Chen Shee Xiong - 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 enhancement**:
  1. **Integrating** 2 main features of EYLAH by common generic class. (Pull request #140, #232, #357 #367)
     - **What it does:**
       Ensure both of the features integrated in the main application.
     - **Justification:**
       The integration can help to reduce the duplicate code in the `EYLAH` which cased by the features done by different sub group.
     - **Highlights:**
       During the implementation, I faced a lot of difficulty as we working on the existing `AddressBook Level-3`, the code base given does not show how to integrate 2 different features. I have decided to modify some common use class to generic version in order can be implemented by both features.
     - **Credits:**
       N.A.
  2. Added the expense splitter storage component to allowed the application to automatically save the **Receipt** in local file. (Pull request #197)
     - **What it does:**
       Allow EYLAH to save the receipt details into a local files after execution of each command.
     - **Justification:**
       This will allow the user to be able to view the receipt details even they accidentally close the application or the application crashing. The receipt details will store in the local file after each command to ensure the updated data in the local storage.
     - **Highlights:**
       The structure of the receipt storage is much more complicated from the original

`AddressBook Level-3` storage. So when I implementing this storage, I am required to understand how the Json file store the value. The receipt storage has much more complicated structure compare to other storage structure. It contains nested json structure to store the different entry of item and person involved.

- **Credits:**
  N.A.

- **Minor enhancement**:

1. Create `ExitCommand` and `BackCommand` for the application (Pull request #357)
   - **What it does:**
     Allows user to go back to main menu when in either one of the feature mode and exit the application at any point of time.
   - **Justification:**
     The user now have the option to go back main menu if want to use another feature. They also able to exit the application through the command instead of close the app through the mouse curser to click the exit button.
   - **Highlights:**
     The `ExitCommand` in the main menu handle differently as it directly handle by `Eylah.java` instead of going through the `ModelManager#execute()` to create the `ExitCommand` object.
   - **Credits:**
     N.A.

- **Code contributed**:[Functional code and Test Code]
- **Other contributions**:
  - Project management:
    - In charge of project repository management and ensure the weekly task achievement.
    - Participated in planning the project timeline for EYLAH.
  - Enhancements to existing features:
    - Adapted the given Address Book Storage into EYLAH's Storage.
  - Documentation:
    - Updated Developer Guide for Done Receipt Command (Pull request #386, #388)
    - Updated Developer Guide for whole Design Section (Pull request #369, #380)
    - Updated User Guide for Back, Exit, Help and Done Receipt Command (Pull request #380, #386, #394)
  - Community:
    - PRs reviewed (with non-trivial review comments): (Pull requests: #306, #176, #177)
    - Contributed to forum discussions (examples: 1, 2)
    - Reported bugs and suggestions for other teams in the class (examples: https://github.com/sheexiong/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.

Display the available commands : `help`

In this section, you will learn more about `help` command, how to use it and the expected outcome after using the `help` command.

Summary of Help Command:
`help` You can use the `help` command, if you are uncertain, to see the list of available commands. This command can be used everywhere in the application.

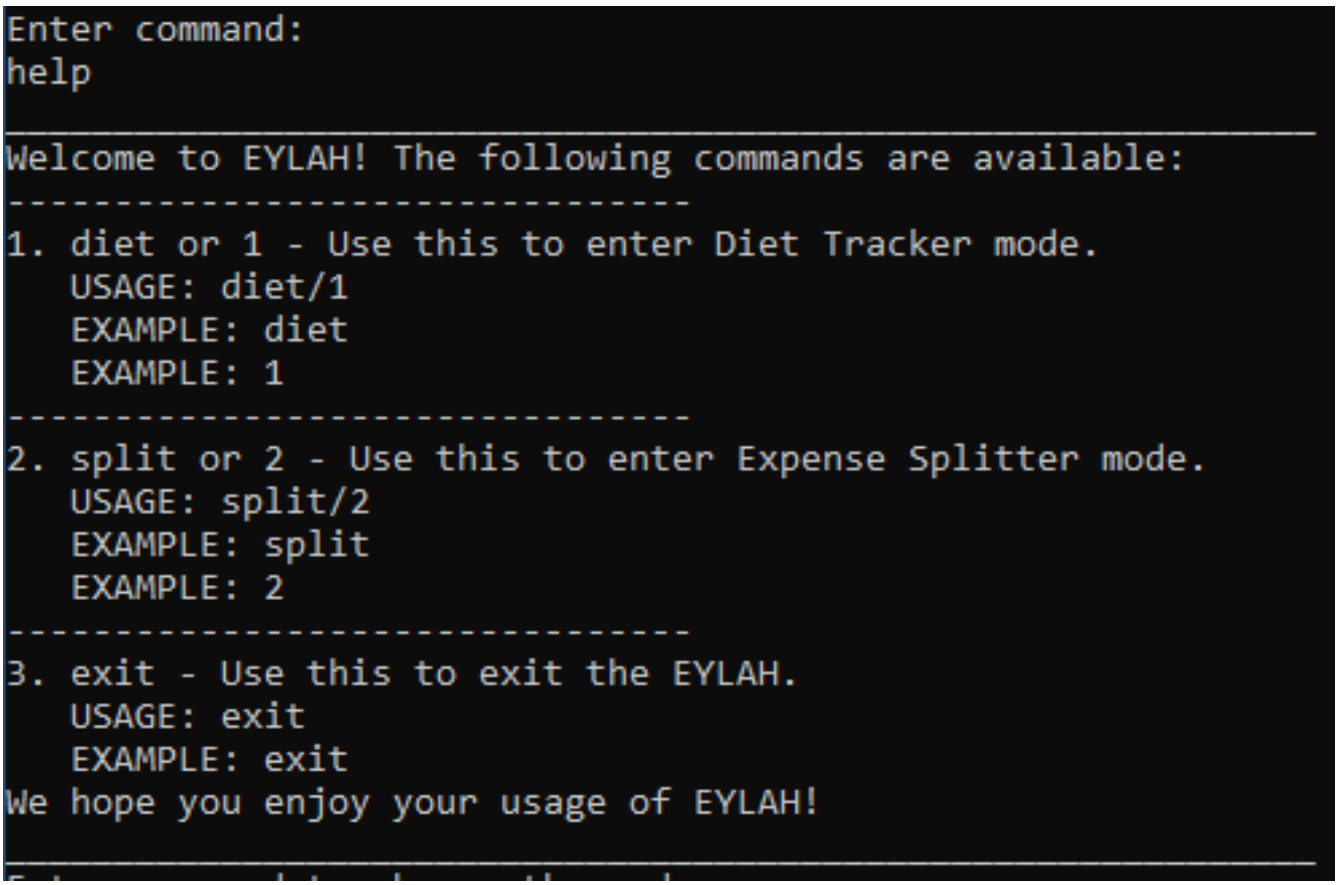
How to use the Help Command:

Format:
`help`

Valid Example:
`help`

Expected outcome:

Main Menu:



Diet Tracker Mode:

```
Currently at DIET
Enter command:
help
Welcome to Diet Tracker! The following commands are available:
--------------------------------
1. add - Use this to add a food item to your Diet Tracker.
   USAGE: add -n NAME -c CALORIES
   EXAMPLE: add -n Mushroom Aglio -c 360
--------------------------------
2. delete - Use this to delete a food item at the specified index in your list.
   USAGE: delete INDEX
    EXAMPLE: delete 1
--------------------------------
3. edit - Use this to edit a food item at the specified index in your list.
   USAGE: edit -i INDEX [-n NAME] [-c CALORIES]
   EXAMPLE: edit -i 1 -n Prawn Aglio -c 520
--------------------------------
4. list - Use this to list the food for a given period of time or food of a certain tag.
   USAGE: list [-a] [-d NUMDAYS] [-t TAG]
   EXAMPLE: list -a
--------------------------------
5. mode - Use this to change the mode (GAIN, MAINTAIN, LOSS) of the Diet Tracker.
   USAGE: mode [-g] [-l] [-m]
   EXAMPLE: mode -g
--------------------------------
6. height - Use this to add your personal height to the Diet Tracker.
   USAGE: height HEIGHT
   EXAMPLE: height 170.2
--------------------------------
7. weight - Use this to add your personal weight to the Diet Tracker.
   USAGE: weight WEIGHT
   EXAMPLE: weight 65.7
--------------------------------
8. bmi - Use this to calculate your bmi.
   USAGE: bmi [-h HEIGHT] [-w WEIGHT]
   EXAMPLE: bmi -h 170.2 -w 65.7
--------------------------------
9. back - Use this to go back to EYLAH main application window in order to switch between Diet Tr
acker and Expense Splitter.
   USAGE: back
   EXAMPLE: back
We hope you enjoy your usage of EYLAH Diet Tracker!
_________________________________________________________________
```

Expense Splitter Mode:

```
Currently at SPLITTER
Enter command:
help
Welcome to Expense Splitter! The following commands are available:
------------------------------
1. additem - Use this to add an item, price and the person(s) involved in the splitting for that
item.
   USAGE: additem -i ITEMNAME -p PRICE -n NAME [-n NAME]...
   EXAMPLE: additem -i pasta -p 19.90 -n alice -n bob -n charlie
------------------------------
2. deleteitem - Use this to delete the item at the specified index in your receipt.
   USAGE: deleteitem INDEX
   EXAMPLE: deleteitem 1
------------------------------
3. listreceipt - Use this to list all the item(s) in the receipt.
   USAGE: listreceipt
   EXAMPLE: listreceipt
------------------------------
4. listamount - Use this to list all the person(s) and the amount they owe.
   USAGE: listamount
   EXAMPLE: listamount
------------------------------
5. donereceipt - Use this to mark the receipt as done after adding all items.
   USAGE: donereceipt
   EXAMPLE: donereceipt
------------------------------
6. paid - Use this to subtract the amount paid by a person at the specified index.
   USAGE: paid INDEX [AMOUNT]
   EXAMPLE: paid 2 3.90
------------------------------
7. clearreceipt - Use this to clear the receipt and start a new receipt.
   USAGE: clearreceipt
   EXAMPLE: clearreceipt
------------------------------
8. back - Use this to back to the main menu of EYLAH.
   USAGE: back
   EXAMPLE: back
------------------------------
We hope you enjoy your usage of EYLAH Expense Splitter!
```

Additional notes and tips

NOTE	Help in main menu and different mode will give different help information.

Mark receipt as done `donereceipt`

In this section, you will learn more about the `donereceipt` command, how to use it and the expected outcome after using the `donereceipt` command.

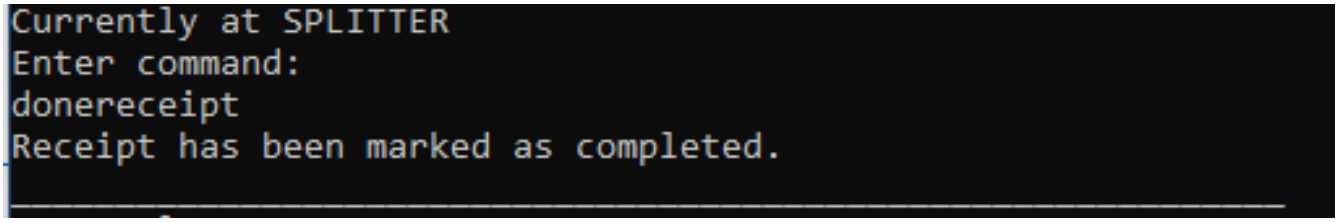
Summary of Done Receipt Command: `donereceipt` *marks the receipt as done when you have completed entering all the items.

How to use the Done Receipt Command:

Format:
`donereceipt`

Example:
`donereceipt`

Expected outcome:



NOTE

- Use this command only after all Items have been correctly added to the Receipt.
- After you use this command, you will be unable to add any new items using the `additem` command or delete any items using the `deleteitem` command.
- However, you are now able to use the `paid` command.

Exiting Expense Splitter to go back to main menu `back`

In this section, you will learn more about the `back` command, how to use it and the expected outcome after using the `back` command.

Summary of Back Command:
`back` allows you to exits Expense Splitter mode to go back to main menu of the application.

How to use the Back Command:

Format:
`back`

Valid Example:
`back`

Expected outcome:

Exiting the Application : `exit`

In this section, you will learn more about `exit` command, how to use it and the expected outcome after using the `exit` command.

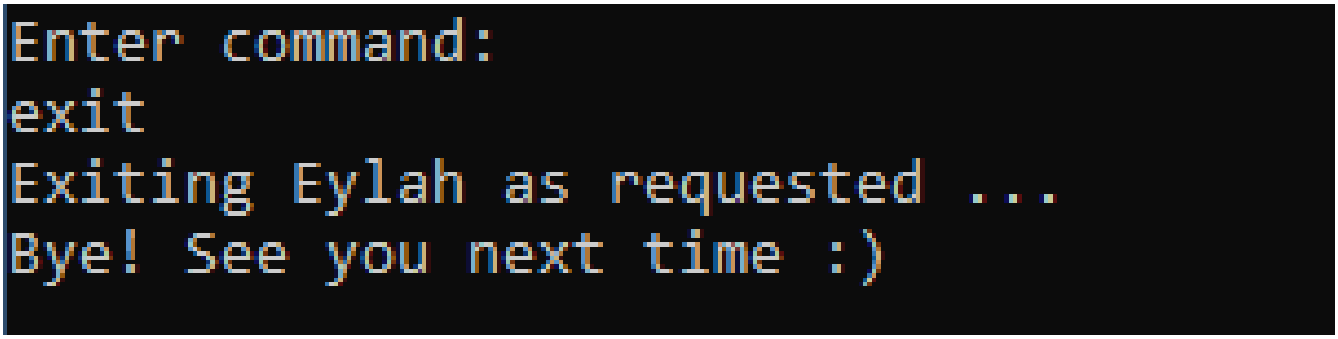
Summary of Exit Command:
`exit` You can use this command everywhere in the application to exit.

How to use the Exit Command:

Format:
`exit`

Valid Example:
`exit`

Expected outcome:



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.

Architecture

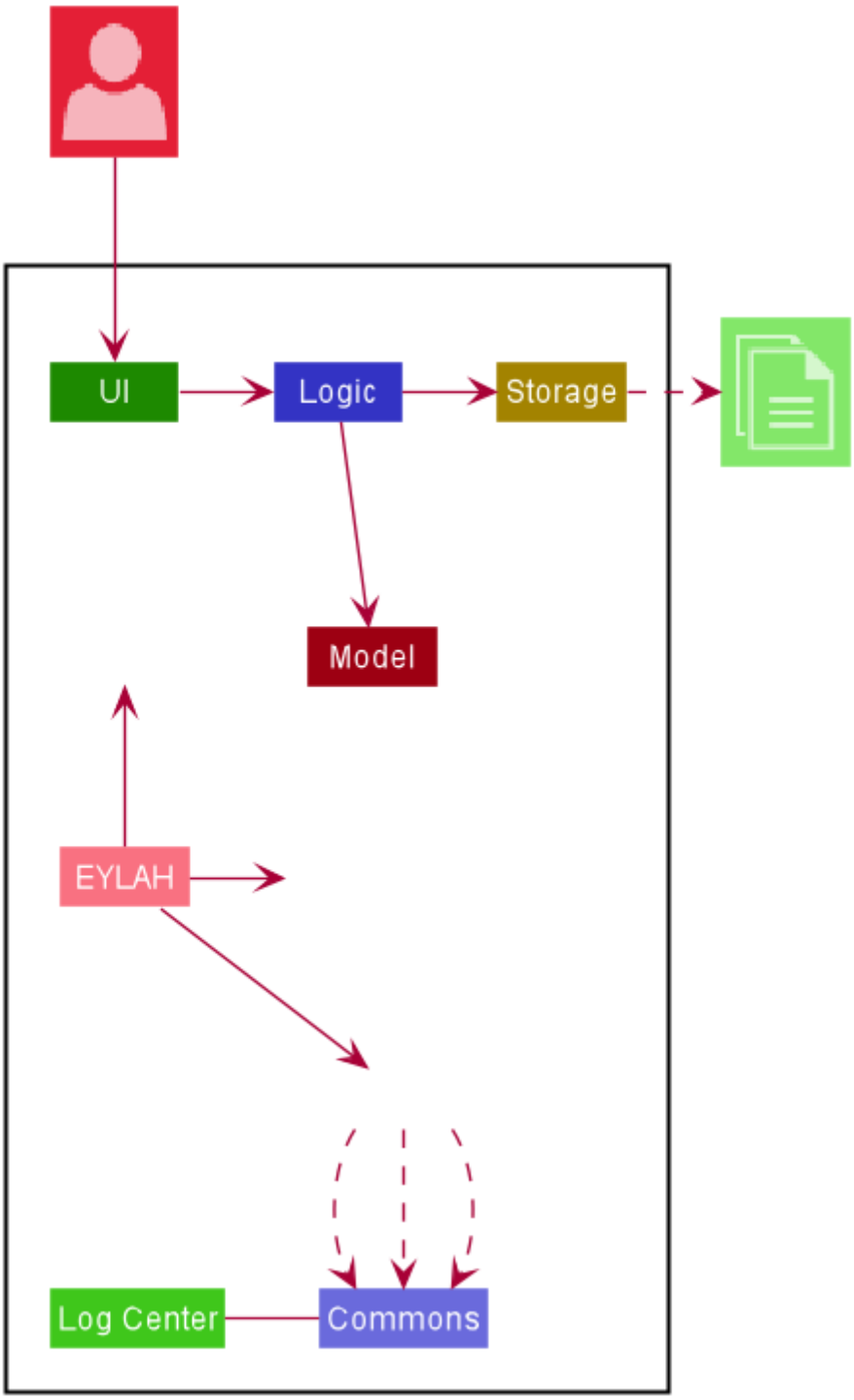


Figure 1. Architecture Diagram

The ***Architecture Diagram*** given above explains the high-level design of the application. Given below is a quick overview of each component.

TIP	The `.puml` files used to create diagrams in this document can be found in the diagrams folder. Refer to the Using PlantUML guide to learn how to create and edit diagrams.

`EYLAH` contains the main class for the application. It is responsible for,

- At app launch: Initializes the components in the correct sequence, and connects them up with each other.

- At shut down: Shuts down the components and invokes cleanup method where necessary.

`Commons` represents a collection of classes used by multiple other components. The following class plays an important role at the architecture level:

- `LogsCenter` : Used by many classes to write log messages to the App's log file.

The rest of the App consists of four components.

- `UI`: The UI of the App.
- `Logic`: The command executor.
- `Model`: Holds the data of the App in-memory.
- `Storage`: Reads data from, and writes data to, the hard disk.

Each of the four components

- Defines its *API* in an `interface` with the same name as the Component.
- Exposes its functionality using a `{Component Name}Manager` class.

For example, the `Logic` component (see the class diagram given below) defines it's API in the `Logic.java` interface and `MODELogic.java` interface and exposes its functionality using the `MODELogicManager.java` class.

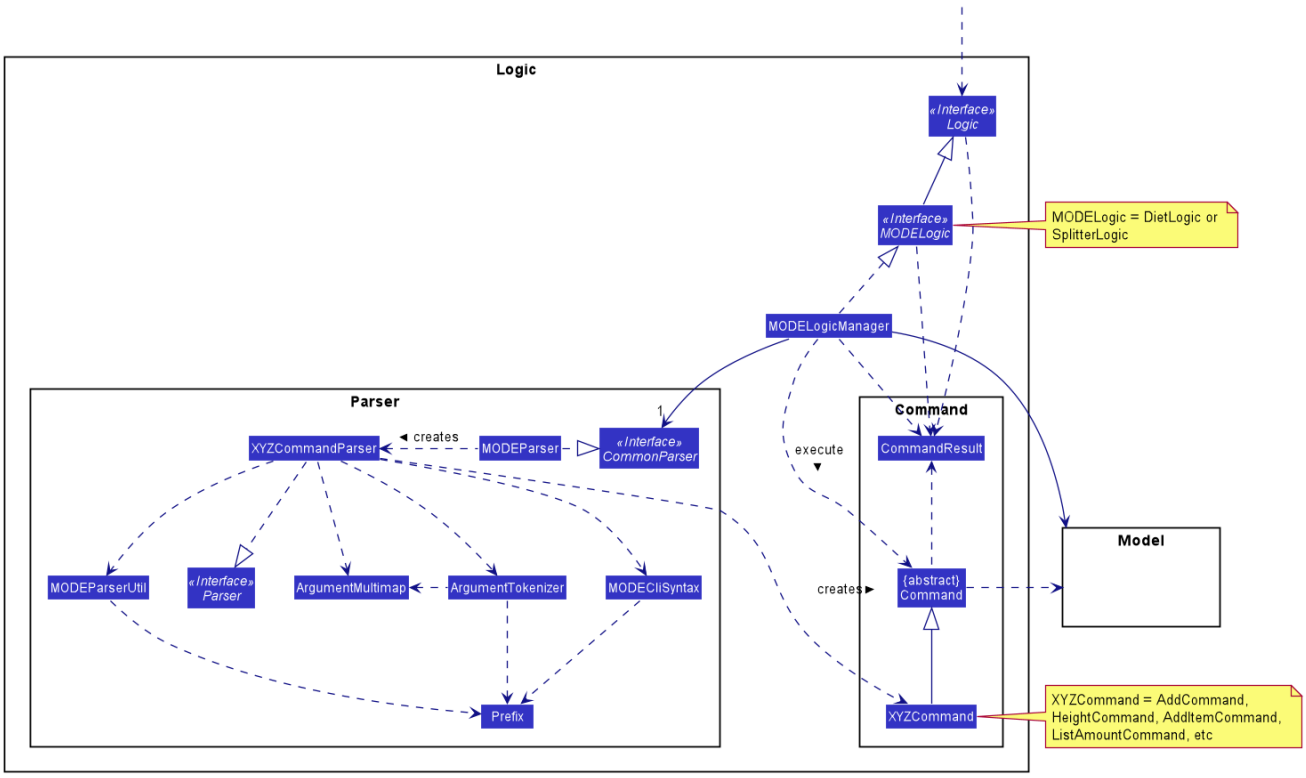


Figure 2. Class Diagram of the Logic Component

How the architecture components interact with each other

The *Sequence Diagram* below shows how the components interact with each other for the scenario where the user issues the command `deleteitem 1`.

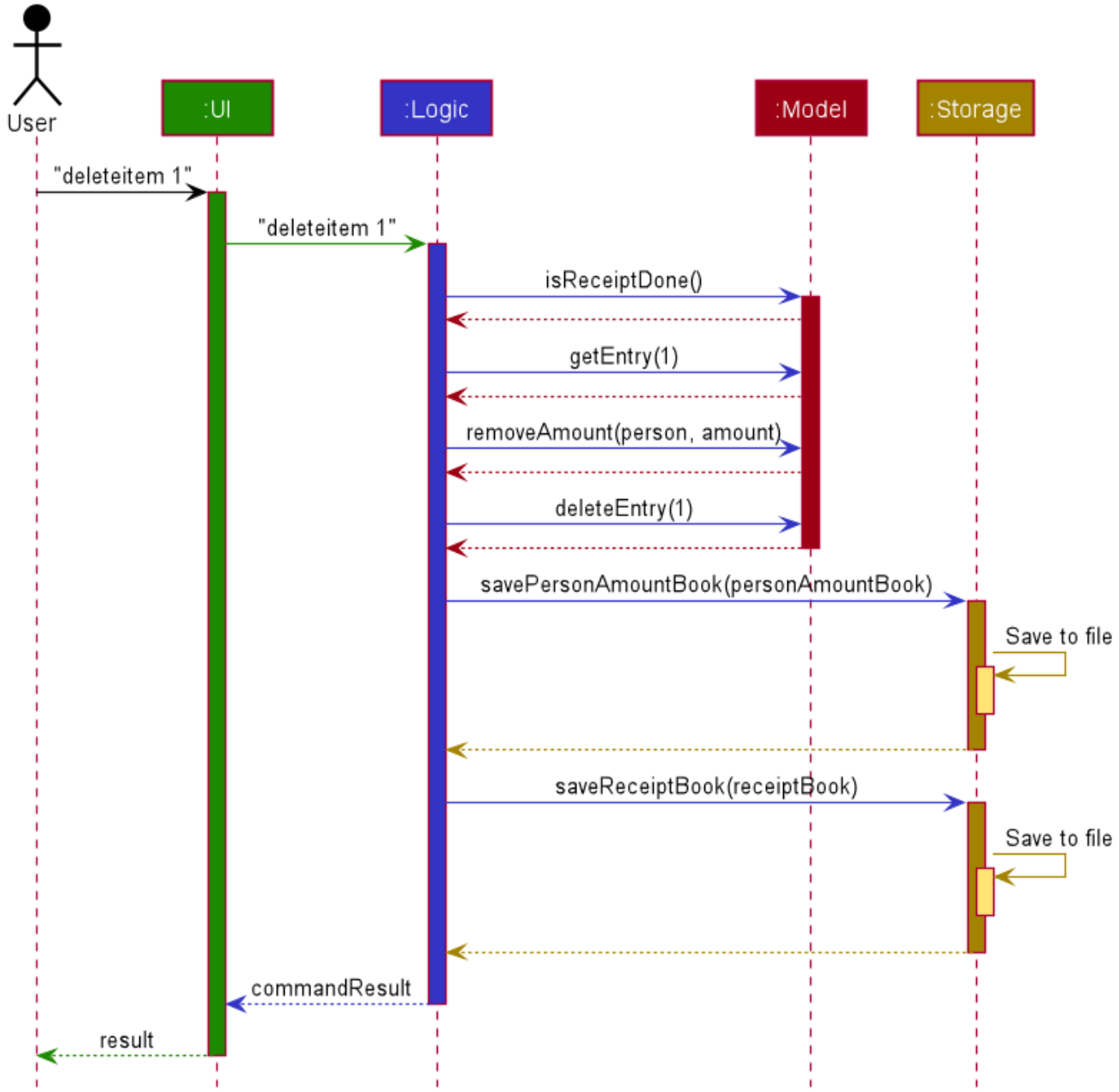


Figure 3. Component interactions for `deleteitem 1` command

UI component

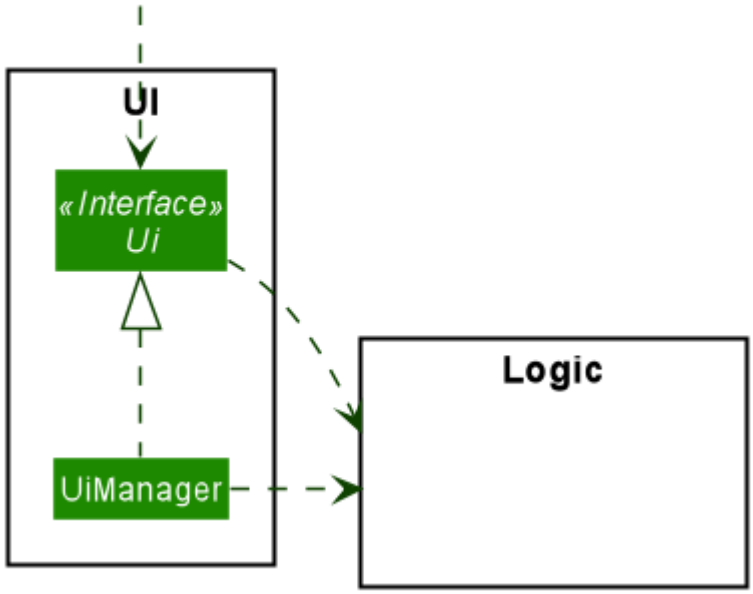


Figure 4. Structure of the UI Component

API : `Ui.java`

Th `UI Component` mainly deals with interactions with the user. It also plays a part in the initialisation of the program printing the logo, welcome message and main menu page to user. This component only has 2 classes, `Ui.java` and `UiManager.java`.

The `UI` component,

- Reading the user input.
- Displaying the result messages to the user.

Logic component

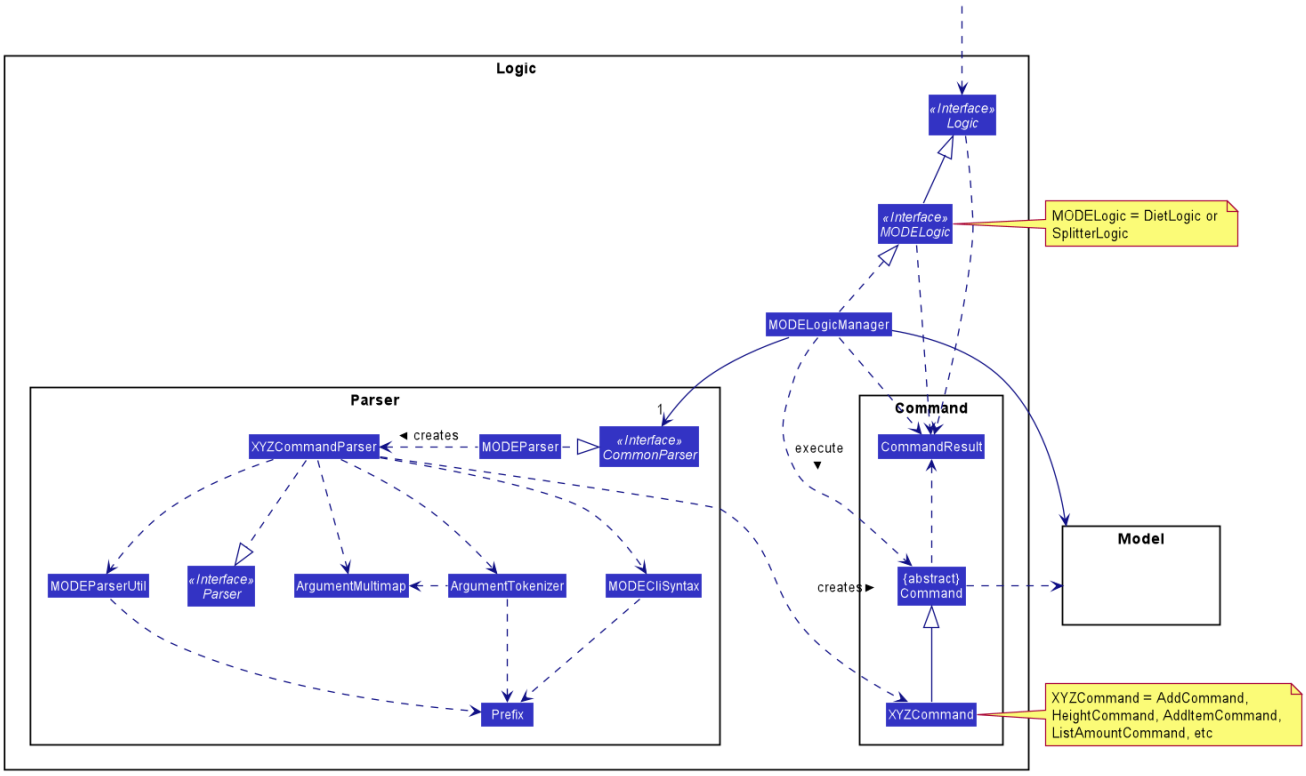


Figure 5. Structure of the Logic Component

API : `Logic.java` `DietLogic.java` `SplitterLogic.java`

The `Logic Component` deals with the logic flows of the App. In each feature mode, the components to deal with the logic flow are different. In `Diet Tracker` mode, `DietLogic.java` and `DietLogicManager.java` are used to handle the logic operation of the APP. In `Expense Splitter` mode, `SplitterLogic.java` and `SplitterLogicManager.java` are used to handle the logic operation.

NOTE

- `MODE` used in the given subsection refers to `Diet` when in `Diet Tracker` mode, `Splitter` when in `Expense Splitter` mode. For example, `MODELogic` given below refers to `SplitterLogic` when in `Expense Splitter` mode.
- `MODEParser` is an exception. When in `Expense Splitter` mode it refers to `ExpenseSplitterParser` while in `Diet Tracker` mode it refers to `FoodBookParser`.

1. `MODELogic` uses the `MODEParser` class to parse the user command.
2. This results in a `Command` object which is executed by the `MODELogicManager`.
3. The command execution can affect the `Model` (e.g. deleting AN entry).
4. The result of the command execution is encapsulated as a `CommandResult` object which is passed back to the `Ui`.
5. In addition, the `CommandResult` object can also instruct the `Eylah.java` application to perform certain actions, such as to go back to main menu or exit the App.

Given below is the Sequence Diagram for interactions within the `Logic Component` for the `execute("deleteitem 1")` API call.

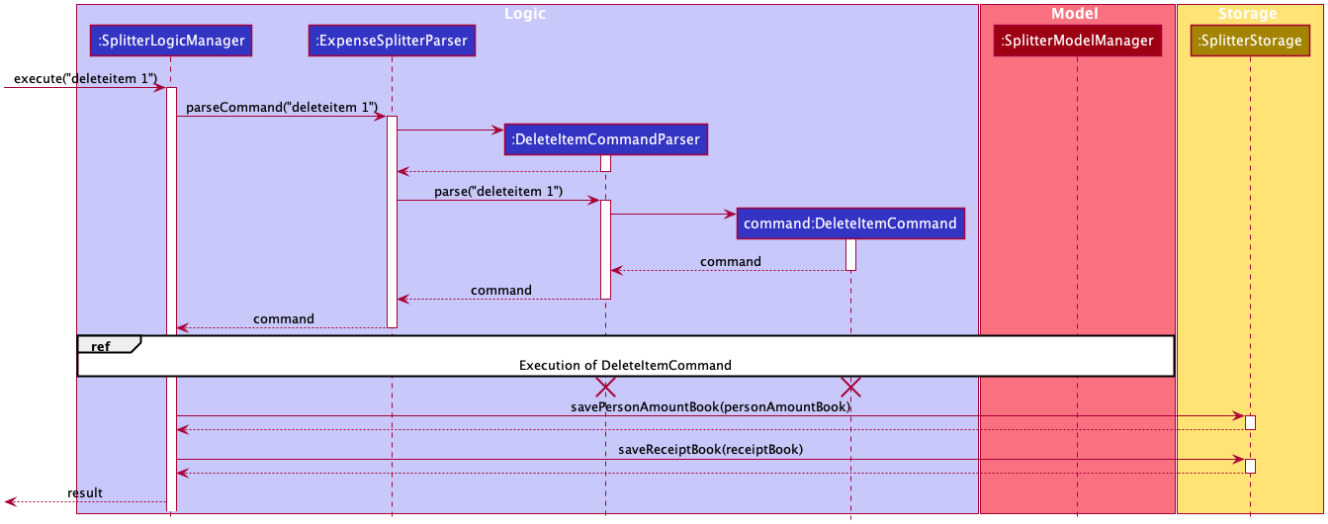


Figure 6. Interactions Inside the Logic Component for the `deleteitem 1` Command in Expense Splitter mode

NOTE	The lifeline for `DeleteItemCommandParser` should end at the destroy marker (X) but due to a limitation of PlantUML, the lifeline reaches the end of diagram.

Model component

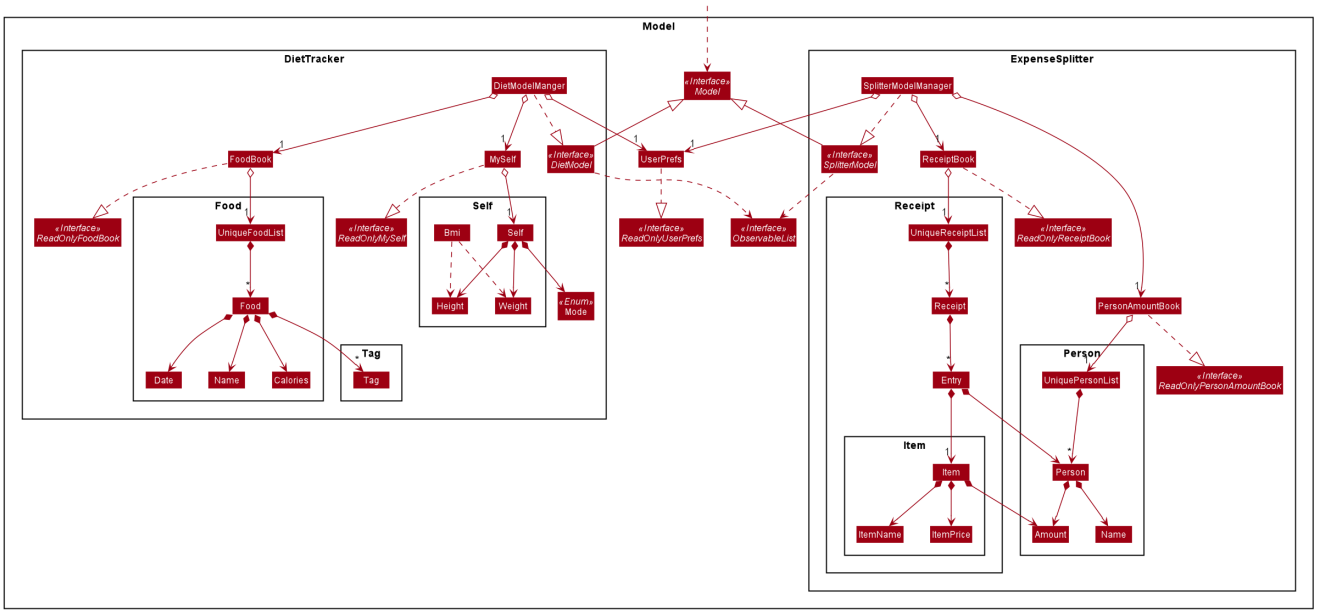


Figure 7. Structure of the Model Component

API : `Model.java` `SplitterModel.java` `DietModel.java`

The `Model Component` deals with the modeling of the object.

NOTE	`MODE` used in the given subsection refers to `Diet` when in `Diet Tracker` mode, `Splitter` when in `Expense Splitter` mode. For example, `MODEModel` given below refers to `SplitterModel` when in `Expense Splitter` mode.

The `MODEModel`,

- stores a `UserPref` object that represents the user's preferences.

- stores the PersonAmountBook and ReceiptBook data in `Expense Splitter` mode.
- stores the FoodBook and Myself data in `Diet Tracker` mode.
- does not depend on any of the other three components.

Storage component

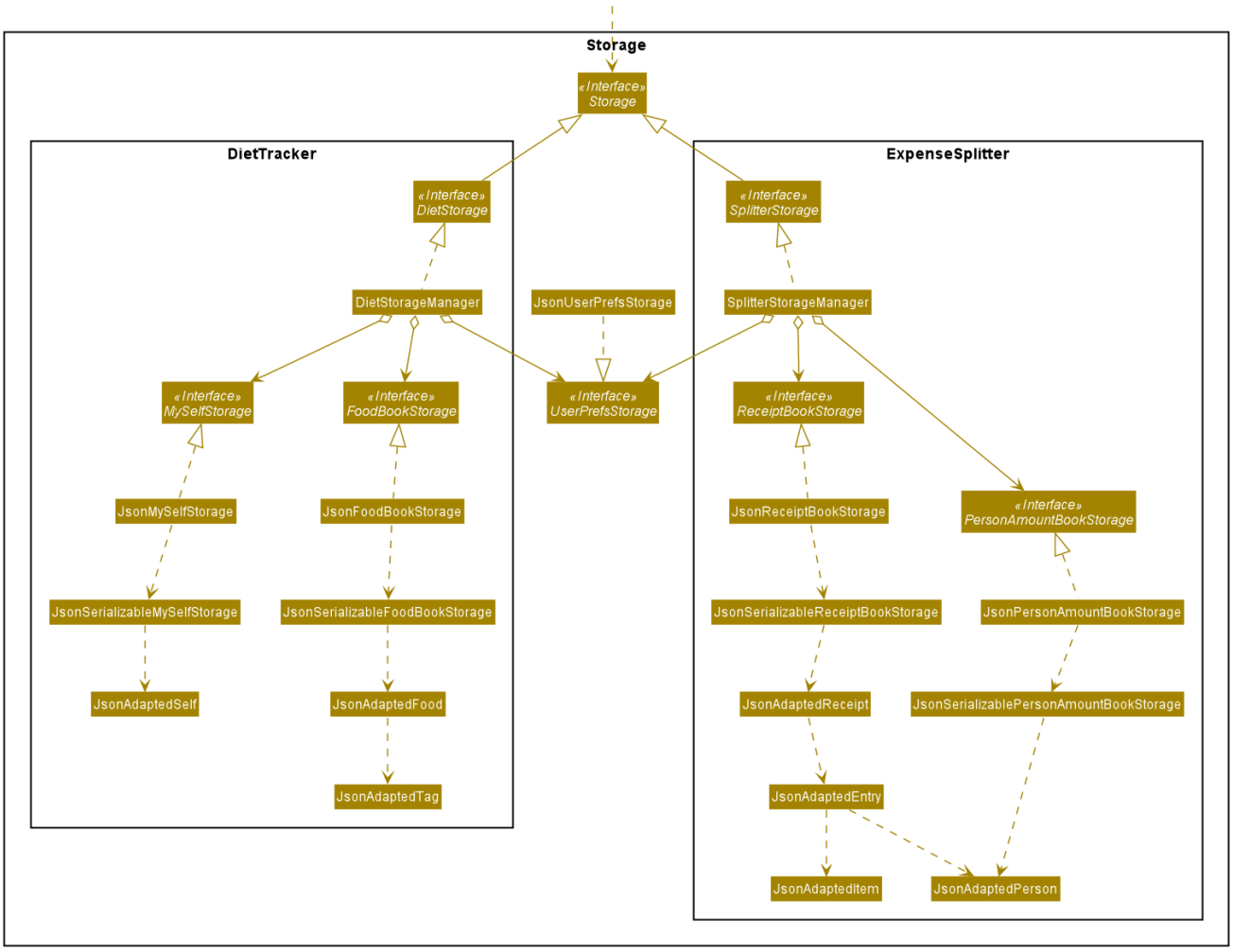


Figure 8. Structure of the Storage Component

API : `Storage.java` `SplitterStorage.java` `DietStorage.java`

The `Storage Component` deals with the operations to write and read from the local files.

NOTE	`MODE` used in the given subsection refers to `Diet` when in `Diet Tracker` mode, `Splitter` when in `Expense Splitter` mode.For example, `MODEStorage` given below refers to `SplitterStorage` when in `Expense Splitter` mode.

The `MODEStorage` component,

- can save `UserPref` objects in json format and read it back.
- can save the PersonAmountBook and ReceiptBook data in json format and read it back in `Expense Splitter` mode.

- can save the FoodBook and MySelf data in json format and read it back in `Diet Tracker` mode.

Common classes

Classes used by multiple components are in the `seedu.eylah.commons` package.

Done Receipt Command

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

What is the Done Receipt Command

The `donereceipt` command allows user to finalize the entries in the receipt. After this command is executed, the entries in the receipt are immutable.

The `donereceipt` command was implemented as a `DoneReceiptCommand` in the `expensespliter/logic` package.

The `donereceipt` has the following input format:

`donereceipt`

NOTE	The receipt will not be editable after this command is executed thus ensuring all entries are inputted correctly. The User can use `listreceipt` to check current entries in the receipt, `deleteitem` and `additem` to delete and add the correct item back.

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

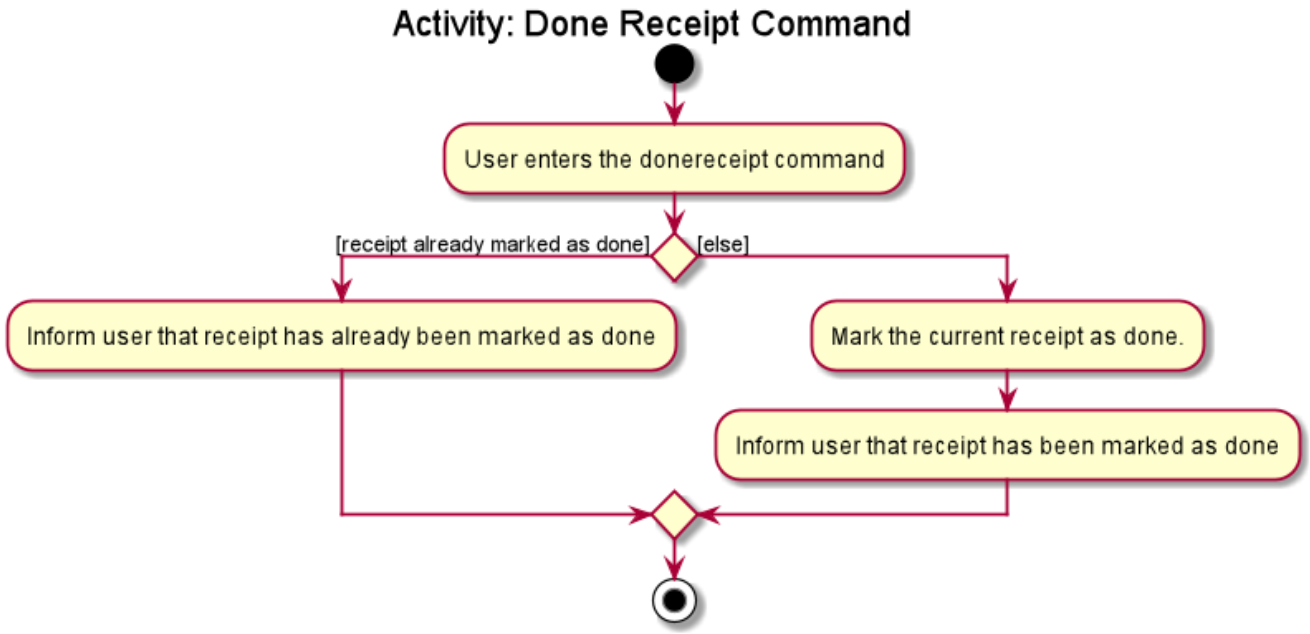


Figure 9. Done Receipt Command Activity Diagram

Structure of Done Receipt Command

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

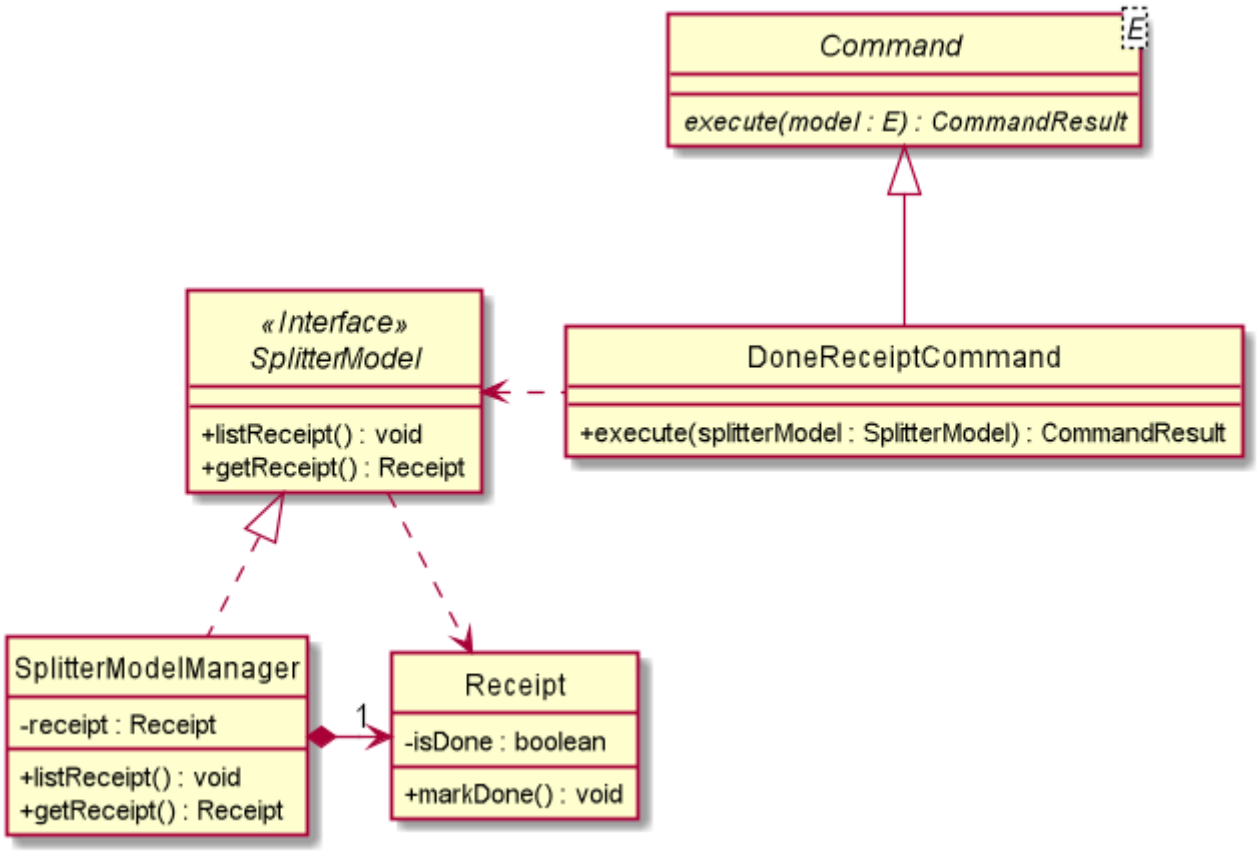


Figure 10. Done Receipt Command Class Diagram

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

Implementation of Done Receipt Command

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

1. The `DoneReceiptCommand#execute(SplitterModel splitterModel)` method is executed.
2. The `SplitterModel#getReceipt()` method is executed and get the current `Receipt`.
3. Then `Receipt#markDone()` method is called.
4. This will invoke the boolean `Receipt#isDone` variable changed to true.

Sequence Diagram for Done Receipt Command

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

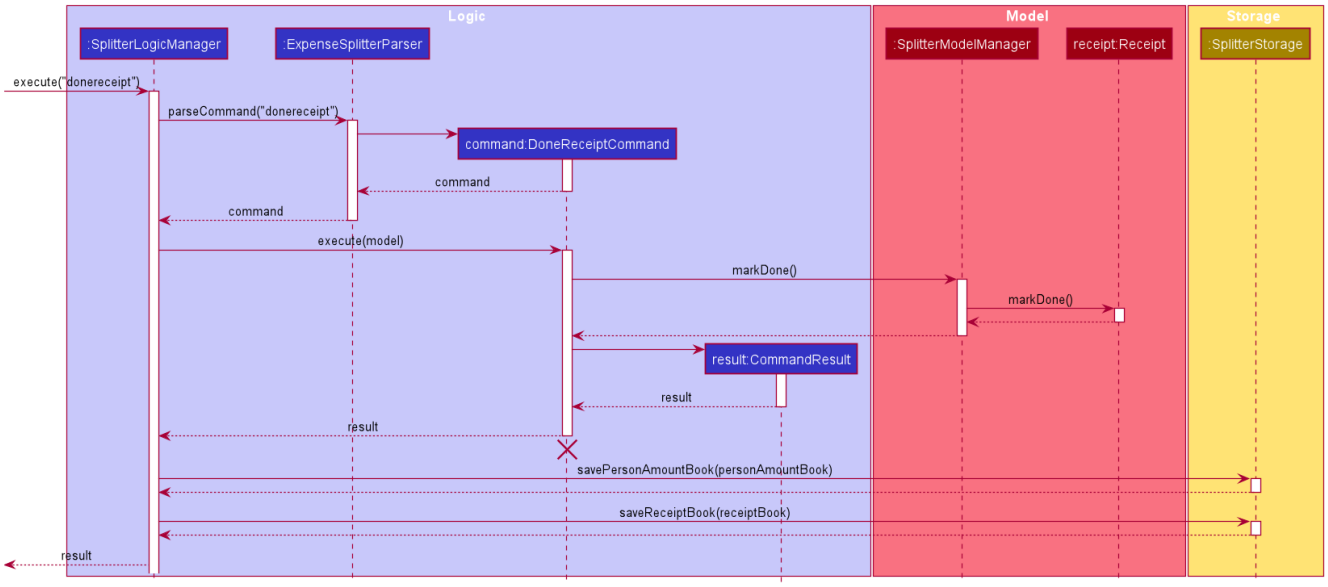


Figure 11. Done Receipt Command Sequence Diagram