Ong Yu Xuan - Project Portfolio

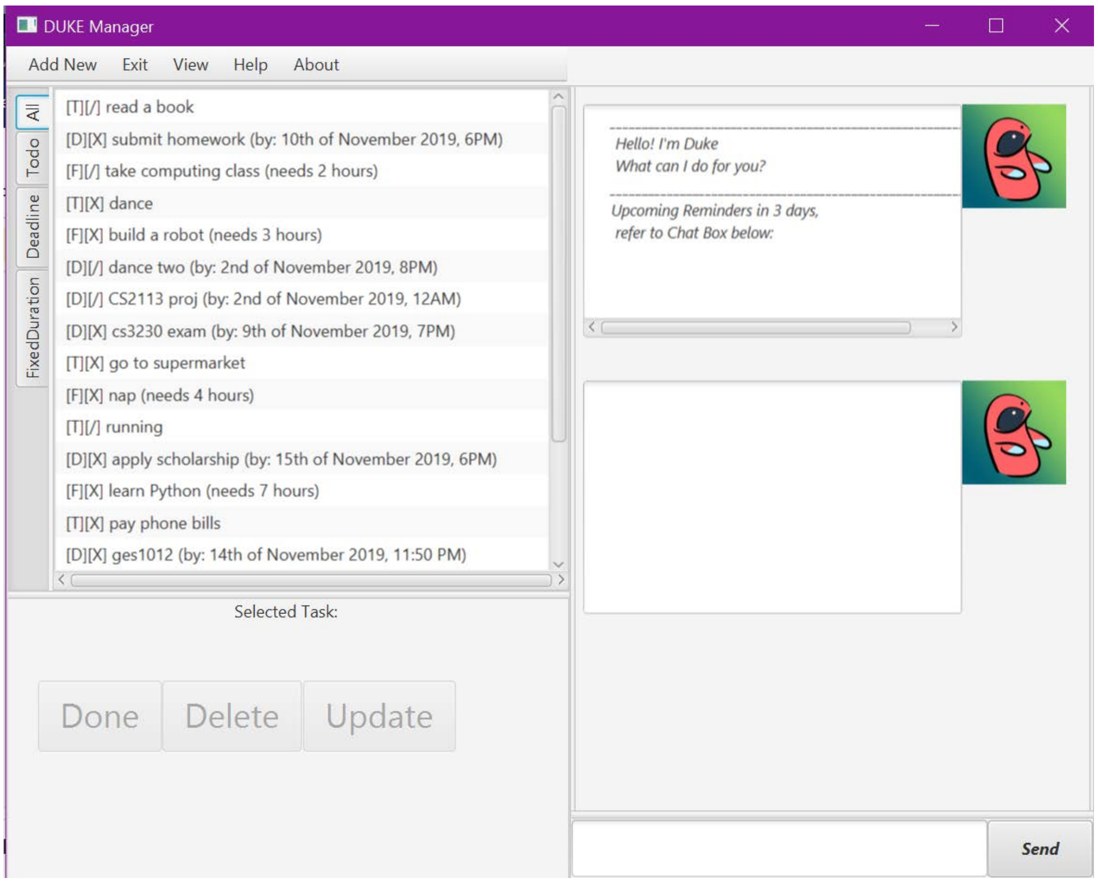
PROJECT: Duke Manager

Overview

DUKE Manager is an electronic handbook (desktop application) which is resulted from extending the project from DUKE in CS2113, using Java and Object-Oriented Programming. Our team of 4 students and I decided to create this desktop application to allow undergraduate engineering students to manage school-related tasks, store contact information of their peers and professors, and keep track of their own budget.

DUKE Manager has a user-friendly Graphical User Interface (GUI) which helps users to quickly familiarize and be comfortable for daily use. Also, it has an option for users who prefer Command Line Interface style by using the text box.

This is what our project looks like:



[T][/] read a book
[D][X] submit homework (by: 10th of November 2019, 6PM)
[F][/] take computing class (needs 2 hours)
[T][X] dance
[F][X] build a robot (needs 3 hours)
[D][/] dance two (by: 2nd of November 2019, 8PM)
[D][/] CS2113 proj (by: 2nd of November 2019, 12AM)
[D][X] cs3230 exam (by: 9th of November 2019, 7PM)
[T][X] go to supermarket
[F][X] nap (needs 4 hours)
[T][/] running
[D][X] apply scholarship (by: 15th of November 2019, 6PM)
[F][X] learn Python (needs 7 hours)
[T][X] pay phone bills
[D][X] ges1012 (by: 14th of November 2019, 11:50 PM)

Hello! I'm Duke
What can I do for you?

Upcoming Reminders in 3 days,
refer to Chat Box below:

My role was mainly to create and implement an Expenditure/Budget Tracker to Duke Manager, which will be highlighted in the sections to come. Amongst the Tracker, I also contributed to sub-features such as backup, and Friendlier Syntax.

Summary of contributions

Enhancement Added: Expenditure Management
What it does: This enhancement allows Duke Manager to be able to store and track the user's budget and expenditures.
Justification: As an academic student, some students would like to monitor their expenditures during the academic term so that they are able to flexibly manage their remaining budget. Therefore this enhancement will be useful for the users in general as a quick way to note their budget.
Highlights: The students can have the ability to reset their budget, or quickly store a budget without any description so that they don't have to spend so much time to ponder about what to write for their life choices.

Other Enhancements Added:
Backup Feature: Upon entering the backup command, users will be able to access the data of Duke Manager for exporting/importing of Duke Files.
Friendlier Syntax: For longer commands, there are shorter alternatives that can be called by the user, such as add contacts can be called by ac as well. Some commands can also be called by other terms which have similar meanings, such as bye and exit.

Code contributed [Project Code Dashboard]

Budget Tracking Related Code: [Budget Storage] [Budget List], [Add Budget Command], [View Budget Command], [Reset Budget Command]

Backup Related Code: [Backup Command]

Other contributions:
Project management:

- Added initial user stories into Github issues, and labelled most of the issues.

Enhancements to existing features:

- Improved exiting of Duke Manager such that it can save all the files used by Duke Manager. – Pull Request #157 ,
- Integrated the Sample codes so that jar file can automatically populate the data files if no files are present (at first launch) – Pull Request #222
- Improved DukeLogger by properly closing handles opened. – Pull Request #236
- Optimised exiting of Duke Manager such that it will save upon pressing Alt + F4 & the "X" button – Pull Request #228, #238

Documentation:

- Created the initial skeleton of the guides on Github (AboutUs, Developer Guide (Scrapped) and User Guide (Scrapped)).
- Created the initial class and architecture diagrams for edit and reuse

Tools:

- Integrated Travis CI into project, to ensure that minimal code quality has been achieved and that test cases are passed. - Pull Request #96

Main contributions to User guide [Add in the User guides]

My main contributions of the [User Guide] are the following:

- Expenditure Tracker Section
- Miscellaneous Inputs - Backing up

The following paragraphs are extracted from my contributions in the User guide:

3.2. Expenditure Tracker

This is a simple expenditure tracker that users can use to track their expenses in school. The initial budget to be entered by the user can either be their initial budget (amount in the bank, or monthly allowance), or the limits they set for themselves (Spending only $250 per month, etc)

The budget tracker commands starts with “budget”.

3.2.1. Create a new budget : new

Creates a new budget if there is no budget created. Upon creation of the new budget, any preexisting budget will be cleared.

Example:
Let’s say you want to create a new budget for the new school term.

- budget new 1000

Format: `budget new <amount>`

- The amount refers to the amount of money that you either currently have, or am limiting yourself to spend.
- Amount is limited to between -999,999 and 999,999.
- All previous entries will be cleared upon entering this command.

3.2.2. Deducts expenses to budget : minus

Deducts the expenses from current available budget, with an optional description.
If the user does not input any description, it will input "No Description" instead.

Example:
Let’s say you spent some money to purchase for an E-scooter.

- budget minus 999 Bought an E-scooter

Format: `budget minus <amount> <(Optional)Description>`

- Amount refers to the amount spent.
- Amount is limited to between -999,999 and 999,999.
- Description is optional; it acts like a note on what is the amount spent on.
- If there are no descriptions detected, it will input the remarks as “No Description”.

- The command “minus” can be replaced with a “-”. The spacing between - and <amount> should still be there.
    - Eg: budget - 999 Bought an E-scooter

3.2.3. Add earnings to budget : add

Adds the earnings to the current available budget, with an optional description.

Example:
Let’s say you spent some money to purchase for an E-scooter.

- budget add 400 Sold an E-scooter

Format: `budget add <amount> <(Optional)Description>`

- Amount refers to the amount earned or obtained.
- Amount is limited to between -999,999 and 999,999.
- Description is optional; it acts like a note on what is the amount spent on.
- If there are no descriptions detected, it will input the remarks as “No Description”.
- The command “add” can be replaced with a “+”. The spacing between + and <amount> should still be there.
    - Eg: budget + 400 Sold an E-scooter

3.2.4. View currently available budget : view

Shows the user the current available budget, as well as the total earnings and expenses recorded.

Format: `budget view`

3.2.5. Resets budget : reset

Resets the budget list with the initial input being the one that is defined. The existing list will be cleared as well.

Example:
Let’s say you want to reset your monthly allowance for the new year.

- budget reset 1000
  Resets the budget to $1000

Format: `budget reset <amount>`

- The amount refers to the amount of money that you either currently have, or am limiting yourself to spend.
- Amount is limited to between -999,999 and 999,999.
- All previous entries will be cleared upon entering this command.

3.2.6. Undoes budget : undo

Undoes the latest entry in the budget list and updates the budget accordingly.

Format: `budget undo`

Main contributions to developer guide

My main contributions of the [Developer Guide] are the following:

- Backup Feature
- Budget Feature

The following paragraphs are extracted from my contributions in the User guide:

3.2. Backup Feature

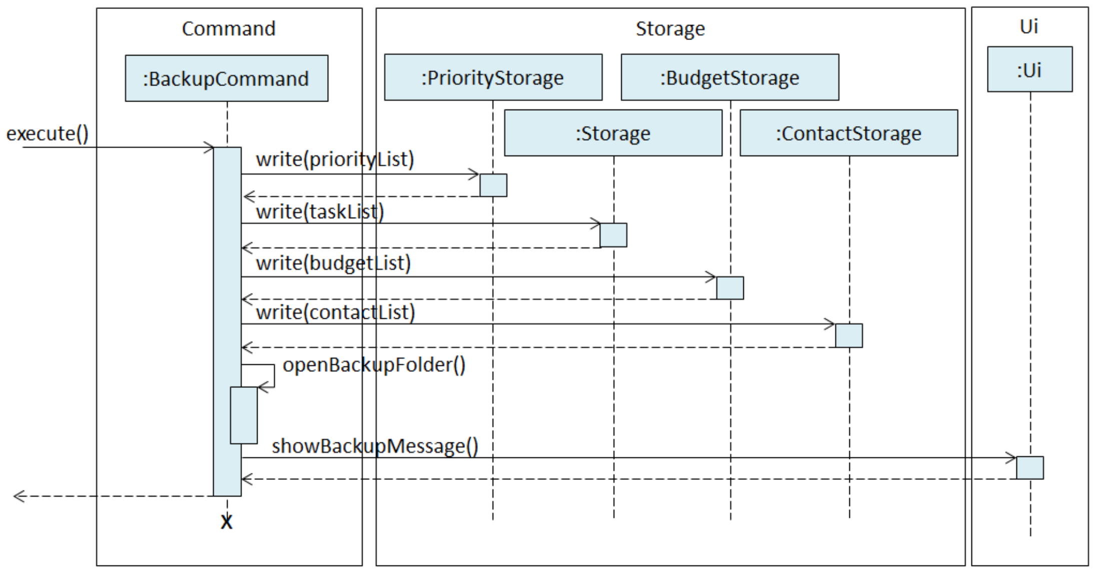
The backup feature will allow the user to manage the data files of Duke Manager. This will save the current state of the application, and launch the file location in the user's file explorer.

3.2.1. Implementation of Backup feature

The feature is activated by the user entering "backup" into Duke Manager. Upon entering the class BackupCommand, the following steps will be done:

1. It will write the current state of the lists in Duke Manager by calling storage.write(), priorityStorage.write(), budgetStorage.write() and contactStorage.write().
2. After writing successfully into the files, it will call the method openBackupFolder, which opens the directory of the saved data files in file explorer.
3. After opening the file, it will call showBackupMessage, where Duke Manager shows the user that backup is complete and that the file location is opened by Duke Manager.

Below is the sequence diagram for the backup feature.



3.2.2. Design Considerations for backup feature

<u>Aspect: Process of backing up.</u>

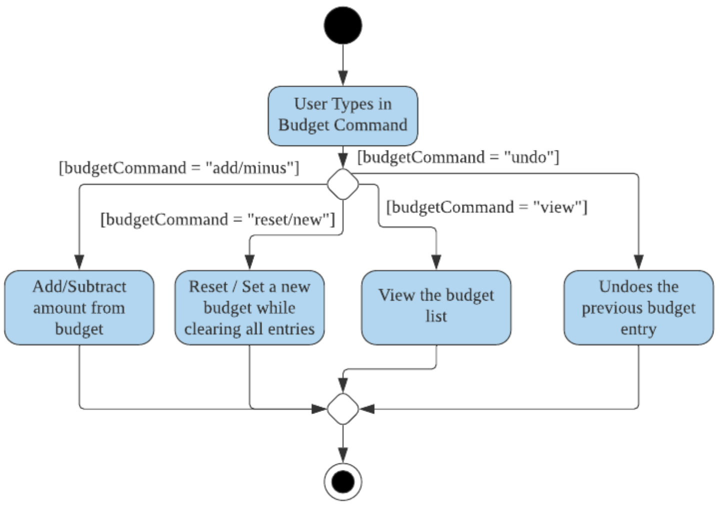
As the normal case of backing up is for the importing / exporting of the data in Duke Manager, it is more appropriate to have the ease of opening the file location immediately after the command backup, instead of manually trying to find where the files are at after typing backup. Therefore it was implemented so that upon pressing backup, the user can have access to the data files directly.

3.3. Budget Feature

Duke Manager also has a built in budget storing feature for engineers to monitor their budget. They can add or subtract from their budget, as well as view and reset their budget.

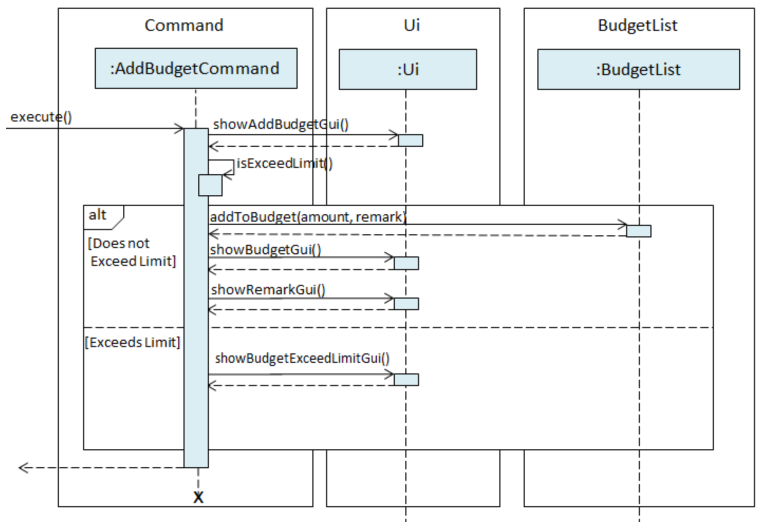
3.3.1. Implementation

The following diagram shows the activity diagram of the process of calling the budget in Duke Manager.

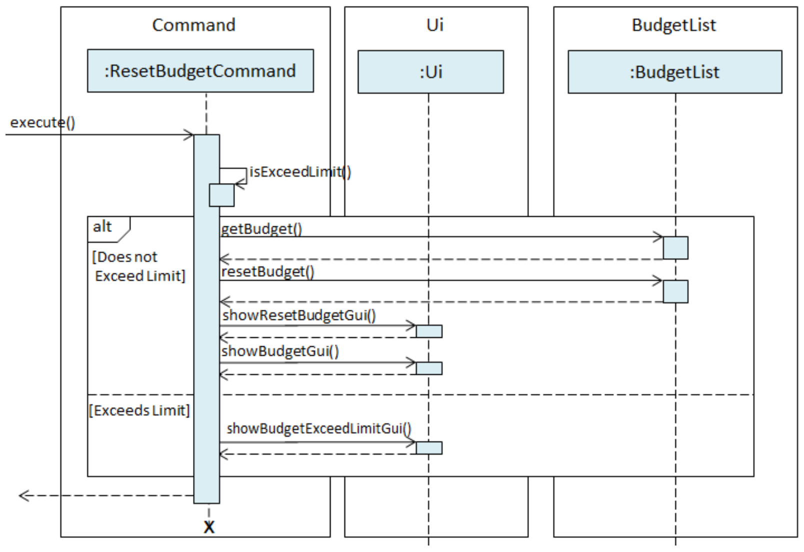

User Types in Budget Command
[budgetCommand = "add/minus"]
[budgetCommand = "undo"]
[budgetCommand = "reset/new"]
[budgetCommand = "view"]
Add/Subtract amount from budget
Reset / Set a new budget while clearing all entries
View the budget list
Undoes the previous budget entry

The feature is triggered with the starting word “budget”. It will then check for the next word in the string command, whether it is one of the valid terms such as “view” and “new”. Upon validating, it will enter the different Commands (AddBudgetCommand, ViewBudgetCommand, ResetBudgetCommand, UndoBudgetCommand) where it will perform the necessary actions for the stated command.
The feature is triggered with the starting word “budget”. It will then check for the next word in the string command, whether it is one of the valid terms such as “view” and “new”. Upon validating, it will enter the different Commands (AddBudgetCommand, ViewBudgetCommand, ResetBudgetCommand, UndoBudgetCommand) where it will perform the necessary actions for the stated command.
There are 3 cases to consider, alongside their sequence diagram:

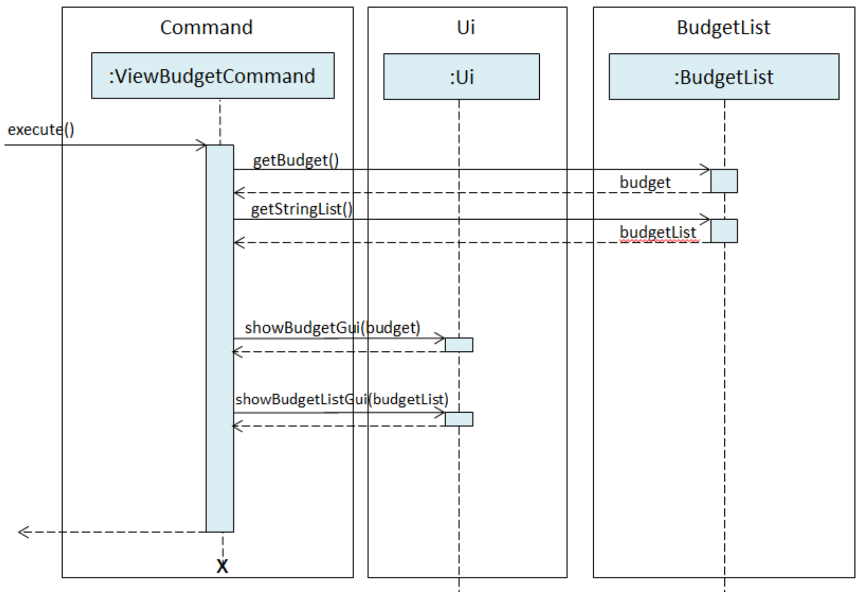
1. Adding/Subtracting from budget: Upon receiving the command, Duke Manager will enter the class AddBudgetCommand(), and shows the user the amount before adding, and after adding. It will also store the change to the existing BudgetList



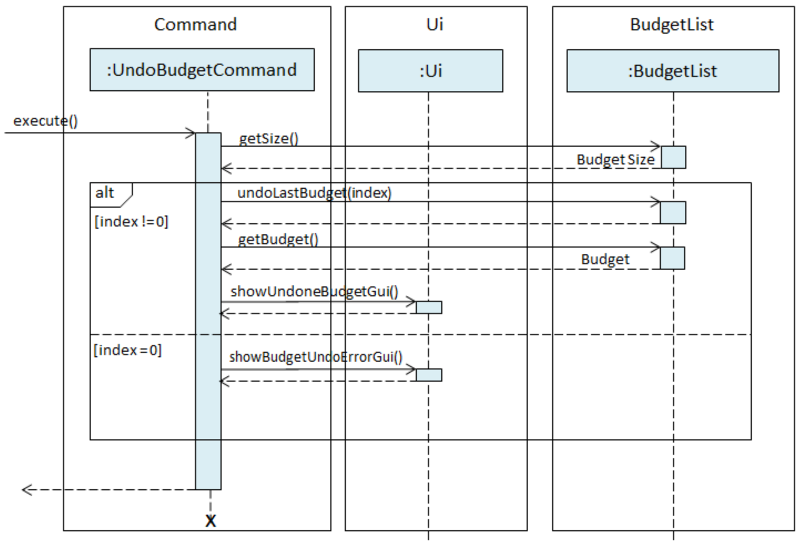
2. Reset/New budget: Upon receiving the command, Duke Manager will show the user its previous budget, update the budget with the input amount inside BudgetList and then show the user the new budget.



3. View budget: Upon receiving the command, Duke Manager will show the budget to the user by calling showBudget().



4. Undo budget: Upon receiving the command, Duke Manager will proceed to remove the latest budget in the list from the stack, recalculate the budget and then show the updated budget to the user.



3.3.2. Design considerations

<u>Aspect: Simplicity</u>

The choice of implementing just the addition, subtraction, resetting and undoing of budget is due to the consideration of making the budget tracking aspect more intuitive and user friendly. Due to the fast pace of our academic life, noting down of how much we spent should not be tedious, but straightforward and easy. Therefore a simple input for the amount would be ideal. The description is made optional to give a choice for users who do not require the details to be input.

Other Contributions

Additional contributions to the User guide:

- Miscellaneous Inputs – Friendlier Syntax
- FAQ & General Formatting

Additional contributions to the Developer guide:

- Initial Architecture and Class Diagrams creation
- Updated User Stories with issues in Github

Contributions to About Us:

- Roles & Responsibilities