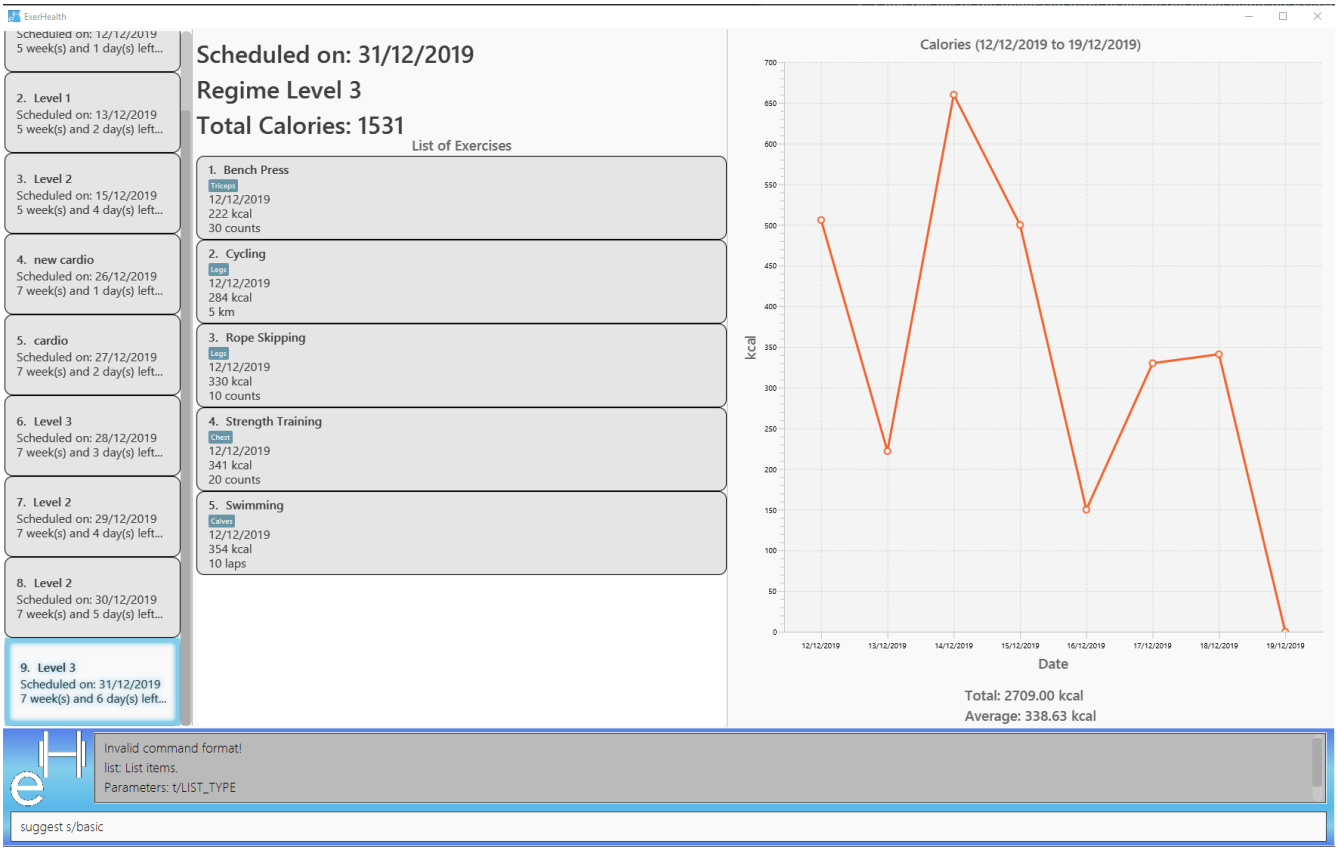
Tan Chee Peng - Project Portfolio

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:



My role in this project was to design and develop the scheduling and resolving of scheduling conflicts features. The following sections will illustrate these additions to the application in finer details, as well as any relevant documentation that I have added into the user and developer guide.

Summary of contributions

This section shows a summary of the contributions I have made to the codebase of ExerHealth. It also shows the overview of the documentation provided and other helpful contributions to the

team project.

- **Major enhancement 1**: Added the ability to `schedule` exercise regimes
  - What it does: The `schedule` command will allow users to schedule a stored exercise regime at a particular date of their choice.
  - Justification: This feature will enable users to keep track and plan what exercise regimes they are supposed to complete. It also acts as a calendar to help users keep track of the regimes in the next few days, weeks or months.
  - Highlights: This enhancement works well with existing features as well as future commands that will be added to ExerHealth. An in-depth analysis of design alternatives was necessary to ensure that future developers, including me, are able to extends and enhance this feature if need be. The implementation itself was challenging because scheduling can lead to conflicts in what regimes to complete at a particular date. However, this difficulty was remedied as described in **Major enhancement 2**.
- **Major enhancement 2**: Added the ability to `resolve` scheduling conflicts
  - What it does: The `resolve` command will allow users to `resolve` conflicts that arise when two regimes are scheduled on the same date.
  - Justification: This feature will allow users the flexibility to select which exercises they wish to do from the conflicting scheduled regimes. It also allows users to store the resolved regime and use it in the future.
  - Highlights: This enhancement works well with **major enhancement 1** as there is a direct link between the two. The implementation of this feature was particularly challenging because multiple resources had to be kept track of. Also, there was a need to prevent users from proceeding on with normal usage of the program until they resolve the scheduling conflict. The aforementioned requirement was technically challenging and multiple facets had to be considered in the implementation of the feature.
- **Minor enhancement 1**: Added the display panel in the center of our UI to display more information on the selected resource on the left panel of our UI.
- **Code contributed**: [RepoSense]
- **Other contributions**:
  - Project management:
    - Maintainer of repository on GitHub
    - Managed releases `v1.2` - `v1.4` (3 releases) on GitHub
  - Testing:
    - Refactored large portions of testing codebase for fellow developers: #120
    - Wrote tests for existing and new features to increase code coverage (Pull requests: #119, #126)
  - Community:
    - PRs reviewed (with non-trivial review comments): #81, #90, #102

Contributions to the User Guide

We had to update the User Guide that was provided by the basic application(**addressbook**) so that it will reflect the features that we have added.

> *Given below are sections I contributed to the **ExerHealth User Guide**. They show the additions I have made for the `schedule` and `resolve` features. The sections below will showcase my ability to write documentation targeting end-users.*

Scheduling exercises: `schedule`

Schedules a regime

Schedules an exercise regime for a certain date. If the regime clashes with another scheduled regime, you will be prompted to resolve the conflict via a popup window. Refer to [resolve] for details on resolving scheduling conflicts.

Format: `schedule n/REGIME_NAME d/DATE`

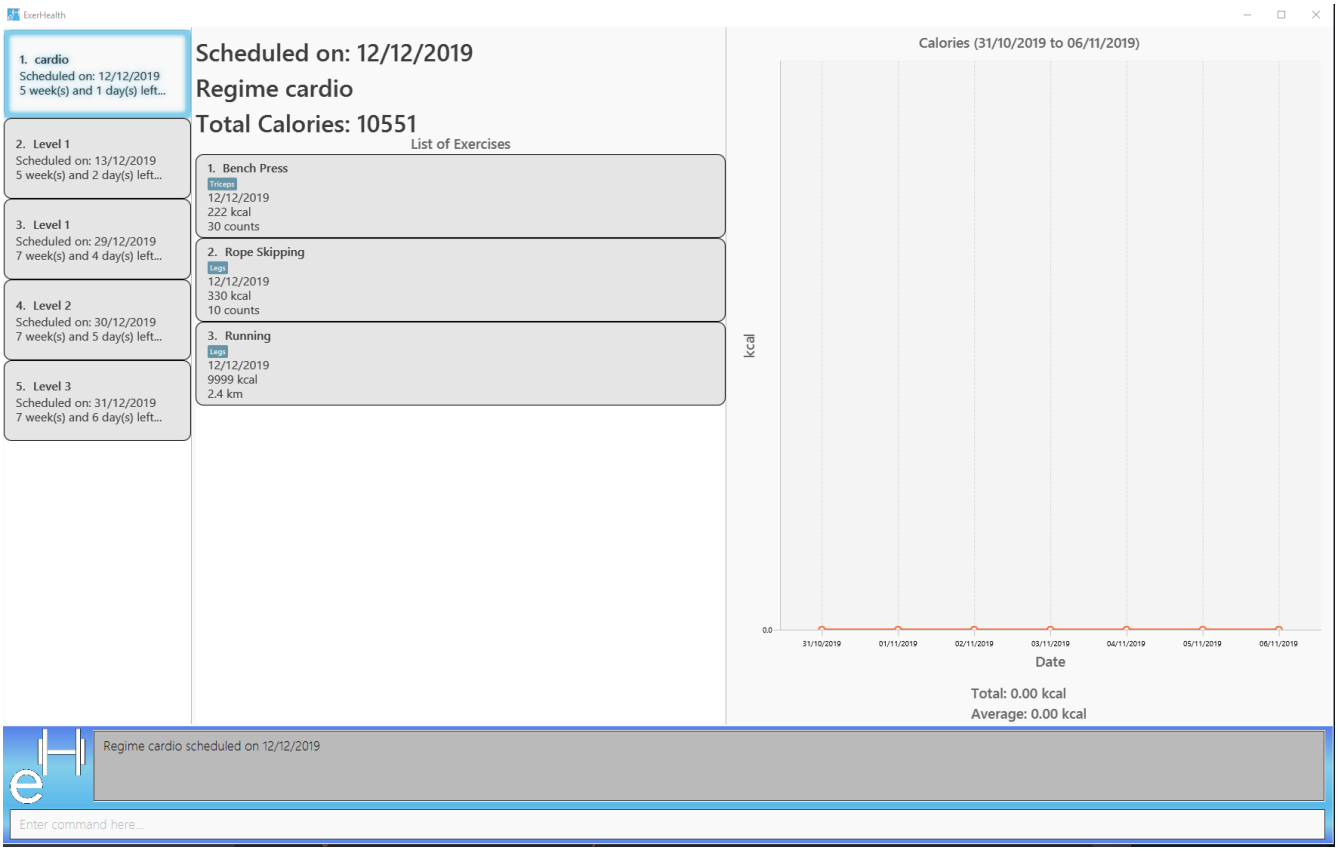
- To schedule a regime, `REGIME_NAME` must exist in your regime list. You can use `list t/regime` command to view the regimes that you currently have.
- The format of `DATE` must be of the form `dd/MM/YYYY`. For example, `12/12/2019` or `01/01/2020`. Even if the number has only one digit, please ensure you include a 0 at the front to ensure that it adheres to the format required.
- You will not be allowed to schedule a regime on a date that falls before the date displayed on your system clock
- You are allowed to schedule a regime of the same name on the same date. The resolve window will pop up for you as per normal.

Example 1:

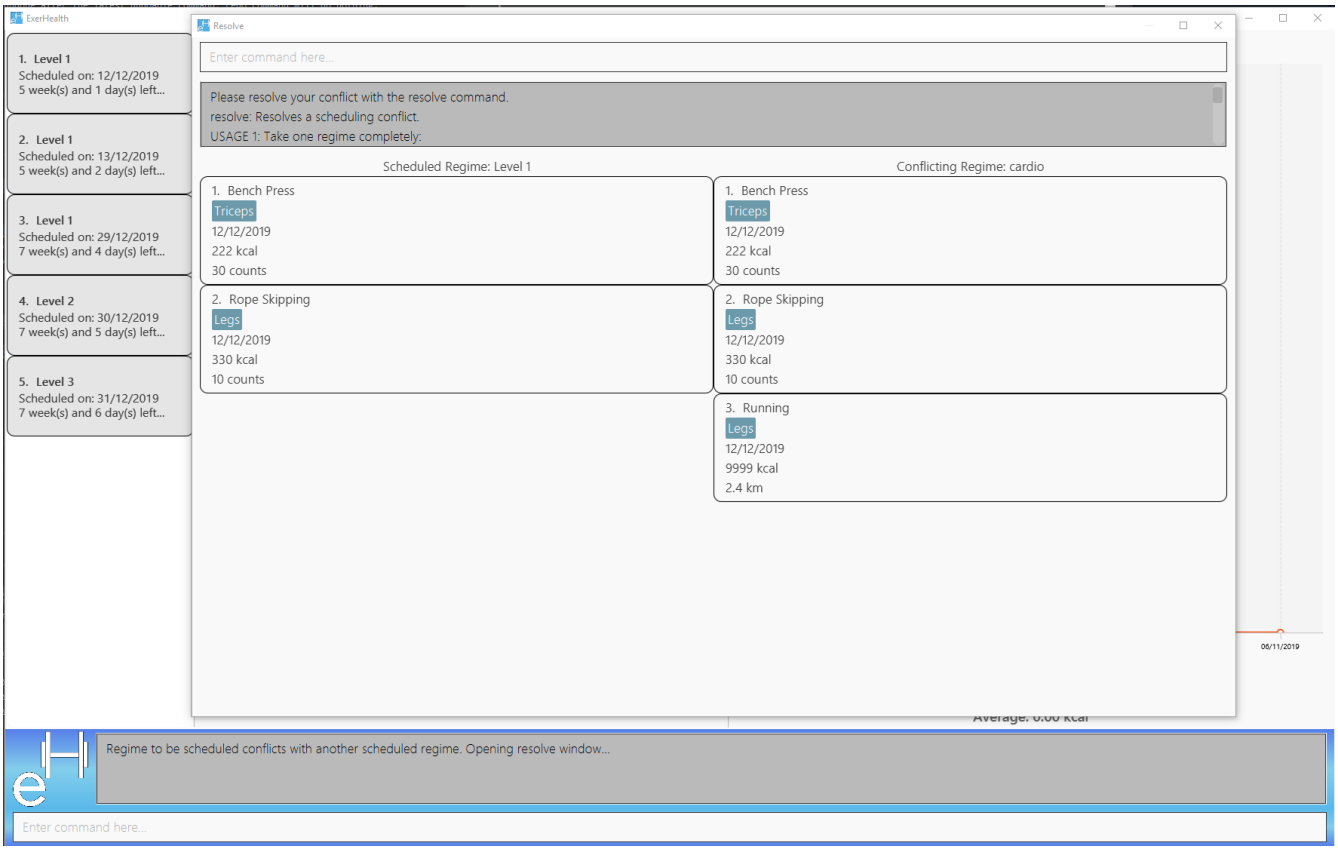
- `schedule n/cardio d/12/12/2019`

Schedules the regime called `cardio` on the date `12/12/2019`. If there are no other regimes scheduled on `12/12/2019` then the command is successful. Otherwise, you will be prompted to resolve the scheduling conflict.

Expected Result:



Expected Result (if other schedule exist on `12/12/2019`):



If the resolve window pops up for you, please refer to [resolve] for details on resolving a scheduling conflict.

Completes a schedule regime

Once a scheduled regime is completed, you can add that completed schedule to the exercise tracker. The schedule is then deleted from the scheduling list.

Format: `schedule i/INDEX_OF_SCHEDULE`

> **TIP** The `INDEX_OF_SCHEDULE` provided must be a valid index from your schedule list. Please use `list t/schedule` to view you index of the schedule you wish to complete.

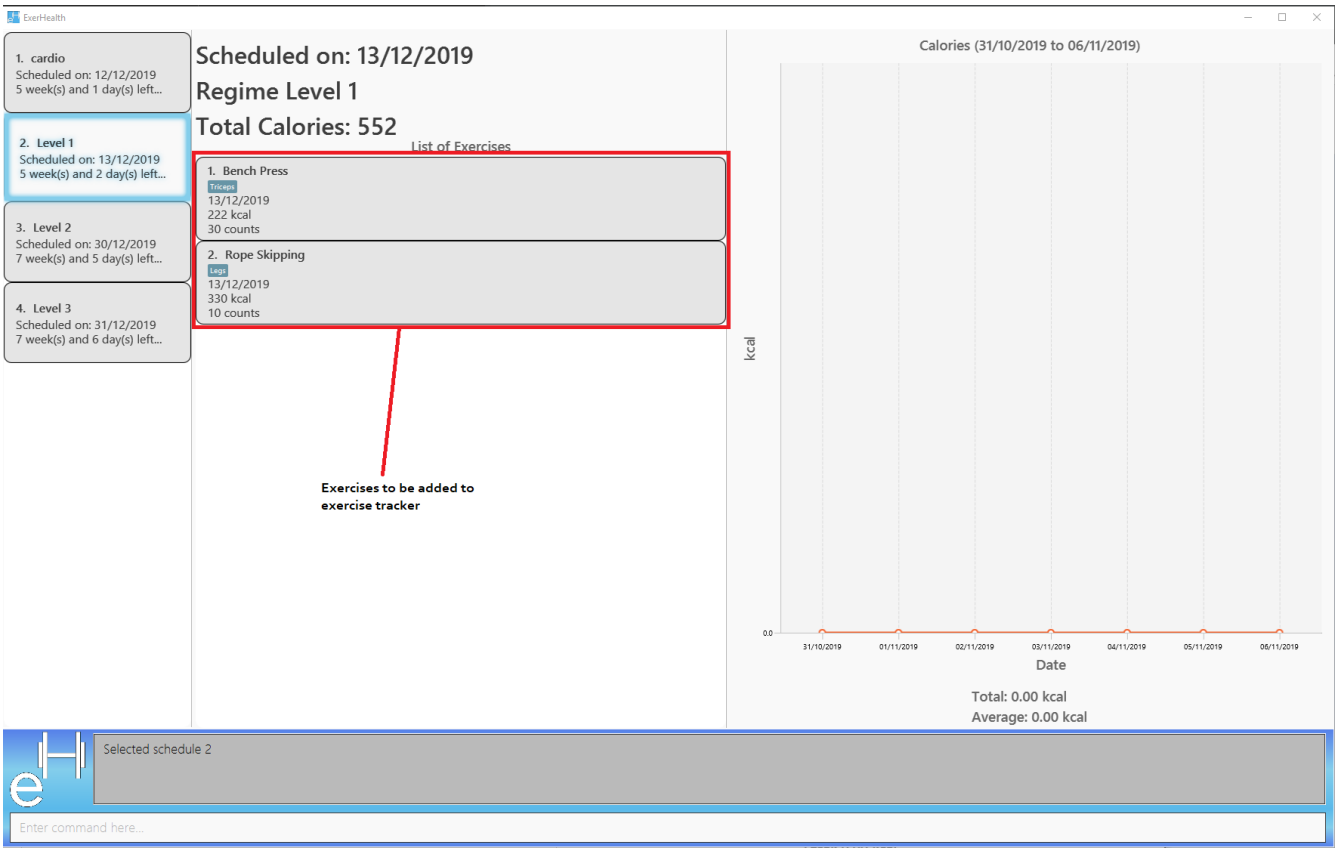
Example:

- `schedule i/2`

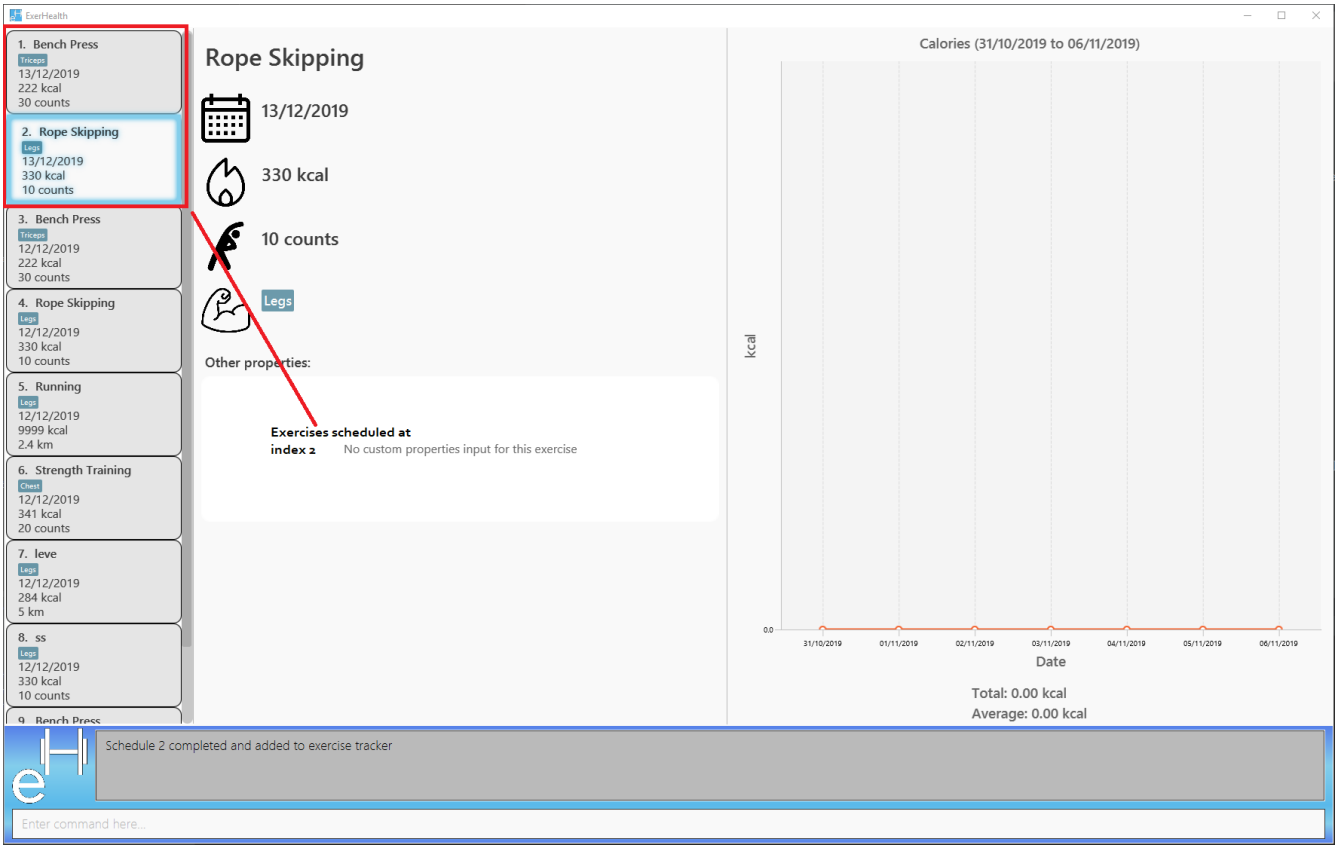
Completes all the exercises that are in the schedule at index `2`. All the exercises scheduled will be added to the exercise list and the schedule at index `2` is deleted.

Expected Result:

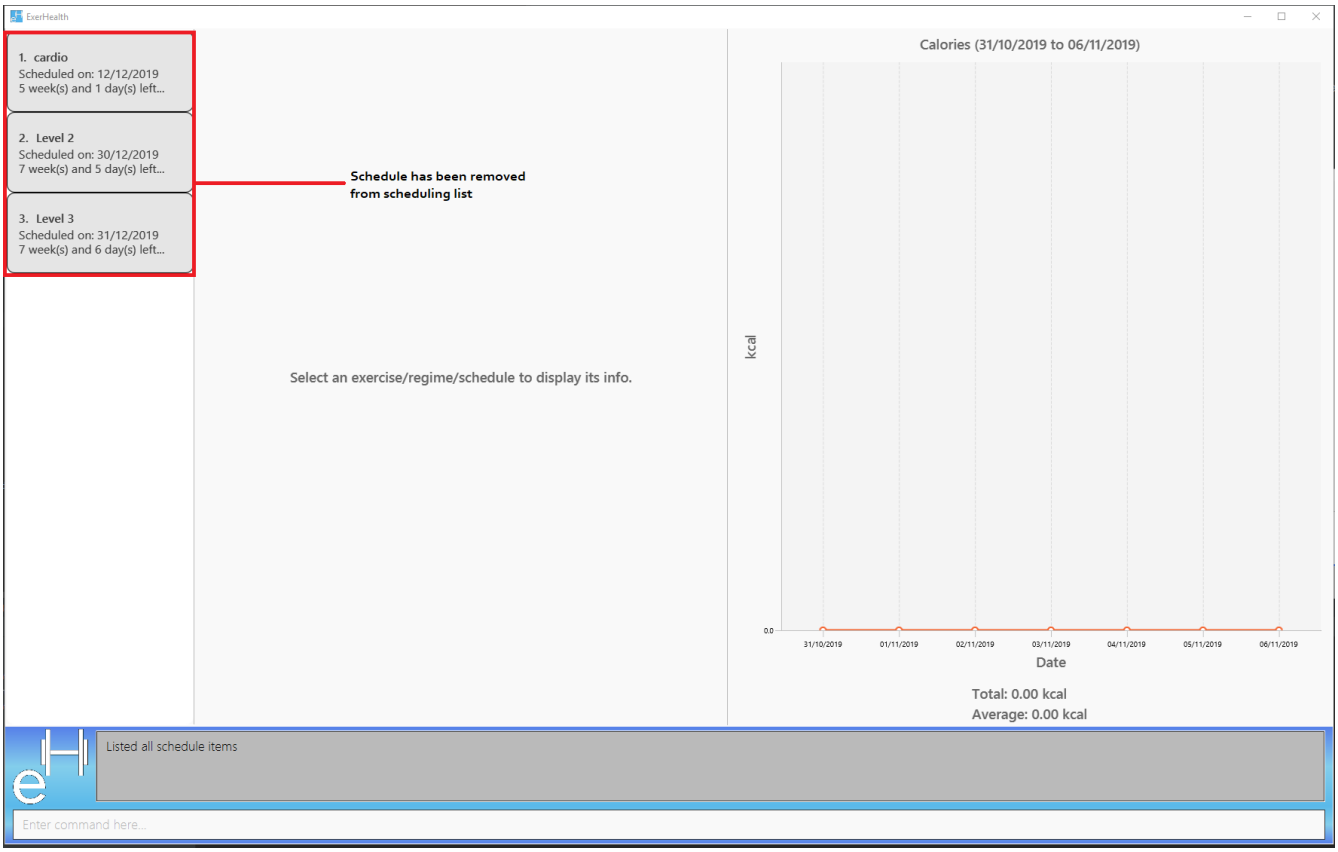
Before execution of `schedule i/2`



In your exercise tracker



In your schedule tracker after completing execution of `schedule i/2`



Contributions to the Developer Guide

We had to update the Developer Guide that was provided by the basic application(**addressbook**) so

it will convey to future developers the implementation details of our features.

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

Resolve feature

Rationale

There are multiple times where if the user wishes to schedule a regime, they find themselves in trouble over which kind of exercise regime they can fit into their schedule. The motivation behind this feature is so that users can customise their own schedules to their own liking. The alternative of an auto scheduler will restrict users from having the regime of their liking be scheduled. Instead of forcing users to adhere to some pre-generated resolution, we allow the users to make their own choice and choose their own exercise regime to be scheduled.

Implementation

The resolve feature is used when there is a scheduling conflict that happens within ExerHealth. This feature will alter the state of the program. The state is known by `MainApp` and it is either `State.IN_CONFLICT` or `State.NORMAL`. Only when the state is `State.IN_CONFLICT` will `resolve` commands be allowed.

For the implementation of the resolve feature, the `ResolveCommand` will hold a `Conflict` object which is then passed into `Model`. The concrete implementation, `ModelManager` then resolves the conflict that is being held there. Each `Conflict` object will hold 1 conflicting `schedule` and 1 `schedule` that was originally scheduled on the date.

Shown below is the class diagram for the implementation of the `Resolve` feature.

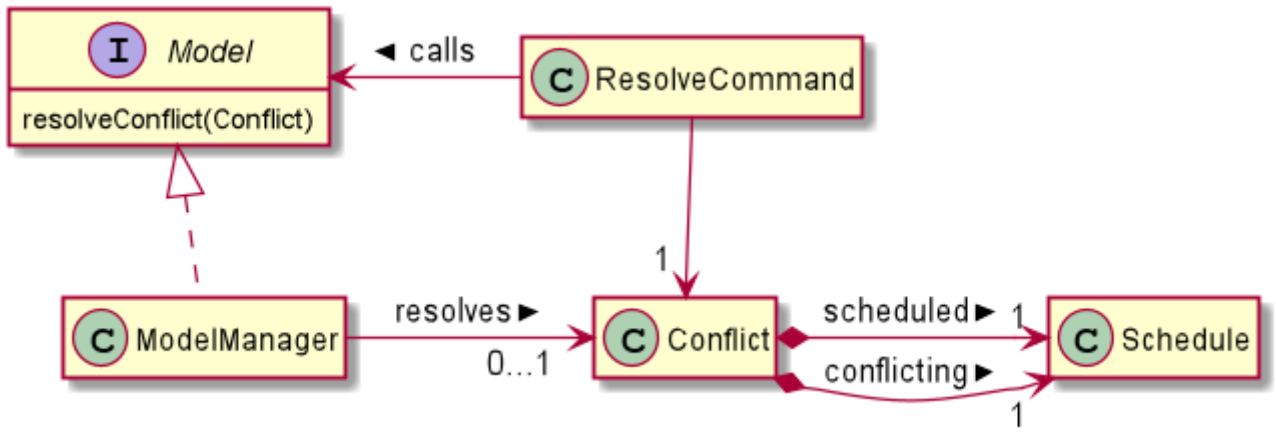


Figure 1. Class diagram for Resolve Command

With regards to the flow of the program for a scheduling conflict, the steps are laid out below:

Step 1. User enters a `schedule` command that will cause a scheduling conflict. The `ScheduleCommand` will change `MainApp` state to `State.IN_CONFLICT`.

NOTE	`schedule` can conflict with another `schedule` when the dates from the 2 schedules are the same. The method `model.hasSchedule()` returns `true` if that happens.

Step 2. A `CommandResult` object is returned to `MainWindow` where the flag `showResolve` is set to `true`.

Step 3. Upon receipt of the object, `MainWindow` will show the resolve window and the user is required to resolve the conflict.

NOTE	The `ResolveWindow` will block all inputs to `MainWindow` and only allow `resolve` command to be entered.

Shown below is the sequence diagram for when a scheduling conflict happens:

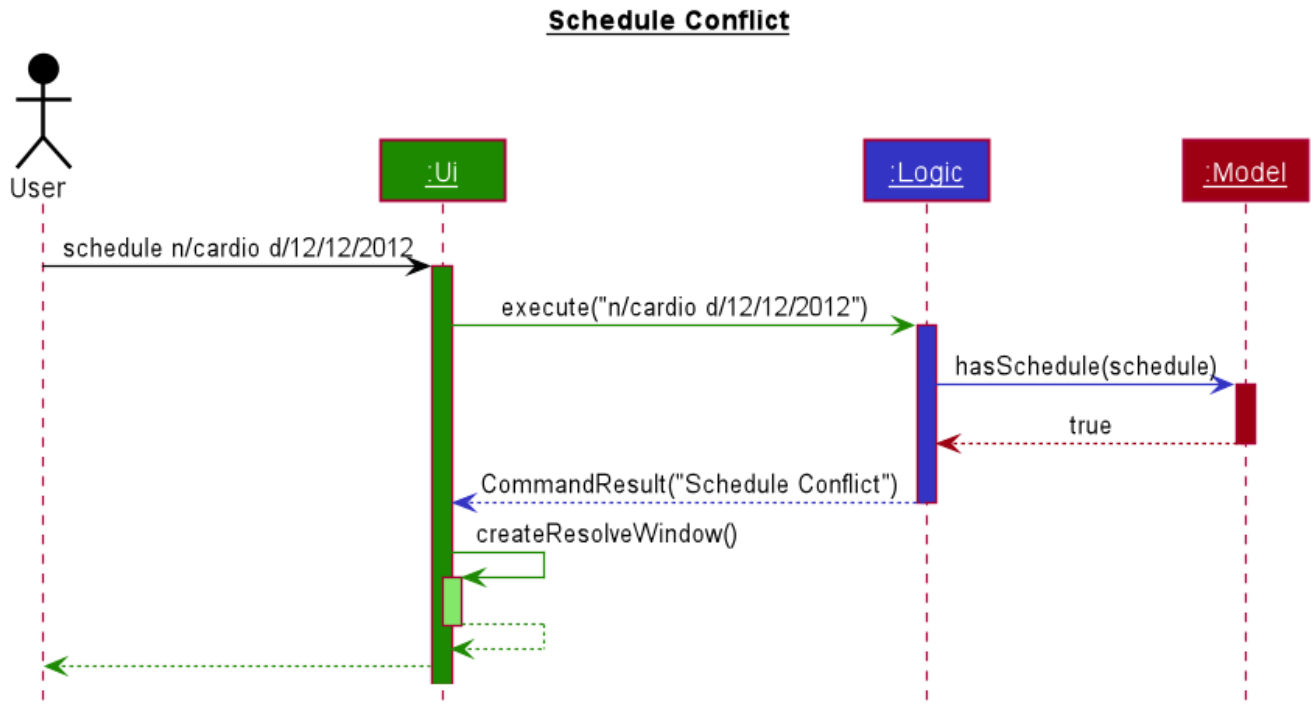


Figure 2. Sequence diagram when a scheduling conflict happens

Step 5. When the user is prompted with the `ResolveWindow`, all the conflicting exercises will be shown in one page. The previously `scheduled regime` on the left and the `conflicting regime` on the right.

Step 6. Once the user issue a `resolve` command correctly, the `model` and `storage` of ExerHealth will be updated to reflect the changes. A new regime will be added for the user from the `resolve`.

NOTE	The `ResolveWindow` will only take one valid `resolve` command and `Ui` will close the `ResolveWindow` immediately after the command finishes. The newly made schedule will result in a new `regime` being added to the user's `RegimeList`, so the name of the `regime` in the `resolve` command cannot have any conflicts with current names in `RegimeList`.

Step 7. The `ResolveWindow` then closes upon successful `resolve` and the application continues.

The following activity diagram summarizes what happens when a user enters a `schedule` command:

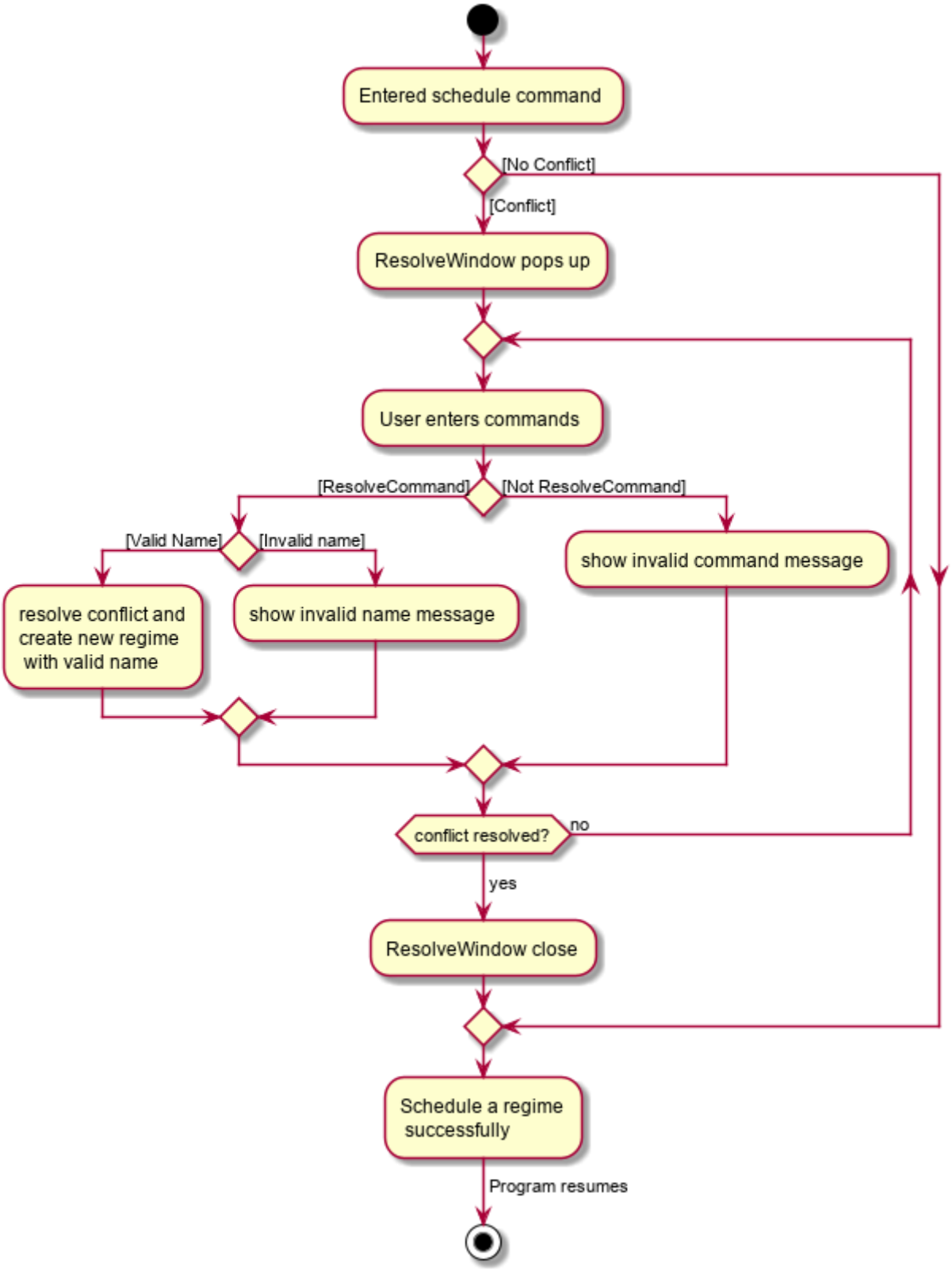


Figure 3. Activity diagram for when a user enters a `schedule` command

Design Considerations

Aspect: Signalling Schedule Conflict

- **Choice 1 (current choice):** Using `CommandResult` object
    - Pros:

        - Makes use of existing objects in codebase making it easier to implement
        - UI does not have to handle logic when encountering schedule conflicts. It only has to show the `ResolveWindow` and pass the data accordingly.
    - Cons:
        - If we have to signal different types of outcomes to the UI, the `CommandResult` class will become bloated.
- **Choice 2:** throw `ScheduleException`
    - Pros:
        - Easy to implement. `ScheduleCommand` just has to throw an exception and `UI` catches it.
    - Cons:
        - `UI`s execute methods will contain multiple `try/catch` which acts like a control flow mechanism which increases code smell.
        - If there is a need to pass down information from executed Commands, an exception is unable to convey any sort of complex information that the `UI` can act on. Thus, encapsulating information in an object will be more open to extension compared to throwing an exception.