Goh Si Ning - Project Portfolio

About Jarvis

Jarvis is the product of the team project by my team of 4 software engineering students and I under the School of Computing, National University of Singapore module CS2103T Software Engineering. We were tasked with enhancing a basic command line interface (CLI) desktop address-book application, and we decided to morph it into an all encompassing student life organiser which enables students to track and manage their tasks, finances, co-curricular activities (CCAs) and courses taken at NUS.

As students, we often need to access many different applications daily in order to manage individual aspects of our life. Our team hopes that Jarvis will make things easier for students by providing a platform that consolidates the most important features. Therefore, Jarvis has 4 separate features for each concern faced by students: Planner, Finance Tracker, Cca Tracker, and Course Planner.

About Finance Tracker

For this project, I was in charge of implementing the Finance Tracker feature, which keeps track of the user's purchases and monthly subscriptions (aka installments), as well as total monthly expenditure and remaining budget if the user has added a monthly spending limit for himself.

Shown below is the graphical user interface (GUI) of Jarvis. Our application consists of 4 different tabs - one for each of the above-mentioned features. The application is currently at the Finance Tracker tab, which is underlined in red. When the user enters a command in the command box, Jarvis will return a response (whether his command was successfully executed, or if not, why it was rejected) in the result box.

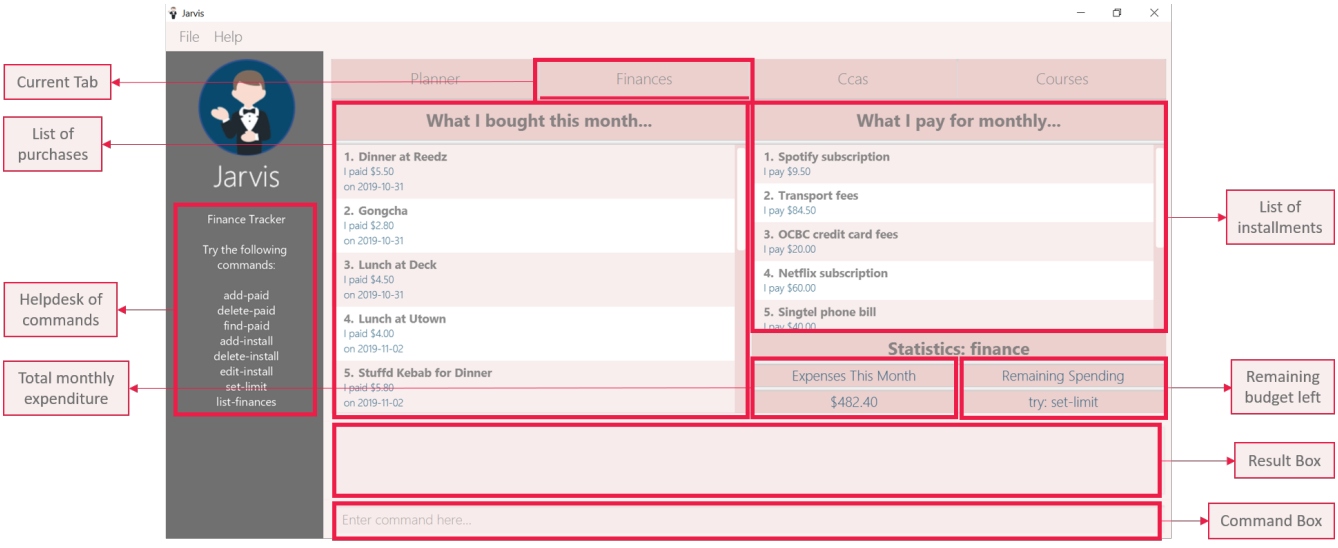

Figure 1. The graphical user interface for Jarvis under the Finance Tracker tab

In addition to developing both the front-end and back-end of the Finance Tracker feature, I also contributed to adopting the original version of the developer's guide as well as most of the

diagrams in the final developer guide. The following sections will illustrate these enhancements in more details as well as the relevant documentation in the user and developer guides in relation to these enhancements.

Summary of Contributions

This section shows a summary of my coding, documentation, and other helpful contributions to the team project.

Enhancement Added - Finance Tracker

Description of enhancement

As mentioned above, I was in charge of the Finance Tracker feature. This allowed the users to be able to track their purchases and monthly installments, as well as track the amount that they are spending with respect to the limit they have set for themselves.

Users are able to add, delete, and find purchases as well as add, delete, and edit installments. In addition, they can set a monthly spending limit for themselves. Our application will then keep track of their total spending and send warnings when their spending limit is close to being reached or has been exceeded.

Justification

As university students, we are expected to take care of more finances such as monthly phone bills and credit card bills. Furthermore, we have a lot of daily purchases that we might easily lose track of. Thus, the Finance Tracker feature will allow the user to better manage his monthly spendings and ensure he does not overspend unknowingly.

For his convenience, the user does not need to add the date of purchase as Jarvis will automatically tag today's date to the user's purchase. Each installment subscription fee is also only added to the total monthly expenditure once.

Code contributed

Please click to see my code contribution on Reposense as well as my pull requests on GitHub.

Documentation

All sections related to the Finance Tracker in the user guide and developer guide were done by me. They will be elaborated in the later sections, namely Contributions to User Guide and Contributions to Developer Guide.

Other Contributions

Enhancements to existing features

Due to extensive testing on the Finance Tracker feature, I was able to contribute to increasing the overall test coverage from 71.46% to 75.54%. This can be viewed from pull request #137.

In addition, I revised the overall user interface to change the order of the tabs. This increased the cohesiveness of the application from the perspective of a student using Jarvis, according to the frequency that they would access each feature. This can be viewed from pull request #362.

Community

I have also reviewed several pull requests from my team mates (with non-trivial review comments), as can be seen from pull requests #104, #136, and #361.

In addition, my code for the Finance Tracker feature I implemented was taken by my team mate to adopt for his Cca Tracker feature. This was due to the extensive OOP I had practiced throughout my code, which he also thought was a good design to follow. This can be seen from pull request #107.

Lastly, I reported bugs and offered helpful suggestions for another application created by my fellow cohort mates. This can be seen from the following examples in this link.

Documentation

This section will summarise my contribution to the overall set of documentation for the team project, excluding the Finance Tracker feature, which would be elaborated from here.

User Guide

I made changes to the existing user guide to standardise how each section was displayed in terms of the language used in explanation of the commands and the format of example commands given. This greatly improved the cohesiveness of the user guide and made it more reader-friendly. This can be viewed from pull request #176.

Developer Guide

Most of the initial version of the developer guide adopted from the previous application was done by me, as can be seen from pull request #14. This includes adding the use cases, non-functional requirements as well as glossary.

In the latest version of the developer guide, I was responsible for most of the class, activity and sequence diagrams for the entire team. While there were existing diagrams made by my team mate for a previous version of the developer guide, these diagrams needed to be changed as:

1. They did not strictly follow the guidelines of the project, or
2. They did not reflect the updated implementation of some features made by the team in the subsequent few weeks.

In addition, I repackaged the diagrams for easier perusal by my team mates for their documentation. The above-mentioned can be viewed from the pull requests #368, #371, and #373.

The following are examples of diagrams that I created for my fellow team mates using plantUML.

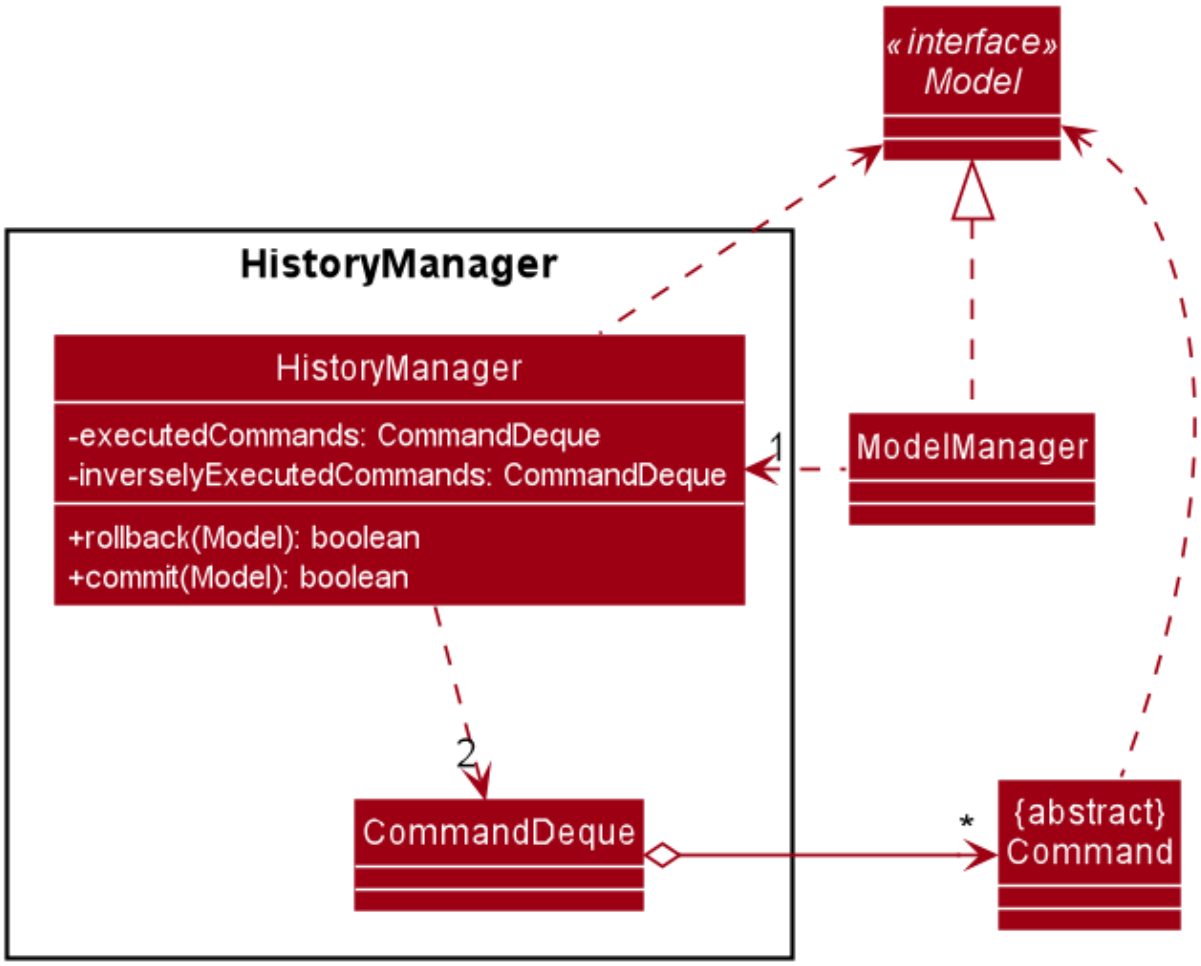

Figure 2. HistoryManagerModelClassDiagram for the Undo-Redo feature

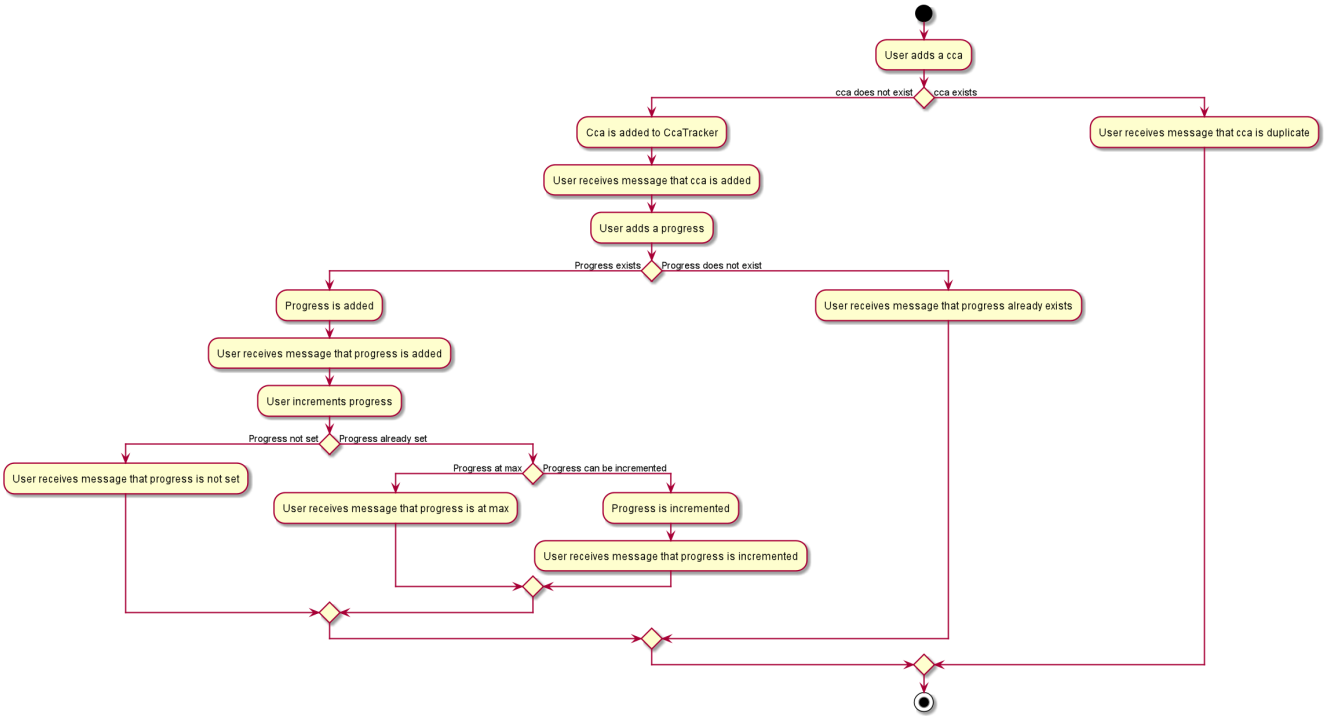

Figure 3. IncreaseCcaProgressActivityDiagram for the Cca Tracker feature

Contributions to User Guide

As we morphed the existing address-book application, we had to update the original user guide to reflect the changes in features implemented. My team decided to have each member responsible for their feature to write their own documentation in the user guide. Thus, the Finance Tracker section was done mostly by me.

The section will contain excerpts from our Jarvis User Guide, showing additions that I have made with my Finance Tracker feature. The command that I would be illustrating is the `find-paid` command.

Search for a purchase with keyword(s)

This command searches through the existing purchases stored in Jarvis and displays to the user all purchases which have descriptions matching the keyword input by the user.

The following contains a modified excerpt from our user guide:

To have a quick view to see what you have been eating for dinner over the last month, you can used `find-paid` to pull up purchases with descriptions matching `KEYWORD` provided.

Format: `find-paid KEYWORD`

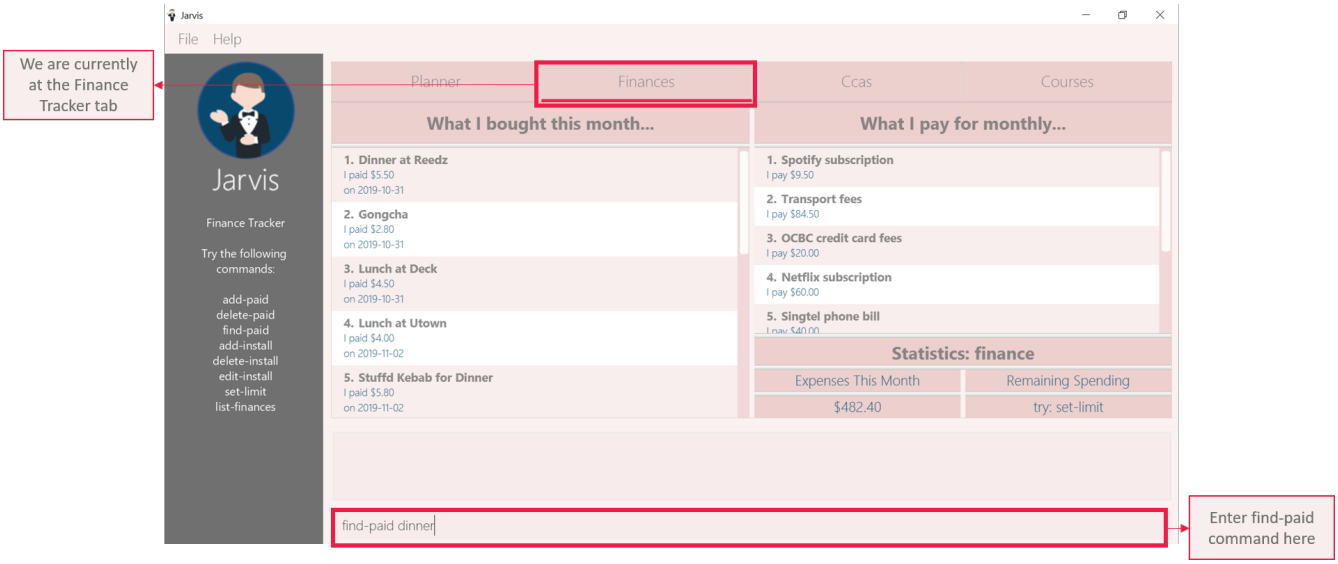

Figure 4. Step 1: Enter `find-paid` command in command box

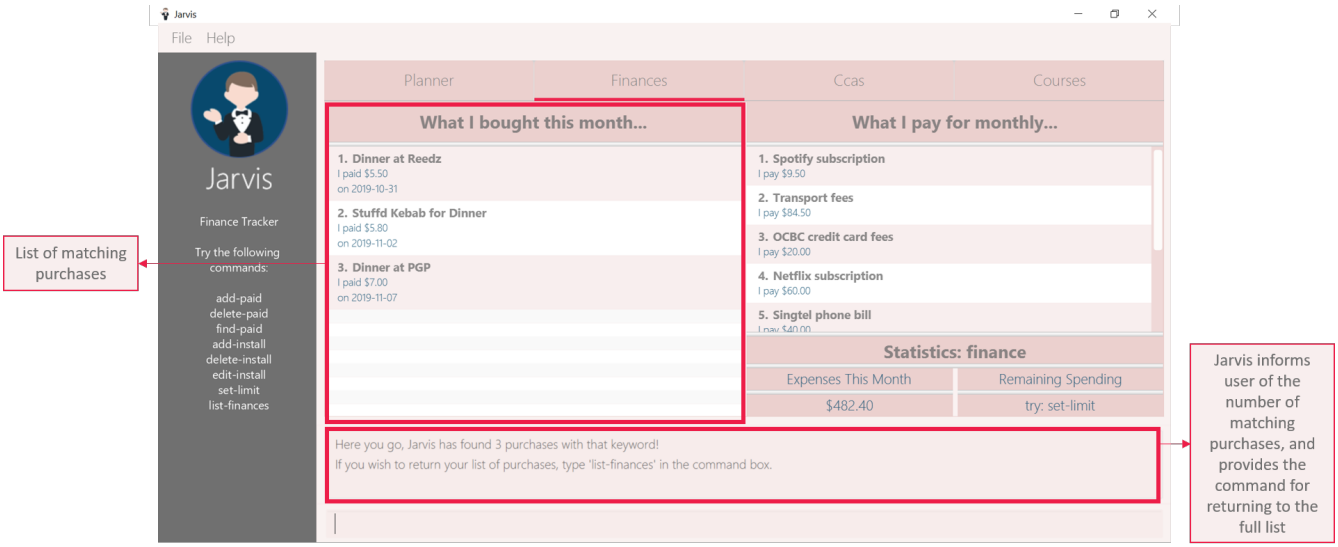

Figure 5. Step 2: Jarvis displays all matching purchases to keyword

NOTE If there are no existing purchases in the application that matches the keyword given by the user, the result box will display the following message:

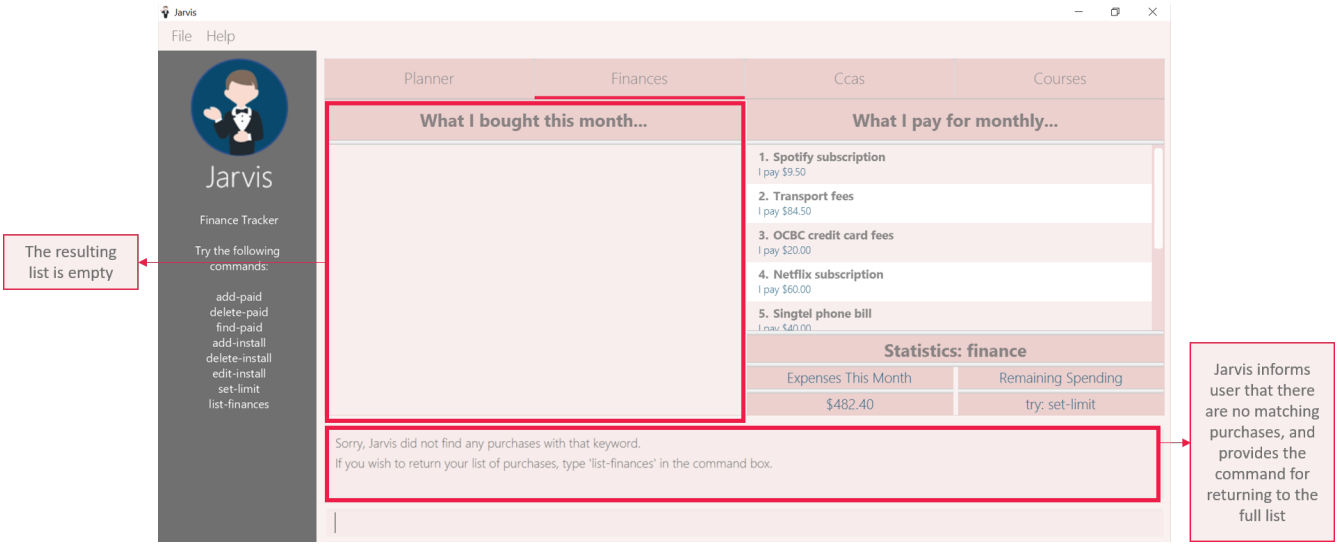

Figure 6. No purchases found in Finance Tracker

For examples of more commands for the Finance Tracker feature, please click to view our user guide.

Contributions to Developer Guide

Similar to the user guide, my team also decided that each member should be responsible to write their own documentation in the developer guide for their individual feature. The following section shows my addition to the Jarvis Developer Guide for the Finance Tracker feature. It will contain modified excerpts from the developer guide.

Finance Tracker Model

The Finance Tracker mechanism is facilitated by the `FinanceTrackerModel` interface, which is implemented by `Model`. This allows the model to be associated with the finance tracker feature and provides an interface between the components of the feature and the updating of the overall model.

Some of the more significant methods within the `FinanceTracker` are shown below:

- `Model#addPurchase(Purchase)` - Adds a single use payment to the top of the list
- `Model#deletePurchase(Index)` - Deletes single use payment at that index
- `Model#addInstallment(Installment)` - Adds an installment
- `Model#deleteInstallment(Index)` - Deletes installment at that index
- `Model#setInstallment(Installment, Installment)` - Replaces an existing installment with a new installment
- `Model#calculateTotalSpending()` - Calculates the total expenditure by the user for this month
- `Model#calculateRemainingAmount()` - Calculates the remaining spending amount available to user

Components of the Finance Tracker

The Finance Tracker feature closely follows the extendable OOP solution already implemented within AB3. In the Finance Tracker, the `Installment` objects and the `Purchase` objects manage most aspects related to this feature, along with a single `MonthlyLimit` object. `Installment` and `Purchase` objects are stored in their respective `ObservableList` - `InstallmentList` and `PurchaseList`, which provide an abstraction with `add`, `delete`, and `set` operations that are called by `FinanceTracker` and its model.

Shown below is the class diagram for the Finance Tracker.

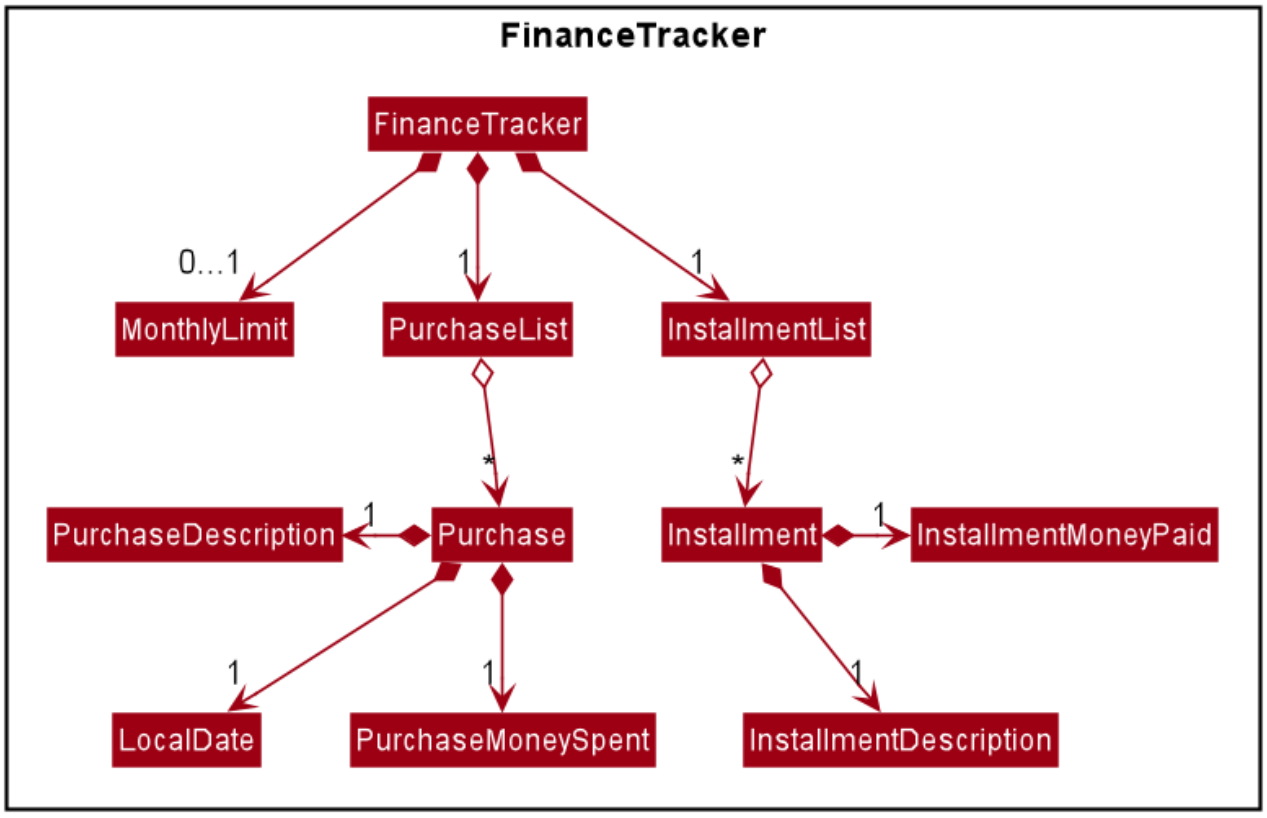

Figure 7. Finance Tracker Class Diagram

For more elaboration on the components of the Finance Tracker feature, please click to view our developer guide.

Implementation

To demonstrate the implementation of the Finance Tracker feature, I will be elaborating on the command that allows the user to edit an existing installment with the fields he has specified.

User enters command `edit-install`

The user has to specify the index of the installment he wishes to edit, as well as any of the fields he wishes to change. If the index does not exist, the system will inform the user of the error. As long as the fields provided by the user to be edited are valid (prefixed with "d/" and "a/"), the correct installment will be accurately edited. This is reflected in the activity diagram below.

NOTE	An index is considered invalid if the numerical value provided is less than or equal to zero, or greater than the largest index in `InstallmentList`.

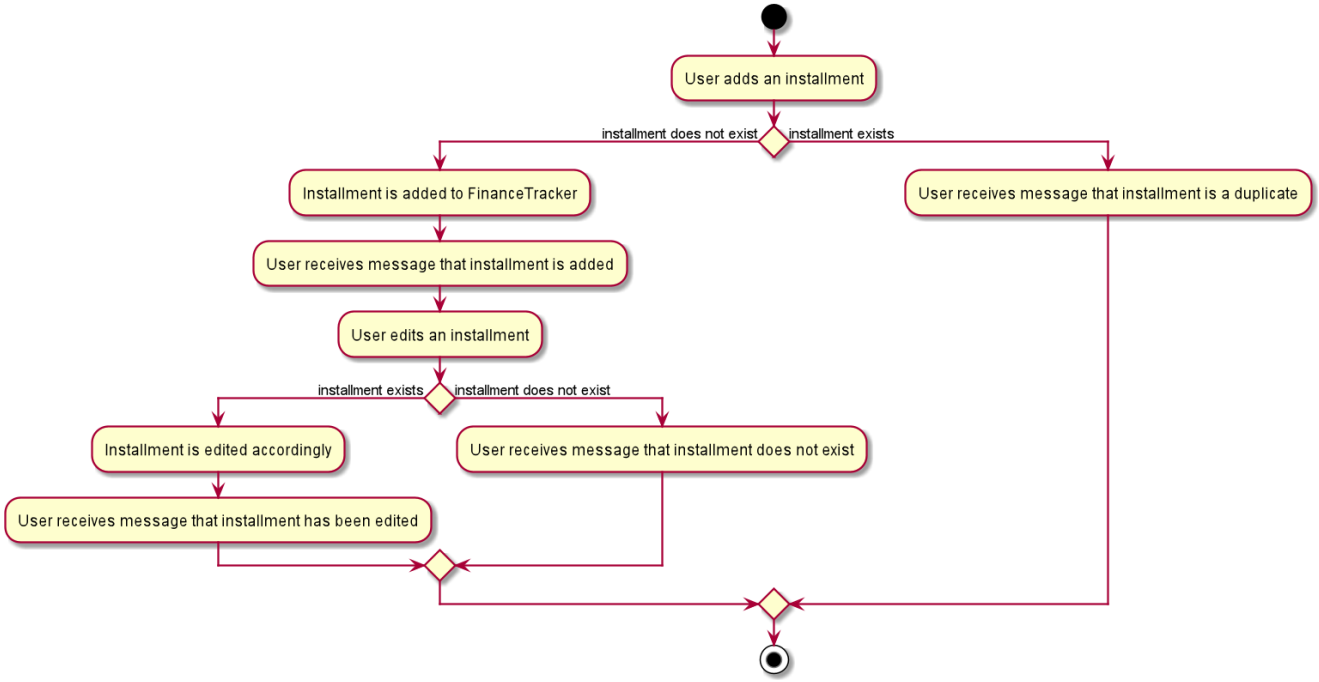

Figure 8. Activity Diagram for edit-install command

Execution of command `edit-install`

The following sequence diagram illustrates how an `Installment` is edited when a user types in a `edit-install` command.

In the execute method of EditInstallmentCommand, the calling of the `#setInstallment` method at the `Model` level triggers a cascading series of `#settInstallment` method which culminates in target installment being edited with the corresponding fields.

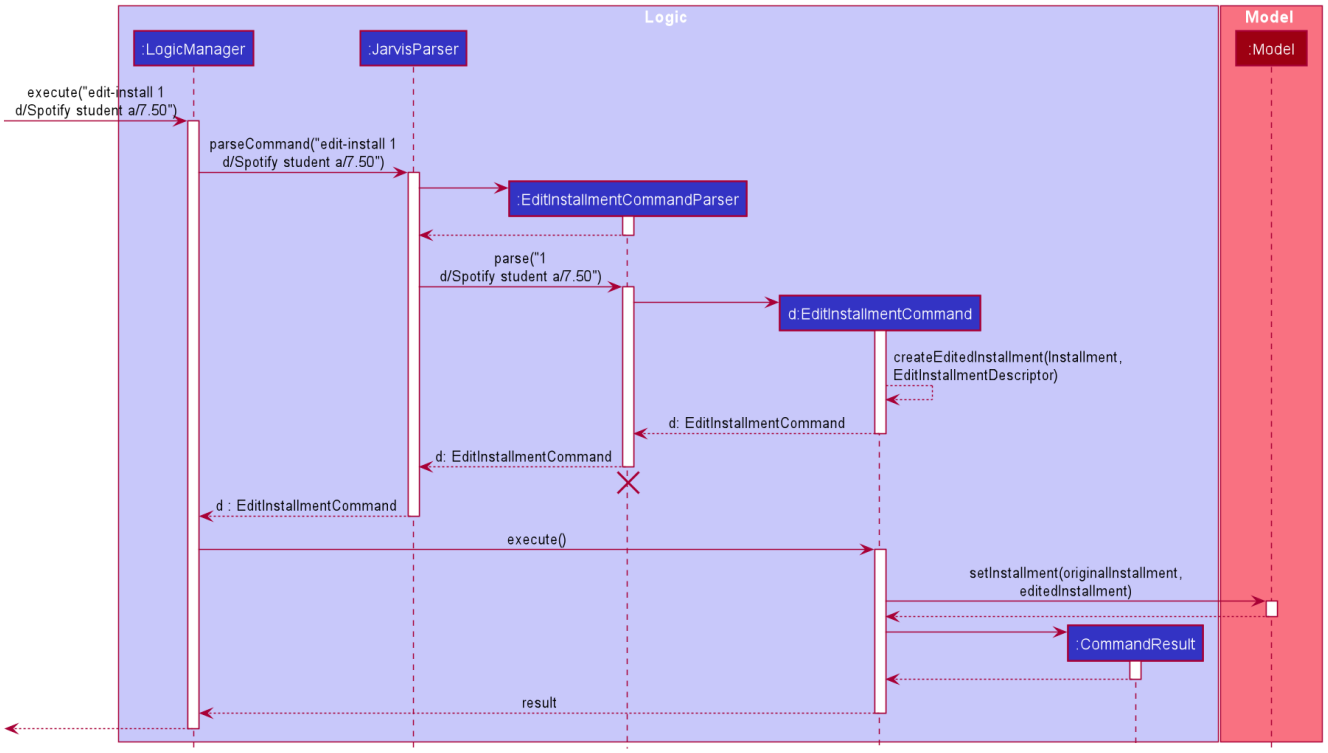

Figure 9. Sequence Diagram for edit-install command

Design Considerations

When designing the implementation of the Finance Tracker feature, I had to think of the best way to create and manage the various objects in the Finance Tracker as there were many sub-components. Thus, I had to consider whether to encapsulate the fields of the `Installment` and `Purchase` objects.

Current choice

Our current choice was to encapsulate the constituent objects in `Installment` and `Purchase` objects in their own wrapper classes. As mentioned above, `Installment` would contain `InstallmentDescription` and `InstallmentMoneyPaid` objects while `Purchase` would contain `PurchaseDescription` and `PurchaseMoneySpent` objects. The following is the analysis surrounding this design choice.

Pros: Increases OOP (hides implementation; increases extensibility and maintainability of objects)

Cons: Major changes to current code base; steep increase in code due to greater abstraction

Alternative

The alternative would have been to leave the corresponding fields as primitive data types. This would have been the easier alternative at the time as it was the original implementation. Furthermore, since the `Installment` and `Purchase` objects were not extremely complex, further encapsulation might not have been imperative.

Our Thoughts

Taking the long-term vision of the application to be continuously developed into consideration, I decided to increase OOP as much as possible. The decision was also based on following good software engineering principles.