Evelyne Juliet - Project Portfolio

PROJECT: iFridge

Overview

iFridge is a desktop grocery management application. The user interacts with it using a CLI, and it has a GUI created with JavaFX. It is written in Java, and has a lot of code.

Summary of contributions

- **Major enhancement**: added **the ability to undo/redo previous commands** for grocery list, shopping list, and template list command.
  - What it does: allows the user to undo all previous commands one at a time. Preceding undo commands can be reversed by using the redo command.
  - Justification: This feature improves the product significantly because a user can make mistakes in commands and the app should provide a convenient way to rectify them.
  - Highlights: This enhancement affects existing commands and commands 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 commands and undo/redo implementation needs to be adjusted to the different types of list and its properties.
- **Minor enhancement**: added a **reminder command** to conveniently check for grocery item(s) expiring soon.
  - What it does: Display all grocery items expiring within specified number of days.
  - Justification: This feature increase helps users to better able keep track of their grocery items based on their expiry dates when there are a lot of grocery items in the grocery list.
  - Highlights: This enhancement adds more convenience and usability to users as they can easily check for only grocery items which they are interested in with respect to their expiry dates and remind them on grocery items that are expiring soon.
- **Minor enhancement**: added a **reminder default command** which serves as an iFridge settings.
  - What it does: Saves the user preference on the default number of days used to determine whether a grocery item is expiring soon or not.
  - Justification: This is used in color coding grocery list and waste list based on expiry dates, as well as for reminder command to check reminders for expiring food when no specific number of days is specified.
  - Highlights: This enhancement adds more convenience to users as they can set their preference on color coding conditions for grocery list and waste list which will be saved

even when the app is closed and reused when the app is reopened. It also gives the impression that the app can be personalized based on their preference as changes can be seen directly in the user interface.

- **Minor enhancement**: designed and implemented the overall user interface of the application.
  - What it does: It provides the ability to switch between the different types of list and color code the food items based on their expiry dates (mentioned earlier).
  - Justification: Linked the logic of different commands to the user interface so any changes to the model will be visible and easily understandable by the users.
  - Highlights: This enhancement improves the overall usability of the app.
- **Code contributed**: The samples of my functional and test code can be viewed here.
- **Other contributions**:
  - Project management:
  - Documentation:
    - Did cosmetic tweaks to existing contents of the User Guide: #30
  - Community:
    - Reported bugs and suggestions for other teams in the class (examples: 1, 2, 3)
  - Tools:

Contributions to the User Guide

> *Below are sections I have contributed to the User Guide. They showcase my ability to write documentation targeting end-users.*

Reminder settings: `glist remDefault`

Change and saves default number of days (n) which grocery item is expiring in, to colour code grocery items based on expiry dates.
-**red** = has expired
-**orange** = is expiring within n days
-**green** = not expiring within n days
Format: `glist remDefault r/NUMBER_OF_DAYS`

Examples:

- `glist remDefault r/3`

Default number of days is set to 3 days if not yet specified.
This command cannot be undone/redone.

Checking reminders on expiry dates: `glist rem`

Display list with all grocery items expiring within n days.
Format: `glist rem r/NUMBER_OF_DAYS`

Display list with all grocery items expiring within default number of days. Format: `glist rem` Examples:

- `glist rem r/3`
- `glist rem`

Undo Grocery List: `glist undo`

Undo the grocery list and the corresponding waste list (if applicable).
Format: `glist undo`

Redo Grocery List: `glist redo`

Redo the grocery list and the corresponding waste list (if applicable).
Format: `glist redo`

Undo Shopping List: `slist undo`

Undo the shopping list.
Format: `slist undo`

'slist mergebought' command cannot be undone.

Redo Shopping List: `slist redo`

Redo the shopping list.
Format: `slist redo`

'slist merge bought' command cannot be redone.

Undo Template List: `tlist undo`

Undo the template list and the corresponding template item list (if applicable).
Format: `tlist undo`

Undoing a template list will only display template list. Undoing a template item list will display which template is undone.

Redo Template List: `tlist redo`

Redo the template list and the corresponding template item list (if applicable).
Format: `tlist redo`

Redoing a template list will only display template list. Redoing a template item list will display which template is redone.

Contributions to the Developer Guide

Below are sections I have contributed to the Developer Guide. They showcase my ability to write technical documentation and the technical depth of my contributions to the project.

UI component

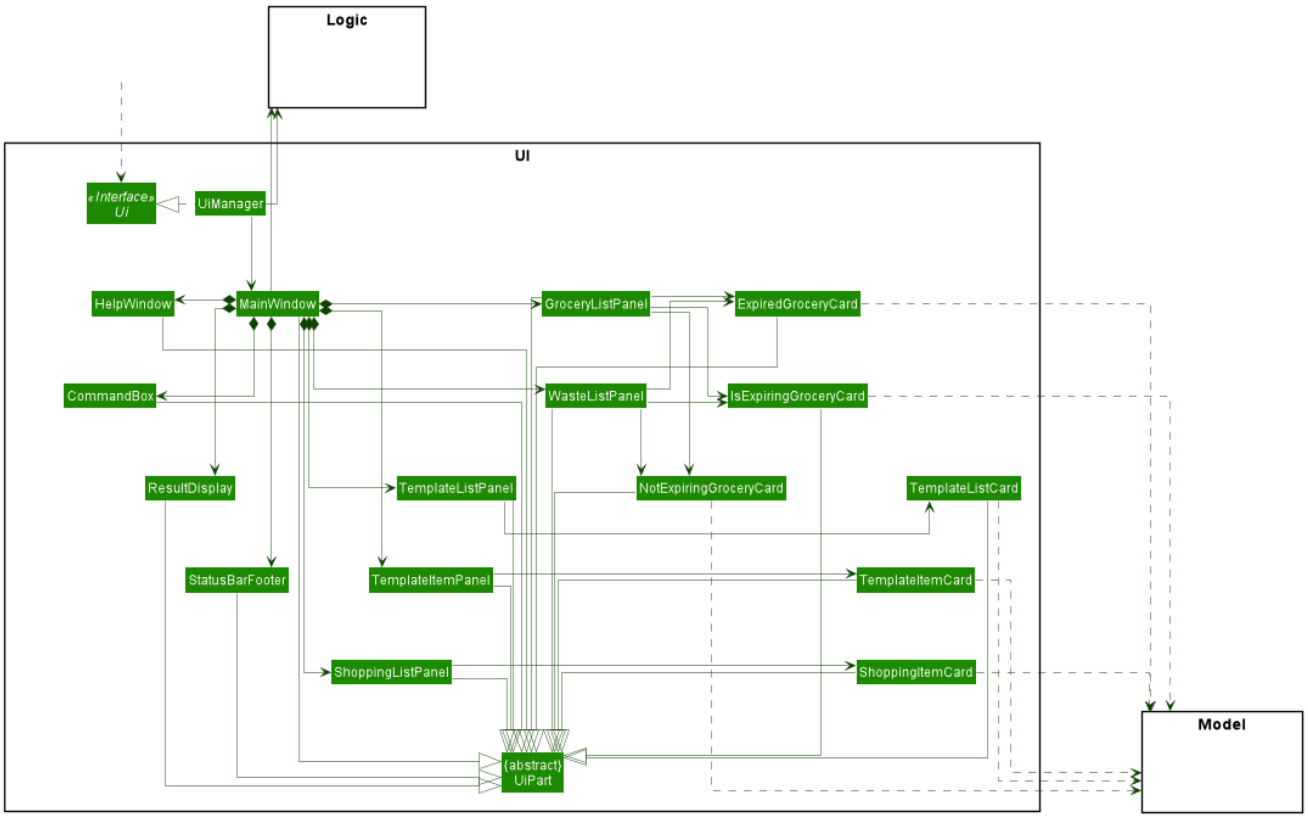


Figure 1. Structure of the UI Component

API : `Ui.java`

The UI consists of a `MainWindow` that is made up of parts e.g.`CommandBox`, `ResultDisplay`, `GroceryListPanel`, `StatusBarFooter` etc. All these, including the `MainWindow`, inherit from the abstract `UiPart` class.

The `UI` component uses JavaFx UI framework. The layout of these UI parts are defined in matching `.fxml` files that are in the `src/main/resources/view` folder. For example, the layout of the `MainWindow` is specified in `MainWindow.fxml`

The `UI` component,

- Executes user commands using the `Logic` component.
- Listens for changes to `Model` data so that the UI can be updated with the modified data.

Reminder Default Feature

Implementation

Color coding for grocery list is based on the default number of days set in the iFridge settings in the user prefs. Changing the default reminder number of days will update the color coding in the grocery list accordingly. It will also be saved when the app is closed and used again when the app is relaunched.

Given below is the Sequence Diagram for interactions within the `Logic` component for the `execute("remDefault r/3")` API call.

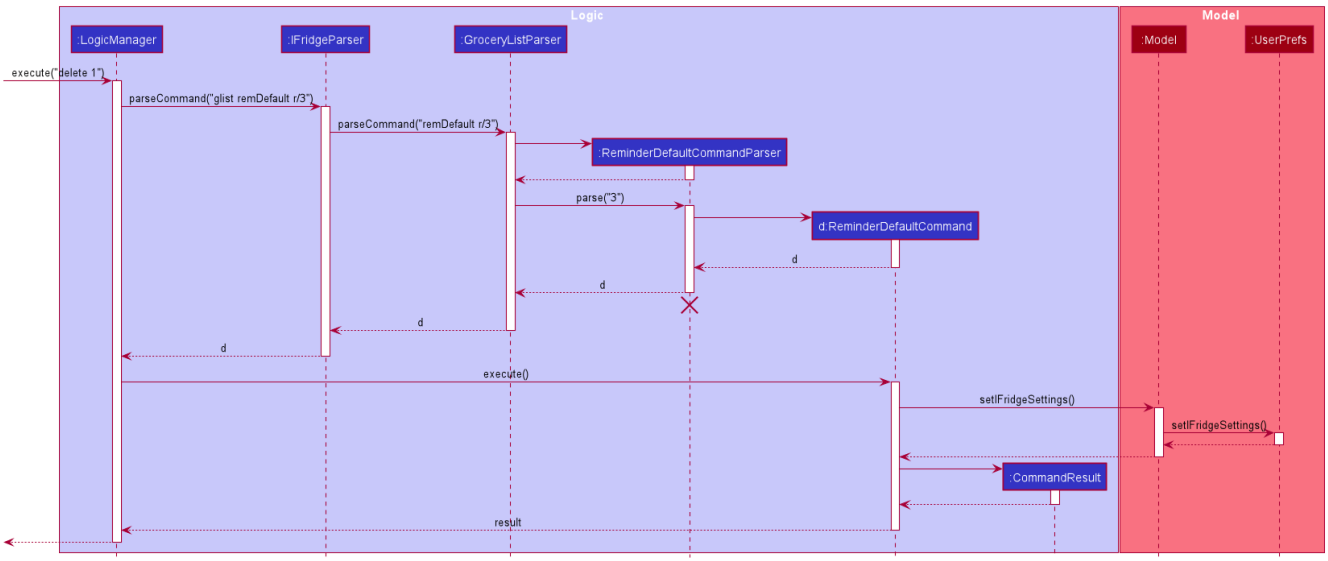


Figure 2. Interactions Inside the Logic Component for the `remDefault r/3` Command

General Undo/Redo Feature

Implementation

There are 3 types of undo/redo feature, glist undo/redo for grocery list, slist undo/redo for shopping list, and tlist undo/redo for template list.

Design Considerations

Aspect: How undo/redo is implemented

Alternative 1 (current choice): Create undo/redo separately for different lists.

- Pros: More flexibility for user in choosing which list to undo.
- Cons: Does not support commands which connects between the different lists which has an undo/redo feature of its own (eg. mergebought command which links shopping list and grocery list cannot be undone, as both shopping list and grocery list have their own undo/redo feature and complications may occur due to the interdependency between the different lists)

Alternative 2: Implement undo/redo universally so undo/redo will undo/redo any type of the last

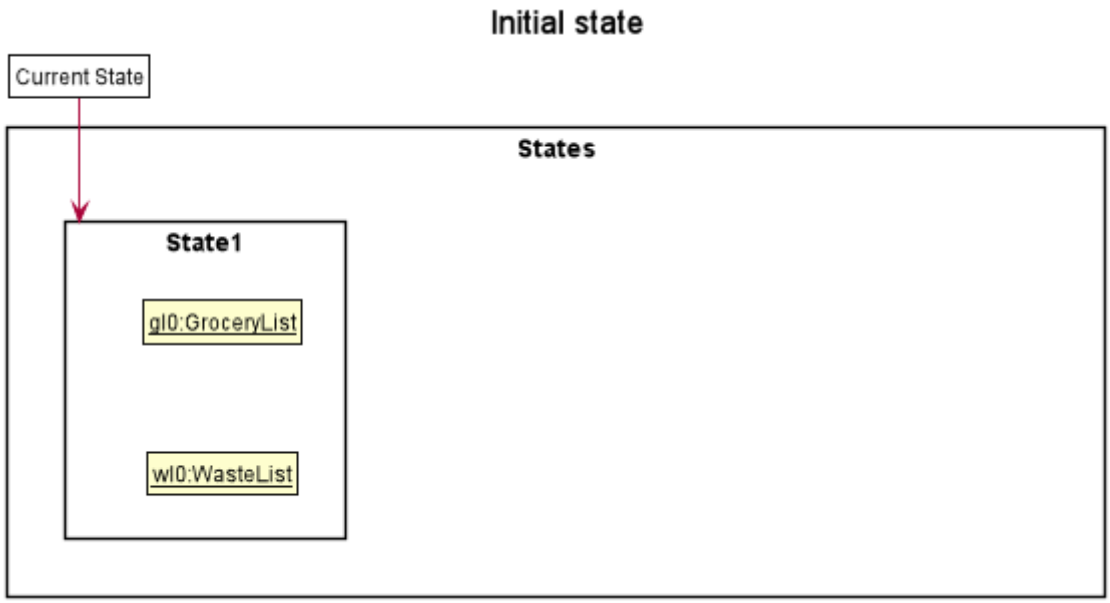
command executed.

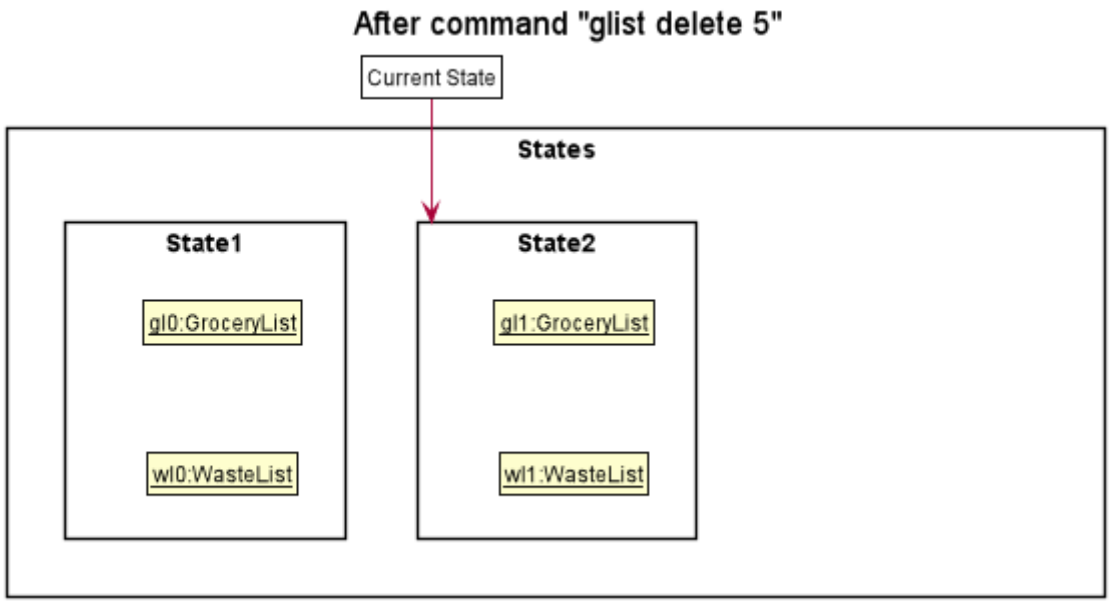
- Pros: Supports undoing/redoing commands which connects between different lists as there will be no complications arising from the interdependency of the list.
- Cons: Less flexibility to choose which list to undo.

Grocery Undo/Redo Feature

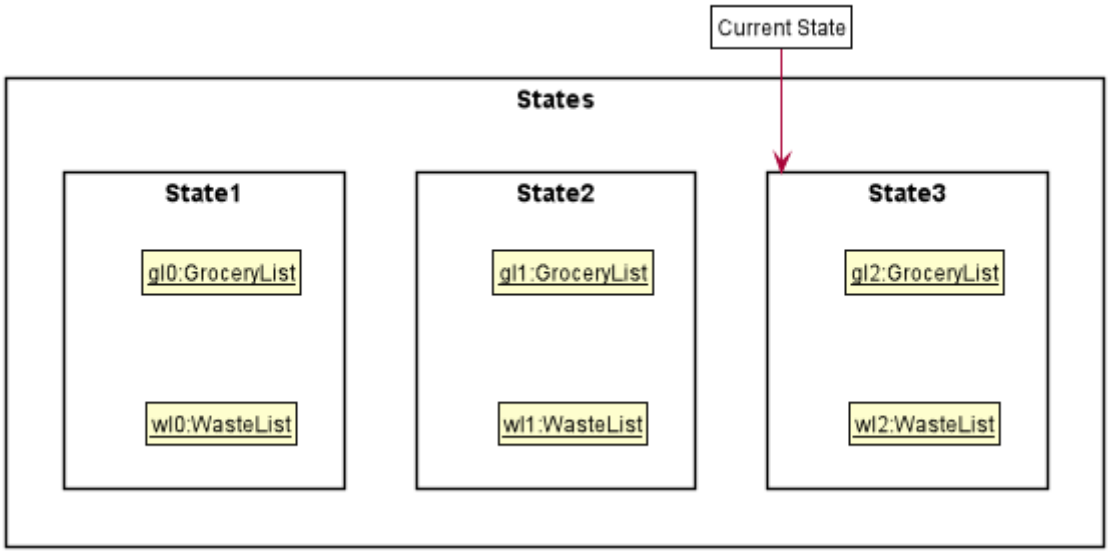
Implementation

Versioned Grocery List extends Grocery List and contains different states of grocery list. Versioned Waste List extends Waste List and contains different states of waste list. It supports any kinds of grocery command which modifies the content of the grocery list. Since the delete grocery command modifies both grocery list and waste list, each grocery command will call `Model#commitGroceryList` and `Model#commitWasteList` so that undoing/redoing a grocery delete command will update both grocery list and waste list, while the other commands will only modify the grocery list.

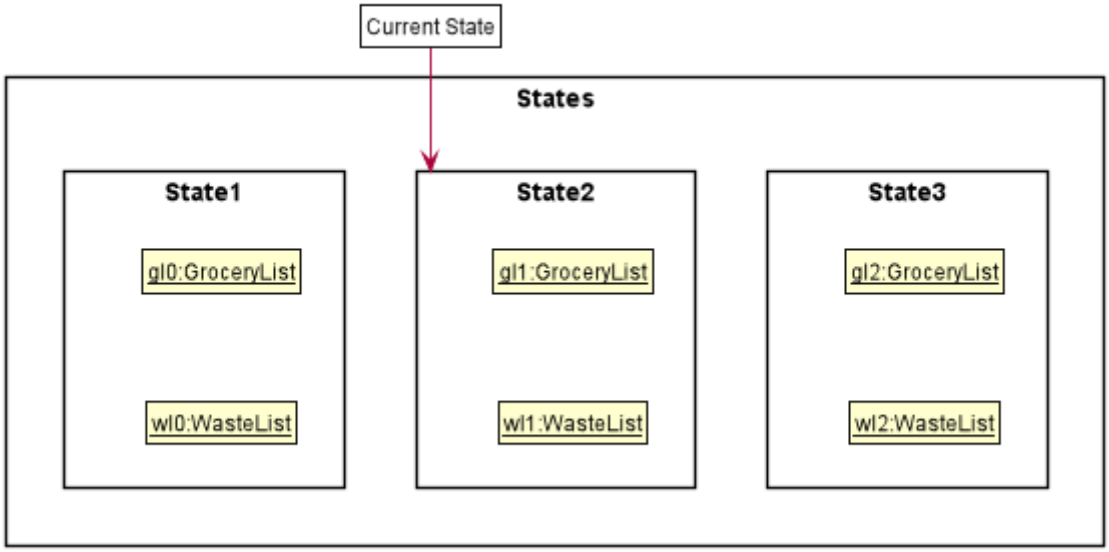




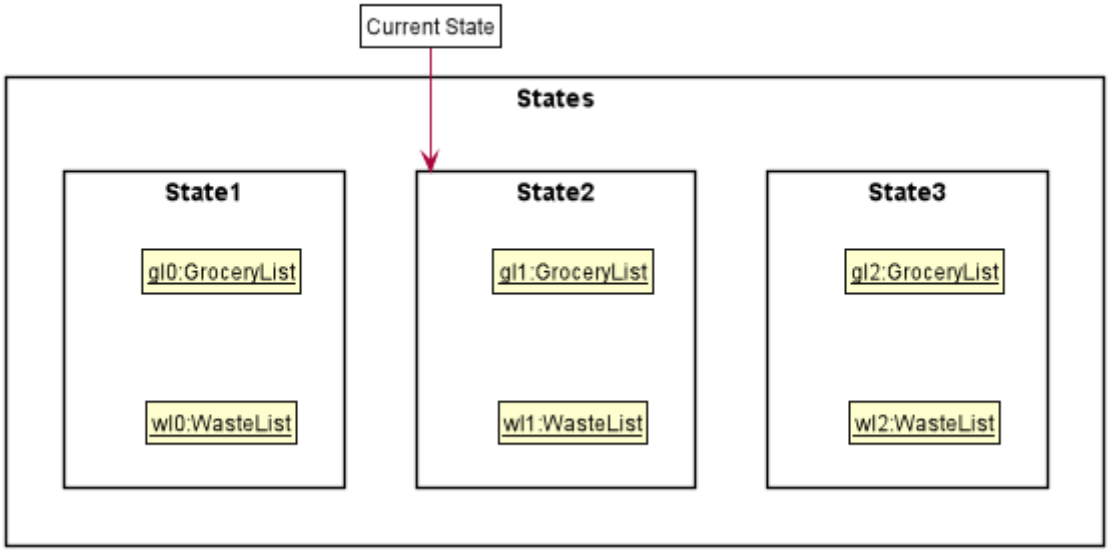
After command "glist add n/Apple a/2units e/10/12/2019"

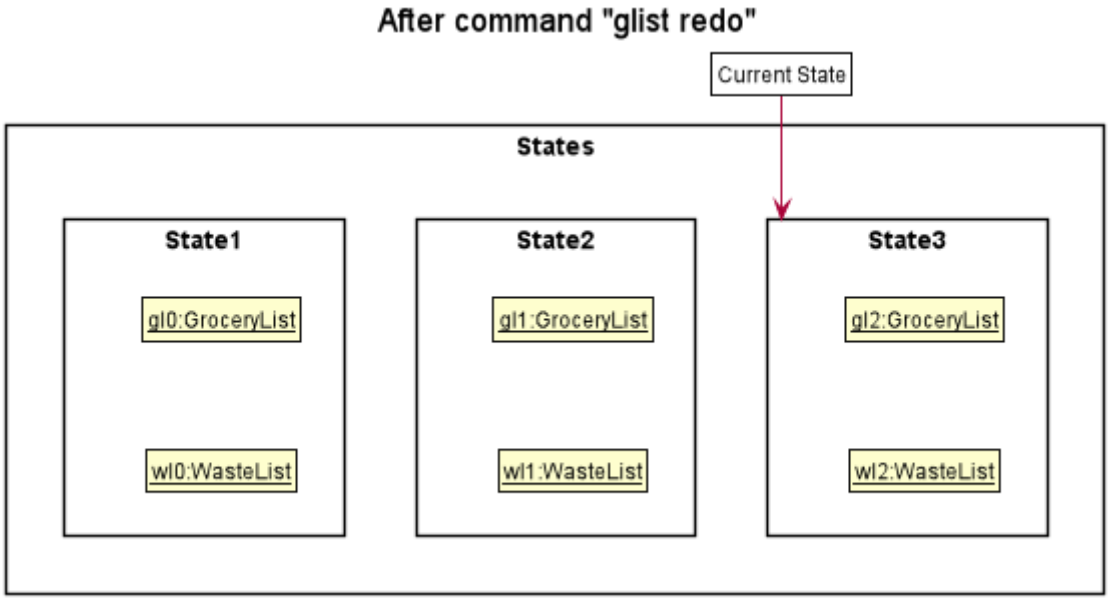


After command "glist undo"



After command "glist list"





Shopping Undo/Redo Feature

Implementation

Versioned Shopping List extends Shopping List and contains different states of shopping list. Versioned Bought List extends Grocery List and contains different states of bought list. It supports any kinds of shopping command which modifies the content of the shopping list except for mergebought command. Since the bought shopping command modifies both shopping list and bought list, each shopping command excluding mergebought command will call `Model#commitShoppingList` and `Model#commitBoughtList` so that undoing/redoing a bought shopping command will update both shopping list and bought list, while the other commands will only modify the shopping list.

Template Undo/Redo Feature

Implementation

Versioned Template List extends Template List and contains different states of template list, previous templates, new templates, and index list. It supports template list command undo/redo, and template item command undo/redo. Each template command will call `Model#commitTemplateList` which updates the corresponding lists in the versioned template list.

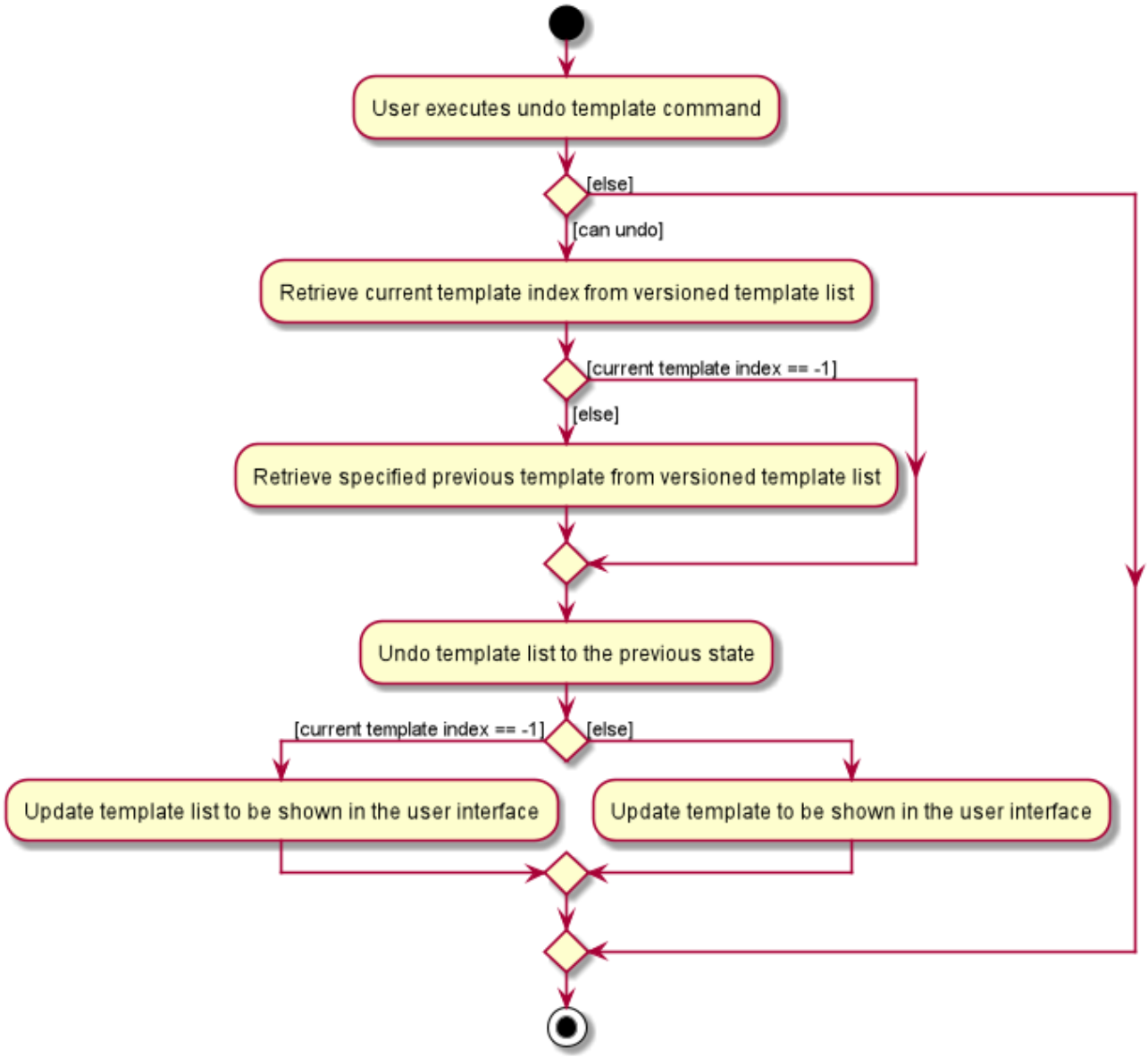


Figure 3. The following activity diagram shows what happens when the user enters an undo template command

When a template list command is undone/redone, the user interface will update the template list panel and clear the template item panel. When a template item command is undone/redone, the user interface will update the template item panel with the corresponding updated template from the prevTemplate/newTemplate list respectively. The index list is used to determine whether a template list command or a template item command is being undone/redone. If the current index is -1, the current state pointer is pointing to a template list command, else, it is pointing to a template item command.

Design Considerations

Aspect: How template undo/redo is implemented

Alternative 1 (current choice): Template undo/redo feature covers both template list command and template item command

- Pros: Prevents issues surfacing from interdependency between template list and template item command

- Cons: Less flexibility for users in choosing to undo/redo which list

Alternative 2: Create undo/redo separately for template list command and template item command

- Pros: More flexibility as users can choose which list to undo/redo
- Cons: Harder to implement as we need to check for interdependency between the two list and how it affects the other list' state before performing the corresponding undo/redo