Ye Chenchen - Project Portfolio

PROJECT: FitHelper

---

1. Overview

FitHelper is a desktop diet-and-exercise-recording application made for Users who want to keep fit. It is built based on **AddressBook** (Level 3), a desktop address book application used for teaching Software Engineering principles.

FitHelper enables Users to record their basic profile data, weight records, daily food intake and sports. Rather than just keeping the raw data, **FitHelper** also provided useful analysis and other customized services, such as calendar view and reminders.

The application is mainly written in **Java** and built by a considerable **19k Lines of Code**. The codebase is well-maintained by reasonable amount of tests, and the program is written in high-quality code. A detailed and comprehensive set of guides are also provided for both application developers and users.

2. Summary of contributions

2.1. Major Enhancement

2.1.1. Build FitHelper Entry in Model, Storage Part and Its Basic Commands (#143)

- **Content**
    - **Model Part**: Create Entry class (for both food and sport entries) and all of its attributes classes. Modify Model and Model Manager to build a basic skeleton of FitHelper.
    - **Storage Part**: Create Json-adaptive Entry and Json-adaptive FitHelper main storage. Modify Storage and Storage Manager to enable FitHelper store both type of entry (food and sports).
    - **Logic Part**: Update all command prefix and flags. Create basic commands and corresponding parsers related to Entry, including `add`, `edit`, `find`, `delete`.
- **Justification**
  This feature builds the basic skeleton of the whole application in Model and Storage Part. The creation of Entry allows user to record their daily food intake and sports.
- **Highlights**
  This feature allows other developers to build all other features related to Entry, such as Today Page, Calendar Page.

2.1.2. Build Profile in Model, Storage, UI Part and Its Commands

- **Content**
  - **Model Part**: Create Profile class and all of its attributes classes. A UserProfile class is built to wrap up user profile, which is parallel to the FitHelper class. Modify Model and Model Manager to include the profile system. (#170, #186)
  - **Storage Part**: Create Json-adaptive Profile and Json-adaptive UserProfile main storage. Modify Storage and Storage Manager to establish a separate new database for profile-related data. (#203, #248)
  - **UI Part**: Build Profile Page, which shows a table of user profile data. (#186, #203)
  - **Logic Part**: Enable Profile Page switch and enable user to update profile's attributes value by corresponding commands and command parsers, including `profile` and `update`. (#203, #214)
- **Justification**
  - This feature influences all Model, Storage, Logic and UI part in the application. It builds and manipulated a separate new database just for user profile.
  - Profile Page is an essential part of the application, as user's basic data can be used in other features and analysis to provide a more customized service.
- **Highlights**
  Originally, the application only contains a system dealing with Entry and Entry data. This feature creates a new parallel system, handling user's profile data from back end to front end.

2.1.3. Build Weight in Model, Storage, UI Part and Its Commands

- **Content**
  - **Model Part**: Create Weight class and all of its attributes classes. A WeightRecords class is built to wrap up all weight records, which is parallel to the FitHelper and UserProfile class. Modify Model and Model Manager to include the weight system. (#186, #240)
  - **Storage Part**: Create Json-adaptive Weight and Json-adaptive WeightRecords main storage. Modify Storage and Storage Manager to establish a separate new database for weight-related data. (#240)
  - **UI Part**: Build Weight Page, which shows a notification indicating the gap to user target data and two Trend Graphs for weight and data. (#186, #242)
  - **Logic Part**: Enable Weight Page switch and enable user to manipulate weight database by corresponding commands and command parsers, including `weight`, `addWeight`, `editWeight`, `deleteWeight` and `clearWeight`. (#240, #311)
- **Justification**
  - This feature influences all Model, Storage, Logic and UI part in the application. It builds and manipulated a separate new database just for user weight records.
  - Weight Page is one of the main parts in this application, as keep tracking of user's weight and provided useful analysis can significantly help the user to lose weight and keep fit.
- **Highlights**

    - With this feature, all three database for this database are built:
        - FitHelper for entry and diary data
        - UserProfile for profile data
        - WeightRecords for weight data
    - This feature is also strongly linked with profile. One command may change both profile and weight database and GUI page.

2.2. Code contribution

- **Code contributed**: [Functional code]

2.3. Other contributions

- Project management:
    - Create Milestone `v1.0`-`v2.0` on GitHub
    - Updated and closed issues and milestone regularly on GitHub
- Enhancements to existing features:
    - Updated the GUI color scheme (#214)
- Documentation:
    - Make each session in User Guide aligned in structure, and summarize all commands (#324)
- Community:
    - PRs reviewed (with non-trivial review comments) (#242)
    - Reported bugs and suggestions for other teams in the class (examples: #1, #4, #7)

3. 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.

Common Fields and Keywords

- **x**/ entry type
- **n**/ entry name
- **t**/ entry time
- **l**/ entry location
- **c**/ entry calorie
- **s**/ entry status
- **r**/ entry remark

- **i**/ entry index
- **d**/ date in format yyyy-MM-dd
- **m**/ mode
- **dc**/ diary content
- **dr**/ duration
- **attr**/ profile attribute name
- **v**/ profile attribute value/ weight value
- **by**/ sorting criterion
- **o**/ sorting order
- **k**/ keywords for searching/checking
- **-f** force change flag

Keep User Profile

Profile Page: `profile`

Profile page serves to be a summary for basic user data.
The profile information includes: Name, Age, Gender, Address, Height, Target Weight, Current Weight and Current BMI.

Format: `profile`

Update Profile Data : `update`

Update user data in the profile by attributes. Profile attributes include: Name, Age, Address, Gender, Height and Target Weight.
Every `update` command will lead to the profile page.

Format: `update [-f] attr/ATTRIBUTE v/VALUE`

- If no user profile data is provided by the user, FitHelper will initialize with the sample profile data.
- The updated attribute name is **not** case-sensitive and can include spaces, but the name must match some fields in user profile.
- e.g. Both `attr/target weight` and `attr/TARGETWEIGHT` are acceptable.
- Any updated value should follow its original data type.
- If the chosen updated attribute has already had original value, **flag** `-f` need to be used to enable **force overwrite**.

Examples:

- `update attr/height v/160`

- `update -f attr/name v/Alice Wang`

Keep Weight Records

Weight Page: `weight`

Weight page serves to be a summary for user’s weight and BMI changes according to time.
It shows user data in graph for easy understanding. By default, it will generate graph from all history data chronologically.

- **Gap notification**
  The top notification shows the comparison between user current weight and target weight.
  - If current weight is **larger** than target, the gap between the two will be highlighted.
  - If current weight is **the same or less** than the target, a succeed notification will be generated.
- **Trend Graph - Weight**
  Display a trend graph of user’s weight according to time.
- **Trend Graph - BMI**
  Display a trend graph of user’s BMI according to time. The BMI value is calculated by weight and height value at that date.

Format: `weight`

Add Weight Records : `addWeight`

Add a new weight record into the weight records database. A weight record is related to date and weight value, and a auto-computed BMI value will be stored as well.
Every `addWeight` command will lead to the weight page. If a new weight record is added successfully, two new points will be added into the two trend graphs separately.

Format: `addWeight v/WEIGHT_VALUE [d/DATE]`

- If no weight record exists in the database, "Not Available Now" will be shown in Profile Page’s Current Weight and Current BMI fields.
- The date of a new weight record can be **omitted** when user inputs the `addWeight` command. By default, it will refer to the date of today.
- The date should be in format of `yyyy-MM-dd`, and should **not after the date of today**.
- **No two weight records should have the same date.** If adding a new weight record with the same date as an existing weight, a warning will be generated, and thus will fail to add.

Examples:

- `addWeight v/50.0 d/2020-02-01`
- `addWeight v/52.30`

Edit Weight Records : `editWeight`

Edit an existing weight record in the weight records database. A weight record is identified by its **unique date**, and user can find the date on the x-axis of the Weight Trend Graph.
User are able to edit the weight value, and corresponding BMI value will be auto-computed using the new weight value and user's **current height**.
Every `editWeight` command will lead to the weight page. If a weight record is edited successfully, two new points will change their positions on the two trend graphs separately.

Format: `editWeight [d/DATE] v/NEW_WEIGHT_VALUE`

- The date should be in format of `yyyy-MM-dd`. If no existing weight record is on the input date, the input date is considered as invalid, and thus a warning will be thrown.
- If the date is **omitted** when user inputs the `editWeight` command, by default, it will refer to the date of today.
- If the edited weight record is the **latest weight record** in the database, an update in Profile Page's Current Weight and Current BMI fields can be found as well.
- If the new weight value is the same as original weight value in the weight records, an exception will be thrown.

Examples:

- `editWeight d/2020-02-01 v/51.0`
- `editWeight v/52.40`

Delete Weight Records : `deleteWeight`

Delete an existing weight record in the weight records database. Same as `editWeight` command, a weight record is identified by its **unique date**, and user can find the date on the x-axis of the Weight Trend Graph.
Every `deleteWeight` command will lead to the weight page. If a weight record is deleted successfully, two corresponding points will be removed from the two trend graphs separately.

Format: `deleteWeight [d/DATE]`

- The date should be in format of `yyyy-MM-dd`. If no existing weight record is on the input date, the input date is considered as invalid, and thus a warning will be thrown.
- If the date is **omitted** when user inputs the `deleteWeight` command, by default, it will refer to the date of today.
- If the deleted weight record is the **latest weight record** in the database, the second latest weight record will be used to update Profile Page's Current Weight and Current BMI fields.

Examples:

- `deleteWeight d/2020-02-01`
- `deleteWeight`

Clear Weight Records : `clearWeight`

Clear all weight records in the weight records database.
Weight Page’s graphs will be empty, and Profile Page’s Current Weight and Current BMI fields will be `Not Available Now` after clear all weight records.

Format: `clearWeight`

Exit the program : `exit`/`bye`/`quit`

Exits the program.

Format: `exit` or `bye` or `quit`

Save the data

fitness log book data are saved in the hard disk automatically after any command that changes the data.
There is no need to save manually.

> Three local database in Json format will exist after running FitHelper:
>
> - **fithelper.json** : data related to entries and diaries.
> - **userprofile.json** : data related to user profile.
> - **weightrecords.json** : data related to weight records.

Command Summary

- **Help - switch to Help Page** : `help`
- **Entry - switch to DashBoard** : `home`
- **Entry - add an entry** `add n/NAME t/TIME l/LOCATION c/CALORIE [r/remark]…`
- **Entry - list all entries** : `list`
- **Entry - view reminders** : `reminder`
- **Entry - edit an entry** : `edit INDEX [n/NAME] [t/TIME] [l/LOCATION] [c/CALORIE] [r/remark]…`
- **Entry - find by name** : `find KEYWORD [MORE_KEYWORDS]`
- **Entry - delete an entry** : `delete INDEX`
- **Entry - sort entry list** : `sort [x/TYPE] by/SORT_BY [o/ORDER]`
- **Entry - check calorie reference** : `check x/TYPE k/ONE_OR_MORE_KEYWORDS`

- **Diary - switch to Diary Page** : `diary`
- **Diary - add a diary** : `addDiary d/DATE dc/DIARYCONTENT`
- **Diary - edit a diary** : `editDiary d/DATE dc/DIARYCONTENT`
- **Diary - delete a diary** : `deleteDiary d/DATE`
- **Diary - find a diary** : `findDiary [d/DATE] [dc/DIARYCONTENT]`
- **Diary - clear all diaries** : `clearDiary`

- **Clear - clear all entries and diaries** : `clear`
- **Undo - undo commands related to entries and diaries** : `undo`
- **Redo - redo commands related to entries and diaries** : `redo`

- **Calendar - switch to Calendar Page** : `calendar`
- **Calendar - display from a referenced date** : `calendar d/DATE`
- **Calendar - change to list mode** : `calendar m/ls [d/DATE]`
- **Calendar - change to timetable mode** : `calendnar m/tb [d/DATE]`
- **Calendar - display entries from a particular date** : `calendar sh/DATE`

- **Today - switch to Today Page** : `today`

- **Profile - switch to Profile Page**: `profile`
- **Profile - update profile data**: `update [-f] attr/ATTRIBUTE V/VALUE`

- **Weight - switch to Weight Page**: `weight`
- **Weight - add a weight record** `addWeight v/VALUE [d/DATE]`
- **Weight - edit a weight record** `editWeight [d/DATE] v/VALUE`
- **Weight - delete a weight record** `deleteWeight [d/DATE]`
- **Weight - clear all weight records** `clearWeight`

- **Exit the Program**: `exit` or `bye` or `quit`

4. 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.

Storage component

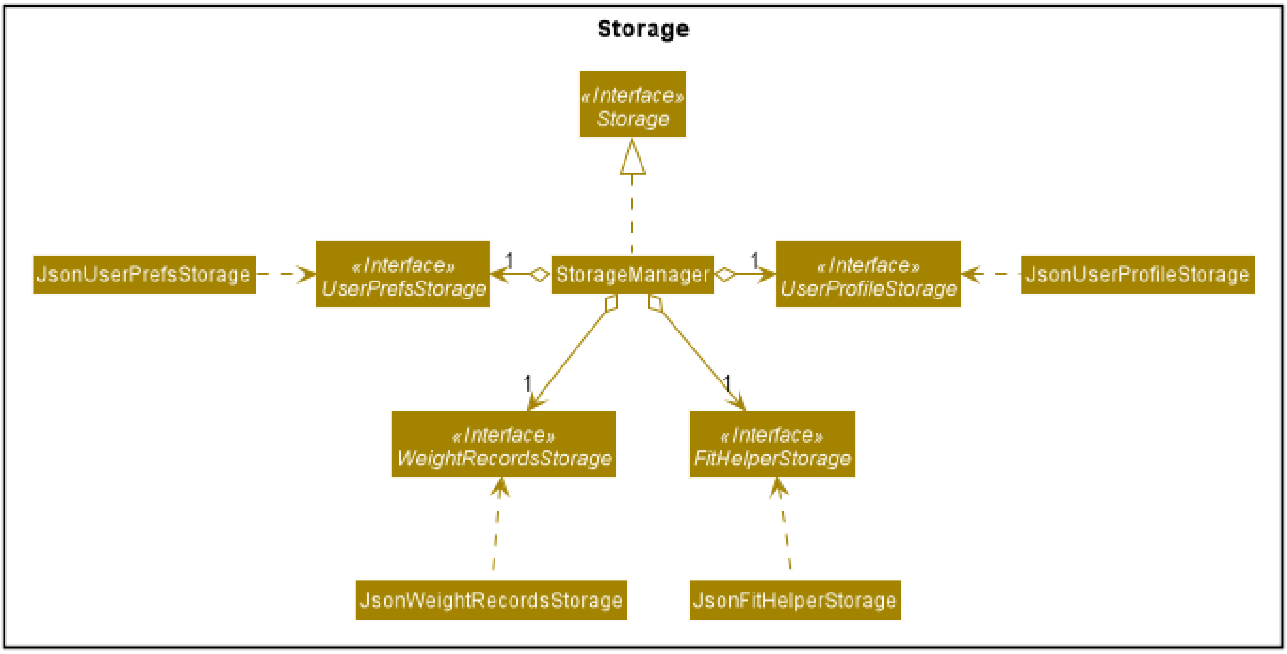


Figure 1. Structure of the Storage Component

API : `Storage.java`

The `Storage` component,

- saves a `UserPrefsStorage` object in json format and can read it back.
- saves a `FitHelperStorage` object in json format (***fithelper.json***) and can read it back. This database includes all data related to entries and diaries.
    - `FitHelperStorage` consists of lists of `Entry` and `Diary`, and thus these two types of objects can be saved in json format and read back too.
- saves a `UserProfileStorage` object in json format (***userprofile.json***) and can read it back. This database includes all data related to user profile attributes.
    - `UserProfileStorage` consists of a list of `Profile`, and objects in type of `Profile` can be saved in json format and read back.
- saves a `WeightRecordsStorage` object in json format (***weightrecords.json***) and can read it back. This database includes all data related to weight records.
    - `WeightRecordsStorage` consists of a list of `Weight`, and objects in type of `Weight` can be saved in json format and read back.

Add Weight Records

FitHelper allows the user to track with their weight change easily by allowing user to add their current weight and previous weight.

Sample

An example usage scenario and how the `addWeight` command behaves at each step is shown below.

Step 1.

- The user launches the application for the first time.
- `UniqueWeightList` in Model contains no default weights before the user adds any.
- `weightrecords.json` in local Storage contains no weight records as well.

Step 2.

- The user inputs `addWeight` command word, followed by `v/WEIGHT_VALUE` and an optional `d/DATE`.
- `UI` passes the input to `Logic`.
- `Logic` then uses a few `Parser` classes to extract layers of information out as seen from steps 3 to 5.

Step 3.

- `Logic` passes the user input to `FitHelperParser`.
- `FitHelperParser` identifies that this is a `AddWeightCommand` through the command word "addWeight".
- It then creates a `AddWeightCommandParser` to parse the input into a `AddWeightCommand` and return back.

Step 4.

- `Logic` gets the `AddWeightCommand` and execute it.
- The execution firstly check is the new weight date is after today's date and if there is already a existing weight in the UniqueList.
- Both of these two cases will throw corresponding `CommandException`.
- Then the execution add the new `Weight` into model.
- Finally, it returns a `CommandResult` to `UI`, containing the response to the user and the displayPage, which equals to `WEIGHT` page.

Step 5.

- `UI` displays the response in the `CommandResult`.
- In addition, UI will change to display Weight Page after updating Profile Page and Weight Page.

Implementation

Storage

A weight is stored with three attributes in the `weightrecords.json` database:

- `date` : the date of the weight record in format of `yyyy-MM-dd`, if no date is provided by the user, the **default value** is the date of today
- `weightValue` : a double value with two decimal places.

- `bmi` : the BMI value is also a double value with two decimal places. It is auto-computed and stored, using the formula : `BMI = Weight Value(kg) / Height(m)^2`. The Height value gets from user profile in `userprofile.json` database.

Model

- A single weight is represented as model `Weight` with the attributes of `Date`, `WeightValue`, and `Bmi`.
- In `ModelManager`, all weights are represented by `WeightRecords weightRecords`.
    - The `WeightRecords` class implements `ReadOnlyWeightRecords` interface, and therefore can return an **unmodifiable** version of a **unique** list of weights.
    - The `WeightRecords` wraps a `UniqueWeightList` which allows adding and iterating. **Unique** here refers to the constraint that no two weight with the same date can exist in the list/database.
- In `ModelManager`, a `FilteredList<Weight> filteredWeight` object is used to store and update a filtered version of all weights.
    - The FilteredList wraps a `ObservableList` and filters using a provided `Predicate`.

UI

When user input `addWeight` command to `UI`, the input is passed to `Logic` part as a `String`.

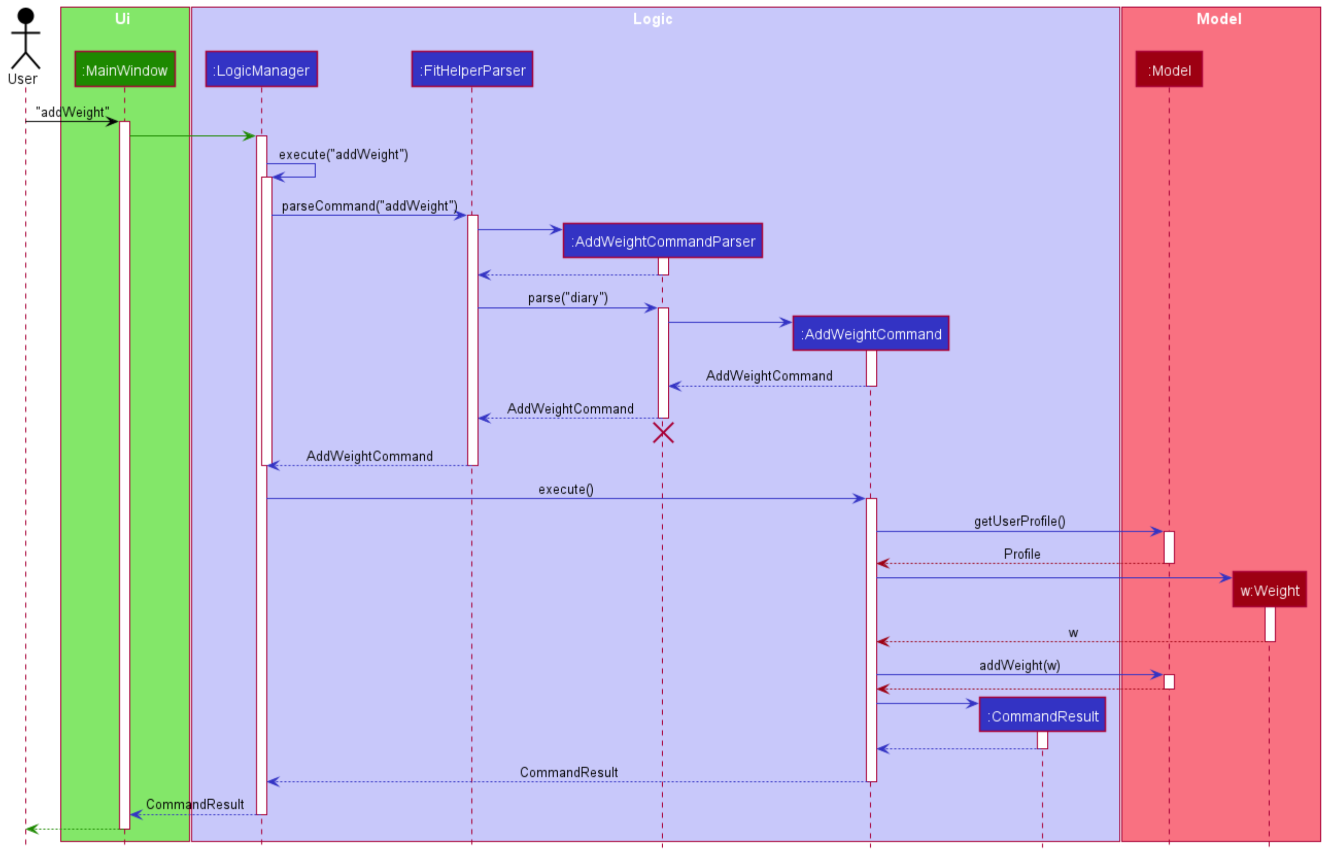
After `addWeight` command is executed, a `CommandResult` with `DisplayPage` equals `WEIGHT` will be passed back to `MainWindow` in `UI` part. Then:

- Firstly, it will call `updateProfilePage()`, since if the newly added weight has the lasted date, Current Weight and Current BMI in uer profile will need to be updated.
- Secondly, it will call `updateWeightPage()`, since if a new weight is added successfully, new points should be added on to Weight Line Chart and BMI Line Chart. The text content of top notification will also be updated if the gap between Current Weight and Target Weight is changed.
- Lastly, it will call `showWeightPage()`. This allows the Main Window auto-switch to Weight Page after each `addWeight` command by user.

Logic

The Sequence Diagram below shows how the components interact with each other for the mentioned scenario in sample.

Sequence Diagram for Add Weight Feature



Appendix A: Product Scope

Target user profile:

- has a need to control weight, therefore need to record daily food intake and sports
- prefer desktop apps over other types
- can type fast
- prefers typing over mouse input
- is reasonably comfortable using CLI apps

Value proposition: achieve fitness control faster than a typical mouse/GUI driven app

Appendix B: Non Functional Requirements

1. Should work on any mainstream OS as long as it has Java 11 or above installed.
2. Should be able to hold up to 1000 entries without a noticeable sluggishness in performance for typical usage
3. Should be able to function normally without internet access.
4. A user with above average typing speed for regular English text (i.e. not code, not system admin commands) should be able to accomplish most of the tasks faster using commands than using the mouse.
5. A user can get response from the system within 5 seconds after command input.

6. A user can be familiar with the system commands and interface within half an hour usage.

{More to be added}

Appendix C: Glossary

Mainstream OS

Windows, Linux, Unix, OS-X

Table 1. Command Prefix

Prefix	Meaning	Used in the following Command(s)
x/	Type of entry	add, check, delete, edit, find
i/	Index of entry	edit, delete, edit
n/	Name	add, edit
t/	Time in format of "date hour minute"	add, edit
l/	Location	add, edit
c/	Calorie	add, edit
s/	Status	add, edit
r/	Remark	edit
d/	Date in format of **yyyy-MM-dd**	calendar, addWeight
dr/	Duration in format of **yyyy-MM-dd yyyy-MM-dd**	add, edit
dc/	Dairy contents	dairy
k/	Keyword	check, find
attr/	Attribute in user profile	update
v/	Attribute Value in user profile	update, addWeight

Table 2. Possible Command Flags

Command	Flag	Meaning
Sort	-a	Sort in **ascending** order
Sort	-d	Sort in **descending** order
Sort	-t	Sort according to **time**
Sort	-c	Sort according to **calorie intake**
Update	-f	**Force update** even with existing value

Appendix D: Instructions for Manual Testing

Given below are instructions to test the app manually.

NOTE These instructions only provide a starting point for testers to work on; testers are expected to do more *exploratory* testing.

Launch and Shutdown

1. Initial launch
   a. Download the jar file and copy into an empty folder
   b. Double-click the jar file
      Expected: Shows the welcome page of FitHelper. On the left hand side, the user can see a list of page name. Users are able to click on the button or using corresponding command to direct to that page.
   c. The window size is fixed.
2. Shutdown
   a. Users are able to shutdown the application using CLI with following commands:
      - `exit`
      - `quit`
      - `bye`
   b. Users can also choose to shutdown the application by clicking on X button on the right top side if the window.
   c. User data will be auto-saved if user choose to shutdown the application. Three local data file in json format can be find:
      - `fithelper.json` : containg data related to entries and diaries.
      - `userprofile.json` : containing data related to user profile.
      - `weightrecords.json` : containing data related to all weight records.

Adding A New Weight Record

1. Add **first weight record** while there is no previous weight record in the database.
   a. Prerequisites: None. Users are able to use `addWeight` command at any page.
   b. Test case: `addWeight v/50.0`
      Expected:
      - A new `Weight` is added into `weightrecords` database, with `WeightValue` equals 50.0, `Date` with default value(today's date) and `BMI` calculated by `Height`.
      - The window is automatically directed to weight page. A new point is shown on both Weight Line Graph and BMI Line Graph. The top notification is also updated.

        - In profile page, Current Weight and Current BMI change from "Not Available Now" to the newest value.

    c. Test case: `addWeight v/49.0 d/2050-01-01`
       Expected: No new weight record is added since the date is after current date. An error message is shown in the command result box.

2. Add new weight record when there is already **some previous weight records existing** in the database.

    a. Prerequisites: None. Users are able to use `addWeight` command at any page.

    b. Test case: `addWeight v/48.0`
       Expected: No new weight record is added since there is existing weight record with the same date (by default is today’s date) in the data base. An error message is shown in the command result box.

    c. Test case : `addWeight v/47.0 d/2020-03-01`
       Expected:

        - A new `Weight` is added into `weightrecords` database, with `WeightValue` equals 47.0, `Date` with 2020-03-01 and `BMI` calculated by `Height`.
        - The window is automatically directed to weight page. A new point is shown on both Weight Line Graph and BMI Line Graph, and form a new trend line with previous data points. The top notification is also updated.
        - In profile page, Current Weight and Current BMI remain the same, since the newly added weight record is not the most recent record in the database.

Saving data

1. Dealing with missing/corrupted data files

    a. If the application is launched and shut down at least once, there will be three local database in json format.

    b. Delete `fithelper.json`, and launch FitHelper again. All user manipulation on entries and diaries will be cleared. `Dashboard`, `Today`, `Calendar` and `Diary` Page will restart with sample data.

    c. Delete `userprofile.json`, and launch FitHelper again. All user manipulation on user profile will be clear. `Profile` page will restart with sample user data.

    d. Delete `weightrecords.json`, and launch FitHelper again. All user manipulation on weight records will be clear. `Profile` page will show Current Weight and Current BMI as "Not Available Now", and `Weight` Page will have no data point on the trend line graph.