Zhu Ruicong - Project Portfolio

PROJECT: Pet Store Helper

Overview

Pet Store Helper(PSH) is a desktop application used for pet store owners to manage different aspects of their pet store including pet logging, scheduling and inventory management. The user interacts with it using a CLI, and it has a GUI created with JavaFX. It is written in Java, and has about 15kLOC.

Summary of contributions

- **Major enhancement**: **added the inventory display system**
  - What it does: allows the user to view the summary of weekly food types and amounts required to maintain the pets in the store.
  - Justification: This feature improves the usefulness of the product for pet store owners significantly as it helps the user to visualize, understand and organize food consumptions in the pet shop.
  - Highlights: This enhancement uses a split screen which led to the redesign of the UI component. A major challenge tackled was the synchronization of data of the Inventory system with the Pet system, which required an in-depth analysis of design alternatives of the data model.
- **Minor enhancement**: added a feature where users can click on each food item in the inventory to view the breakdown of food amounts by pet names.
- **Code contributed**: [Functional code] [Test code]
- **Other contributions**:
  - Contribution to Documentation:
    - Maintained the AboutUs page.[#6]
    - Maintained the Model component(2.4) of the Developer Guide.[#74]
    - Maintained Inventory feature(3.3) of Developer Guide.[#74]
    - Compiled and edited the materials for the Final Demo.
  - Enhancements to existing features:
    - Refactored Model package of AB3 and UI component for the use of PSH[#13][#19]
    - In charge of resolving major integration issues at the end of the morphing phase.[#35]
  - Community:
    - PRs reviewed (with non-trivial review comments): #23, #48, #64, #78, #137

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.

Model component

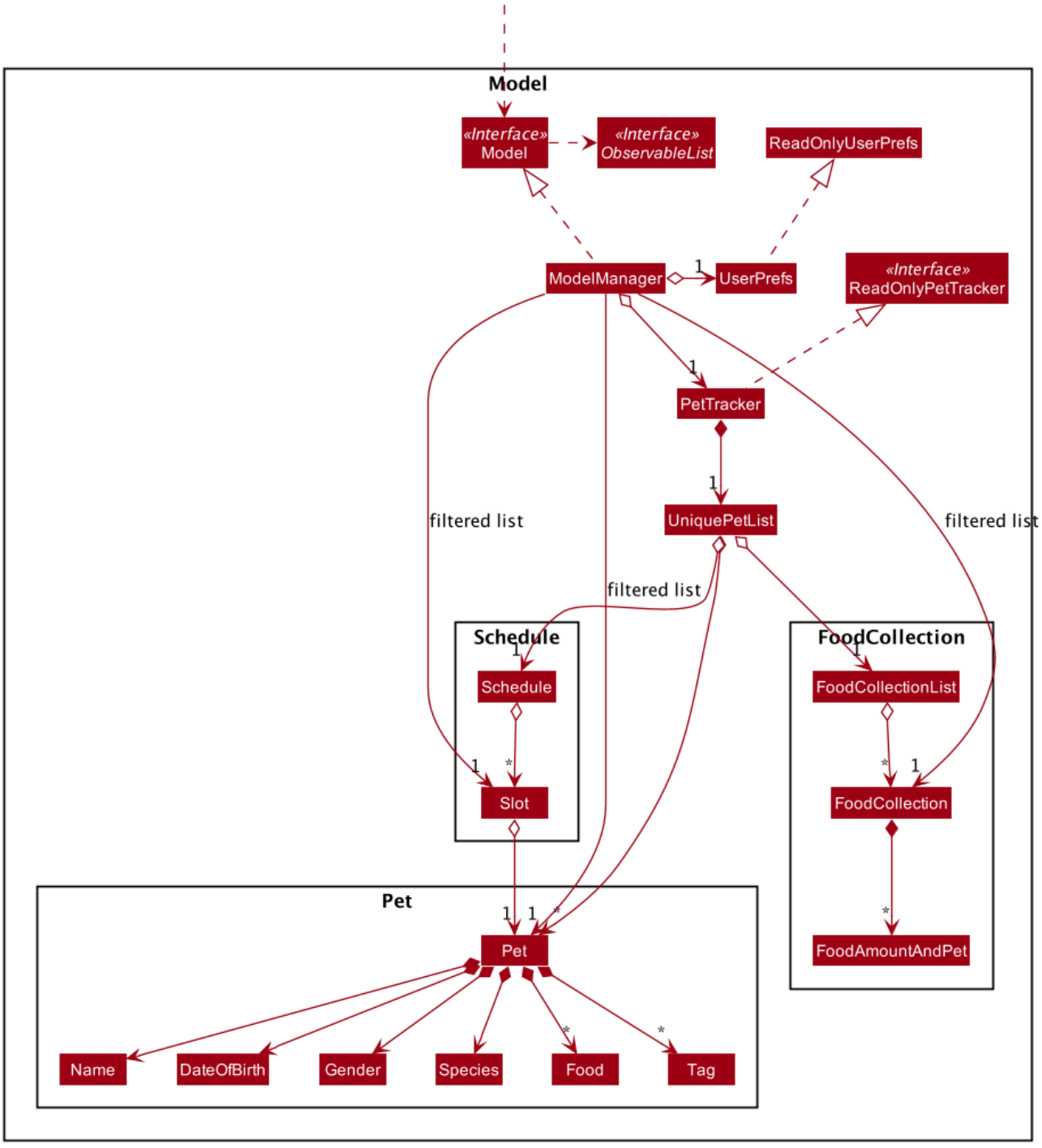

Figure 1. Structure of the Model Component

API : `Model.java`

The `Model`,

- stores a `UserPref` object that represents the user's preferences.
- stores the Pet Tracker data. Note that the Pet Tracker keeps track of both the schedule system and the pet system.
- exposes an `ObservableList<Pet>`, an `ObservableList<Slot>`, and an `ObservableList<FoodCollection>` that are unmodifiable and can be 'observed' e.g. the UI can be bound to this list so that the UI automatically updates when the data in the list change.
- does not depend on any of the other three components.

Inventory feature

Implementation

The Inventory feature gives a summary of all the food items involved in a pet tracker system. It is supported by `FoodCollection` which resembles a collection of food of the same type and `FoodCollectionList` which is a list of these collections. A `FoodCollectionList` is stored as an attribute of `UniquePetList` for the following reasons:

- The list of `FoodCollection` items associated with a `UniquePetList` can be directly derived from the `UniquePetList` itself.
- Changes in FoodCollection occurs only if there is a change in `UniquePetList#internalList`.

Data stored in `FoodCollectionList` is exposed to `ModelManager` through `UniquePetList` and `PetTracker` as an unmodifiable `ObservableList<FoodCollection>`. `ModelManager` then passes the list of `FoodCollection` to ui for display as a list of `DisplayItem` when `display i` is called.

The following shows a typical usage scenario that involves the Display Inventory feature.

- Step 1: The user launches the application. A `UniquePetList` is initialized in `PetTracker`, upon which a `FoodCollectionList` item is created to store the food data of the pets in the list(if it is an empty list, `FoodCollectionList` is also stores an empty list of `FoodCollection`)
- Step 2: The user executes 'display i' command. The `display` command calls Model#ChangeDisplaySystem() and the `i` display type determines the displayed list is switched to `ObservableList<FoodCollection>`. `Model$getFilteredDisplayList()` then acquires the list and sends it to Ui unit for display.
- Step 3: The user inputs a command that modifies the `UniquePetList`, e.g 'editpet 1 f/catfood:100'. `UniquePetList$internalList` is an instance of `ObservableList<Pet>`. Thus when it is modified, a `ListChangeListener<Pet>` is woken up and it calls `UniquePetList$updateFoodCollectionList()` to update the `FoodCollectionList` according to the modified Pet list.

The sequence diagram below is an illustration of the flow of events that happen in the logical component when `Step 2` above occurs.

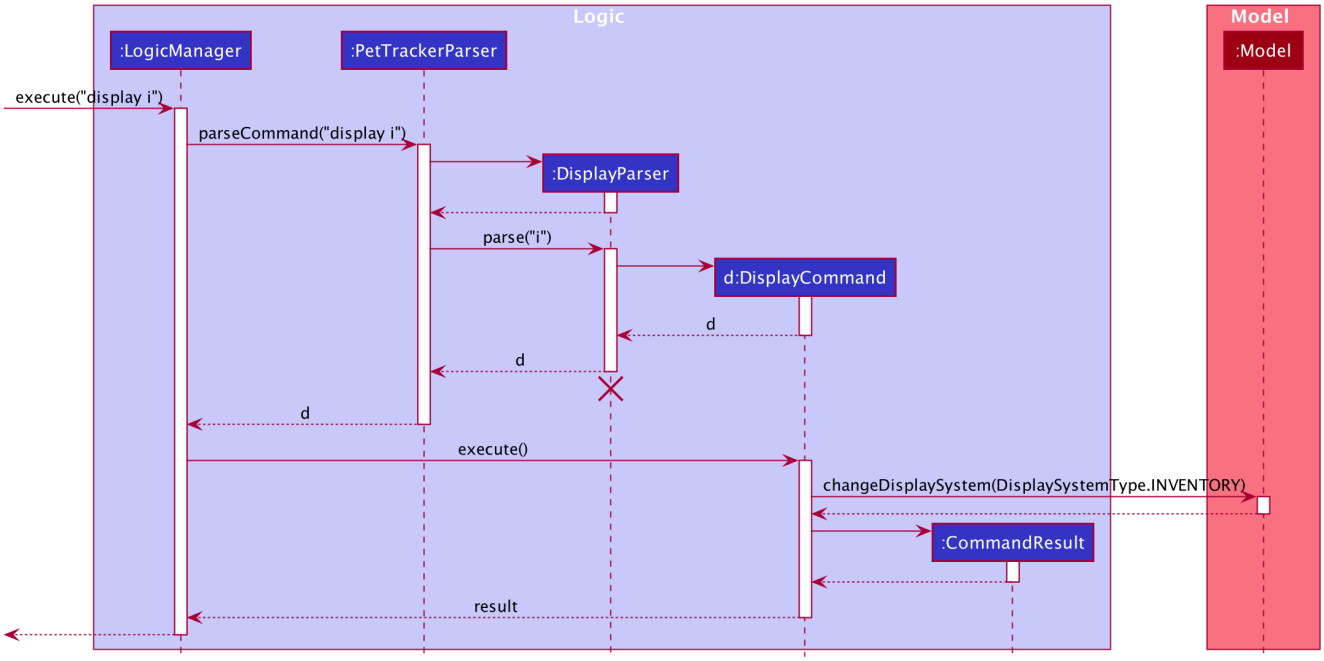

Figure 2. Interactions Inside the Logic Component for the `display i` Command

Design Considerations

Aspect: Maintaining the collection of food in a pet tracker

- Alternative 1(current choice): Maintains the list as an attribute of `UniquePetList`.
  - Pros: Easier to initialize and update the list.
  - Cons: Less extendability. Adding additional food items in inventory(independent of pet list) is difficult.
- Alternative 2: Maintains a list of food collections separate from `UniquePetList`.
  - Pros: Higher Extendability that supports more independent operations of FoodCollection List.
  - Cons: More difficult to constantly update and maintain the food collection list should food list changes.

Aspect: Updating the collection of food when pet list is modified.

- Alternative 1(current choice): Replace the entire list by a new food collection list created from the updated pet list.
  - Pros: Easy to implement and no adaptation is required for different types of modification of pet list.
  - Cons: Computationally intensive when there is a huge pet list.
- Alternative 2: Modify `FoodCollection` affected by the command.
  - Pros: Less computationally intensive and more responsive given a large database.
  - Cons: Adaptations for each pet related commands is required since the food list can be affected in different ways.(e.g addition, modification, deletion)