# Kelvin Harris - Project Portfolio

## PROJECT: iFridge

## Overview

iFridge is a desktop grocery management application to encourage home cooks to manage their food waste. The user interacts with it using a CLI, and it has a GUI created with JavaFX. It is written in Java, and has about 20kLoC.

## Summary of contributions

- **Major enhancement**: added **the ability for user to manage his/her grocery items systematically**.
  - What it does: The grocery list management allows the user to keep track of their grocery items with ease.
  - Justification: Fridge management is a difficult process as it is hard to keep track of all the food items in the fridge and consume them before they expire. iFridge's grocery list solves the problem by allowing the user to key in the data of the food items.

    The list supports basic features such as `glist add` and `glist delete` as well as other complementing features such as `glist sort` and `glist find`. The grocery list management is the backbone of the app as grocery items management is needed to support management of other lists (waste list, template list and shopping list). In addition, the grocery list is likely to be the most frequented list by the user.

  - Highlights: While implementing this feature, I had to implement new commands such as `glist use` and `glist sort`. `glist use` is differentiated from `glist edit` as the underlying implementation of `glist use` appears to be different from `glist edit`.

    On the other hand, `glist sort` is implemented to further improve the convenience level for the user. As of now, the sorting type supported is in ascending lexicographical order and ascending date.

- **Minor enhancement**: Cross-conversion between units in `Amount` class
  - What it does: Allows for cross-conversion between different units supported by the app.
  - Justification: Users would prefer to input the amount field based on their own preferences of units. They would not want to be restricted by the units already specified as they would have to do their own conversions manually. Thus, we decided to implement the unit conversions to allow for greater convenience for the user.
  - Highlights: While implementing this feature, the choices of units are narrowed down to those that are most frequently used. Both the units and values can be easily retrieved by

methods from the `Amount` class. As of now, the conversion methods are hard-coded as I was not able to find a library that is fully appropriate for the feature. The conversion methods in `Amount` class were heavily abstracted to provide next developers the ease of understanding and adding support for other units as they wish.

- **Code contributed**: The samples of my functional and test code can be viewed here.
- **Other contributions**:
  - Project management:
    - Added user stories as issues on GitHub
    - Reviewed pull requests by team members
    - Opened issues when bugs were found (Issue #219)
    - Fixed bugs after they were found (PR #218)
  - Community:
    - Reported bugs and suggestions for other teams in the class (Issue #5, #2, #1)
  - Tools:
    - Setup the Github repo, Travis and Coveralls.
    - Integrated TravisCI to the team repo (PR #7)
    - Integrated Coveralls to the team repo (PR #237)

# 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.*

## Grocery List Management

The grocery list is the

### Adding a food item: `glist add`

Adds a grocery item to the grocery list.
Format: `glist add n/ITEM_NAME e/EXPIRY_DATE a/AMOUNT [t/TAG]`

- `e/EXPIRY_DATE` must follow the format `dd/MM/yyyy`.
- `a/AMOUNT` must have a magnitude i.e. measurable quantity. Magnitude and unit can be separated by a space.
- There may be more than one tag field.
- The input fields can be in any order.
- Item with **either same name or expiry date** as the existing ones can be added.
- Item with **same name and expiry date** as the existing ones cannot be added.

Examples:

- `glist add n/Fuji apples e/30/10/2019 t/healthy a/10units`
  Add Fuji apples of quantity 10 units, tagged as "healthy" and expiring on 30 October 2019.
- `glist add n/salad a/3 e/25/09/2019`
  Add salad of quantity 3, untagged and expiring on 25 September 2019.
- `glist add n/tea a/200 ml e/18/10/2019 t/fresh t/drink`
  Add tea of quantity 200 ml, tagged as "fresh" and "drink" and expiring on 18 October 2019.

### Listing all grocery items: `glist list`

Shows a list of grocery items in the grocery list.
Format: `glist list`

### Deleting a grocery item: `glist delete`

Deletes the specified grocery item from the grocery list when it is done being used. If the grocery item is not completely used up when deleted (i.e. `amount` > 0), the item will be moved to the waste list.
Format: `glist delete INDEX`

### Use a grocery item: `glist use`

Reduces the amount left of a grocery item by the specified amount.
Format: `glist use INDEX a/AMOUNT`

- Only the units in the following unit categories are supported:
  - `Weight`: `kg`, `g`, `oz`, `lbs`
  - `Volume`: `L`, `ml`
  - `Quantity`: `units`
- Spaces between magnitude and unit is allowed.
- Cross conversion between units of the same categories is supported. (Refer here for more info)
- If the unit of input amount is different from the unit of item, the resultant amount would follow the unit of the original item. e.g. If an item of `1kg` is used by `300g`, it would become `0.7kg` and not `700g`.

Examples:

- `glist use 2 a/300g`
  Reduces the amount of 2nd item by 300 grams.
- `glist use 3 a/5 L`
  Reduces the amount of 3rd item by 5 liters.

## Edit a grocery item: `glist edit`

Edits an existing item in the grocery list.
Format: `glist edit INDEX [n/ITEM_NAME] [e/EXPIRY_DATE] [t/TAG]`

- Edits an item at the specified `INDEX`. The index refers to the number shown in the displayed list. The index **must be a positive integer** 1, 2, 3, …
- At least one of the optional fields must be provided. At least one of the provided fields must be different from the original item's field.
- Amount cannot be edited, but can only be modified through `use`.
- Existing values will be updated to input values.
- When editing tags, the existing tags of the item will be removed. i.e. adding of tags is not cumulative.
- You can remove all the item's tags by typing `t/` without specifying any tags after it.
- **As the grocery list does not allow duplicates (items with same name and expiry date), any attempt of edit to achieve duplication is not allowed and would result in an error.**
- **User is encouraged to practise discretion when editing an item as iFridge is unable to recognise items based on its name and compare between the name and amount of an item.** For example, if an item named `Milk` with amount `2L` is edited to have name `Beef`, iFridge would allow the edit, even though `Beef` with amount `2L` does not sound logical.

Examples:

- `glist edit 1 n/Fuji apple t/healthy`
  Edits the name and tag of the 1st item to be `Fuji apple` and `healthy` respectively.
- `glist edit 2 n/Olive oil t/`
  Edits the name of the 2nd item to `Olive oil` and clears all existing tags.

### Sort the grocery list: `glist sort`

Sorts the grocery list based on the type of sorting.
Format: `glist sort by/TYPE`

Sorting is done on the original grocery list. Hence, for example, when `sort` is done after `find`, both the resultant `find` list and the original list will be sorted.

The type of sorting supported is as follows:

- `alphabetical`: Sort the grocery list in ascending alphabetical order
- `expiry`: Sort the grocery list in ascending expiry date. i.e. from oldest to newest.

Examples:

- `glist sort by/alphabetical`
- `glist sort by/expiry`

### Finding items by keywords: `glist find`

Finds items whose name or tag contain any of the given keywords.
Format: `glist find KEYWORD [MORE_KEYWORDS]`

- The search is case insensitive. e.g `apple` will match `Apple`
- The order of the keywords does not matter. e.g. `apple milk` will match `milk apple`
- Only the item name and tag are searched.
- Only full words will be matched e.g. `appl` will not match `apple`
- Grocery items matching at least one keyword will be returned (i.e. it uses an `OR` search). e.g. `apple dinner` will return `Apple juice`, `Pizza` tagged with `dinner`, and `Apple pie` tagged with `dinner`.

Examples:

- `glist find apple`
  Returns `Fuji apple` and `Apple loaf cake` and other items tagged as `apple`
- `glist find milo doughnut roasted`
  Returns any grocery item which contains any of the word `milo`, `doughnut`, or `roasted` as either

name or tag or both

# 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.*

## Use Grocery Feature

### Implementations

The `glist use` feature in the grocery list allows user to use their food items based on the `AMOUNT` inputted.

This implementation of this feature is highly dependent on the `Amount` class as deduction of values is done by the `Amount` class itself.

The sequence diagram for interactions between the Logic and Model components when a user executes the `glist use` command is shown below. The subsequent sequence diagram shows a lower level picture of how a grocery item is used.

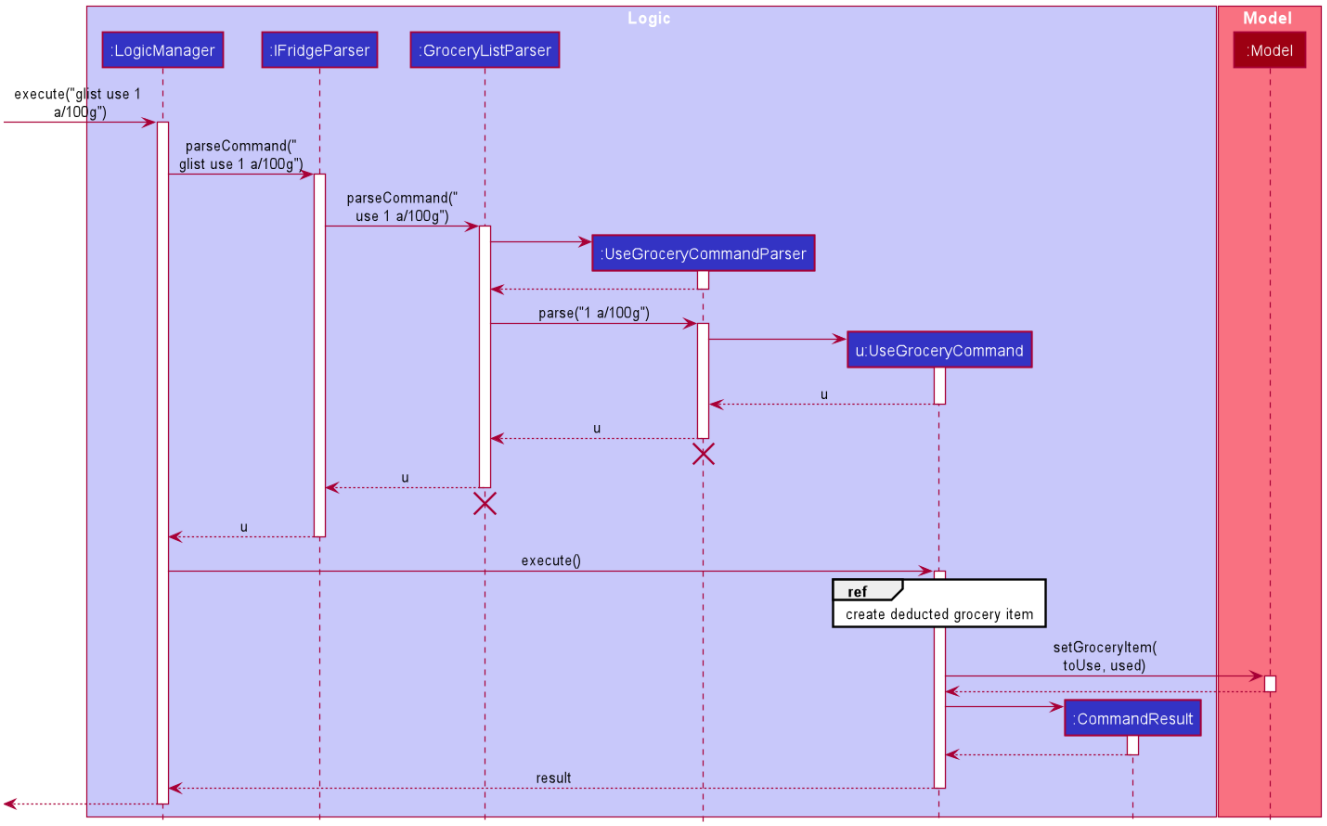



*Figure 1. The sequence diagram to show how use grocery command is parsed and executed.*

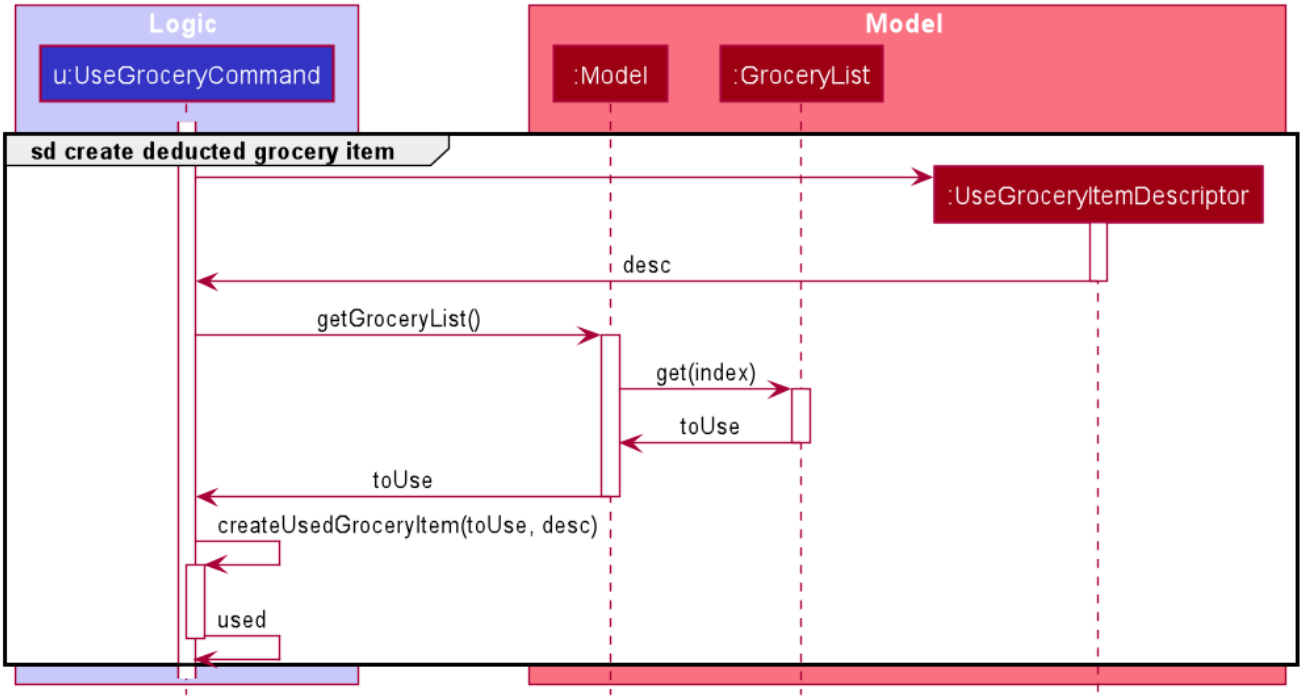
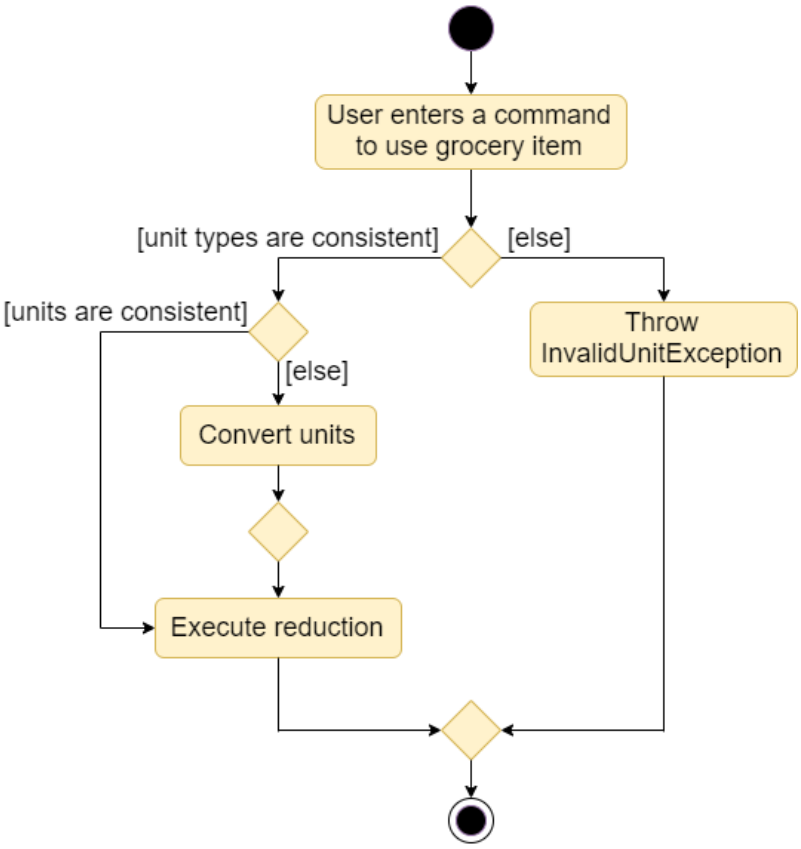



*Figure 2. The sequence diagram for the reference frame to show the process of how a grocery item is used.*

The current implementation for using a grocery item is by overwriting the existing grocery item with a new grocery item object with its amount field deducted, as shown in the diagram above. The rest of the fields are copied over without any other modifications.

The `glist use` command is also able to support unit conversion. Currently, the implementation of the unit conversion are calculated manually inside the `Amount` class. No external library is used. (Refer to About the amount parameter)

The following activity diagram summarises how the unit conversion is done.




The conversions of units are done in the `Amount` class. Unit type is necessary in the implementation to allow for keeping track of different unit groups across different lists in the application. For example, `kg`, `g`, `lbs`, and `oz` are all categorised under the unit type `Weight`.

**About the Amount Parameter**

- Several commands involve computation of amounts. Units are classified under three unit types:
  - **Volume:** ml, L
  - **Weight:** g, kg, oz, lbs
  - **Quantity:** units
- Conversion of units can only be applied on the same classification. The conversion implemented in this app is shown in the table below.

| Amount in the unit specified | Amount in standard form (kg, L, units) |
|---|---|
| 1ml | 0.001L |
| 1g | 0.001kg |
| 1oz | 0.00283495kg |
| 1lbs | 0.453592kg |

- A certain minimum value is imposed on the `Amount` class (0.1ml for Volume, 0.1kg for Weight, and 0.1units for Quantity) to ensure consistency. Any value below the restriction would result in an error.

## Design Considerations

### Aspect: Storing of the value and unit

*Table 1. Design considerations for storing value and unit in `Amount` class*

| Alternative 1 (Chosen Implementation) | Alternative 2 | Alternative 3 |
|---|---|---|
| Storing the value and unit as a combined string.<br>• Pros:<br>◦ Maintains consistency with other fields (name, expiry date).<br>• Cons:<br>◦ Difficult to parse and manipulate. | Storing the value and unit as a float and a string<br>• Pros:<br>◦ Parsing would be easier.<br>• Cons:<br>◦ Consistency is compromised. | Storing the value and unit as two classes of itw own (i.e. Value class and Unit class)<br>• Pros:<br>◦ More OOP.<br>◦ More scalable and maintainable.<br>• Cons:<br>◦ Consistency is compromised.<br>◦ Harder to implement.<br>◦ Implementation might be overkill/redundant. |

### Aspect: Deducting the `Amount`

*Table 2. Design considerations of the `glist use` command*

| Alternative 1 (Chosen Implementation) | Alternative 2 |
|---|---|
| Create a new grocery item and replace it with the old one.<br>• Pros:<br>◦ This method is easier to implement and would be less prone to bugs.<br>• Cons:<br>◦ Less efficient in terms of runtime. | Modify the `Amount` in the grocery item.<br>• Pros:<br>◦ Will be more efficient and use less memory.<br>• Cons:<br>◦ Mutable `Amount` field may result in unforeseen changes, hence more prone to bugs. |

### Aspect: Keeping track of unit type

*Table 3. Design considerations for the unit conversion in `glist use`*

| Alternative 1 (Chosen Implementation) | Alternative 2 |
| --- | --- |
| Keeping the original unit of the item.<br>• Pros:<br>◦ Easy to maintain.<br>• Cons:<br>◦ May not be very intuitive for the user. | Changing the original unit of item to the one input by user.<br>• Pros:<br>◦ Will be more intuitive to the user. e.g. After using `650ml` of a `1L` milk, it might be more intuitive to show `350ml` instead of `0.35L`.<br>• Cons:<br>◦ Difficult to implement and maintain, due to its subjectiveness.<br>◦ Consistency may be compromised. |

# Grocery List Manual Testing

1. Adding a grocery item while all items are listed
   a. Prerequisites: List all grocery items in the grocery list using the `glist list` command. The grocery list can be empty or contain some items.
   b. Test case: `glist add n/Coffee a/200ml e/19/11/2019 t/caffeine`
   Expected: If the grocery list is empty, a grocery item would be added to the grocery list. Details of the added grocery item would also be shown in the status message.
2. Deleting a grocery item while all items are listed
   a. Prerequisites: List all grocery items in the grocery list using the `glist list` command. Multiple grocery items exist in the grocery list.
   b. Test case: `glist delete 1`
   Expected: First grocery item is deleted from the grocery list. Details of the deleted grocery item is shown in the status message.
   c. Test case: `glist delete 0`
   Expected: No grocery item is deleted. Error details is shown in the status message.
   d. Other incorrect delete commands to try: `glist delete`, `glist delete x` (where x is larger than the list size). Expected: Similar to previous.
3. Editing a grocery item in the grocery list
   a. Prerequisites: List all grocery items in the grocery list using the `glist list` command. Multiple grocery items exist in the grocery list. Item to be edited must not same name and expiry date as any other item in the list.
   b. Test case: `glist edit 1 n/Papaya`
   Expected: If the first grocery item has unique name and expiry date in the grocery list, the grocery item's name is edited to Papaya. Other fields remain the same. Details of the edited grocery item is shown in the status message.
   If the first grocery item does not have unique name and expiry date in the grocery list, the

grocery item's name is not edited. Error details is shown in the status message.

c. Test case: `glist edit 2 a/500ml`
Expected: Error details is shown in the status message as amount field cannot be edited.

4. Using a grocery item in the grocery list

a. Prerequisites: List all grocery items in the grocery list using the `glist list` command. Multiple grocery items exist in the grocery list. Item to be used has the same unit type and is not used up completely. Amount left exceeds amount to be used.

b. Test case: `glist use 1 a/50ml`
Expected: The amount of the first grocery item is deducted by 50ml. Other fields remain the same. Details of the used grocery item is shown in the status message.

c. Other invalid use commands to try: `glist use 2 a/400g` (where the item has unit of L), `glist use 2 a/30lbs` (where the amount of the item is less than 30lbs).

5. Sorting the grocery list

a. Prerequisites: List all grocery items in the grocery list using the `glist list` command. Multiple grocery items exist in the grocery list.

b. Test case: `glist sort by/expiry`
Expected: The displayed grocery list would be sorted based on the grocery items' expiry date in ascending order (from earliest to most recent).

c. Test case: `glist sort by/alphabetical`
Expected: The displayed grocery list would be sorted based on alphabetical order of the grocery items' name in ascending order.

6. Finding items based on name or tag

a. Prerequisites: List all grocery items in the grocery list using the `glist list` command. Multiple grocery items exist in the grocery list. Name or tag to be found may exist or not exist in any of the grocery items.

b. Test case: `glist find apple snack`
Expected: The displayed grocery list would now contain items that has `apple` and `snack` as either their name or tag. If no items match, the displayed list would be empty.

c. Other invalid find commands to try: `glist find b@nana` (would result in error as name and tag cannot contain any non-alphanumeric character).

# PROJECT: PowerPointLabs

---

*{Optionally, you may include other projects in your portfolio.}*