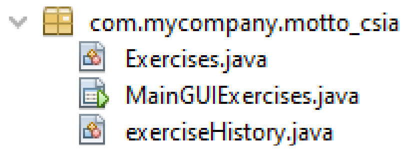
Criterion C: Development

Introduction

The primary purpose of this program requested by the client, Thanavee Sereeyothin, is to be able to add custom exercises, custom sets, and to view the history of those exercises. The program was developed with Java in the NetBeans IDE and the graphical user interface was created with various Java Swing tools for the client to interact with.

Program Structure

Class overview:



```
com.mycompany.motto_csia
    Exercises.java
    MainGUIExercises.java
    exerciseHistory.java
```

The “MainGUIExercises.java” is the main class. Once the main class is run, the main GUI appears and the user can interact with the program by inputting the appropriate values. This main GUI class is dependent on the two other classes “Exercises.java” and “exerciseHistory.java” as it calls on the function from those classes.

“Exercises.java” is the object template class to store data for different exercises. The class contains data attributes which are vital to running the program, such as the name of the exercise and the body part for the exercise. Users can create an instance of the “Exercise” class, and existing exercises objects are also premade for the user to use. A linked list of objects are used to store “Exercise” objects.

On the other hand, “exerciseHistory” is a class which stores the history of exercises. This is used for storing the history of the exercises. Its attributes are from the “Exercises” class as it mainly stores ArrayLists of exercise history. The user also creates an instance of “exerciseHistory” every time an exercise is added, and the ArrayLists are modified through the GUI.

Each class used by the main GUI contains private attributes and methods which are only accessible after instances are made, meaning that the program uses **Encapsulation**. There is also usage of **Abstraction** as methods within the main GUI such as “updateExerciseTable” and “updateExerciseHistoryTable” are called in separate instances to manage the complexity of the program.

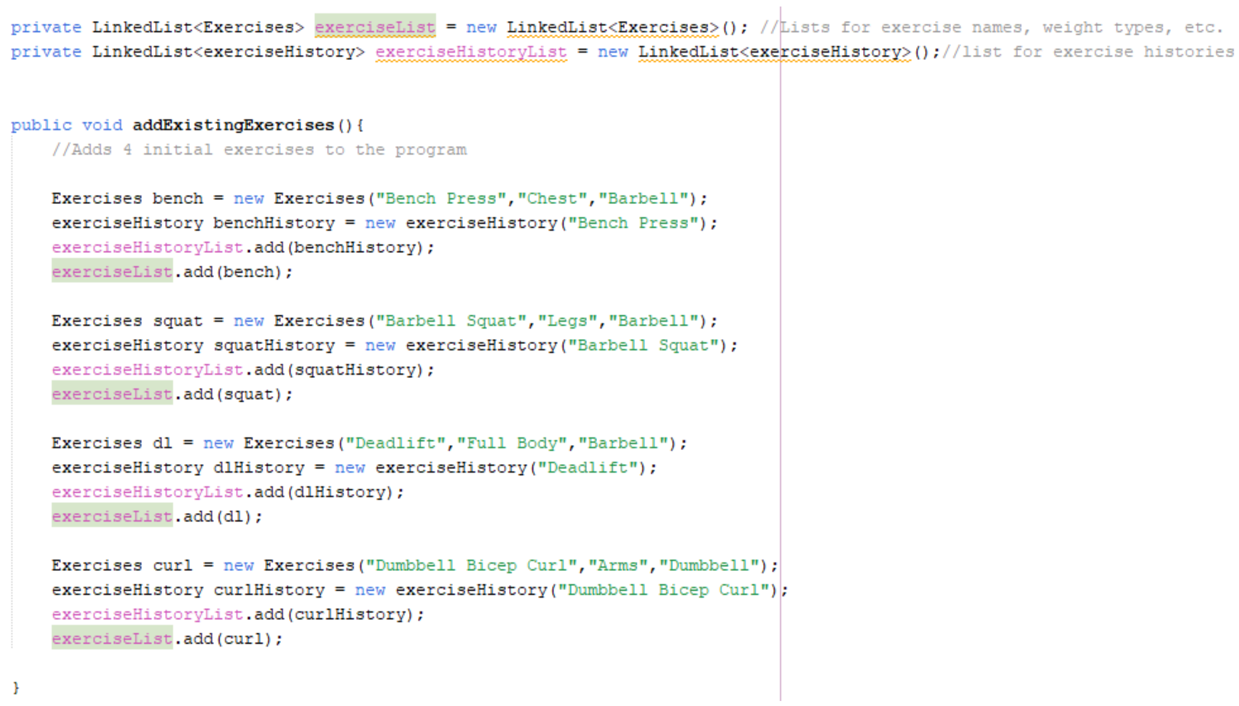
Data Structures

LinkedLists - LinkedLists were used to store multiple instances of “Exercises” and “exerciseHistory”. This dynamic data structure was used, as the amount of exercises would be continually added, in which it is more advantageous to have a dynamic data structure.

ArrayLists - ArrayLists were used as attributes within “exerciseHistory” instances to store the history of the respective exercise. The dynamicity of ArrayLists were used as sets are expected to be continually added and removed. ArrayLists were also used over other types of lists because of the ability to remove specific indexes, which is convenient as the user can select the index they wish to remove from the ArrayList.

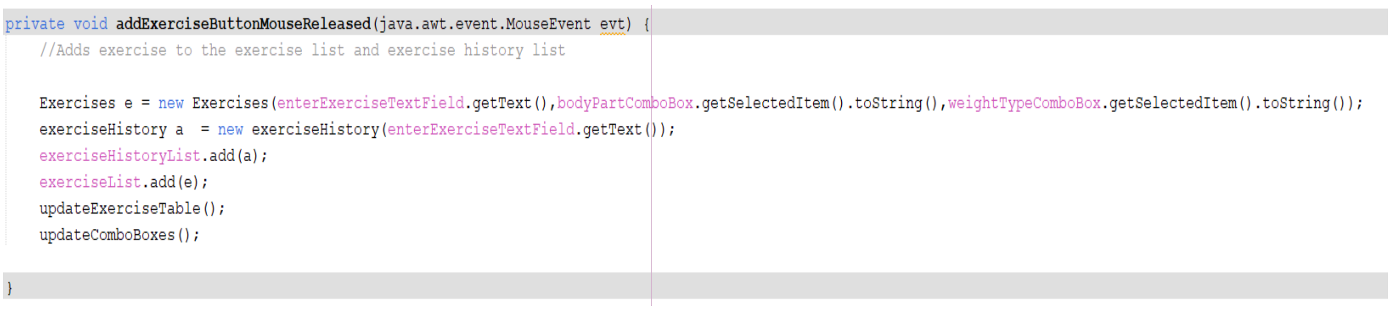
Unique Functionality

1. Adding Exercises



```java
private LinkedList<Exercises> exerciseList = new LinkedList<Exercises>(); //Lists for exercise names, weight types, etc.
private LinkedList<exerciseHistory> exerciseHistoryList = new LinkedList<exerciseHistory>();//list for exercise histories


public void addExistingExercises(){
    //Adds 4 initial exercises to the program

    Exercises bench = new Exercises("Bench Press","Chest","Barbell");
    exerciseHistory benchHistory = new exerciseHistory("Bench Press");
    exerciseHistoryList.add(benchHistory);
    exerciseList.add(bench);

    Exercises squat = new Exercises("Barbell Squat","Legs","Barbell");
    exerciseHistory squatHistory = new exerciseHistory("Barbell Squat");
    exerciseHistoryList.add(squatHistory);
    exerciseList.add(squat);

    Exercises dl = new Exercises("Deadlift","Full Body","Barbell");
    exerciseHistory dlHistory = new exerciseHistory("Deadlift");
    exerciseHistoryList.add(dlHistory);
    exerciseList.add(dl);

    Exercises curl = new Exercises("Dumbbell Bicep Curl","Arms","Dumbbell");
    exerciseHistory curlHistory = new exerciseHistory("Dumbbell Bicep Curl");
    exerciseHistoryList.add(curlHistory);
    exerciseList.add(curl);

}
```

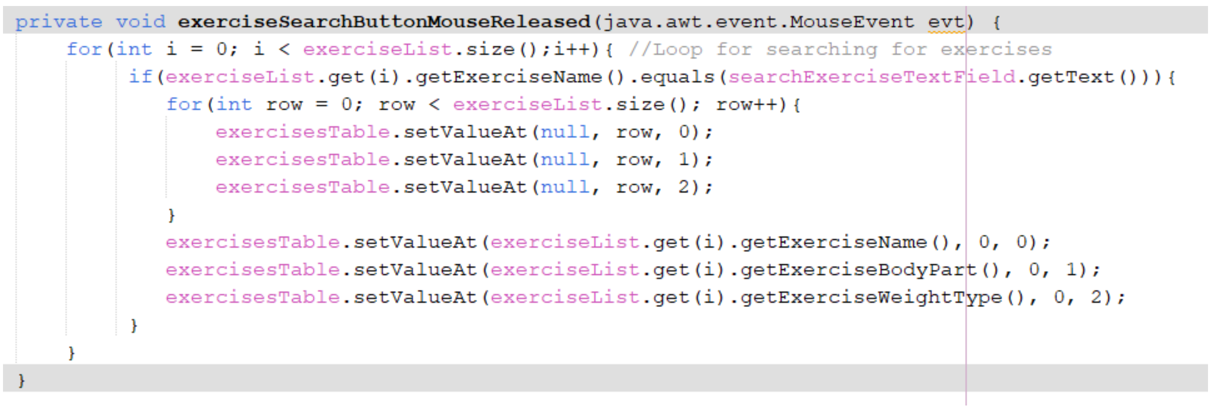
The following code creates lists for the “Exercise” and “exerciseHistory” objects. The “addExistingExercises” method adds 4 premade exercises for the user to use, and the method is called within the main class.



```java
private void addExerciseButtonMouseReleased(java.awt.event.MouseEvent evt) {
    //Adds exercise to the exercise list and exercise history list

    Exercises e = new Exercises(enterExerciseTextField.getText(),bodyPartComboBox.getSelectedItem().toString(),weightTypeComboBox.getSelectedItem().toString());
    exerciseHistory a  = new exerciseHistory(enterExerciseTextField.getText());
    exerciseHistoryList.add(a);
    exerciseList.add(e);
    updateExerciseTable();
    updateComboBoxes();

}
```

The mouseEvent method is called when the “Add Exercise” button is clicked. The exercise information is subsequently taken from the text fields and combo boxes to create a new instance of both classes, which are then stored in their respective lists.

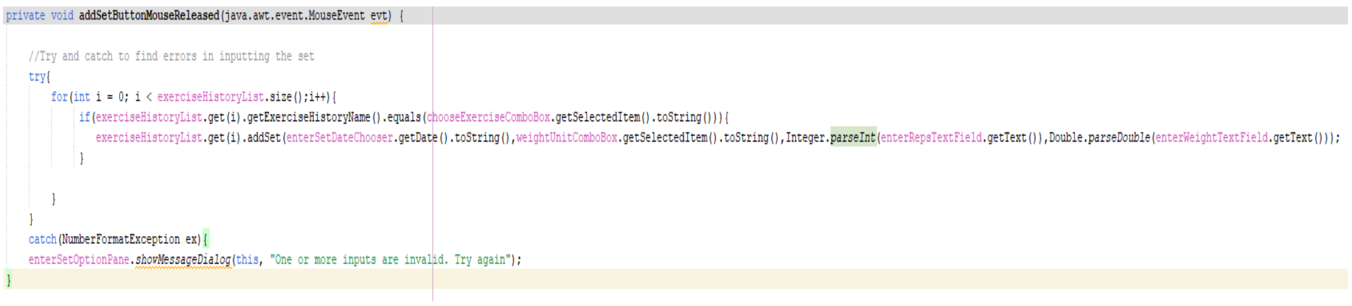
2. Searching for exercises



```java
private void exerciseSearchButtonMouseReleased(java.awt.event.MouseEvent evt) {
    for(int i = 0; i < exerciseList.size();i++){ //Loop for searching for exercises
        if(exerciseList.get(i).getExerciseName().equals(searchExerciseTextField.getText())){
            for(int row = 0; row < exerciseList.size(); row++){
                exercisesTable.setValueAt(null, row, 0);
                exercisesTable.setValueAt(null, row, 1);
                exercisesTable.setValueAt(null, row, 2);
            }
            exercisesTable.setValueAt(exerciseList.get(i).getExerciseName(), 0, 0);
            exercisesTable.setValueAt(exerciseList.get(i).getExerciseBodyPart(), 0, 1);
            exercisesTable.setValueAt(exerciseList.get(i).getExerciseWeightType(), 0, 2);
        }
    }
}
```

The "exerciseSearchButtonMouseReleased" method is called when the button to search for exercise is clicked. This does a simple linear/sequential search through the exercise list to search for what the user inputted in the search text field. After the search is performed, the display table is reset and a singular value is fetched from the exercise list to display in the table.

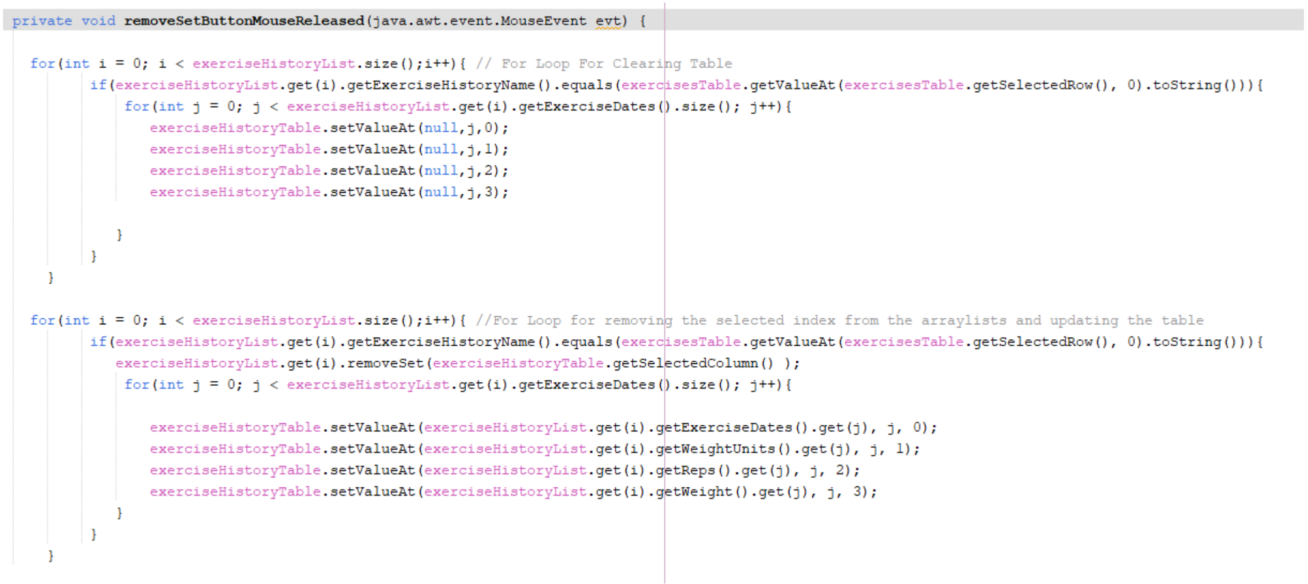
3. Adding Sets



```java
private void addSetButtonMouseReleased(java.awt.event.MouseEvent evt) {

    //Try and catch to find errors in inputting the set
    try{
        for(int i = 0; i < exerciseHistoryList.size();i++){
            if(exerciseHistoryList.get(i).getExerciseHistoryName().equals(chooseExerciseComboBox.getSelectedItem().toString())){
              exerciseHistoryList.get(i).addSet(enterSetDateChooser.getDate().toString(),weightUnitComboBox.getSelectedItem().toString(),Integer.parseInt(enterRepsTextField.getText()),Double.parseDouble(enterWeightTextField.getText()));
            }


        }
    }
    catch(NumberFormatException ex){
    enterSetOptionPane.showMessageDialog(this, "One or more inputs are invalid. Try again");
}
```

The "addSetButtonMouseReleased" is called when the button to add set is clicked. It initially uses a sequential search to match the selected exercise in the combo box with the one on the list. The method utilizes the "addSet" function within the exerciseHistory instance to append the data from the GUI to its various ArrayLists. There is also a try and catch for error handling when a user inputs an invalid value.

4. Removing Sets from Exercise History



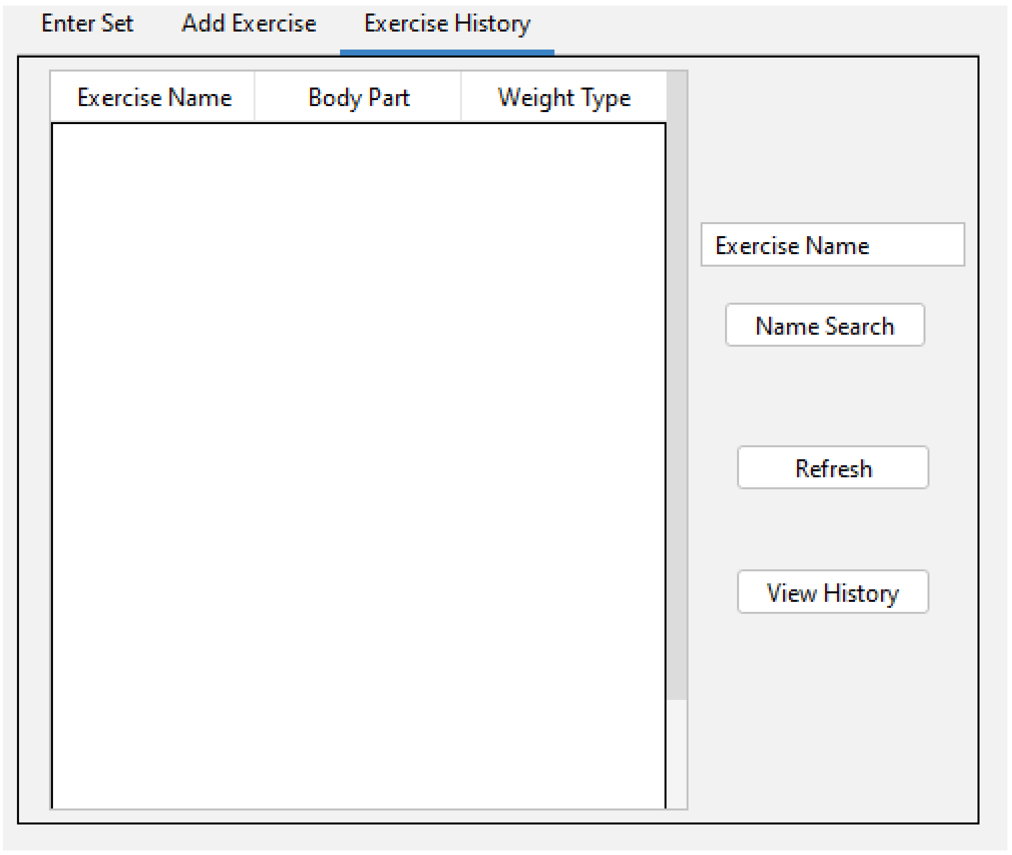
```java
private void removeSetButtonMouseReleased(java.awt.event.MouseEvent evt) {

  for(int i = 0; i < exerciseHistoryList.size();i++){ // For Loop For Clearing Table
         if(exerciseHistoryList.get(i).getExerciseHistoryName().equals(exercisesTable.getValueAt(exercisesTable.getSelectedRow(), 0).toString())){
             for(int j = 0; j < exerciseHistoryList.get(i).getExerciseDates().size(); j++){
                exerciseHistoryTable.setValueAt(null,j,0);
                exerciseHistoryTable.setValueAt(null,j,1);
                exerciseHistoryTable.setValueAt(null,j,2);
                exerciseHistoryTable.setValueAt(null,j,3);

             }
         }
    }

  for(int i = 0; i < exerciseHistoryList.size();i++){ //For Loop for removing the selected index from the arraylists and updating the table
         if(exerciseHistoryList.get(i).getExerciseHistoryName().equals(exercisesTable.getValueAt(exercisesTable.getSelectedRow(), 0).toString())){
            exerciseHistoryList.get(i).removeSet(exerciseHistoryTable.getSelectedColumn() );
             for(int j = 0; j < exerciseHistoryList.get(i).getExerciseDates().size(); j++){

                exerciseHistoryTable.setValueAt(exerciseHistoryList.get(i).getExerciseDates().get(j), j, 0);
                exerciseHistoryTable.setValueAt(exerciseHistoryList.get(i).getWeightUnits().get(j), j, 1);
                exerciseHistoryTable.setValueAt(exerciseHistoryList.get(i).getReps().get(j), j, 2);
                exerciseHistoryTable.setValueAt(exerciseHistoryList.get(i).getWeight().get(j), j, 3);
             }
         }
    }
```

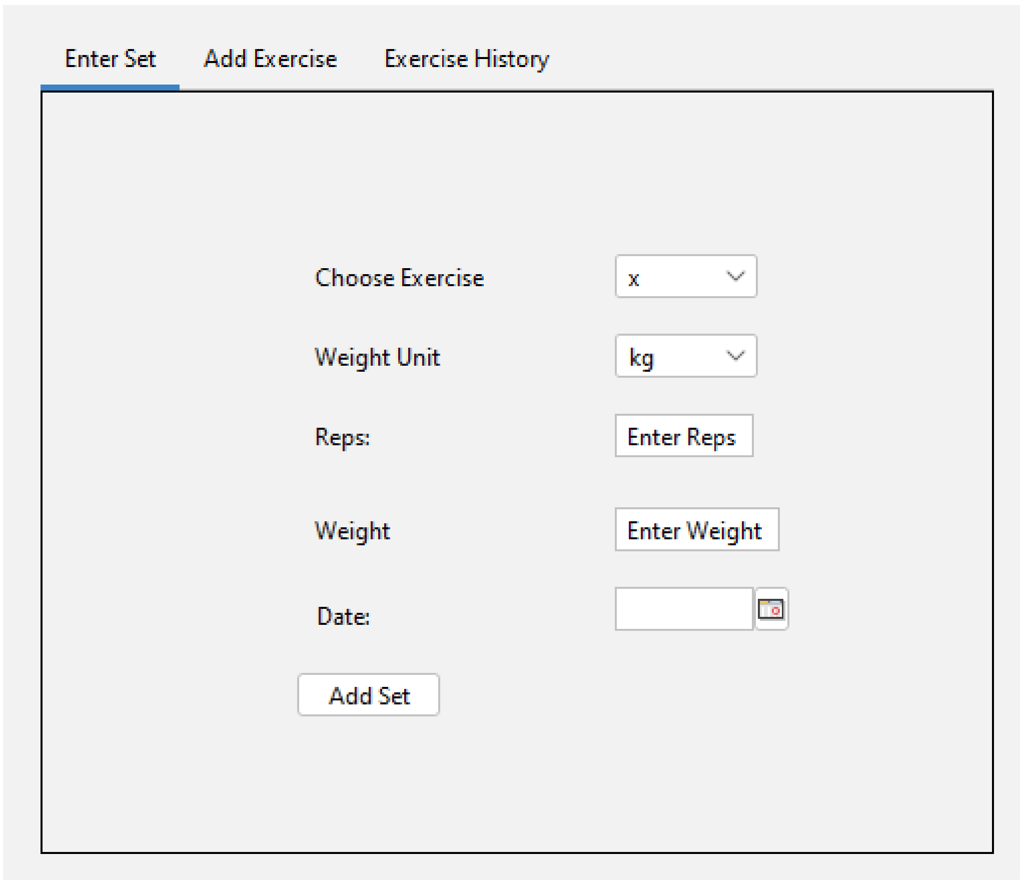
The “removeSetButtonMouseReleased” method is called when the corresponding button is pressed. The initial for loop resets the exercise history table. The second for loop fetches the array the user is selecting, removes the respective index in the ArrayList, and re updates the exercise history display table.

GUI Elements

The following Java Swing tools were used for an easily interactable and intuitive GUI:

- JTextFields
- JComboBox
- JDateChooser
- JButtons
- JLabels
- JTabbedPane
- JDialog
- JOptionPane
- JMenuBar


Enter Set
Add Exercise
Exercise History
Exercise Name
Body Part
Weight Type
Exercise Name
Name Search
Refresh
View History


Enter Set
Add Exercise
Exercise History
Choose Exercise
x
Weight Unit
kg
Reps:
Enter Reps
Weight
Enter Weight
Date:
Add Set

Software Tools Used

The Apache NetBeans IDE version 13 was used to develop this program. The user friendly design allowed me to easily create a GUI interface which matched the needs of my client. An additional library called “jcalendar-1.4.jar” was used to add a date selection element to the GUI to make inputting the date easier for the user.

Word Count: 762