Zheng Shaopeng - Project Portfolio

PROJECT: $AVE IT

Overview

$AVE IT is a desktop budget management application. The user interacts with it using a CLI, and it has a GUI created with JavaFX. It is written in Java, and has about 10 kLoC.

Summary of contributions

- **Enhancement added**: `calendar` feature in cooperate with `go` command which provides easy navigation between dates for users.
  - What it does: The `calendar` enables users to navigate between nearby dates easily by just a simple click. While `go` command allows user to navigate to any valid date with just a simple line of command.
  - Justification: There will be occasions when users will just like to view the nearby dates just check their expenditure records. Despite that users prefer command line input,but I feel that it is inconvenient for them to keep on type in command line for each of the navigation, thus `calendar` is developed to solve such problem. On the other hand, users may also want to navigate to a date which is far away. In this situation, `calendar` will not be helpful, thus `go` command is developed to resolve such issue, enable the users to navigate with just a simple line of command. In any of the scenario, we ensure that
  - Highlights:
    - This feature works with other features such as `budget` and `find`.
      - When users navigate to a different `month`, `budget` will be auto updated to display relevant data for that `month`.
      - Similarly, this feature is able to function as usual when the user enters `find` mode.
    - I felt that the implementation was quite challenging because it requires a certain knowledge on how to use tools such as JavaFX and SceneBuilder which were used to design the graphical user interface (GUI). With no prior experience, the learning curve was steep which required more effort to learn about these tools.
- **Enhancement added**: `repeat` feature which is a recurring expenditure.
  - What it does: Users can record a fix expenditure records which is recurring daily, weekly, monthly or annually.
  - Justification: It is a norm that users have a certain fixed type of spending such as transport. Thus, the team has take into account of such situation and developed `repeat` which caters to such need.

    - Highlights:
        - Initial stage, there is consideration to just make it a mass operation of `exp` type command. This approach is much more simpler for the developers, however it will cause lots of inconvenience to the users. For example, when users wants to edit all those recurring expenditures' `info`, they have to spend time to edit each of them individually and this is very troublesome. Thus, I have to change the implementation approach by introducing a `repeat` class. This implementation is much more challenging especially when there is data structure and storage involved.

- **Minor enhancement**: helps to develop in developing `budget` feature.
- **Code contributed**: [RepoSense Code Dashboard]
- **Other contributions**:
    - Project management:
        - Managed releases `v1.1` - `v1.4` (3 releases) on GitHub
    - Enhancements to existing features:
        - Updated the GUI for calendar view (Pull requests #86, #88, #102)
        - Wrote additional tests for existing features to increase coverage from 40% to 50% (Pull requests #213, #215, #216)
    - Documentation:
        - Updated developer guide to standardise the style and formatting(Pull requests #122).
        - Added documentation for `calendar` feature in the User Guide. (: #230).
        - Added documentation for `calendar` and `repeat` feature in the Developer Guide. (: #225).
    - Community:
        - PRs reviewed (with non-trivial review comments): #97, #146, #148, #159

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.*

Calendar (Zheng Shaopeng)

The *calendar* feature aims to ease users' navigation experience.

1. Users are able to traverse between different dates by clicking on the calendar.
   Or,
2. Users are able to make use of `go` command to navigate to another date.

With reference to the diagram below, **Header** displays the year and month which the calendar is displaying. **Box 1** displays the day which you are looking at while **Box 2** refers to the today's date.

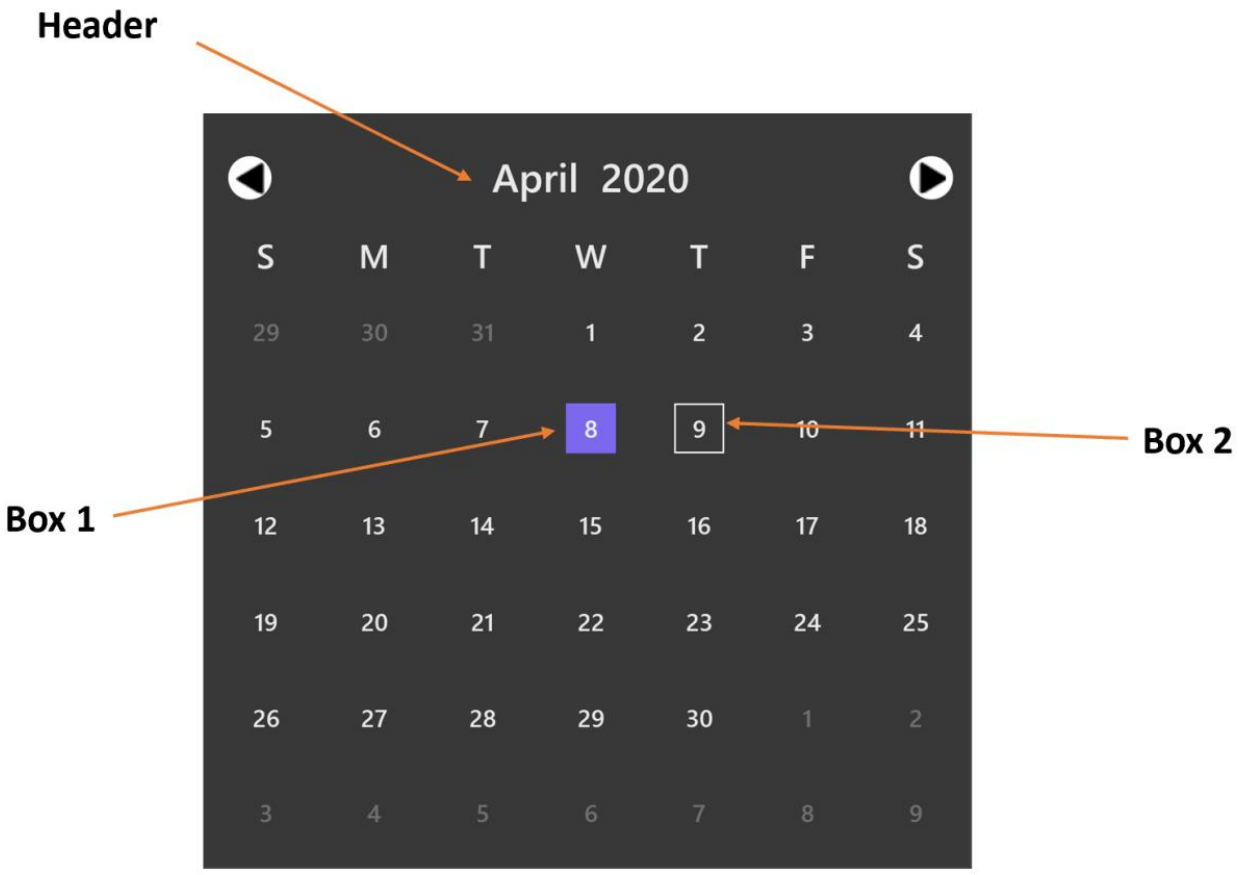

Figure 1. Calendar View

Navigate using calendar

It may be troublesome and inconvenient for you to keep on making use of the command line input to navigate between the dates, especially when the date you want to view is just one or two days before.
In this situation, you can interact with the `calendar` and navigate with a simple click.

For example, we are now viewing expenditure records which are on `2020-04-09`. And we wish to view previous day's expenditure record. We can simply click on the date on the calendar. Below are two diagrams which will show the operation.

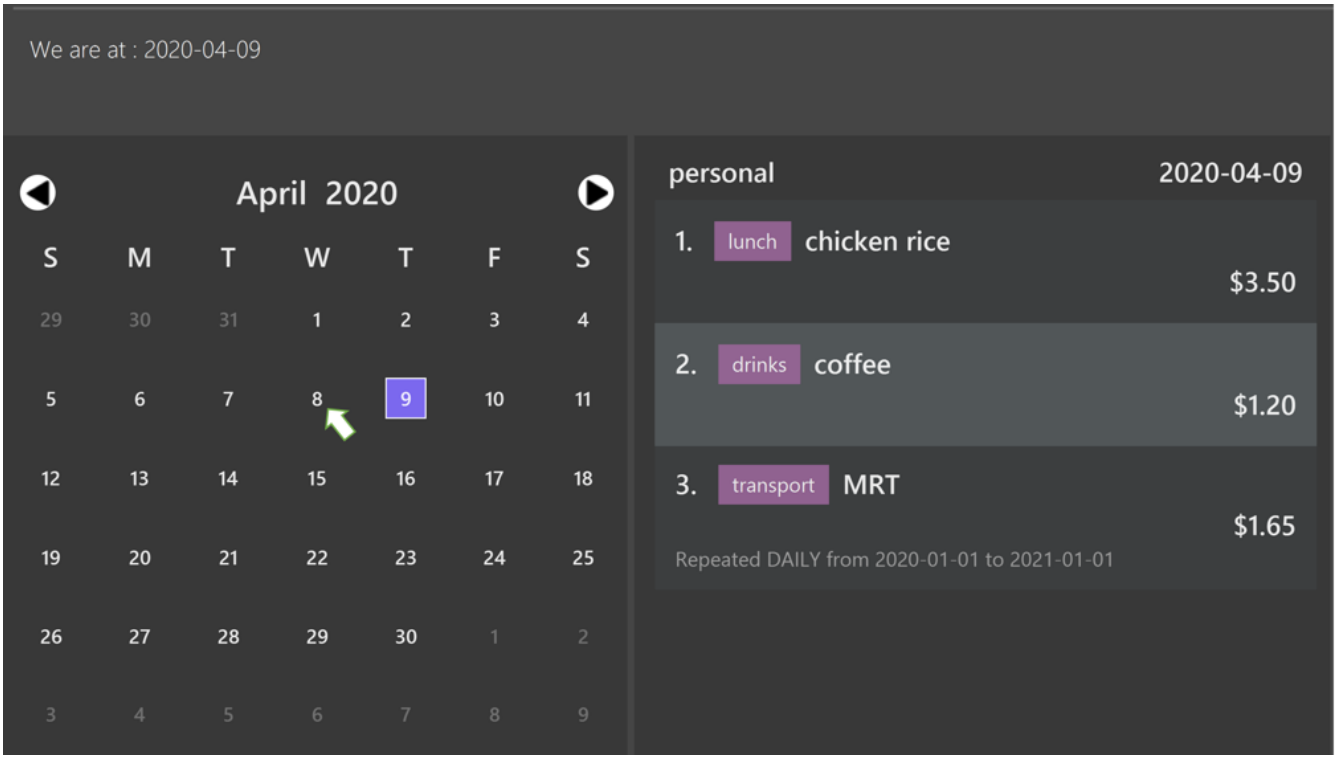

Figure 2. Before click (9 April)

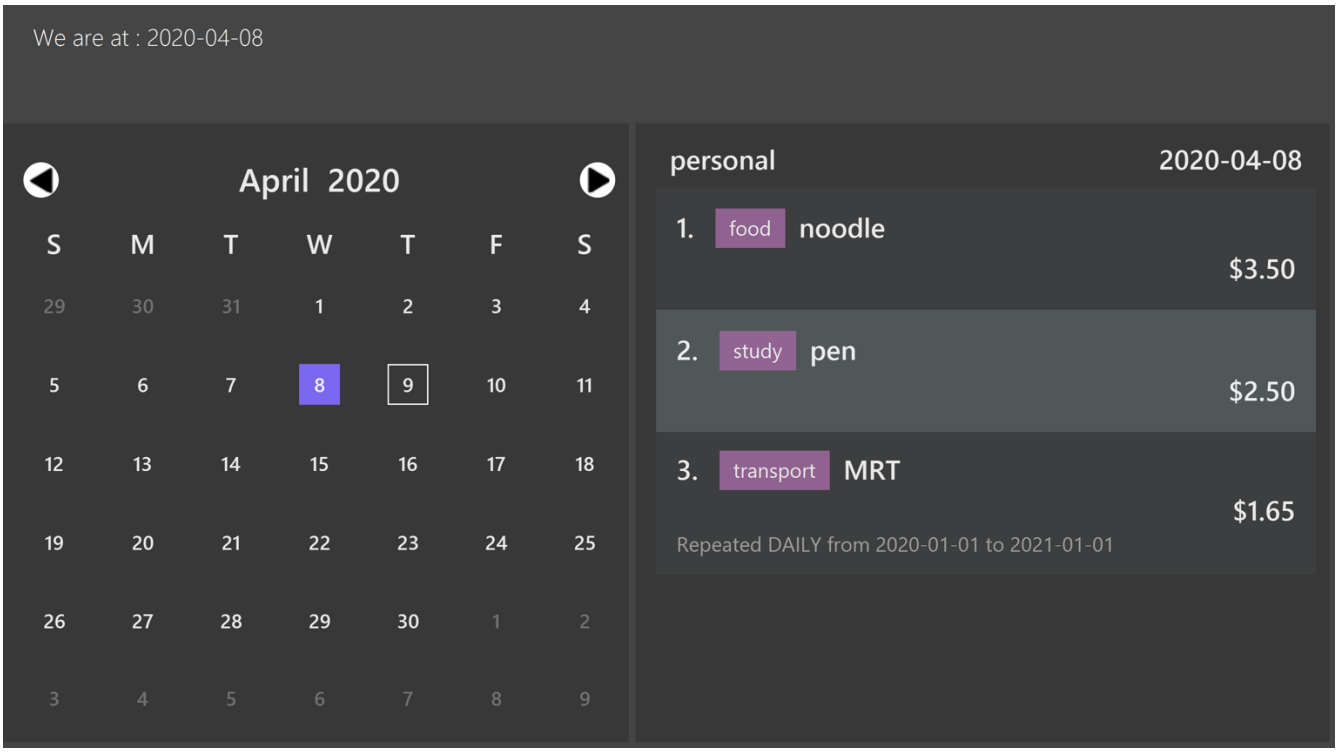

Figure 3. After click (8 April)

Navigate with `go` command

The developer team has taken into account the case when users wish to navigate to another date which is way before or after. Thus, we have integrate the `calendar` with `go` command.

For example, we are now viewing expenditure records which are on `2020-04-09` and we wish to navigate to `2019-04-09` through `go` command.

1. We need to input `go 2019-04-09` into the command box which is shown in the diagram below.

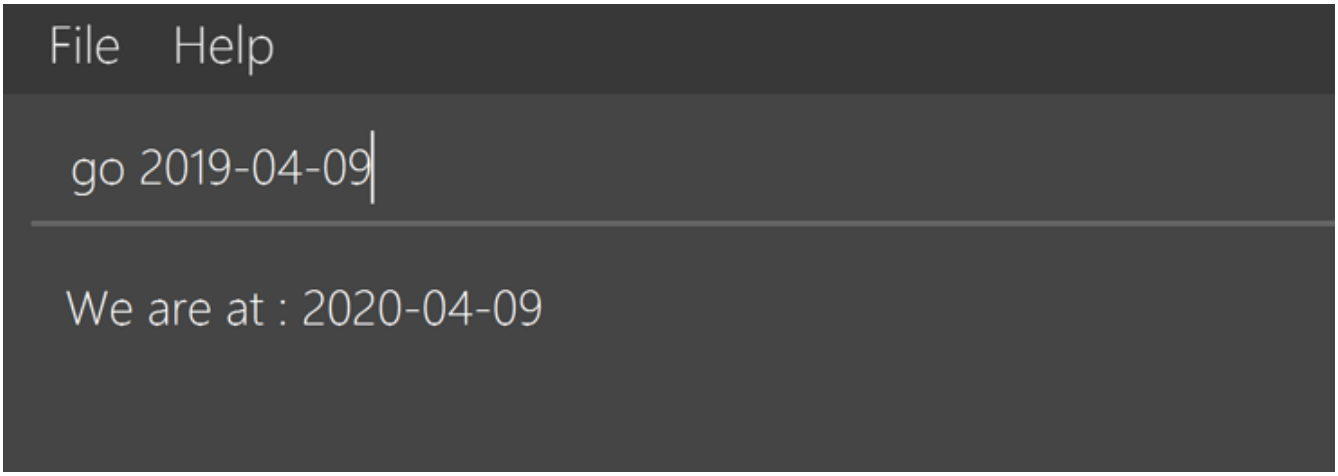

Figure 4. `go` command example

2. A response will be given to indicate that the date has changed.

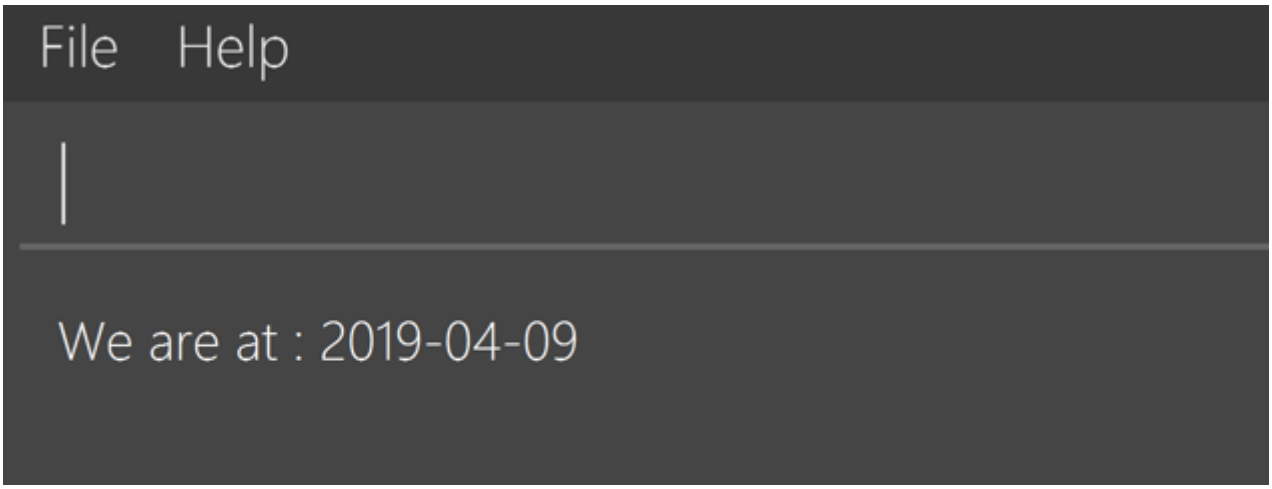

Figure 5. `go` command response

3. The `calendar` and the expenditure records will update accordingly.

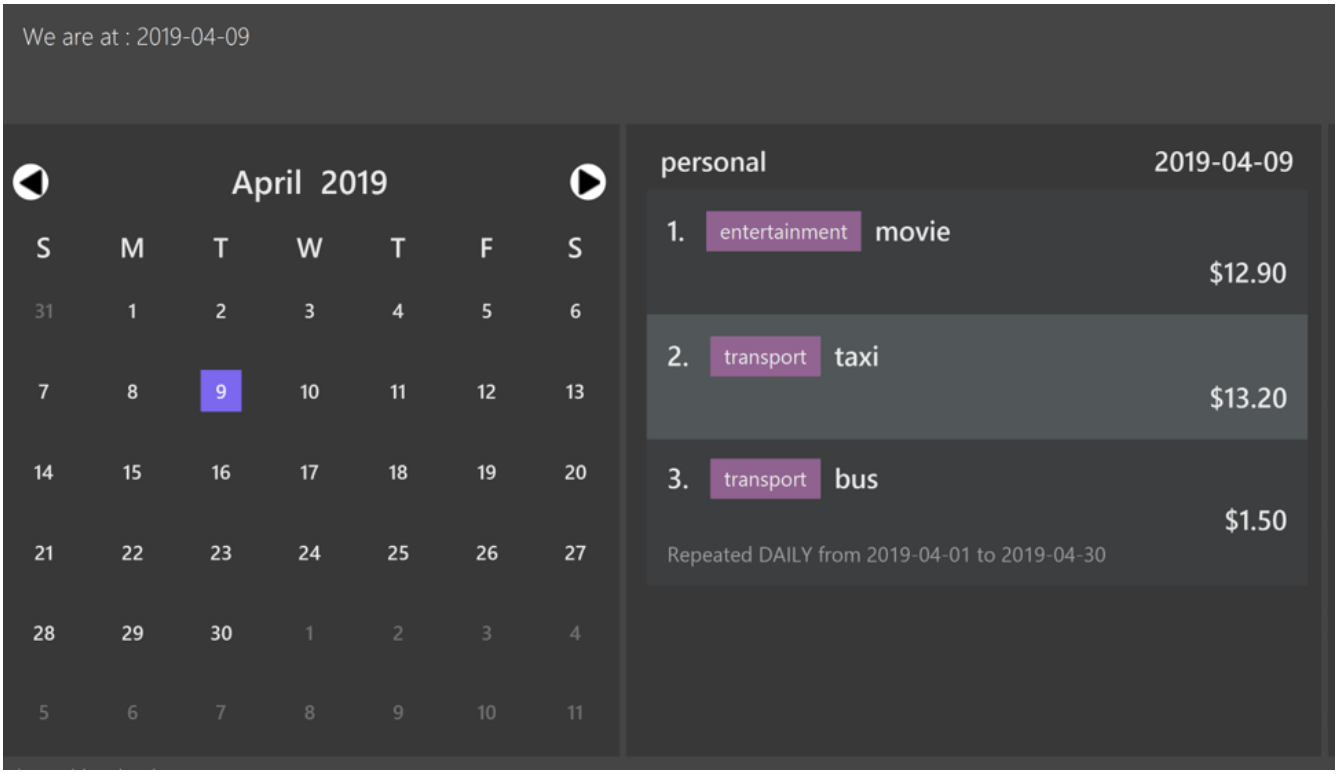

Figure 6. `calendar` and expenditure records are updated accordingly.

Repeat (Zheng Shaopeng)

The *Repeat* feature allows you to add recurring *expenditures*.
You are able to create fixed expenditures records which will be recurring daily, weekly, monthly or annually. As shown in the diagram below, a `repeat` will have a different display as compared to `expenditure`. "Repeat details" includes the frequency of recurring, start date and end date are displayed.

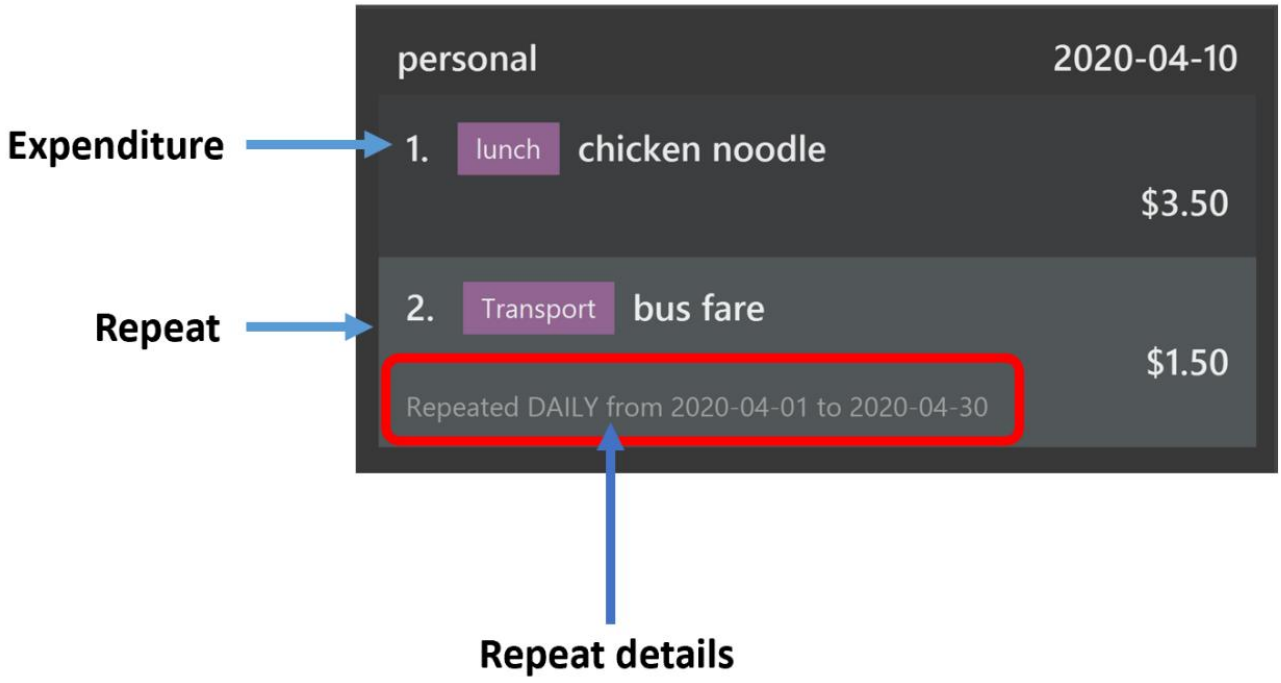

Figure 7. Display difference between `Expenditure` and `Repeat`

`Repeat` type commands

`Repeat` has very similar command as `Expenditure`, with just a few of values to be specified.
Here, we will be using `repeat add` command as an example, while other `repeat` commands works the same way.
Format: `repeat add -i INFO -a AMOUNT -sd START_DATE -ed END_DATE -p PERIOD [-t TAG]`

1. Key in the command into the command box. We will be using `repeat add -i bubble tea -a 3.50 -sd 2020-04-01 -ed 2020-04-30 -p weekly -t drink` as an example.

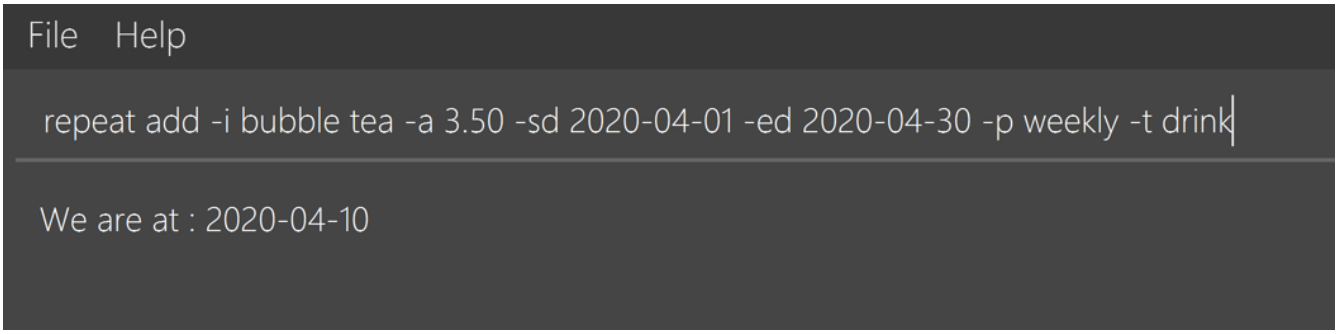

Figure 8. Input command

2. A response will be given to indicate that this recurring expenditure has been recorded.

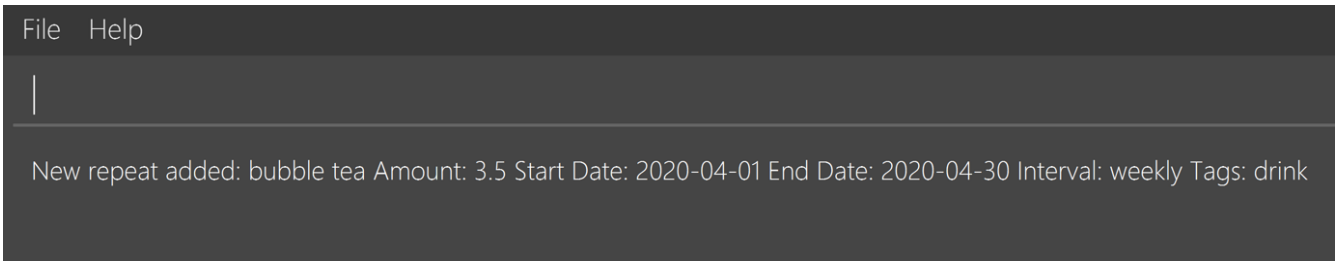

Figure 9. Response given

3. As the `START_DATE` is 2020-04-01 and the `PERIOD` is set to weekly, thus the first record will be at 2020-04-01 and last record will be on 2020-04-29.

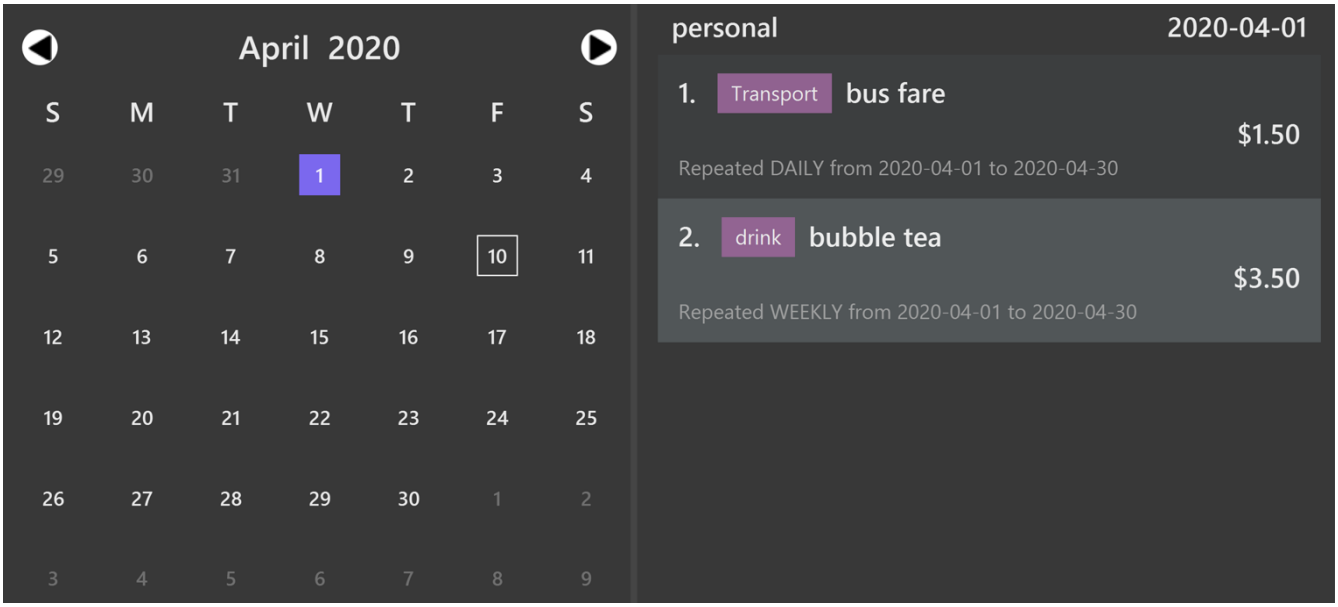

Figure 10. Added to 2020-04-01

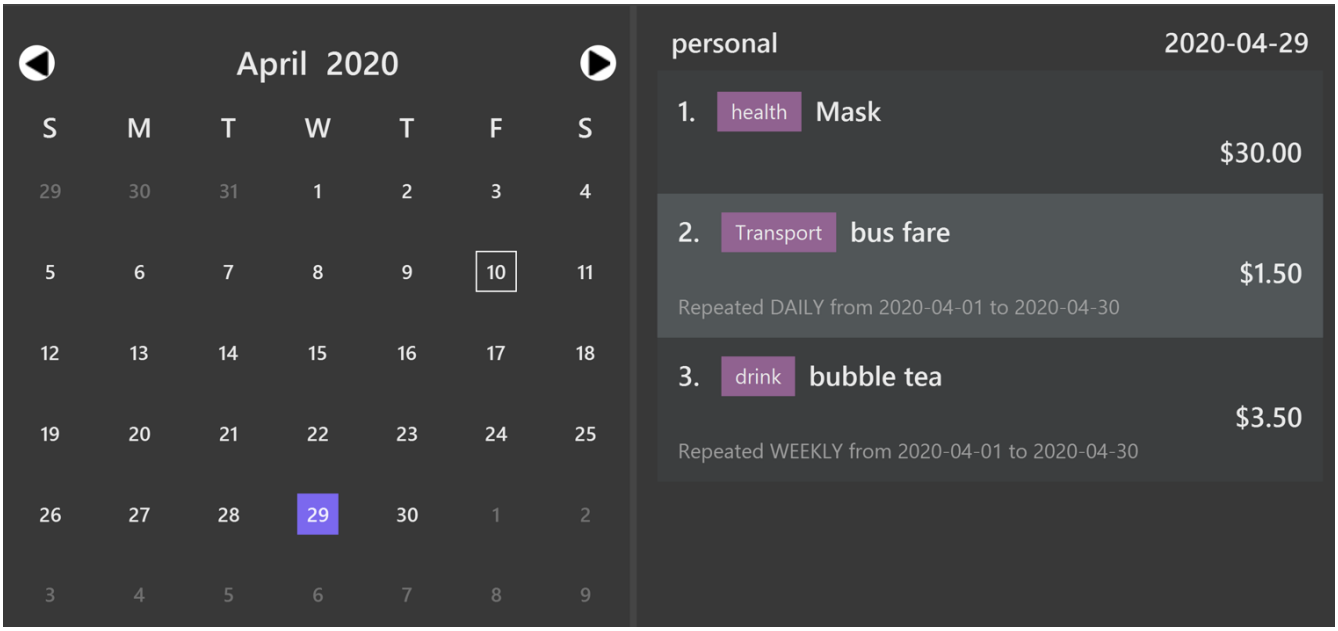

Figure 11. Added to 2020-04-29

NOTE

For `repeat add -i rental fee -a 300 -sd 2020-03-31 -ed 2021-03-31 -p monthly -t housing`, as the start date is `2020-03-31`, the next tentative date should be `2020-04-31` but this date is invalid. Hence, this expenditure record will be shown on `2020-04-30` instead. For May, it will be still shown on `2020-05-31`.
This applies to leap year too

Refer to Repeat Commands for more details on how to add, edit and delete `Repeat` type expenditures.

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.

Recurring Expenditure feature `Repeat` (Zheng Shaopeng)

Recurring expenditure is one of the main features in `$AVE IT` and it is an expenditure automatically logged for user at their preferred frequency.

Rationale

`Repeat` allows user to keep track of expenditures that will occur either *daily*, *weekly*, *monthly* or *annually* without the need to key in the expenditures every day or month. Hence, this will provide more convenience for users as well as address the need for such a feature since recurring expenditures are common. For example, day to day commuting expenses.

Implementation

Below is a class diagram shows different components that `Repeat` contains.

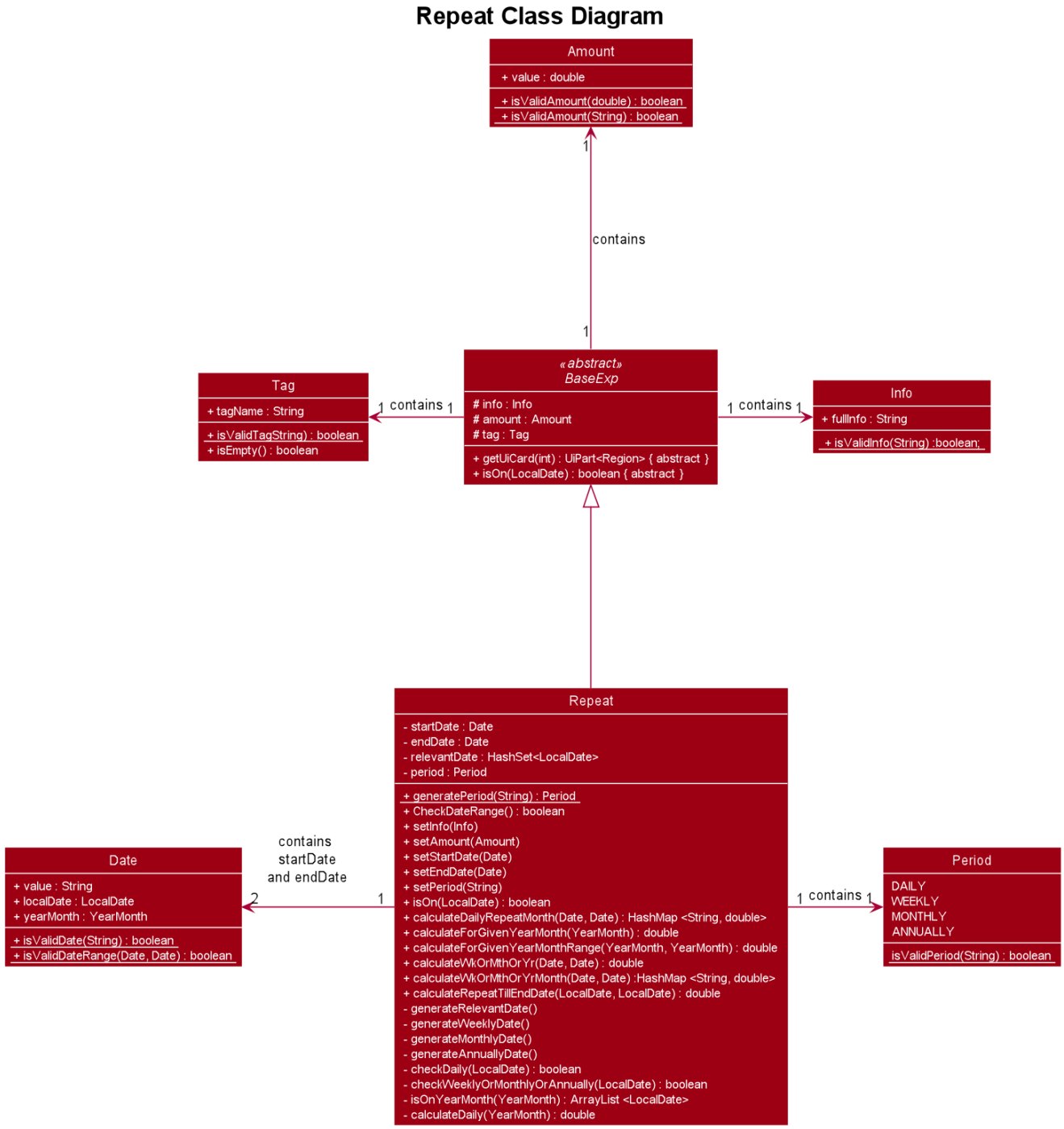

Figure 12. Class diagram for showing what `Repeat` consist.

For each account, it has its own `list` which all the `Repeat` objects are stored. There are different types of command that is cater for `Repeat` such as add, edit and delete. The following activity diagram shows what how a `Repeat` can be added.

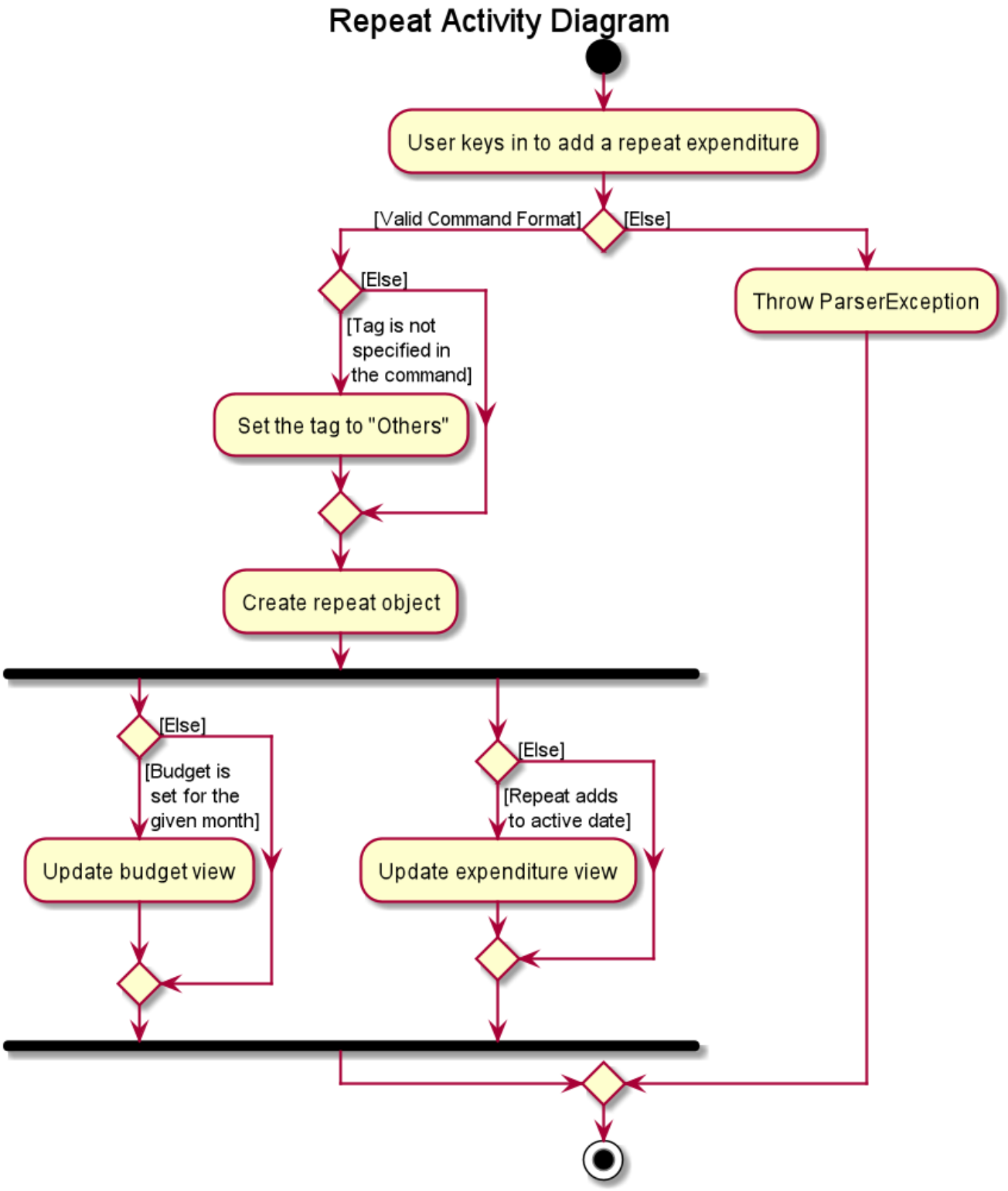

Figure 13. Activity diagram for `repeat add`

Design Consideration

Alternatives	Pros	Cons
(Current choice) Have a repeat class which extends `BaseExp`.	Able to mass delete and edit all the expenditures under this `Repeat` easily.	Hard to implement, especially when we have to calculate monthly spending so to generate report and statistics.

Mass operation: add `Expenditure` object to all those dates which state in the command.	Easy to implement.	User are unable to edit all the expenditures which are recurring. Users have to delete such expenditures one by one.

Calendar feature (Zheng Shaopeng)

Rationale

Calendar is a feature that has a clickable calendar which users can use to navigate between the different days. It also shows the date of the expenditures the user is viewing as well as today.

Implementation

The implementation of the above functions will be described separately in this section.

The users are given two different choice on how to navigate between the days:
1. UI interaction with the calendar view.
2. Make use of `go YYYY-MM-DD` command.

The following sequence diagram shows you how the `go YYYY-MM-DD` (E.g. `go 2020-04-01`) command works.

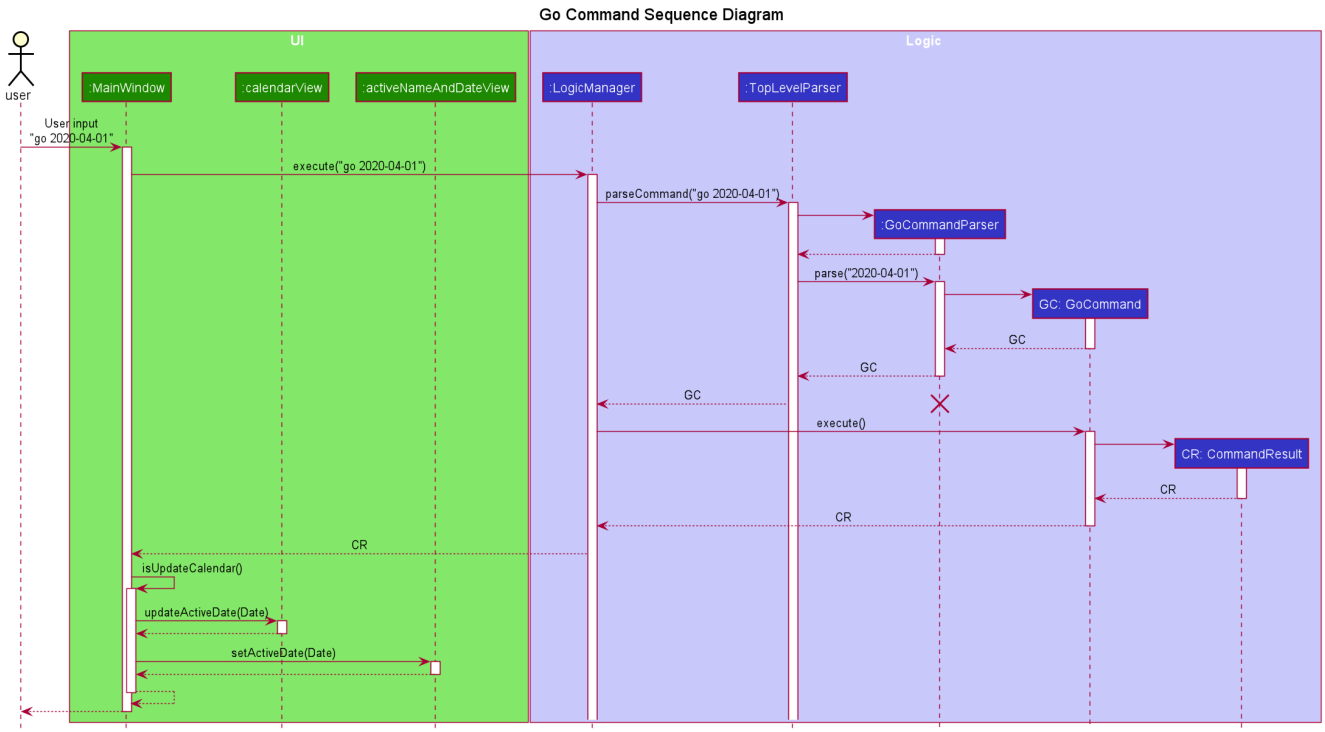

Figure 14. `UI` and `LOGIC` component for the `go 2020-04-01` command

Upon completion of the above command, the calendar view will update the active date to be `2020-04-01` and expenditures records for `2020-04-01` will be displayed.

If the user chooses to navigate through UI interaction with the calendar view (aka clicking on the

date that is shown on the calendar). The implementation is very similar to the `go` command, `calendarView` will invoke `go` command when user click on the dates.

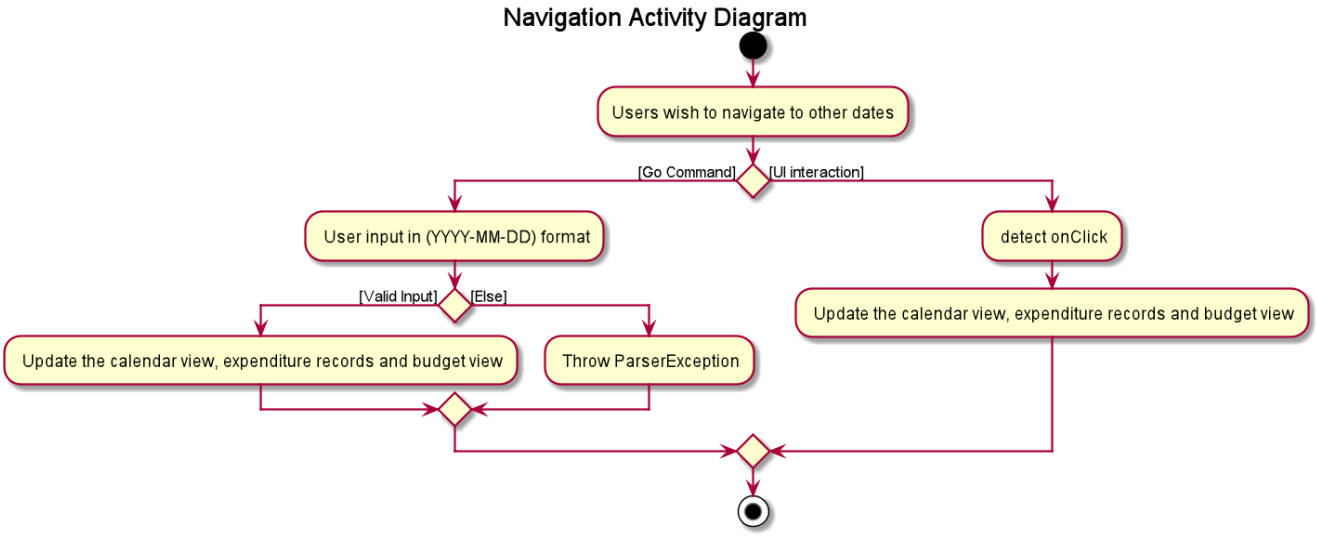

Figure 15. Activity diagram showing what happen when user wants to navigate to other date

Design Consideration

This section contains some of our design considerations for the calendar feature.

Consideration: How are we going to present the expenditure records.

Alternatives	Pros	Cons
1. Make use of a month list to contain all the expenditure records of the given month.	This is able to provide a concise view of expenditure view especially when there are only a small number of records.	This looks like excel sheet and users have to scroll all the way up if they want to view a date which is much earlier.
2. **[current choice]** Make use of a calendar view and only list out a given date's expenditure record. This automatically helps user to organize the records according to date.	User can make use of the calendar view to navigate between the dates, this is much more convenient than scrolling through a list. This helps user to organize the expenditure and keep it tidy. Especially helpful if there is lots of records.	It is much more troublesome to implement.

NOTE

Dates with negative year are allowed. E.g. `-1234-03-21` is allowed.
The developer team follows the range which is specified in the LocalDate API, released by Oracle.