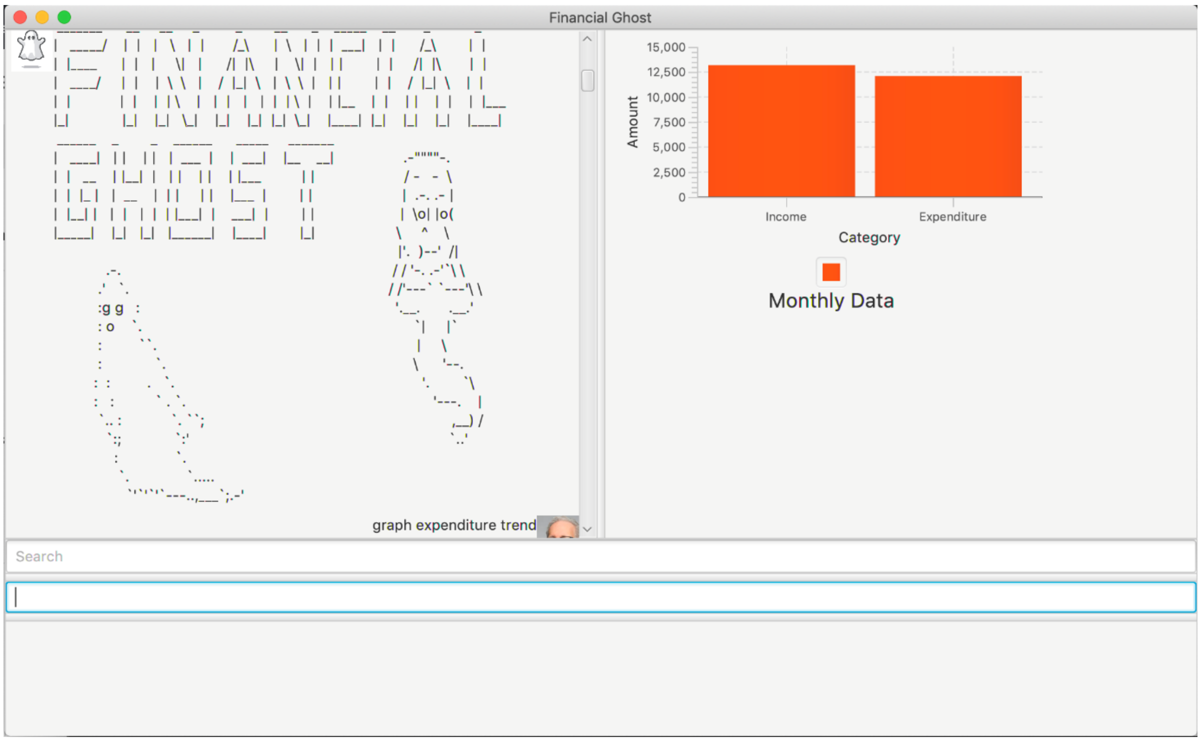
Chen Chao – Project Portfolio for Financial Ghosts

About the Project

My team and I were tasked with enhancing a basic command line interface (CLI) application for our Software Engineering project. We chose to morph it into a financial planner called Financial Ghost. Besides enabling the user to track their basic finances, this single-user application also includes other tools to manage the user's goal setting, outstanding loans, instalments and bank accounts. A CLI application with GUI output, Financial Ghost can both read and display user data efficiently, and present statistics based on the user's financial information. Below illustrates a typical usage of Financial Ghost.



I undertook the task to write the base classes and implement the instalments functionality during the morphing process. In addition to that, I wrote the code for all commands for autocomplete feature and history feature. I will illustrate my contribution in the following section, together with my contributions towards the documentation of Financial Ghost.

Summary of contributions

- **Major enhancement: added the ability to manage the user's Instalments.**
  - **What it does:** Allows the user to record their financial instalments in which it is automatically added to their expenditure monthly when the payment is due.
  - **Justification:** This feature enhances the practicality of the product by using the current financial information tracked by the application to help the user record their Instalments and help them to record it as an expenditure monthly. By doing so, they can be financially accountable for their expenses. This feature is thus a core feature of our product as tracking instalments are a necessity for any user tracking their finances.
  - **Highlights:** This enhancement helps the user to track their instalments that they have paid thus far and gives them a reminder on the remaining portion of the instalment. Furthermore, it automatically removes the instalments when the user pays for his instalment fully to ensure the user's financial statement is updated.
- **Minor Enhancement: added autocomplete drop down list to command text bar.**
  - **What it does:** Automatically search through the command list and return a list of command that autocompletes what the user is trying to type.
  - **Justifications:** It will be difficult for the user to memorise all the commands. The drop-down list suggests possible commands as he/she types, making keying in commands much easier.
  - **Highlights:** Users can make use of the drop-down list to type commands faster. They can directly select the correct suggestion without having to manually type out the command word in full.
- **Minor Enhancement: added history function to command text bar.**
  - **What it does:** Every command that the user type into the command bar is recorded and if the in a single usage. If the user presses up, he/she will be able to see the previous commands that they key in.
  - **Justifications:** It will be trouble some if the user accidentally type something wrong in the command which is a minor format mistake. This feature allows the user to go back to the previous command that the user type in and alter that mistake instead of re-typing the whole command again. This helps the user to save time when such an error occurs.
  - **Highlights:** The user can press up to see previous commands that he/she previously entered, they can also press down to see the next command. The top most command that it memorised is the first command that the user have typed

and the bottom most command returns the user to a blank empty command text bar.

Functional Code: [Code]

Other Contributions:

- **Project Management:**
  - Managed Milestone v1.4 and creating issues for the team to complete.
  - Manage release for V-mid1.3.
- **Enhancements to existing features:**
  - Improved efficiency and readability of code in command (PR: #127)
  - Wrote additional tests for existing features to increase coverage
- **Documentation:**
  - Helped with formatting of User Guide and Developer Guide to enhance readability
  - Wrote Quick Start for User Guide and Setting Up for Developer Guide
- **Community:**
  - Providing Code base for initial Morphing: Account Class (PR: #35)
  - Resolving Admin Issues:
    - Checkstyle Phase 1, Phase 2 (PR: #43, #44, #45, #47, #230)
    - Authorship Tagging (PR: #118)
  - Review PRs
  - Reported bugs and suggestions for other teams in the class
  - Contributed to the forum for requests for using third-party libraries: issue #26
- **Tools:**
  - Integrated a third-party library (Controlsfx) to allow for autocompletion binding.

Contributions to User Guide

This section summarises my contribution to the User Guide. It comprises of two sections: Instalment Management and Help feature. Due to space constraints, I have removed some trivial features like listing and deletion of Instalment. Please refer to Section 3.5. Instalment management and Section 3.9 Help for the remainders of my contributions.

The following are extractions I have taken from the User Guide.

3.5. Instalment Management

This section contains the commands to manage the user's outstanding instalments.

3.5.1 Add instalment: add instalment

Allows the user to keep track of the uncompleted payments that they have to settle via recurring monthly instalments. Also shows a percentage bar at the same time.

Format: `add instalment [desc] /amt [cost] /within [number of months] months /from [d/m/yyyy] /percentage [annual interest rate]`

- The right output will list all the instalments with the added instalment.
- The left output will show a success message.

Examples:

1. User enters `add instalment motorbike /amt 20000 /within 200 months /from lstwk /percentage 3.8` into the command bar and press `Enter` on the keyboard.
2. The instalment list will be displayed in the pane on the right as enclosed by the yellow box, showing the progress towards the respective instalments in square brackets e.g [71.48%].
3. The success message will be shown on the left pane as enclosed by the green box.

3.5.4 Recurring Instalments

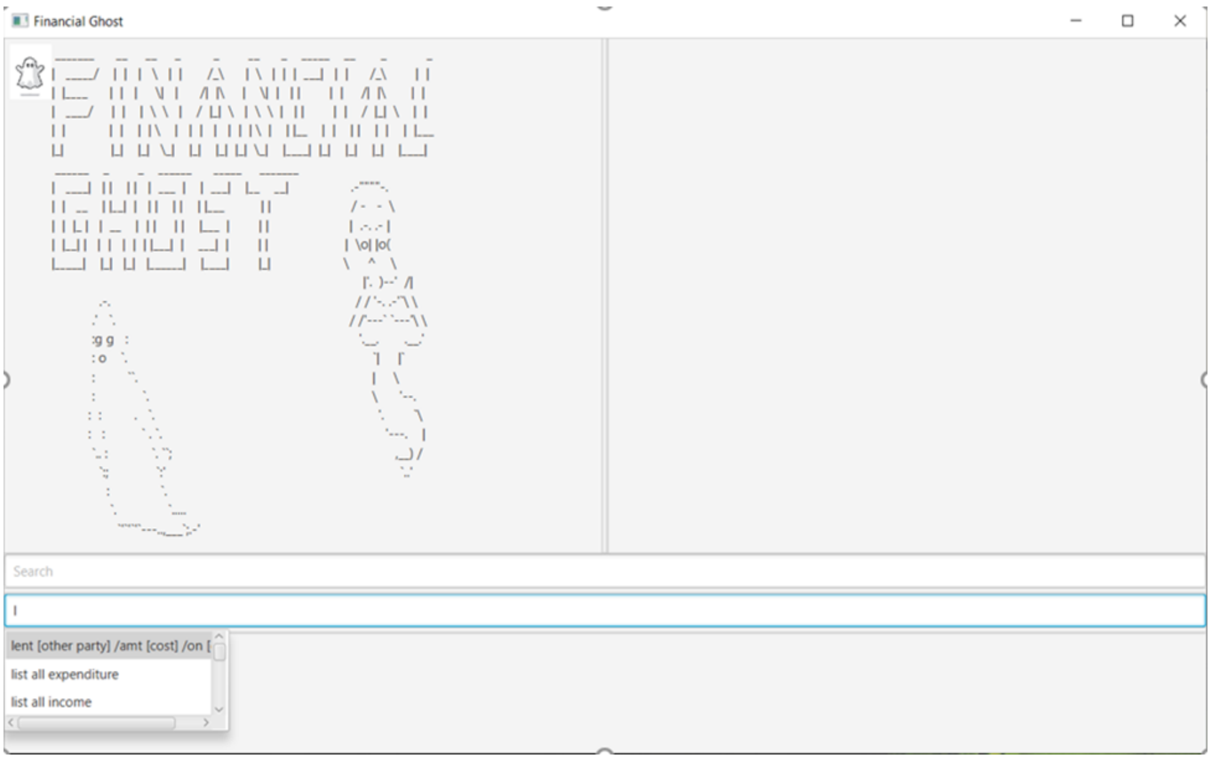
Automatically count monthly instalments as a part of Expenditure until last instalment and from there on it will terminate. Notify the user monthly when it deducts and when it ends. Auto remove it when it is done. No command required from the user.

3.9. Help

3.9.1 Auto-Complete

This function autocompletes user commands using a dropdown list.

With the help of the drop-down list of suggestions, you do not need to memorise any command. When you type your commands inside the command box, you will notice that a drop-down list appears.
Example: If you type the letters “l", a drop-down list consisting of all options that starts with “l” appears, as shown in the screenshot below.



```
l
lent [other party] /amt [cost] /on [
list all expenditure
list all income
```

When the drop-down list is shown, you can:

- Use ↑ and ↓ arrow keys to navigate through the list and press Enter to select the highlighted option, or
- Use your mouse to click on the option to select it

After you have selected the command you wish to enter, you should see it displayed in the command box.
Now you can:

- Edit those inside the “[]” to fill up the command format and remove “[]” at the same time. Press Enter to execute the command.
- Execute the command if there is no such “[]” in the command.

Contributions to Developer Guide

This section summarises my contribution to the Developer Guide. It comprises of two sections: Instalment Management and Help feature. Similarly, I have removed some features. Please refer to Section 3.5. Instalment management and Section 3.9 Help for the remainders of my contributions.

The following are extractions from the Developer Guide. With some features removed.

3.5. Instalment Feature

This feature allows users to manage and set their instalments.

It comprises of the following functions:

1. **Add Instalment Function**
2. **Automatically Update Instalment Payment Feature**
3. **Delete Instalment Function**
4. **List Instalments Function**

3.5.2. Automatically Update Instalment Payment Feature

The auto update instalment payments feature is facilitated by the AutoUpdateInstalmentCommand from the Logic Component. An Instalment object is already instantiated. An AutoUpdateInstalmentCommand object has current date that it is created as it's attributes. All Instalment objects is automatically pushed as an expenditure on its payment date every month.

Given below is an example usage scenario of how the auto update instalment payments feature behaves at each step.

Step 1: The user calls the any command from the GUI with its respective parameters.

Step 2: The LogicManager automatically instantiates an AutoUpdateInstalmentCommand object and calls executes() on the AutoUpdateInstalmentObject.

Step 3: The Logic Component will interact with account and extract all Instalment objects and iterate through the list of instalments.

Step 4: If the current date is equal to the pay date of a particular instalment object, and the payment is not done, the Logic Component will interact with Account to add the instalment as an expenditure.

Step 5: The Storage component will overwrite the original txt file with the new list which contains the added expenditure and updated instalment payment.

Step 6: Step 4 to 5 is repeated for all the instalment object till all of them is checked through.

3.3.5. Design Considerations

- **Implementation of AddInstalmentCommand**

  Alternative 1: Overriding execute() (Current Choice)

  Pros: Easy for developers to understand as all commands override execute().

  Cons: All classes of commands that is part of the moneycommands package must write their own overriding method of execute().

  Alternative 2: Writing individual executeAddInstalmentCommand()

  Pros: Can cater to the needs of adding instalments directly and do not have to implement it in every moneycommands.

  Cons: Harder for developers to understand.

- **Saving of new Instalment to txt file**

  Alternative 1: Override entire text file (Current Choice)

  Pros: Easy for developers to implement.

  Cons: May have performance issues in terms of memory usage.

  Alternative 2: Append to text file.

  Pros: Efficient data storage technique.

  Cons: Implementation issues and is hard to track the respective data files of various types.

3.9. Help Feature

This feature allows users to manage and set their instalments.

It comprises of the following functions:

1. Autocomplete function
2. History function

3.9.1. AutoComplete Function

The AutoComplete Function is facilitated by the MainWindow from the UI Component. A dynamic drop-down list is created in this process.

The dynamic drop-down list is created using **org.controlsfx.control.textfield.TextFields** API. The **bindAutoCompletion** method in the **TextFields** creates a new auto-completion binding between the given textField and the given suggestion provider.

Possible commands are checked against the user input and added to the TreeSet **suggestedCommands** accordingly, which will be displayed as the drop-down list.

A command is added if the following conditions are satisfied:

1. The user input is not empty. This is to prevent the list from dropping down every single time when the command bar is cleared.

2. The user input is a valid command. There is no need for any suggestions if the user keys in a command that matches with one of the correct commands
3. The command starts with the user input. This is so as to suggest the correct user command that the user is trying to type.

Given below is an example usage scenario and how the AutoComplete mechanism behaves at every step.

Step 1: The user types in the first letter of the commands that they want to type into the command bar.

Step 2: Upon key release of the character typed into the command bar, the Logic Manager (FG) calls autoCompleteFunction and an AutoComplete object is created. A TreeSet of suggested commands is created at the same time.

Step 3: AutoComplete object will iterate through the library of commands and push the match commands that satisfied the three conditions above into the TreeSet of suggested Commands.

Step 4: The Treeset of suggested commands will then be returned and will be bind with the TextField UserInput.

Step 5: As user continues to key in more letter into the command bar, Step 2 to 4 is repeated over and over again until the user choose to select a command as mentioned in Condition 2 above.

<u>3.9.3. Design Considerations</u>

- **Auto-Completion Binding**

Alternative 1: Only suggests Commands that starts with user input (Current Choice)

Pros: Suggested Commands are accurate.

Alternative 2: Suggesting any commands that contains user input

Pros: Displays more suggestions.

Cons: Unnecessary and Inaccurate suggestions tend to appear.

- **Data Structure of suggestedCommands in Autocompletion**

TreeSet is being used. TreeSet seems to be the most feasible data structure as the list of suggestedCommands needs to be sorted and TreeSet's basic operation have a time complexity of O(logn) time which is decently fast. This allows the binding to take place fast and allows the program to give almost immediate suggestions to the user as the user types.

- **Disabling History function when Autocompletion happens**

As both features of the help function acts on the same TextField UserInput (Command bar), KeyEvent that can be captured by both features. The list of suggestedCommands can be navigated with the ↑, ↓, and enter keys. However, these three key events can also be captured by the History function or the command bar itself. Hence, it is necessary to disable them when the dynamic drop-down list of suggested commands is being thrown.