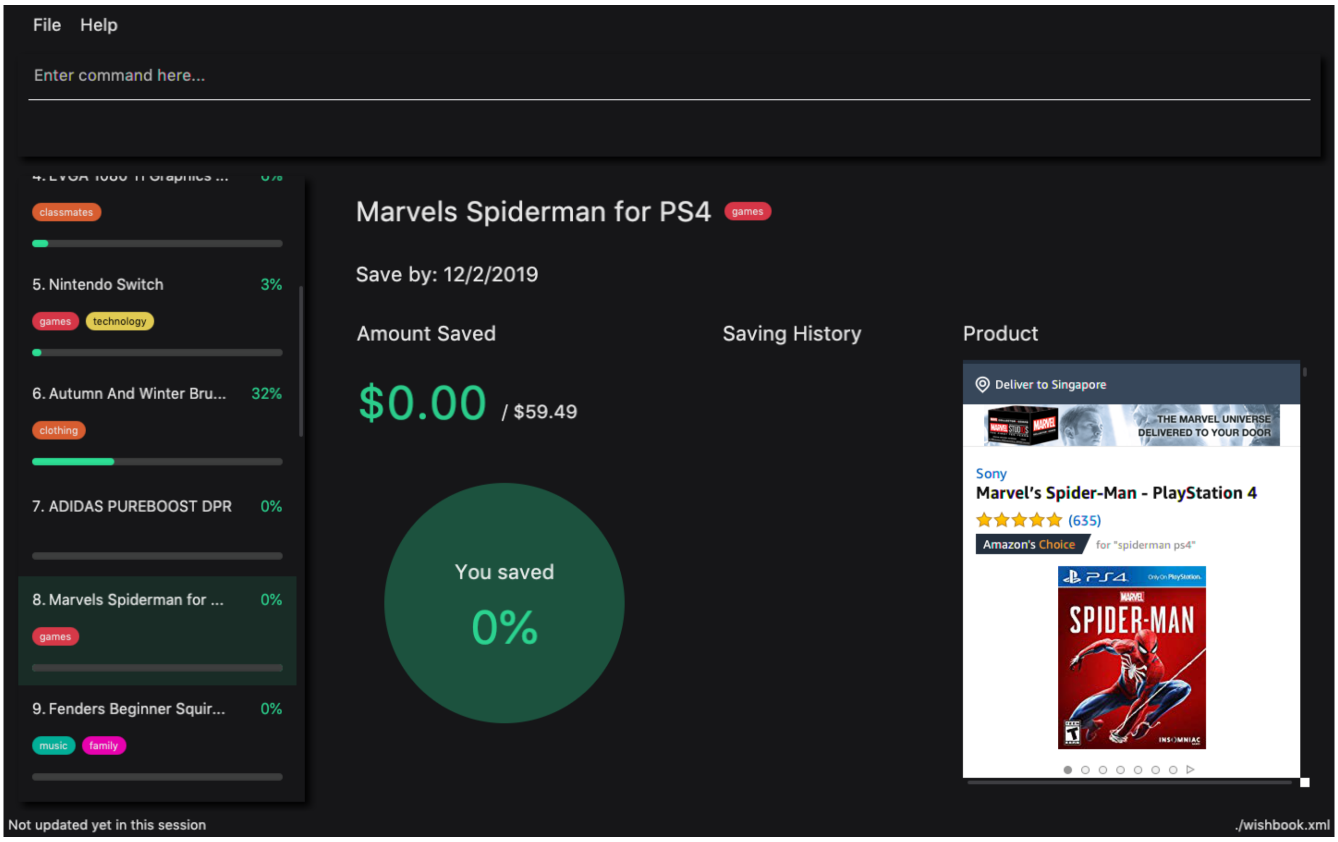
Lionel Tan - Project Portfolio

PROJECT: WishBook

Overview

WishBook is a desktop savings assistant application written in Java, built for people who want to set savings goals (Wishes) and keep track of their progress in saving toward those wishes. It is created by Team T16-1 for CS2103T: Software Engineering module offered by School of Computing, National University of Singapore, in AY2018/2019 Semester 1. The image below shows the main layout of our application:



WishBook aims to promote responsible saving and spending among users, by helping them be more prudent with their income and encouraging regular, incremental savings to attain their wishes. The source code of WishBook can be found here.

Being in charge of the Logic Component and documentation, I was tasked with implementing new commands, while also making sure that all changes made by our team were consistently documented in our User Guide and Developer Guide. This project portfolio consists of the contributions I have made to WishBook over the semester.

Summary of contributions

Given below are the contributions I have made to WishBook's development. Major components were delegated to each member of the team, and I was tasked with the Logic component.

Major Enhancements

- **Move Command**: added Move Command
    - **What it does**: Allows users to move funds between wishes, or to transfer between wishes and the unused funds.
    - **Justification**: Previously, to reallocate funds from one wish to another, the user had to save a negated amount from one of their wishes, and then save that positive amount into the targeted wish. It can be quite troublesome to shift funds between wishes, as the user would have to type two separate commands. As such, I developed the move command that allows the user to transfer funds between wishes directly in a single command.
- **Save to unused funds**:
    - **What it does**: Allow users to save toward unused funds directly. Also, any excess saved amount will be channelled automatically to unused funds.
    - **Justification**: Previously, if the input amount causes the wish's saved amount to exceed its price, the excess amount would be stored inside that wish. However, this excess amount cannot be reallocated, since the wish has already been fulfilled. Thus, I modified the Save command to channel any excess saved amount to an UnusedFunds within WishBook. The money inside the UnusedFunds can later be redistributed to other wishes, via the Move Command.

Minor Enhancements

- **Savings History**: add option to show savings history for History Command.
- **List**: add option to display only completed or uncompleted wishes.

Other Contributions

- Shorthand alias: Add shorthand equivalent alias for commands.
    - Useful to increase speed and efficiency while using WishBook
    - users would now be able to type a command with only 1 or 2 characters, instead of typing the whole word. (Pull Request #3)
- Documentation:
    - Documented new functions added to User Guide.
    - Consistently updated the User Guide to reflect the current function usage in our application.
    - Refactored Developer Guide to make it specific to our WishBook implementation.
- Refactored code and documentation:
    - Contributed regularly to refactoring the code from the previous AddressBook-Level 4 application.
    - Refactor terms specific to AddressBook-Level 4, to suit our current WishBook application
- Reported bugs and issues: #61 #90

Code contributed: Project Code DashBoard Link

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.

Save money for a particular wish: `save`

Channel a specified amount of money to savings for a specified wish.
Format: `save INDEX AMOUNT`

- `INDEX` should be a positive integer 1, 2, 3… no larger than the number of wishes.
- If `INDEX` is 0, `AMOUNT` will be channelled directly to `unusedFunds`.
- If `AMOUNT` saved to `INDEX` is greater than the amount needed to fulfil that wish, excess funds will be channelled to `unusedFunds`.
- If `AMOUNT` is negative, money will be removed from amount saved for that wish.
- `AMOUNT` will not be accepted if:
    - `AMOUNT` brings the savings value for that wish to below 0.
    - The wish at `INDEX` is already fulfilled.

Examples:

- `save 1 1000`
  Attempt to save $1000 for the wish at index 1.
- `save 1 -100.50`
  Attempt to remove $100.50 from the savings for the wish at index 1.
- `save 0 100.50`
  Attempt save $100.50 to `unusedFunds`.

Move money between wishes: `move`

Moves funds from one wish to another.
Format: `move FROM_WISH_INDEX TO_WISH_INDEX AMOUNT`

- `FROM_WISH_INDEX` and `TO_WISH_INDEX` should be a positive integer 1, 2, 3… no larger than the number of wishes
- If `FROM_WISH_INDEX` is 0, `AMOUNT` will be channelled from `unusedFunds` to `TO_WISH_INDEX`.
- If `TO_WISH_INDEX` is 0, `AMOUNT` will be channelled from `FROM_WISH_INDEX` to `unusedFunds`.
- `AMOUNT` from `unusedFunds` will only be successfully channelled if the exact amount requested is present in `unusedFunds`.
- If `FROM_WISH_INDEX` equals `TO_WISH_INDEX`, both indexes will not be accepted.
- If `AMOUNT` saved to `TO_WISH_INDEX` is greater than the amount needed to fulfil that wish, excess funds will be channelled to `unusedFunds`.
- `AMOUNT` will not be accepted if:
  - `AMOUNT` is negative.
  - `AMOUNT` brings the savings amount of wish at `FROM_WISH_INDEX` to below 0.
  - Either wish at `FROM_WISH_INDEX` or `TO_WISH_INDEX` is already fulfilled.

NOTE Index 0 is specially allocated for `unused funds`. Excess funds when user attempts to save to a wish will be automatically allocated to `unusedFunds`. The user can also choose to channel funds from `unusedFunds` to a valid wish.

Examples:

- `move 1 2 10`
  Attempt to move $10 from the wish at index 1 to the wish at index 2.
- `move 0 1 10`
  Attempt to move $10 from `unusedFunds` to the wish at index 1.
- `move 1 0 10`
  Attempt to move $10 from the wish at index 1 to `unusedFunds`.

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.

Save Amount feature

Current Implementation

The Save Amount feature is executed through a `SaveCommand` by the user, which after parsing, is facilitated mainly by the `ModelManager` which implements `Model`. Wish stores the `price` and `savedAmount` of `Wish`, helping to track the progress of the savings towards the `price`. Meanwhile, WishBook stores an `unusedFunds`, which is an unallocated pool of funds that can be used in the future. After adding a saving, the `filteredSortedWishes` in `ModelManager` is updated to reflect the latest observable WishBook.

Given below is an example usage scenario and how the SaveCommand behaves at each step:

Step 1. The user executes `save 1 10`, to save $10 into an existing wish with `Index` 1 and `Price` $15. The $10 is wrapped in an `Amount` and a `SaveCommand` instance is created with the `Amount`. `Amount` is then used to make an updated instance of the `Wish` at index 1 whose `SavedAmount` will be updated. `Model#updateWish` is then called to update this wish with the old one in `WishBook`.

NOTE The `Index` of each `Wish` is labelled at the side of the app.

The resultant wish will have the following properties:

- Name: *1TB Toshiba SSD*
- SavedAmount: 10.00
- Price: 15.00
- Date: 20/4/2018 (20th April 2018)
- URL: `empty string`
- Remark: `empty string`
- Tags: `none`
- Fulfilled: `false`
- Expired: `false`

NOTE `Amount` to be saved can be a negative value where it would mean a withdrawal of money from a particular wish.

NOTE `SavedAmount` of a wish cannot be negative. This means that an `Amount` cannot be negative enough to cause `SavedAmount` to be negative.

Step 2. The user decides to execute `save 1 10` again. However, SaveCommand checks that `savedAmount > price`. SaveCommand#execute creates a new updated `Wish` with `savedAmount = wish.getPrice()`.

The resultant wish will have the following properties:

- Name: *1TB Toshiba SSD*
- SavedAmount: 15.00
- Price: 15.00
- Date: 20/4/2018 (20th April 2018)
- URL: `empty string`
- Remark: `empty string`
- Tags: `none`
- Fulfilled: `true`

- Expired: `false`

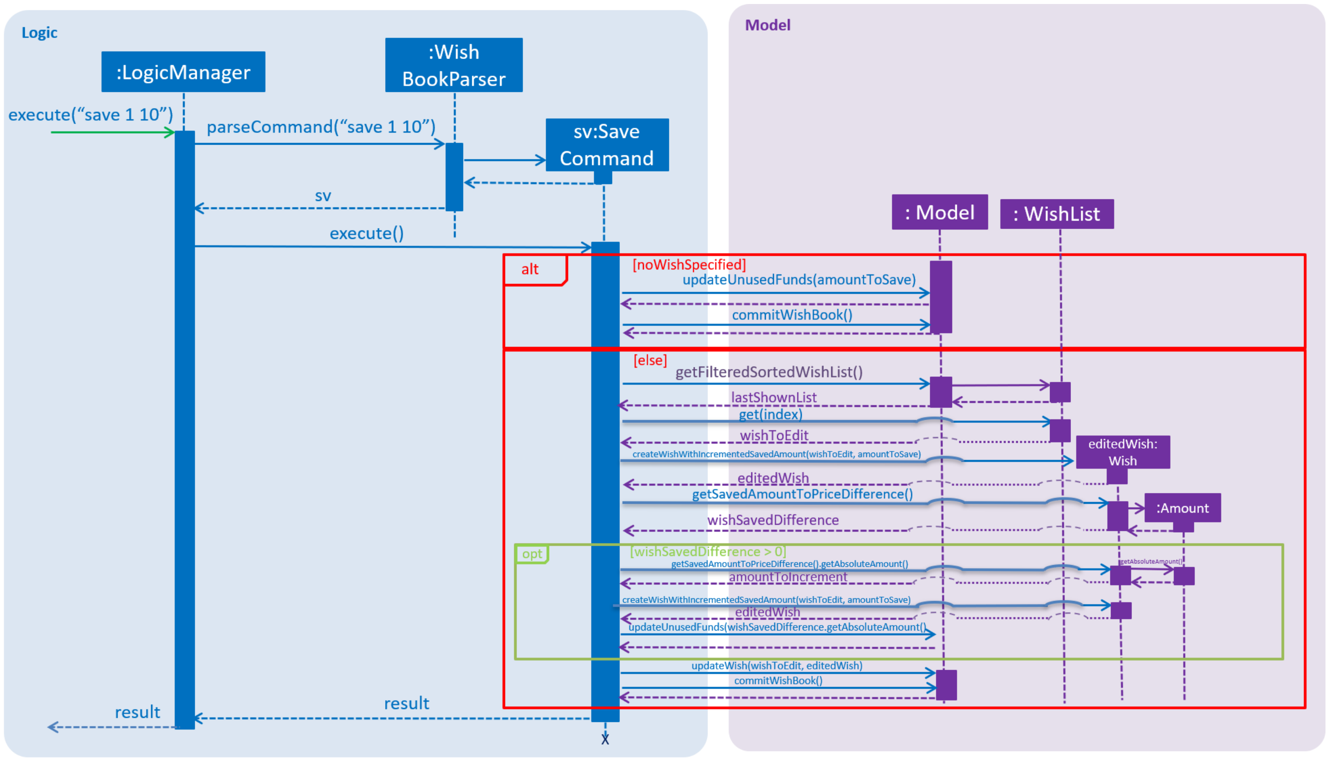
Step 3. The excess amount of $5 is stored in a new `Amount` variable `excess`. SaveCommand#execute then calls Model#updateUnusedFunds(excess) to update the `unusedFunds` in WishBook.

In WishBook, the result would be:

- unusedFunds: 5.00

Step 4. The user tries to execute `save 1 10` again. However, since the value for Wish#isFulfilled is true, the amount will not be saved. SaveCommand#execute will throw a CommandException, with the message "Wish has already been fulfilled!".

The following sequence diagram shows how the save operation works:



Logic
Model
:LogicManager
:Wish BookParser
sv:Save Command
: Model
: WishList
execute("save 1 10")
parseCommand("save 1 10")
sv
execute()
alt
[noWishSpecified]
updateUnusedFunds(amountToSave)
commitWishBook()
[else]
getFilteredSortedWishList()
lastShownList
get(index)
wishToEdit
createWishWithIncrementedSavedAmount(wishToEdit, amountToSave)
editedWish: Wish
editedWish
getSavedAmountToPriceDifference()
:Amount
wishSavedDifference
opt
[wishSavedDifference > 0]
getSavedAmountToPriceDifference().getAbsoluteAmount()
getAbsoluteAmount()
amountToIncrement
createWishWithIncrementedSavedAmount(wishToEdit, amountToSave)
editedWish
updateUnusedFunds(wishSavedDifference.getAbsoluteAmount())
updateWish(wishToEdit, editedWish)
commitWishBook()
result
result
X

Design Considerations

Aspect: Data structure to support the unusedFunds feature

- **Alternative 1 (current choice):** Store it in a `SavedAmount` variable in `WishBook`.
  - Pros: Easy to implement.
  - Cons: More methods needed when needing to move funds from `unusedFunds` to other wishes.
- **Alternative 2:** Store it as a pseudo wish with index 0.
  - Pros: It can be treated as another `wish`, hence existing methods can be used without needing to create much more new ones.
  - Cons: Requires dealing with an extra wish that has to be hidden on the `WishListPanel` and displayed separately on the UI. We must remember to skip this wish in methods that involve displaying the WishList.

[Proposed] Savings Notifications

Justification

Some users may have many wishes, all of which have a different targeted date of completion and different price. It may thus be difficult for users to keep track of how much they need to consistently save to fulfil their various wishes on time. This Savings Notification feature will allow users to opt for daily/weekly/monthly notifications for each specific wish, reminding them of the amount that they need to save at the beginning of the chosen time period. This will help users to consistently save towards their wishes.