Chang Hui Zhen Project Portfolio

Overview

Kitchen Helper is a command-line application that was developed by a team of 6 Information Security students. We aim to help housewives with the management of their kitchen-related activities. This application enable users to keep track of their ingredients as well as the chores to be completed.

Contributions

Enhancements implemented

1. Major
    a. Adding of recipe
        i. Functionality: This is a representation of a collection of ingredients that are commonly cooked together.
        ii. Justification: With this functionality, users will be able to check for the ingredients that are used for a certain dish and would be available for future reference.
        iii. Highlight: Users would not need to worry that their recipe or ingredient name has to be restricted (i.e. can contain whitespace, any category)
    b. Cooking a recipe (Logical checks before deduction)
        i. Functionality: This is a batch deduction of ingredients that are found in the recipe specified. Before deduction, a series of checks were performed. If all the checks passed, the automatic deducted will be proceeded, an error message otherwise.
        ii. Justification: This feature ensure that the user will be notified when there is an expired ingredient before proceeding with the cooking of the recipe. This is important as the user might not have noticed that there is insufficient non-expired ingredient.
        iii. Highlight: The user is notified with specific error messages on several situations including sufficienct ingredients, insufficient ingredients and when there is sufficienct ingredients that included expired ingredients.
    c. Resetting the application
        i. Functionality: All application data will be wiped and restarted from fresh.
        ii. Justification: User can choose to clear the inventory without having to run delete each of the recipes/ ingredients/ chores themselve.
        iii. Highlight: This provides automatic clearance to the application.

Code Contributions

Code contribution dashboard

Team-based tasks

1. Setting up the team repo
2. Release Management: Managed `v1.0` to `v2.1` releases
3. General code enhancements: adding missing JavaDocs, remove dead codes
4. Integration of code snippets and maintaining repository

Documentation

1. User Guide
    a. Sections: 3.1.3. Resetting the application: `reset`, 3.4.1. Adding a recipe `addrecipe`, 3.4.5. Cooking a recipe `cookrecipe`,
2. Developer Guide
    a. Sections: 1.1 Background, 3.2. Ui Component, 3.6. Common Classes, 4.2.1 Addition of Recipe, 4.2.3. Cooking a recipe (Design Considerations), 4.5. Logging, Apppendix D & E, Appendix F.6., F.8.

Review/mentoring contributions

1. Pull Requests reviewed (with non-trivial comments): #41, #118, #122, #124, #125, #137, #141, #153, #156, #157, #165, #243, #246, #247, #254
2. Reported Bugs: #1, #2, #3, #4, #5, #6, #7, #8, #9, #10, #11, #12

Contributions to the User Guide

3.1.3. Resetting the application: reset

You can reset the application which will wipe out all existing data. **Format**: `reset`

3.4.1. Adding a recipe: `addrecipe`

Adds a new unique recipe into the List in Kitchen Helper. A recipe is a list of ingredients that are used to cook a dish.

Format: `addrecipe /n <RECIPE_NAME> /i <INGREDIENT_NAME>:<QUANTITY>:<CATEGORY>[,..]`

- `RECIPE_NAME` is the name of your recipe.
- `INGREDIENT_NAME` is the name of your ingredient.
- `QUANTITY` number of servings of the ingredient.
- `CATEGORY` is the CATEGORY of your ingredient.

The different types of `CATEGORY` are listed below: + `Meat` + `Vegetable` + `Staple` + `Fruit` + `Dairy` + `Drink` + `Miscellaneous`

Any `CATEGORY` that does not falls in the list could be put under `Miscellaneous`.

> All `RECIPE_NAME` has to be unique. You can check the list of existing recipes by using listrecipe all Please note that `RECIPE_NAME` and `INGREDIENT_NAME` can contain spaces. These will not be removed after addition. (i.e. "Chicken___Stew" where _ is space will remain)

Example	Outcome
Command: `addrecipe /n Rice Ball /i Rice:3:staple` **Description**: Creates a new recipe called `Rice Ball` which contains one ingredient, `3` portions of `Rice`.	addrecipe /n Rice Ball /i Rice:3:staple Rice Ball Recipe has been created with 1 ingredients inside. ==
Command: `addrecipe /n Chicken Salad /i Chicken Breast:2:meat, Lettuce:4:vegetable` **Description**: Creates a new recipe called `Chicken Salad` which contains two ingredient, `2` portions of `Chicken breast` and `4` portions of `Lettuce`.	Chicken Salad Recipe has been created with 2 ingredients inside. ==
Command: `addrecipe /n Chicken Salad /i Chicken Breast:2:meat, Lettuce:4:vegetable` **Description**: A duplicate recipe has been found	There is an existing recipe with the same name! ==

3.4.5. Cooking a recipe: `cookrecipe`

Cooks a recipe specified by the user by the recipe's name.

Format: `cookrecipe /n <RECIPE_NAME> /p <NUMBER_OF_PAX>`

- `RECIPE_NAME` is the name of your recipe.
- `NUMBER_OF_PAX` is the pax count for the specified recipe.

> Please note that expired ingredients cannot be cooked and will be prompted to clear them.

> Please note that the ingredients used in the recipe will be matched strictly by their `INGREDIENT_NAME` and `CATEGORY` when cooking a recipe. You may refer to the addrecipe command section

The `ingredient`s used in the specified recipe will be automatically deducted when there is sufficient non-expired `ingredient`s.

Situation	Example	Outcome
Sufficient ingredients for all ingredients required in the specified recipe.	**Command**: `cookrecipe /n chicken salad /p 2` **Description**: Cooks the recipe `Chicken Salad` for 2 people	Kitchen Helper is trying to cook! Cooks the 'chicken salad' recipe with a pax 2. ========== ==
Insufficient ingredients for all ingredients required in the specified recipe regardless if the `ingredients` have expired or not.	**Command**: `cookrecipe /n Chicken Salad /p 3` **Description**: Cooks the recipe `Chicken Salad` for 3 people	cookrecipe /n Chicken Salad /p 3 Kitchen Helper is trying to cook! There are insufficient/missing ingredients! ========== ==

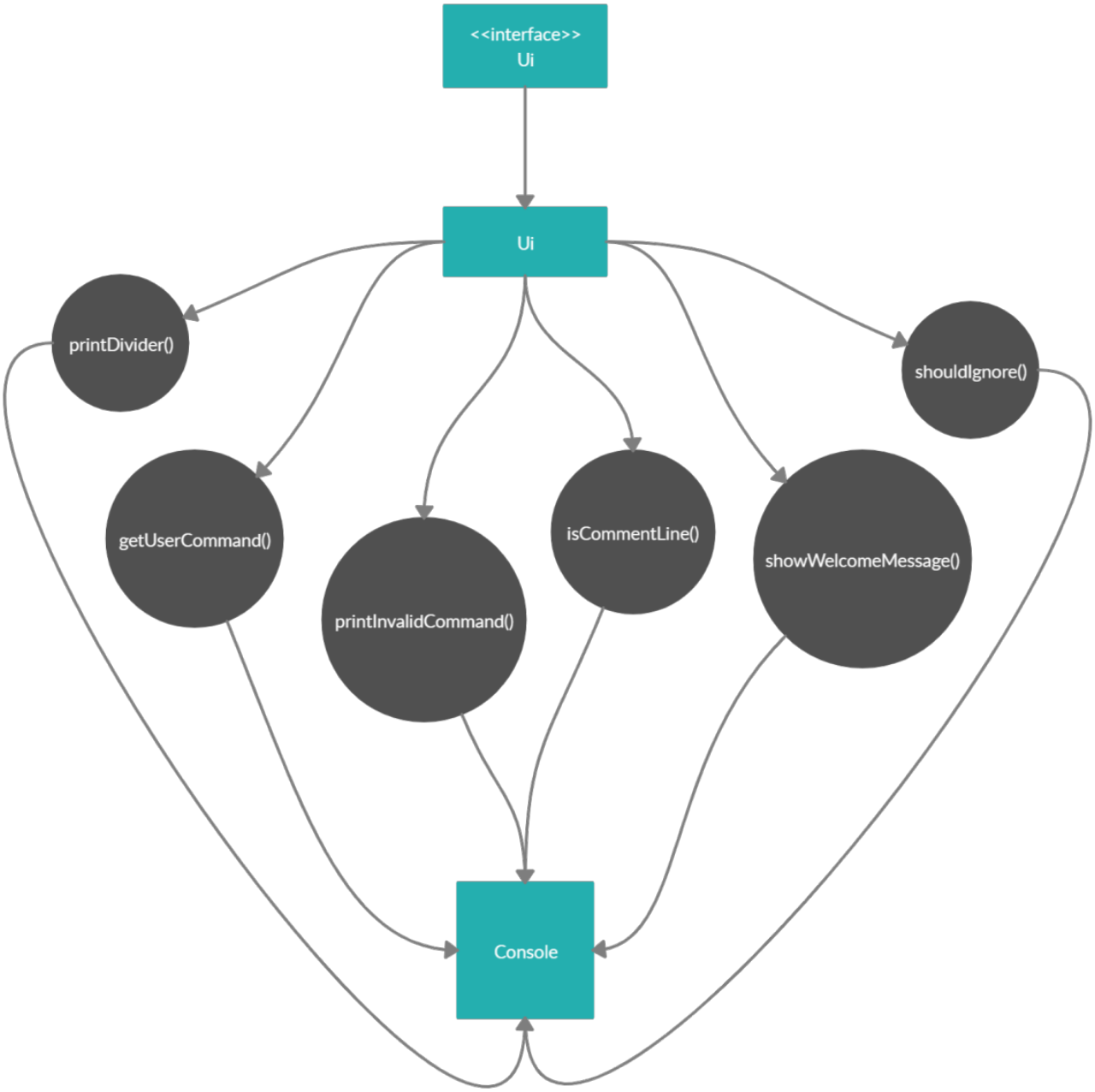
Insufficient non-expired ingredients available.	**Command**: `cookrecipe /n warm milk /p 2` **Description**: Cooks the recipe `warm milk` for 2 people	Kitchen Helper is trying to cook! There are insufficient/missing ingredients! Please check for these expired ingredients: hl milk ========== ==

Contributions to the Developer Guide

1.1. Background

Kitchen Helper, born from the need to keep track of kitchen inventory, is an application that is designed to manage kitchen inventory and chores. Users will be able to reduce food wastage and save money through the convenience of viewing the contents of the inventory.

3.2. Ui Component



API: `Ui.java`

The `Ui` component is a singleton class where all interaction will be made through this component

The `Ui` component,

- Executes user commands using the command component
- Listens for changes and outputs messages from the Command component

3.6. Common Classes

Classes used by multiple components are in the `seedu.kitchenhelper.object` package.

4.2.1. Addition of recipe

Users can add a new recipe to the application where there must be at least one or more `ingredient`s. The failure to do so will trigger an exception where the user will be notified of an invalid command and the syntax of the addition of recipe will be displayed.

> It is important that the name of the new recipe has not appeared in the list of recipes in the application.

When the user attempts to create a new recipe, the `AddRecipeCommand`, 'Parser' and `Recipe` class will be accessed and the following sequence of actions are called to create a `recipe` object:

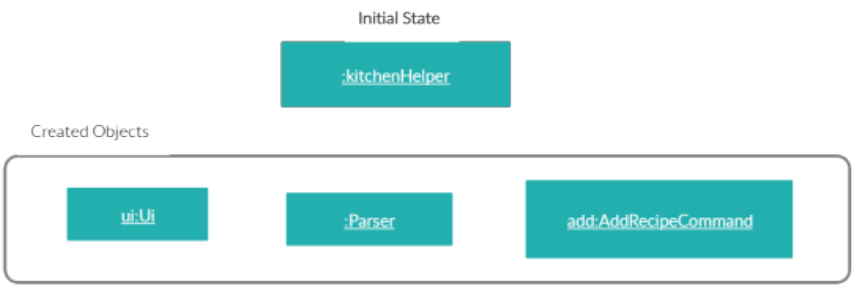
Implementation

When the user attempts to create a new recipe, the `AddRecipeCommand`, 'Parser' and `Recipe` class will be accessed and the following sequence of actions are called to create a `recipe` object:

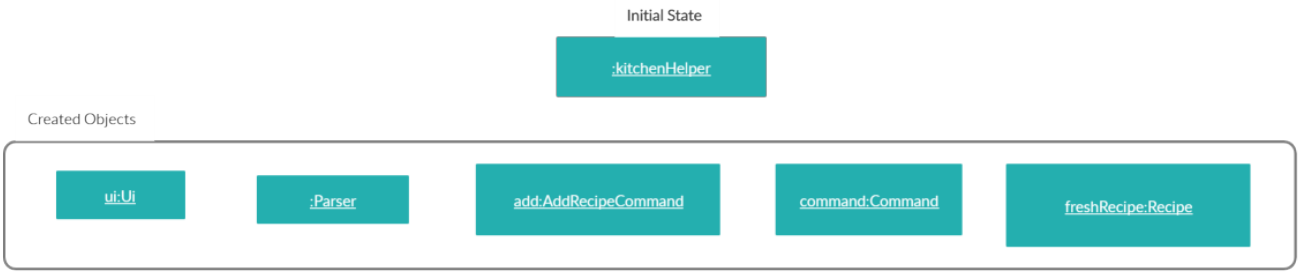
1. User executes `addrecipe /n Chicken Salad /i Chicken Breast:2:meat, Lettuce:4:vegetable`
    a. A `Ui` object will be created and calls `Ui#getUserCommand()`
    b. Input will be parsed in `Command#parseUserCommand()` and identified with the keyword `addrecipe`.

1. Parsing of user input and creation of command object
    a. This will automatically trigger the parsing of the user's input string into a suitable format for the addition of `recipe` object in `Command#prepareAddRecipe()`.
    b. A `AddRecipeCommand` object will be created and calls `AddRecipeCommand#setAttributesOfCmd()` to set the contents of the command into reader friendly formats.

1. Executing Command
    a. The newly created object will call `#AddRecipeCommand#execute` which starts the process of adding a recipe, thus calling `Recipe#AddRecipe()`.
    b. A `Recipe` object will be created with its name that was parsed in step 2.
    c. An additional step is included where a check for an existing recipe with the same name is conducted with `#AddRecipeCommand#checkIfRecipeExist()`. A `KitchenHelperException` exception will be triggered when there is an existing recipe.



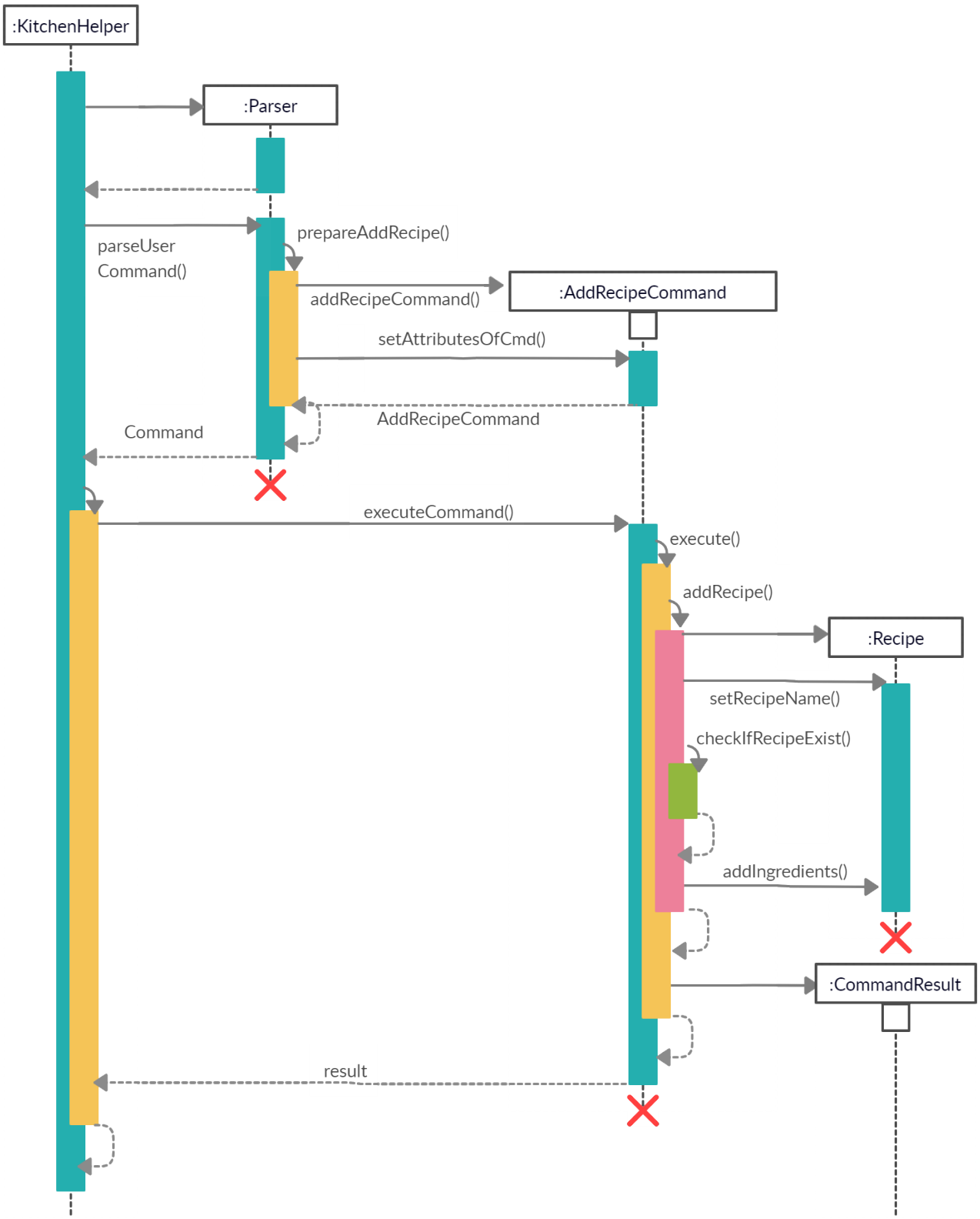
1. `Ingredient`s parsed in step 2 will be added to the newly created recipe according to their category through the calling of `Recipe#addIngredientsToRecipe()`.



All description and warnings to the user utilises the `UI` class, which controls the printing of the text

on the console.

The following sequence diagram shows how the `addrecipe` command works



Design Considerations

Aspect: Parsing of the user's input command

Alternative 1 (current choice): The key parameters that are required are divided by the delimiter of

‘/’ followed by a specific letter. `(i.e. /i)`

Pros	User would be able to have strings that may contain spaces (i.e. /n Chicken Salad /i Breast meat:2:meat)
Cons	The order of delimiters needs to be standardized, users will not be able to re-order the delimiters.

Alternative 2: Multiple prompts for user’s input of a recipe name and ingredient(s)

Pros	Users would not have to make sure that their command is syntactically right
Cons	The constant prompting could subject the application to a negative experience in the difficulty to use the commands.

Alternative 3: User’s command are divided by space

Pros	The parsing can be easily done by calling Java built-in function `.split()`
Cons	Values for each variable cannot contain spaces which makes the application restrictive.

4.2.3. Cooking of recipe

The feature allows the user to cook a recipe if there are sufficient ingredients. The user will also indicate how many pax this recipe would be cooked for.

Design considerations

Aspect: Preparing the deduction of ingredients when cooking a recipe

Alternative 1 (current choice): Checks for existence of recipe, existence of ingredients for the specified recipe and sufficiency of ingredients

Pros	Minimizes erroneous deduction of insufficient and nonexistent ingredients
	Cons

Alternative 2: Deductions are to be made to existing and available ingredients and users are notified when there are insufficient ingredients

Pros	Lesser overhead as there is lesser checks to be done
Cons	Hidden bugs and exceptions have to be well-covered to ensure that the deduction would be of the right value

Aspect: Searching for the corresponding ingredients of a recipe/ Searching through list of recipes to check for existence of recipe

Alternative 1 (current choice): Linear search, iterate through the arraylist of ingredients/ recipes and checking

Pros	Lesser use of complex data structure will save memory
Cons	Not optimal as search will be O(n), larger amount of data may take a longer time

Alternative 2: building an index on the first letter of the recipe name

Pros	More efficient search as pool of search space would be significantly smaller
Cons	Needs to be constantly maintained which incurs overhead.