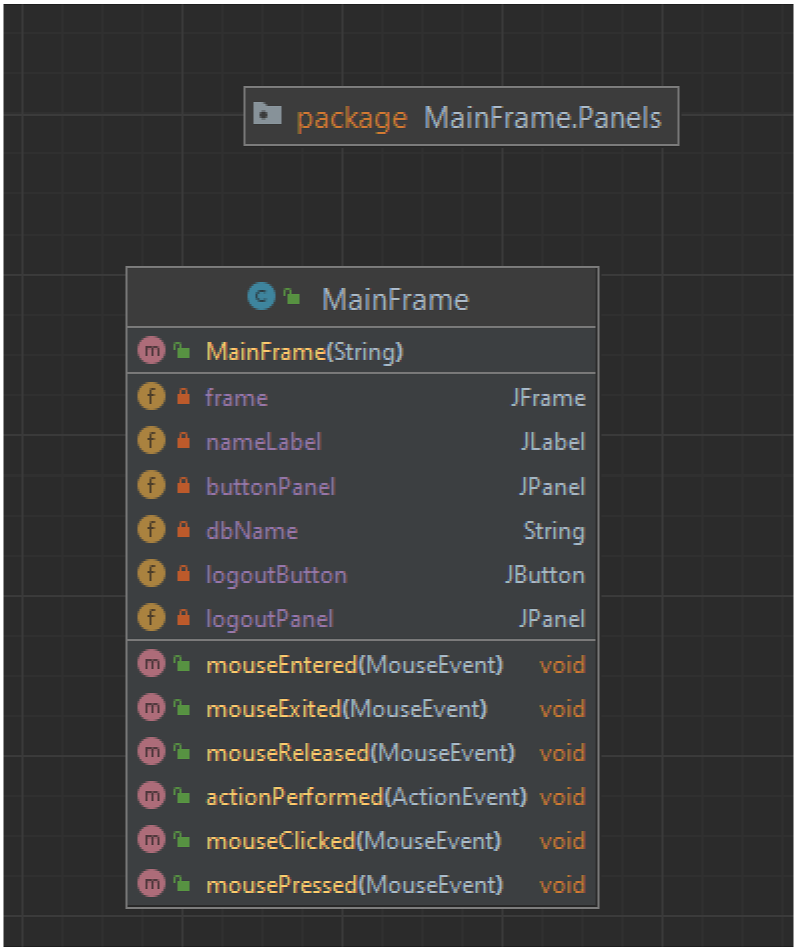
CreateAccountStudent



Forgot Password



Student MainFrame

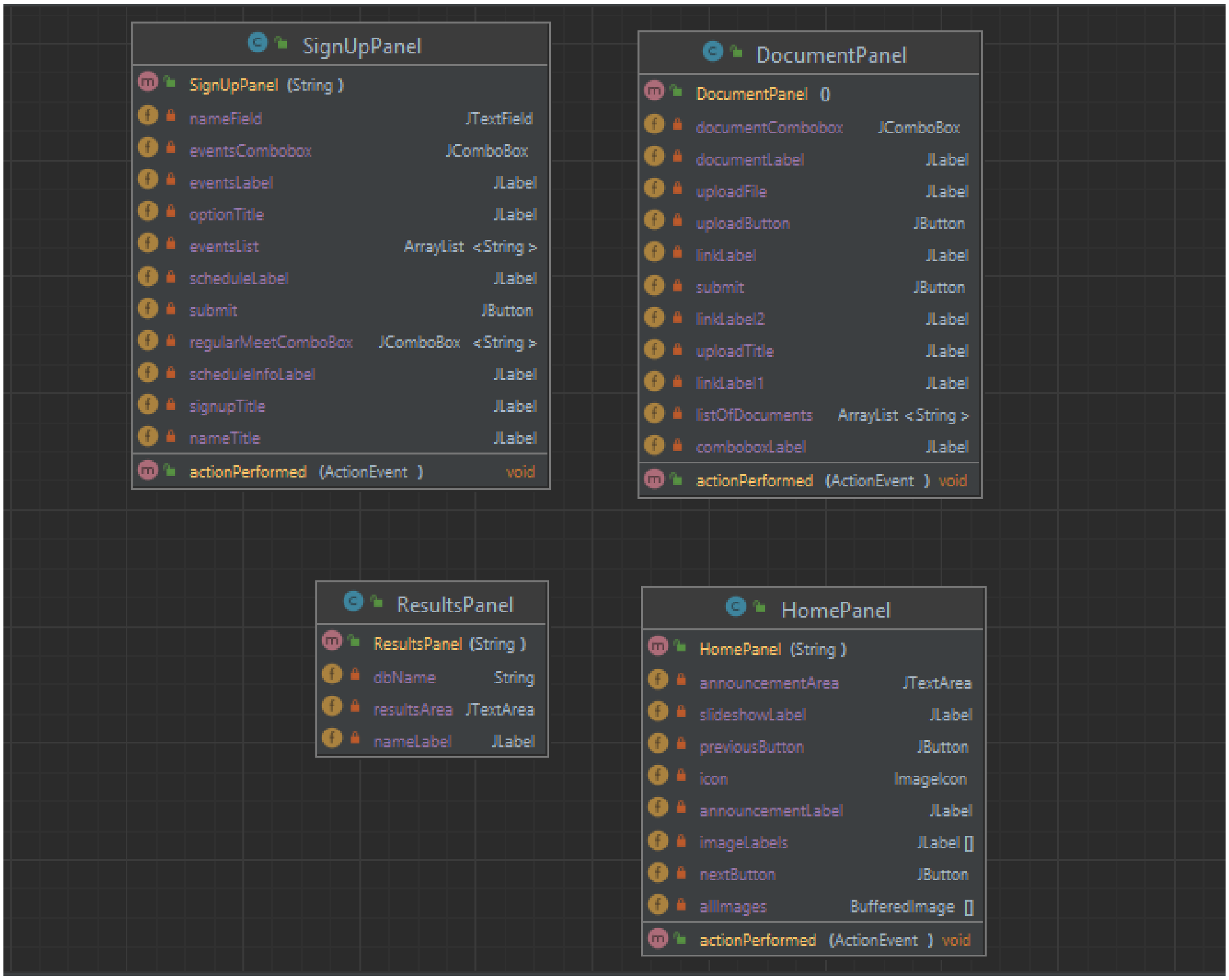


package MainFrame.Panels

MainFrame

MainFrame(String)	
frame	JFrame
nameLabel	JLabel
buttonPanel	JPanel
dbName	String
logoutButton	JButton
logoutPanel	JPanel
mouseEntered(MouseEvent)	void
mouseExited(MouseEvent)	void
mouseReleased(MouseEvent)	void
actionPerformed(ActionEvent)	void
mouseClicked(MouseEvent)	void
mousePressed(MouseEvent)	void

<u>Student Tabbed Panels</u>



SignUpPanel

SignUpPanel (String)	
nameField	JTextField
eventsCombobox	JComboBox
eventsLabel	JLabel
optionTitle	JLabel
eventsList	ArrayList < String >
scheduleLabel	JLabel
submit	JButton
regularMeetComboBox	JComboBox < String >
scheduleInfoLabel	JLabel
signupTitle	JLabel
nameTitle	JLabel
actionPerformed (ActionEvent)	void

DocumentPanel

DocumentPanel ()	
documentCombobox	JComboBox
documentLabel	JLabel
uploadFile	JLabel
uploadButton	JButton
linkLabel	JLabel
submit	JButton
linkLabel2	JLabel
uploadTitle	JLabel
linkLabel1	JLabel
listOfDocuments	ArrayList < String >
comboboxLabel	JLabel
actionPerformed (ActionEvent)	void

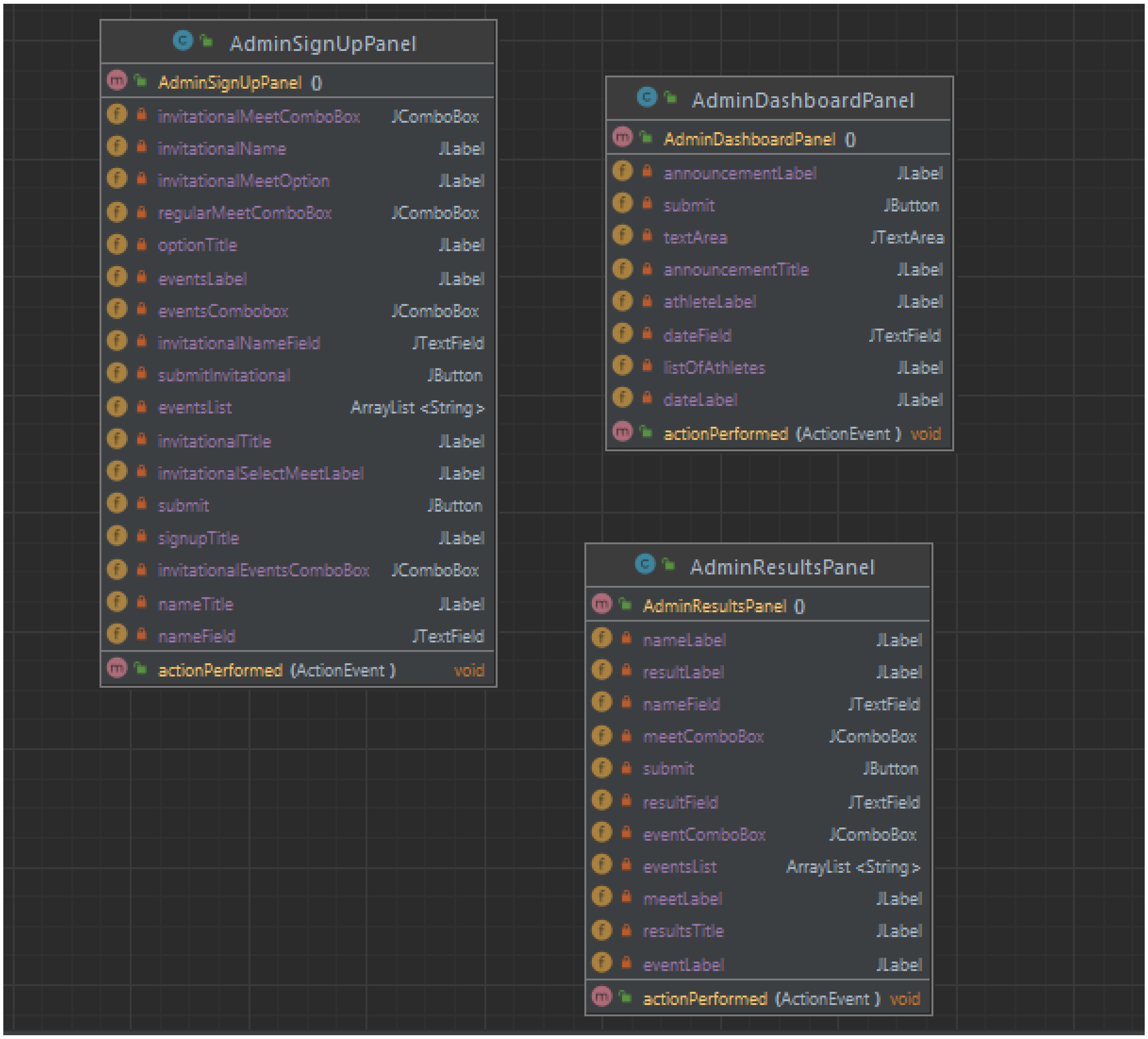
ResultsPanel

ResultsPanel (String)	
dbName	String
resultsArea	JTextArea
nameLabel	JLabel

HomePanel

HomePanel (String)	
announcementArea	JTextArea
slideshowLabel	JLabel
previousButton	JButton
icon	ImageIcon
announcementLabel	JLabel
imageLabels	JLabel []
nextButton	JButton
allImages	BufferedImage []
actionPerformed (ActionEvent)	void

Administrator Tabbed Panels



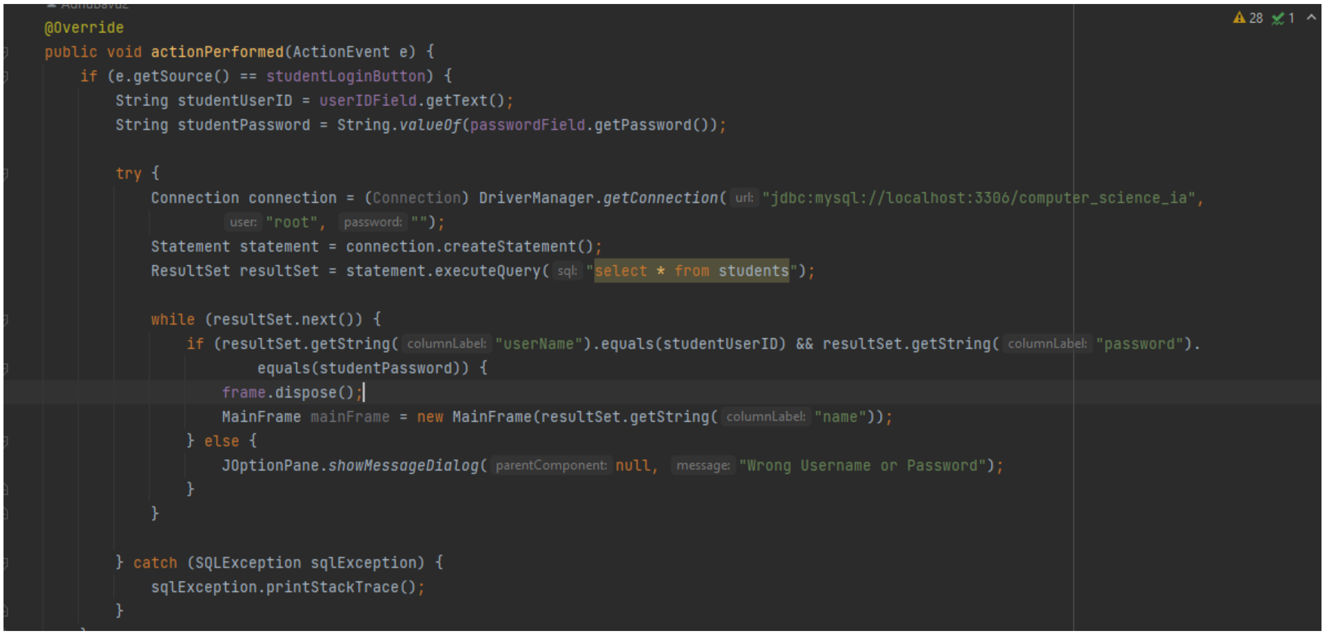
AdminSignUpPanel	
AdminSignUpPanel ()	
invitationalMeetComboBox	JComboBox
invitationalName	JLabel
invitationalMeetOption	JLabel
regularMeetComboBox	JComboBox
optionTitle	JLabel
eventsLabel	JLabel
eventsCombobox	JComboBox
invitationalNameField	JTextField
submitInvitational	JButton
eventsList	ArrayList <String>
invitationalTitle	JLabel
invitationalSelectMeetLabel	JLabel
submit	JButton
signupTitle	JLabel
invitationalEventsComboBox	JComboBox
nameTitle	JLabel
nameField	JTextField
actionPerformed (ActionEvent)	void

AdminDashboardPanel	
AdminDashboardPanel ()	
announcementLabel	JLabel
submit	JButton
textArea	JTextArea
announcementTitle	JLabel
athleteLabel	JLabel
dateField	JTextField
listOfAthletes	JLabel
dateLabel	JLabel
actionPerformed (ActionEvent)	void

AdminResultsPanel	
AdminResultsPanel ()	
nameLabel	JLabel
resultLabel	JLabel
nameField	JTextField
meetComboBox	JComboBox
submit	JButton
resultField	JTextField
eventComboBox	JComboBox
eventsList	ArrayList <String>
meetLabel	JLabel
resultsTitle	JLabel
eventLabel	JLabel
actionPerformed (ActionEvent)	void

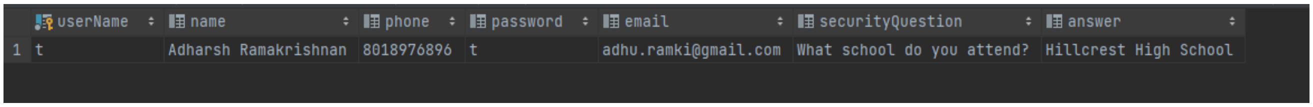
Complex Code:

<u>Login System Validation</u>



```
@Override
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == studentLoginButton) {
        String studentUserID = userIDField.getText();
        String studentPassword = String.valueOf(passwordField.getPassword());

        try {
            Connection connection = (Connection) DriverManager.getConnection( url: "jdbc:mysql://localhost:3306/computer_science_ia",
                    user: "root",  password: "");
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery( sql: "select * from students");

            while (resultSet.next()) {
                if (resultSet.getString( columnLabel: "userName").equals(studentUserID) && resultSet.getString( columnLabel: "password").
                        equals(studentPassword)) {
                    frame.dispose();
                    MainFrame mainFrame = new MainFrame(resultSet.getString( columnLabel: "name"));
                } else {
                    JOptionPane.showMessageDialog( parentComponent: null,  message: "Wrong Username or Password");
                }
            }

        } catch (SQLException sqlException) {
            sqlException.printStackTrace();
        }
```

Figure 1 - Snapshot of the Login system that verifies the UserID and password in the registered database.

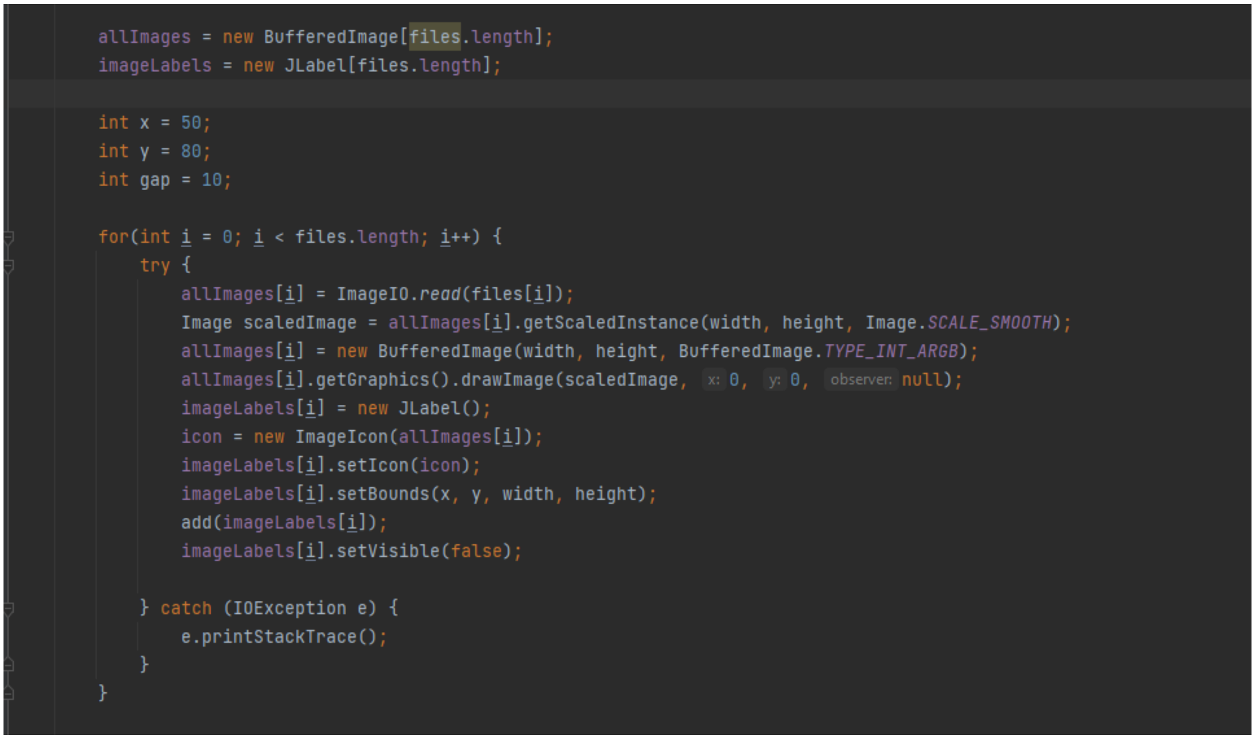


	userName	name	phone	password	email	securityQuestion	answer
1	t	Adharsh Ramakrishnan	8018976896	t	adhu.ramki@gmail.com	What school do you attend?	Hillcrest High School

Figure 2 - Snapshot of the Database table that holds the login ID and password

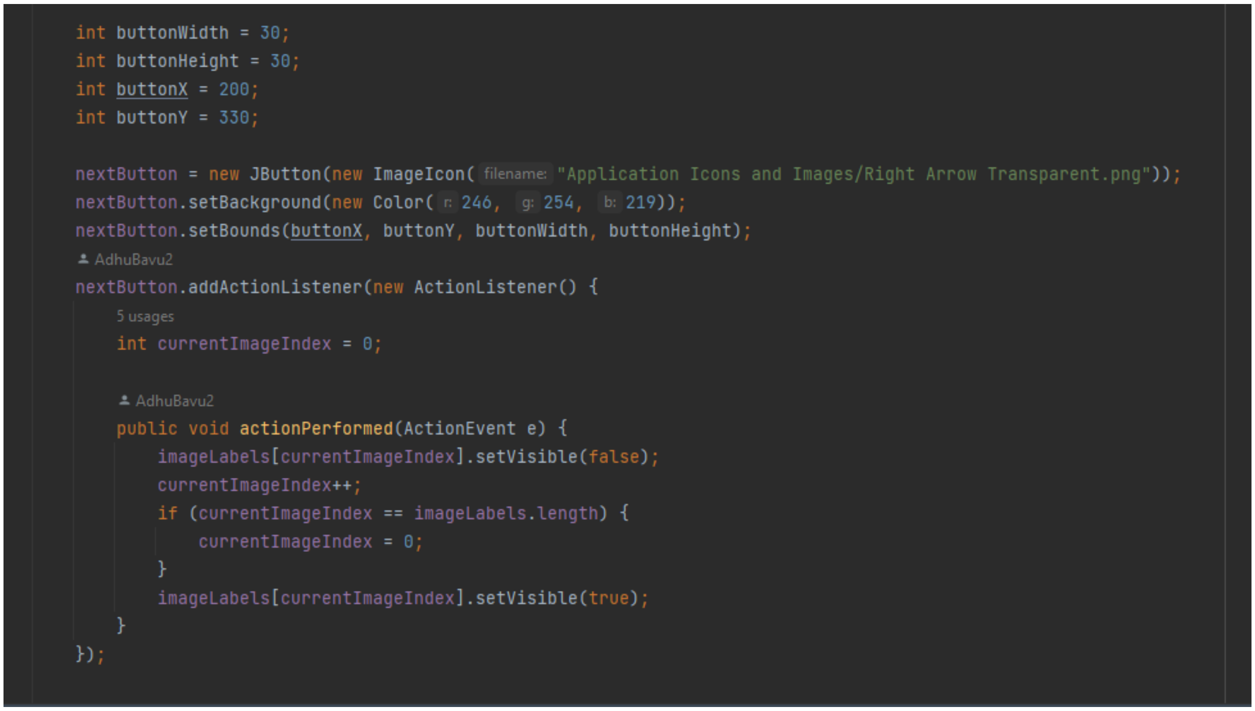
The code above displays the logic for determining a safe and secure login system. When the user enters their UserID and password, their values are stored as Strings. These values are then compared to the values registered in the database. If the values match perfectly, the login frame is closed and the MainFrame is opened but if the values do not match, an error message is shown. To check through every value in the database, a while loop is implemented, which keeps checking the values until the condition is met. If the condition is not met, then an error message saying, “Wrong Username or Password” is displayed. Due to the use of SQL in order to store data, many exceptions can occur when trying to access and modify data in the database through Java Swing. In order to catch and handle these exceptions, a try-catch block is used in order to catch and handle any SQL Exceptions.

Home Page - Image Slider



```
allImages = new BufferedImage[files.length];
imageLabels = new JLabel[files.length];

int x = 50;
int y = 80;
int gap = 10;

for(int i = 0; i < files.length; i++) {
    try {
        allImages[i] = ImageIO.read(files[i]);
        Image scaledImage = allImages[i].getScaledInstance(width, height, Image.SCALE_SMOOTH);
        allImages[i] = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);
        allImages[i].getGraphics().drawImage(scaledImage,  x: 0,  y: 0,  observer: null);
        imageLabels[i] = new JLabel();
        icon = new ImageIcon(allImages[i]);
        imageLabels[i].setIcon(icon);
        imageLabels[i].setBounds(x, y, width, height);
        add(imageLabels[i]);
        imageLabels[i].setVisible(false);

    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Figure 3 - Snapshot of a section of the HomePanel class that loops through images in a folder and displays them all accordingly.

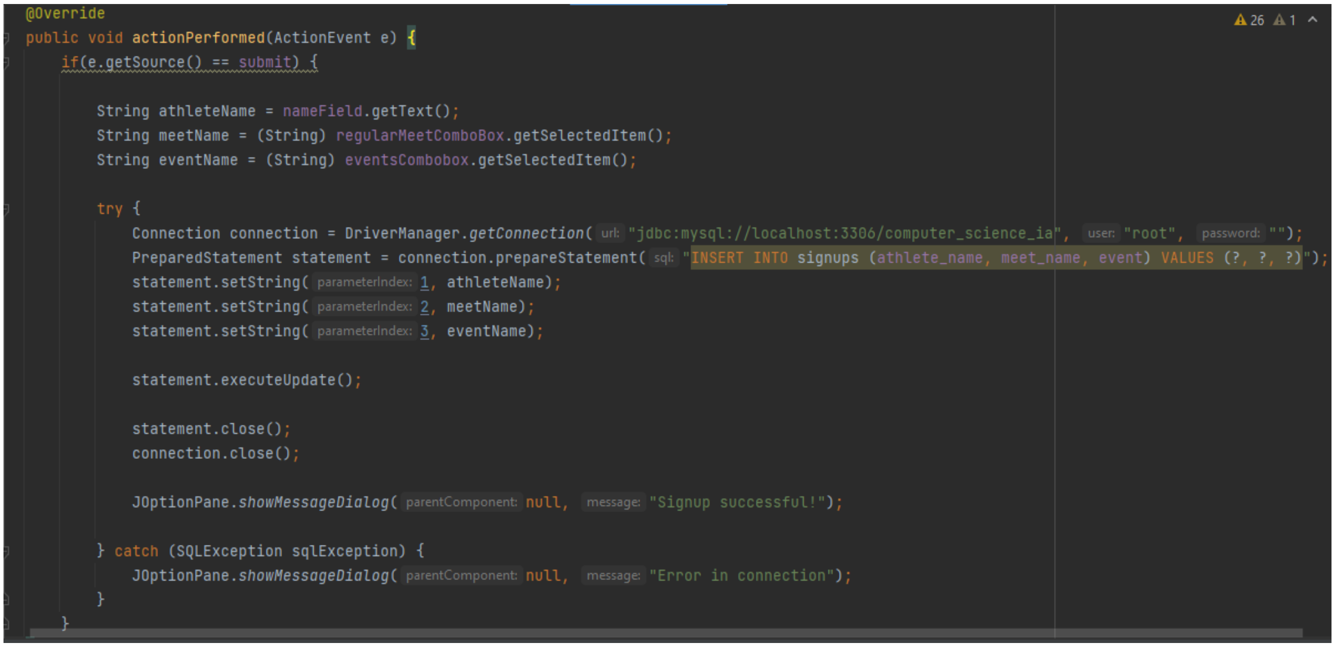


```
int buttonWidth = 30;
int buttonHeight = 30;
int buttonX = 200;
int buttonY = 330;

nextButton = new JButton(new ImageIcon( filename: "Application Icons and Images/Right Arrow Transparent.png"));
nextButton.setBackground(new Color( r: 246,  g: 254,  b: 219));
nextButton.setBounds(buttonX, buttonY, buttonWidth, buttonHeight);
AdhuBavu2
nextButton.addActionListener(new ActionListener() {
    5 usages
    int currentImageIndex = 0;

    AdhuBavu2
    public void actionPerformed(ActionEvent e) {
        imageLabels[currentImageIndex].setVisible(false);
        currentImageIndex++;
        if (currentImageIndex == imageLabels.length) {
            currentImageIndex = 0;
        }
        imageLabels[currentImageIndex].setVisible(true);
    }
});
```

Figure 4 - Snapshot of code that shows how images can be moved when a button is clicked

The code in Figures 3 and 4 show an advanced feature in the application. One requirement of the application was to display images relating to the Track and Field program. In order to make this efficient and functional, I decided to employ a system where administrators could simply add images to a certain folder and the program would loop through that folder in order to display all of the images in it. For this, a for loop is employed which loops through the folder and scales all of the images to the same size. In order to move the images, a button is used which moves right to display the next picture and moves left to display the previous picture. Similar to Figures 1 and 2, the ActionListener interface is implemented in order to get event logs when buttons are clicked. This helps the program update dynamically which allows for a smooth and easy user experience.

Sign Up Page - Database Updating

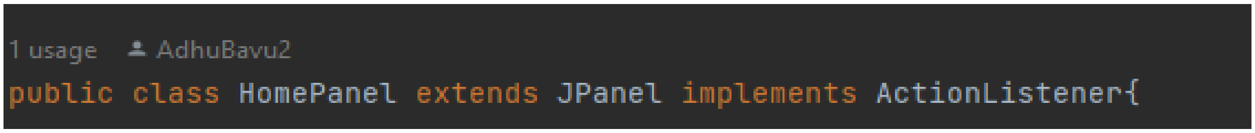


```
@Override
public void actionPerformed(ActionEvent e) {
    if(e.getSource() == submit) {

        String athleteName = nameField.getText();
        String meetName = (String) regularMeetComboBox.getSelectedItem();
        String eventName = (String) eventsCombobox.getSelectedItem();

        try {
            Connection connection = DriverManager.getConnection( url: "jdbc:mysql://localhost:3306/computer_science_ia",  user: "root",  password: "");
            PreparedStatement statement = connection.prepareStatement( sql: "INSERT INTO signups (athlete_name, meet_name, event) VALUES (?, ?, ?)");
            statement.setString( parameterIndex: 1, athleteName);
            statement.setString( parameterIndex: 2, meetName);
            statement.setString( parameterIndex: 3, eventName);

            statement.executeUpdate();

            statement.close();
            connection.close();

            JOptionPane.showMessageDialog( parentComponent: null,  message: "Signup successful!");

        } catch (SQLException sqlException) {
            JOptionPane.showMessageDialog( parentComponent: null,  message: "Error in connection");
        }
    }
```

Figure 5 - Snapshot of code inserting values into the Database

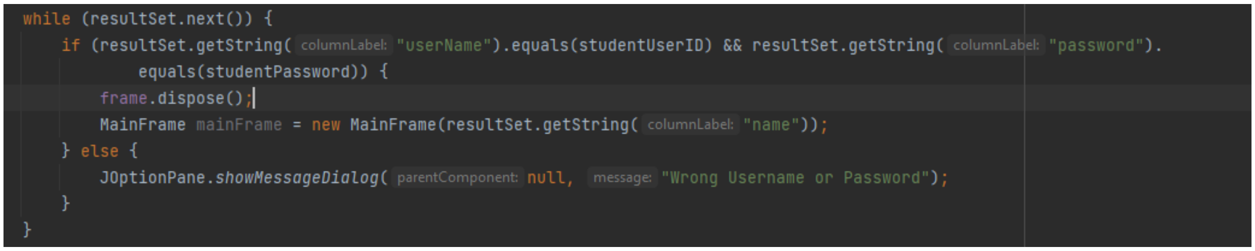
The snapshot above shows the code for the SignUpPanel class which allows for users to sign up to regular meets. These sign ups are stored as String values and are then entered into the database. The values in the database allows for administrators to see student sign ups which saves administrators time by not worrying about them signing athletes up. In the code, the values are directly inserted into the SQL database as this allows the program to dynamically update and allows for easy management for both students and administrators.

Inheritance, Interfaces, and Polymorphism



```
1 usage   AdhuBavu2
public class HomePanel extends JPanel implements ActionListener{
```

Figure 6 - Snapshot of the class HomePanel which extends the JPanel class and implements the ActionListener interface

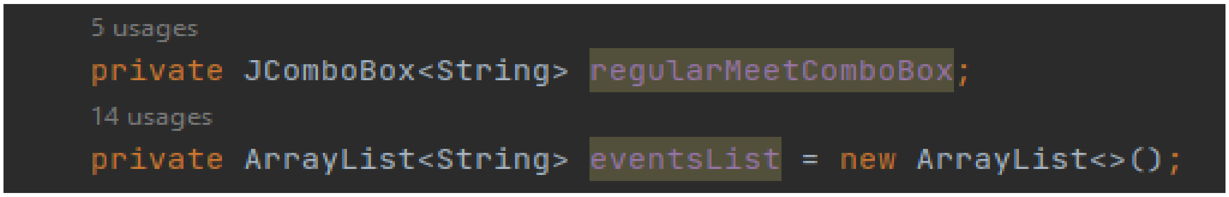


```
while (resultSet.next()) {
    if (resultSet.getString( columnLabel: "userName").equals(studentUserID) && resultSet.getString( columnLabel: "password").
            equals(studentPassword)) {
        frame.dispose();
        MainFrame mainFrame = new MainFrame(resultSet.getString( columnLabel: "name"));
    } else {
        JOptionPane.showMessageDialog( parentComponent: null,  message: "Wrong Username or Password");
    }
}
```

Figure 7 - Snapshot of code that creates an instance of the MainFrame class and goes to it when met by conditions in the database

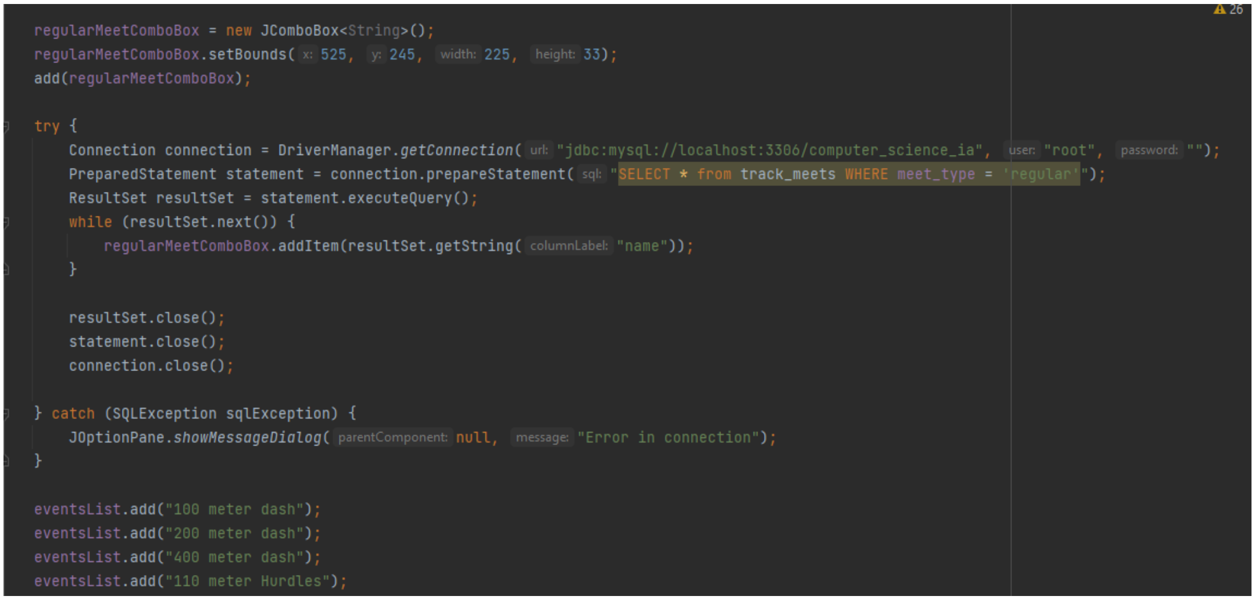
The code above shows an example of using both inheritance and interfaces in an application in order for the UI to dynamically update in a program. The HomePanel class extends the JPanel class which contains the components for a JPanel. Because of this, the HomePanel class is already a panel and already has the properties of a JPanel and allows for easier displaying of UI, especially in a place where there are tabs used for navigation. The ActionListener interface allows for buttons and other components to interact with the application and perform some task (usually database modifications or updates) when the button is clicked.

Use of Data Structures - Lists



```
5 usages
private JComboBox<String> regularMeetComboBox;
14 usages
private ArrayList<String> eventsList = new ArrayList<>();
```

Figure 8 - Instance variables creating a list of JComboBox and an ArrayList.

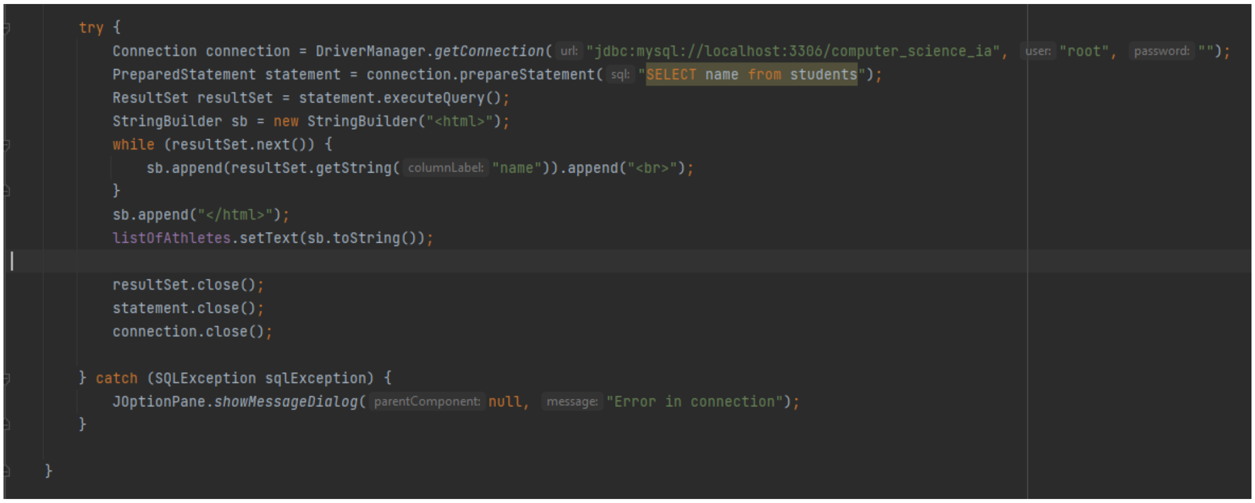


```
regularMeetComboBox = new JComboBox<String>();
regularMeetComboBox.setBounds( x: 525,  y: 245,  width: 225,  height: 33);
add(regularMeetComboBox);

try {
    Connection connection = DriverManager.getConnection( url: "jdbc:mysql://localhost:3306/computer_science_ia",  user: "root",  password: "");
    PreparedStatement statement = connection.prepareStatement( sql: "SELECT * from track_meets WHERE meet_type = 'regular'");
    ResultSet resultSet = statement.executeQuery();
    while (resultSet.next()) {
        regularMeetComboBox.addItem(resultSet.getString( columnLabel: "name"));
    }

    resultSet.close();
    statement.close();
    connection.close();

} catch (SQLException sqlException) {
    JOptionPane.showMessageDialog( parentComponent: null,  message: "Error in connection");
}

eventsList.add("100 meter dash");
eventsList.add("200 meter dash");
eventsList.add("400 meter dash");
eventsList.add("110 meter Hurdles");
```

Figure 9 - Filling the JComboBox list and Array list with values and storing those values in the database.

Figure 8, shown above, creates a JComboBox object of type String which stores all the different “regular” meets so students can sign up for the regular meets. A JComboBox list is used because it allows the application to add the different meets in the database. An ArrayList was used to store the different events in the track meet rather than directly storing them in the database itself because connecting to the database takes a lot of time and power and since there are only 14 events and they never change, an ArrayList was an easy way to store the different events in the track meet.

<u>Creating and Retrieving from Database</u>

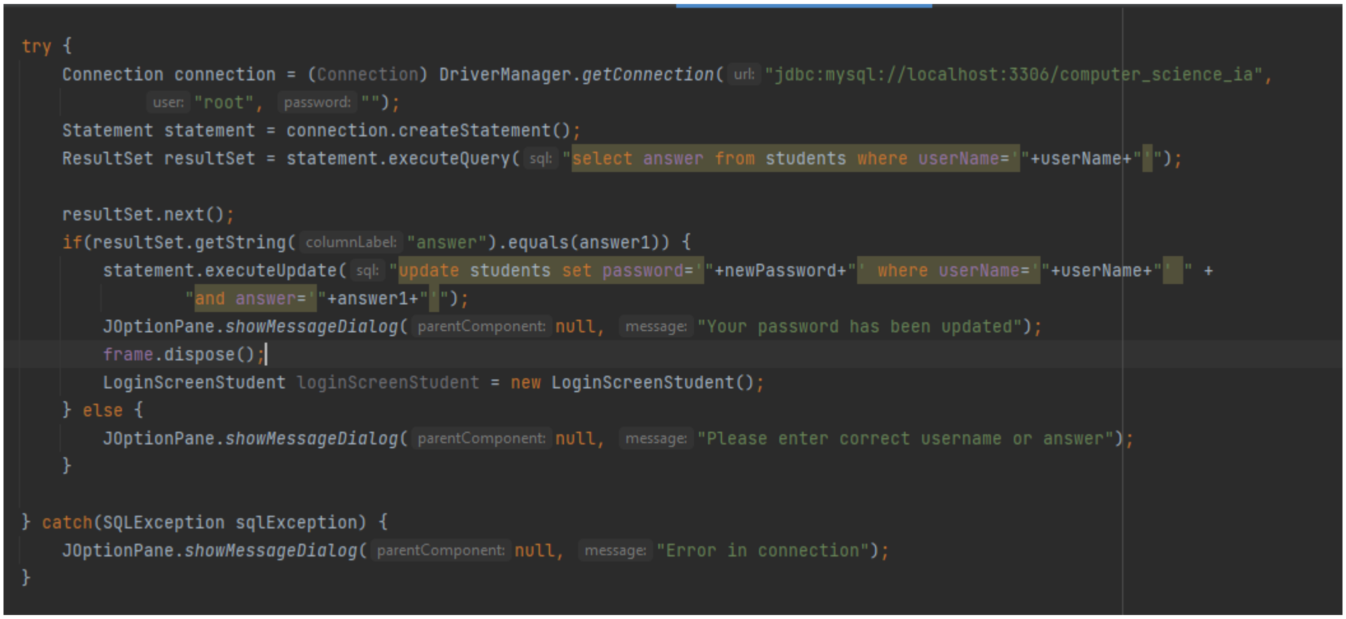


```
    try {
        Connection connection = DriverManager.getConnection( url: "jdbc:mysql://localhost:3306/computer_science_ia",  user: "root",  password: "");
        PreparedStatement statement = connection.prepareStatement( sql: "SELECT name from students");
        ResultSet resultSet = statement.executeQuery();
        StringBuilder sb = new StringBuilder("<html>");
        while (resultSet.next()) {
            sb.append(resultSet.getString( columnLabel: "name")).append("<br>");
        }
        sb.append("</html>");
        listOfAthletes.setText(sb.toString());

        resultSet.close();
        statement.close();
        connection.close();

    } catch (SQLException sqlException) {
        JOptionPane.showMessageDialog( parentComponent: null,  message: "Error in connection");
    }

}
```

Figure 10 - Selecting the students from a database table and appending them in a label

The snapshot above shows a snippet of code from the AdminDashboardPanel. This panel allows for administrators to add announcements and see the total number of students registered through the system as well. This is important because administrators need to be able to see the total number of students registered so they can monitor and put in times for them. The list is created by searching through the "name" column in the "students" table and displaying all of the names registered by creating a JLabel and appending the new names onto it.

Dynamic Updating in the Database



```
try {
    Connection connection = (Connection) DriverManager.getConnection( url: "jdbc:mysql://localhost:3306/computer_science_ia",
            user: "root",  password: "");
    Statement statement = connection.createStatement();
    ResultSet resultSet = statement.executeQuery( sql: "select answer from students where userName='"+userName+"'");

    resultSet.next();
    if(resultSet.getString( columnLabel: "answer").equals(answer1)) {
        statement.executeUpdate( sql: "update students set password='"+newPassword+"' where userName='"+userName+"' " +
                "and answer='"+answer1+"'");
        JOptionPane.showMessageDialog( parentComponent: null,  message: "Your password has been updated");
        frame.dispose();
        LoginScreenStudent loginScreenStudent = new LoginScreenStudent();
    } else {
        JOptionPane.showMessageDialog( parentComponent: null,  message: "Please enter correct username or answer");
    }

} catch(SQLException sqlException) {
    JOptionPane.showMessageDialog( parentComponent: null,  message: "Error in connection");
}
```

Figure 11 - Updating the database dynamically when user wishes to change their password

The code above takes place in the ForgotPasswordStudent class. This class attempts to reset the students password in-case they forget. When students create their account, they select a security question and answer it. The forgot password page asks them for their userID and will find the matching security question in the database. If the person answers the security question correctly, their new password will be saved and they can log in using that, however if their question is answered incorrectly, then the password will not be reset. This is done through a query where if their answer is correctly matching in the database, then an UPDATE statement is run changing the password.

Conclusion

Overall, this application uses various algorithmic commands and UI elements that control the front-end as well as update the database in order to create a smooth and error-free application. This application uses:

1. Polymorphism

2. Inheritance
3. Interfaces
4. Data Structures including ArrayLists and 1-Dimensional arrays
5. Adding data from the Java Application to the Database directly
6. Retrieving data from the Database to the Java Application directly
7. Sorting and organizing through Arrays
8. Encapsulation and protection of variables
9. Catching and Handling exceptions
10. Searching for specific data in the database through the Application

(1046 words)