IB Computer Science HL Internal Assessment

Candidate Code: jjg437
Session Number: 000883-0027

Criterion A: Planning

Scenario:

The client for this project is my coworker Bonita Keller, who I work with at a local gym's kid daycare/activities center. Inside and outside of the job, Keller teaches students and young children important academic skills useful for school and upcoming school years. Outside of the job, she specifically works with a nonprofit organization.

With the pandemic coming around, her job became much more difficult. With in person school, there were only a handful of students that attended her classes, as most of them already were provided with the assistance from their teachers at school. However, as the pandemic hit and schools were set to virtual, it's been a lot harder for students to stay on track with their academics. So, many joined her classes. She used to have to manually input grades for her students in person, but now that it's virtual, it may be difficult to do the same with the increasing influx of students.

At work, Bonita heard that I was interested in a career in Computer Science and asked if I could provide any help with her problem; specifically a program dedicated to organizing and recording her students' grades.

The previous summer, I had touched up upon SQLite (as my mother suggested for me to do) and believed my knowledge with this would be a great use for Bonita's scenario. Because Bonita has little experience with computer science, making the database program a web application/graphical user interface would allow her to have a much easier time operating it.

Rationale for Solution

The program will allow Bonita to have a much more efficient and faster method of inputting and organizing student's information/data. For example, instead of having to manually input all her student's information, the program can just automatically calculate certain data based on the information she inputs into the program. The program will be easily accessible, less time consuming, and interactive.

I will use the programming language Java (specifically object oriented) because it's extremely compatible and I have experience with it and creating graphical user interfaces. I also think it would be very efficient and fast for this project's execution due to the clear structure it provides and the amount of repeated operations that will undergo. Bonita let me know she also is operating with a Windows system, which is why I wanted to use Java as well as I can operate it on my Macbook as well as a Windows system.

As stated before, I gained experience with SQLite over the previous summer, which is the reason I chose it. While studying the program, I learned that it provides lots of reliability, efficiency, performance, accessibility (compatibility with other computer systems), and affordability. I knew that SQLite was compatible with the Java programming language I was using as well, which was helpful in choosing a database program. It wasn't too difficult to make this decision.

Criteria for Success

The client should have the ability to add, edit, and delete the following items:

- Students
- Classes
- assignments

The program should also calculate grades based on the inputted data (as a percentage).

A database and graphical user interface should be the interactive visual aspect of the program.

Class assigner should:

- Add classes
- Remove classes

Finally, in order for easier interaction inside the program, a tool to look for students' grades and information based on their name should be implemented. (ie. search bar)

Criterion B: ROT and Design

(all diagrams/explanations are created through Google Drawings)

RECORD OF TASKS:

Task Number	Planned action	Planned outcome	Estimate time	Target completion date	Criterion
1	Preliminary discussion with Bonita Keller (client)	Bonita explained the problem he is facing and the basic requirements he believes my application should have to solve it	15 minutes	September 17, 2021	A
2	Initial discussion with advisor-Comp Sci teacher (Mr. Vasquez)	After completion of Criterion A, Mr. Vasquez approved of my ideas for the application and provided me with	10 minutes	October 5, 2021	A

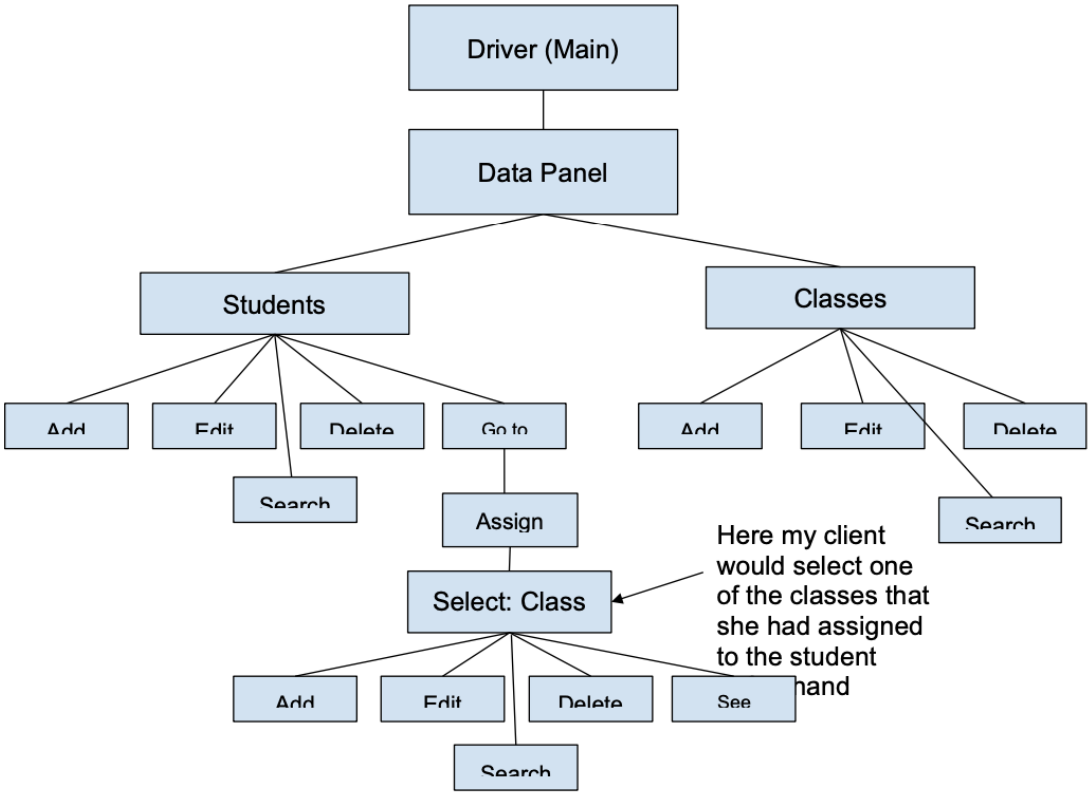
		resources to gain further knowledge			
3	Second discussion with Bonita	Solutions discussed, possible features of applications discussed, basic outline created	15 minutes	October 8, 2021	A
4	Third discussion with Bonita	Programming language and database program chosen (Java OOP, SQLite), outcomes and solutions agreed on	15 minutes	October 22, 2021	A
5	Begin/Finish Criterion A	Define criteria for success	2 weeks	October 31, 2021	A
6	Brainstorm possible design	Develop an idea for the basic structure and design of the application	1 week	October 31, 2021	A/B
7	Design flow charts/diagrams	Construct diagrams and flowcharts to help represent the process of the program and how it will function	2 weeks	November 12, 2021	B
8	Develop pseudocode	Construct pseudocode to help represent how the code may be implemented in the final product	1 week	November 19, 2021	B
9	Begin/Finish	Begin and	8 weeks	February 1,	C

	Software Design	complete software		2022	
10	Fix bugs, increase efficiency	After the first draft of the software is completed, test all features and fix any errors	1 week	February 8, 2022	C
11	Test Software	Using the test criteria below, test the software and ensure it fills the client's needs	1 week	February 15, 2022	C
12	Deliver final product	Deliver the final product to the client for testing	15 minutes	February 28, 2022	E
13	Final interview with client	Interview with the client after they test the product for feedback	15 minutes	March 11, 2022	E
14	Test Software	Using the test criteria below, test the software and ensure it fills the client's needs	1 week	March 18, 2022	C
15	Create video demonstration	Construct a video to represent the functionality of the application	1-2 weeks	March 31, 2022	D
16	Reflect	Reflect on overall product, possible improvements, etc.	1 day	March 31, 2022	E

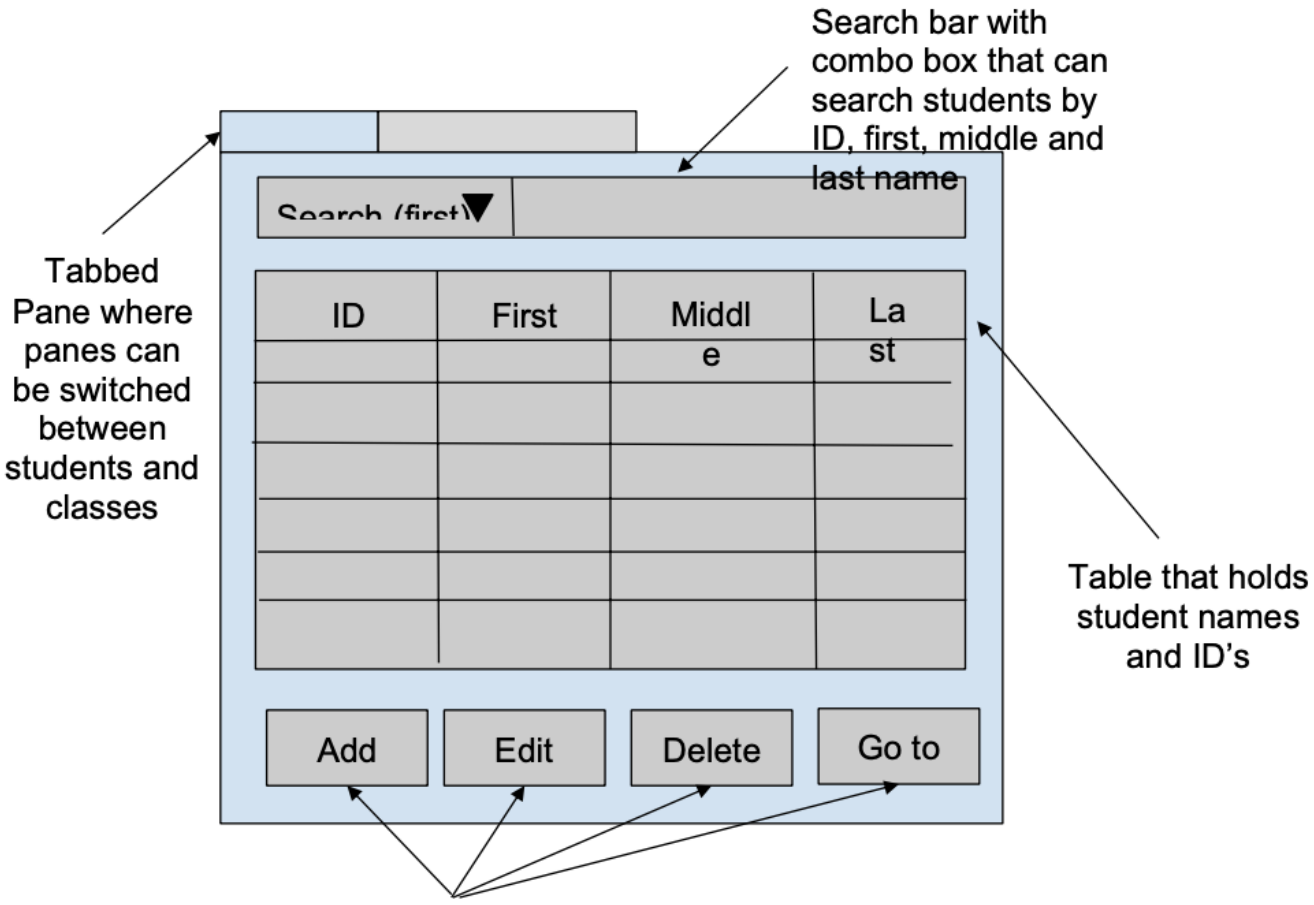
TEST CRITERIA:

Action to Test	Test Method
Program can successfully create database	Check the files of the program to make sure that a "database.db" file is created
A graphical user interface that properly represents all the information	Check all the information from the database if being presented in the GUI. Test all the buttons and features in the GUI to make sure that they are successfully working.
Program can add, edit and delete students	Create a dummy student and see if it appears in the student table of the program. Edit dummy students to see if changes are made in the table. Delete dummy student to check if it was deleted from the table. Use an external program (DB Browser for SQLite) that can open database files to check if it was successfully added, edited, and deleted.
Program can add, edit and delete classes	Create a dummy class and see if it appears in the class table of the program. Edit dummy class to see if changes are made in the table. Delete dummy class to check if it was deleted from the table. Use an external program (DB Browser for SQLite) that can open database files to check if it was successfully added, edited, and deleted.
Program can assign classes for each individual student	Add a few classes and students to the program. Then go to the student's page to add a few classes. Check the combo box in the student page frame if it was successfully added. Remove some classes then see if combo box updates. Use an external program (DB Browser for SQLite) that can open database files to check if assigned classes were successfully added or removed from the database.
Client will be able to add edit and delete assignments	Select a class in the student page frame then create a dummy assignment and see if it appears in the assignment table of the program. Edit dummy assignment to see if changes are made in the table. Delete dummy assignment to check if it was deleted from the table. Use an external program (DB Browser for SQLite) that can open database files to check if it was successfully added, edited, and deleted.
Grade Calculator that will calculator a student's percentage in the class and grade	Add a few assignments for a selected class, then see if class percentage is successfully calculated and class grade falls in the proper range.
Database is used to present information back to the program when it is restarted	Add a few students, classes, and assignments to the program. Close the program then run the program again to check all the tables to see if all the rows are still there.
Program a search tool to find a student/class/parent	Add multiple students, classes, and assignments to the database. Use the search tool in each to see if the correct student comes up under the respective search conditions.

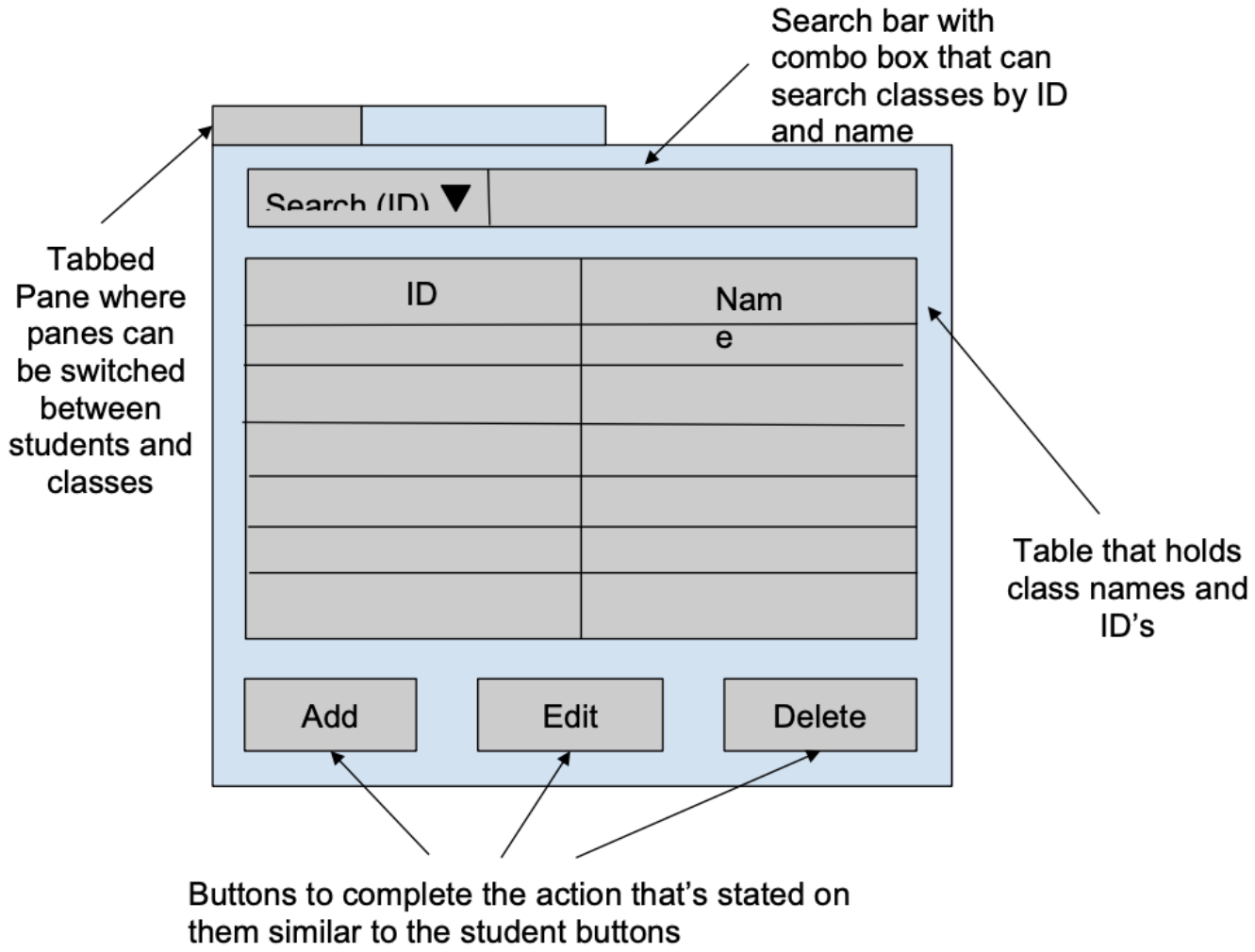
Site Plan, Wireframes:


Driver (Main)
Data Panel
Students
Classes
Add
Edit
Delete
Go to
Search
Add
Edit
Delete
Search
Assign
Select: Class
Here my client would select one of the classes that she had assigned to the student
Add
Edit
Delete
See
Search

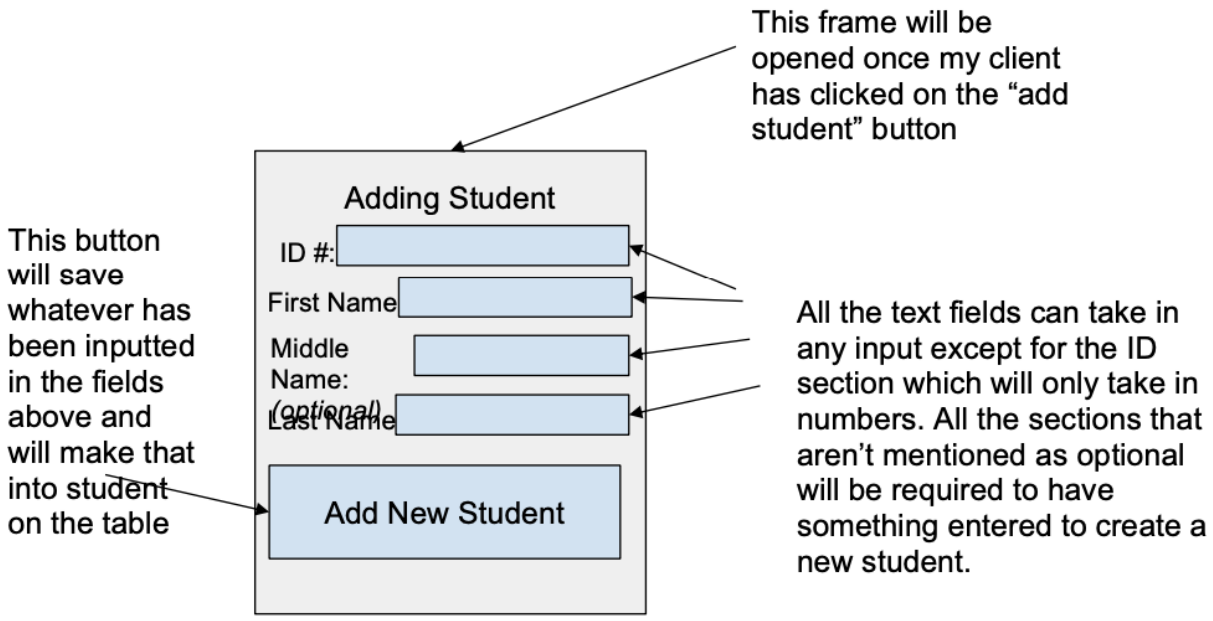
Main Menu for students:



Main menu for Classes:

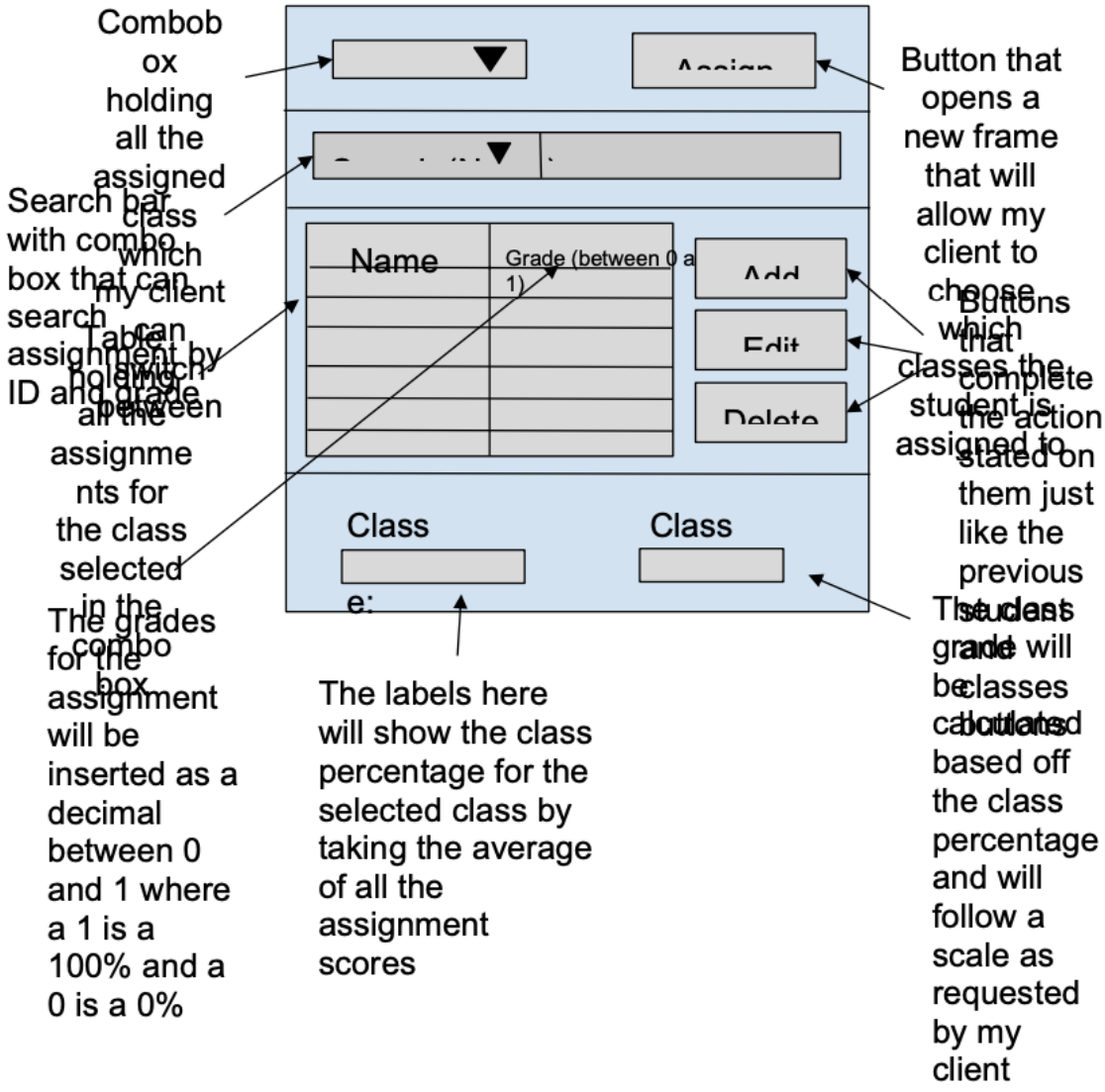


Adding/Editing Menu (example of adding student menu below):



My program will also include an editing students menu, which will have a panel similar to the one above except that it will pre-fill the text fields with the student’s information, and the button name will be changed to “Save Changes.” Just as there’s an adding and editing students menu, those menus will also be made for classes and assignments as well which will follow the same format, but accommodate for the columns that are present in each.

Student Page Menu:



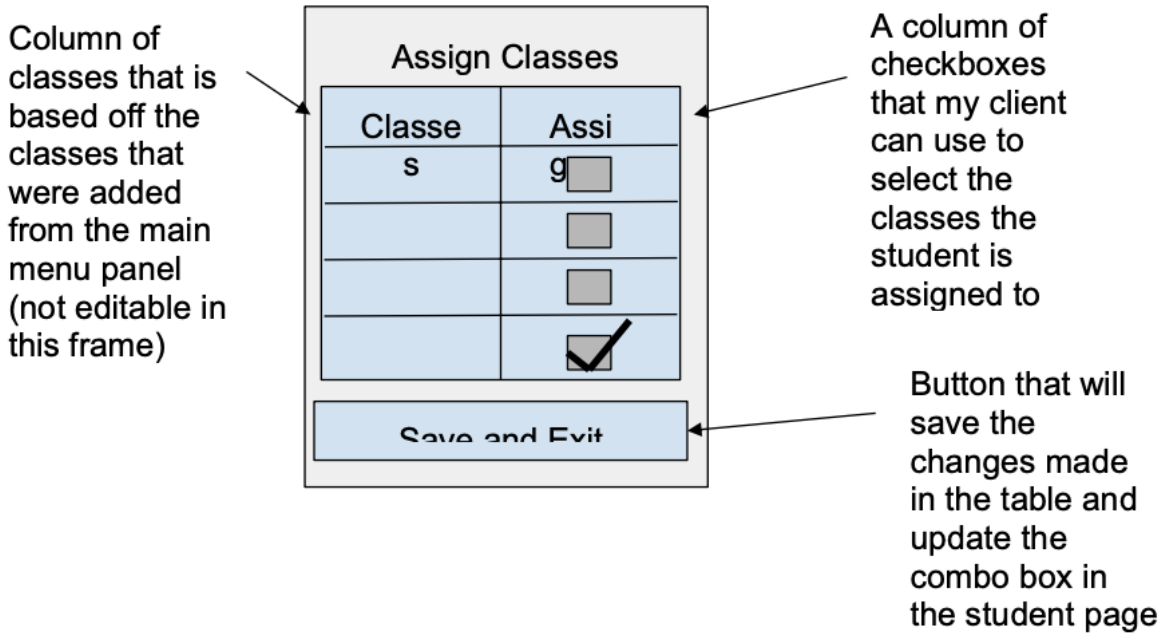
Scale of class grades for student page menu:

Percentage Range	Grade
100% - 90%	A
89% - 80%	B
79% - 70%	C
69 - 60%	D

59% or below	F

Each student that is inserted into the program will be able to have their own student page in which my client can customize the classes that they are assigned along with adding the assignments and grades they've received.

Assign Classes Menu:



This frame is accessed by clicking on the "assign classes to student button" from the student page. Here my client will be able to select the classes she wants to assign to the student or if she had previously assigned students before, will show the classes that have already been assigned.

UML:


SQLite
+ buildDatabase ()
+ DatabaseToJTables()
+ addStudentToDatabse
(int id, String first, String middle, String last)
+ deleteStudentFromDatabase (String id)
+ editStudentFromDatabase
(int originalID, int newID, String first, String middle, String last)
+ addClassToDatabase (int id, String name)
+ deleteClassFromDatabse(String id)
+ editClassFromDatabase
(int originalID, int newID, String name)
+ getAndAddClassNamesFromDatabase(JTable table)
+ saveClassSelectionToDatabase(JTable table): ArrayList<String>
+ getAssignedClasses (int id): ArrayList<String>
+ addAssignmentToDatabase
(String classSelected, String name, double grade)
+ deleteAssignmentFromDatbase
(String classSelected, String name, double grade)
+ editAssignmentFromDatabase
(String classSelected, String name, double grade, String assignmentName)
+ assignmentDatabasetoJTable
(JTable table, String classSelected)
AddingClassFrame
- addPanel: JPanel
- id: JLabel
- name: JLabel
- idTextField: JTextField
- nameTextField: JTextField
- addNewClassButton: JButton
- initComponents()
- addNewClassButtonActionPerformed
(ActionEvent evt)
EditingClassFrame
- addPanel: JPanel
- id: JLabel
- name: JLabel
- idTextField: JTextField
- nameTextField: JTextField
- saveChangesButton: JButton
- initComponents()
- saveChangesButtonActionPerformed
(ActionEvent evt)
+ getIdTextField(): JTextField
+ getNameTextField(): JTextField
+ setIdTextField(JTextField idTextField)
+ setNameTextField(JTextField NameTextField)
AddingAssignmentFrame
- addPanel: JPanel
- name: JLabel
- grade: JLabel
- nameTextField: JTextField
- gradeTextField: JTextField
- addNewAssignmentButton: JButton
- currentClass: String
- initComponents()
- addNewAssignmentButtonActionPerformed
(ActionEvent evt)
Driver
panel: DataPanel
+ main(String[] args)
DataPanel
- tabbedPane: JTabbedPane
- studentSplitePane: JSplitPane
- classSplitePane: JSplitPane
- studentScrollPane: JScrollPane
- classScrollPane: JScrollPane
- studentTable: JTable
- ClassTable: JTable
- studentButtonsPanel: JPanel
- classButtonsPanel: JPanel
- addStudentButton: JButton
- editStudentButton: JButton
- deleteStudentButton: JButton
- goToStudentPageButton: JButton
- addClassButton: JButton
- editClassButton: JButton
- deleteClassButton: JButton
- originalID: int
- initComponents()
- addingStudentButtonActionPerformed (ActionEvent evt)
+ addStudentRowToJTable
(Object[] dataRow)
- editStudentButtonActionPerformed (ActionEvent evt)
+ editStudentRowOnJTable
(int id, String first, String middle, String last)
- deleteStudentButtonActionPerformed (ActionEvent evt)
- addingClassButtonActionPerformed (ActionEvent evt)
+ addClassRowToJTable
(Object[] dataRow)
- editClassButtonActionPerformed (ActionEvent evt)
+ editClasstRowOnJTable
(int id, String name)
- deleteClassButtonActionPerformed (ActionEvent evt)
- goToStudentPageButtonActionPerformed (ActionEvent evt)
- getOriginalID(): int
- setOriginalID(int originalID)
- centerTables()
AssingingClassesFrame
- assigningClassesPanel: JPanel
- assignClassesSplitPlane: JSplitPlane
- JTableScrollPlane: JScrollPlane
- assignClassesJTable: JTable
- saveButton: JButton
- initComponents()
- saveButtonActionPerformed(ActionEvent evt)
+ getAssignClassesJTable(): JTable
+ setAssignClassesJTable(JTable assignClassesJTable)
EditingAssignmentFrame
- addPanel: JPanel
- name: JLabel
- grade: JLabel
- nameTextField: JTextField
- gradeTextField: JTextField
- saveChangesButton: JButton
- currentClass: String
- assignmentName: String
- initComponents()
- saveChangesButtonActionPerformed
(ActionEvent evt)
+ getGradeTextField(): JTextField
+ getNameTextField(): JTextField
+ setGradeTextField(JTextField gradeTextField)
+ setNameTextField(JTextField nameTextField)
AddingStudentFrame
- addPanel: JPanel
- id: JLabel
- first: JLabel
- middle: JLabel
- last: JLabel
- idTextField: JTextField
- firstTextField: JTextField
- middleTextField: JTextField
- lastTextField: JTextField
- addNewStudentButton: JButton
- initComponents()
- addNewStudentButtonActionPerformed
(ActionEvent evt)
EditingStudentFrame
- editPanel: JPanel
- id: JLabel
- first: JLabel
- middle: JLabel
- last: JLabel
- idTextField: JTextField
- firstTextField: JTextField
- middleTextField: JTextField
- lastTextField: JTextField
- saveChangesButton: JButton
- initComponents()
- saveChangesButtonActionPerformed
(ActionEvent evt)
+ getIDTextField(): JTextField
+ getFirstTextField(): JTextField
+ getMiddleTextField(): JTextField
+ getLastTextField(): JTextField
+ setIDTextField(JTextField idTextField)
+ setFirstTextField(JTextField firstTextField)
+ setMiddleTextField(JTextField middleTextField)
+ setLastTextField(JTextField lastTextField)
StudentPageFrame
- mainSplitPane: JSplitPlane
- JTableAndGradesSplitPane: JSplitPlane
- JTableSplitPlane: JSplitePlane
- classSelectionPanel: JPanel
- bottomPanel: JPanel
- JTablePanel: JPanel
- JTableButtonsPanel: JPanel
- gradesPanel: JPanel
- selectClassLabel: JLabel
- classSelectionComboBox: JComboBox
- assignClassesButton: JButton
- assignmentJTableScrollPlane: JScrollPlane
- assignmentJTable: JTable
- addAssignmentButton: JButton
- editAssignmentButton: JButton
- deleteAssignmentButton: JButton
- classPercentageLabel: JLabel
- classGradeLabel: JLabel
- classPercentageTextField: JTextField
- classGradeTextField: JTextField
- initComponents()
- initializeComboBox(ArrayList<String> classList)
- classSelectionComboBoxActionPerformed
(ActionEvent evt)
- assignClassesButtonActionPerformed
(ActionEvent evt)
- addAssignmentButtonActionPerformed
(jActionEvent evt)
- addAssignmentRowToJTable (Object[] row)
- editAssignmentButtonActionPerformed
(jActionEvent evt)
- editAssignmentRowOnJTable (Object[] row)
- deleteAssignmentButtonActionPerformed
(ActionEvent evt)
+ getAssignmentJTable(): JTable
- calculateClassPercentage(JTable table): double
- displayPercentageAndGrade()

Classes:

SQLite:
Here I will have most of my sqlite commands in a single class that I can reference through my program.

Driver:
This is where the DataPanel will be made which is the main panel of my program with the class including the main method.

DataPanel:
This class will make the tabbed pane from the first two panel designs with everything inside.

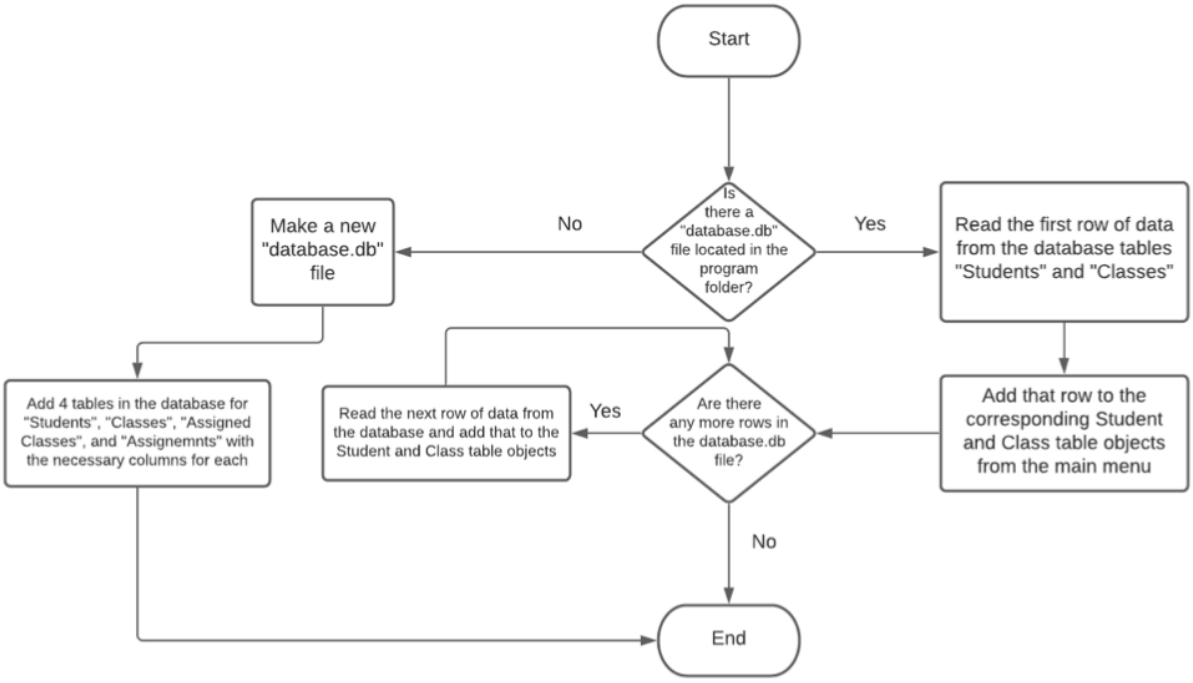
All “Adding” or “Editing” Frame classes:
The purpose of all these classes is to make their respective frames and save the changes they made to the database from what was entered in the text fields and update the respective tables. These classes generally have the same type of instance variables but are just dependent on what JTables they are working for. And the main difference between the adding and editing is the inclusion of getters and setters for the editing class.

StudentPageFrame:
This class will make a fully functioning student page menu where my client can see that individual’s assignments and grades.

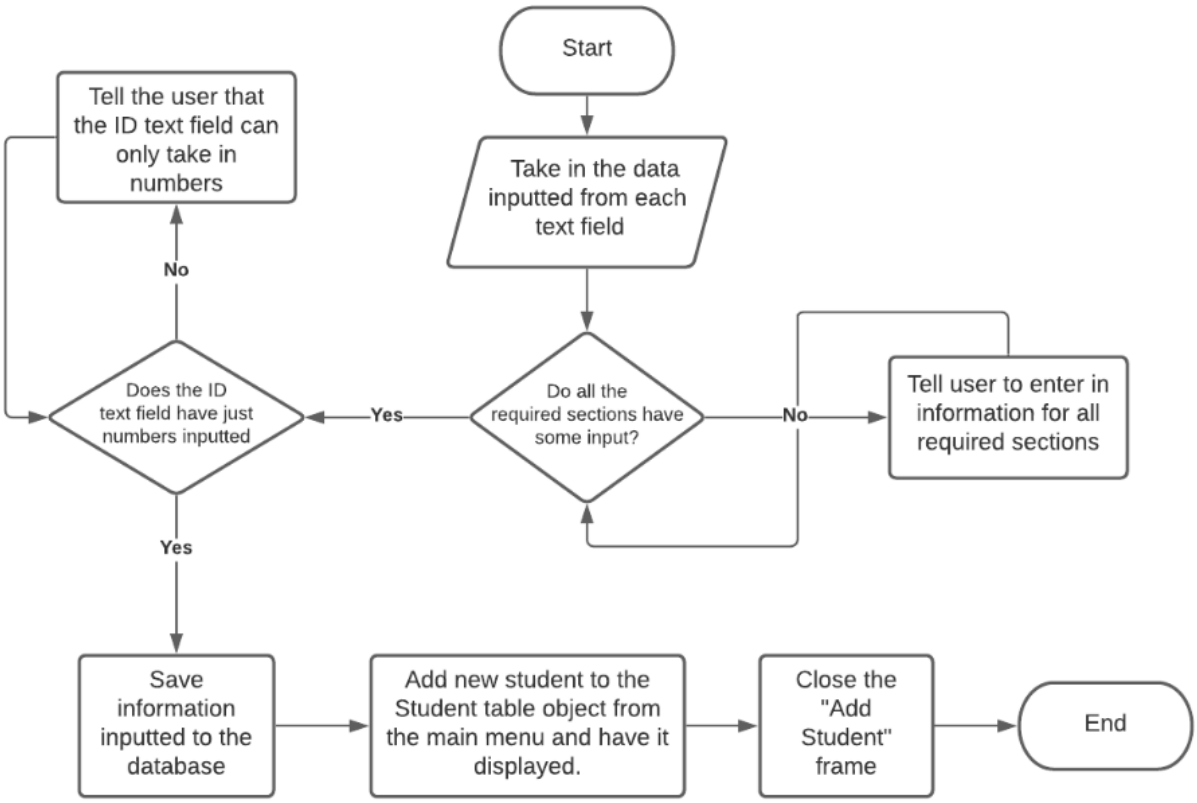
AssigningClassesFrame:
This class will make the assigned classes menu that my client can interact with to assign or remove classes from a student.

FLOWCHARTS:

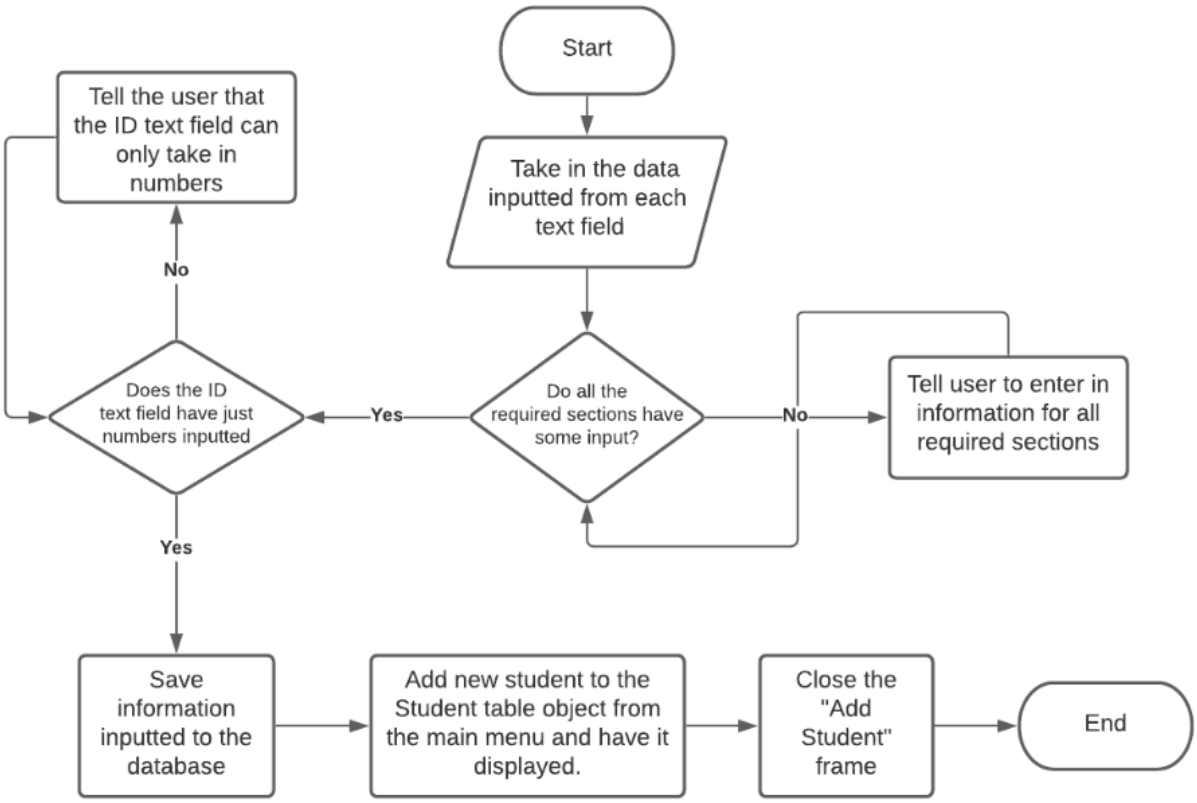
The program first starting up (database initialization):



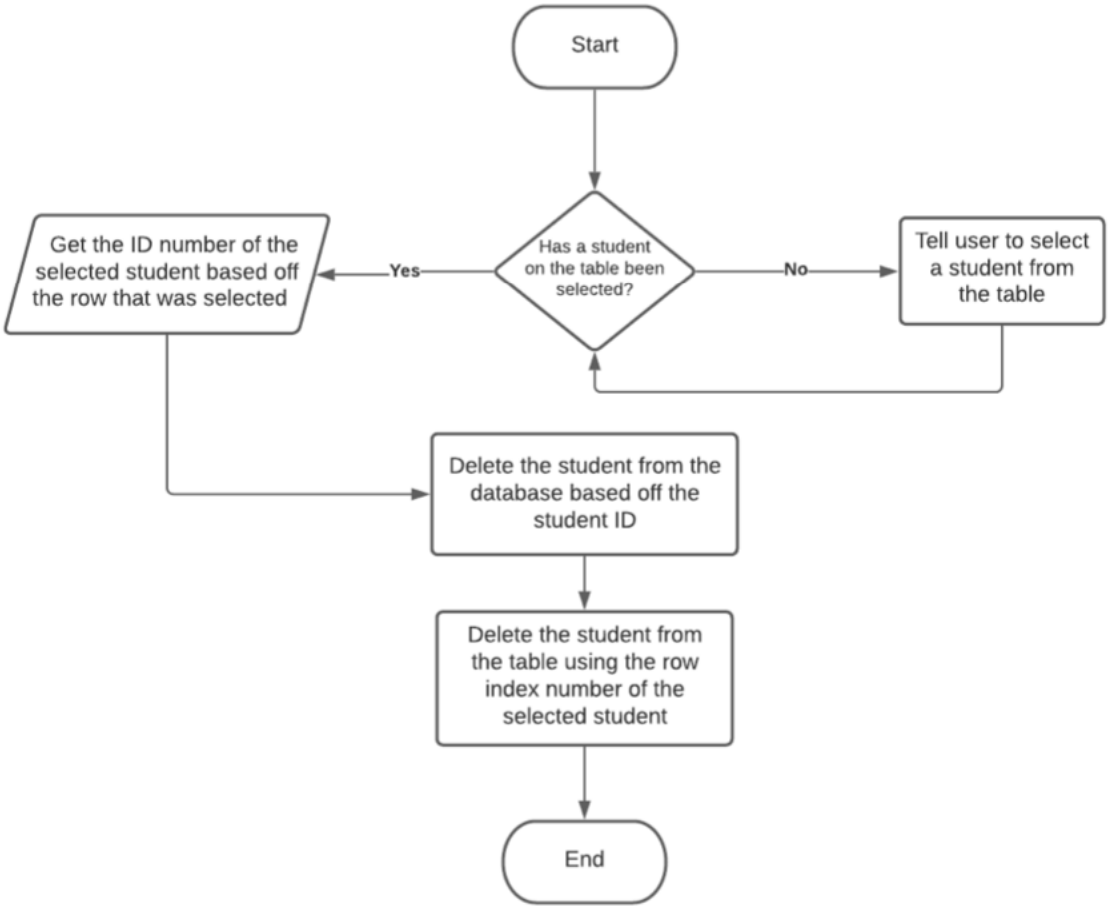
Adding students to program once the “Add New Student” button has been clicked:



Adding students to program once the “Add New Student” button has been clicked:



Deleting students from the program once the “Delete Student” button has been clicked:



Although the last three flow charts only show some of the adding, editing, and deleting for students, the classes and assignments follow a similar pattern, accommodating once again for the columns that are present. The assignments will also have to keep track of what student and class they are a part of when adding, editing, or deleting it from the database.

Assigning classes to the student once the “Save and Exit” button is clicked:

Criterion C: Development

(all diagrams/explanations are created through Google Drawings; screenshots are from actual interface and IDE)

First, some of the techniques I used included arrays, array lists, encapsulation, try-catch blocks, a graphical user interface, static variables and methods, and throws / SQL.

a. Graphical User Interface

Main Frame

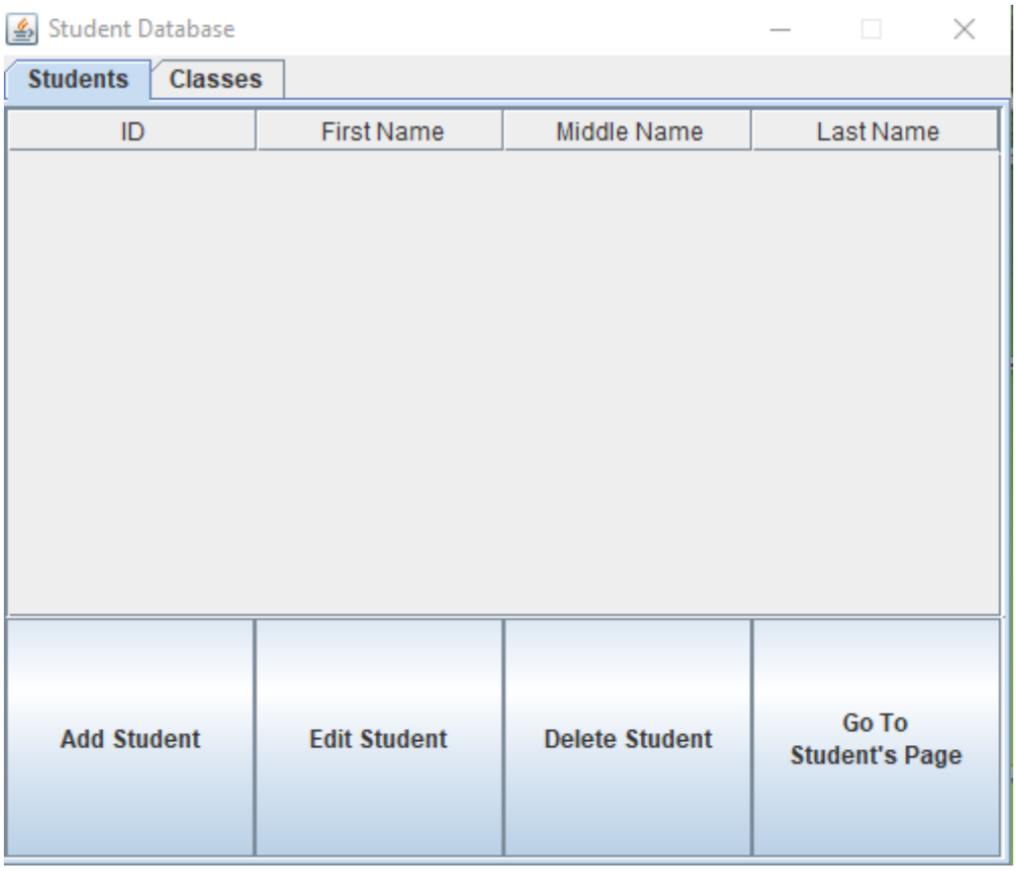


Figure 1. Panel of student table and corresponding buttons

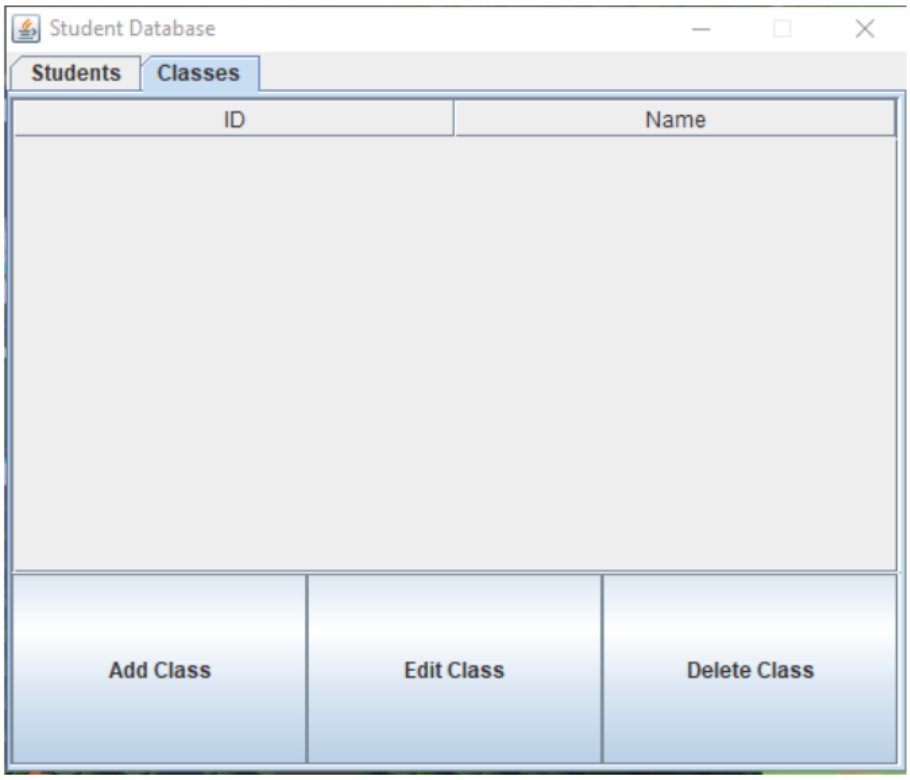


Figure 2. Panel of class table and corresponding buttons

```
tabbedPanes = new javax.swing.JTabbedPane();
studentSplitPane = new javax.swing.JSplitPane();
studentScrollPane = new javax.swing.JScrollPane();
StudentTable = new javax.swing.JTable();
studentButtonsPanel = new javax.swing.JPanel();
addingStudentButton = new javax.swing.JButton();
editStudentButton = new javax.swing.JButton();
deleteStudentButton = new javax.swing.JButton();
goToStudentPageButton = new javax.swing.JButton();
classSplitPlane = new javax.swing.JSplitPane();
classScrollPanel = new javax.swing.JScrollPane();
classTable = new javax.swing.JTable();
classButtonPanel = new javax.swing.JPanel();
addClassButton = new javax.swing.JButton();
editClassButton = new javax.swing.JButton();
deleteClassButton = new javax.swing.JButton();
```

Figure 3. Creation of GUI components for figures 1 and 2 except for the JPanel which is extended from the class

By using a JTabbedPane in figures 1 and 2, the user can switch between the following screens at any point without having to add additional buttons. By using a JSplitPlane inside each of the tabbed panes, I could set the divider line horizontally so that the top stores a JScrollPlane which stores a JTable, and the bottom stores the JButtons.

Student Page Frame

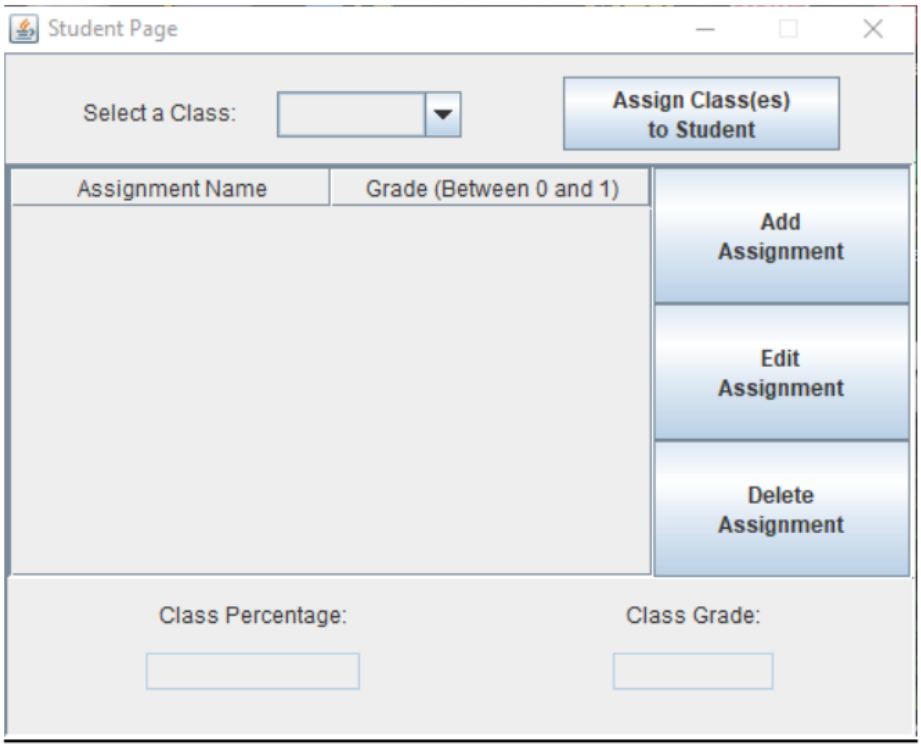


Figure 4. Student page frame accessed by "Go to Student's Page" button once a student is made and selected

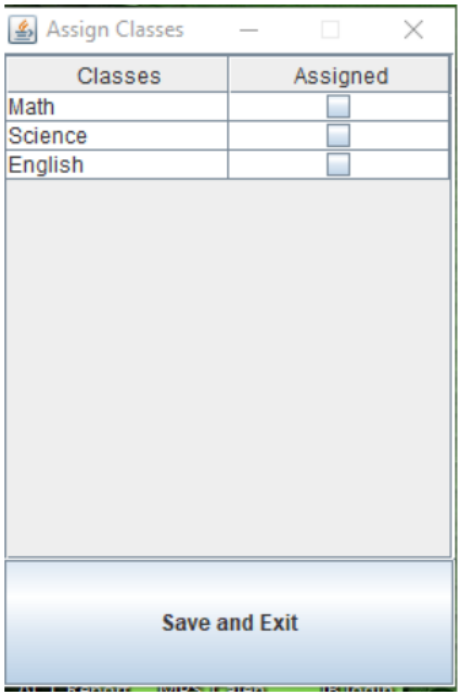


Figure 5. Assign classes frame accessed by clicking "Assign Class(es) to Student" button in figure 4

In figure 4, just like the previous figures has a JTable and JButtons for assignments, but also included a JComboBox to select the classes that were assigned to the student from the JFrame in figure 5. By adding assignments to the JTable in figure 4, the JLabels under "Class Percentage" and "Class Grade" will show the average percentage and letter grade based on all of them.

Action Listeners

By clicking on the "Add Student" button in figure 1, it will open another JFrame for which the user can input the student's information. But an ActionListener method will be required to create that frame once it is clicked.

```
addingStudentButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        addingStudentButtonActionPerformed(evt);
    }
});
```

```
private void addingStudentButtonActionPerformed(java.awt.event.ActionEvent evt) {
    AddingStudentFrame addFrame = new AddingStudentFrame();
    addFrame.setVisible(true);
}
```

Figure 6 & 7. adding ActionListener to JButton and creating method to open another frame

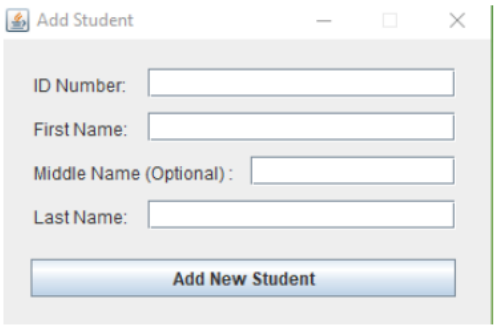


Figure 8. JFrame to add a student (Opened by “Add Student Button”)

The user can now add the following information into the JTextFields on figure 8 to add a new student to the program. But to successfully add the students, the program will need to use a try-catch block in order to prevent any incorrect inputs.

b. Try-Catch blocks

```
82      try {
83
84       int savedID = Integer.parseInt(idTextField.getText());
85
86       if (savedID <= 0)
87       {
88           JOptionPane.showMessageDialog(null, "Please enter an ID number between 1 and 999999999");
89           return;
90       }
91
92       String savedFirst = firstTextField.getText();
93       String savedMiddle = middleTextField.getText();
94       String savedLast = lastTextField.getText();
95
96       SQLite.addStudentToDatabse(savedID, savedFirst, savedMiddle, savedLast);
97
98       Object[] row = {savedID, savedFirst, savedMiddle, savedLast};
99
100      DataPanel.addStudentRowToJTable(row);
101      dispose();
102      }
103
104      catch(NumberFormatException e) {
105          JOptionPane.showMessageDialog(null, "Please Enter Digits Only For The ID");
106      } catch (ClassNotFoundException | SQLException ex) {
107          Logger.getLogger(AddingStudentFrame.class.getName()).log(Level.SEVERE, null, ex);
108      }
```

Figure 9. Try catch block from the ActionListener method on the “Add New Student” button in figure

Starting from Line 84 in figure 9, in order to get the ID Number from the corresponding JTextField, idTextField.getText() is called which returns the value in the field as a string. But to convert it into an integer, I would have to call the parseInt() method from the Integer wrapper class. If the textField cannot be converted into the integer due to letters or other symbols being present, it will call a NumberFormatException error which I can use in my catch to create a JOptionPane telling the user to input “Digits Only.” Although I can still save the textID as a string and avoid the use of a try-catch, I still need it as an int to later pass it to my DataPanel method in line 100 to add the ID to the student JTable.

c. SQL / throws

```
54  public static void addStudentToDatabse (int id, String first, String middle, String last) throws ClassNotFoundException, SQLException
55  {
56      Class.forName("org.sqlite.JDBC");
        Connection conn = DriverManager.getConnection("jdbc:sqlite:Database.db");
58
59      String sqlStatement = "INSERT INTO Students (ID,First,Middle,Last) " +
60      "VALUES (" + id + "," + "'" + first + "'" + "," +"'" + middle + "'" + "," + "'" + last + "'" + ");";
61
62      Statement state = conn.createStatement();
63      state.executeUpdate(sqlStatement);
64
65      state.close();
66      conn.close();
67  }
```

Figure 10. SQL method that add students into database

```
69      public static void deleteStudentFromDatabase (String id) throws ClassNotFoundException, SQLException
70      {
71          Class.forName("org.sqlite.JDBC");
72          Connection conn = DriverManager.getConnection("jdbc:sqlite:Database.db");
73
74          String sqlStatement = "DELETE FROM Students WHERE ID = " + id + ";";
75
76          Statement state = conn.createStatement();
77          state.executeUpdate(sqlStatement);
78
79          state.close();
80          conn.close();
81
82      }
83
84      public static void editStudentFromDatabase (int originalID, int newID, String first, String middle, String last) throws ClassNotFoundException
85      {
86          Class.forName("org.sqlite.JDBC");
87          Connection conn = DriverManager.getConnection("jdbc:sqlite:Database.db");
88
89          String sqlStatement = "UPDATE Students "
90                              + "SET ID = " + newID
91                              + ", First = " + "'" + first + "'"
92                              + ", Middle = " + "'" + middle + "'"
93                              + ", Last = " + "'" + last + "'"
94                              + " WHERE ID = " + originalID + ";";
95
96          Statement state = conn.createStatement();
97          state.executeUpdate(sqlStatement);
98
99          state.close();
100         conn.close();
101     }
102
```

Figure 11. SQL methods to edit and delete students from database

In looking at figure 9, on line 96 a SQLite.addStudentToDatabase() method is called which took all the variables I saved from the JTextFields, and puts them into the database under a “Students” table which has the same columns as the JTable seen in figure 1. In figure 10, adding the students to the database will require concatenation inside the sqlStatement String which will then be executed by the Statement object. Figures 10 and 11 should give a general idea behind the order of how most SQL methods are made in this program. Even looking at the throws of each method, they all have the same throws exceptions when trying to connect to the Java JDBC API to use SQLite (ClassNotFoundException) and creating the connection to the database file (SQLException).

Restarting Program

```
15      public static void DatabaseToJTables (JTable table1, JTable table2) throws ClassNotFoundException, SQLException
16      {
17
18          Class.forName("org.sqlite.JDBC");
19          Connection conn = DriverManager.getConnection("jdbc:sqlite:Database.db");
20
21          String sqlStatement1 = "SELECT * FROM Students";
22          String sqlStatement2 = "SELECT * FROM Classes";
23
24          Statement state1 = conn.createStatement();
25          Statement state2 = conn.createStatement();
26
27          ResultSet rs1 = state1.executeQuery(sqlStatement1);
28          ResultSet rs2 = state2.executeQuery(sqlStatement2);
29
30          DefaultTableModel modelOne = (DefaultTableModel)table1.getModel();
31          DefaultTableModel modelTwo = (DefaultTableModel)table2.getModel();
32
33          while(rs1.next())
34              {
35                  int id = rs1.getInt("ID");
36                  String first = rs1.getString("First");
37                  String middle = rs1.getString("Middle");
38                  String last = rs1.getString("Last");
39
40                  Object[] databaseRow1 = {id, first, middle, last};
41                  modelOne.addRow(databaseRow1);
42              }
43
44          while(rs2.next())
45              {
46                  int id = rs2.getInt("ID");
47                  String name = rs2.getString("Name");
48
49                  Object[] databaseRow2 = {id, name};
50                  modelTwo.addRow(databaseRow2);
51              }
52      }
```

Figure 12. SQL method that gathers all the data from the database, and updates the JTables with it.

Normally used when the project first starts

```
public class DataPanel extends JPanel {

    public DataPanel() throws ClassNotFoundException, SQLException{
        initComponents();
        centerTables();
        SQLite.DatabaseToJTables(StudentTable, classTable);
    }
```

Figure 13. Method from figure 12 being called into the constructor of the JPanel in figures 1 and 2.

A big advantage about using a database to store all the information, is that once the program is closed, the information stored when using the program is saved to the database. So when the

program runs again, I can call a method in the constructor of the JPanel to send all the information to the student and class JTables.

d. Encapsulation

```
    private javax.swing.JTextField firstTextField;
    private java.awt.Label id;
    private javax.swing.JTextField idTextField;
    private java.awt.Label last;
    private javax.swing.JTextField lastTextField;
```

Figure 14. Private JTextField variables from EditingSutdentFrame class

```
    private void editStudentButtonActionPerformed(java.awt.event.ActionEvent evt) {
198         if (StudentTable.getSelectionModel().isSelectionEmpty())
199         {
200             JOptionPane.showMessageDialog(null, "Please Select a student from the table");
201             return;
202         }
203
204         int rowIndex = StudentTable.getSelectedRow();
205
206         Object id = StudentTable.getModel().getValueAt(rowIndex, 0);
207
208         DataPanel.setOriginalID((Integer)id);
209
210         Object first = StudentTable.getModel().getValueAt(rowIndex, 1);
211         Object middle = StudentTable.getModel().getValueAt(rowIndex, 2);
212         Object last = StudentTable.getModel().getValueAt(rowIndex, 3);
213
214         EditingStudentFrame editFrame = new EditingStudentFrame();
215
216         JTextField idField = editFrame.getIdTextField();
217         JTextField firstField = editFrame.getFirstTextField();
218         JTextField middleField = editFrame.getMiddleTextField();
219         JTextField lastField = editFrame.getLastTextField();
220
221         idField.setText(Integer.toString((int)id));
222         firstField.setText((String)first);
223         middleField.setText((String)middle);
224         lastField.setText((String)last);
225
226         editFrame.setIdTextField(idField);
227         editFrame.setFirstTextField(firstField);
228         editFrame.setMiddleTextField(middleField);
229         editFrame.setLastTextField(lastField);
230
231         editFrame.setVisible(true);
232     }
```

Figure 15. Use of encapsulation on the JTextField variables in figure 14 for the ActionListener method on the “Edit Student” button

Once the editing student JFrame is made when the “edit button” is clicked, a convenient feature I added to the frame is to have it set up where the information on the selected row in the student JTable shows up again in the JTextFields. But in order to do that, getters and setters would

have to be made for each of the JTextField objects so that in the ActionListener method can get each of the fields (figure 15. lines 216-219), set the text inside each field equal to the info on the selected row (221-224), and set it back to the text fields in the frame (226-229).

e. Static Variables and Methods

Throughout my project, a struggle I faced was updating my main visual objects in my program like my JTables and JTextFields which often had to be done from different classes and JFrames. In doing some research I found that a solution to that was to make the objects static so I don't have to reference an instance of those objects every time when trying to change it (1BestCsharp blog, 2019).

```
private static javax.swing.JTable classTable;

private static javax.swing.JTable StudentTable;

private static javax.swing.JTable assignmentJTable;
private javax.swing.JScrollPane assignmentJTableScrollPane;
private javax.swing.JPanel bottomPanel;
private java.awt.Label classGradeLabel;
private static javax.swing.JTextField classGradeTextField;
private java.awt.Label classPercentageLabel;
private static javax.swing.JTextField classPercentageTextField;
private static javax.swing.JComboBox<String> classSelectionComboBox;
```

Figure 16-18. list of GUI static variables objects used in product

```
public static void addStudentRowToJTable(Object[] dataRow)
{
    DefaultTableModel model = (DefaultTableModel) StudentTable.getModel();
    model.addRow(dataRow);
}
```

Figure 19. static method that uses the student JTable to add rows to it (also referenced in line 100 in figure 7)

```
public static void editStudentRowOnJTable(int id, String first, String middle, String last)
{
    int rowIndex = StudentTable.getSelectedRow();
    DefaultTableModel model = (DefaultTableModel) StudentTable.getModel();
    model.setValueAt(id, rowIndex, 0);
    model.setValueAt(first, rowIndex, 1);
    model.setValueAt(middle, rowIndex, 2);
    model.setValueAt(last, rowIndex, 3);
}
```

Figure 20. static methods that edits a student row on the Jtable

```
public static void addClassRowToJTable (Object[] dataRow)
{
    DefaultTableModel model = (DefaultTableModel) classTable.getModel();
    model.addRow(dataRow);
}
```

```
public static void editClassRowOnJTable(int id, String name)
{
    int rowIndex = classTable.getSelectedRow();
    DefaultTableModel model = (DefaultTableModel) classTable.getModel();
    model.setValueAt(id, rowIndex, 0);
    model.setValueAt(name, rowIndex, 1);
}
```

Figure 21-22. Add and edit methods for classTable that complete the same task in figures 19 & 20

```
13  public class SQLite {
14
15      public static void DatabaseToJTables (JTable table1, JTable table2) throws ClassNotFoundException, SQLException
16  {...37 lines }
53
54      public static void addStudentToDatabse (int id, String first, String middle, String last) throws ClassNotFoundException, SQLException
55  {...10 lines }
65
66      public static void deleteStudentFromDatabase (String id) throws ClassNotFoundException, SQLException
67  {...11 lines }
78
79      public static void editStudentFromDatabase (int originalID, int newID, String first, String middle, String last) throws ClassNotFoundException, SQLException
80  {...14 lines }
94
95      public static void addClassToDatabase (int id, String name) throws ClassNotFoundException, SQLException
96  {...10 lines }
106
107     public static void deleteClassFromDatabse(String id) throws ClassNotFoundException, SQLException
108 {...9 lines }
117
118     public static void editClassFromDatabase(int originalID, int newID, String name) throws ClassNotFoundException, SQLException
119 {...12 lines }
131
132     public static void getAndAddClassNamesFromDatabase(JTable table) throws SQLException, ClassNotFoundException
133 {...17 lines }
150
151     public static ArrayList<String> saveClassSelectionToDatabase(JTable table) throws SQLException, ClassNotFoundException
152 {...47 lines }
199
200     public static ArrayList<String> getAssignedClasses (int id) throws ClassNotFoundException, SQLException
201 {...19 lines }
220
221     public static void addAssignmentToDatabase (String classSelected, String name, double grade) throws ClassNotFoundException, SQLException
222 {...14 lines }
236     public static void deleteAssignmentFromDatbase (String classSelected, String name, double grade) throws ClassNotFoundException, SQLException
237 {...10 lines }
247     public static void editAssignmentFromDatabase(String classSelected, String name, double grade, String assignmentName) throws ClassNotFoundException, SQLException
248 {...12 lines }
260
261     public static void assignmentDatabasetoJTable(JTable table, String classSelected) throws ClassNotFoundException, SQLException
262 {...19 lines }
281 }
282
```

Figure 23. List of all SQLite methods in SQLite class that are all static methods for quick access in other classes

f. Arrays and ArrayLists

In looking at line 98 in figure 9, the majority of the arrays I used in my program was to save the information collected by the JTextFields, put it in an object array, and send that to my addStudent method in figure 19 to update the student JTable. However, I did use an ArrayList to save the classes selected from figure 5, so they can be updated to the JComboBox in figure 4.

```
158
159     public static ArrayList<String> saveClassSelectionToDatabase(JTable table) throws SQLException, ClassNotFoundException
160     {
161         Class.forName("org.sqlite.JDBC");
162         Connection conn = DriverManager.getConnection("jdbc:sqlite:Database.db");
163         Statement state = conn.createStatement();
164         ArrayList<String> classList = new ArrayList<>();
165
166         for (int i = 0; i < table.getRowCount(); i++)
167         {
168             Object assigned = table.getValueAt(i, 1);
169
170             if (assigned == null) {
171
172             }
173
174             else if ((boolean)assigned == true)
175             {
176                 String classSelected = (String)table.getValueAt(i, 0);
177                 classList.add(classSelected);
178                 String sqlStatement = "SELECT ID FROM Classes WHERE Name = " + "'" + classSelected + "'";
179                 ResultSet rs = state.executeQuery(sqlStatement);
180                 int classID = rs.getInt(1);
181                 sqlStatement = "INSERT INTO Assigned_Classes (Student_ID, Class_ID)" +
182                                " VALUES (" + DataPanel.getOriginalID() + ", " + classID + ")";
183                 state.executeUpdate(sqlStatement);
184             }
185
186             else if ((boolean)assigned == false)
187             {
188                 String classSelected = (String)table.getValueAt(i, 0);
189                 String sqlStatement = "DELETE FROM Assigned_Classes WHERE Student_ID = " + DataPanel.getOriginalID() +
190                                       " AND Class_ID = (SELECT ID FROM Classes WHERE Name = '" + classSelected + "')";
191                 state.executeUpdate(sqlStatement);
192             }
193         }
194
195         String sqlStatement = "DELETE   FROM Assigned_Classes " +
196                               "WHERE    ROWID NOT IN" +
197                               "         (" +
198                               "         SELECT  min(ROWID)" +
199                               "         FROM    Assigned_Classes" +
200                               "         GROUP BY" +
201                               "                 Student_ID" +
202                               "         ,       Class_ID" +
203                               "         )";
204         state.executeUpdate(sqlStatement);
205         return classList;
206     }
```

Figure 24. SQL return method that saves the class selection from figure 4 into the ClassList ArrayList and gets returned at the end

Although figure 24 shows the use of an ArrayList, it also shows some of the more complex SQL methods that are used in my program. In my method, the parameter I have is JTable as this method is normally called to take in the JTable from figure 4. Then I check the boolean column (the second column) to see whether any of the classes were selected or not. A tricky thing about booleans in JTables, is that if a boolean cell is not clicked at all in the table, then the cell returns a null instead of a default boolean value of false. The only way a false is returned is if a boolean cell is clicked to turn true, then clicked again to make it false. So I have to first save the value at the cell as an object and check if it's null before I cast it into a boolean. The way the boolean column is set up can still be useful, as I can use the false as a way to check if an assigned class has been removed from the student and so I can then delete it from the database.

When it comes to the classList ArrayList, I can use it to add the classes under the condition that the boolean is true, save those classes to the database, and return that ArrayList to where it was called to set up the JComboBox from figure 4.

```
private void goToStudentPageButtonActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        if (StudentTable.getSelectionModel().isSelectionEmpty())
        {
            JOptionPane.showMessageDialog(null, "Please Select a student from the table");
            return;
        }

        int rowIndex = StudentTable.getSelectedRow();
        Object id = StudentTable.getModel().getValueAt(rowIndex, 0);
        setOriginalID((Integer)id);

        ArrayList<String> classList = SQLite.getAssignedClasses(originalID);
        StudentPageFrame pageFrame = new StudentPageFrame();
        StudentPageFrame.initializeComboBox(classList);

        pageFrame.setVisible(true);
    } catch (ClassNotFoundException | SQLException ex) {
        Logger.getLogger(DataPanel.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Figure 25. "Go To Student's Page" button ActionListener method that calls a SQLite class to retrieve the classes selected for the student to initialize the JComboBox

```
207     public static ArrayList<String> getAssignedClasses (int id) throws ClassNotFoundException, SQLException
208     {
209         ArrayList<String> classList = new ArrayList<>();
210 
211         Class.forName("org.sqlite.JDBC");
212         Connection conn = DriverManager.getConnection("jdbc:sqlite:Database.db");
213         Statement state = conn.createStatement();
214 
215         String sqlStatement = "SELECT Name FROM Classes WHERE ID IN" +
216         "(SELECT Class_ID FROM Assigned_Classes WHERE Student_ID = (" + id + "))";
217         ResultSet rs = state.executeQuery(sqlStatement);
218 
219         while(rs.next())
220         {
221             String classResult = rs.getString(1);
222             classList.add(classResult);
223         }
224 
225         return classList;
226     }
```

Figure 26. SQLite method called in figure 22 that gets the assigned classes from the student at the database

Here are a few more instances where an ArrayList is used, all with the same purpose of gathering a list of selected classes from the student and sending it to the ActionListener to initialize the JComboBox. Although the JComboBox gets updated every time the selection of classes is saved, it needs to get updated also when the frame is first made. That is where figure 25 plays a role, as it calls the SQLite class in figure 26, and sends an ArrayList of the selected classes to an initializeComboBox method that adds all the class names from the list into the JComboBox.

Criterion E: Evaluation

Meeting Success Criteria

MET:

✓ The client has the ability to add, edit, and delete the following items:
- Students
- Classes
- assignments

✓ The program is able to calculate grades based on the inputted data (as a percentage).

✓ A database and graphical user interface (JTable) is shown to be the interactive visual aspect of the program.

✓ Class assigner is able to

- Add classes
- Remove classes

NOT MET:

- **X** A tool to look for students' grades and information based on their name (ie. search bar)

Recommendations for Further Development

- Minor Improvements –

  - A better layout of the GUI as instead of having the JButtons open a new frame every time, they could instead place a new panel on top of the frame so my client can not worry about exiting out of multiple frames.

- Major Improvements –

  - An inclusion of a search bar in the program as it will make looking for students a lot easier when around 100+ students end up getting added to the program. This will also complete my last required success criteria.
  - Major bug fixes: Although all of the features of the program work individually, when I use them in conjunction with each other may cause a few errors (ex. changing the ID of a class when that class is already assigned to a student). By being able to fix those bugs, it will make the product a lot more usable for my client.

Client Feedback

After a meeting with my client, the client seemed satisfied with the results. Although my client enjoyed the product overall, there were a few errors I discussed with her to ensure that the product can reach its full potential. As stated previously, some major improvements to the program can be fixing the issue in which students can carry the same identification number. I also discussed with my client the

possibility of a search tool. The client provided me with similar feedback in which a search tool would most likely help her, especially once students start to learn in person more frequently. Overall, my client was satisfied with the result.