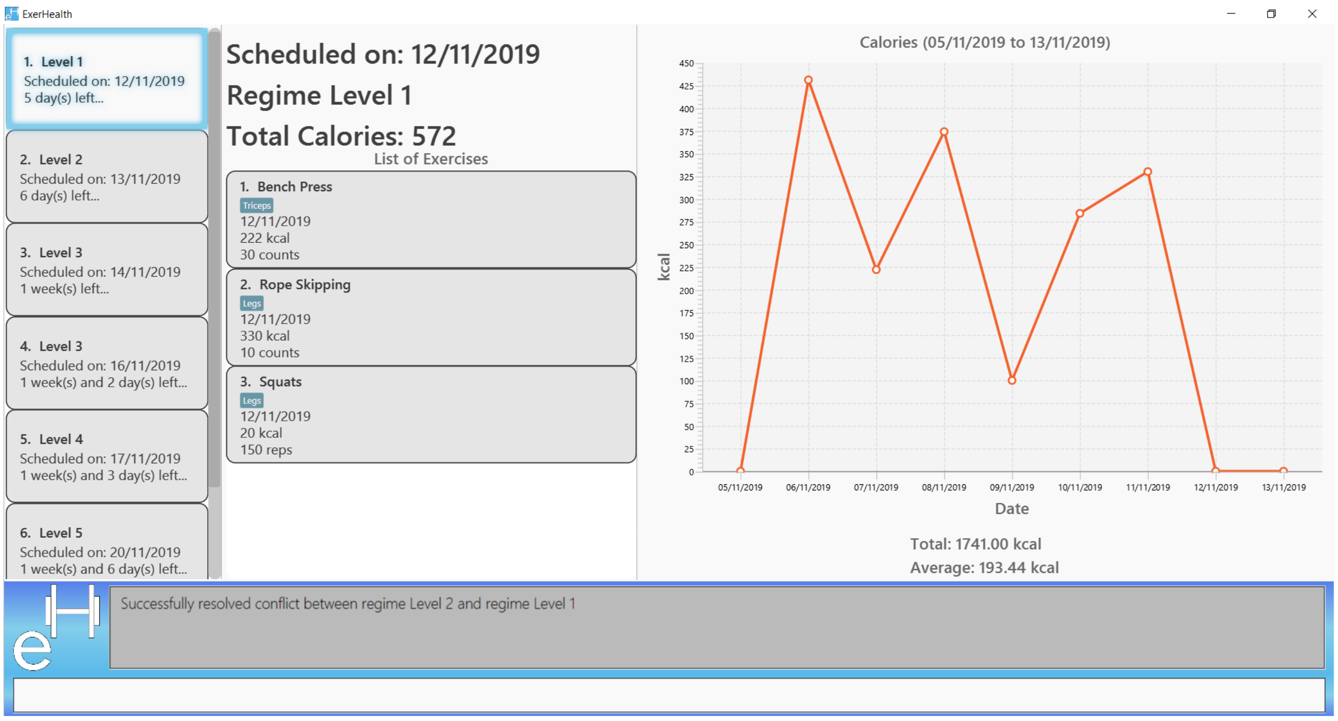
Kwek Kee En - Project Portfolio

PROJECT: ExerHealth

Overview

My team of 5 computer science students were tasked with changing a basic command line application. Our team decided to morph the application into ExerHealth. **ExerHealth** is a desktop application used for tracking and scheduling the user’s exercises. The application has statistical analysis of exercises users have completed in the past. Additionally, it also acts as a personal trainer by suggesting different exercises which both beginners and advanced users can choose from to incorporate into their exercise regimes. The user interacts with it using a command line interface, and it has a GUI created with JavaFX.

Below is a screenshot of what our desktop application looks like:



1. Level 1
Scheduled on: 12/11/2019
5 day(s) left...

2. Level 2
Scheduled on: 13/11/2019
6 day(s) left...

3. Level 3
Scheduled on: 14/11/2019
1 week(s) left...

4. Level 3
Scheduled on: 16/11/2019
1 week(s) and 2 day(s) left...

5. Level 4
Scheduled on: 17/11/2019
1 week(s) and 3 day(s) left...

6. Level 5
Scheduled on: 20/11/2019
1 week(s) and 6 day(s) left...

Scheduled on: 12/11/2019
Regime Level 1
Total Calories: 572
List of Exercises

1. Bench Press
Triceps
12/11/2019
222 kcal
30 counts

2. Rope Skipping
Legs
12/11/2019
330 kcal
10 counts

3. Squats
Legs
12/11/2019
20 kcal
150 reps

Calories (05/11/2019 to 13/11/2019)

Total: 1741.00 kcal
Average: 193.44 kcal

Successfully resolved conflict between regime Level 2 and regime Level 1

Summary of contributions

- **Major enhancement**: added **the ability to search for suggestions**
    - What it does: The command `suggest` allows the user to search for suggestions.
    - Justification: This feature enables new users to have a starting point in their exercise regime. This feature also offers experienced users suggestions based on the type of exercises the user wants to do.
    - Highlights: This enhancement works well with existing features, such as Custom Properties, and can be expanded on. It required an in-depth analysis of design alternatives to ensure that future extensions or further enhancements can be smooth. The implementation too was challenging as it required multiple new predicate and utility classes.
- **Minor enhancement 1**: Added the display panel on the left hand side of the UI to show the respective information after a command is executed.
- **Minor enhancement 2**: Allowed the command box to be automatically focused on upon opening the application so that user does not need to click on the box to start typing.
- **Code contributed**: RepoSense
- **Other contributions**:
    - Project management:
        - Managed releases `v1.2` - `v1.4` (3 releases) on GitHub
        - Wrote additional tests for existing features to increase coverage (Pull requests #137, #96)
    - Enhancements to existing features:
        - Refactored the GUI (Pull requests #121
    - Community:
        - PRs reviewed (with non-trivial review comments): #114, #81
        - Reported bugs and suggestions for other projects (#199, #198, #196, #190, #185)

Contributions to the User Guide

Given below are sections I contributed to the User Guide. They showcase my ability to write documentation targeting end-users.

Suggesting ideas: `suggest`

Suggest basic exercises

Recommends exercises from ExerHealth’s inbuilt database for beginners.

Format: `suggest s/basic`

Suggest possible exercises

Suggests exercises matching specified tags.

Based on matching muscle tags

Format: `suggest s/possible o/OPERATION_TYPE [m/MUSCLE CUSTOM_PROPERTY_PREFIX_NAME/VALUE]`

- You must choose one of the following operation type: `and` / `or` i.e. commands such as `suggest s/possible m/Chest m/Legs` will fail whereas `suggest s/possible o/or m/Chest m/Legs` will succeed.
- You have to enter at least one property (muscle/custom property) to search for suggestions i.e. commands such as `suggest s/possible o/and` and `suggest s/possible o/or` will both fail whereas `suggest s/possible o/or m/Chest` will succeed.
- The operation type is optional if there is only one tag provided i.e. commands such as `suggest s/possible o/and m/Chest`, `suggest s/possible o/or m/Chest` and `suggest s/possible m/Chest` will all achieve the same outcome - display all the exercises tagged with "Chest" in the exercise tracker and ExerHealth’s database.

Based on matching custom properties

Similar to matching muscles tags, you can search for suggestions with matching custom property tags.

After creating custom properties and tracking exercises, you can search for suggestions with those custom properties.

Example: Suppose you have created a new custom property and have been tracking a few exercises with said custom property:

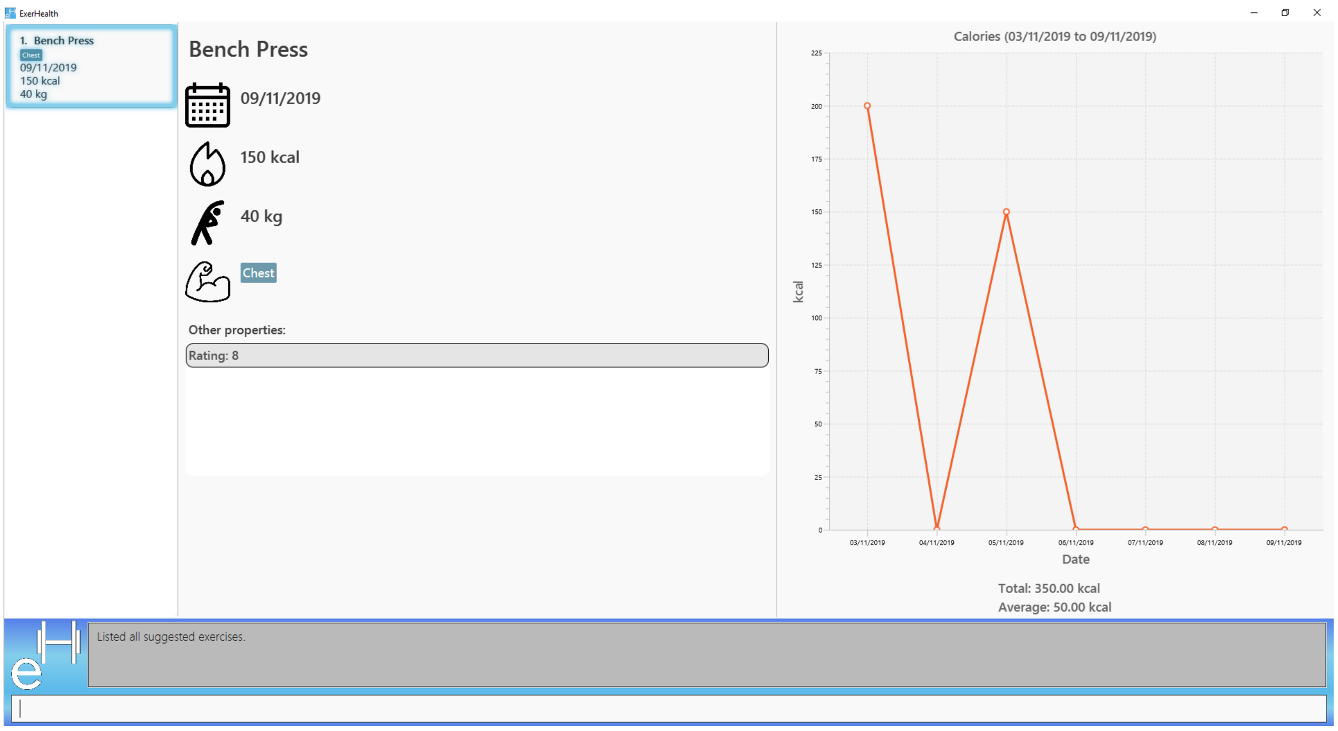
1. `custom s/r f/Rating p/Number`
2. `add t/exercise n/Run d/03/11/2019 c/200 q/10 u/km m/Legs r/8`
3. `add t/exercise n/Bench Press d/05/11/2019 c/150 q/40 u/kg m/Chest r/8`

Then, the following input will display a list of exercises which are tagged with "Chest" and have a rating of 8.

`suggest s/possible o/and m/Chest r/8`

Thus the command will display only the exercise named "Bench Press".

Expected Result:

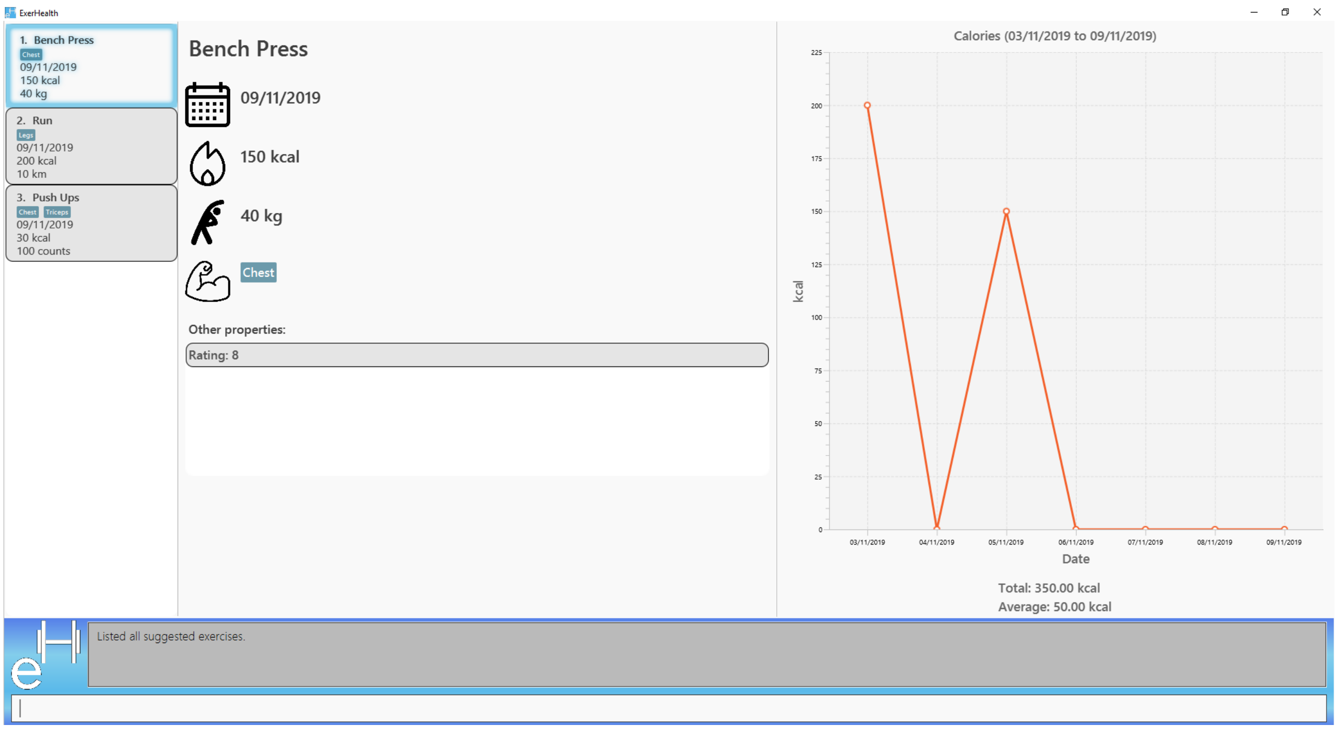


Listed all suggested exercises.

Figure 1. Exercises with a `Chest` *tag and a rating of* `8` *are shown.*

The input, `suggest s/possible o/or m/Chest r/8`, however, will display a list of exercises with "Chest" **or** have a rating of 8.

Expected Result:



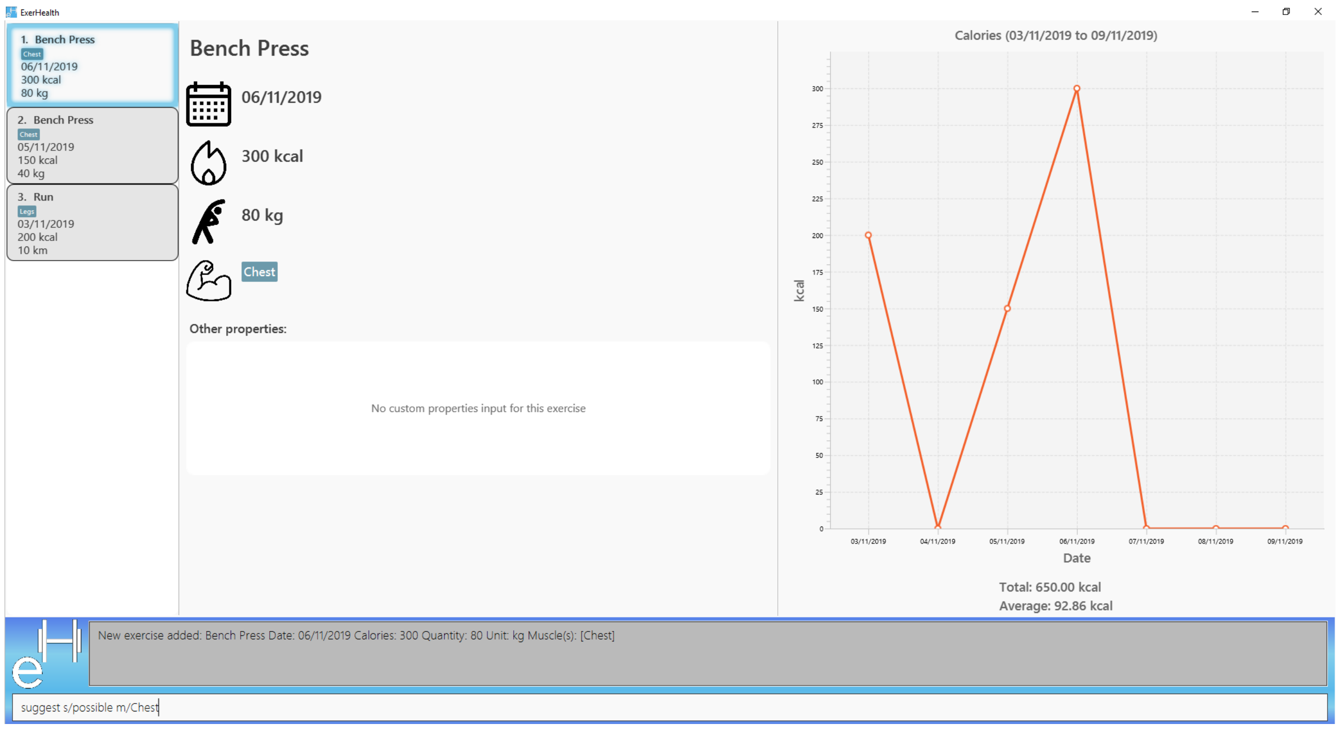
Listed all suggested exercises.

Figure 2. Exercises with a `Chest` *tag and exercises with a rating of* `8` *are shown.*

As seen on the image above, the two previously added exercises named "Bench Press" and "Run" are displayed because they each have a rating of 8. In addition to the tracked exercises, ExerHealth will also display suggestions in its database. Hence the exercise named "Push Ups" is displayed as it has a "Chest" muscle tag.

Duplicates

Sometimes, you may track exercises of the same name. Instead of displaying all suggestions of the same name, this function display the information of the most recently tracked exercise of that name. As can be seen below, there are two exercises named "Bench Press". What suggest command will do is to use the information from the most recent occurence.



1. Bench Press
Chest
06/11/2019
300 kcal
80 kg

2. Bench Press
Chest
05/11/2019
150 kcal
40 kg

3. Run
Legs
03/11/2019
200 kcal
10 km

Bench Press
06/11/2019
300 kcal
80 kg
Chest

Other properties:

No custom properties input for this exercise

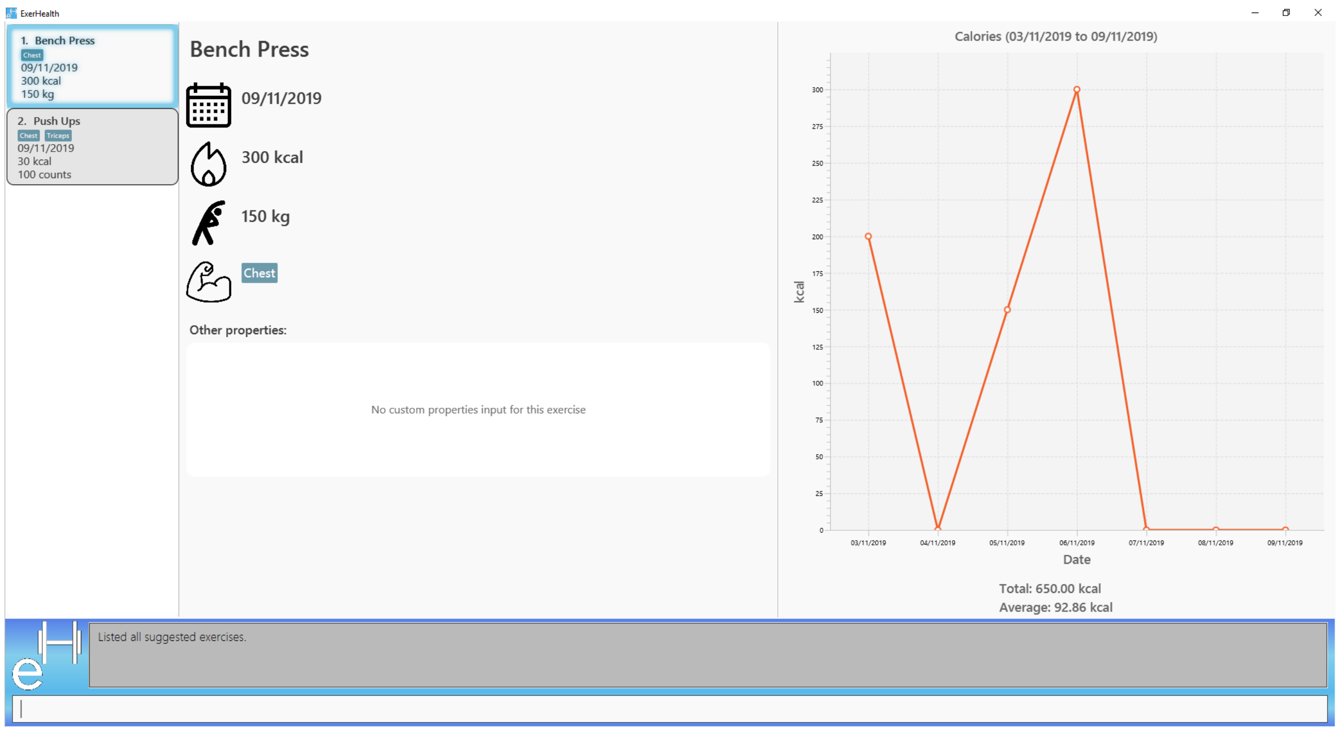
Total: 650.00 kcal
Average: 92.86 kcal

```
New exercise added: Bench Press Date: 06/11/2019 Calories: 300 Quantity: 80 Unit: kg Muscle(s): [Chest]
```

```
suggest s/possible m/Chest
```

Figure 3. Before entering the new suggest command

Expected Result:



1. Bench Press
Chest
09/11/2019
300 kcal
150 kg

2. Push Ups
Chest Triceps
09/11/2019
30 kcal
100 counts

Bench Press
09/11/2019
300 kcal
150 kg
Chest

Other properties:

No custom properties input for this exercise

Total: 650.00 kcal
Average: 92.86 kcal

```
Listed all suggested exercises.
```

Figure 4. Only the latest `Bench Press` *exercise is displayed*

As seen on the image above, the information from the "Bench Press" on "06/11/2019" is displayed instead of the one on "05/11/2019" (observe that the calories are different).

Contributions to the Developer Guide

Given below are sections I contributed to the Developer Guide. They showcase my ability to write technical documentation and the technical depth of my contributions to the project.

Suggest

Rationale

Beginners now have a plethora of choices and may be overwhelmed when deciding on what exercises to do. Thus, we decided to provide users with sample exercise routines to reduce the inertia of starting this lifestyle change. On other hand, regular gym goers may face a repetitive and mundane exercise routine or may want to experiment with different exercises. As such, put it briefly, we decided to give users the ability to discover exercises based on the characteristics they are interested in.

This feature presents a cohesive function that all users can benefit from. This also makes our application well-rounded so that users can better achieve their fitness goals.

Overview

The sample exercise routines are currently implemented in ExerHealth’s database as a hard-coded set of exercises. More importantly, the `SuggestPossible` command which caters to more experienced gym goers utilises the exercises that the user has already done, in addition to ExerHealth’s database. Hence, we allow users to search for suggestions based on `Muscle` and `CustomProperty`.

Current Implementation

The `SuggestBasic` command displays a list of exercises from our database to the user. The `SuggestPossible` command is created by parsing the user’s inputs to form a `Predicate` before filtering ExerHealth’s database and the user’s tracked exercises.

The following activity diagram summarizes what happens when a user enters a suggest possible command:

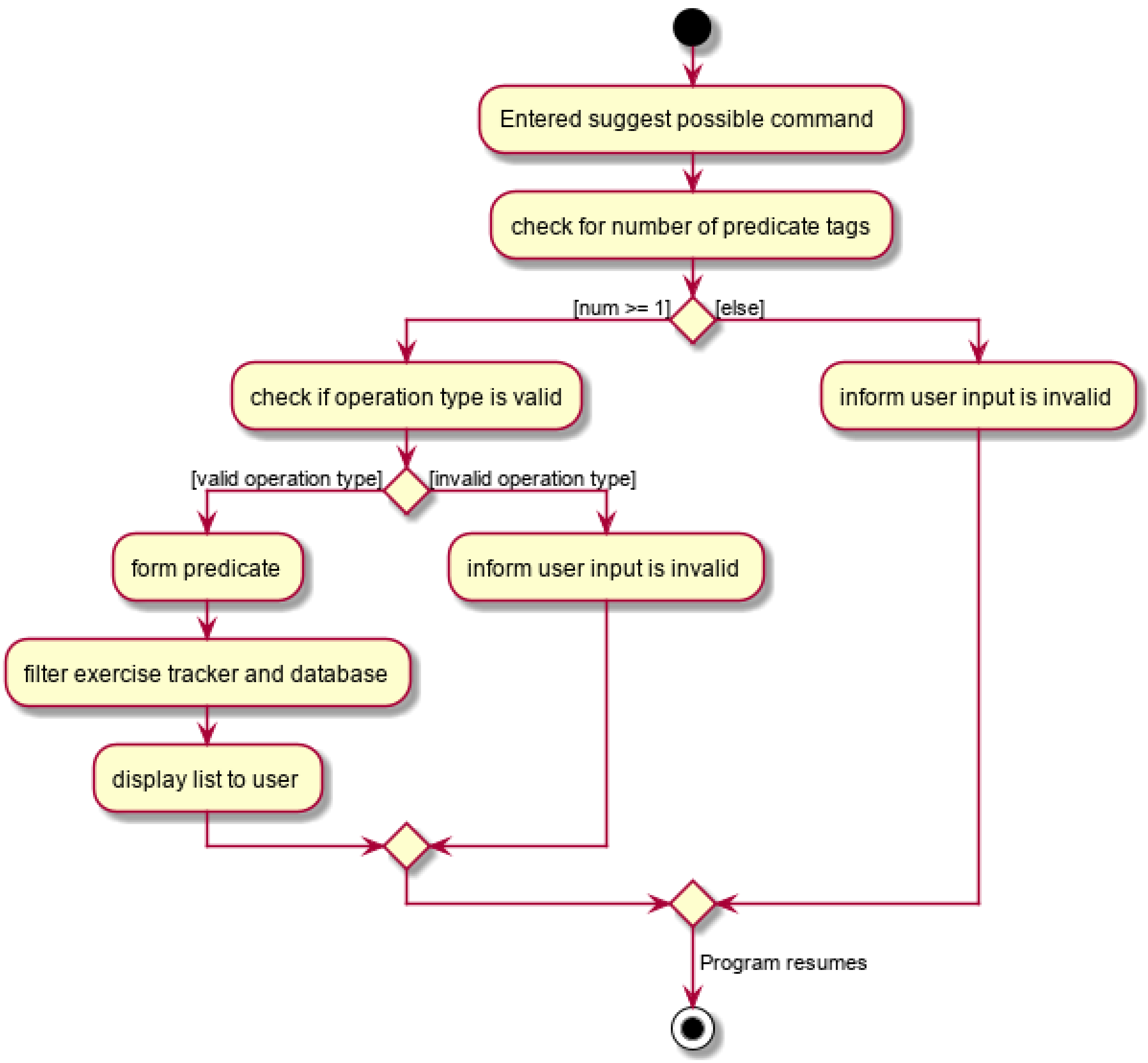

Entered suggest possible command
check for number of predicate tags
[num >= 1]
[else]
check if operation type is valid
inform user input is invalid
[valid operation type]
[invalid operation type]
form predicate
inform user input is invalid
filter exercise tracker and database
display list to user
Program resumes

Figure 5. Activity diagram showing the workflow of a suggest possible command

In detail, when a `SuggestPossibleCommand` is entered, the `Logic` component is responsible for parsing the inputs into a `Predicate`. The `Predicate` is then used to instantiate a `SuggestPossibleCommand`, and later used to filter a list of `Exercise` when the command is executed. The interactions between the multiple objects can be captured using a sequence diagram.

The following sequence diagram shows the sequence flow from the when a user enters a valid `SuggestPossibleCommand`:

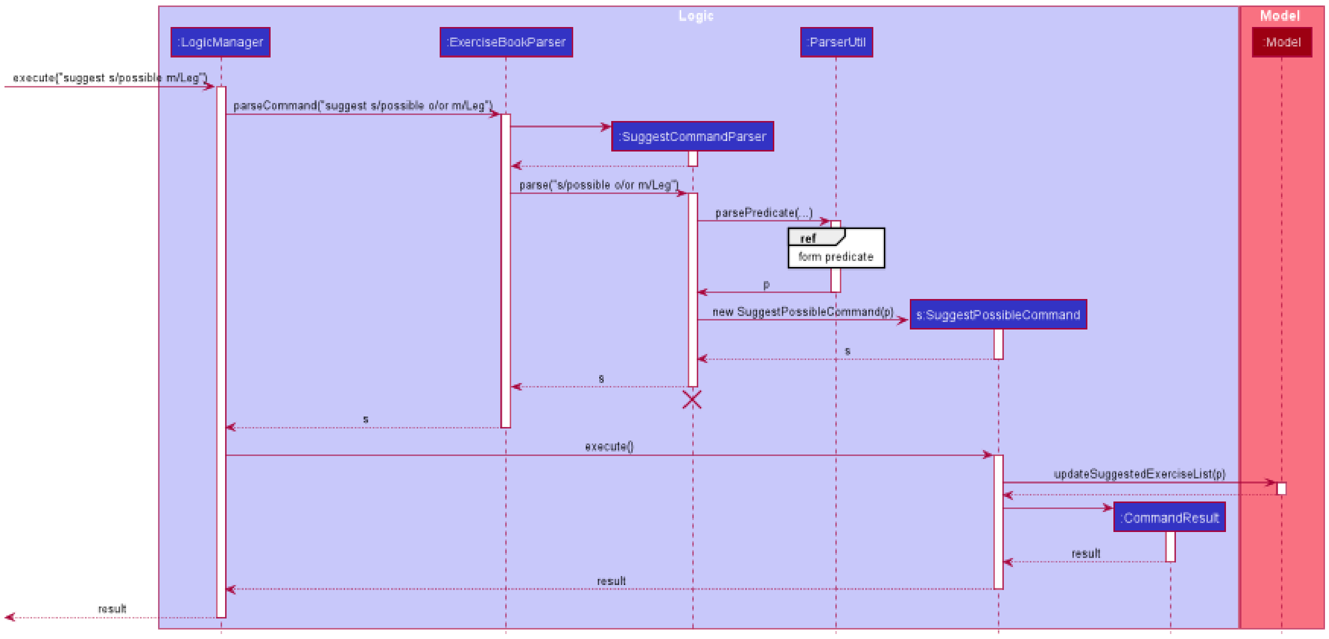


Figure 6. Sequence diagram of a suggest possible command

From the sequence diagram:

1. When the `LogicManager` receive the `execute` command, it calls the `parseCommand` method of `ExerciseBookParser`.
2. `ExerciseBookParser` will receive `suggest` as the command type and instantiate `SuggestCommandParser` to further parse the command.
3. `SuggestCommandParser` will receive `s/possible` as the suggest type and calls the `parsePredicate` method of `ParserUtil` to parse the user input to create an `ExercisePredicate` object (named `p` in the diagram).
4. `SuggestCommandParser` will instantiate `SuggestPossibleCommand` with the `ExercisePredicate` as the constructor parameter.
5. The `SuggestPossibleCommand` object is then returned to `SuggestCommandParser`, `ExerciseBookParser` and lastly back to `LogicManager` to execute.
6. `LogicManager` will proceed to call execute `SuggestPossibleCommand`.
7. `SuggestPossibleCommand` then calls the `updateSuggestedExerciseList` method in `ModelManager`, passing in the predicate to filter the list of suggest exercises.
8. `SuggestPossibleCommand` creates a new `CommandResult` to be returned.

In step 3, the process in which the `ExercisePredicate` object is created can be explored deeper.

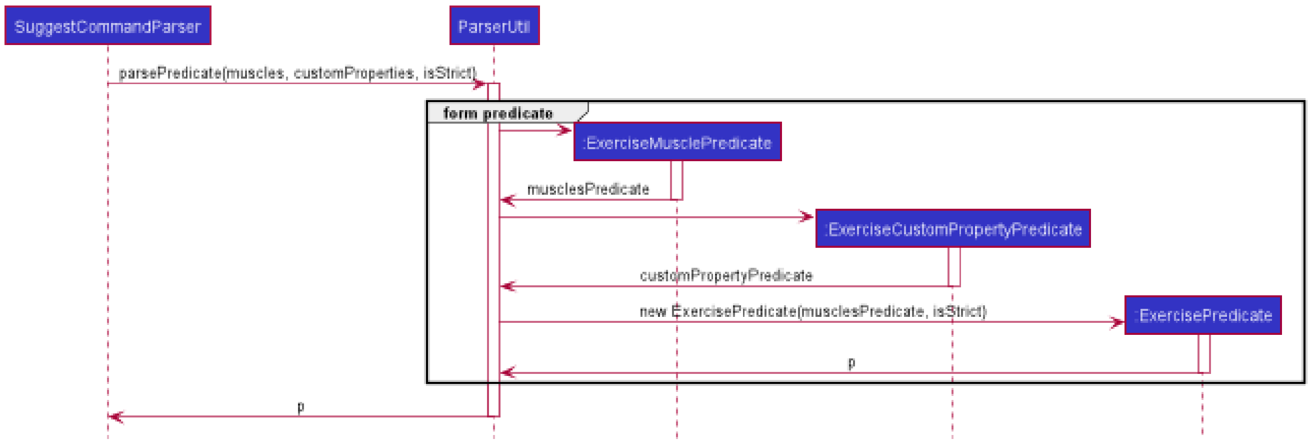

SuggestCommandParser
ParserUtil
parsePredicate(muscles, customProperties, isStrict)
form predicate
:ExerciseMusclePredicate
musclesPredicate
:ExerciseCustomPropertyPredicate
customPropertyPredicate
new ExercisePredicate(musclesPredicate, isStrict)
:ExercisePredicate
p
p

Figure 7. Sequence diagram of how an `ExercisePredicate` is created

From the sequence diagram above:

1. `ParserUtil` creates `ExerciseMusclePredicate` and `ExerciseCustomPropertyPredicate` with the input parameters.
2. Since there were no CustomProperty tags to filter, `ParserUtil` creates `ExercisePredicate` with only the `musclesPredicate` and the boolean `isStrict`.
3. The resulting `ExercisePredicate` is then returned to `ParserUtil` and then `SuggestCommandParser`.

The diagram below shows the structure of a `ExercisePredicate` object. A `ExercisePredicate` contains a `list` of `BasePropertyPredicate`, where each contains a `Collection` of `Muscle` or `CustomProperty`.

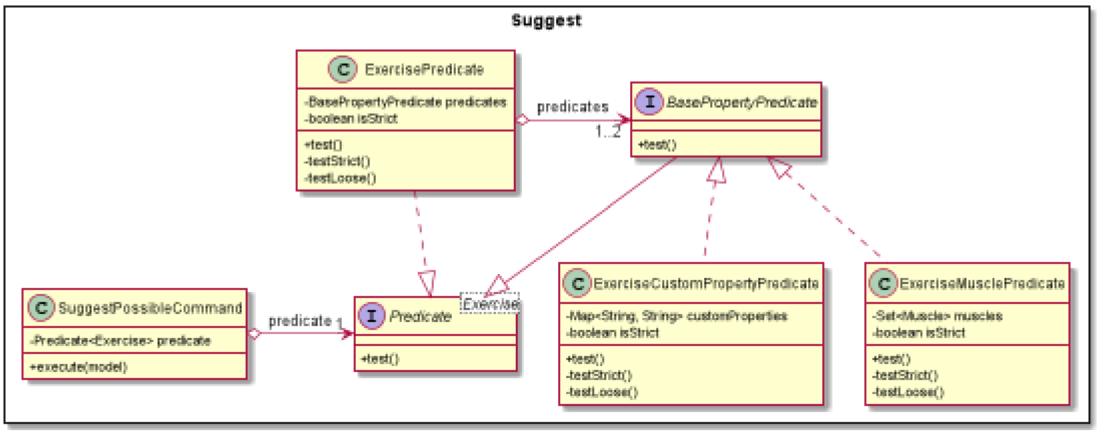


Figure 8. Class diagram of the classes behind the suggest possible feature

Creating classes such as `ExerciseCustomPropertyPredicate` and `ExerciseMusclePredicate` allow us to conduct better testing because we can compare the `Collection` of `Muscle`/`CustomProperty` being considered.

Design Considerations

Aspect: Implementation of predicate creation

- **Choice 1:** `SuggestPossibleCommand` to handle the predicates.
  - Pros:
    - Easy to implement and understand. The class `SuggestPossibleCommand` contains the parsing and creation of the predicate all in one place as it stores the tags, and creates the predicate and filters the list of exercises.
  - Cons:
    - Violation of SRP as `SuggestPossibleCommand`, in addition to updating the model, has to create the predicate.
- **Choice 2 (current choice):** Predicate class to handle all predicates.
  - Pros:
    - Adheres to Single Responsibility Principle (SRP) and Separation of Concern (SoC).
  - Cons:
    - Increases the complexity of the code as more classes are needed, also increasing the lines of code written.