Zhao Tianqi - Project Portfolio

PROJECT: Pet Store Helper

Overview

Pet Store Helper is a desktop application used for pet store owner to manage the store. The user interacts with it using a CLI, and it has a GUI created with JavaFX. It is written in Java, and has about 16 kLoC.

Summary of contributions

- **Major Enhancement**: **Added the command to display overall statistics**
  - What it does: Allows the user to take a look at the overall statistics of pets, recent schedule, and food data in the application.
  - Justification: This feature is important because it allows users, mostly pet store owners, to have an overview of the situation in his store. The owner can refer to the data to decide which species he should buy in more. Also, it allows him to have a rough idea of when are the busy hours in recent days, and the weekly food consumption as well. Also, it saves effort for users to click into different displays one by one to check the situation of the store.
  - Highlights: This feature uses a pie chart, a timetable, and a bar chart to show the statistics of pet, recent schedule, and food respectively. It is allows visualisation of the overall situation of the pet store in a user friendly manner.
  - Credits: Resizing of charts and panes as the window resizes. Source from the link: https://www.javacodegeeks.com/2014/04/javafx-tip-1-resizable-canvas.html
- **Minor enhancement**: Added warning messages to the user. When the user input contains more than 1 entry for name, species, date of birth, gender, slot time, and slot duration, there will be a warning message displayed to user. It states that the application only accepts the last entry for each field.
- **Code contributed**: [Functional code] [Test code]
- **Other contributions**:
  - Project management:
    - Managed releases `v1.3.1`, `v1.4` (2 releases) on GitHub
  - Enhancements to existing features:
    - Refactored Logic component of AB3 to adapt to Pet Store Helper. (Pull Request #15)
    - Added warning message (Pull request #81)
    - Changed find commands to filter pets that contain the keyword (instead of pets that has certain word which match the keyword exactly) (Pull Request #136)

        - Make find commands to switch display automatically. (e.g. `findslots n/keyword` will switch to display of filtered slots.) Allowed find slots by more than 1 dates (Pull Request #79)
        - Avoid confusion in index involved commands (i.e. edit and delete commands). Make sure the index refers to the whole list when commands is in different system with displayed screen. (e.g. editpet when screen is displaying inventory)(Pull Request #87)
        - Fix bugs reported after practical exam dry run (#134, #136, #133)
    - Documentation:
        - Did cosmetic tweaks to existing contents of the User Guide: #48
    - Community:
        - PRs reviewed (with non-trivial review comments): #61

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.

Statistics: `stats`

Written by Zhao Tianqi

Provides statistics about the pet tracker, schedule system, and inventory.

- There is a pie chart representing the ratio of different pet species.
- A timetable that shows an overall schedule for recent 3 days.
- A bar chart that shows the weekly consumption of different pet food.

Pet-related commands

Written by Zhao Tianqi

Adding a pet: `addpet`

Adds a pet to the pet tracker system.

Format: `addpet n/NAME g/GENDER b/DATE OF BIRTH s/SPECIES f/FOOD : AMOUNT [t/TAG]…`

- The date of birth must be in the format of d/M/yyyy, e.g. 01/01/2019, 1/7/2018
- The gender must be either `female` or `male`. Letter case does not matter.
- Food is specified as a type of food complied with quantity of weekly consumption in an arbitrary unit. The food name and amount should be separated by a colon ":". There can be more than 1 types of food for one pet.
- A pet can have any number of tags (including 0). Each tag must be restricted to one word .
- The application ignores letter case of user input. The name, species, gender of pets, and name of food will be displayed in the format of "Xxx Xxx ...".

Example:

- `addpet n/Garfield g/male b/01/01/2019 s/cat f/Brand A: 30 t/lazy t/lasagna`

Finding pets by names: `findpets`

Finds pets whose name contains any of the given keywords. The application will automatically change to the pet display system.

Format: `findpets PETNAME [MORE PETNAMES]…`

- At least one argument must be supplied.
- Pets matching at least 1 keyword will be returned (i.e. `OR` search).

Example:

- `findpets garfield odie`
  Returns a list of pets, whose names either contain `garfield` or `odie` or both.

Editing a pet: `editpet`

Edits any field of an existing pet in the system.

Format: `editpet INDEX [n/NAME] [g/GENDER] [t/TAG]…`

- If the app is displaying pets, the index refers to the index number shown in the displayed pets list, and must be a positive integer, e.g. 1, 2, 3, … Otherwise, the index refers to the number in the whole pet list.
- The existing field(s) of the pet will be removed, i.e adding of list of food and tags is not cumulative.
- You can remove all tags of a pet by typing `t/` without specifying any tags after it.
- Similarly to `addpet` command, pet name, species, gender, and name of food will be displayed in the format of "Xxx Xxx…".

Example:

- `display p`
  `editpet 2 n/Coco b/02/01/2020 t/cuddly t/grey`
  Overwrites information of the 2nd pet in the system with name "Coco", date of birth "2 Jan 2020, and 2 tags of “cuddly”, “grey”.
- `findpets garfield`
  `editpet 2 n/Coco`
  Overwrites the name of the 2nd pet in the results of `findpets garfield` to "Coco"
- `display s`
  `editpet 1 n/garfield` Overwrites name of the 1st pet in the whole pet list to be "Garfield".

Deleting a pet: `deletepet`

Deletes the specified pet from the system.

Format: `deletepet INDEX`

> - If the app is displaying pets, the index refers to the index number shown in the displayed pets list, and must be a positive integer, e.g. 1, 2, 3, … Otherwise, the index refers to the number in the whole pet list.

Examples:

- `display p`
  `deletepet 2`
  Deletes the 2nd pet in the system.
- `findpets n/garfield`
  `deletepet 2`
  Deletes the 2nd pet in the results of the `findpets garfield` command.
- `display i`
  `deletepet 2` Deletes 2nd pet in the whole pet list.

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.

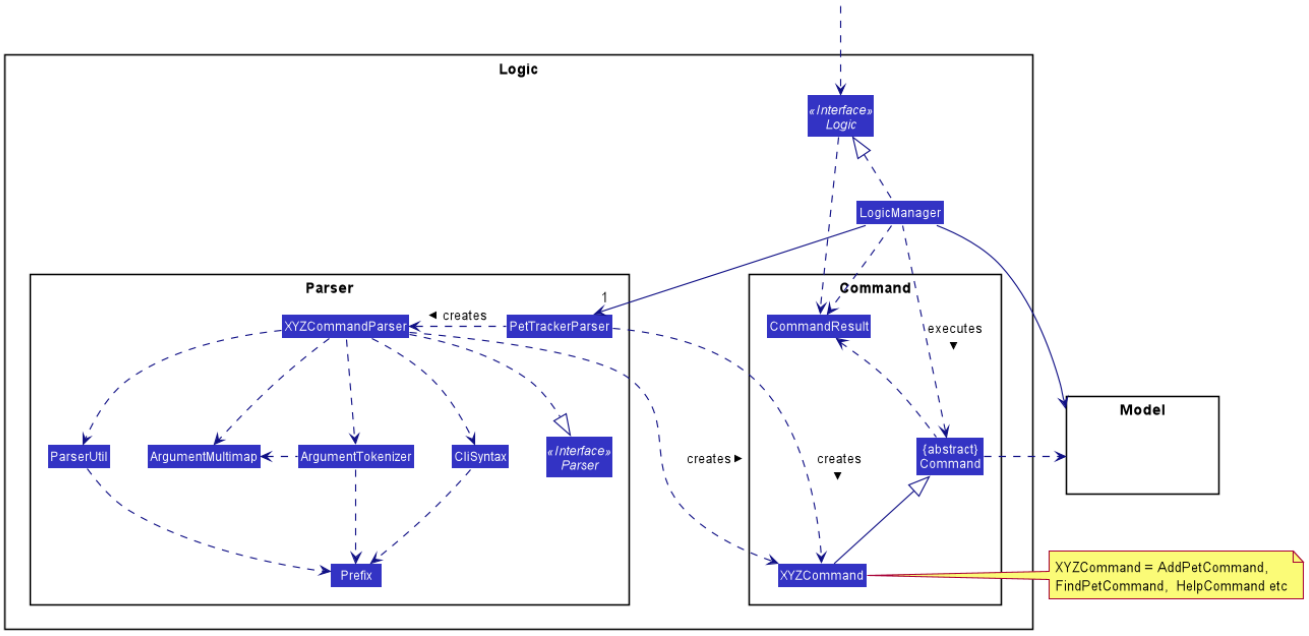

Figure 1. Class Diagram of the Logic Component

Statistics

Written by Zhao Tianqi

Implementation

We are generating the overall statistics of Pet Store Helper and translate the data in a user-friendly manner.

- `OverallStats` under UI component handles the translation of three sets of data: list of pets, schedule, and list of inventory.
- Data in `OverallStats` is obtained from `Logic`.
- The statistics displayed will automatically update if there is a change in any related information.

How we implemented overall statistic on UI:

- The statistics for pets are shown in a form of pie chart, while the pets are grouped according to their species.
- The schedule statistics is in the form of a timetable of recent 3 days. Each slot is represented as a shaded rectangle in the timetable.
- The inventory data are generated from the list of pets, and grouped together by their names, such that users have a better understanding of overall food consumption. The list of inventory is represented as a bar chart.

Here is the process of how the overall statistics is displayed to the user:

- Step 1: The user key in the command 'stats', then the Logic component processes the input and creates an instance of `StatsCommand`. The `StatsCommand` first calls `Model#updateAll()` to make sure

that the model will update `filteredPets`, `filteredSlots`, and `filteredFoodCollections` to show the full list of `Pet`, `Slot` and `FoodCollection` in the `PetTracker`. After that, the `StatsCommand` calls `Model#changeDisplaySystem(DisplaySystemType.STATISTICS)` such that the application will later switch the window to show `OverallStats`. In the Ui component, there is a new instance of `OverallStats` created, while `Model#getFilteredPets`, `Model#getFilteredSlots`, and `Model#getFilteredFoodCollections` are passed in the instance for processing. `OverallStats` then generates diagrams according to the data passes in.

- Step 2: The user inputs a command that modifies the UniquePetList, e.g 'editpet 1 s/cat'. The `CommandResult` of any `EditPetCommand` has `type` with `DisplaySystemType.NO_CHANGE`. The Ui component identifies the `type` in the `CommandResult` and then refresh the window if it is `NO_CHANGE`. In this case, the window will refresh and create another new instance of `OverallStats` with the updated list of pets. Therefore the window always shows statistics of the most updated list of pets, slots and food collections.

Following is the sequential diagram of the command `stats`

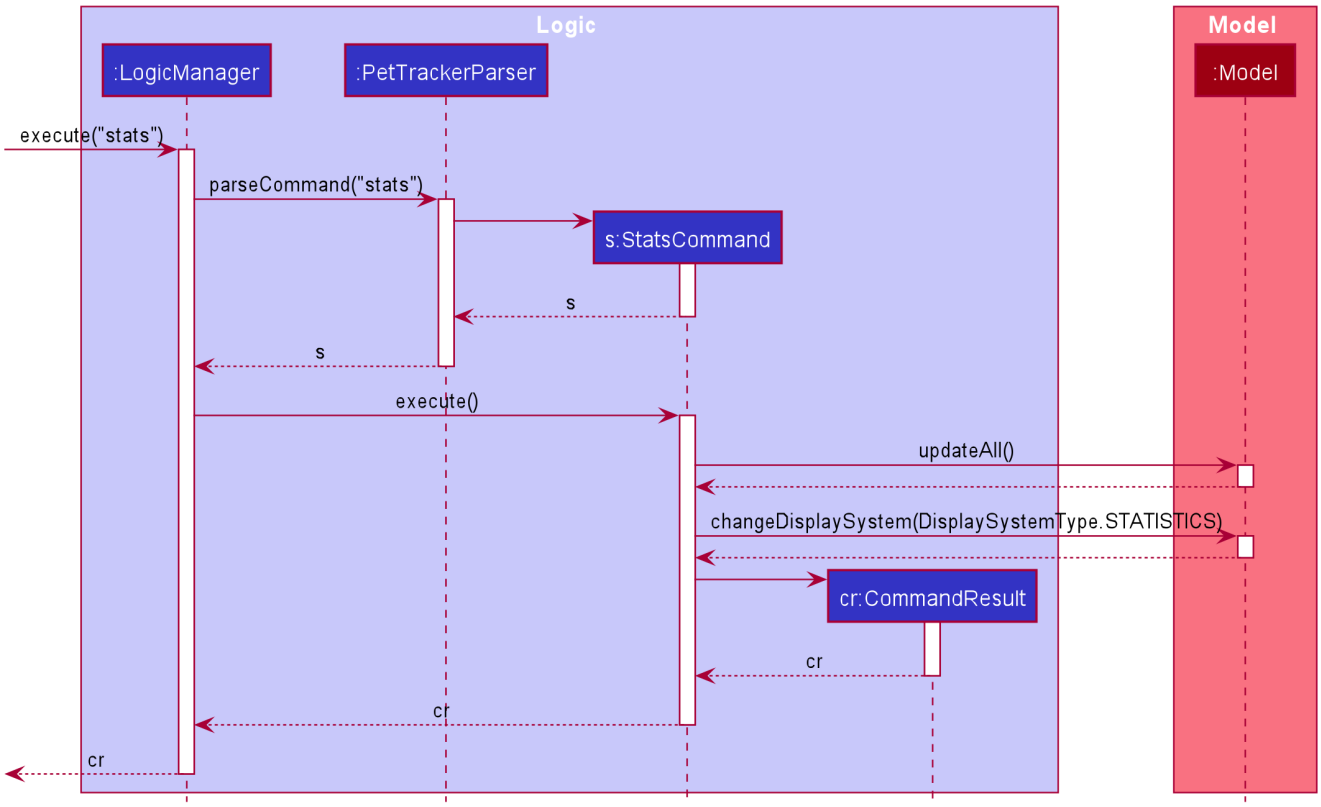

Figure 2. Interactions Inside the **Logic** *Component for the* `stats` *Command*

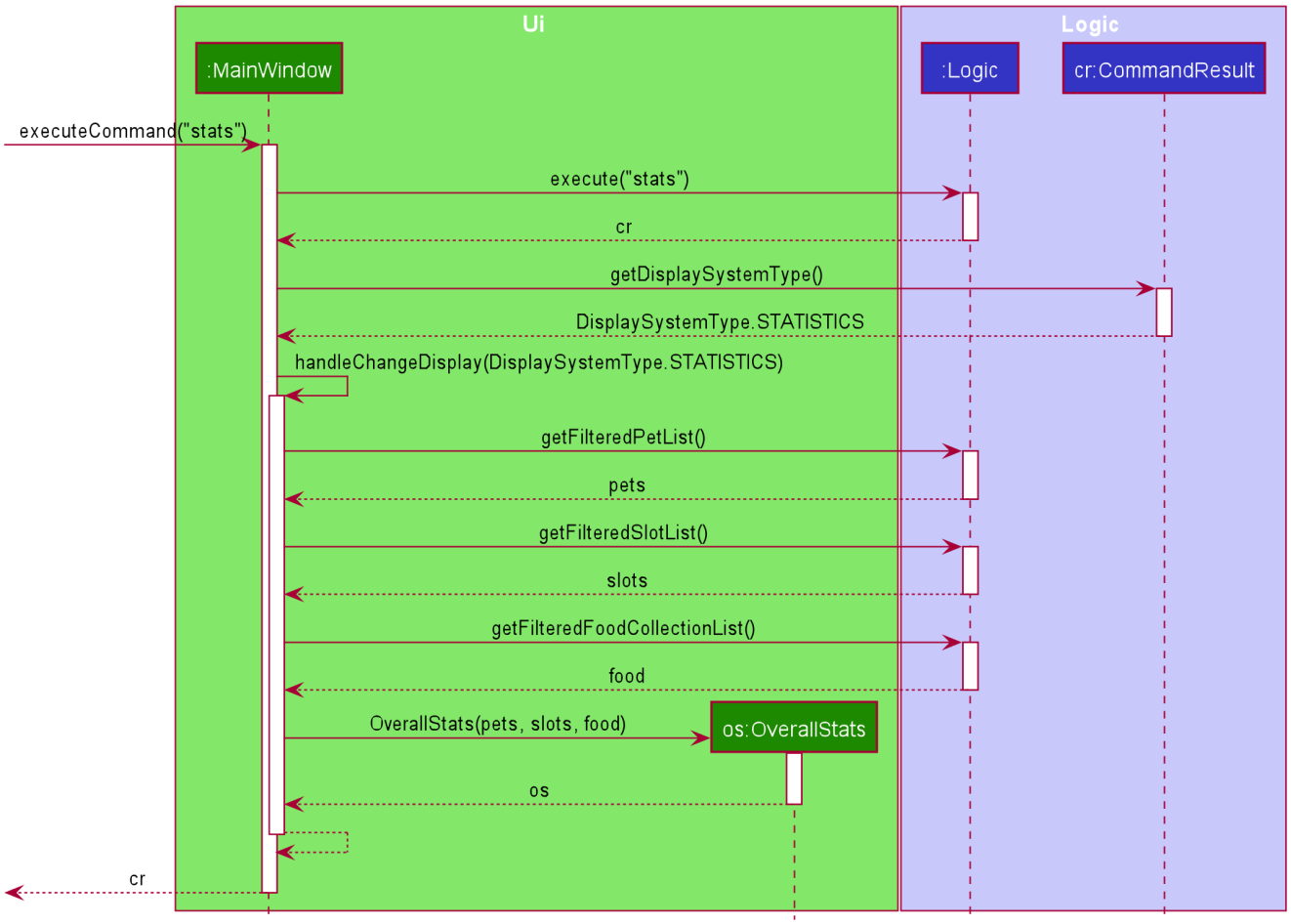

Figure 3. Interactions Inside the **Ui** *Component for the* `stats` *Command*

Design Considerations

Aspect: How to update diagrams when there are changes

- Alternative 1 (current choice): Refresh the screen
  - Pros: Easy to implement and only need to refer to `model` to get data.
  - Cons: Takes time to process the whole lists of data even though there is only one small change in one of the item (pets, slots or foo collection). This approach might be time consuming when the data size is too large.
- Alternative 2: Make use of `Listener` to detect change in `UniquePetList` and make changes accordingly
  - Pros: Avoid unnecessary processing of data. e.g. The diagrams need not be regenerated when there is no change in pet species, pet food, and recent schedule.
  - Cons: More complicated implementation. There might be more coupling between `OverallStats` under Ui component and Listener class under Model component.

Conflict feature

Written by Zhao Tianqi

Implementation

The conflict feature shows a list of all slots that has overlapping time period.

- It makes uses of `SlotConflictPredicate` which implements `SlotPredicate` to filter slots that has a conflicting timing with some other slots.
- If the screen is displaying shortlisted slots, `conflics` only shows slots with conflicts among the shortlisted list. Otherwise, it shows conflicts in the full slot list
- The command automatically switch display to slots.

Below is a scenario of using `conflicts` command:

- Step 1: The user inputs "conflicts" in the command line. The Logic component processes the input and creates a new instance of `ConflictCommand`
- Step 2: The `ConflictCommand` access `filteredSlots` in `Model`, and create a new instance of `SlotConflictPredicate` with `filteredSlots`.
- Step 3: `Model` filters slots that is conflicting among the `filteredSlots` using the `SlotConflictPredicate`.
- Step 4: `ConflictCommand` changes `currentDisplaySystemType` in `Model` to `DisplaySystemType.SCHEDUEL` such that the window will display shorlisted conflicting slots.

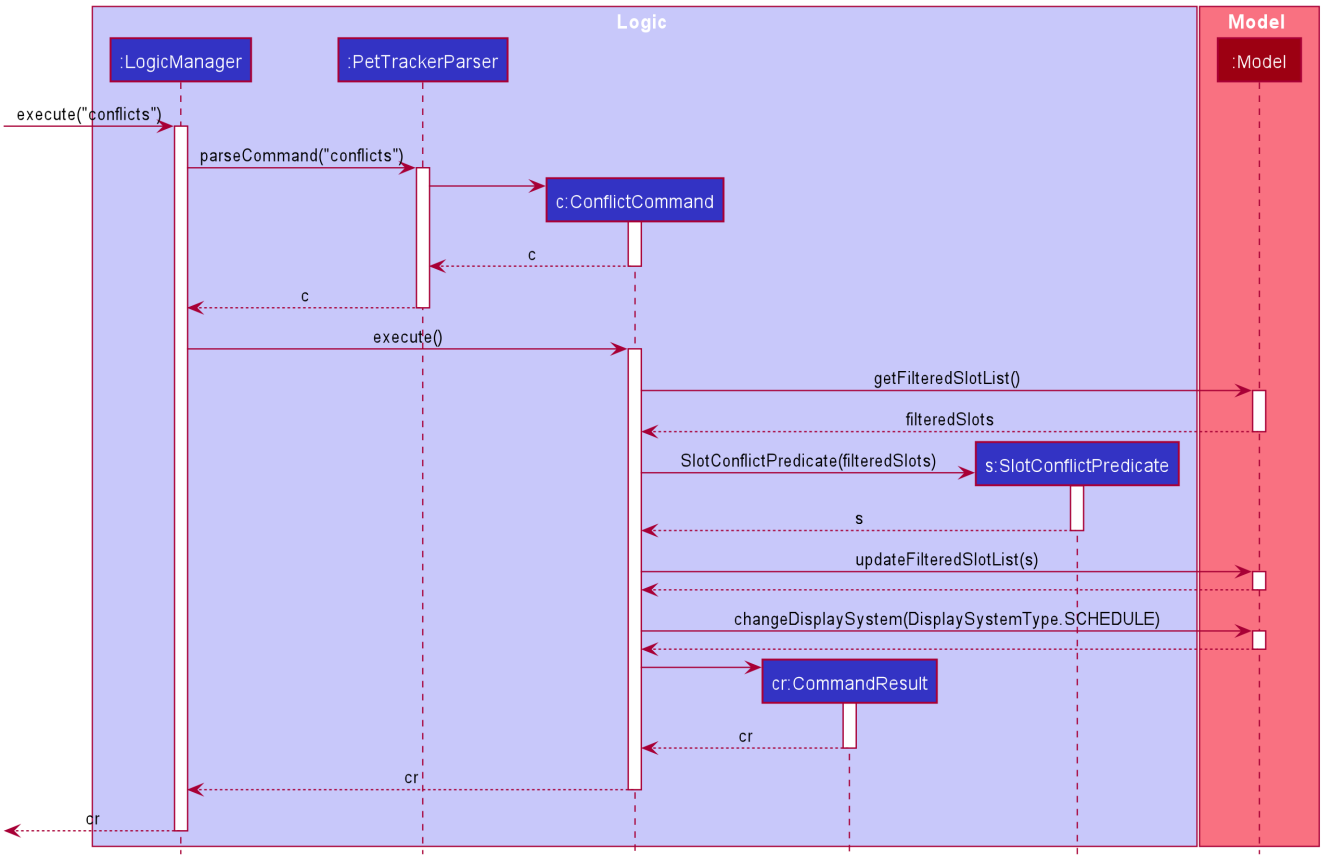

Figure 4. Interactions Inside the Logic Component for the `conflicts` Command

Design Considerations

Aspect: How to filter conflicting slots

- Alternative 1 (current choice): Make use of Predicate
  - Pros: More organised and more object-oriented style of programming. Also, the use of predicate can be extended to other commands, e.g. `findslots`.
  - Cons: Relatively more complicated to implement
- Alternative 2: Directly check for conflicting timing among the list of slots
  - Pros: More straightforward in implementation
  - Cons: Low level of OOP. It should not be encouraged.