**Imported Libraries**

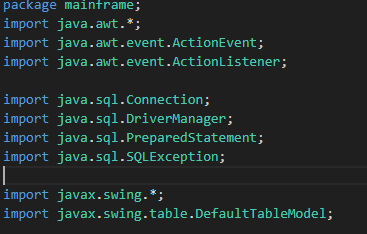


Figure 1: imported libraries

The code above is an example of all the libra used in the project. Javax.swing is used to create and manage GUI elements. Java.sql is used to manage the databases used to organize students and other items. Java.awt is used to obtain user inputs and allow java.swing elements to execute methods based on those inputs.

**Creating GUI**

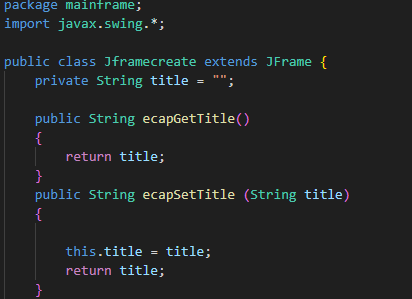


Figure 2: Importation of Swing, creation of main parent JFrame class and the encapsulation of the “title” variable.

This code imports the Java Swing library to create the class Jframecreate, which serves as the parent class to all the other Jframes. Because of its intended purpose as an overall parent class, the title variable is encapsulated. This ensures that the title can only be accessed through certain get()/set() methods, decreasing the chances of it being accidentally interfered with when creating or modifying child classes.

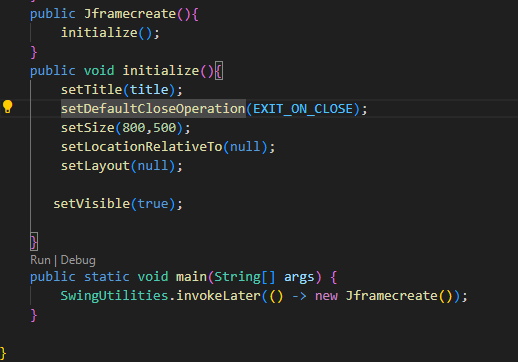


Figure 3: setting the bounds of the Jframe

The code above sets the bounds for the Jframe. By setting the layout to null, I can determine where elements are placed on the frame with more precision, allowing me to better meet the client’s need for a simple, easy-to-navigate GUI. Furthermore, allowing other Jframe classes to inherit the bounds of Jframecreate ensures that all pages have uniform dimensions. The invokeLater method ensures thread safety, making sure that the code only runs when it is needed.

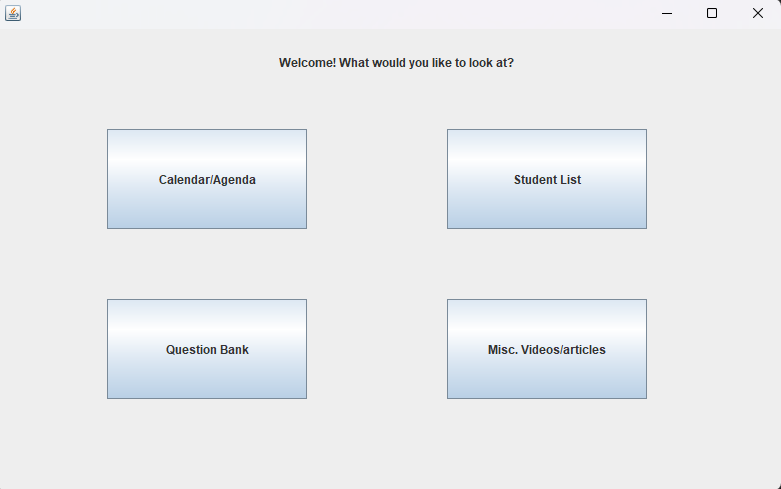


Figure 4: example of the Jframecreate childclass homeframe. All child classes are the same size and have same function(are deleted when the ‘X’ button is clicked)

**Obtaining User Input**

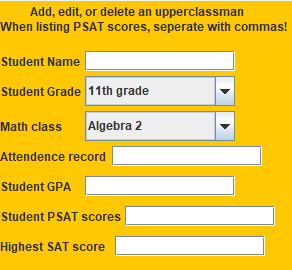


Figure 5: Gui created by figure 6 where the user can input student information

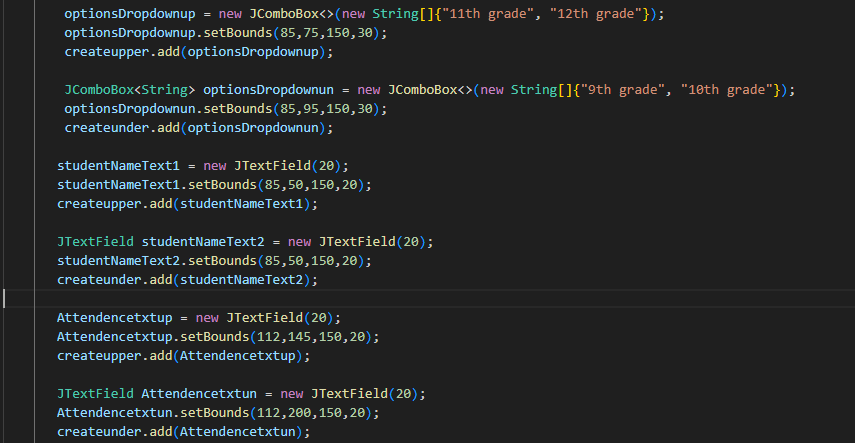


Figure 6: Creating elements for the Jpanels

The code above creates the text fields and dropdowns via which the client can provide information on the students. The use of dropdowns(Jcomboboxes) is highly practical, as it limits the chances of the user providing an invalid input. The setBounds methods in the code shows the value of not using a layout manager, as the elements can be placed in precise places to ensure ease of use and navigability.



Figure 7: Action listener and conditionals used to execute methods using user inputs.

Above, the actionlistener from Java.awt is used to detect user interaction with Jswing elements. To ensure readabilty, all the buttons that could use the actionlistener are used in one if-else if statement. This also ensures that two actions are not carried out simultaneously, which could cause the application to crash or for user-inputted data to be incorrectly handled.

**Creating tables with SQL**

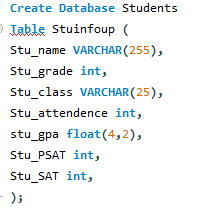


Figure 8: Example of SQL query

This code is an example of a SQL query used to create a database and a table. The table is represented in java as a 2d array. The query not only contains the column names, but also what sort of entries are acceptable for that column. The column named Stu\_gpa, for example, only accepts floats that are 4 digits or less in length, with at most 2 digits after the decimel point(i.e 04.20, 02.1, etc.). This is done to ensure only valid GPAs are accepted.

**Representing SQL databases in Java**

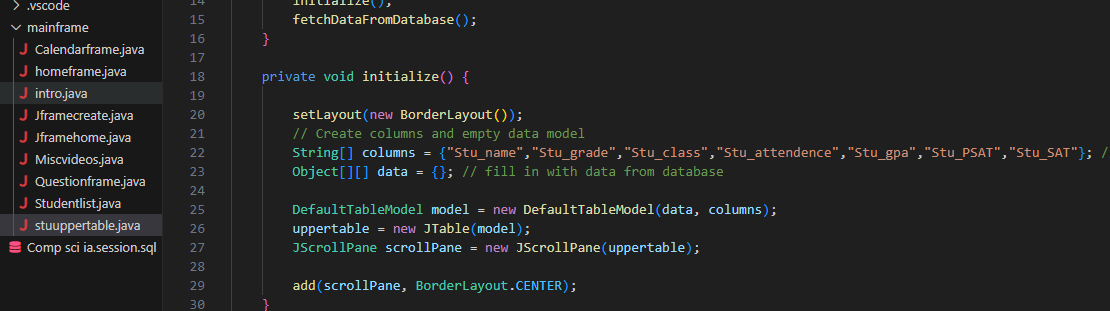


Figure 9: creating table

This code creates a Jtable that will be populated with the data from the database. To properly represent the data, a 2d list is used. A scroll panel is also implemented in case the list grows to large to view on a single page.

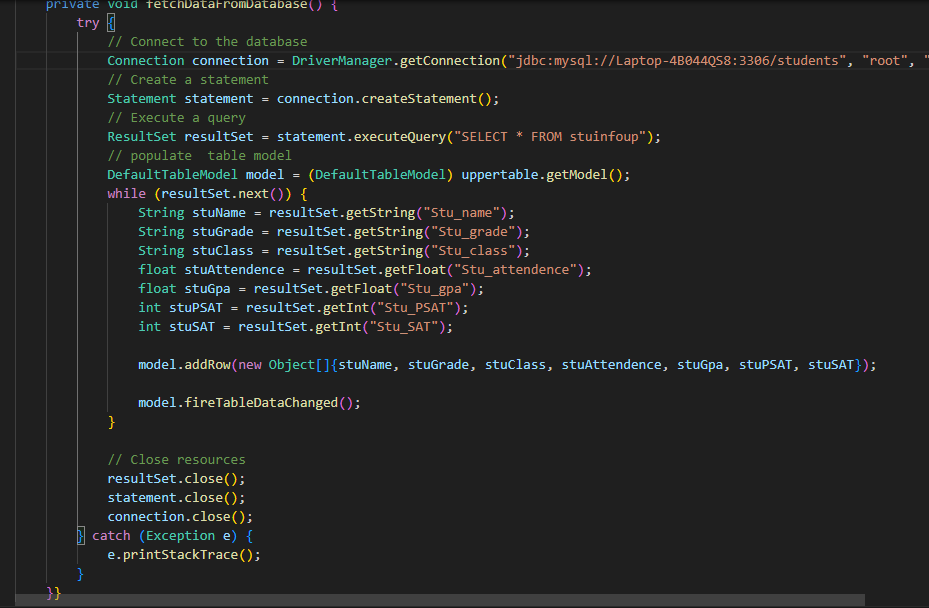


Figure 10: connecting to database and populating table

This method populated the Jtable initialized in figure 9. Firstly, it establishes a connection to the SQL database “Students” via the use of a driver. Then, a Statement is created, which is used to execute a query in the table stuinfoup. Then, a model is applied to the JTable which assigns values to the columns of the Jtable based on the values that are in the corresponding column of the SQL table. A complete row with the appropriate values is then added to the JTable. This is done until there are no more values left in the SQL table to be added. To ensure that the database cannot be accidentally altered, the connection to the database is closed. The method fireTableDataChanged informs the Jtable when there have been new inputs to the table, updating it right after the user does something. “Catch” is used in case there are invalid inputs, blocking them from being added to the table.

**Adding student to database and JTable**

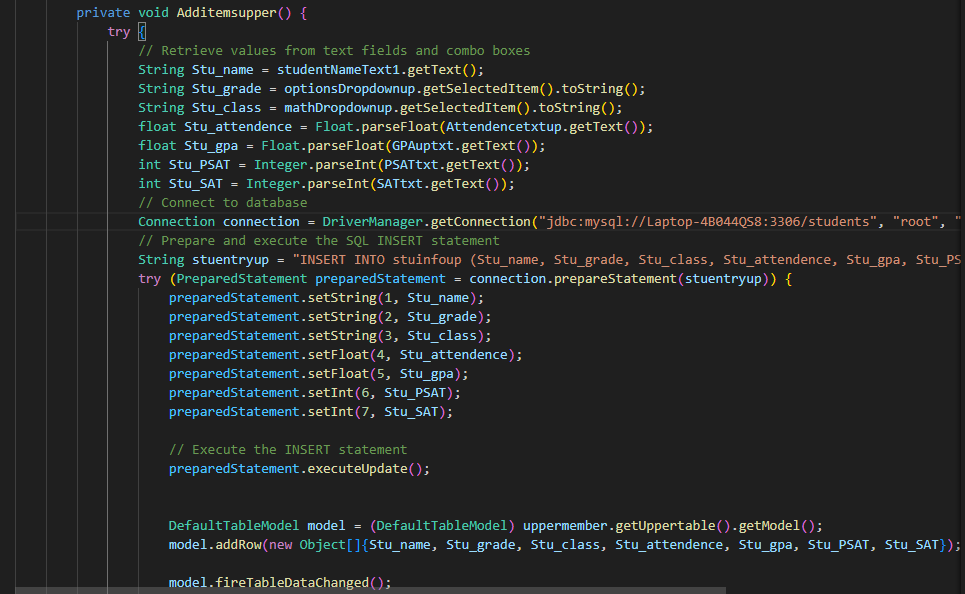


Figure 11: Code adds inputted values to database

This code uses a preparedStatement to add values to the stuinfoup table. A prescribedStatement is like a statement, except that it uses inputs to create a statement with a certain structure. In this case, the inputs from the text fields and comboboxes are used to create a new entry into the SQL database. It then adds a corresponding row into the Jtable. FireTableDataChanged is used here as well to communicate that there has been a change to the database and the table.

**Deleting Student**

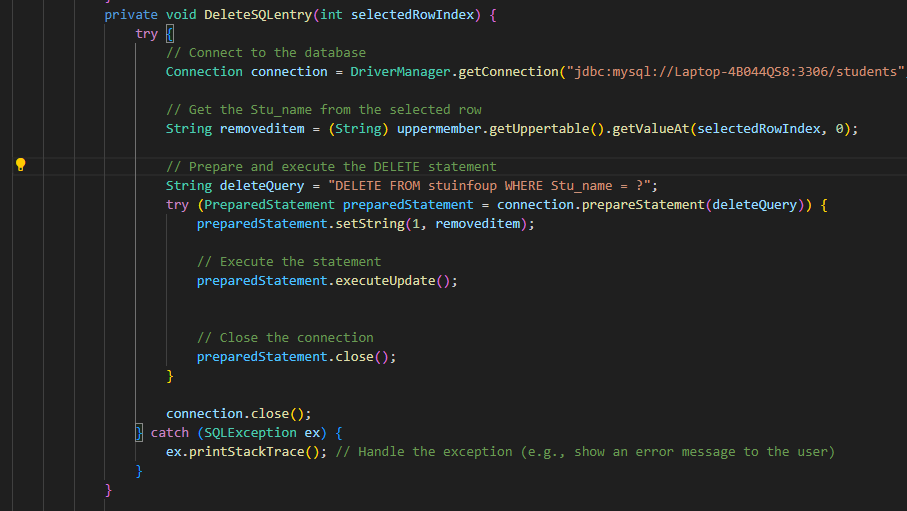


Figure 12: Code deletes selected user from database

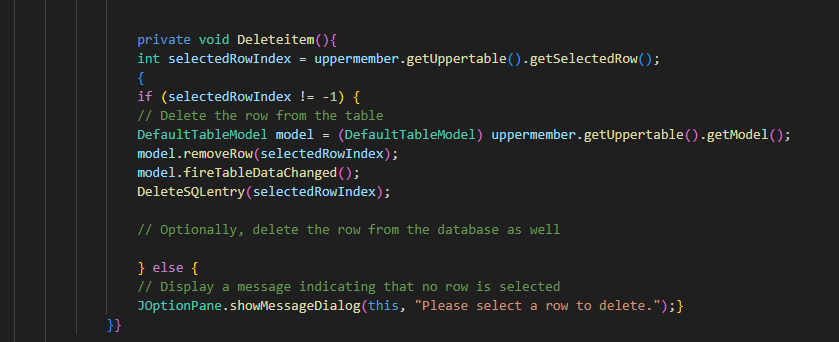


Figure 13: Code deletes selected item from table.

The code in figure 13 removes an entry from the Jtable. It first checks if an item has been selected. If it has, it will be deleted from both the table and the database. If a value has not been selected, a pop-up will open prompting the user to select the item they wish to delete. The method DeleteSQLentry(shown in figure 12) works similarly to the method for adding values to the database.



Figure 14: Shows items being edited in chart

This code shows how values are edited in the chart. To ensure that the correct value is being edited in the database, the Stu-name value in the chart is called to find the corresponding database entry. Then, we change the values in the chart based on the information input by the user. The user can easily edit information because selecting an entry automatically fills in its information into the corresponding text fields and comboboxes.

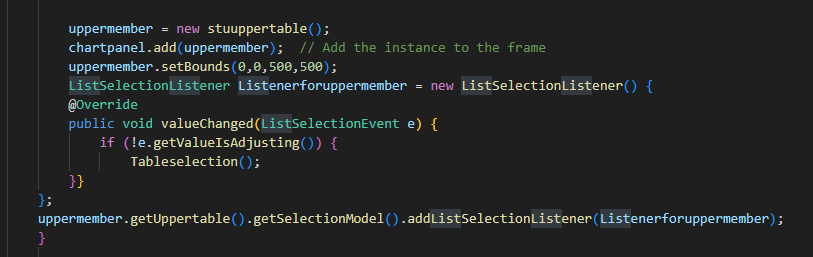


Figure 15: shows how listlistener is applied to uppermember

This code shows how the text field and comboboxes are filled in when an item is selected. The Listselectionlistener is instantiated and added to the table, making sure that the boxes are only filled in after they’re selected.

Word count: 879