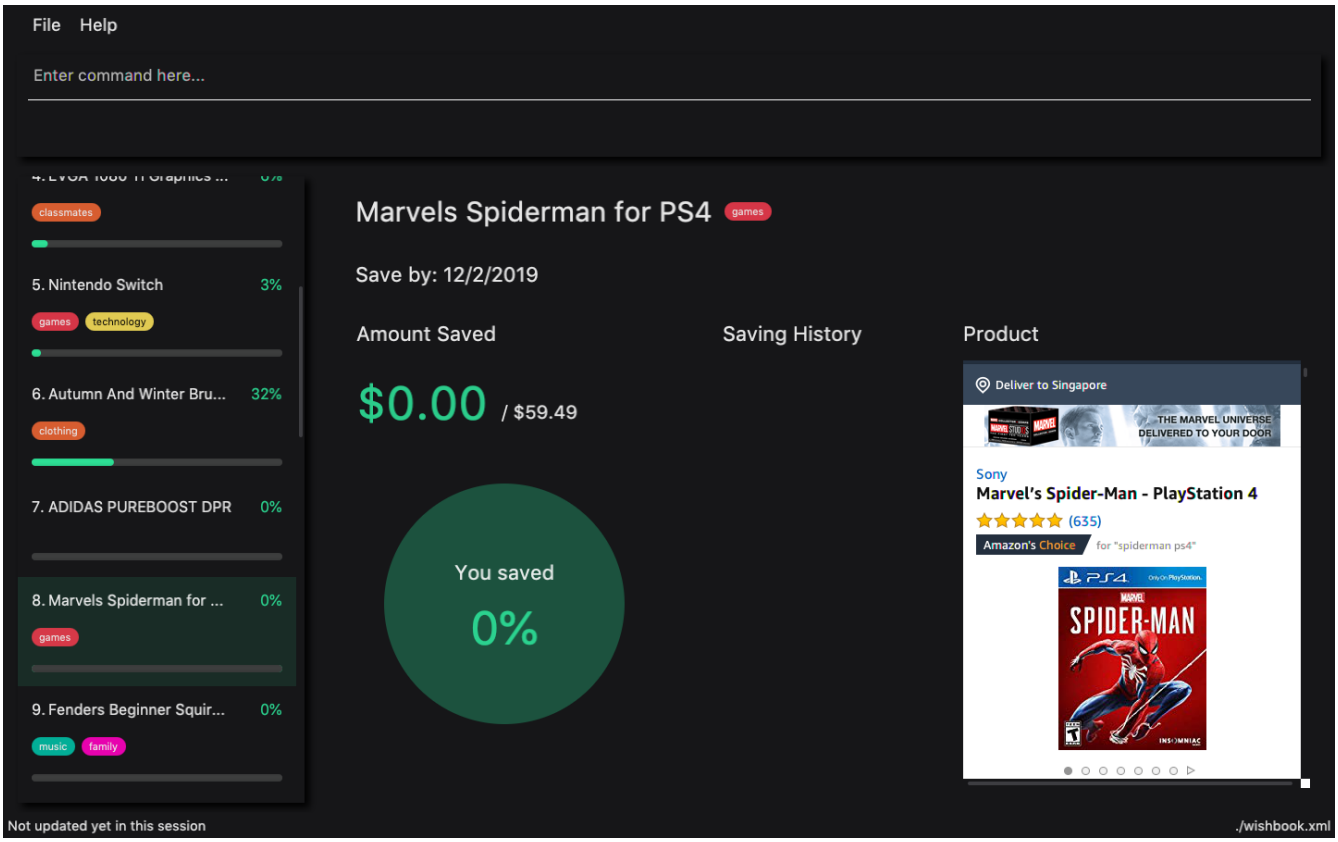
Lim Ji Ho (jiholim) - Project Portfolio

PROJECT: WishBook

Overview

WishBook is a desktop saving assistant application built for people who want to set saving goals and keep track of their progress. WishBook makes saving money easy, motivating, & fun and aims to help users to complete their saving goals by making sure they stick to the target and maintain progress regularly. WishBook boasts a beautifully designed Graphical User Interface (GUI) and makes use of a Command Line Interface (CLI) for most of the user interaction. The application is written in Java and contains about 10 kLoC.

Below is a high definition product preview photo.



Summary of contributions

- **Major enhancement**: A complete redesign of GUI (UI & UX) and front-end development
  - What it does: Displays an intuitive and data-rich graphical user interface that achieves the product's goal and delivers smooth user experience when the features are being used.
  - Justification: The redesign of the application is necessary as it unifies the different set of features that are built into a well-integrated product. It improves the product significantly because user can reap the maximum intended benefits of the features.
  - Highlights: This enhancement affects existing ui components and elements to be added in future. It required an in-depth analysis of design alternatives. The implementation too was

challenging as it required changes to existing ui-related codebase

- **Minor enhancements**:
  - Devised and built appropriate event handlers to present changes in model on ui
  - Various Bug Fixes
  - Update HelpWindow display
- **Code contributed**: Project Code Dashboard link
- **Other contributions**:
  - Project management:
    - Managed milestones `v1.1` - `v1.4` and one `v1.5` (release) on GitHub
    - Helped with the issue tracking using labels and assignees on Github
  - Documentation:
    - Updated README file to add descriptions of the application with relevant mockup picture (Pull requests #7)
    - Updated User Guide and Developer Guide for the features I added.
  - Community:
    - Review pull requests weekly (see here)
    - Reported bugs and gave suggestions for other group's project.

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.

List wishes : `list`

Shows a list of all the wishes you have set, sorted by date by default, based on the given filter. If no filter is specified, all wishes in the WishBook will be listed.
Format: `list [FILTER]`

- `list`
  Lists all the wishes in the WishBook.
- `list -c`
  Lists all the completed wishes in the WishBook.
- `list -u`
  Lists all the uncompleted wishes in the WishBook.

> - Only wishes in the current state of the wishbook will be listed.
> - Deleted wishes will not be displayed.

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.

List feature

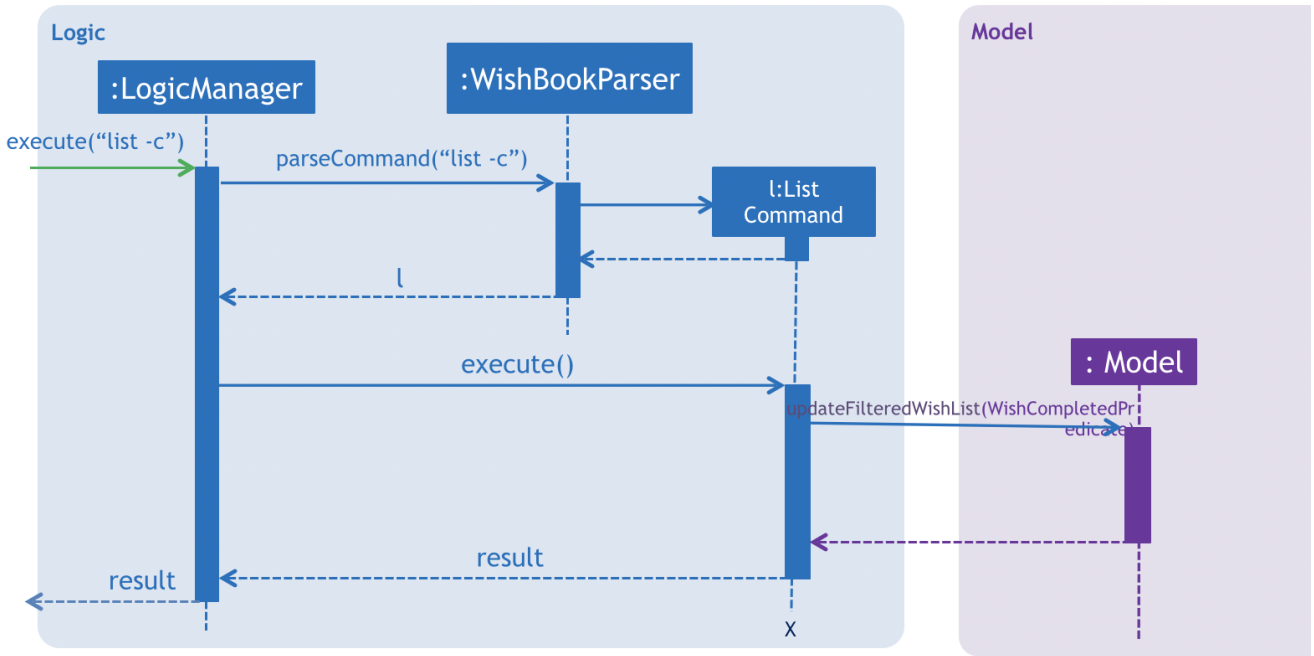
The `list -c and list -u` command allows the user to view the list of all wishes, completed and ongoing, respectively. A wish is completed if the savedAmount is greater or equal to the price of the wish.

Current Implementation

Given below is an example usage scenario and how the list overdue mechanism behaves at each step:

1. The user executes the command `list -c`.
2. `model.updateFilteredWishList()` will update the wish list with `WishCompletedPredicate` as the parameter (boolean). `wish.isFulfilled()` is called to check whether the wish is completed or not.
3. The updated wish list would be reflected on the UI to be displayed to the user.

The following sequence diagram shows how the Wish Detail Panel displays its updated content:



Redesign of User Interface

The UI has been redesigned to implement the following UI components required for WishBook:

- Command Box
- Wish List Panel
- Wish Detail Panel

Wish List Panel

The Wish List Panel consists of a list of Wish Card which contains 4 UI elements:

- `WishCard#nameLabel` - A `Text` element that displays the wish's name.
- `WishCard#progressLabel` - A `Text` element that displays the wish's saving progress in percentage format.
- `WishCard#tags` - A `FlowPane` element that contains a `Text` element which displays the wish's assigned tags.
- `WishCard#progressBar` - A `progressBar` element that visually presents the percentage of the wish's current saving progress.

Whenever the user adds a new wish or edits an existing wish, a new WishCard containing the wish will be added to the Wish List Panel or the content in the existing WishCard will be updated respectively.

The user will be able to view the wish's current saving progress both in terms of text on the progressLabel (e.g. '80%') and the progressBar. Also, the user will be able to see all the tags he/she assigned to categorize the wish.

Problem with the old design

The UI (MainWindow) constructs the `WishListPanel` by obtaining an `ObservableList` of wish cards from `Model`, this list is assigned when UI starts, and will never be re-assigned. The UI "observes" the list and updates when it is modified.

This approach works well for the `WishListPanel` because WishBook contains only 1 list of wish cards. However, the saving history list in the `WishDetailPanel` can not be updated in the same manner because Model component will change its card list's reference when a user adds a new wish or updates the content of the wish.

In this case, the `WishDetailPanel` in UI will not be updated because the card list of which UI has reference to is actually not changed.

Design considerations

- **Alternative 1 (current choice)**: Have a wishList in `Model` and keep it updated with the current list of cards
    - Explanation: The UI needs only 1 reference to this `wishList`, each time user executes any

changes, `wishList` is cleared and the new list of cards is copy to the `wishList`.

- Pros: The structure of `Model` and UI component needs not be changed
- Cons: Need to keep a copy of the current card list, copying the whole list of cards for each command operation has bad effect on performance .

- **Alternative 2**: Model component raises an event when its current card list's reference is changed
  - Explanation: When user adds a new wish or executes save, `Model` will raise an event (`WishPanelUpdatedEvent`), which is subscribed by UI, then UI can re-assign its list of cards and update the cards panel accordingly.
  - Pros: Better performance
  - Cons: Need to re-design `Model` and UI components

Wish Detail Panel

The Wish Detail Panel consists of 3 UI sub-components:

- `WishDetailSavingAmount` that contains `Text` elements to display price and the saving progress of the wish
- `WishDetailSavingHistory` that contains a `List` of history of saving inputs of the wish
- `WishBrowserPanel` that displays `WebView` of the URL of the wish.

Whenever the user adds a new wish or edits an existing wish, the content of the wish in Wish Detail Panel will be updated. The user will be able to view the wish's current saving progress and the history of his/her saving inputs of the wish in the list format. Also, the user will be able to browse through the wish's product page via its assigned URL.

Current Implementation

Every time a new Wish is added or an existing wish is updated by the commands such as save, it raises a `WishDataUpdatedEvent`. The UI will then handle that event and update the `WishDetailPanel` with the new version of wish.

Given below is an example usage scenario and how the WishBook behaves and `WishDetailPanel` is updated at each step:

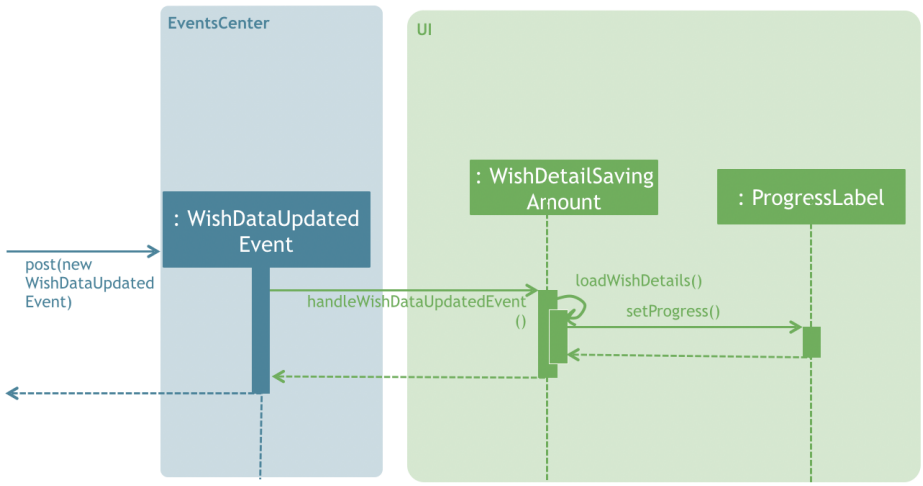
1. The user executes the command `save 1 1000`.

NOTE	If a command fails its execution, WishDataUpdatedEvent will not be posted.

1. The save command updates the model with the new wish and raises a new `WishDataUpdatedEvent`.
2. `WishDetailSavingAmount` and `WishDetailSavingHistory` responds to the `WishDataUpdatedEvent` with `WishListPanel#handleWishUpdatedEvent()`.
3. The `WishDetailSavingAmount` updates the wish's current saving progress when `WishDetailSavingAmount#loadWishDetails` is called.

4. The progress is calculated from when `Wish#getProgress` is called. The value is saveAmount / price. Then the progress label for the wish is set to that fraction.
5. The `WishDetailSavingHistory` updates the wish's saving history list when `WishDetailSavingHistory#loadWishDetails` is called.
6. The saving history list is cleared.
7. The new set of history entry is retrieved from `wishTransaction#getWishMap` and the saved amount is calculated from subtracting previous saving amount from the next one.
8. The saving history list is now filled with the new list of updated saving history.

The following sequence diagram shows how the Wish Detail Panel displays its updated content:



Design Considerations

Aspect: How to update the progress and saving history on UI

- **Alternative 1 (current choice)**: Clear all the sub components and add new sub components accordingly
  - Pros: No matter which progress or history is changed, or what type of change (ie. delete, add, or edit), this change can be handled by the same method each time.
  - Cons: It is redundant to clear everything and replace them with new sub components.
- **Alternative 2**: Handle different kinds of changes to the progress or history lists.
  - Pros: It is a lot faster to only change the sub component that is affected.
  - Cons: There are too many cases for how the lists can be changed. (ie. a different change is needed for each of these cases: wish is deleted/edited/created/cleared, or a `wishTransaction` is deleted/added)