Liao Meng - Project Portfolio

PROJECT: FitHelper

Overview

FitHelper is a desktop diet-and-exercise-recording application. It is built based on AddressBook (Level 3), a desktop address book application used for teaching Software Engineering principles. The user interacts with it using a CLI, and it has a GUI created with JavaFX. It is written in Java, and has about 21 kLoC.

Summary of contributions

- **Major enhancement 1**: added the ability to **search for calorie data of common food/sports** #216 #249
  - What it does: allows the user to search for calorie intake/consumption of food/sports by keywords.
  - Justification: This feature makes it easier for the user to estimate how much calorie he/she will gain/burn by eating/exercising, so that he/she can create food/sports entry without the need to search online for relevant information.
  - Highlights: The application will perform up to 3 rounds of search, each round with different criteria (from most strict matching to less strict matching), to find matching data. This ensures that more relevant data are found and added first, and at the same time loosely-matching data can be shown when strictly-matching data are not found.
  - Credits: The calorie data are collected from the following websites: https://www.calories.info/ and https://www.nutristrategy.com/caloriesburned.htm
- **Major enhancement 2**: added the ability to **sort entry and reminder list** #261 #306
  - What it does: allows the application to sort and display the entries in food and/or sports entry list based on time/name/calorie value in ascending or descending order. The user can specify sort order and criterion. Note that the entries in the reminders list will also be sorted in the same manner.
  - Justification: This feature allows the user to view entries in a particular order. It also helps the user to view entry that he/she may deem more important first (for example entry with a higher calorie value, as it has a larger impact on the user's fitness), which facilitates editing.
  - Highlights: the user can specify which list to sort and in what order.
- **Minor enhancement**: added and updated the Help Page, which shows the list of valid user commands and their usage, and the link to the user guide #181 #338.
- **Code contributed**: [Functional and Test code]
- **Other contributions**:
  - Project management:

        - Managed release `v1.2.1` on GitHub
    - Team-Based tasks:
        - Frequently performed manual testing of the application, and gave feedback about bugs/feature flaws to team members promptly.
        - Review and comment team members' PR #91.
    - Testing:
        - Added test cases #323 #332.
    - Documentation:
        - Contributed to the User Guide and Developer Guide #224 #262 #344 #355 #356
        - Reviewed and edited sections of the User Guide and Developer Guide according to the change in application design and functionality #338 #339 #342 #349.
    - Community:
        - Contributed to forum discussion #118.
        - Reported bugs and flaws in other team's product PED.

Contributions to the User Guide

Given below are sections I contributed to the User Guide.

Sort food/sports entry list: `sort`

Sorts food or sports entry list, or both of them, based on starting time or calorie value or name of the entry (case insensitive), in either ascending or descending order. In addition, the reminders entry list will also be sorted in the same manner specified by the user. The user must specify the sorting criterion `cal/c OR time/t OR name/n`. If the user does not specify the type of entry list, both lists will be sorted. And if the user does not specify the sorting order `a OR d`, the default order is descending (i.e. entry that starts later or entry with higher calorie value or entry whose name starts with later alphabet comes first).

format: `sort [x/TYPE] by/SORT_BY [o/ORDER]`

Examples:

- `sort x/f by/time o/a` (sort food entry list and reminders list in ascending order of recording time, i.e. the older entry comes first)
- `sort by/c` (sort both food and sports entry list, as well as reminders list, in descending order of calorie value, i.e. entry with higher calorie intake/consumption comes first)

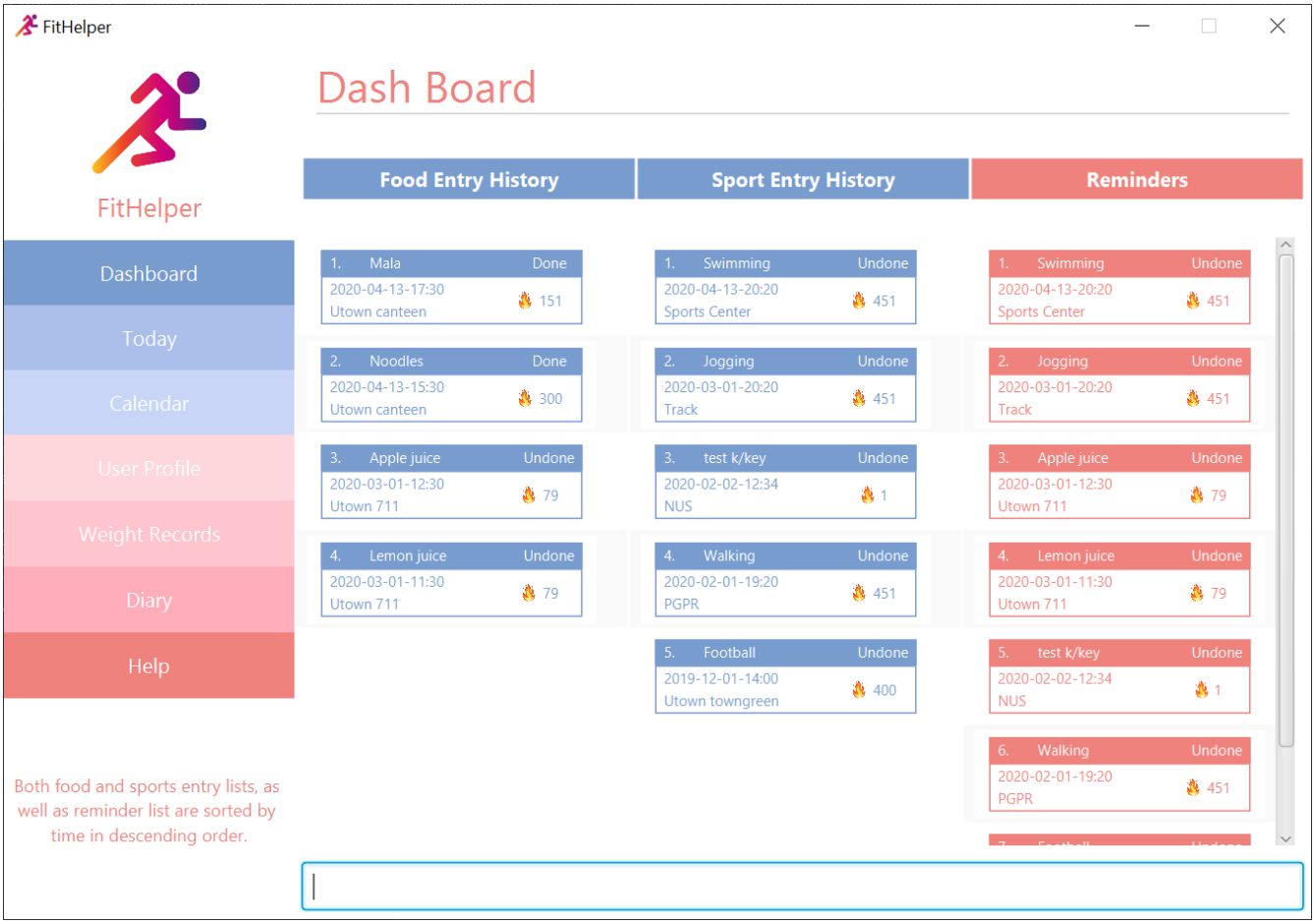


Figure 1. Dashboard page after all 3 lists are sorted by time in descending order (later entry comes at top)

Check calorie intake/consumption of common food/sports: `check`

Searches through the pre-set database for calorie intake/consumption information about common food/sports (i.e. a datum) whose name matches/contains keywords specified by the user. Note that due to space constraint, at most 3 matching data will be shown at the left-bottom corner of the application. Each datum shows the name of the food/sports and its calorie intake/consumption information: for food, it will be in cal per serving (weight per serving in gram will also be shown); for sports, it will be in cal per hour for a 70kg person. Please note after the command is executed, the application will automatically go to Dashboard page **without resetting the display status**.

The searching works as follows:

- Retrieves food/sports data based on check type specified by the user.
- Checks and adds food/sports datum whose name **equals one of the keywords**. If fewer than 3 food/sports data are added so far, checks and adds food/sports datum whose name **contains the keywords as a whole**. If still fewer than 3 food/sports data are added so far, checks and adds food/sports datum whose name **contains one of the keywords**.
- If no datum is added after all the steps, replies user that search failed. Otherwise, shows user the data.

format: `check x/TYPE k/ONE_OR_MORE_KEYWORDS`

Examples:

- `check x/food k/apple` (searches for calorie intake of food whose name match/contain keyword "apple")

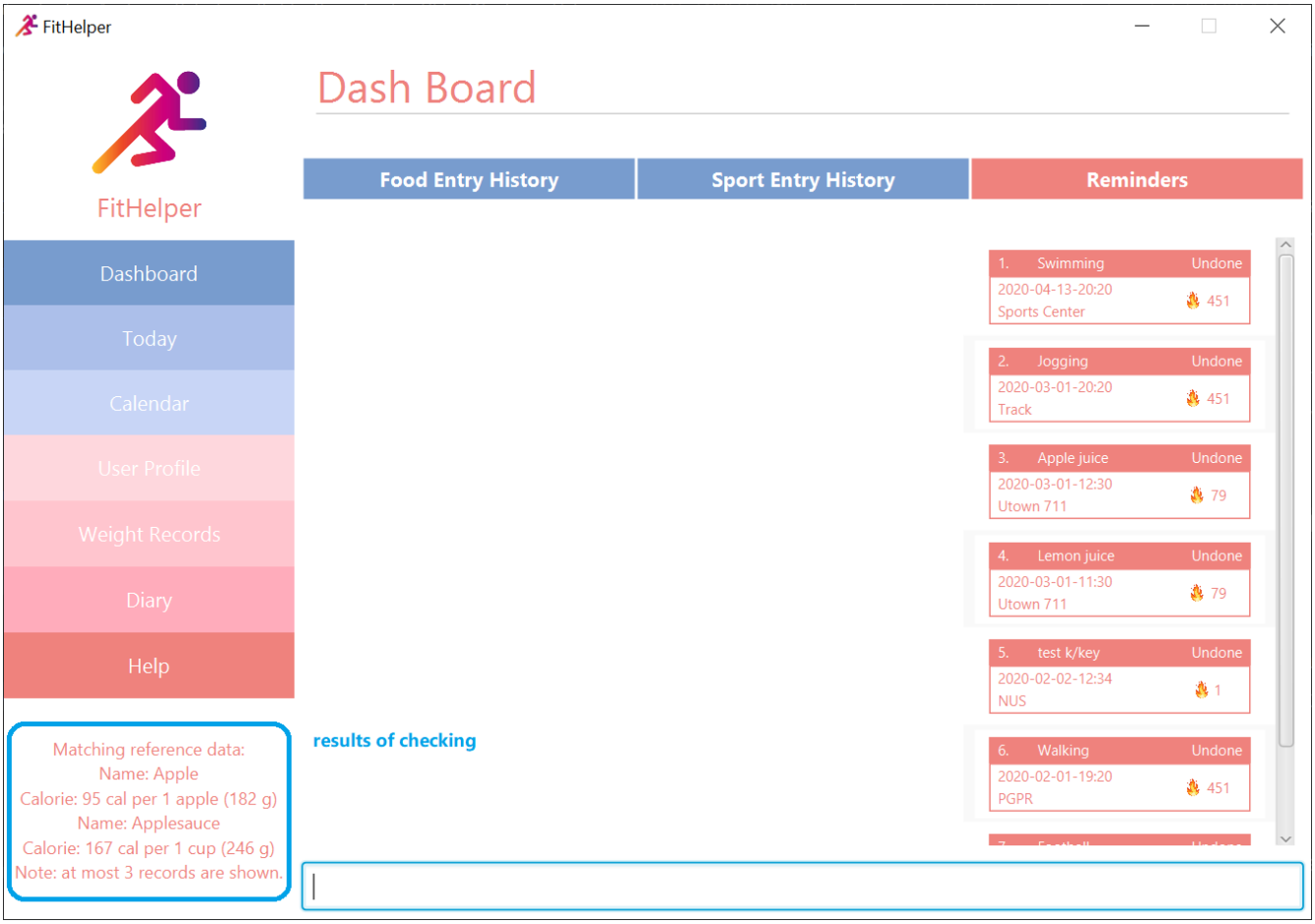


Figure 2. Check command result (matching data found)

Given below are sections I reviewed and edited in the User Guide.

General rules about command format

- A prefix followed by a word in `UPPER_CASE` form a parameter to be supplied by the user.

Example: in `add n/NAME`, `n/NAME` is a parameter which can be `n/running`.

- Parameters in square brackets are optional e.g `n/NAME [r/REMARK]` means both `n/swimming r/fun` and `n/swimming` are both valid user inputs.
- Parameters not in square brackets are compulsory e.g. `x/TYPE k/KEYWORDS` means you must input both `TYPE` and `KEYWORDS` parameters.
- Items with `⋯` after them can be used multiple times including zero times e.g. `[r/remark]⋯` can be used as (i.e. 0 times), `r/really fun`, `r/really fun r/helps me lose weight` etc.
- Parameters can be in any order e.g. if a command format is `add n/NAME t/TIME`, then `add n/Bob t/2020-04-10-08:00` and `add t/2020-04-10-18:00 n/Bob` are both acceptable.
- The field `TYPE` in this document refers to type of entries. The only valid values are food OR sports, or in acronym f OR s.
- The field `TIME` needs to be in the format of yyyyy-mm-dd-hh:MM, e.g. `2020-04-10-03:00`
- The field `DATE` should be entered in the format of `yyyy-mm-dd`, e.g. `2020-02-02`.
- Our time parser will auto-correct some invalid date/time entered. More specifically, as long as the month value is valid and the day value is between 1 and 31, the parser will auto-correct an invalid date to the last valid date of the year-month specified by the user.

Example: `2020-02-31-12:34` will be auto-corrected as `2020-02-29-12:34` since `2020-02-29` is the last valid date of Feb 2020. `2020-22-31` and `2020-02-40` will not be auto-corrected since the month value is invalid or the day value is beyond 31.

- The field `INDEX` should be a positive integer representing a one-based index of an entry in an list.
- The field `DURATION` should be a positive double number, representing duration in hours. e.g. `dr/1.5` for 1.5 hours.
- You may input multiple parameters of a same prefix, but only the last input value will be read.

Example: `add n/Alice n/Bob n/Charlie` is equivalent to `add c/Charlie`.

- Please do not input parameters not required by a particular command.

Example: the format of sort command is `sort [x/TYPE] by/SORT_BY [o/ORDER]`, so you should not input other parameters like `n/NAME`, `k/KEYWORDS` or `v/VALUE`.

- The flag `-f` should be added **just after the command word**, e.g. `update -f`.

Contributions to the Developer Guide

Given below are sections I contributed to the Developer Guide.

Check calorie intake/consumption of some common food/sports

This function allows the user to search for calorie intake/consumption of food/sports by keywords. It helps the user better estimate how much calorie he/she will gain/burn by eating/exercising, so that he/she can add food/sports entry into the application without the need to search online for relevant information.

Implementation

The check function is achieved by calling the `FitHelper` inside the `ModelManager` to search through either `FoodCalorieTable` or `SportsCalorieTable` for `CalorieDatum` that contain the keywords specified by the user.

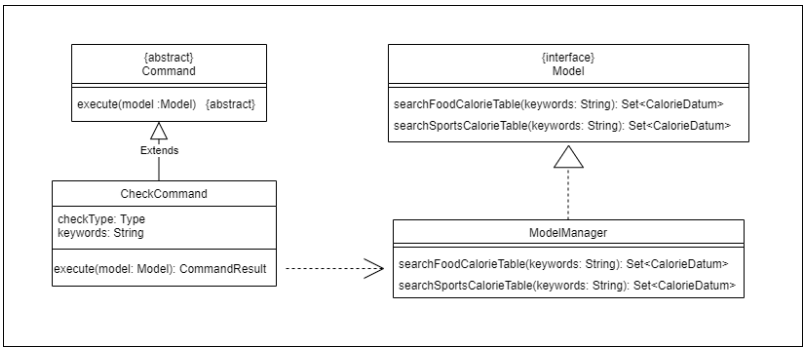
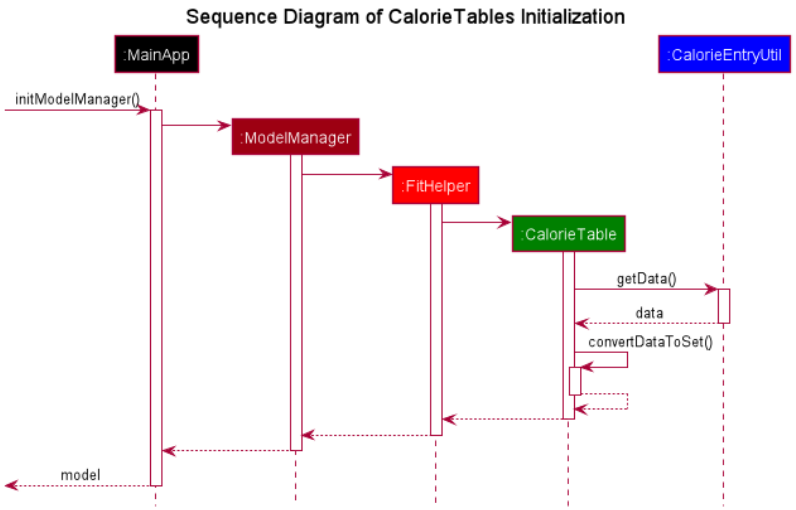


Figure 3. Class diagram showing the interaction between CheckCommand, a concrete subclass of Command, and ModelManager, a concrete class implementing Model interface.

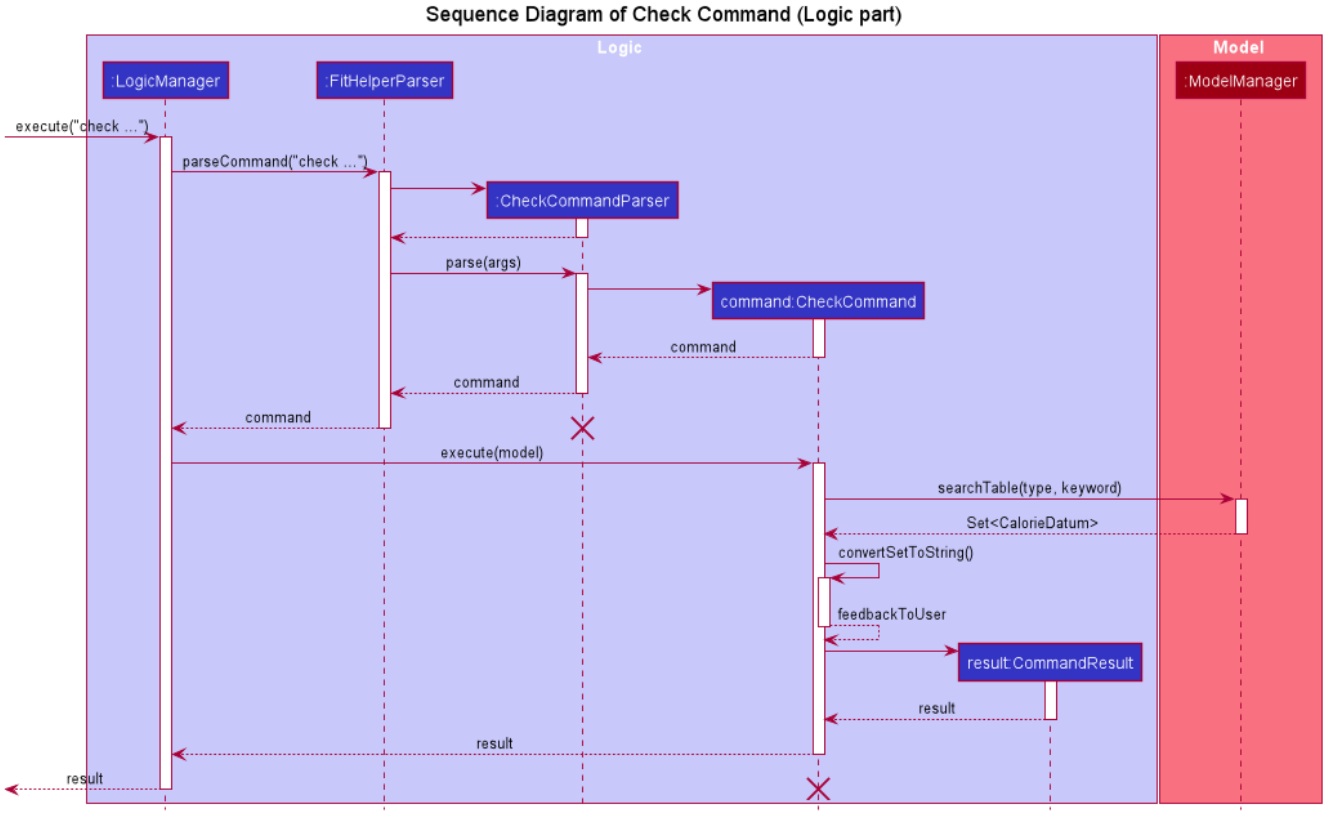
Given below are example usage scenario:

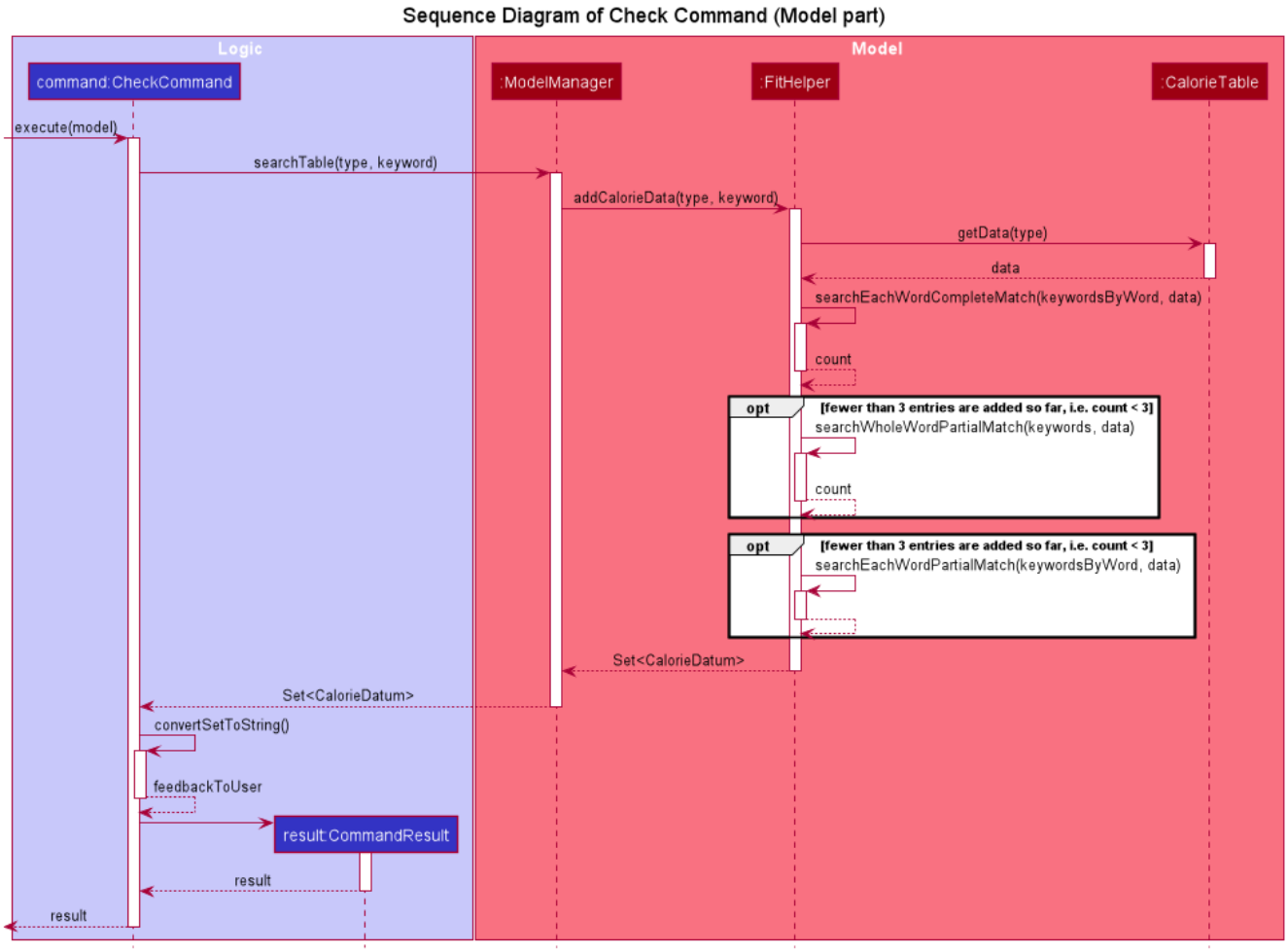
Initialization: when the application is launched, `ModelManager` will initialize a `FitHelper`, which will in turn initialize both `FoodCalorieTable` and `SportsCalorieTable` to contain pre-set data which is a LinkedHashSet of one type of `CalorieDatum` (either `FoodCalorieDatum` or `SportsCalorieDatum`).



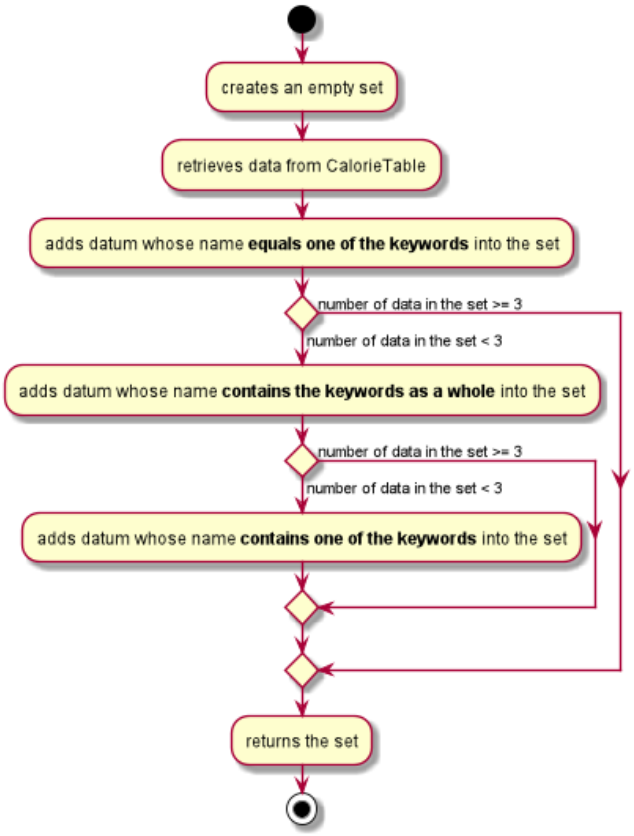
Case 1: when the user enter the command `check x/sports k/swim`, the `LogicManager` will create a `CheckCommand`, which asks `ModelManager` to let `FitHelper` to search through `SportsCalorieTable` to add first 3 `CalorieDatum` whose name matches the keyword `swim` into a set, and return the set to `CheckCommand`. Since the set contains at least one `CalorieDatum` (meaning there is some matching data), the `CheckCommand` returns a `CommandResult` whose `feedbackToUser` contains a success message followed by the string representation of each matching datum.

Case 2: when the user enter the command `check x/f k/swim`, the `LogicManager` will create a `CheckCommand`, which asks `ModelManager` to let ` FitHelper` to search through `FoodCalorieTable` to add first 3 `CalorieDatum`s whose name contains the keyword `swim` into a set, and return the set to `CheckCommand`. Since the set contains no `CalorieDatum` (meaning there is no matching data), the `CheckCommand` returns a `CommandResult` whose `feedbackToUser` contains a failure message followed by the string representation of the keyword.





The detailed searching mechanism is illustrated in the following activity diagram:



Design Considerations

Aspect: Data structure to store entries

- **Alternative 1 (current choice):** Use an `LinkedHashSet` as a field in CalorieTable to store the entries.
  - Pros: Easy to implement partial-key search (compare the keyword with the name of each entry in the set). Ensure that the database contains no duplicate data since a Set does allow duplicate elements.
  - Cons: O(n) complexity for finding matching entries, where n is the number of entries in the set.
- **Alternative 2:** Use a `HashMap` as a field in CalorieTable to store the entries. The key is the name of the entry and the value is the entry.
  - Pros: (theoretically) O(1) time complexity for finding an entry given a complete keyword, regardless of how many entries are in the HashMap.
  - Cons: hard to implement partial-key search (i.e. the keyword is only part of the name of the entry).

Store calorie data defined by the user [proposed in v2.0]

This proposed function allows the user to create his/her own calorie data and store them into the application database. These data can be later searched by and shown to the user via the existing `check` command. This function helps the user to record calorie data of food/sports not included in the pre-set database (or data more accurate than the ones stored in the pre-set database), so that he/she do not need to remember these data or search them online when he/she need to add same food/sports entry into the application in the future.

proposed implementation

To store user-defined calorie data internally, we need current FitHelper class in the Model component to associate 2 new classes `UserFoodCalorieTable` and `UserSportsCalorieTable`, which will store user-defined calorie intake of food and calorie consumption of sports respectively.

To allow the application to access these user-defined data in the future, we need another class in the Storage component responsible for writing internally data into Json file and reading the Json file back to internally stored data when the user reopens the application.

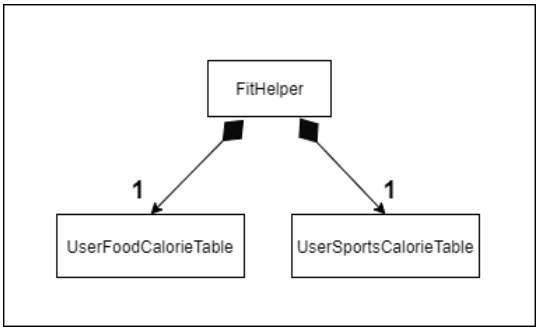


Figure 4. Additional classes associated with FitHelper for storing user-defined data (existing classes associated with FitHelper are omitted)

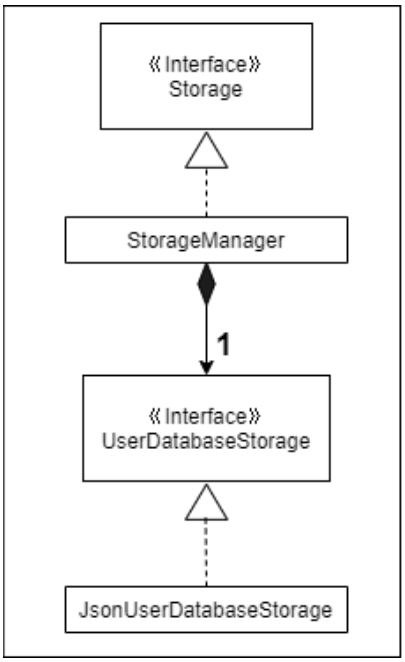


Figure 5. Additional Storage Class for writing and reading user-defined data (existing storage classes are omitted)

To allow the `check` command to search for user-defined data, the command will be modified to retrieve data from `UserFoodCalorieTable` or `UserSportsCalorieTable` as well, depending on the search type specified by the user.

Given below are sections I reviewed and edited in the Developer Guide.

Appendix A: User Stories

Priorities: High (must have) - * * *, Medium (nice to have) - * *, Low (unlikely to have) - *

Priority	As a …	I want to …	So that I can…
* * *	new user	record my basic information such as name and gender	have a more complete profile
* * *	user who is concerned about body shape	record and update my current height and weight	have a clear view of my current body condition

Priority	As a …	I want to …	So that I can…
* * *	user who wants to lose weight	set my target weight	have a clear target to work towards
* * *	user who wants to set diet plans	add a food entry	can plan my diet
* * *	user who wants to control calorie intake	view the calorie in each food entry	can keep track of my calorie intake
* * *	user who wants to set sports plans	add a sport entry	can plan for my sport exercises
* * *	user who wants to increase calorie consumption	view the calorie consumption for each sport entry	can keep track of my calorie consumption
* * *	user who wants to adjust my diet/sports plans	edit a food/sports entry	can have my plans and records updated
* * *	user who wants to remove my diet/sports plans	delete a food/sports entry	
* * *	user who wants to search for an entry	search by keywords in the entry name	can find related entries without having to scan through all the entries
* * *	user who wants see today’s plans	switch to Today Page and view the daily food/sports plans	can have a general idea of the daily diet/sports arrangements
* * *	user who needs some suggestions for my daily plan	switch to Today Page and view FitHelper feedback	I know whether my daily food/sports plan is suitable
* * *	user who wants to know my daily performance	switch to Today Page and view my performance report	I know my food calorie intake distribution and my task completion
* * *	user who types wrongly sometimes	undo my previous command	I do not need to delete explicitly using a long command
* * *	user who types wrongly sometimes	redo my previous undo command	I can re-executed a undone command
* * *	user who wants to keep a diary	add a diary log for a specific day	note down my schedules, feelings, goals and so on as a self-encouragement
* * *	user who wants to append more content to a previous diary	append new content to existing diaries	enrich my previous diaries' content

Priority	As a …	I want to …	So that I can…
* * *	user who wants to replace the content of a previous diary with new content	edit existing diaries	modify the content to an updated version
* * *	user who wants to remove some diary logs	delete existing diaries	keep abandon some diary logs that I do not want to keep
* * *	user who wants to clear my diary	clear all existing diaries	I can re-start my diary from a white paper
* *	user who wants keep fit	acknowledge my weight change trend according to time	keep track of my weight change easily
* *	user who wants to lose weight	compare between my current weight and target weight	know the gap clearly
* *	user	update my basic information such as address and name if necessary	have an updated profile at any time
* *	user	view pending tasks and status of daily calories goals in a calendar	have cleaner display of data
* *	user who wants to have a clean user interface	clear entries regularly	do not need to see irrelevant information
* *	user	leave the application when I need	It does not occupy additional space in my computer
* *	user	list all entries by certain criteria	I can filter the tasks by what I am looking for
* *	user	get reminders for tasks not done	I can focused on these tasks and complete them
* *	user who do not know very well about dieting and exercising	check calorie intake/consumption of common food and sports	I can input calorie intake/consumption without having to search about these information online.
* *	first-time user	view help page	I can know the functions of the application quickly

Appendix B: 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.

Sorting entry lists

1. sort and display the entry lists stored in the sample FitHelper

    a. Prerequisite: The data stored in FitHelper are the sample data. To restore to sample data, delete all files stored in the data folder and launch the application.

    b. Test case: `sort by/time`
    Expected:

    - The application will go to or remain at the dashboard page.
    - Entries in all three lists (Food Entry History, Sports Entry History and Reminders) are sorted by time in the descending order (entry with a later time comes first i.e. at top).

    c. Test case: `sort x/s by/cal o/a`
    Expected:

    - The application will go to or remain at the dashboard page.
    - Entries in Sports Entry History and Reminders list are sorted by calorie value in the ascending order (entry with a lower calorie value comes first i.e. at top).
    - Entries in Food Entry History list remain unchanged.

    d. Test case: `sort x/s o/a`
    Expected:

    - The application shows invalid command format error, followed by correct command format and example command in the bottom left corner.

Checking calorie data

1. search for calorie intake/consumption of common food/sports
    a. Prerequisite: None, as long as the application is launched properly.
    b. Test case: `check x/s k/swim`
    Expected:
        - The application will go to or remain at the dashboard page.
        - The application shows calorie consumption per hour of common swimming styles in the bottom left corner.
    c. Test case: `check x/f k/swim`
    Expected:
        - The application will go to or remain at the dashboard page.
        - The application shows no matching results message in the bottom left corner.
    d. Test case: `check k/swim`
    Expected:
        - The application shows invalid command format error, followed by correct command format and example command in the bottom left corner.

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.