Yen Peng - Project Portfolio

Project Portfolio for ORGANice

The purpose of this project portfolio is to present my main contributions to ORGANice.

1. Introduction

This section covers the project scope and description, followed an explanation of the formats used.

1.1 Project Scope

ORGANice is the product of a software engineering project done by a team of 5 students, including myself, who are second-year Computer Science students from the National University of Singapore (NUS). Under a two-month-long time frame, we conceptualised ORGANice and molded it from an existing application, known as "Address Book", that stores and manages people's contact information. My role in this project is to design and write the code for the `list` and `sort` features for ORGANice, and to offer advice for the design aspect of the application. For the languages used, ORGANice is written in Java while its Graphical User Interface (GUI) is written in JavaFX.

1.1 Project Description

ORGANice is a Command Line Interface (CLI) desktop application, meaning that user interaction is mainly through typing in commands. This application is targeted at hospital administrators and its main features include storing hospital information, finding compatible donors for patients awaiting organ transplants, and following up on administrative matters related to the organ matching process.

ORGANice has the following features:

- Maintain the information of doctors, donors, and patients using `add` and `edit`
- Find information quickly using `find` and `list`
- Match compatible organ donors with patients and sort the matches using `match` and `sort`
- Keep track of patient-donor pairs slated for further compatibility tests using `processing` and subsequent administrative tasks using `processingmarkdone`
- Mark patient-donor pair as `done` to remove from future matching results

1.2 Meaning of the Formats Used

The following is a list of various formats used and what they mean:

- `sort` — Purple texts are technical terms used in this project.
- #153 — Blue underlined texts consist of a link that opens up a relevant webpage.

2. Summary of Contributions

This section elaborates on the enhancements I did to ORGANice with the relevant links (in blue).

- **Major enhancement**: Added the ability to sort matches according to different criteria. (Pull request #145)
    - What it does: Allows the user to sort patient-donor pairs according to compatibility rate, the priority of the patients, or donors' organ expiry date.
    - Justification: Helps the hospital to make more informed decisions after knowing which pairs are the most compatible, which patient to prioritize, and which organs are expiring soon respectively.
    - Highlights: Able to add in more sorting options in future enhancements. The implementation is challenging as it required an in-depth analysis of how the match feature is implemented.
- **Minor enhancement**: Added a list feature that allows the user to list down all persons or persons of a specified type in ORGANice. (Pull request #113)
- **Minor enhancement**: Added an organ class to ORGANice. (Pull request #86)
- **Minor enhancement**: Designed the logo of ORGANice, and updated the application window to display title and logo. (Pull request #136)
- **Code contributed**: You may click on the following link to see a sample of my code. (Functional and Test code)
- **Other contributions**:
    - Project Management
        - Created issues on the project board to keep track of tasks to be done.
    - Enhancements to existing features
        - Fixed bug related to the list feature. (Pull request #236)
    - Documentation
        - Updated information about our team in the "About Us" section of ORGANice. (Pull request #11)
        - Authored the sections that guide users through the features 'Edit', 'List', 'Sort' and 'Help' in the User Guide. (Pull request #133)
        - Authored the implementations and use cases of the 'List' and 'Sort' features in the Developer Guide. (Pull requests #131, #153)
    - Community
        - Reviewed pull requests with non-trivial review comments. (Pull requests #148, #150)

3. Contributions to the User Guide

This section showcases my ability to write documentation targeting end-users through an excerpt of what I have written for the User Guide, and this includes the `sort`, `list`, `help` and `edit` features.

{Start of the excerpt from User Guide}

Sorting matches: `sort`

After using the match command, you can sort the list of matches to determine which patients to prioritize for an organ transplant.

The sort command can only be used after a match command to sort the resulting list of patient-donor pairs generated. There are three criteria in which you can sort the matches, namely:

- Sorting based on donor's organ expiry date
- Sorting based on priority
- Sorting based on the compatibility rate of match

These criteria will be further explained below.

Note that **only one criterion can be used in a single sort command**.

Sorting based on donor's organ expiry date

To find out which organs are expiring soon, you can sort the list of matched donors based on the donor's organ expiry date (from the earliest to the latest expiry date).

Format: `sort expiry`

For example, after running a match command on a specific patient with `match ic/NRIC`, you will get the list of donors that are compatible with said patient. You can then run `sort expiry` to obtain the list of donors with earlier organ expiry dates near the top of the list.

Sorting based on priority

To determine which patient to prioritize for an organ transplant, you can sort the list of matched patients based on their priority (from highest to lowest). You may refer to [Glossary] to understand what priority means.

Format: `sort priority`

For example, after running `match ic/all`, you will obtain a list of patients. You can then run `sort priority` to obtain the list with higher priority patients near the top of the list.

Note that in the case where patients have the **same priority**, those with more matched donors will be displayed first. If these two factors are the same for a group of patients, they will be displayed according to their names in alphabetical order.

Sorting based on the compatibility rate of match

To determine which patient-donor pair is the most compatible, you can sort the list of compatible donors of that patient based on the compatibility rate of the match (from highest to lowest). You may refer to [Glossary] for more information on the compatibility rate.

Format: `sort rate`

Take for instance, after running a match command on a specific patient with `match ic/NRIC`, you will get a list of donors that are compatible with said patient. You can then run `sort rate` to obtain the list of donors with higher compatibility rates near the top of the list.

Listing persons: `list`

This list command can show you the list of persons in ORGANice.

Listing all persons

Simply type `list` in the Command Box and press kbd:[Enter]. ORGANice will show you all patients, doctors, and donors present in the system.

Format: `list`

Listing persons of a specified type

If you wish to see a certain type of person only, you can provide the type parameter in the list command.

Optional parameter: Type - `t/TYPE`

For instance, if you only want to see all doctors, you can enter the following: `list t/doctor`. Likewise, you can change the type parameter in the same manner to see only patients or donors.

Viewing help: `help`

There may be times where you need more information on how ORGANice works. To access the user guide of ORGANice, you can do the following:

Simply type `help` in the Command Box and press kbd:[Enter]. You will see a link given in a pop-up "Help" window. You may click on the kbd:[Copy URL] button and paste it in a browser to see our user guide for more information on how to use ORGANice.

To exit the window, click the kbd:[x] button found near the top of the window to close it and return to the default display.

Editing a person's attribute: `edit`

If you want to make any changes to a person's attribute, you can edit the information using an edit command. If there are multiple attributes to change, you can do so in the same edit command.

The format is such that after the `edit` keyword, provide the person's NRIC, followed by the attribute(s) that you want to change in the edit command.

Format: `edit NRIC` followed by attributes to update

For instance, if you want to change a patient's phone number to 91234567, you can provide his or her NRIC followed by the new phone number as follows: `edit S8732457G p/91234567`.

You can only make changes to attributes that the person has. For example, you cannot change a doctor's age because a doctor does not have that attribute in ORGANice.

Besides that, some attributes cannot be edited. These include:

- NRIC
- Type

You may want to refer to [List of Attributes] to see what attributes apply to each type of person in ORGANice.

{End of the excerpt from User Guide}

4. Contributions to the Developer Guide

This section showcases my ability to write technical documentation and to visually represent a system using Unified Modelling Language (UML) diagrams in the Developer Guide. The sequence shown in the excerpt below will be use cases for `list` and `sort` features, followed by their implementations, and lastly, the `Model` class diagram that I have updated.

{Start of the excerpt from Developer Guide}

Use case: List all patients/donors/doctors

MSS

1. User requests to list patients/donors/doctors.
2. ORGANice shows the list of patients/donors/doctors.

   Use case ends.

Extensions

1a. ORGANice detects an invalid parameter.

1a1. ORGANice returns error message.

Use case ends.

Use case: Sort list according to priority/rate/expiry

MSS

1. User requests to sort the list of donors of a particular patient.
2. ORGANice sorts the list of donors according to the parameters.

   Use case ends.

Extensions

1a. ORGANice detects invalid parameter(s).

1a1. ORGANice prompts for valid parameter(s).

1a2. User enters correct parameter(s).

Steps 1a1-1a2 are repeated until the data entered are correct.

Use case resumes at step 2.

List feature

This section describes how the list feature is implemented.

The list feature is implemented using the `ListCommand(Type type)`. When the list command is executed, the method `ModelManager#updateFilteredPersonList(Predicate<Person> predicate)` takes in a predicate to decide what type of `Person` to show on the `filteredPersons` list.

Therefore, in addition to the existing `PREDICATE_SHOW_ALL_PERSONS` predicate, three other predicates are introduced in the `Model` interface to update the list.

The code snippet below shows what these predicates are:

```
    /** {@code Predicate} that always evaluate to true */
    Predicate<Person> PREDICATE_SHOW_ALL_PERSONS = unused -> true;

    /** {@code Predicate} that always evaluate to true if person is a doctor */
    Predicate<Person> PREDICATE_SHOW_ALL_DOCTORS = person -> person.getType().
isDoctor();

    /** {@code Predicate} that always evaluate to true  if person is a donor */
    Predicate<Person> PREDICATE_SHOW_ALL_DONORS = person -> person.getType().isDonor(
);

    /** {@code Predicate} that always evaluate to true if person is a patient */
    Predicate<Person> PREDICATE_SHOW_ALL_PATIENTS = person -> person.getType()
.isPatient();
```

The activity diagram below illustrates what happens when a user uses the list command.

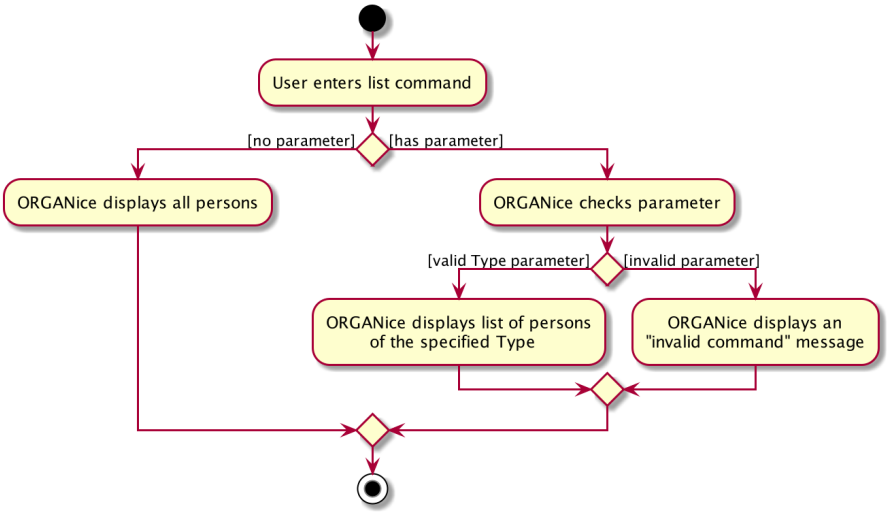

Figure 1. Activity Diagram of a List Command

When the user enters a list command, the result differs depending on the parameter of the command. If there is no parameter after 'list', ORGANice will simply list all existing doctors, patients, and donors on the list. If there is a parameter, ORGANice will check if it is a valid `TYPE` parameter and display the correct list of persons that corresponds to this type. If the parameter is not valid, an error message will be displayed to indicate that the command is invalid.

Sort feature

This section describes how the sort feature is implemented.

Sorting is done by using `Comparators` in the `Logic` component and makes use of the `SortCommand`. This command only works after a `MatchCommand` is called by the user, because it takes in `listOfMatches` returned from calling a `MatchCommand`. Importantly, this `listOfMatches` is an `ObservableList`, which cannot be modified. Therefore, it makes use of `SortedList` to wrap around the `ObservableList` so that the list can be modified by sorting. Thus far, the sort command can only take in one `String` parameter, but more methods can be added in the future for more sorting options.

Three methods of sorting are implemented to sort based on the following categories:

- Sorting based on donor's organ expiry date with `sort expiry`
- Sorting based on priority with `sort priority`
- Sorting based on compatibility rate of the match with `sort rate`

These three methods' implementation will be further elaborated upon below.

Sort by Expiry

Sorting by Expiry sorts a list of `MatchedDonor` with the `ModelManager#sortByOrganExpiryDate()` method.

In this method, the `ExpiryComparator` is used to compare two matched donors' organ expiry dates, and this requires the `OrganExpiryDate` of the donors to be parsed into a data format that can be compared.

Sort by Compatibility Rate

Sorting by Compatibility Rate sorts a list of `MatchedPatient` using the `ModelManager#sortBySuccessRate()` method.

The `CompatibilityRateComparator` is created to compare two matched patients' rate of compatibility for the match.

Given below is a Sequence Diagram to show the interactions with the `Logic` component when executing the `execute(sort rate)` method, assuming the user has entered a valid `match ic/NRIC` command before this.

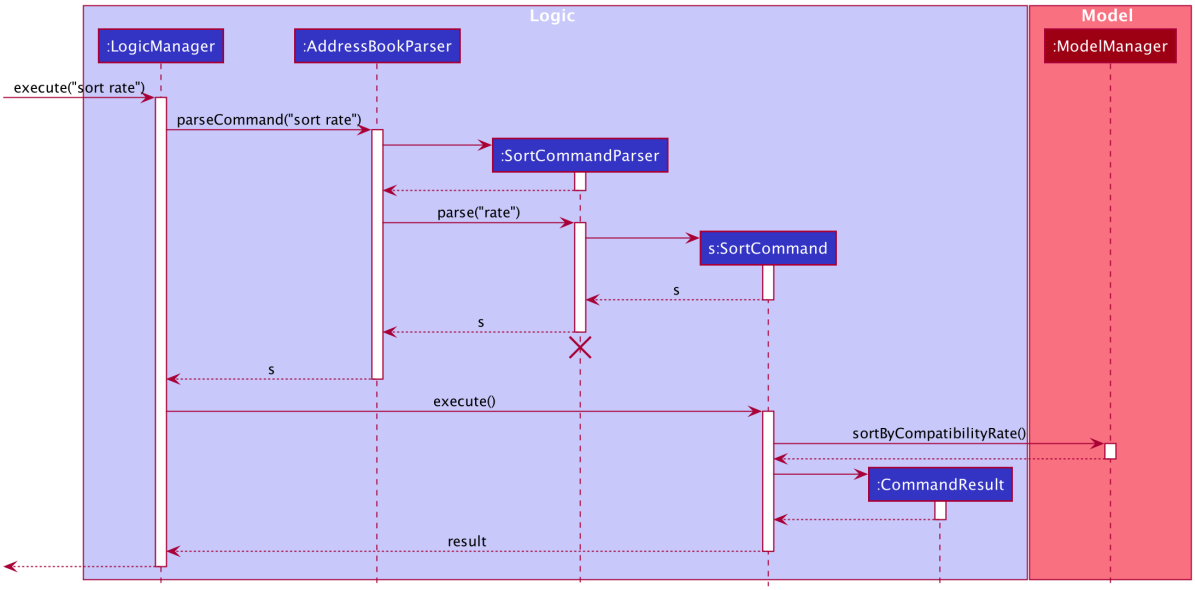

Figure 2. Sequential Diagram of the `sort rate` command

After the command is given by the user, it would be parsed by the `AddressBookParser`. The `SortCommandParser` then parses the parameter `rate` and a `SortCommand` is created. The `LogicManager` will then execute the command and call the `sortByCompatibilityRate()` method from the `ModelManager`. The list of `MatchedPatient` will then be sorted based on their match's compatibility rate.

Sort by Priority

Sorting by Expiry sorts a list of `MatchedPatient` using the `ModelManager#sortByPriority()` method.

In this method, there are two other comparators created in addition to the `Priority Comparator`, namely the `NameComparator` and the `NumOfMatchesComparator`. These two comparators serve as tie-breakers in cases where patients have the same priority. In other words, for patients of the same priority, those with more matched donors will be displayed first. If these two factors are the same for a group of patients, they will be displayed according to their names in alphabetical order.

Under the `sortByPriority()` method, the order of applying the comparators to the list of patients is, firstly, the 'NameComparator', followed by `NumOfMatchesComparator`, and lastly, the `PriorityComparator`. This means that the patients are first sorted by names in alphabetical order, followed by the number of matched donors they have in descending order (highest to lowest), and then their priority level in descending order (from high, to medium, to low).

All three comparators makes use of the `compareTo` method under the `Comparable` class, which can sort in lexical order for String, and numeric order for Sorting integers. In particular, the `PriorityComparator` makes use of the `compareTo` method implemented in the `Priority` class. A code snippet of this method is shown below.

```
@Override
public int compareTo(Priority priority) {
Integer thisPriorityNumber = 0;
Integer otherPriorityNumber = 0;

        if (this.isHighPriority()) {
            thisPriorityNumber = 3;
        } else if (this.isMediumPriority()) {
            thisPriorityNumber = 2;
        } else if (this.isLowPriority()) {
            thisPriorityNumber = 1;
        }
        if (priority.isHighPriority()) {
            otherPriorityNumber = 3;
        } else if (priority.isMediumPriority()) {
            otherPriorityNumber = 2;
        } else if (priority.isLowPriority()) {
            otherPriorityNumber = 1;
        }
        return thisPriorityNumber.compareTo(otherPriorityNumber);
    }
```

This implementation is the result of using `String` to represent the different priorities in the `Priority` class. Therefore, they are converted into `Integer` for easier comparison. An alternative solution would be to make use of `Enum` class to define each priority so that this `compareTo` method implementation can be simplified.

Design considerations

This section will explain two aspects considered in designing the sort feature.

Aspect: Method of storing the sorted result

- **Alternative 1 (current choice):** Use `SortedList`.
    - Pros: Fast iteration time of O(n * log n), which is important for ORGANice to load and display the sorted result, and more harmonious with current implementations (able to wrap `ObservableList` and sort the content).
    - Cons: Not the most efficient algorithm for insertions or deletions, but ORGANice does not require them.
- **Alternative 2:** Use `ArrayList`.
    - Pros: Fast index-based access and works on any `Collection` class.
    - Cons: Slower than `SortedList` as it takes O(n) time to iterate and display the results.

Aspect: Method of sorting

- **Alternative 1 (current choice):** Use `Comparator` and `Comparable`.
  - Pros: Simpler implementation as able to make use of existing methods in ORGANice such as `setComparator`.
  - Cons: More coding involved.
- **Alternative 2:** Use `Collections.sort()`.
  - Pros: Has a time complexity of O(n*log(n)) which is relatively fast.
  - Cons: Cannot make use of existing methods in ORGANice that work for Comparator.

Model component

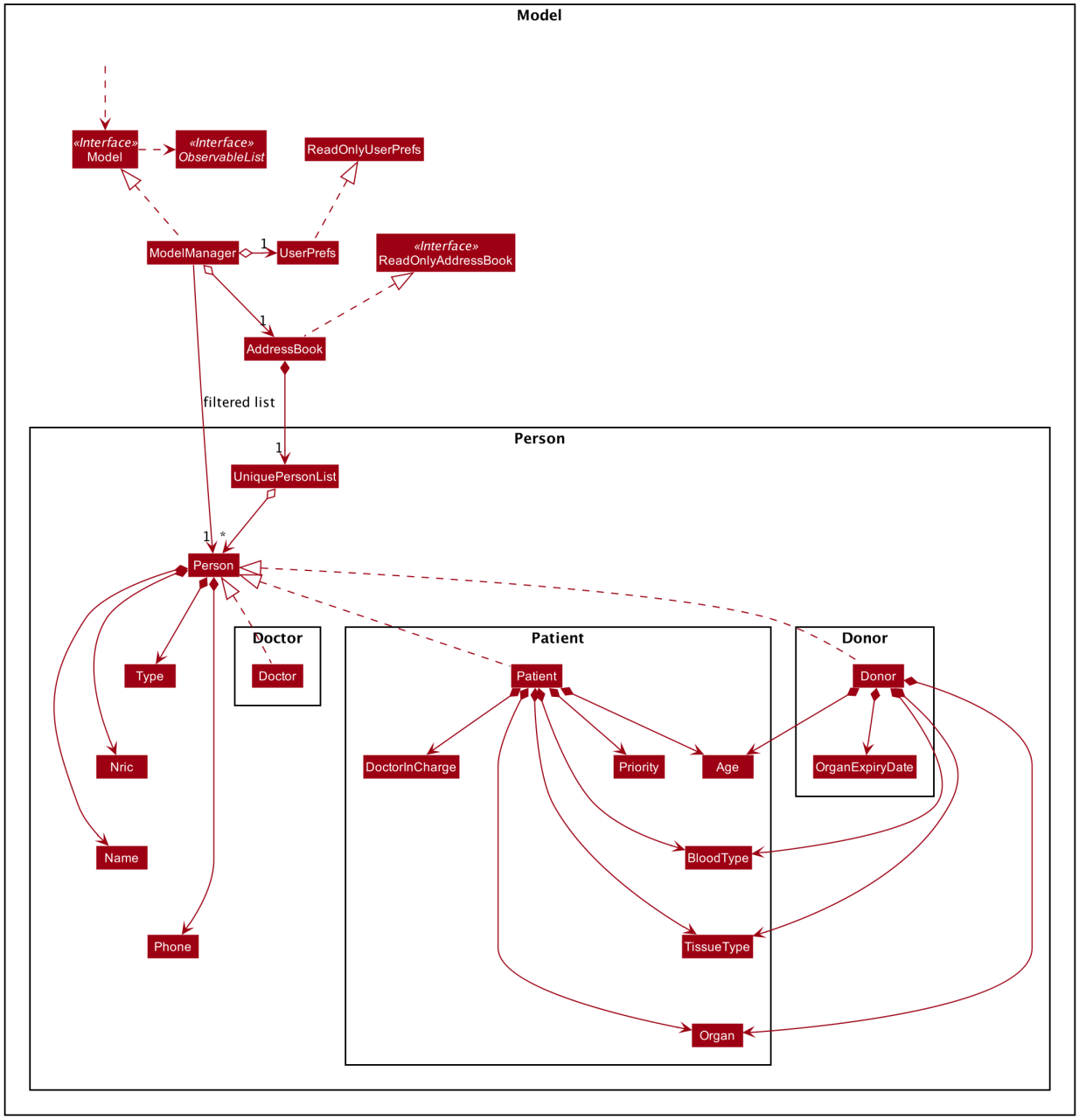

Figure 3. Structure of the Model Component

{End of the excerpt from Developer Guide}