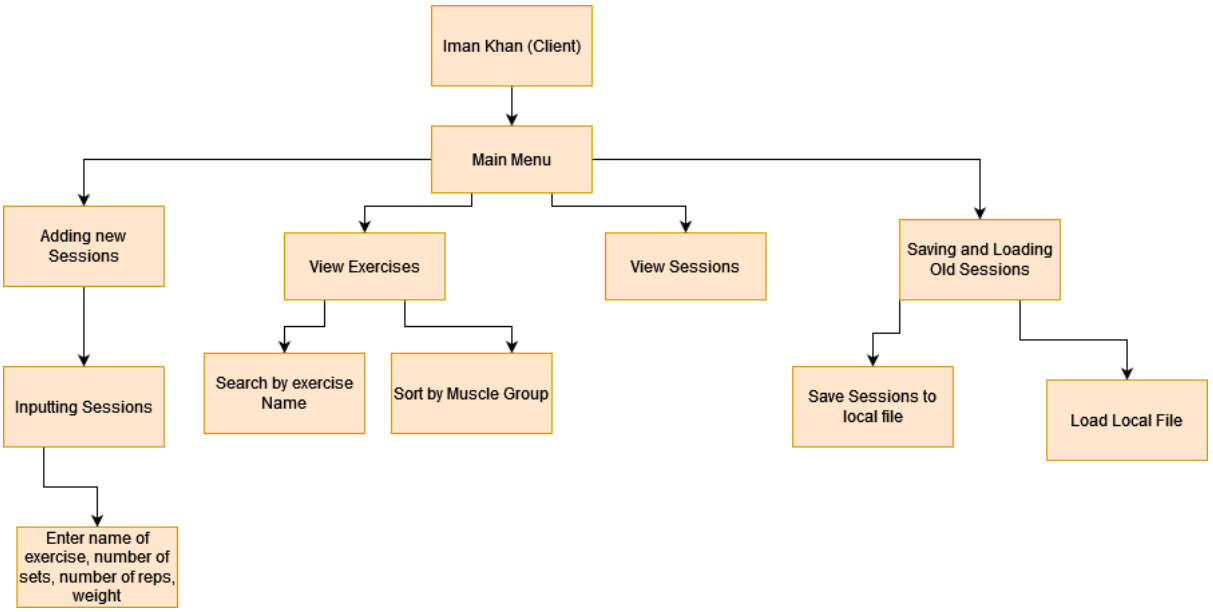
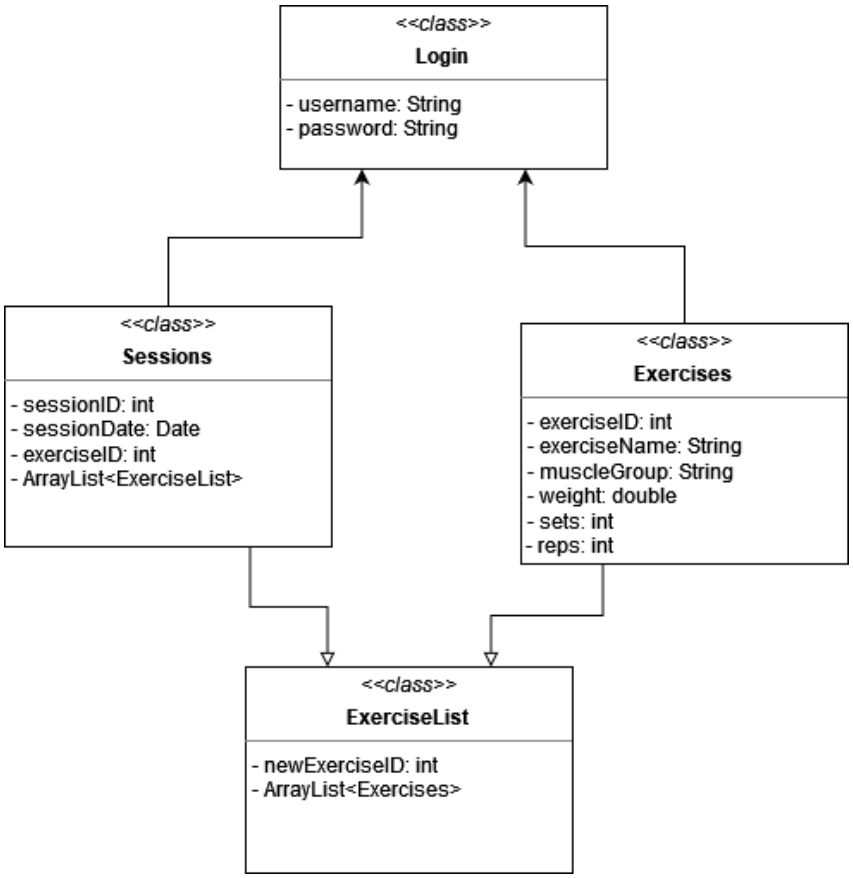
Criterion B: Design

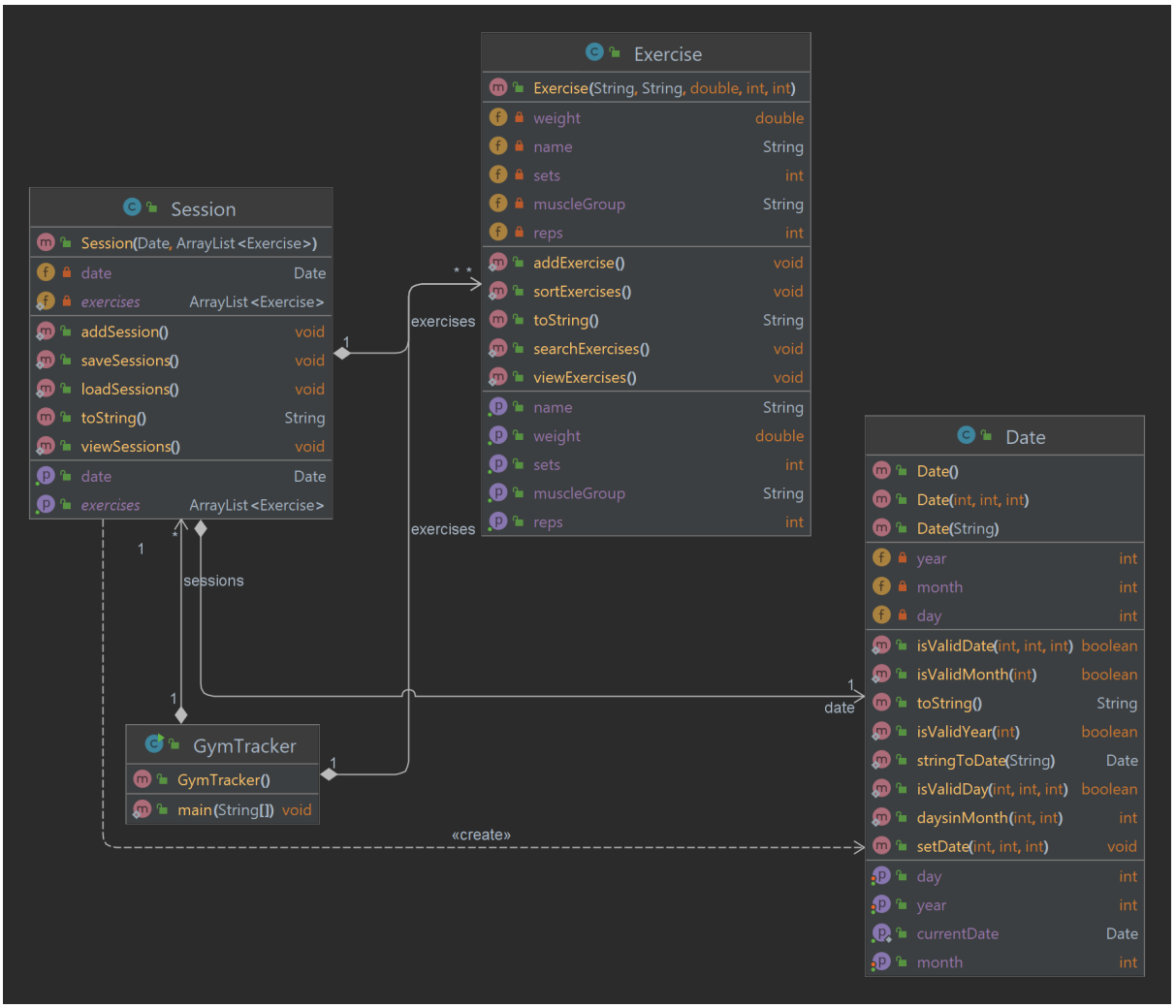
1. **Decomposition Diagram:** This diagram shows the links between different functions showing the flow of operations of the system.



2. **Initial UML Class Diagram:** This diagram shows a brainstorm of the different classes and variables I would need in my code, however this is open to be expanded as I progress.

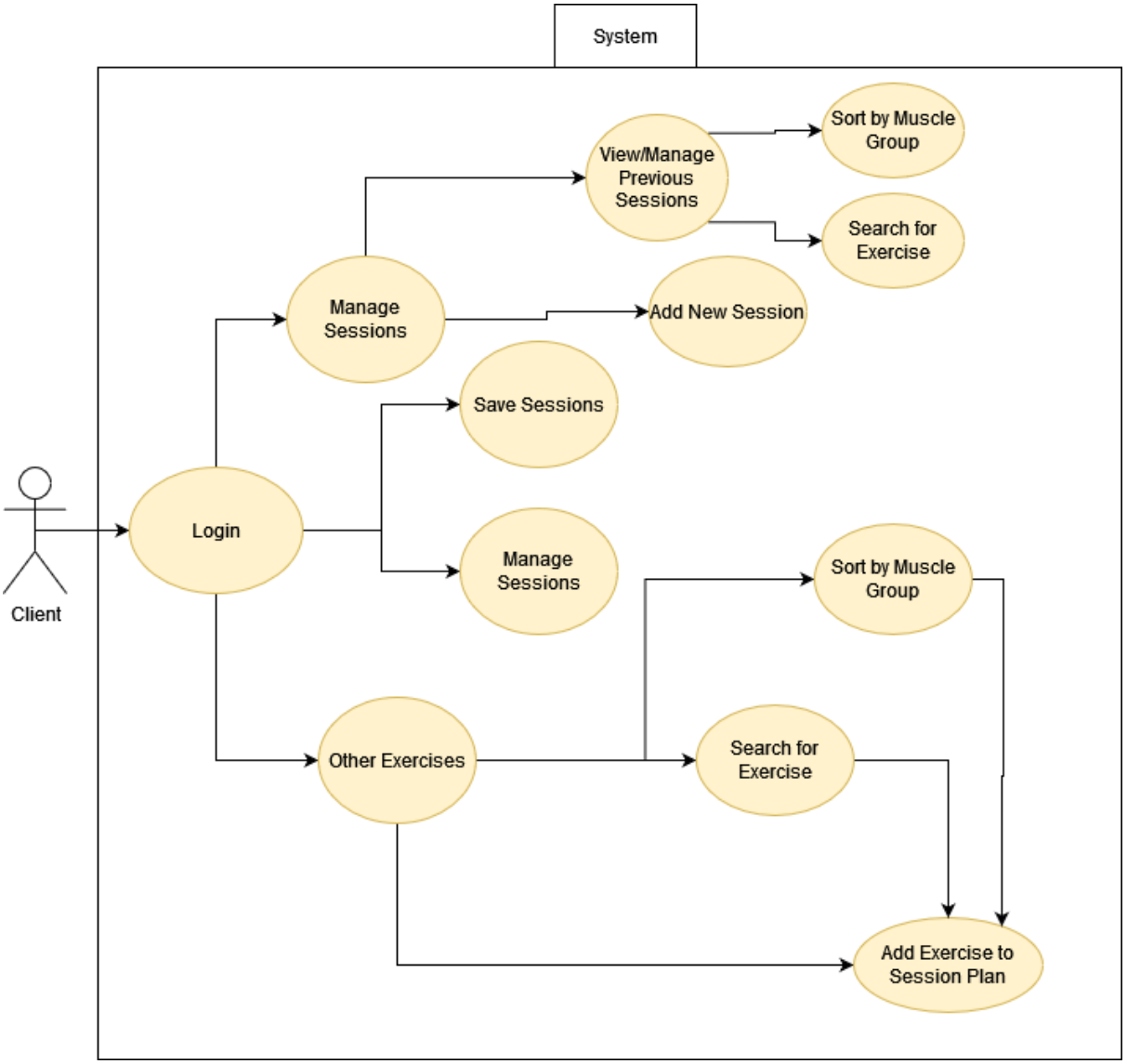


3. **Final UML Class Diagram:** This diagram shows my UML class Diagram complete with methods, variables and relationships.



I decided to get rid of the ExerciseList class as when it comes to lifting exercises the amount of possible exercises would be too large and it may end up confusing the user. Hence it would be easier if the user has control of the name of the Exercise as well, as some people have different abbreviations or names for different Exercises. This allows for the interaction between the solution and the user to be more efficient. Additionally, I used a date class in order to save the sessions under a Date data type instead of a String as it is more efficient in validating if the actual date inputted by the user is a possible date.

4. **Use Case Diagram:** This diagram outlines the user's interaction with the system and what they will see



Data Dictionary

Attribute	Data Type	Modifier	Description
		Session	
date	Date	private	This is the Date that the session took place as every session will need a Date in order to track progress and will be saved as a Date data type using a custom made Date class.
exercises	ArrayList	private	This is the ArrayList of the different exercises that will be inside each session.

Date			
year	int	private	This is the year of the Date class that will be expressed as an integer.
month	int	private	This is the month of the Date class that will be expressed as an integer.
day	int		This is the date of the Date class that will be expressed as an integer.
Exercise			
name	String	private	This is the name of the exercise that the user will input, the user can search for exercise names as well.
muscleGroup	String	private	This is the muscle group that the exercise targets, the user can use this to structure their sessions as well as search and sort through exercises.
weight	double	private	This is a variable that the user can change based on what weight that they are using for the exercise currently.
sets	int	private	This is a variable that the user can change based on how many sets they are doing per exercise.
reps	int	private	This is a variable that the user can change based on how many reps they are doing per exercise.

Test Plan

Class: Exercise
Variable: weight: double

Type of Validation	Test Data Type	Input Data	Expected Output	Test Passed X Test Failed ✓
Range Check (1 <= x < 1000)	**Normal**	**“25.5”**	**Accept**	✓
	Extreme	**“24304.2”**	**Reject and input is needed again**	**X**

	Abnormal	**“1000.0”**	**Reject and input is needed again**	**X**

Class: Exercise
Variable: sets: int

Type of Validation	**Test Data Type**	**Input Data**	**Expected Output**	**Test Passed X Test Failed ✓**
Format Check (integers only) **Range Check (1 <= x <= 15)**	**Normal**	**“5”**	**Accept**	**✓**
	Extreme	**“Bob”**	**Reject and input is needed again**	**X**
	Abnormal	**“16”**	**Reject and input is needed again**	**X**

Class: Exercise
Variable: reps: int

Type of Validation	**Test Data Type**	**Input Data**	**Expected Output**	**Test Passed X Test Failed ✓**
Format Check (integers only) **Range Check (1 <= x <= 25)**	**Normal**	**“5”**	**Accept**	**✓**
	Extreme	**“Bob”**	**Reject and input is needed again**	**X**
	Abnormal	**“26”**	**Reject and input is needed again**	**X**

Class: Exercise
Variables: name, muscleGroup: String

Type of Validation	**Test Data Type**	**Input Data**	**Expected Output**	**Test Passed X Test Failed ✓**
Presence Check	**Normal**	**“!= null”**	**Accept**	**✓**
	Extreme	**“null”**	**Reject and input is needed again**	**X**

	Abnormal	**null**	**Reject and input is needed again**	**X**

Class: Session
Variable: day: year

Type of Validation	**Test Data Type**	**Input Data**	**Expected Output**	**Test Passed X Test Failed ✓**
Format Check (integers only) **Range Check (2000 >= x >= 2200)**	**Normal**	**“2023”**	**Accept**	**✓**
	Extreme	**“Bob”**	**Reject and input is needed again**	**X**
	Abnormal	**“1999”**	**Reject and input is needed again**	**X**

Class: Session
Variable: day: month

Type of Validation	**Test Data Type**	**Input Data**	**Expected Output**	**Test Passed X Test Failed ✓**
Format Check (integers only) **Range Check (1 <= x <= 12)**	**Normal**	**“6”**	**Accept**	**✓**
	Extreme	**“Bob”**	**Reject and input is needed again**	**X**
	Abnormal	**“13”**	**Reject and input is needed again**	**X**

Class: Session
Variable: day: int

Type of Validation	**Test Data Type**	**Input Data**	**Expected Output**	**Test Passed X Test Failed ✓**
Format Check (integers only) **Range Check depending on value of month**	**Normal**	**“10”**	**Accept**	**✓**
	Extreme	**“Bob”**	**Reject and input is needed again**	**X**

(1 <= x <= 31) (1 <= x <= 30) (1 <= x <= 30)	Abnormal	“31”	Reject and input is needed again	X

Pseudocode

Sorting Algorithm:

Each exercise will be an ArrayList which will contain a muscle group for each exercise. THe exercises will be sorted alphabetically by muscle group in ascending order in order for the client to view the exercises which target the same muscle group so that they can see the dispersion for each muscle group and try and modify their routine in order to help my client reach their personal goals. An efficient way to do this would be using a selection sort.

EXERCISES = ArrayList being sorted
NUMB = number of elements in the ArrayList
SMALLEST_ELEMENT = area for holding the smallest element found in that pass
CURRENT_SMALLEST_POSITION = the value of the current position in which to place the smallest element
I = index for outer loop
J = index for inner loop.

```
1     Selection_sort_algorithm
2           CURRENT_SMALLEST_POSITION = 1
3           loop while CURRENT_SMALLEST_POSITION <= (NUMB - 1)
4                 I = CURRENT_SMALLEST_POSITION
5                 SMALLEST_ELEMENT = EXERCISES(I)
6                 J = I + 1
7                 loop while J <= NUMB
8                       if EXERCISES(J) < SMALLEST_ELEMENT then
9                             I = J
10                      SMALLEST_ELEMENT = EXERCISES(J)
11                end if
12                J = J + 1
13          end loop
14          EXERCISES(I) = EXERCISES(CURRENT_SMALLEST_POSITION)
15          EXERCISES(CURRENT_SMALLEST_POSITION) = SMALLEST_ELEMENT
16          add 1 to CURRENT_SMALLEST_POSITION
17    end loop
18  end

adapted from (Robertson, Lesley Anne)
```

Searching Algorithm:

Assuming that the algorithm has already been sorted into ascending order using the sorting algorithm already. The searching algorithm will search for and return the exercise using its exercise name. To aid in the process of efficiency, and ease of processing power, a Binary Search. The efficiency of this algorithm is O ($log_2 n$). This is more efficient than other search algorithms such as the Linear/Sequential search, with an efficiency of O(n).

EXERCISES = ArrayList being sorted
EXERCISE_NAME = exercise name being searched for
IS_FOUND = flag containing true if found, false otherwise
PLACE = the index of the key element in the collection
LENGTH = number of items (size) of the ArrayList (EXERCISES)
LOW = lower index or left index of a sub-array where the key is searched
HIGH = highest index or right index of a sub-array where the key is searched

```
1     function binarySearch(EXERCISES, EXERCISE_NAME) : String
2           set IS_FOUND to false
3           set LOW to 0
4           set HIGH to LENGTH
5           set PLACE to -1
6           loop while not FOUND and (LOW < HIGH)
7                 set INDEX to (LOW + HIGH) / 2
8                 if EXERCISES.get(INDEX) == EXERCISE_NAME then
9                       set IS_FOUND to true
10                      set PLACE to INDEX
11                      exit loop
12                else
13                      if EXERCISE_NAME < EXERCISE.get(INDEX) then
14                            set HIGH to INDEX – 1
15                      else
16                            set LOW  to INDEX + 1
17                      end if
18                end if
19          end loop
20          return EXERCISE.get(PLACE)
21    end function

adapted from (Robertson, Lesley Anne)
```

Extensions and Modifications (from criteria D)

Extension or Modification	Location	Development
Total number of exercises per muscle group	Exercise.java	Add a counter that starts at 0 and increments for each unique exercise with the same muscle group. Which could then be displayed in the viewExercises method for each muscle group. To do this I would create a nested for loop going through every exercise, and compare the muscleGroup for each exercise in list to each other, and increment a counter when the muscleGroup in the first for loop is equal to the muscleGroup in the second for loop and the exerciseName in each of the loops are not equal. As if they are equal it is only counted as one exercise, so it would be pointless to tally it as the purpose is to see the number of different exercises.
Change dates to have days of the week as well	Date.java Session.java	Within the Date class use the calendar in order to get the instance with a day of the week as well instead of just the numerical value. Doing this would change the constructors in the Date class to include a String for the day of the week, as well as new setters. Within the Session.java class, specifically the addSession method I would just use a getter in order to get the day of the month for the date that was inputted by the user after all validation has been done. Rather than having the user inputting the day and having to compare the day inputted with the correct day, I would just get the value for the day from the Date class and add this to part of each session. Doing this would allow for the client to be able to track their workouts more easily as by doing this it would allow for the client to see what days of the week certain exercises are being done on, as this can help the client plan future workouts and get consistent rest between each session. Although this can be done with the numerical date, by using the String for the day it would make it easier for the client to subconsciously process.