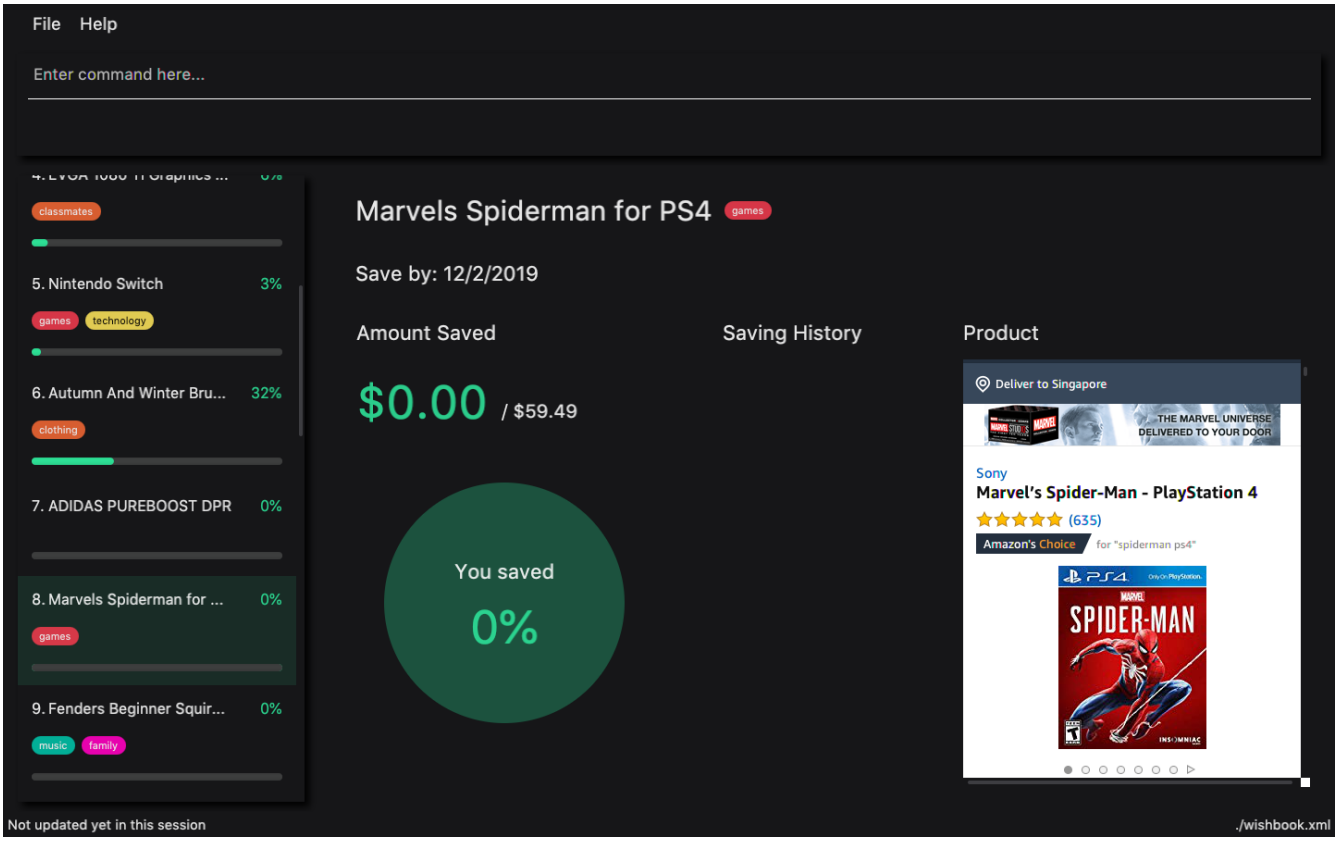
Dominic Chong (bannified) - Project Portfolio

PROJECT: WishBook

Overview

WishBook is a desktop application used to help one allocate one's disposable income to a number of desired products. WishBook enables users to better organize their desirables, allowing them to visualize their spendings in a pleasant way. The user interacts with it by using a Command Line Interface (CLI), and it has a GUI created with JavaFX. It is written in Java, and has about 10 kLoC.

WishBook was developed by the GaveSoals team, a team of students based in the School of Computing, National University of Singapore.



WishBook is an application that promotes sustainable and responsible consumerism, by having users be more frugal with their income while rewarding themselves at appropriate times, when they can afford to.

Wishbook is an application adapted from addressbook-level4, the original application developed by the se-edu team.

The source code of addressbook-level4 can be found here.

The main features of WishBook are:

- Add a Wish (of a desired item) to the WishBook
- Add funds to a Wish to make progress in fulfilling a Wish

- Add remarks to a Wish
- Add funds to WishBook without an explicit Wish
- Keep track of the user's progress to fulfilling a Wish, so that the user can buy it.
- Chang the details of a Wish after it has been added
- Find Wishes by name or tags
- View all completed or uncompleted Wishes
- Delete an unwanted Wish
- View of the Wish's product page (if applicable)
- Undo and Redo commands
- View history of savings

Summary of contributions

Mentioned below are the contributions I have made with regards to WishBook's development. Major components of the application were delegated to every person in the team, and I was tasked with the Model component.

- **Major enhancement**: Add Wish Command
  - What it does: allows the user to add a `Wish` by entering a product's details.
  - Justification: This feature is one of the core functionalities of the application, as a Wish is the primary entity that a User will interact with when using WishBook.
  - Highlights:
    - Since this feature is core to the application, defensive programming has to be practiced when implementing this feature, as the Wish will likely affect features being added in the future.
    - The Add Command also effectively creates Wish entries for the Wishbook to operate on, and is the entry point of the lifecycle of our application's usage, bugs occurring in this command will cascade to other Wish-related commands, hence its implementation has to be safe and well-tested.
- **Minor enhancements**:
  - Add an option for the user to state the Duration that a Wish will last, when adding a Wish with Add Command.
  - Change syntax of Save Command to make it more user friendly
  - Make Save Command show the change in SavedAmount of a wish from before
- **Code contributed**: Project Code Dashboard link
- **Other contributions**:
  - As Team Lead of GaveSoals:
    - Managed milestones `v1.1` - `v1.4` and one `v1.5` (release) on GitHub

- Handle distribution of tasks and features for the team
- Ensure that development is steady and milestones are met
- Manage issue tracker

◦ Enhancements to existing features:

- Add age option (for dueDate) when adding a Wish (Pull requests #74)
- Application-wide refactoring to suit our desired WishBook product (Pull requests #12)

◦ Community:

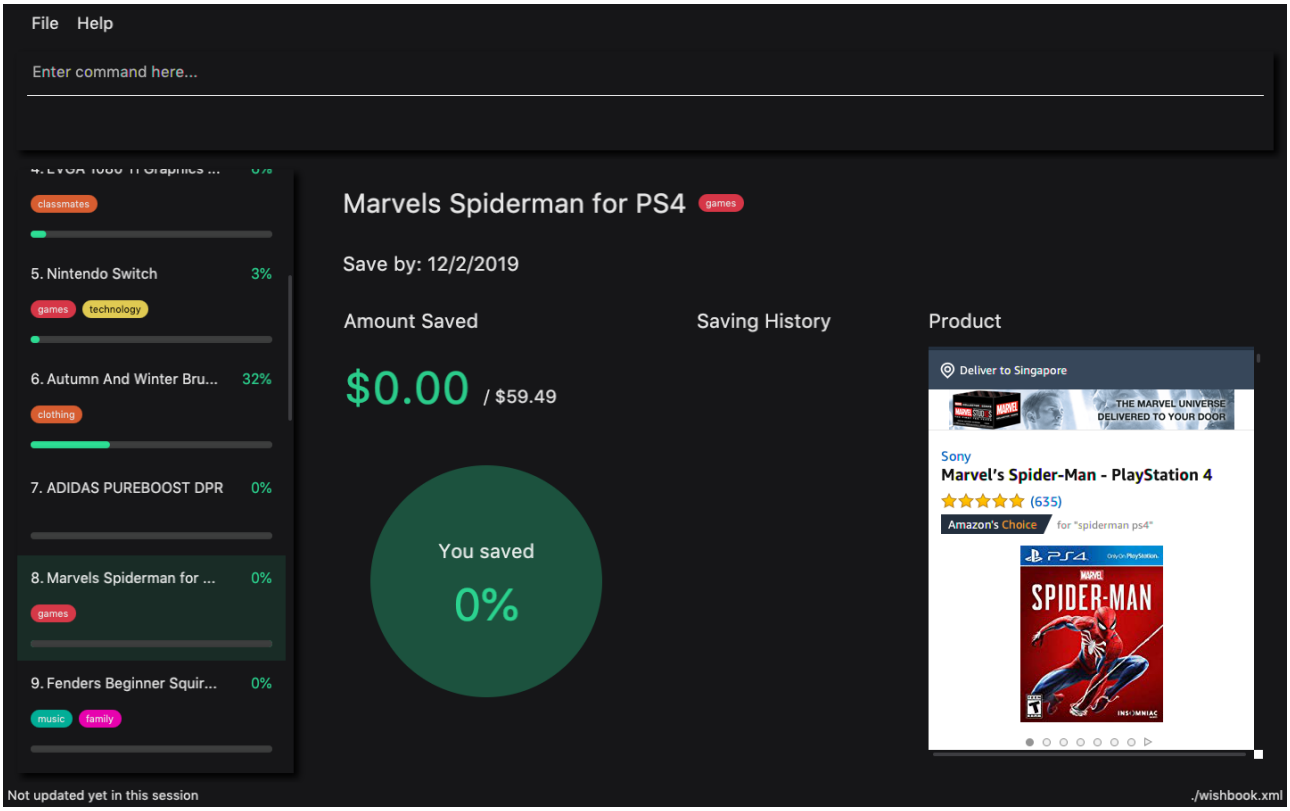
- PR reviewed (with non-trivial comments): #1, #7, #55
- Reported bugs for other teams 1, 2, 3

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.

Quick Start

1. Ensure you have Java version 9 or later installed in your Computer.
2. Download the latest `wishbook.jar` here.
3. Copy the file to the folder you want to use as the home folder for your WishBook app.
4. Double-click the file to start the app. The GUI should appear in a few seconds.



5. Type the command in the command box and press kbd:[Enter] to execute it.
   e.g. typing `help` and pressing kbd:[Enter] will open the help window.
6. Some example commands you can try:
   - `list` : Lists all the items you have set as wishes (sorted by due date).
   - `addn/uPhoneXX p/1000 a/5m` : adds an item "uPhoneXX" as a goal to be completed in 5 months.
   - `help`: displays list of command with usage.
   - `clear`: clears view
   - `exit` : exits the app
7. Refer to Features for details of each command.

Features

Command Format

- Words in `UPPER_CASE` are the parameters to be supplied by the user e.g. in `add WISH`, `WISH` is a parameter which can be used as add iPhone.
- Items in square brackets are optional e.g. in `list [FILTER]`, `FILTER` is an optional parameter, since the list command can be used as `list` to display all wishes in WishBook.
  - An exception to this is `TIME_GIVEN` and `END_DATE`, whereby only one of the two can be used in any command.
  - The `add` command requires either `TIME_GIVEN` and `END_DATE`.
  - The `edit` command can take either or none of them.
- Items with … after them can be used multiple times including zero times e.g. `[t/TAG]…` can be used as (i.e. 0 times), `t/broke`, `t/needs` etc.
- The `/` symbol between parameters means that you can use either of the parameters types in the command e.g. in `add WISH PRICE [TIME_GIVEN]/[START_DATE to END_DATE]`, you provide either the `TIME_GIVEN` parameter or `START_DATE` and `END_DATE` parameters.

Add a new wish: `add`

Add a new wish to the wish list.
Format: `add n/WISH_NAME p/PRICE [a/PERIOD_GIVEN]/[d/END_DATE] [u/URL] [t/TAG]…`

- `[END_DATE]`: Specified in *dd/mm/yyyy* format.
- `[TIME_GIVEN]`: Specified in terms of years, months, weeks, and days, suffixes (coming after the value) marking such time periods are *'y'*, *'m'*, *'w'*, and *'d'* respectively.

The order of `[TIME_GIVEN]` must be from the biggest unit of time to the smallest unit of time, meaning that the suffix *`y`* cannot come after any of the other three suffixes, and *'w'* cannot come after *'d'*, but can come after *'y'* and *'m'*.

NOTE	If you enter an invalid date, a warning message will be displayed to prompt the user to reenter a valid date. Until all fields provided are valid, the wish will not be added to `WishBook`.

NOTE	The expiry date you enter must be after current date.

Examples:

- `add n/XBoxTwo p/999 a/1y`
- `add n/kfcBook 13inch p/2300 a/6m3w r/For dad t/family t/computing`
- `add n/prinkles p/1.95 d/24/04/2020`
- `add n/prinkles p/1.95 d/24/04/2020 u/www.amazon.com/prinkles t/high`

Edit a wish : `edit`

Edits an existing wish in the wish list.
Format: `edit INDEX [n/WISH_NAME] [p/PRICE] [a/TIME_GIVEN]/[d/END_DATE] [u/URL] [t/TAG]`

- Edits the wish at the specified `INDEX`. `INDEX` refers to the index number shown in the displayed list of goals. `INDEX` must be a positive integer 1, 2, 3, ...
- `INDEX` is labelled at the side of each wish.
- At least one of the optional fields must be provided.
- Existing values will be updated to the input values.
- When editing tags, the existing tags of the wish will be removed i.e. adding of tags is not cumulative.
- You can remove all tags by typing `t/` without specifying any tags after it.

Examples:

- `edit 1 n/Macbook Pro t/Broke wishes`
  Edits the name of the wish and the tag of the 1st wish to be Macbook Pro and Broke wishes respectively
- `edit 2 p/22 a/22w`
  Edits the price and time given to accomplish the 2nd wish to 22 (in the chosen currency) and 22 weeks respectively.

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.

Add Wish feature

Current Implementation

The Add Wish feature is executed through an `AddCommand` by the user, which after parsing, is facilitated mainly by the `ModelManager` which implements `Model`. It also affects `versionedWishBook` and `versionedWishTransaction` by adding the resultant wish to both of their respective data structures. After adding a `Wish`, the `filteredSortedWishes` is also updated to reflect the latest version of WishBook. The UI is also prompted to refresh through a `WishBookChangedEvent`.

`AddCommandParser` parses the user's input for parameters using prefixes, and checks them against their respective regular expressions (regex), specified in their respective classes.

The following prefix/parameter pairs are **compulsory**, where a user's input will be rejected if they are not provided:

- `n/`: Name
- `p/`: Price
- One of the following Date parameters:
  - `d/`: Exact expiry date
  - `a/`: duration (or lifetime) from time when command is entered

The following prefix/parameter pairs are **optional**, where a user's input will be successful even if they are not provided:

- `t/`: tags (more than one allowed)
- `u/`: product's URL (product page)

NOTE

Regarding Duration (`a/`) vs Date (`d/`)

- If `d/` is used, a valid Date should be used.
  - Date comes in the format of `dd/mm/yyyy`, `dd` being days, `mm` being months, `yyyy` being the year, and the
  - Specified date should also be a valid date in the future.
- If `a/` is used, a valid Duration should be used.
  - length instead of `dd/mm/yyyy` format, the format should be `<years>y<months>m<days>d`.
- In any command, only `Duration` or `Date` can be used. Never both.
- If an invalid string for date or duration is provided, a warning will be displayed to prompt the user to enter a valid date or duration.

Given below is an example usage scenario and how an AddCommand is carried out.

Step 1. The user types in a valid `AddCommand`, `add n/1 TB Toshiba SSD p/158 a/200d`, and the current date is 2nd October 2017 (2/10/2017).

The `AddCommandParser` will employ `ParserUtil` to parse the attributes specified after each prefix. The parsing of the `Duration` attribute which follows `a/` in the command will be discussed below.

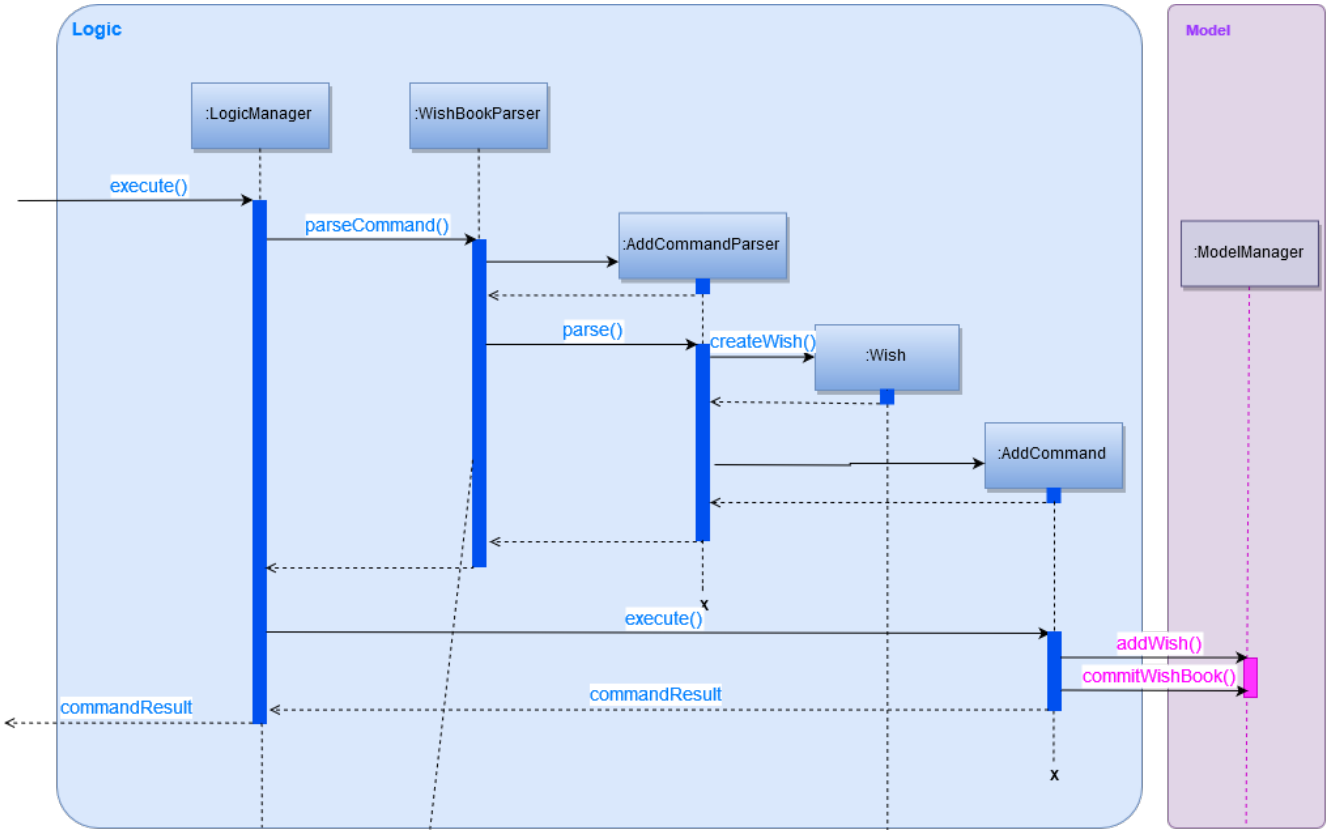
Since `Duration` prefix is used, the computation of a wish's expiry date is handled internally in the `ParserUtil` class, which `ParserUtil#parseDate()` parses and converts the input string into a `Period` object (if input is valid), and adds the resultant `Period` to the current `Date` to get the desired `Date` of the `Wish`.

The resultant `Wish` will have the following properties:

- id: `a randomly-generated UUID`
- Name: *1TB Toshiba SSD*
- SavedAmount: 0.00
- Price: 158.00
- Date: 20/4/2018 (20th April 2018)
- URL: `empty string`
- Remark: `empty string`
- Tags: `none`
- Fulfilled: `false`
- Expired: `false`

The resultant wish is pass into `VersionedWishBook#addWish` and `VersionedWishTransaction#addWish`, which tracks the history of the `WishBook` and `Wish` respectively. The list of wishes shown on the UI is also updated to show all wishes again, as `filteredSortedWishes` is updated to have all wishes in `WishBook` and a `WishBookChangedEvent` is fired.

The following sequence diagram shows how an `AddCommand` is processed in WB:



Step 2. Some time later, the user decides that she wants the exact same wish, but duplicated, and enters the exact same command, but with an exact `Date` instead of `Duration`, so the command entered is `add n/1 TB Toshiba SSD p/158 d/20/4/2018`.

Since `Date` prefix is used, the `ParserUtil` parses the string into a `Date` object, and the resultant object is used directly for the resultant `Wish`.

Similar to in Step 1, the command will be parsed successfully and a second `Wish` will be added, albeit with a different (hidden) id generated.

The resultant `Wish` will have the following properties:

- id: `another randomly-generated UUID`
- Name: *1TB Toshiba SSD*
- SavedAmount: 0.00
- Price: 158.00
- Date: 20/4/2018 (20th April 2018)
- URL: `empty string`
- Remark: `empty string`
- Tags: `none`
- Fulfilled: `false`
- Expired: `false`

Design Considerations

- **Alternative 1 (current choice)**: Different prefixes for `Duration` and `Date`.
    - Pros: More focused user experience. User get more specific feedback depending on their preferred way of inputting date if a wrong input was made. If user uses `a/` and enters an incorrect `Duration`, the user will not receive an error message about the correct format for an exact `Date`, and will only be notified of the correct format of a `Duration`.
    - Pros: Easier to implement and handle isolate errors related to respective input parameters.
    - Cons: More prefixes for user to remember.
- **Alternative 2**: Have `Duration` and `Date` use the same prefix.
    - Pros: More natural usage of one prefix to determine `Wish` 's desired expiry date.
    - Cons: Conflating implementation of `Duration` and `Date`, hence harder to debug.
    - Cons: Tricky to implement, as we are parsing one input for two different desired formats.