

# Ng Jun Guang, Jarrod - Project Portfolio

## PROJECT: EYLAH

---

### Overview

EYELAH is a desktop application specifically programmed for Freshmen staying on NUS Campus. It aims to ease their lives at halls/residential colleges by providing them with an application to split bills easily and help them track their diet. The user interacts with it using a CLI and it is programmed using JAVA 11.

### Summary of contributions

- **Major Enhancements:**

1. Adding of **Height, Weight, Mode, Bmi and Self classes, and their storage components for the Diet Tracker** portion of EYLAH (Pull request [#109](#), [#109](#), [#157](#), [#312](#))

- **What it does:**

These classes are used to stored user's metrics and also perform calculations.

- **Justification:**

The User needs a Health Metrics Tracker to aid his Dieting.

- **Highlights:**

Implementing Self was rather tricky at first, since I had used static references class attributes, thinking that it would be stored without a Storage. Together with Akhil, we managed to resolve this issue.

- **Credits:**

My groupmate Akhil, who helped with the correction for some of the bugs in the implementation of the classes. (Akhil's resolution and refactors - Pull request [#318](#))

2. Suggested **for the renaming of same sounding class components across both DietTracker and ExpenseSplitter for EYLAH** ([#Github Discussion](#))

- **What it does:**

Allows the integration of the both Diet Tracker and Expense Splitter seamlessly in the unified EYLAH class.

- **Highlights:**

I noticed this in my review of my team mate, Shee Xiong's, initial reorganisation of the code to allow for unification of both components. Upon further research, Java does not support **class name aliasing**.

- **Credits:**

I understood more about Java Name Aliasing from [here](#). Shee Xiong, our Team's Integrations IC, went ahead and implemented it eventually in this following PR: (Pull request [#232](#))

- **Minor Enhancements:**

1. Performed defensive coding for **bmi**, **height**, and **weight** commands.  
(Height, Weight & Bmi - Pull request [#351](#), [#378](#))

- **What it does:**

**bmi**, **height** and **weight** commands are commands in Diet Tracker, which includes numerical digits and multiple parameters. Common exploits that could break the program was with Integer Overflow values. I combat these erroneous commands by setting limits on their values by checking them through BigDecimal.

- **Justification:**

Significantly reduces the chances of breaking Diet Tracker and crashing EYLAH.

2. Fixed the Calories bug that occurs when Calories are not in the desired range. (Pull request [#354](#))

- **What it does:**

Properly validates the user's Calories input for any of the Diet Tracker Commands which involve Calories.

- **Highlights**

The original implementation had failed to consider a `VALIDATION_REGEX` for the Calories String itself, since Calories was built using a long instead of directly from a String. Another issue was that you could exceed in the input long for Calories which would crash the program also. This was circumvented using BigInteger.

- **Code contributed:** [[Functional Code](#) and [Test Code](#)]

- **Other contributions:**

- Project management:

- In charge of Diet Tracker functionalities for EYLAH.
    - Drew most UML Diagrams for the various Diet Tracker classes and functions for EYLAH.

- Enhancements to existing features:

- Adapted the given Parser and Command Design into Diet Tracker's Design for EYLAH. (Commands - Pull request [#157](#)) (Parsers - [#107](#))

- Documentation:

- Updated Developer Guide to include Bmi, Height, Weight, Mode, Metrics Commands and diagrams (Pull request [#372](#), [#376](#), [#401](#), [#404](#))
    - Updated Developer Guide Template for Diet Tracker(Pull request [#372](#))
    - Updated User Guide to include the above Commands (Pull request [#361](#))

- Community:

- PRs reviewed: (Pull requests: [#128](#), [#134](#), [#135](#), [#217](#))
    - Reported bugs and suggestions for other teams in the class (examples: <https://github.com/jarrodbob/ped/issues>)

# 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.*

## Diet Tracker : `diet`

Using a Dieting App has never been easier! You can easily achieve the functionalities of a standard Dieting App with Diet Tracker.

Diet Tracker will help you achieve your ideal weight and body mass!

You can store all your food intake everyday and calculate useful information.

## Calculating Body Mass Index (BMI) `bmi`

In this section, you will learn more about the `bmi` command, [how to use it](#) and the [expected outcome](#) after using the `bmi` command.

### Summary of Bmi Command:

`bmi` You can calculate your BMI either through an input height and weight or your previously stored Height and Weight.

It will display the following data:

- BMI value

### How to use the Bmi Command:

- There are 3 ways to use `bmi`.
- The first is if there is no input height and weight. This will use the height and weight that is stored in the Self object.
- The second is if there is either no input height or input weight. This will use the stored Height (in the case of missing input height) or stored Weight (in the case of missing input weight) to do the calculation instead.
- The third is to calculate bmi with an input height and weight.

Format:

```
bmi [-h HEIGHT] [-w WEIGHT]
```

Valid Examples:

- `height 172`  
`weight 65`  
`bmi`

Change your height and your weight to your current measurements before calculating your BMI. BMI is calculated based off the stored height and weight in this instance.

- `height 173.5`

`bmi -w 59.9`

Change your height to your current measurements before calculating your BMI. BMI is calculated based off the stored height and input weight in this instance.

- `bmi -h 172 -w 65.5`

Calculate BMI based on the input height and weight values.

**Expected outcome:**

```
Currently at DIET
Enter command:
bmi -h 172 -w 65.5
The BMI Calculated is: 22.140345
Your BMI is in the Normal category.
```

**Additional tips**

**TIP**

If you are unsure whether you have already input your height and weight, you may `metrics` to check.

## Storing Height `height`

In this section, you will learn more about the `height` command, [how to use it](#) and the [expected outcome](#) after using the `height` command.

### Summary of Height Command:

`height` You can use this command to save your Height to the Diet Tracker.

### How to use the Height Command:

Format:

`height HEIGHT`

Valid Example:

`height 170.2`

**Expected outcome:**

```
Currently at DIET
Enter command:
height 170.2
Added Height: 170.2
```

## Additional notes and tips

NOTE	Height in centimetres (cm). Decimal places are accepted. I.e. <code>height 172.305</code> is accepted.
TIP	Check your stored height with <code>metrics</code> .

---

## Storing Weight `weight`

In this section, you will learn more about the `weight` command, [how to use it](#) and the [expected outcome](#) after using the `weight` command.

### Summary of Weight Command:

`weight` You can use this command to save your Weight to the Diet Tracker.

### How to use the Weight Command:

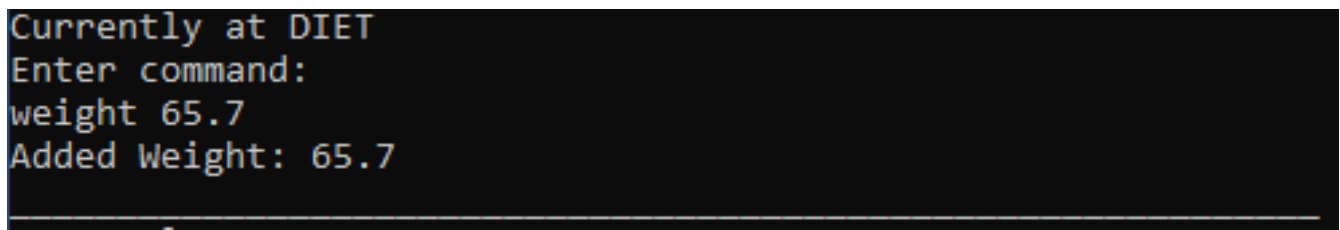
Format:

`weight WEIGHT`

Valid Example:

`weight 65.7`

### Expected outcome:



```
Currently at DIET
Enter command:
weight 65.7
Added Weight: 65.7
```

## Additional notes and tips

NOTE	Weight in kilograms (kg). Decimal places are accepted. I.e. <code>weight 65.77</code> is accepted.
TIP	Check your stored weight with <code>metrics</code> .

---

## Dieting Mode `mode`

In this section, you will learn more about the `mode` command, [how to use it](#) and the [expected outcome](#) after using the `mode` command.

### Summary of Mode Command:

`mode` You can set your desired dieting goal with the different modes that set a limit on your daily calorie intake. The calorie limits are calculated based on that of an average human. Your daily

calories intake status can be monitored with the `list` command.

Switch the dieting mode, based on the following modes:

- Lose Weight (-l) (2000 calorie limit)
- Gain Weight (-g) (3000 calorie limit)
- Maintain (-m) (2500 calorie limit)

### How to use the Mode Command:

Format:

`mode [-l] [-g] [-m]`

Valid Example:

`mode -l`

<b>WARNING</b>	You must only input <b>EXACTLY ONE</b> mode per mode command.
----------------	---

Expected outcome:

```
Currently at DIET
Enter command:
mode -l
Mode Change Successful: LOSS
```

### Additional notes and tips

<b>NOTE</b>	The default mode is MAINTAIN if you have not set your mode.
-------------	---

- |            |   |
|------------|---|
| <b>TIP</b> | <ul style="list-style-type: none"><li>• Switch your Dieting Mode to help yourself reach your diet targets better!</li><li>• Check your currently chosen Dieting Mode with <code>metrics</code>.</li></ul> |
|------------|---|

## Showing your Metrics `metrics`

In this section, you will learn more about the `metrics` command, [how to use it](#) and the [expected outcome](#) after using the `metrics` command.

### Summary of Metrics Command:

`metrics` You can print out your individual metrics (Height, Weight and Mode) to check them.

It will display the following data:

- Your height
- Your weight
- Your chosen Dieting Mode

## How to use the Metrics Command:

Format:

`metrics`

Valid Example:

`metrics`

Expected outcome:

```
Currently at DIET
Enter command:
metrics
Your metrics are as follows:

    Height: 170.2
    Weight: 65.7
    Dieting Mode: LOSS
Your metrics are shown.
```

## Additional notes and tips

### NOTE

Diet Tracker will prompt you if you did not have any previously stored Height, Weight.

### TIP

Use this to check whether you have previously stored a Height, a Weight, or have chosen your Dieting Mode already.

## Editing a Food Item `edit`

In this section, you will learn more about the `edit` command, [how to use it](#) and the [expected outcome](#) after using the `edit` command.

### Summary of Edit Command:

`edit` You can edit either the Food name, or the calories of the food at the specified index.

### How to use the Edit Command:

Format:

`edit -i INDEX [-n NAME] [-c CALORIES]`

- Edits the Food Item at the specified **INDEX**. The index refers to the index number shown in the displayed Food list. The index **must be a positive integer** 1, 2, 3, ...
- At least one of the optional fields must be provided.
- Existing values will be updated to the input values.

Valid Example:

```
edit -i 2 -n Chicken Rice -c 585
```

Edits the name of the food item at index 2 to be 'Chicken Rice' and the calories to be '585'.

#### WARNING

You **MUST** use **list** to check the list of items to identify a target to edit. This would ensure that you get the correct index of the item.

Expected outcome:

```
Currently at DIET
Enter command:
list
These are all food that you have consumed for today!

1. Fishball Noodles Calories: 383 At: 2020-04-13 Tags: [noodles][favourite]
2. Duck Rice Calories: 492 At: 2020-04-13 Tags: [favourite][rice]

Total Calorie Intake : 875

You have 1125 calories left for the day!
All foods over period based on input tag has been listed.

-----
Currently at DIET
Enter command:
edit -i 2 -n Chicken Rice -c 585
Food Edited: Chicken Rice Calories: 585 At: 2020-04-13 Tags: [favourite][rice]
```

Additional notes and tips

#### TIP

You can list based on time period or tags to find the item that you want to edit.

## 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.*

### Diet Tracker feature

The Diet Tracker feature is designed to aid our users in maintaining a healthy lifestyle. The feature comprises of 10 Commands.



- **AddCommand** - Creates a new Food object with its attributes (Name, Calories) and adds it to the FoodBook Storage.
- **DeleteCommand** - Deletes the Food specified by the input index from FoodBook Storage.
- **ListCommand** - Lists the Foods and its attributes (Name, Calories) for the timeframe specified by users based on their user input.
- **EditCommand** - Allows the user to edit an of the Food in Storage.
- **HeightCommand** - Allows users to log their Height in centimeters.
- **WeightCommand** - Allows users to log their Weight in kilograms.
- **BmiCommand** - Calculates the BMI.
- **ModeCommand** - Allows users to toggle between different modes of the diet tracker.
- **MetricsCommand** - Allows users to check their health metrics, like their Height, Weight and Dieting Mode. ""

==== Edit Command

In this section, we will learn more about how the **edit** command is implemented.

### What is the Edit Command

The **edit** command allows users to edit the Name of the Food or the Calories of the Food from the FoodBook via the Index.

The **edit** command was implemented as **EditCommand** in the **diettracker/logic/commands** package.

The **edit** command has the following input format:

```
edit -i INDEX [-n NAME] [-c CALORIES]
```

#### NOTE

- **INDEX** is a compulsory field.
- The Index of the Food to be edited **MUST** be retrieved by using the **list** command.
- At least one of **NAME** or **CALORIES** must be included in the command input.

The following activity diagram illustrates what happens when a user executes the **edit** command:

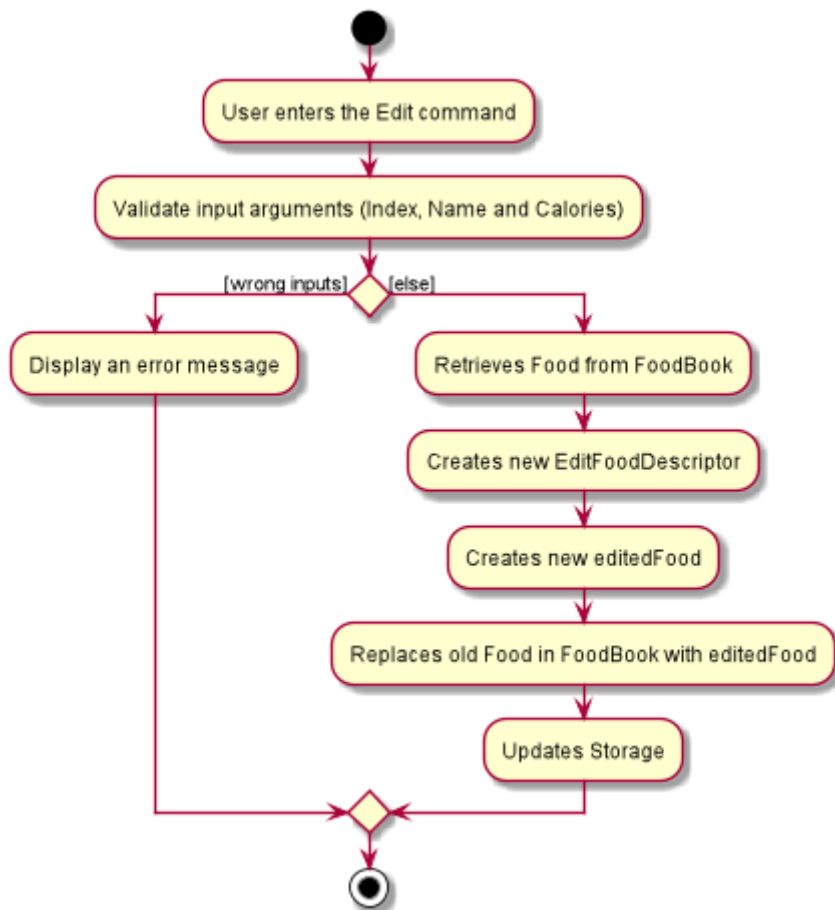


Figure 1. Edit Command Activity Diagram

### Structure of Edit Command

In this section, you will learn more about the relationships between objects related to the `edit` command.

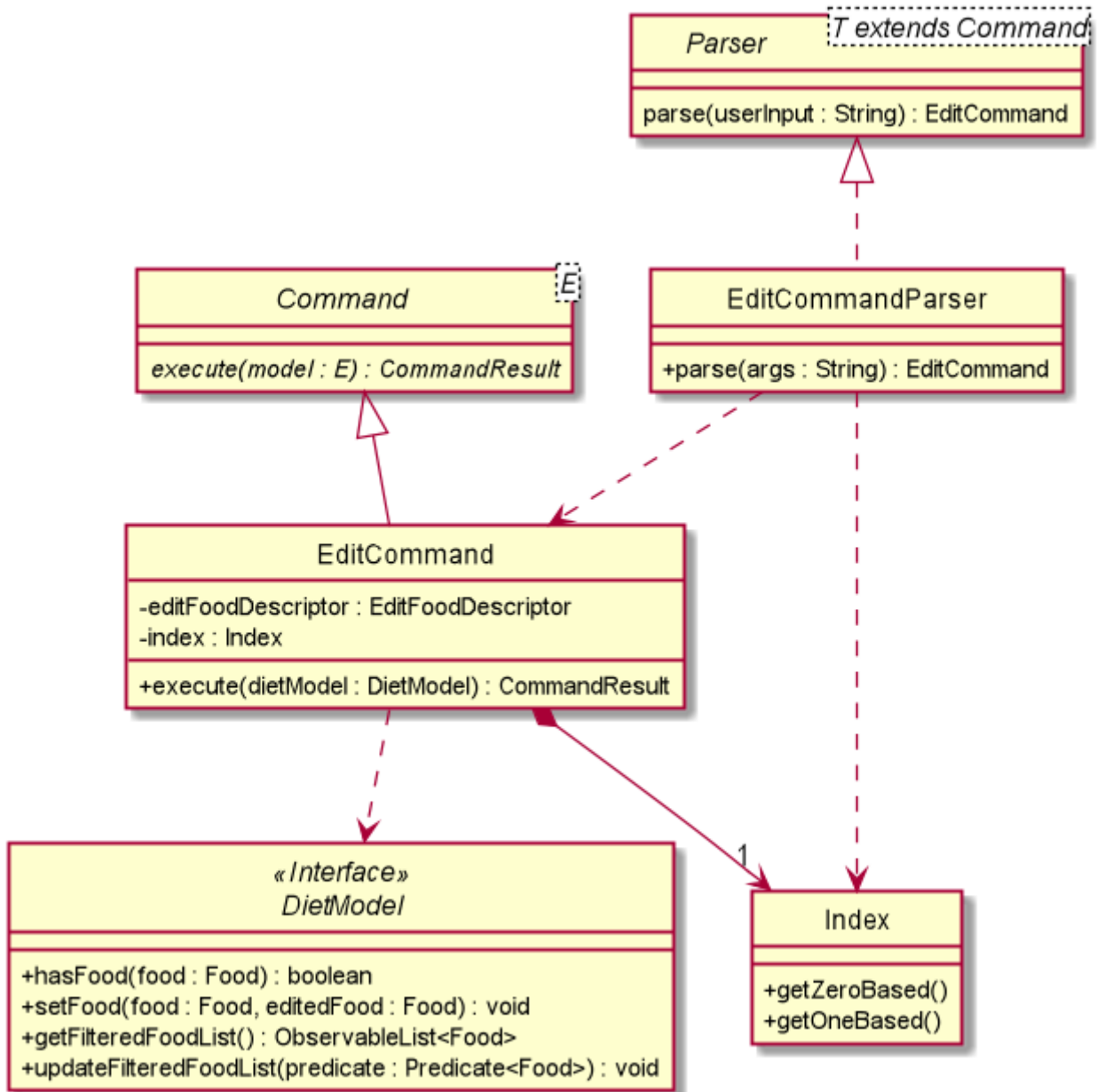


Figure 2. Edit Command Class Diagram

The above class diagram shows the structure of the `EditCommand` and its associated classes and interfaces. Some methods and fields are left out because they are not of concern in `EditCommand`.

## Implementation

The following is a detailed explanation of the operations `EditCommand` performs.

1. The `EditCommand#execute(DietModel dietModel)` method is executed and it validates that the specified `INDEX` to edit is within range. If valid, the item to be edited will be retrieved from Storage using its `Index`.
2. The method `DietModel#getFilteredFoodList()` will then be called to retrieve the List of Foods from Storage. `List#get(int Index)` is then invoked which retrieves the specified Food to be edited.
3. The method `DietModel#setFood(Food toBeEdited, Food editedFood)` will then be called to replace the Food toBeEdited with the Food editedFood in the List of Foods.

4. If successful, a success message will be generated by `CommandResult` and it will be returned with the generated success message. Otherwise, an error message showing the correct command syntax is thrown as `CommandException`.
5. If the command syntax was valid and Food was edited in FoodBook, `LogicManager` calls `FoodBookStorage#saveFoodBook(ReadOnlyFoodBook foodBook)` which saves the new Foods into JSON format after serializing it using `JsonAdaptedFood`.

## Sequence Diagram for Edit Command

The following sequence diagram summarizes what happens during the execution of `edit` command.

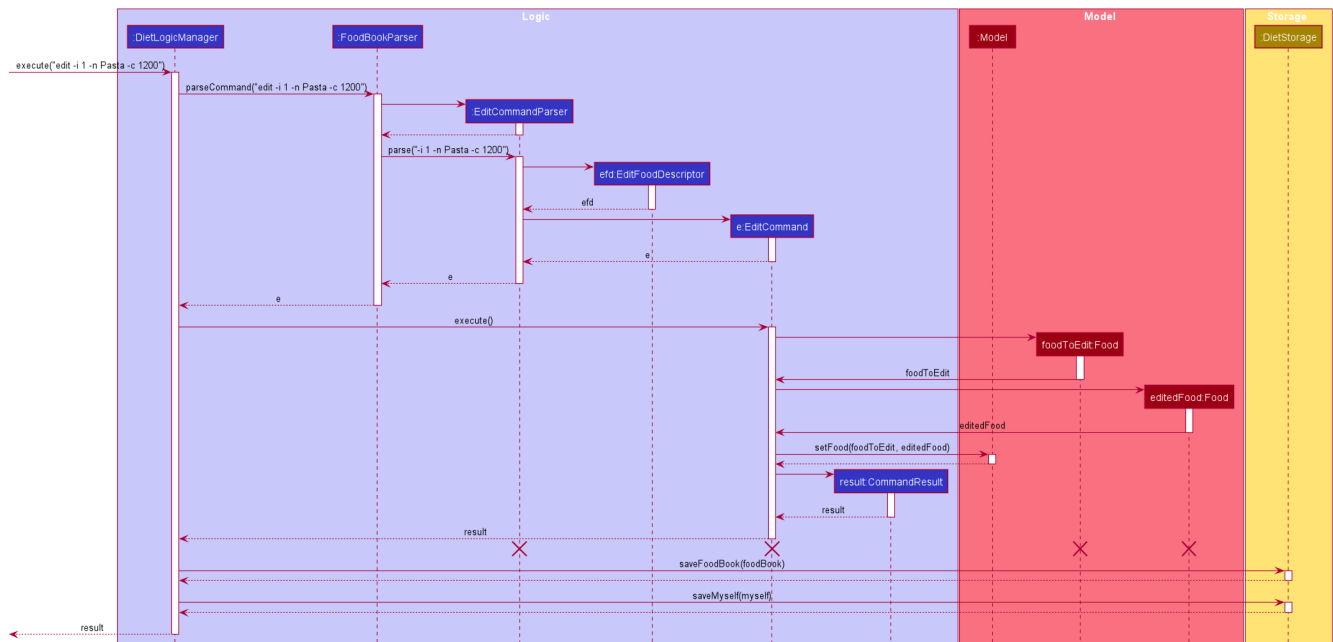


Figure 3. Edit Command Sequence Diagram

## Bmi Command

In this section, we will learn more about how the `bmi` command is implemented.

### What is the Bmi Command

The `bmi` command allows the user to calculate their Body Mass Index (BMI).

The `bmi` command was implemented as `BmiCommand` in the `diettracker/logic/commands` package.

The `bmi` command has the following input format:

`bmi [-h HEIGHT] [-w WEIGHT]`

#### NOTE

- `[-h HEIGHT]` and `[-w WEIGHT]` may be omitted if the user has already stored their Height and Weight.
- If Users have one of Height or Weight stored, they may use just the missing metric to calculate their BMI.
- `HEIGHT` and `WEIGHT` can range from  $>0$  to  $\leq 1000$ .

The following activity diagram illustrates what happens when a user executes the `bmi` command:

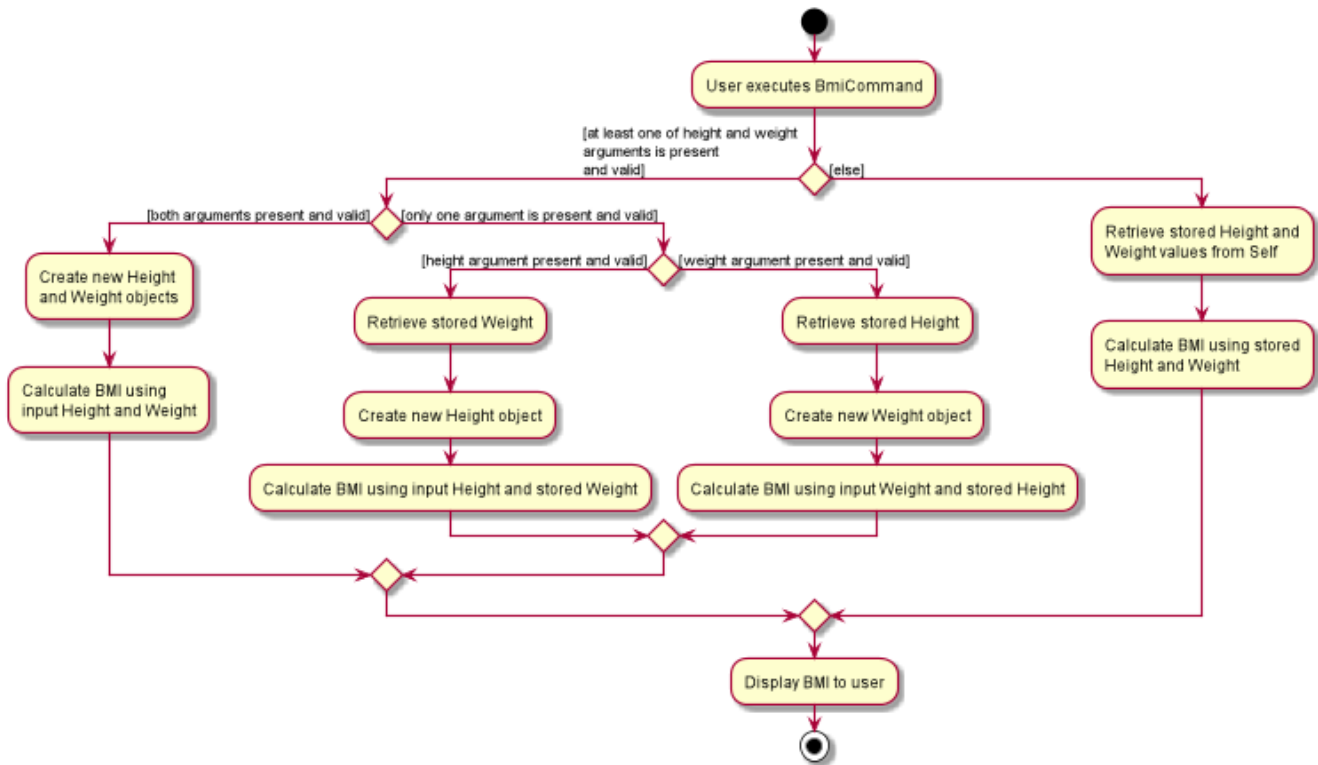


Figure 4. Bmi Command Activity Diagram

#### Structure of Bmi Command

In this section, you will learn more about the relationships between objects related to the `bmi` command.

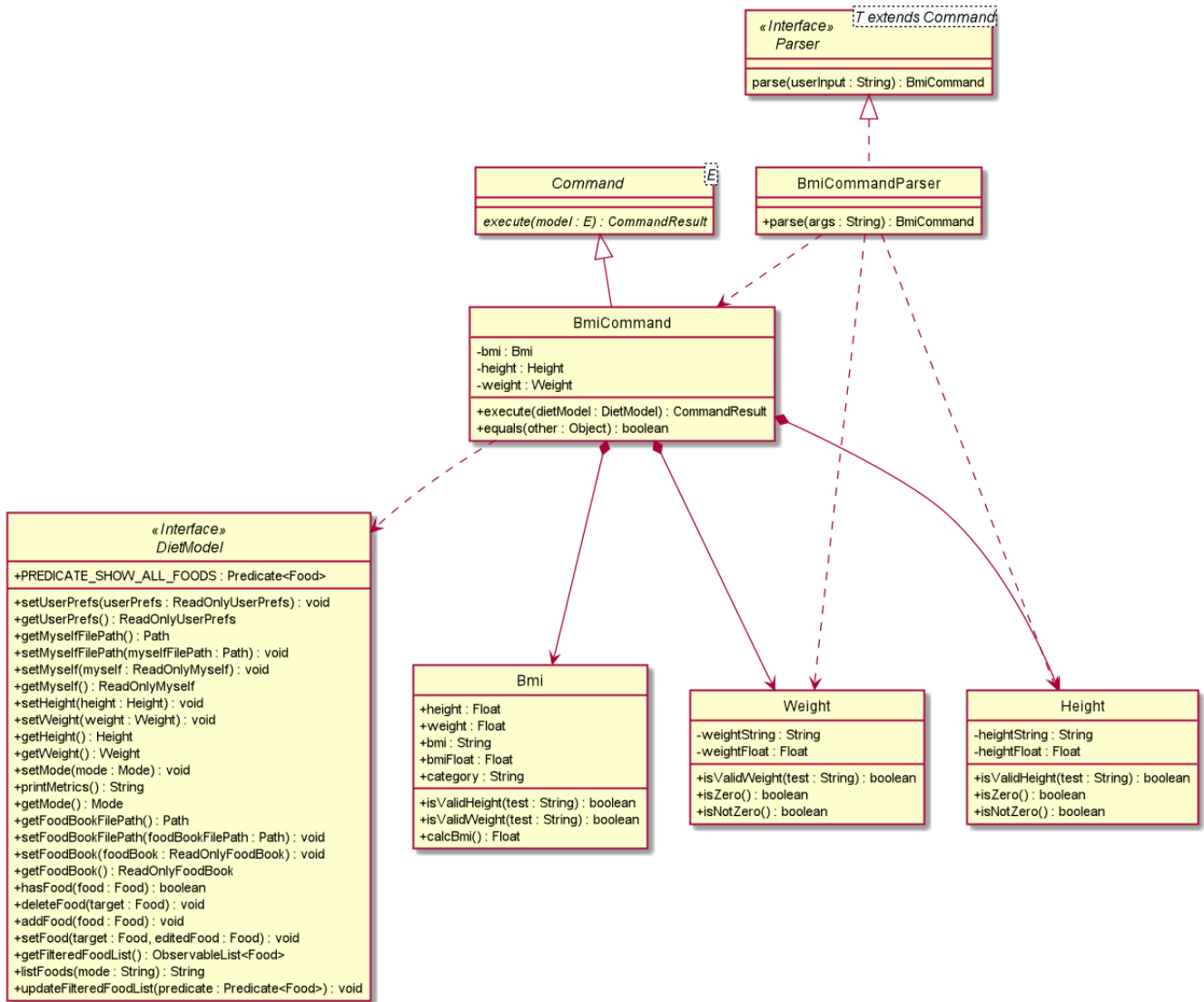


Figure 5. Bmi Command Class Diagram

The above class diagram shows the structure of the **BmiCommand** and its associated classes and interfaces. Some methods and fields are left out because they are not of concern in **BmiCommand**

## Implementation of Bmi Command

The following is a detailed explanation of the operations **BmiCommand** performs. **BmiCommand** has two different usages depending on the user input.

1. The **BmiCommand#execute(Model dietModel)** method is executed and it will return the output of the calculated BMI based on user arguments.
2. If successful, a success message will be generated by **CommandResult** and it will be returned with the generated success message. Otherwise, an error message showing the correct command syntax is thrown as **CommandException**.

## Sequence diagram for Bmi Command

Given below are 2 example usages of **BmiCommand** based on different user input.

### Usage 1: No Height and Weight input

1. User launches application and enters **Diet** mode. The user then enters **bmi** as the command.
2. The FoodBook parser validates this command and sets up the **BmiCommandParser**, which checks for the input.
3. Since there are no arguments, the **BmiCommandParser** will call the empty constructor **BmiCommand()**.
4. **BmiCommand** would then refer to the internal state of the splitterModel under Self, and retrieve the values stored in Self's Height and Weight attributes.
5. **BmiCommand()** will then proceed to calculate the BMI based on the current values of height and weight.

The following is a sample sequence diagram of the **BmiCommand** with no additional user input.

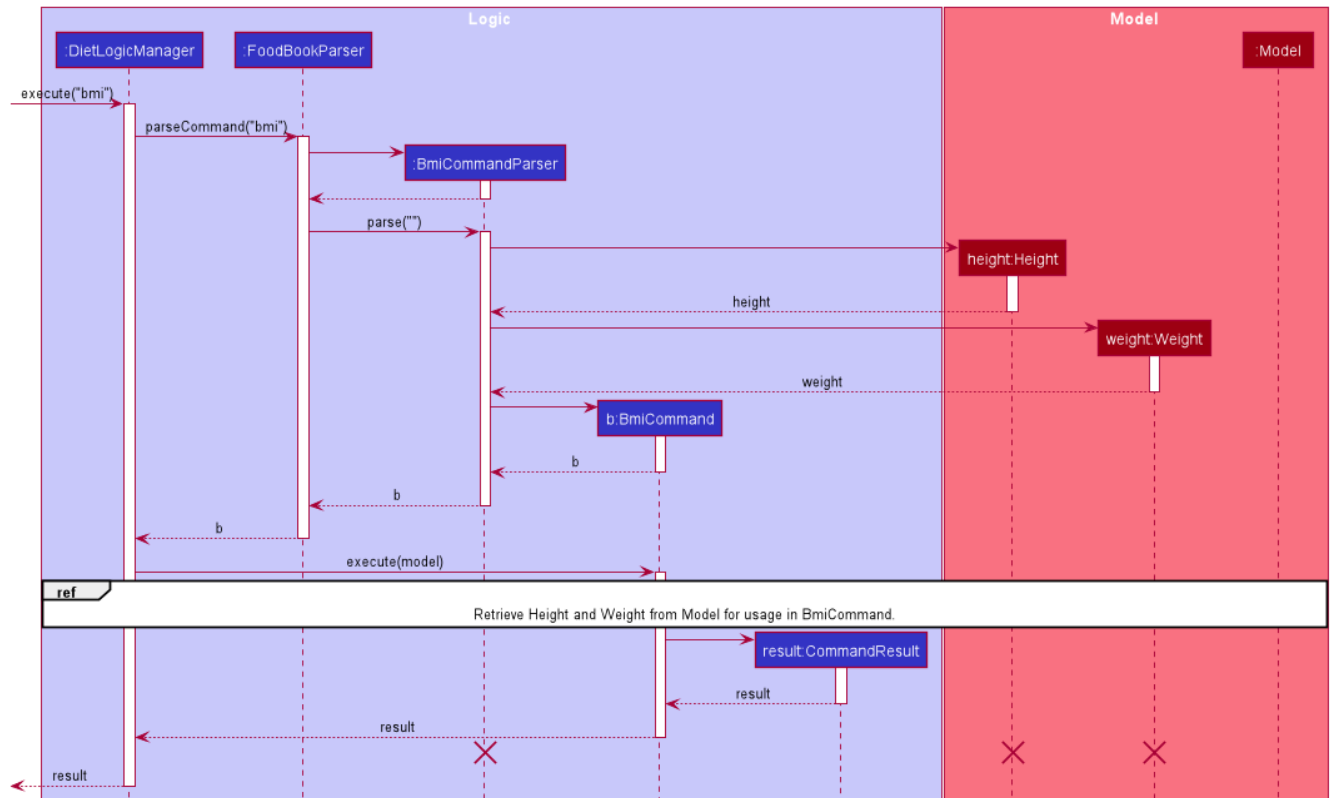


Figure 6. Sequence Diagram Bmi Command Sequence Diagram without Input Arguments

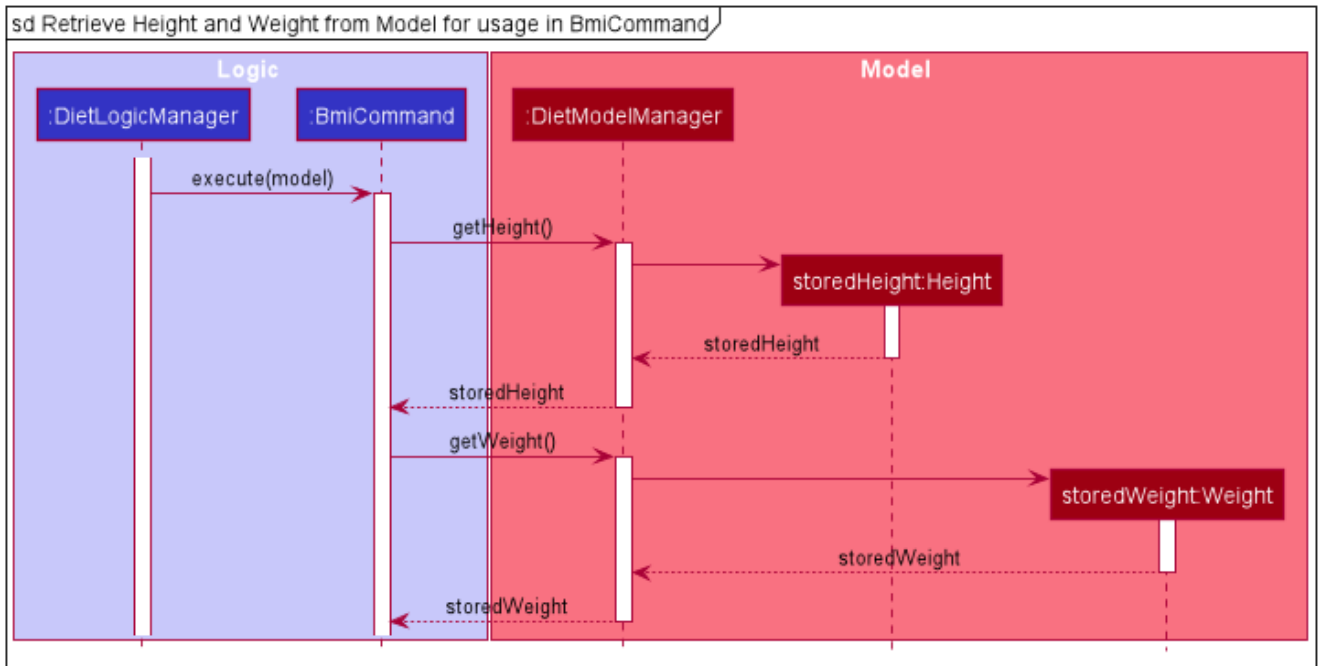


Figure 7. Sequence Diagram: Retrieval of stored Height and Weight from Model

#### NOTE

There is a need to ensure that there are stored values in **Height** and **Weight** attributes in the **Self** class.

#### Usage 2: With Height and Weight input

1. User launches application and enters **Diet** mode. The user then enters **bmi** as the command.
2. The FoodBook parser validates this command and sets up the **BmiCommandParser**, which checks for the input.
3. Since there are no arguments, the **BmiCommandParser** will call the empty constructor **BmiCommand()**.
4. **BmiCommand** would then refer to the internal state of the splitterModel under **Self**, and retrieve the values stored in **Self**'s **Height** and **Weight** attributes.
5. **BmiCommand()** will then proceed to calculate the BMI based on the current values of height and weight.

The following is a sample sequence diagram of the **BmiCommand** with additional user input.



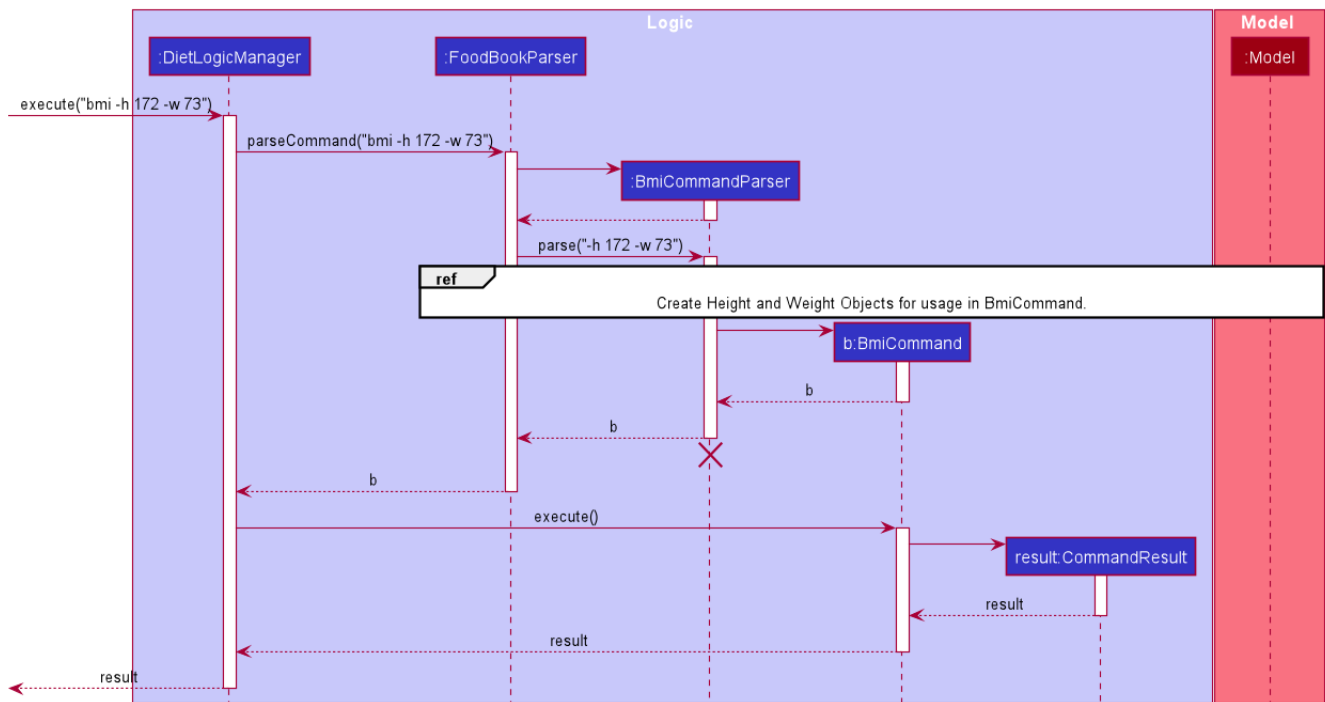


Figure 8. Bmi Command Sequence Diagram with Input Arguments

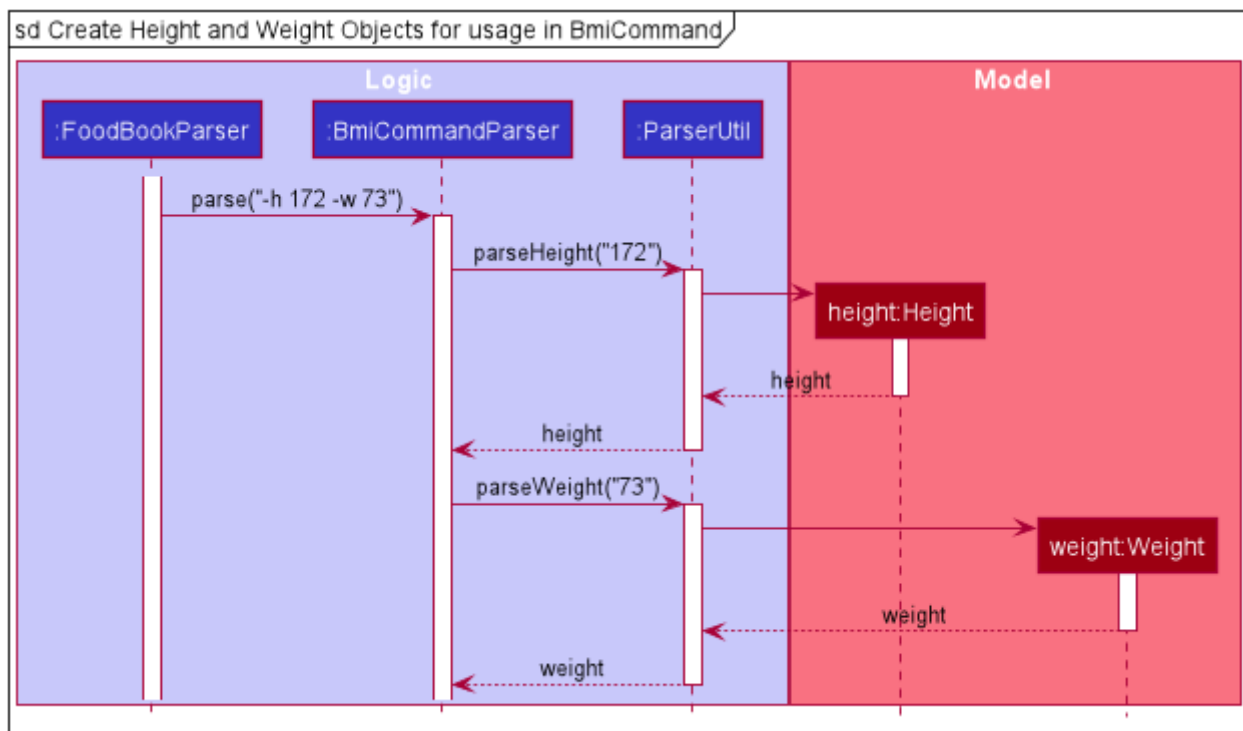


Figure 9. Creation of Height and Weight Objects for usage in Bmi Command

## Design Considerations

Aspect: How `BmiCommand` executes

- Alternative 1 (current choice): Executes with other without arguments
  - Pros: More flexible use of the Command, better user experience overall.
  - Cons: Harder to implement, as there needs to be multiple `BmiCommand` constructors.
- Alternative 2: Executes separately with arguments input and without arguments input

- Pros: Easier to implement, less potential bugs as Command uses a single constructor.
- Cons: We must ensure that the implementation of each individual command are correct.

#### Aspect: Storage of BMI

- Alternative 1 (current choice): No splitterStorage of BMI value, simply prints when user requests.
  - Pros: Less memory used; reduces complexity of the Command and objects involved.
  - Cons: Users may want to access it elsewhere from Self.
- Alternative 2: Storage of BMI value in Self class in Model.
  - Pros: Users have access to it anytime.
  - Cons: Coding complexity. "

==== Height Command

In this section, we will learn more about how the `height` command is implemented.

#### What is the Height Command

The `height` command allows the user to store their Height into the Diet Tracker.

The `height` command was implemented as `HeightCommand` in the `diettracker/logic/commands` package.

The `height` command has the following input format:

`height HEIGHT`

#### NOTE

- `HEIGHT` is a **compulsory** field.
- `HEIGHT` can range from `>0` to `<1000`. `HEIGHT` can be input as a decimal.

The following activity diagram illustrates what happens when a user executes the `height` command:

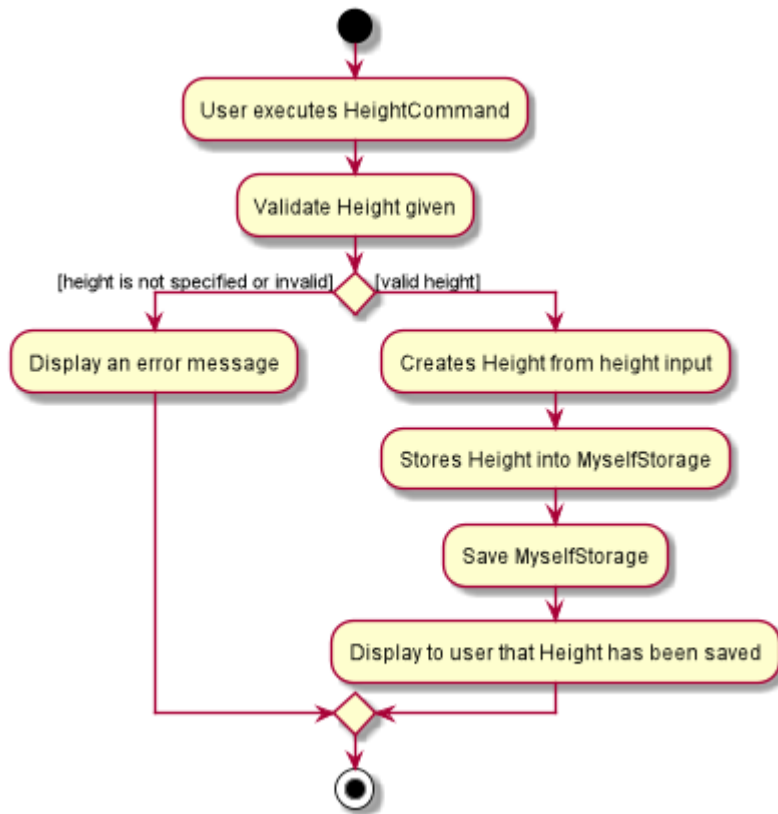


Figure 10. Height Command Activity Diagram

### Structure of Height Command

In this section, you will learn more about the relationships between objects related to the **height** command.

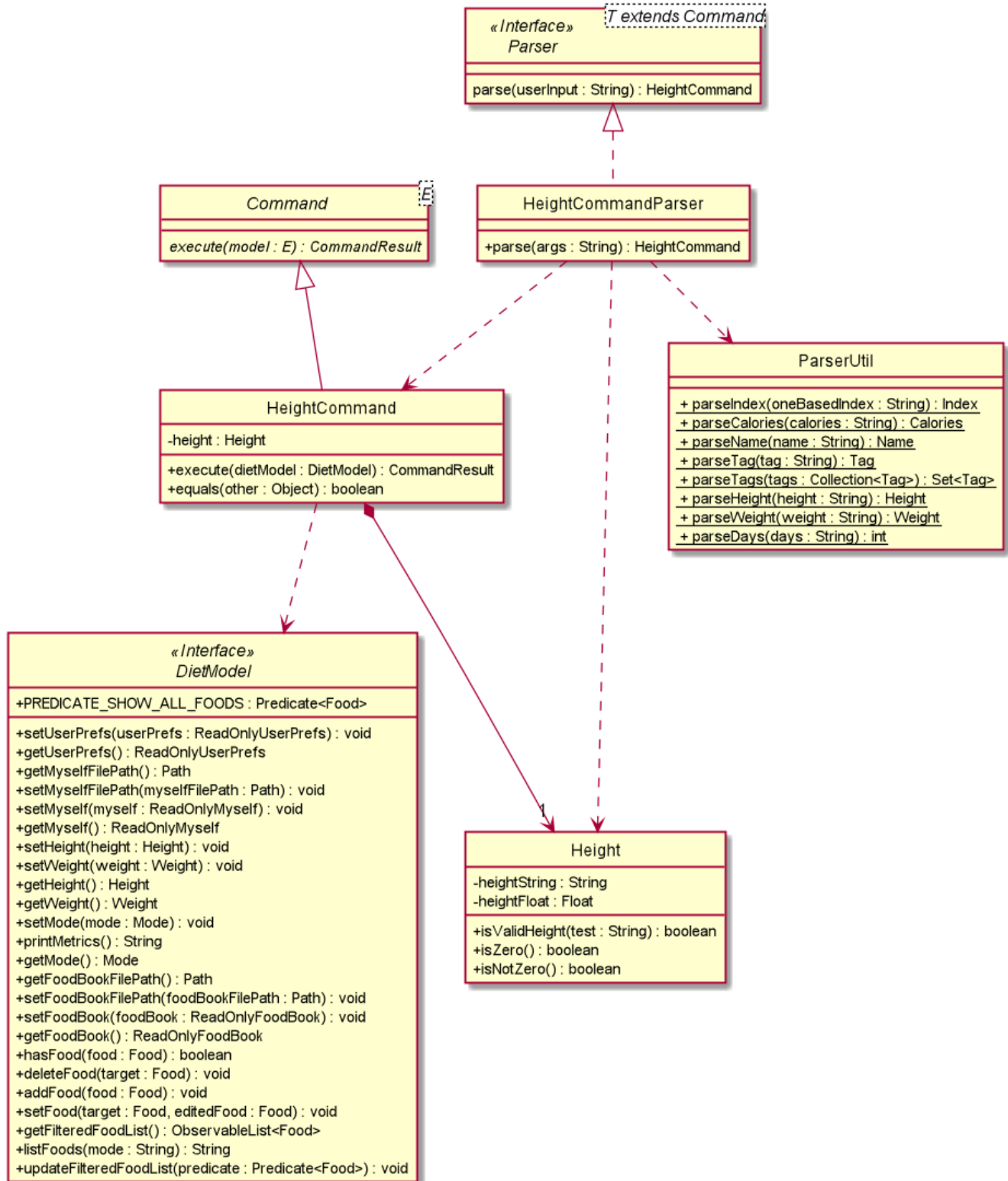


Figure 11. Height Command Class Diagram

The above class diagram shows the structure of the **HeightCommand** and its associated classes and interfaces. Some methods and fields are left out because they are not of concern in **HeightCommand**.

## Implementation of Height Command

The following is a detailed explanation of the operations **HeightCommand** performs.

1. The **HeightCommand#execute(DietModel dietModel)** method is executed and it validates that the specified **HEIGHT** to store is a valid Height. If valid, the height will be stored in the **Self** class.

2. The method `DietModel#setHeight(Height height)` will then be called to set the Height of the `Self` class. `Self#setHeight(Height height)` is invoked which makes a call to its internal Height to replace the value stored.
3. If successful, a success message will be generated by `CommandResult` and it will be returned with the generated success message. Otherwise, an error message showing the correct command syntax is thrown as `CommandException`.

## Sequence diagram for Height Command

The following sequence diagram summarizes what happens during the execution of `height` command.

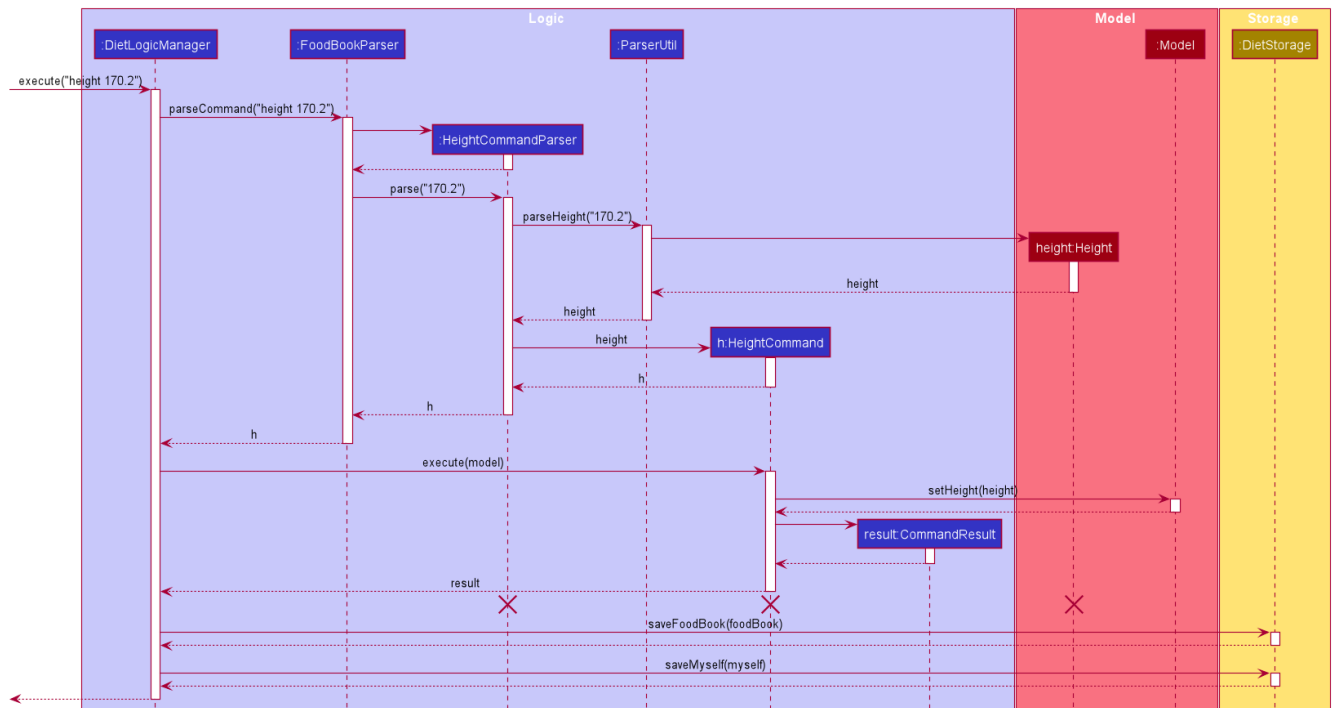


Figure 12. Height Command Sequence Diagram

## Weight Command

In this section, we will learn more about how the `weight` command is implemented.

### What is the Weight Command

The `weight` command allows the user to store their Weight into the Diet Tracker.

The `weight` command was implemented as `WeightCommand` in the `diettracker/logic/commands` package.

The `weight` command has the following input format:

`weight WEIGHT`

#### NOTE

- `WEIGHT` is a **compulsory** field.
- `WEIGHT` can range from `>0` to `<1000`. `WEIGHT` can be input as a decimal.

The following activity diagram illustrates what happens when a user executes the **weight** command:

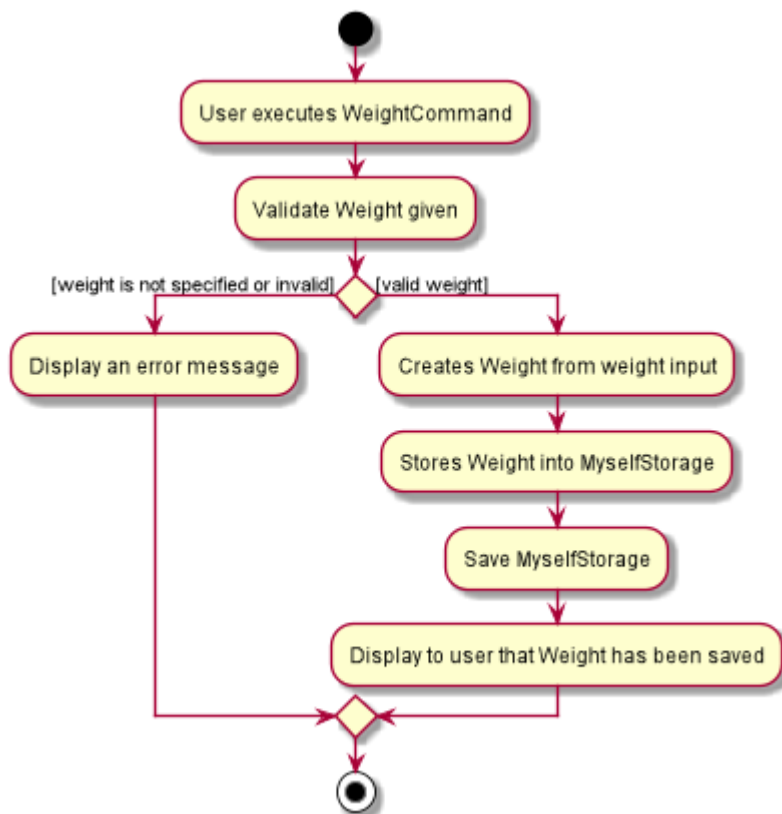


Figure 13. Weight Command Activity Diagram

### Structure of Weight Command

In this section, you will learn more about the relationships between objects related to the **weight** command.

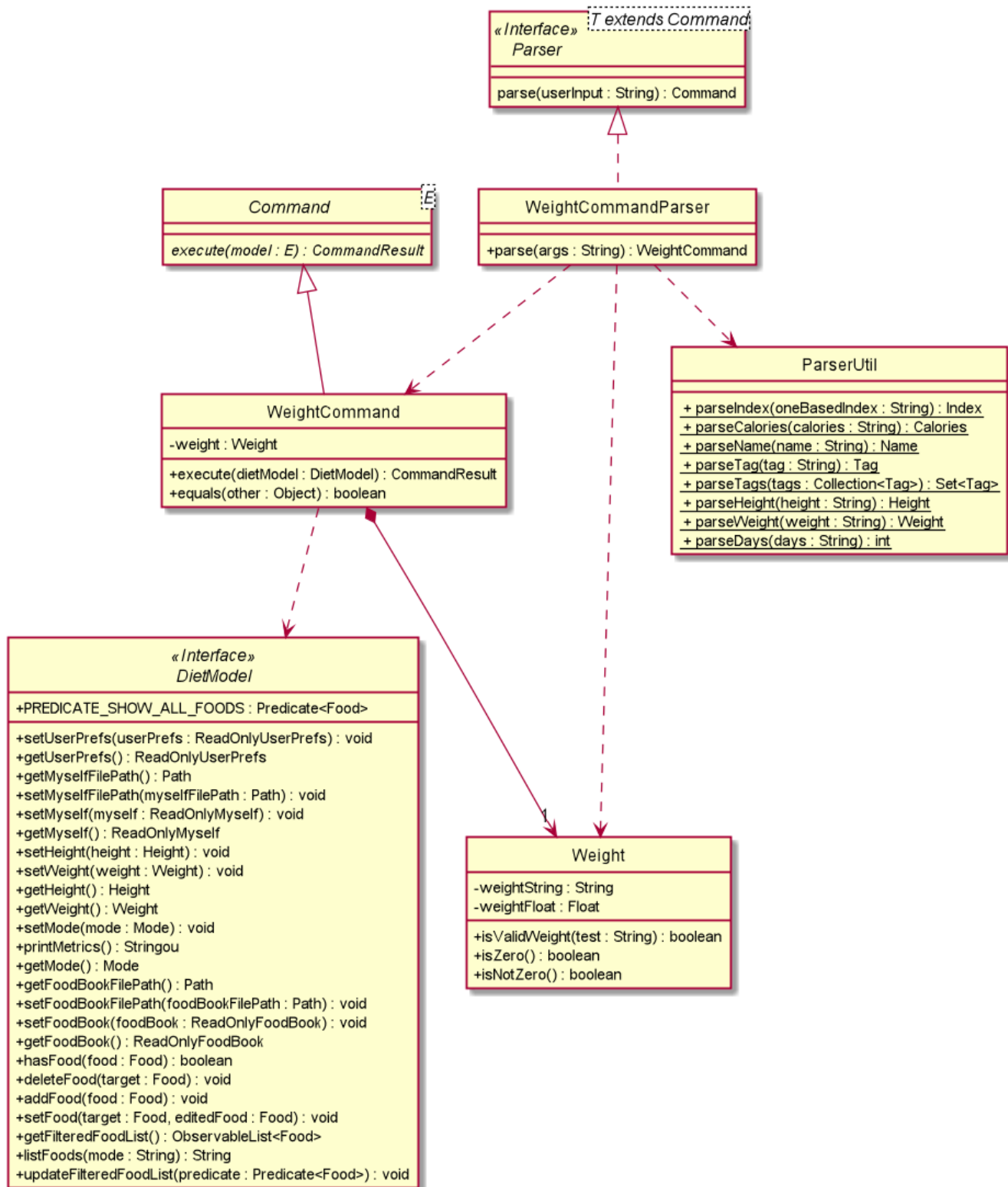


Figure 14. Weight Command Class Diagram

The above class diagram shows the structure of the **WeightCommand** and its associated classes and interfaces. Some methods and fields are left out because they are not of concern in **WeightCommand**.

## Implementation of Weight Command

The following is a detailed explanation of the operations **WeightCommand** performs.

1. The **WeightCommand#execute(DietModel dietModel)** method is executed and it validates that the specified **WEIGHT** to store is a valid **Weight**. If valid, the **Weight** will be stored in the **Self** class.

2. The method `DietModel#setWeight(Weight weight)` will then be called to set the Weight of the `Self` class. `Self#setWeight(Weight weight)` is invoked which makes a call to its internal Height to replace the value stored.
3. If successful, a success message will be generated by `CommandResult` and it will be returned with the generated success message. Otherwise, an error message showing the correct command syntax is thrown as `CommandException`.

## Sequence diagram for Weight Command

The following sequence diagram summarizes what happens during the execution of `weight` command.

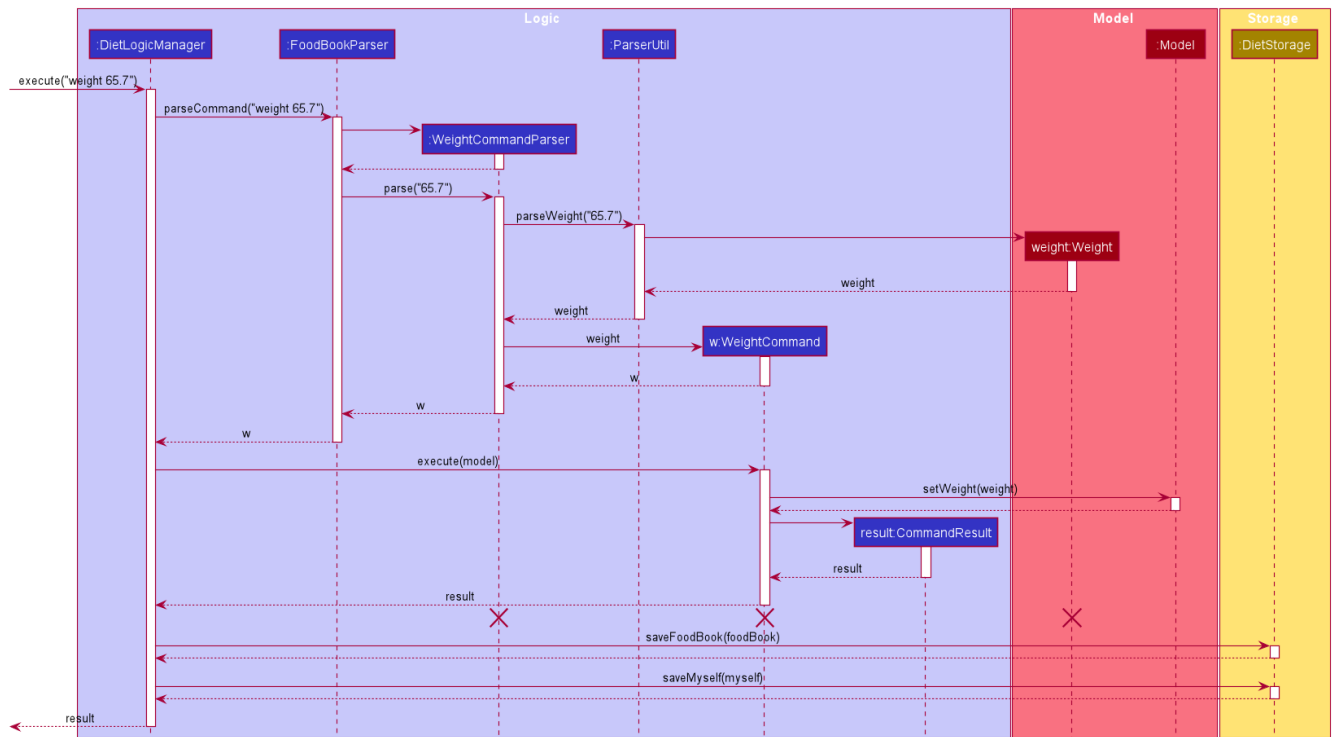


Figure 15. Weight Command Sequence Diagram

## Mode Command

In this section, we will learn more about how the `mode` command is implemented.

### What is the Mode Command

The `mode` command allows the user to store their Dieting Mode into the Diet Tracker.

The `mode` command was implemented as `ModeCommand` in the `diettracker/logic/commands` package.

The `mode` command has the following input format:

`mode [-l] [-g] [-m]`

**NOTE** Users must only enter **EXACTLY ONE** of the given flags for the mode.



The following activity diagram illustrates what happens when a user executes the **mode** command:

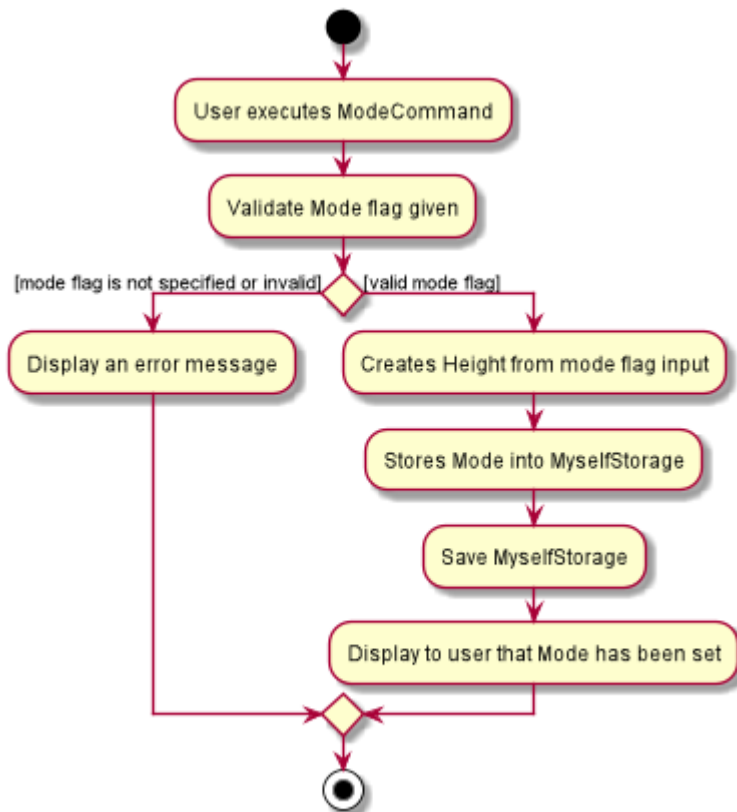


Figure 16. Mode Command Activity Diagram

### Structure of Mode Command

In this section, you will learn more about the relationships between objects related to the **mode** command.

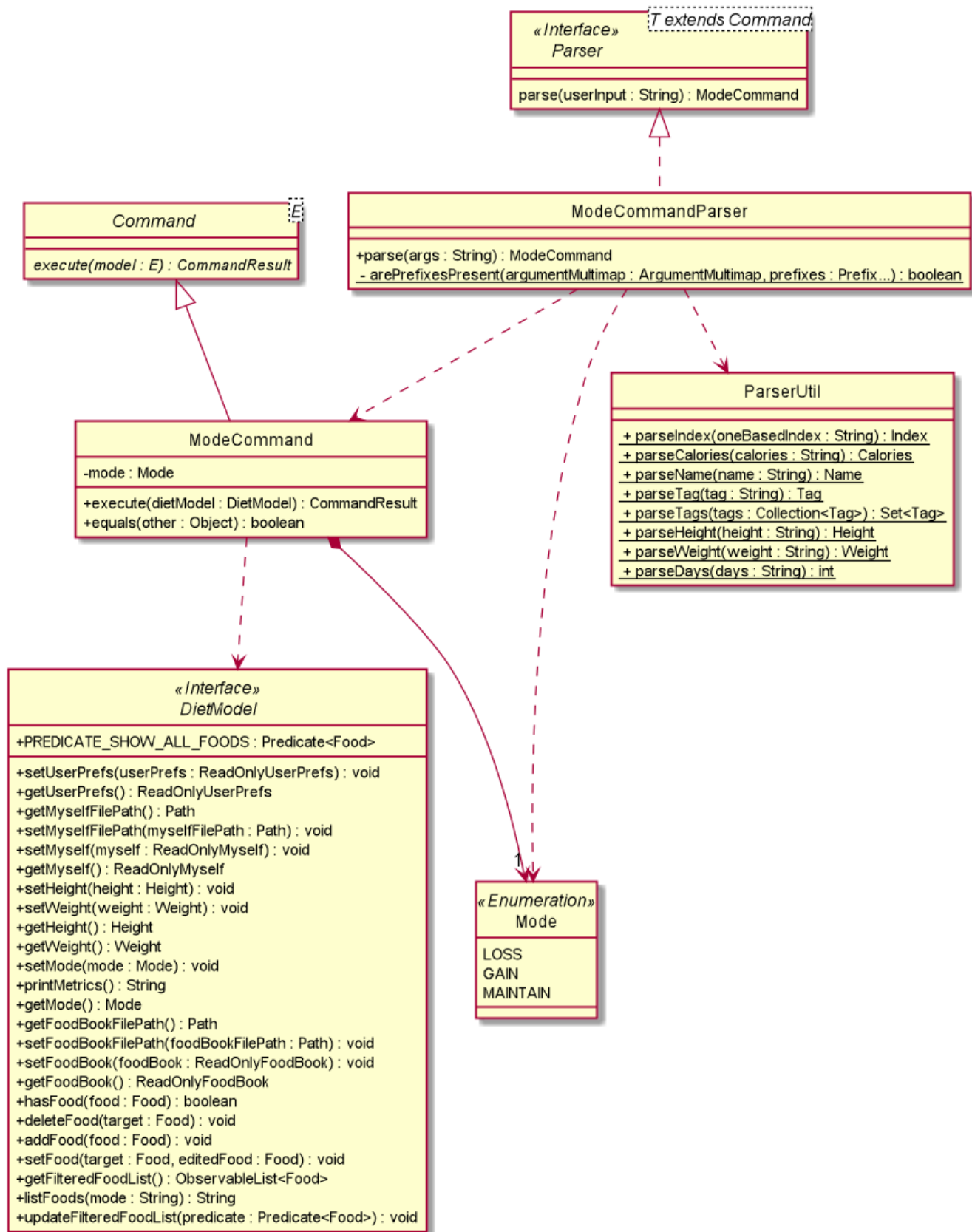


Figure 17. Mode Command Class Diagram

The above class diagram shows the structure of the **ModeCommand** and its associated classes and interfaces. Some methods and fields are left out because they are not of concern in **ModeCommand**.

## Implementation of Mode Command

The following is a detailed explanation of the operations **ModeCommand** performs.

1. The `ModeCommand#execute(DietModel dietModel)` method is executed and it validates that the specified `MODE` (based on the input flag) to store is a valid flag. If valid, the corresponding mode to the flag will be stored in the `Self` class.
2. The method `DietModel#setMode(Mode mode)` will then be called to set the Mode of the 'Self' class. `Self#setMode(Mode mode)` is invoked which makes a call to its internal Mode to replace the value stored.
3. If successful, a success message will be generated by `CommandResult` and it will be returned with the generated success message. Otherwise, an error message showing the correct command syntax is thrown as `CommandException`.

## Sequence diagram for Mode Command

The following sequence diagram summarizes what happens during the execution of `mode` command.

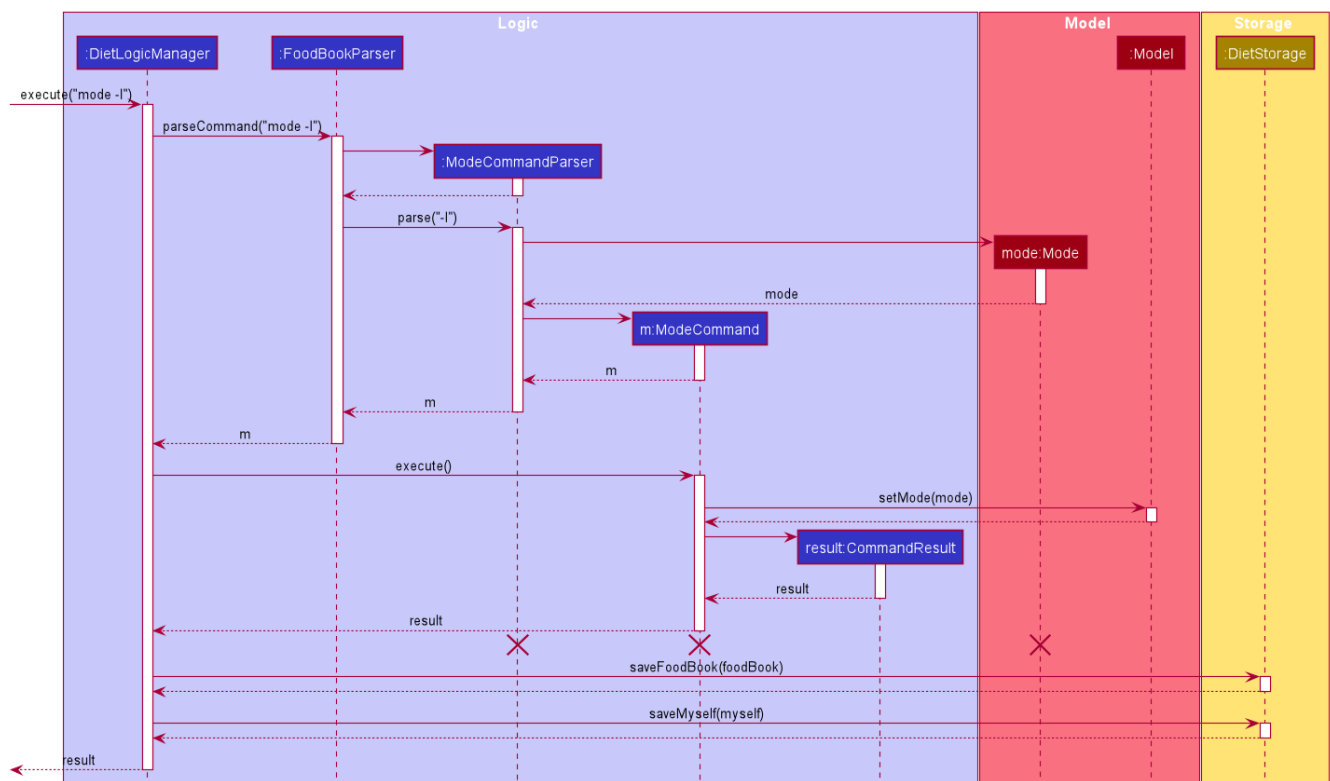


Figure 18. Mode Command Sequence Diagram

## Metrics Command

In this section, we will learn more about how the `metrics` command is implemented.

### What is the Metrics Command

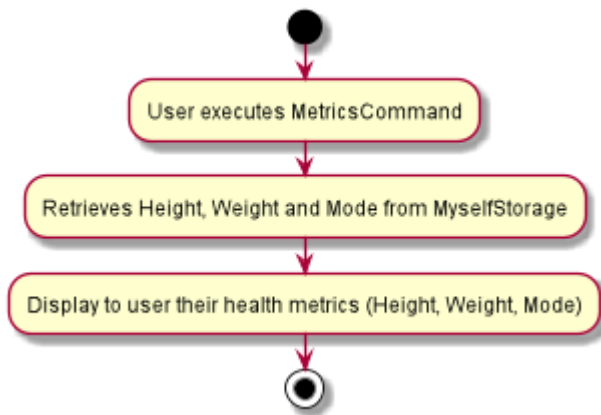
The `metrics` command allows the user to check their health metrics. These include their Height, Weight and Dieting Mode.

The `metrics` command was implemented as `MetricsCommand` in the `diettracker/logic/commands` package.

The `metrics` command has the following input format:

`metrics`

The following activity diagram illustrates what happens when a user executes the `metrics` command:



*Figure 19. Metrics Command Activity Diagram*

### Structure of Metrics Command

In this section, you will learn more about the relationships between objects related to the `metrics` command.

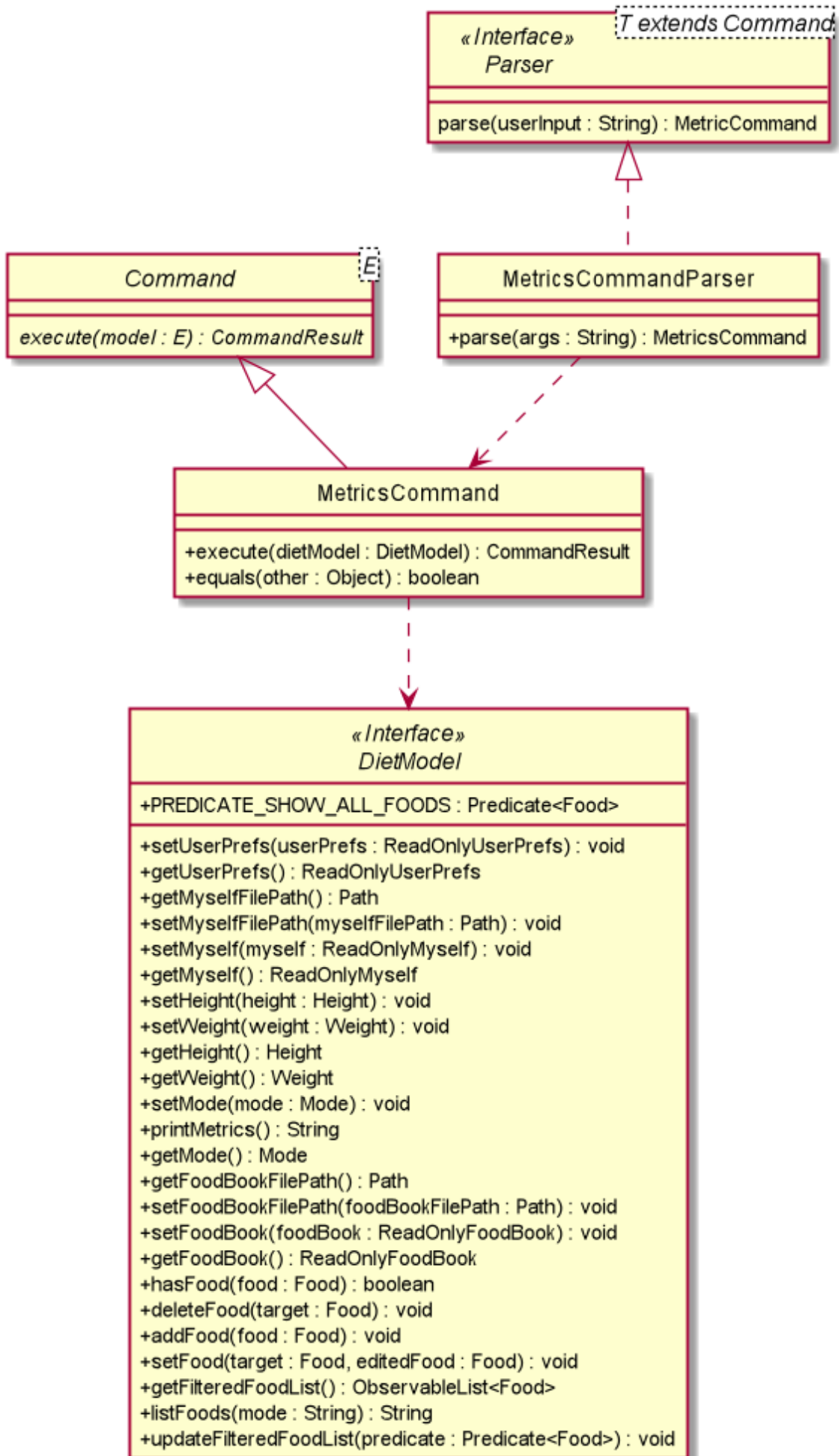


Figure 20. Metrics Command Class Diagram

The above class diagram shows the structure of the `MetricsCommand` and its associated classes and interfaces. Some methods and fields are left out because they are not of concern in `MetricsCommand`

## Implementation of Metrics Command

The following is a detailed explanation of the operations `MetricsCommand` performs.

1. The `MetricsCommand#execute(DietModel dietModel)` method is executed.
2. The `DietModel#printMetrics()` method would then be called to print the User's Metrics.
3. If successful, a success message will be generated by `CommandResult` and it will be returned with the generated success message. Otherwise, an error message showing the correct command syntax is thrown as `CommandException`.

## Sequence diagram for Metrics Command

The following sequence diagram summarizes what happens during the execution of `metrics` command.

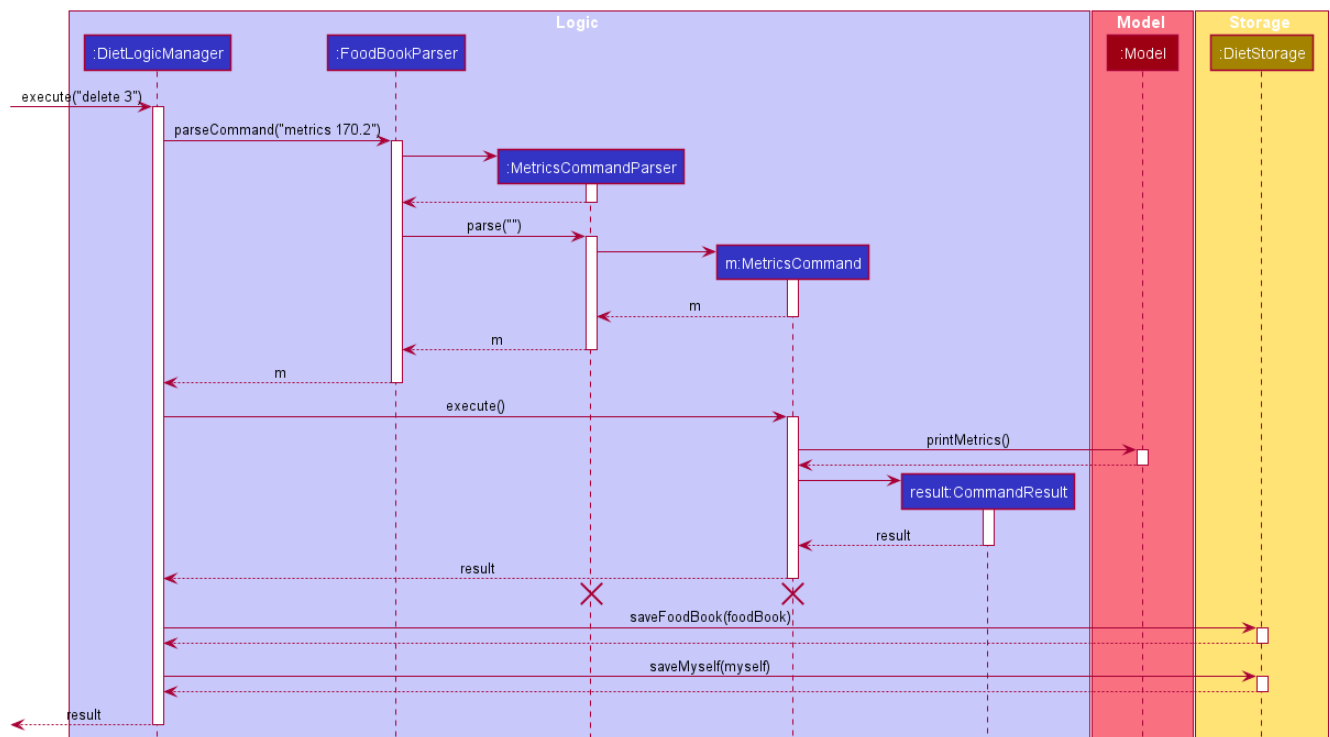


Figure 21. Metrics Command Sequence Diagram

# Testing

Refer to the guide [here](#).

# Dev Ops

Refer to the guide [here](#).

# Editing a Food in Diet Tracker

1. Editing a Food while all Foods are listed
  - a. Edit Command format: `edit -i INDEX [-n NAME] [-c CALORIES]`
  - b. Prerequisites: List all Foods using the `list` command. Multiple Foods in the list.
  - c. Test case: `edit -i 1 -n Pasta`  
Expected: First Food is edited. Details of the edited Food shown in the status message. Timestamp in the status bar is updated. Food is edited and then restored.
  - d. Test case: `edit -i 1`  
Expected: No Food is edited. Error details shown in the status message. Status bar remains the same.
  - e. Other incorrect delete commands to try: `edit`, `edit -i x` (where x is larger than the list size)  
Expected: Similar to previous.

# Calculate BMI In Diet Tracker

1. Calculating BMI for the User or for others.
  - a. Bmi Command format: `bmi [-h HEIGHT] [-w WEIGHT]`
  - b. Test case: `bmi -h 170.2 -w 65.7`  
Expected: BMI will be calculated with the Height of 170.2cm and Weight of 65.7kg, and will be printed and shown to the user. The BMI category will also be displayed to the User.
  - c. Test case: `bmi -h 170.2`  
Prerequisites: Must have stored Weight using `weight WEIGHT`. Expected: BMI will be calculated with the Height of 170.2cm and the User's stored Weight, and will be printed and shown to the user. The BMI category will also be displayed to the User.
  - d. Test case: `bmi -w 65.7`  
Prerequisites: Must have stored Weight using `height HEIGHT`. Expected: BMI will be calculated with the User's stored Height and the Weight of 65.7kg, and will be printed and shown to the user. The BMI category will also be displayed to the User.
  - e. Test case: `bmi`  
Prerequisites: Must have stored Height using `height HEIGHT` and Weight using `weight WEIGHT`. Expected: BMI will be calculated with the User's stored Height and Weight, and will be printed and shown to the user. The BMI category will also be displayed to the User.
  - f. Test case: `bmi -h 1000000000000000000 -w 1000000000000000000`  
Expected: Input height and weight are above the acceptable range. Error details shown in the response message.
  - g. Other incorrect edit commands to try: `bmi -h -1`, `bmi -h 137` (without storing Weight), `bmi -w 67` (without storing Height)  
Expected: BMI will not be calculated. Error Message will be shown with details.

## Store Height In Diet Tracker

1. Storing Height into the Diet Tracker.
  - a. Height Command format: `height HEIGHT`
  - b. Test case: `height 170.2`  
Expected: Stores a Height of 170.2cm into Diet Tracker. Details of the height stored are shown in success message.
  - c. Test case: `height -1`  
Expected: Height will not be stored. Error details shown in the response message.
  - d. Other incorrect height commands to try: `height 0`, `height 10000000000000001`  
Expected: Height will not be stored. Expected: Error message shown with details.

## Store Weight In Diet Tracker

1. Storing Weight into the Diet Tracker.
  - a. Height Command format: `weight WEIGHT`
  - b. Test case: `weight 65.7`  
Expected: Stores a Weight of 65.7kg into Diet Tracker. Details of the weight stored are shown in success message.
  - c. Test case: `weight -1`  
Expected: Weight will not be stored. Error details shown in the response message.
  - d. Other incorrect weight commands to try: `weight 0`, `weight 10000000000000001`  
Expected: Weight will not be stored. Error message shown with details.

## Set Dieting Mode In Diet Tracker

1. Setting a Dieting Mode for Diet Tracker.
  - a. Mode Command format: `mode [-l] [-g] [-m]`
  - b. Test case: `mode -l`  
Expected: Sets the Dieting Mode for the Diet Tracker and stores it.
  - c. Test case: `mode`  
Expected: No Dieting Mode will be set. Error details shown in the response message.

## Check User Metrics In Diet Tracker

1. Checking a Users own health metrics (Height, Weight and Dieting Mode).
  - a. Metrics Command format: `metrics`
  - b. Test case: `metrics`  
Expected: The User's own metrics will be printed out.
  - c. Test case: `metrics -h`  
Expected: No metrics will be shown. Error details shown in the response message.



---

Questions to ask:

1. Should users be able to use BMI without input, or even with just a single (Height/Weight) input?
2. What is different for Calories than other Classes that are stored as attributes?