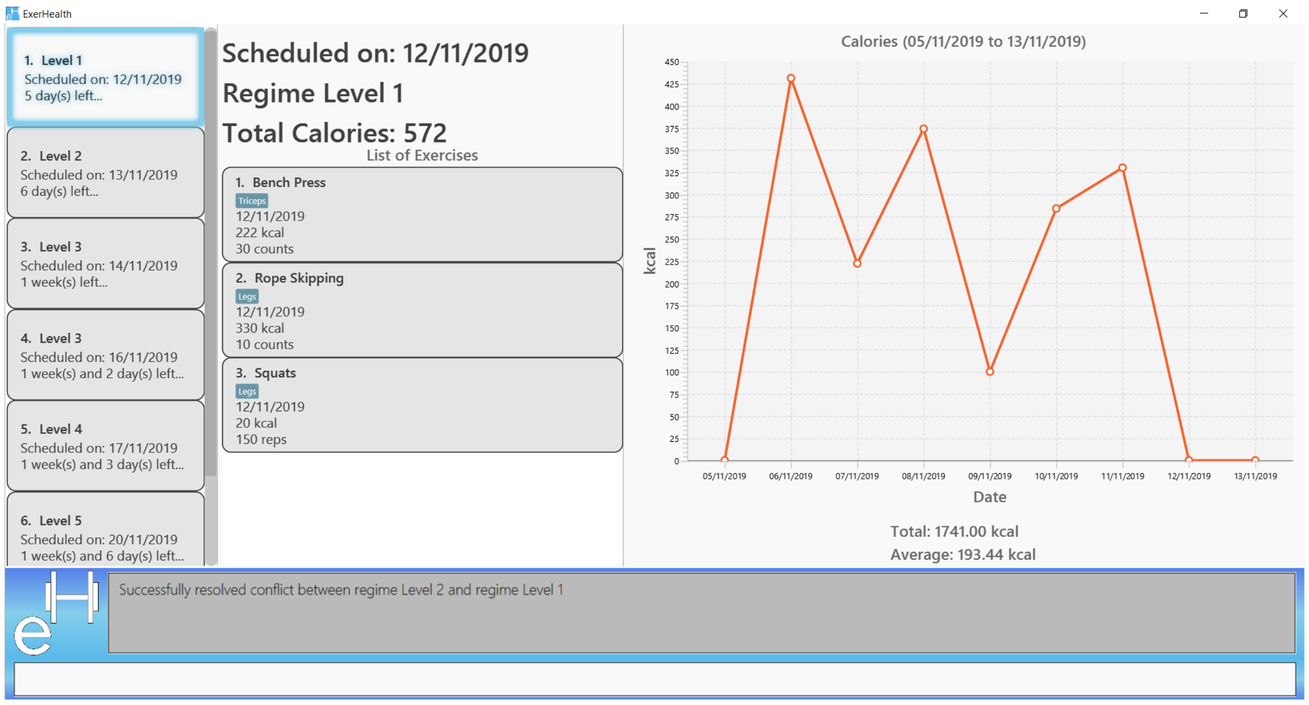
Ho Wei Haw - Project Portfolio

PROJECT: ExerHealth

Overview

ExerHealth is a desktop application that is created by me and another 4 Computer Science students. It enables users to track important information about their exercises and also provides other functionality such as the ability to create their own fitness regimes and the ability to provide statistical analysis of their activities. Users can also define their own properties which they wish to track for their exercises.

Below is a screenshot of our application:


ExerHealth

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

Total: 1741.00 kcal
Average: 193.44 kcal

Successfully resolved conflict between regime Level 2 and regime Level 1

Summary of contributions

- **Major enhancement 1**: Provided a feature that **allows user to define their own custom properties** and **remove them**.
  - What it does: Users can simply add in or remove their own custom properties should they wish to keep track of extra/lesser properties for each exercise.
  - Justification: This feature provides greater flexibility among various users if they wish to include more information which they wish to track. This customisation can help users to tailor suit the app for their personal use.
  - Highlights: This feature complements well with the suggestion feature. By having custom properties, users can now produce exercise suggestions based on the custom properties they had created. Furthermore, the implementation of this feature is quite challenging as there

are a few aspects to take note of: the representation of custom properties in each exercise, the tracking of custom properties that have been created and storage of custom properties.

- **Minor enhancement 1**: Added a `viewcustom` command that allows user to view the custom properties that they have defined.
- **Minor enhancement 2**: Designed the centre panel UI for exercises. This design helps to display both the default and custom properties of each exercise to the user.
- **Minor enhancement 3**: Added a `select` command that allows user to select a specific exercise/regime/schedule/suggestion which they wish to view.
- **Code contributed**: [Code Contribution]
- **Other contributions**:
    - Enhancements to existing features:
        - Updated the UI (Pull requests #129, #133)
        - Wrote additional tests for existing features to increase coverage (Pull requests #140, #194)
        - Involved in the morphing of AddressBook to suit ExerHealth (Pull requests #13)
    - Community:
        - PRs reviewed (with non-trivial review comments): #81, #103
        - Reviewed and provided suggestions for individuals in the class: #86, #229

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.

Custom properties `custom` / `viewcustom`

Adding custom properties: `custom`

Adds in a custom property which you can define for the exercises.

Once a new custom property is created, you can simply use the prefix name which you defined for the property to the `add` and `edit` command to include information for the new property.

- The prefix name can only contain alphabets and should not contain spaces.
- You must choose exactly one of the following as the parameter type for your custom property: `Text`, `Number`, `Date`.
- Every word in the full name of each custom property will be changed to Start Case style, where the first letter of each word is capitalised with the other letters in lower case e.g. `enD DaTe` will be changed to `End Date`.
- The date entered for custom properties with a `DATE` parameter must follow the same requirements as that of add command.
- The text entered for custom properties with a `TEXT` parameter can contain only alphabets.
- The number entered for custom properties with a `NUMBER` parameter must be non-negative.
- You need not include the custom properties when adding a new exercise to the app.

Format: `custom s/PREFIX_NAME f/FULL_NAME p/PARAMETER_TYPE`

TIP The following names and prefix names have been used for existing add / edit command parameters and properties and so, cannot be used.

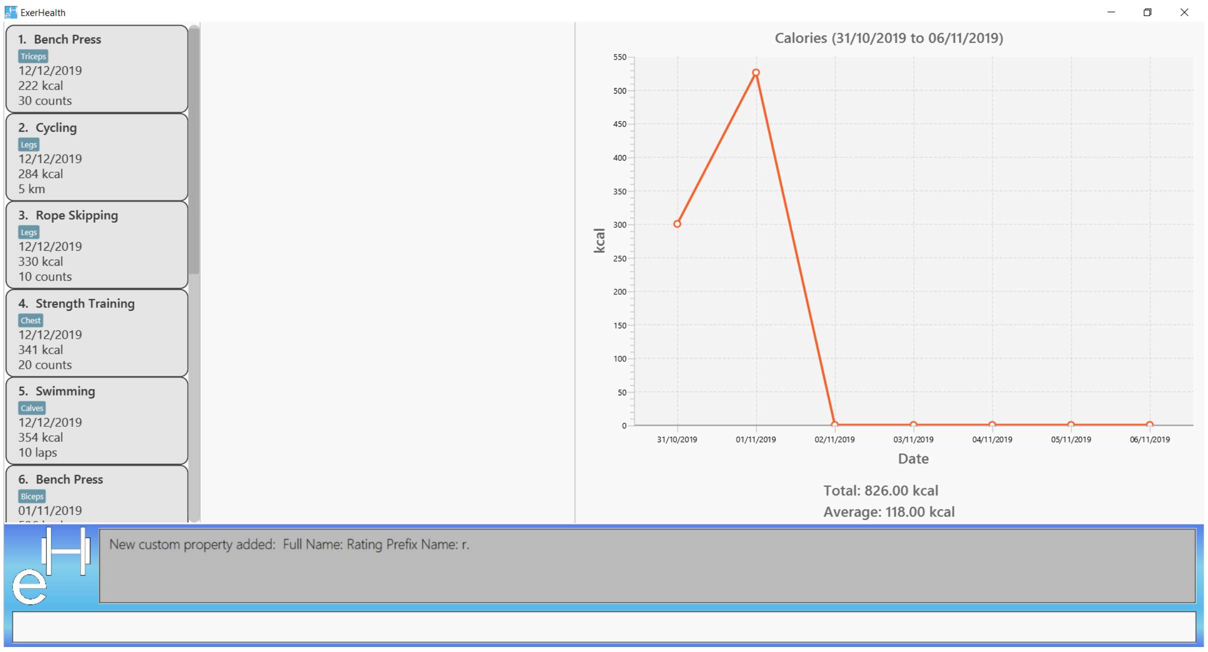
Names used	Prefix names used
Name	n
Date	d
Calories	c
Quantity	q
Unit	u
Muscle	m
-	t
-	i

Example:

- `custom s/r f/Rating p/Number`

Creates a `Rating` property for each of your exercises.

Expected Result:



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

2. Cycling
Legs
12/12/2019
284 kcal
5 km

3. Rope Skipping
Legs
12/12/2019
330 kcal
10 counts

4. Strength Training
Chest
12/12/2019
341 kcal
20 counts

5. Swimming
Calves
12/12/2019
354 kcal
10 laps

6. Bench Press
Biceps
01/11/2019

Total: 826.00 kcal
Average: 118.00 kcal

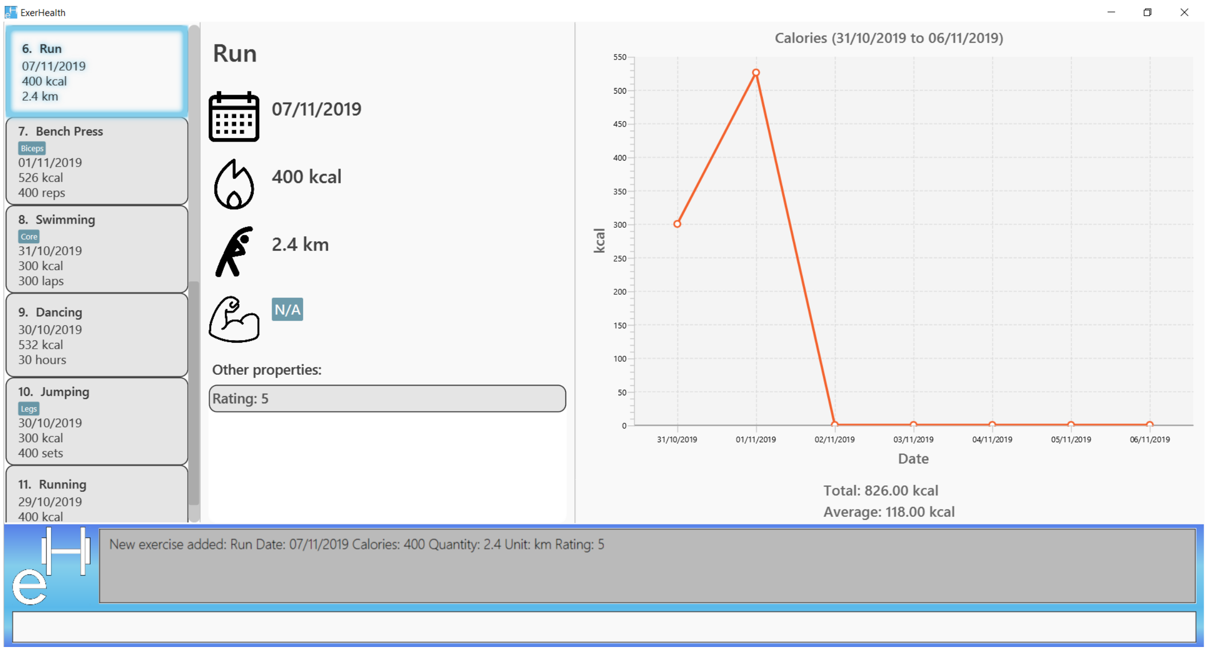
```
New custom property added:  Full Name: Rating Prefix Name: r.
```

Figure 1. Your rating property has been created

You can now add a new exercise with `Rating`!

- `add t/exercise n/Run d/07/11/2019 c/400 q/2.4 u/km r/5`

Expected Result:



6. Run
07/11/2019
400 kcal
2.4 km

7. Bench Press
Biceps
01/11/2019
526 kcal
400 reps

8. Swimming
Core
31/10/2019
300 kcal
300 laps

9. Dancing
30/10/2019
532 kcal
30 hours

10. Jumping
Legs
30/10/2019
300 kcal
400 sets

11. Running
29/10/2019
400 kcal

Run
07/11/2019
400 kcal
2.4 km
N/A
Other properties:
Rating: 5

Total: 826.00 kcal
Average: 118.00 kcal

```
New exercise added: Run Date: 07/11/2019 Calories: 400 Quantity: 2.4 Unit: km Rating: 5
```

Figure 2. A new exercise with the remark property is added

Removing custom properties: `custom`

Removes a custom property which you have previously defined either from a single exercise or from ExerHealth.

In the second case, you will still be able to add back the deleted custom property if you wish to.

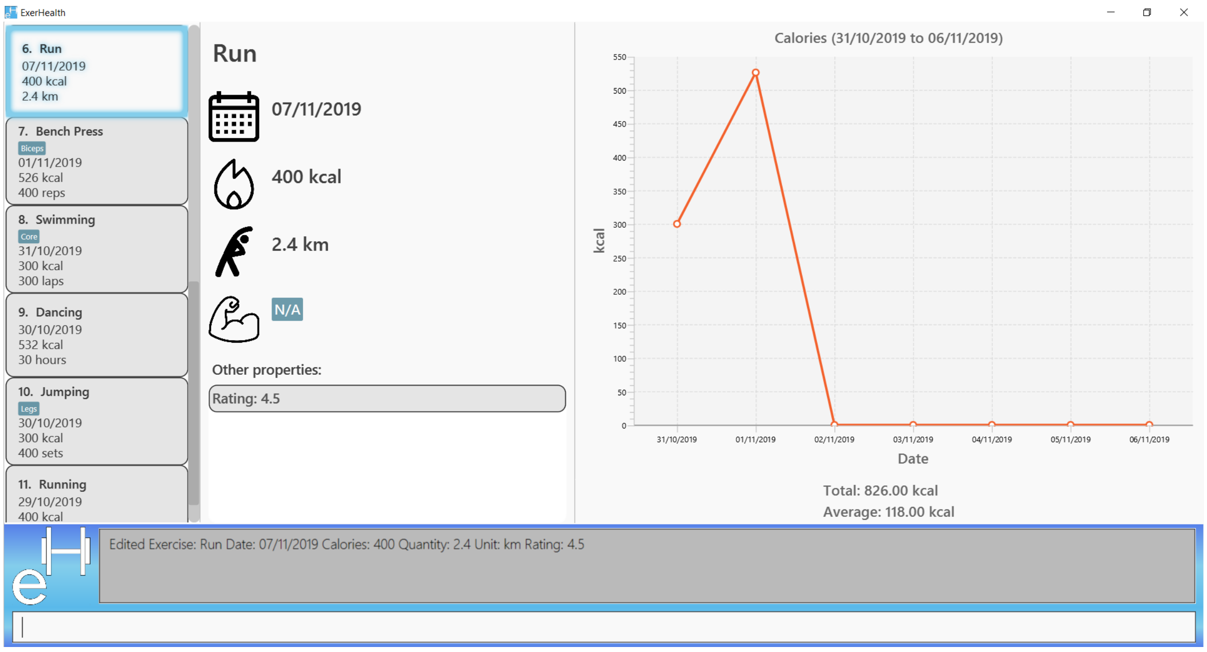
- `FULL_NAME` denotes the name of the previously defined custom property.
- The index, if provided, must be a positive integer 1, 2, 3, …

Format: `custom rm/FULL_NAME [i/INDEX]`

Example:

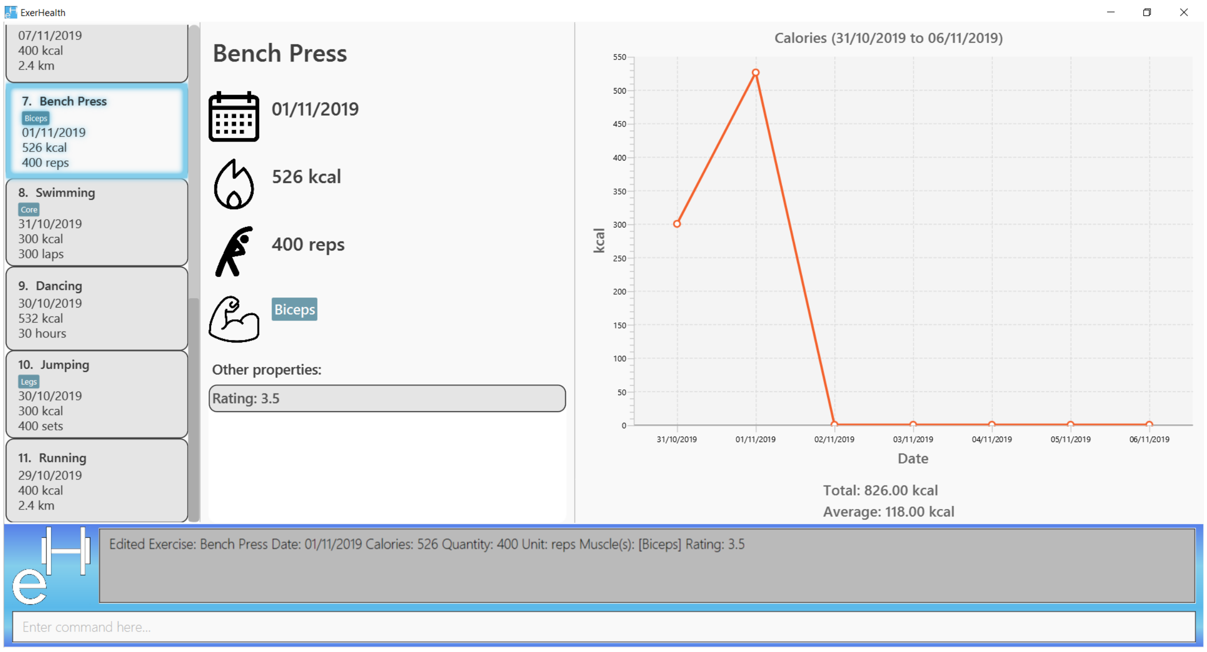
- `custom rm/Rating`

Before the execution, the `Rating` property will be present in exercises that have them.



Edited Exercise: Run Date: 07/11/2019 Calories: 400 Quantity: 2.4 Unit: km Rating: 4.5

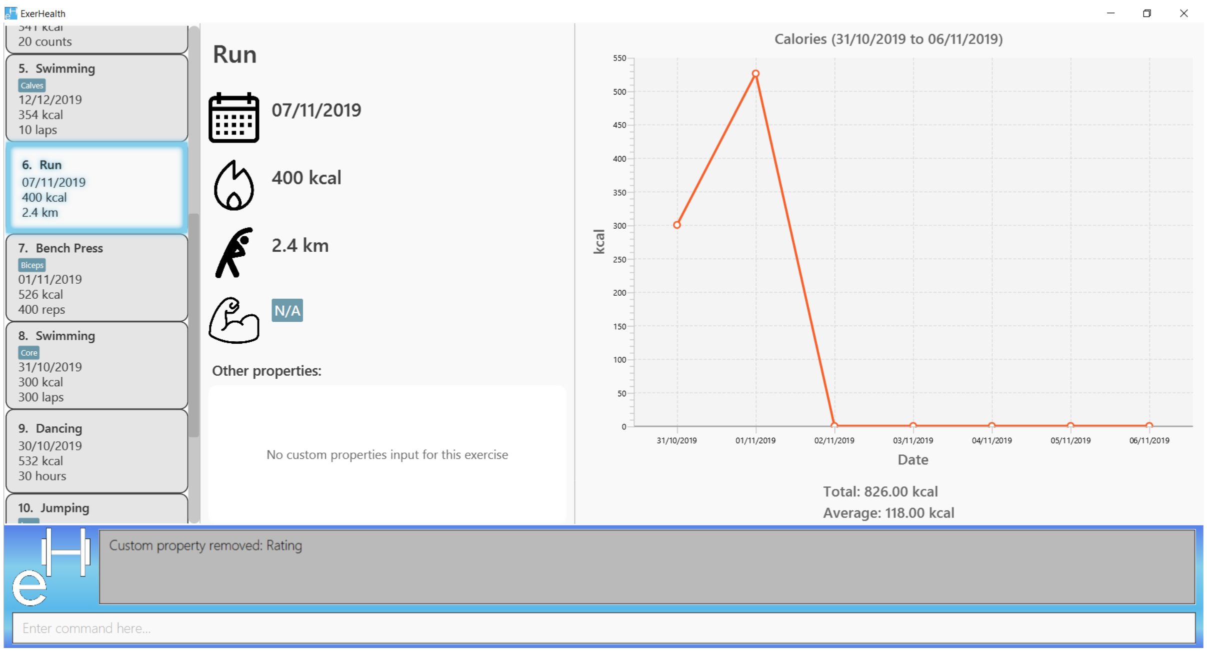
Figure 3. A rating of 4.5 for exercise 6



Edited Exercise: Bench Press Date: 01/11/2019 Calories: 526 Quantity: 400 Unit: reps Muscle(s): [Biceps] Rating: 3.5

Figure 4. A rating of 3.5 for exercise 7

After the execution, the `Rating` property will be removed from all of the exercises and the app as illustrated in the next few figures.


ExerHealth

341 kcal
20 counts

5. Swimming
Calves
12/12/2019
354 kcal
10 laps

6. Run
07/11/2019
400 kcal
2.4 km

7. Bench Press
Biceps
01/11/2019
526 kcal
400 reps

8. Swimming
Core
31/10/2019
300 kcal
300 laps

9. Dancing
30/10/2019
532 kcal
30 hours

10. Jumping

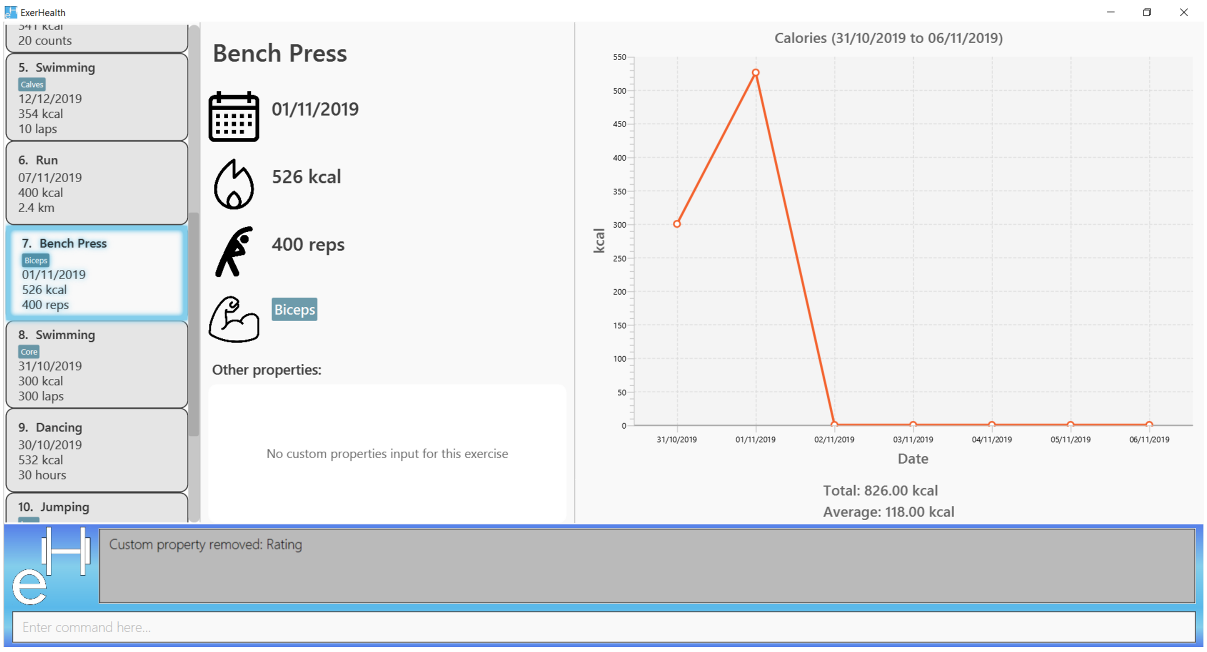
Run
07/11/2019
400 kcal
2.4 km
N/A
Other properties:
No custom properties input for this exercise

Calories (31/10/2019 to 06/11/2019)
Total: 826.00 kcal
Average: 118.00 kcal

Custom property removed: Rating

Enter command here...

Figure 5. Rating in exercise 6 is removed


ExerHealth

341 kcal
20 counts

5. Swimming
Calves
12/12/2019
354 kcal
10 laps

6. Run
07/11/2019
400 kcal
2.4 km

7. Bench Press
Biceps
01/11/2019
526 kcal
400 reps

8. Swimming
Core
31/10/2019
300 kcal
300 laps

9. Dancing
30/10/2019
532 kcal
30 hours

10. Jumping

Bench Press
01/11/2019
526 kcal
400 reps
Biceps
Other properties:
No custom properties input for this exercise

Calories (31/10/2019 to 06/11/2019)
Total: 826.00 kcal
Average: 118.00 kcal

Custom property removed: Rating

Enter command here...

Figure 6. Rating in exercise 7 is also removed

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.

Custom feature

Rationale

A quick conversation with a few of our friends revealed that there are many properties which they intend to keep track for exercises. However, it is unlikely that we can implement all of these properties for the exercises as there may be too much overhead and we can never be certain that we have met all of the users' needs.

Overview

This feature is facilitated by both `PropertyBook` and `CustomProperty`. Whenever a user adds a newly defined custom property, a `CustomProperty` object will be created which is stored in `PropertyBook`. Its corresponding prefix and full name will be tracked by `PropertyBook` to avoid clashes in their uses.

Current Implementation

`CustomProperty` encapsulates a single custom property that the user defines. It contains information such as name, prefix and parameter type of the custom property. The parameter type is supported by an enumeration class `ParameterType` and is restricted to one of the following 3 types: `Number`, `Text`, `Date`.

`PropertyBook` serves as a singleton class that helps to manage all of the custom properties that have been defined by the user. This class acts as an access point for any information relating to the creation or deletion of custom properties.

To keep track of the custom properties and its relevant information, the following are used:

1. `customProperties`: A set containing all of the `CustomProperty` objects that have been created.
2. `customPrefixes`: A set containing all of the `Prefix` objects associated with existing custom properties.
3. `customFullNames`: A set containing the full names of the existing custom properties.
4. `defaultPrefixes`: A set containing all of the `Prefix` objects associated with default properties and parameter types.
5. `defaultFullNames`: A set containing all of the full names of default properties.

Custom names and prefixes are separated from its default counterparts to ensure that the default names and prefixes will always be present when the `PropertyBook` is first initialised.

To help facilitate `PropertyBook` in its custom properties management, the following main methods are implemented:

1. `PropertyBook#isPrefixUsed(Prefix)`: Checks if the given prefix has been used by a default or custom property.
2. `PropertyBook#isFullNameUsed(String)`: Checks if the given name has been used by a default or custom property.
3. `PropertyBook#isFullNameUsedByCustomProperty(String)`: Checks if the given name has been used by a custom property
4. `PropertyBook#addCustomProperty(CustomProperty)`: Adds the new custom property. Each time a custom property is added, the prefix set in `CliSyntax` is also updated.
5. `PropertyBook#removeCustomProperty(CustomProperty)`: Removes a pre-defined custom property. Its

associated prefix is also removed from the prefix set in `CliSyntax`.

All of the crucial associations mentioned above are summarised in the next class diagram.

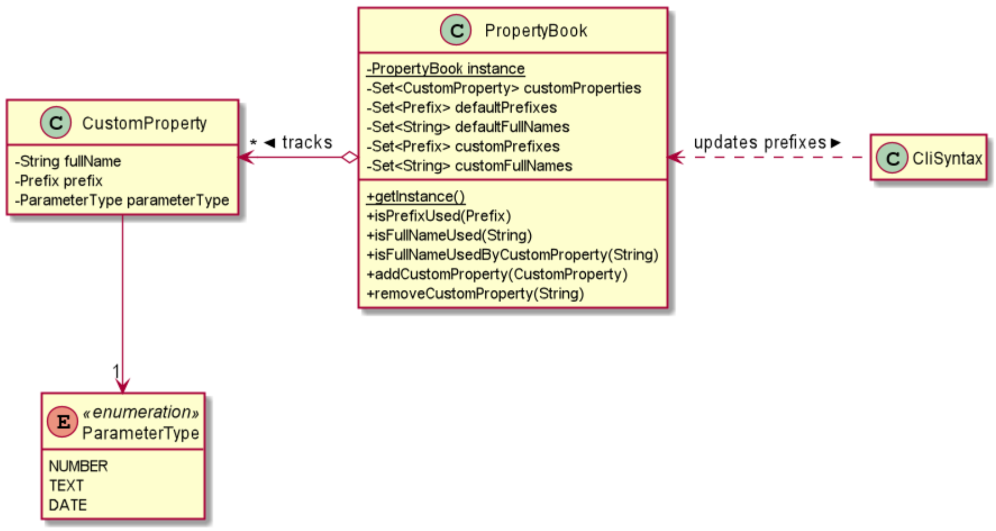

PropertyBook
-PropertyBook instance
-Set<CustomProperty> customProperties
-Set<Prefix> defaultPrefixes
-Set<String> defaultFullNames
-Set<Prefix> customPrefixes
-Set<String> customFullNames
+getInstance()
+isPrefixUsed(Prefix)
+isFullNameUsed(String)
+isFullNameUsedByCustomProperty(String)
+addCustomProperty(CustomProperty)
+removeCustomProperty(String)
CustomProperty
-String fullName
-Prefix prefix
-ParameterType parameterType
* ◂ tracks
updates prefixes ▸
CliSyntax
1
«enumeration»
ParameterType
NUMBER
TEXT
DATE

Figure 7. Class diagram of the associations of `PropertyBook` *and* `CustomProperty`

Adding Custom Properties

To add a new custom property for the exercises, the user can do it through the command `custom s/PREFIX_NAME f/FULL_NAME p/PARAMETER_TYPE`. Examples include `custom s/r f/Rating p/Number` and `custom s/ed f/Ending Date p/Date`.

The following sequence diagram will illustrate how the custom operation works when a custom property is **successfully added**.

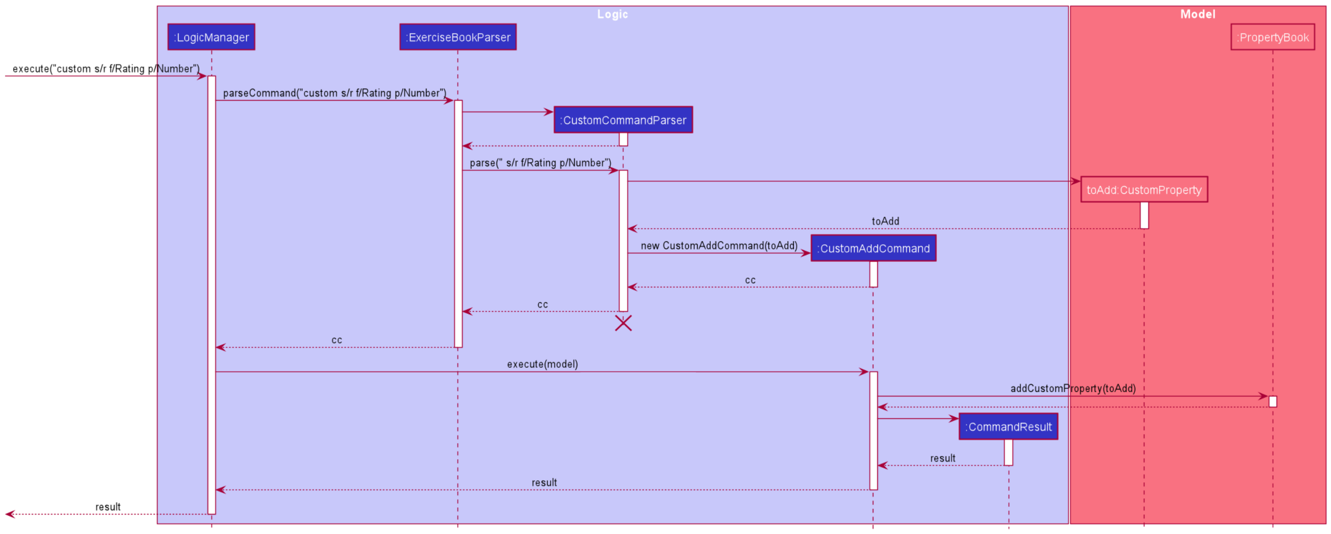


Figure 8. Sequence diagram of a successful addition of a custom property

For further clarity, one can identify the above diagram with the following sequence of steps:

Step 1: User first defines the custom property they wish to add for the exercises.

Step 2: The custom property will be parsed by the app’s parser and a new `CustomProperty` object is created.

Step 3: This `CustomProperty` object will be returned together with a newly created `CustomAddCommand` object.

Step 4: The `execute` method of the `CustomAddCommand` method will be called and the `CustomProperty` object will be added to `PropertyBook`.

Step 5: Finally, a `CommandResult` object will be created and returned.

The above steps illustrate the main success scenario. However, not all additions of a custom property will be successful. The next activity diagram shows the workflow when a new custom property is defined.

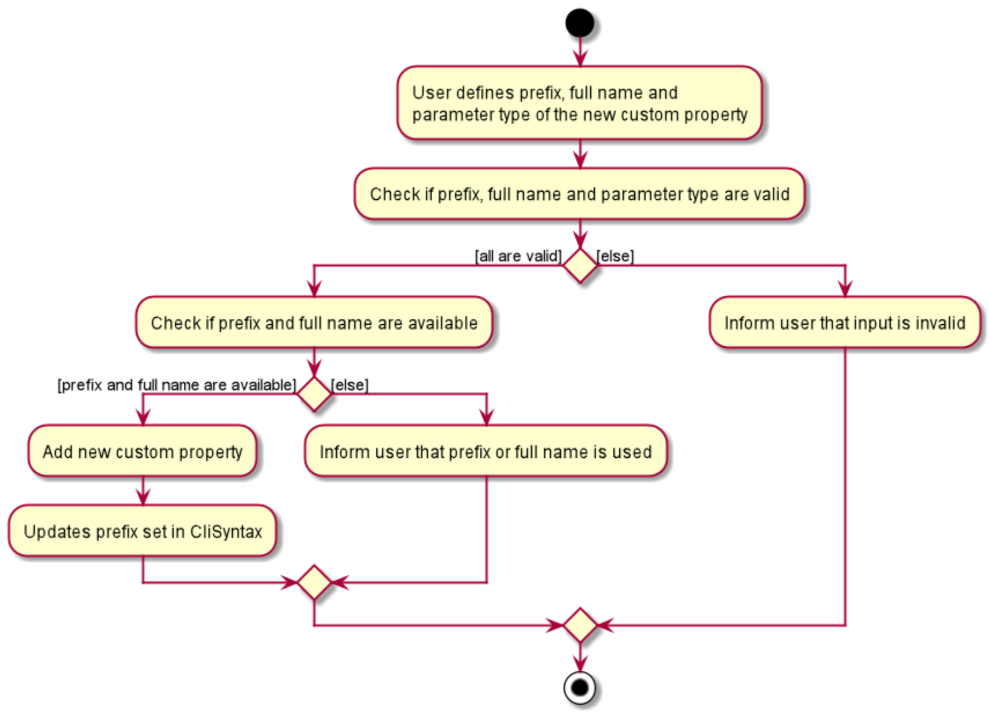

User defines prefix, full name and parameter type of the new custom property
Check if prefix, full name and parameter type are valid
[all are valid]
[else]
Check if prefix and full name are available
Inform user that input is invalid
[prefix and full name are available]
[else]
Add new custom property
Inform user that prefix or full name is used
Updates prefix set in CliSyntax

Figure 9. Activity diagram of the workflow when a new custom property is added

Once a custom property is successfully added into `PropertyBook`, the user can use the prefix of the custom property in `add` or `edit` command.

Removing Custom Properties

Should a user wish to remove a custom property from all of the exercises, he/she can simply make use of the command `custom rm/FULL_NAME`. Alternatively, if the user wishes to remove a custom property just from a single exercise, he/she can choose to enter `custom rm/FULL_NAME i/INDEX` instead. A custom property that has been removed from the `PropertyBook` can be re-added back if the user chooses to.

Viewing Custom Properties

To allow user to view the custom properties which they have defined, they can enter `viewcustom` which will produce a table that lists out the name, prefix and parameter type of each custom property they have defined.

Design Considerations

Aspect: `PropertyBook` design

- **Choice 1 (Current choice)**: Represent `PropertyBook` as a singleton class that will act as the only access point for the addition and removal of custom properties.
  - Pros: Having a singleton helps to provide more utility for methods that rely on the `CustomProperty` objects that have been created.
  - Cons: It makes testing much difficult as the results from the previous test cases are carried over. Furthermore, it increases coupling across the code base.
- **Choice 2**: Represent `PropertyBook` as a usual Java object that can be instantiated many times.
  - Pros: This reduces coupling and makes testing easier as a new `PropertyBook` object independent of the other tests can be created for different tests.
  - Cons: There could be situations where 2 instances of `PropertyBook` objects are created and the addition of a custom property is done to only one instance and not in the other.

Choice 2 seems like an ideal selection at the start as coupling is reduced and it reduces the likelihood of untestable code. However, certain issues arise:

1. `AddCommandParser` and `EditCommandParser` have to gain access to the `CustomProperty` in order to ensure that the values entered for the custom properties in the add/edit commands are valid. However, as the `ExerciseBookParser` in the original code base only takes in a `String` as a parameter, there has to be another way of retrieving the custom properties. While we can change the `ExerciseBookParser` to take in a data structure containing `CustomProperty` objects, this does not seem good as its responsibility is just to ensure that a predefined command is entered and is passed to the correct command parser. A slightly better choice in this case is to make the data structure holding the `CustomProperty` objects a static variable and parsers that require it can access it directly.
2. If the data structure holding the `CustomProperty` object is to be made static, it means that this information is shared among all of the `PropertyBook` instances if Choice 2 was implemented instead. Thus, `PropertyBook` is acting similarly to a singleton.

As such, after much consideration, Choice 1 was implemented. To ensure that testing can be carried out better, the data inside `PropertyBook` is cleared before each test.