Chua Sim Nee - Project Portfolio

PROJECT: ORGANice

1. Introduction

This document aims to show an overview of my involvements in the project, ORGANice. My role in this project was to implement the task list for the users to track the administrative tasks that needs to be done.

1.1. Project Description

ORGANice is developed by a team of five students in National University of Singapore (NUS). All of us are year 2 Computer Science students,taking the Software Engineering module, CS2103T.

1.2. Project Scope

ORGANice was made for our CS2103T module's team project (TP). Over the course of 6 weeks, we were tasked with either enhancing or morphing a Command Line Interface (CLI) desktop addressbook application. We chose to morph it into an organ transplant manager, ORGANice. It has a Graphical User Interface (GUI) which is created with JavaFX.

The target audience of ORGANice are the hospital administrative staffs as we aim to help ease their workload when trying to match all their patients with all the donors' organs they have.

1.3. Project Summary

ORGANice is an organ transplant manager. Currently, hospital administrative staffs have many patients and organ donors information in their records. Whenever they have a new patient who needed an organ, they often have to go through many manual searches and matching to find a compatible donor. We aim to reduce such menial work using ORGANice and speed up this process.
ORGANice is able to:

- `add` and `edit` patients, donors and doctors information.
- `list` the entire database information.
- `match` the patients and show potential compatible donor's information.
- `sort` the result after using `match` command.
- `find` a specific patient, donor or doctor in ORGANice.
- `processing` a patient and a donor who are compatible to generate a list of administrative tasks to do for the patient and donor.
- `done` processing a patient and donor pair and determine if they can be removed from the matching pool.

This is what ORGANice looks like:

Figure 1. The graphical user interface for ORGANice.

1.4. Key formatting

This section explains the formatting used in this document.

`text`	A grey highlight (called a mark-up) indicates that this is a command that can be inputted into the command line and executed by the application.
`Patient`	Blue text with grey highlight indicates a component, class or object in the architecture of the application.

2. Summary of Contributions

This section lists the codes and documentations I contributed to ORGANice.

I was responsible for implementing the `processing`, `processingMarkDone` and `done` commands. I also ensured the code quality is up to standard.

2.1. Main Feature : `Processing`

This section will explain the command, `processing` which I implemented for ORGANice.

- What it does:
  It allow users to view a whole list of administrative tasks which needs to be done to arrange for cross-matching test between a patient and a donor.
- Justification:

Hospital staffs often need to do a lot of paper work and administrative tasks. There is now an increasing number of patients admitting to hospitals and if they need an organ, there will be a standard operating procedure which needs to be followed With ORGANice, staff will not need to manually keep track of which tasks are not done yet for every patients.

- Highlights:
  A default task list will be generated when the user use the command with valid NRICs of donor and patient. The task list generated belong to the donor and patient only. This will not lead to any confusions as it is impossible to have a donor matching with more than one patient or a patient matching with more than one donor.

(Pull requests [#127], [#176], [#267])

2.2. Main Feature : `ProcessingMarkDone`

This section will explain the command, `processingMarkDone` which I implemented for ORGANice.

- What it does:
  It allows users to mark a task on the list of administrative tasks as done.
- Justification:
  This command will allow the hospital staffs to have a better overview of what are the existing tasks which need to be done. This will reduce the chance of hospital staffs doing a single task for more than one time or miss out a task by accident.
- Highlights:
  This feature cannot be used if the patient and donor have not been processed before. This will ensure that the correct list is edited instead of a wrong one. The list will also be generated with either a cross, □, or a tick, ☑ beside each task. It will be very intuitive for the hospital staffs to determine which tasks are done already.

(Pull requests [#127], [#176], [#267])

2.3. Main Feature : `Done`

This section will explain the command, `done` which I implemented for ORGANice.

- What it does:
  It allows users to determine if a patient and a donor is all done.
  Meaning, the patient and donor have gone through a test and we know know if the patient rejected the organ from the donor or not.
  If the patient is not compatible with the donor, then both the patient and donor will be saved back into ORGANice and available for future organ matching detection.
  However, they will not be able to match each other anymore.
  If the patient is compatible with the donor, then both of the patient and donor will be removed from ORGANice.
- Justification:
  Our algorithm is only able to find the compatibility of the blood type and tissue type of the patient and donor pair. However, a cross-matching test is needed to determine if the patient will

reject the donor's organ. Hence, there is a chance that the result of the cross-matching test is negative and the patient and donor will need to find another match.

- Highlights:
  This feature will save a history of all rejected patients a donor have so as to not allow the donor to be matched again with them. This will ensure that the same patient and donor do not go through another round of cross-matching just to return the same negative result.

(Pull requests [#155], [#230], [#261])

2.4. Code Contributed

Please refer to this link to view the code I wrote: [RepoSense]

2.5. Other Contributions

- **Enhancements**
  - Enhance the `edit` feature to allow users to edit a person's by their NRIC instead of their index number.
    (Pull request [#119])
  - Enhance the `edit` feature to allow users to edit other attributes of a person.
    (Pull request [#177])
  - Added `BloodType` and `TissueType` attributes to the `Donor` and `Patient` classes.
    (Pull requests [#82], [#89], [#96])
  - Added `Status` attribute to the `Donor` and `Patient` classes for the `processing` feature.
    (Pull request [#139])
- **Documentations**
  - Updated ReadMe from the given addressbook's ReadMe to be for ORGANice.
    (Pull request [#34])
  - Updated User Guide
    (Pull request [#128], [#135], [#269])
- **Community**
  - Reviewed Pull Requests (with non-trivial review comments): [#85], [#88], [#97], [#115], [#130], [#138], [#251].
  - Reported bugs and offered suggestions for other teams in the class. (Examples: 1, 2)

3. Contributions to User Guide

Given below are sections I contributed to the User Guide. They showcase my ability to write documentation targeting end-users.

{Start of First Extract from User Guide}

3.1. Processing potential matches: `processing`

After finding a list of matches, the hospital will be able to schedule a cross-matching test between a specific donor and patient. To schedule a cross-matching test, there will be a lot of other administrative tasks to be done. Hence, this command will be able to allow you to have an overview of what kind of tasks you need to do for that particular pair of donor and patient before the cross-matching.

3.1.1. `Status` of the donor and patient pair

You can also view the current state of every donor and patient in ORGANice by looking at their `Status`.

The `Status` of patient and donor can be either `Not Processing`, `Processing` or `Done`.

- `Not Processing` : The current donor or patient is not yet matched with anyone and is not going for any cross-matching test yet.
- `Processing` : The current donor or patient has already found a match and is in the midst of preparing for a cross-matching test.
- `Done` : The current donor or patient have completed the cross-matching and the result of the cross-matching shows a positive result. Hence, this donor or patient does not need to be in ORGANice anymore.

All the patients and donors should be `Not Processing` initially.
When the command is used for the first time for the patient and donor pair, the `Status` of the patient and donor will change from `Not Processing` to `Processing`.
If the patient and donor is being processed, their `Status` will remain as `Processing`.

3.1.2. Task list for the donor and patient

This command will show you the task list for the `Processing` donor and patient for you to be able to keep track of the necessary standard of procedure easily.

Currently you can:

- Generate a default task list automatically for newly processed patient and donor if they are `Not Processing` initially.
- View the list which belongs to the patient and donor pair who are already `Processing`.
- Mark a task as done in the task list generated from the patient and donor using another command `processingMarkDone ic/PATIENT NRIC ic/DONOR NRIC TASK NUMBER` which will be explained further later.

3.1.3. `Processing` the donor and patient

When you use this command, the following will occur:

- A default task list will be generated to show the necessary tasks the hospital needs to do for the respective donor and patient.

- The task list will be unique to the specific donor and patient.
- The task list can be updated using another command,`processingMarkDone ic/PATIENT NRIC ic/DONOR NRIC TASK NUMBER`
- The task list will be saved automatically.
- A donor can only be in `Processing` state with only one patient and vice versa. This means, if a donor and a patient is already in `Processing` state, they will not be able to be processed with another donor or patient.
- A screenshot of the list generated can be shown below:

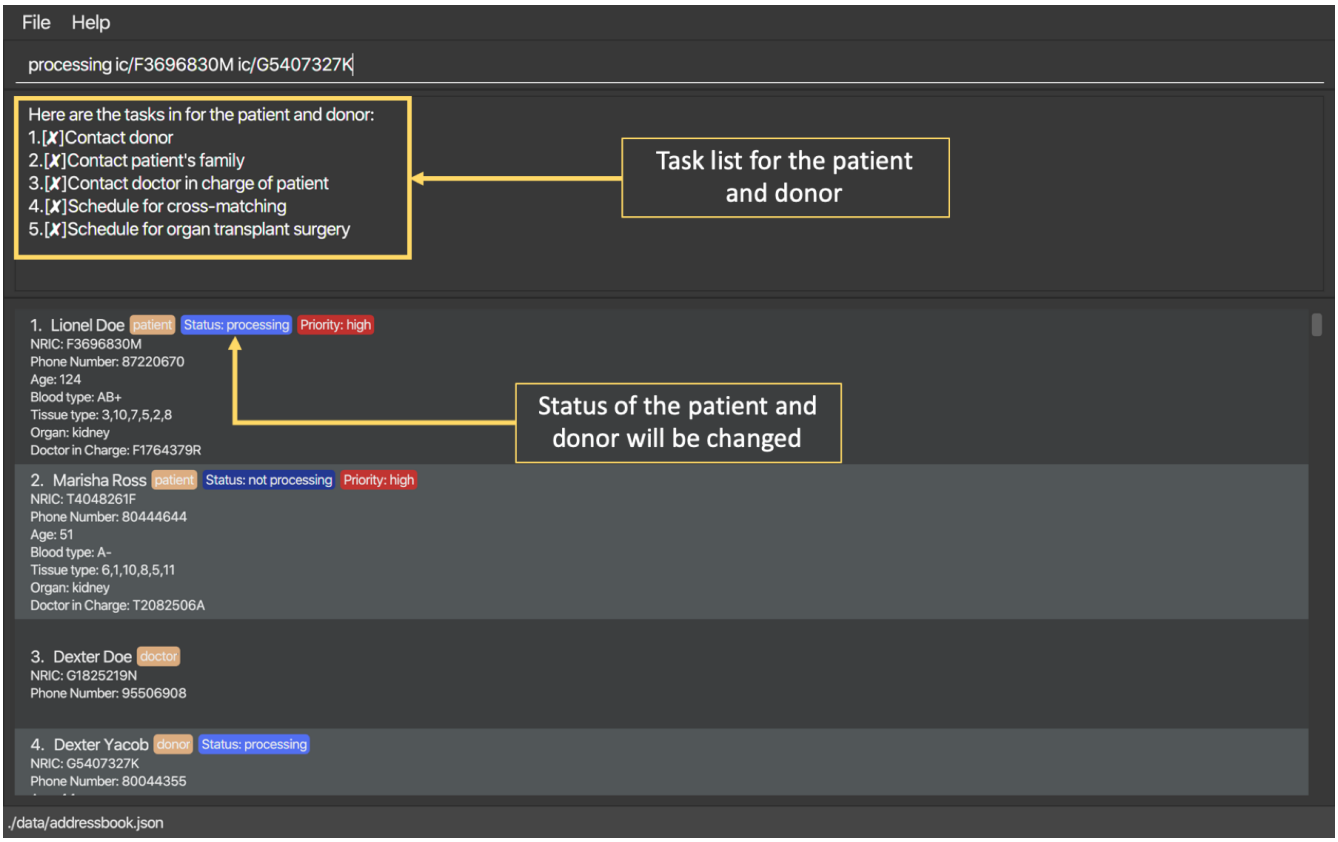

Figure 2. Screenshot of the screen after you use the command on a valid donor and patient

Format: `processing ic/PATIENT NRIC ic/DONOR NRIC`
The order of the NRICs does not matter, as long as the patient and the donor must be valid.

Example: `processing ic/S6764235G ic/S5234567D`

{End of First Extract from User Guide}

{Start of Second Extract from User Guide}

3.2. Checking the task list : `processingMarkDone`

As introduced above, this command serves to help you mark a task on the task list as done. This will allow you to be able to know what other administrative things you need to do for the specific donor and patient before the cross-matching test. Beside each task on the task list, there will be either a cross, □, or a tick, ☑. A tick would mean that the task is completed and a cross will mean that the task has not yet been completed.

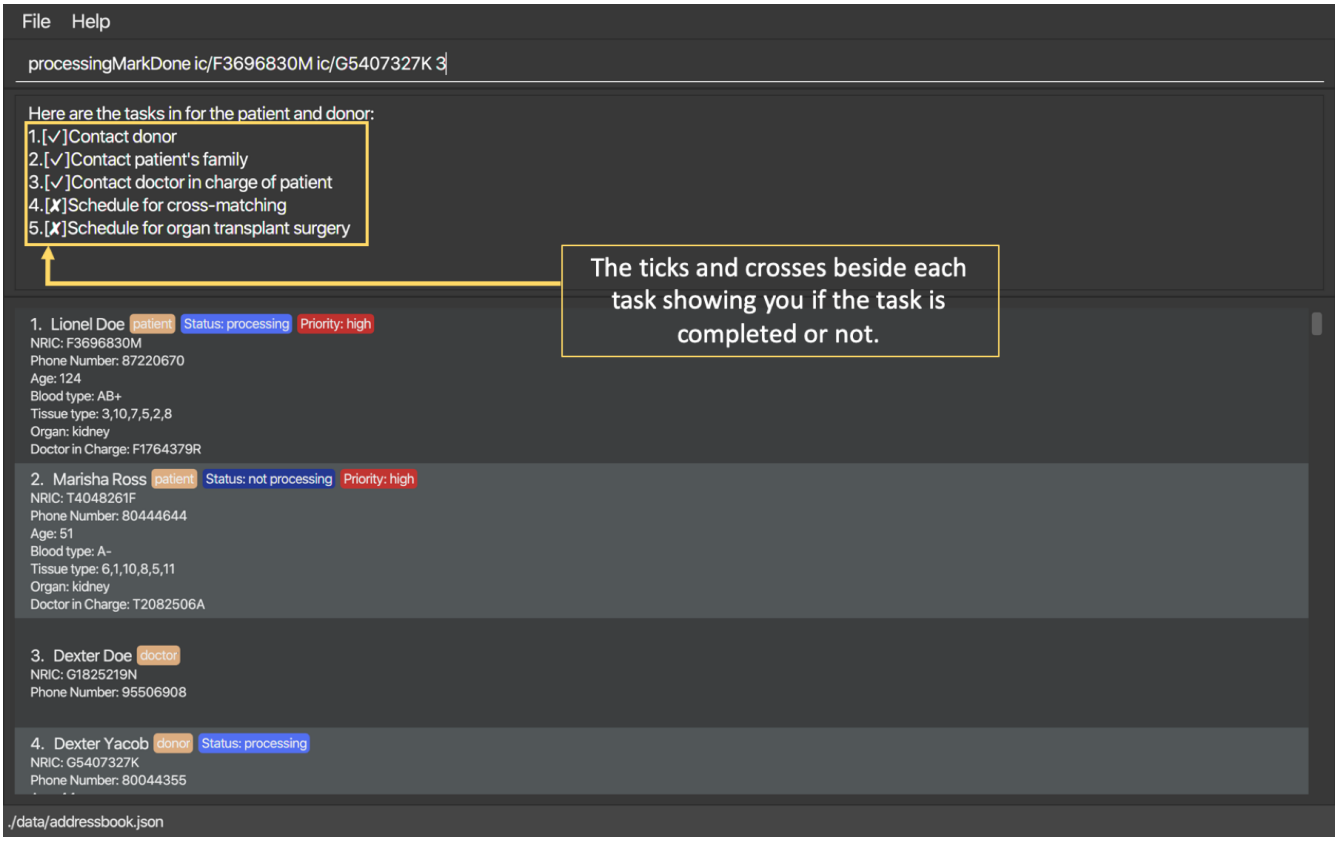

Figure 3. Screenshot of the screen after you use the command on a valid donor and patient and a valid task number

Format: `processingMarkDone ic/PATIENT NRIC ic/DONOR NRIC TASK NUMBER`
The order of the NRICs does not matter, as long as the task number, patient and donor must be valid. The donor and patient need to be in `Processing` state to be considered valid.

Example: `processingMarkDone ic/S6764235G ic/S5234567D 1`

{End of Second Extract from User Guide}

{Start of Third Extract from User Guide}

3.3. Mark as done : done

After the hospital has the results of the cross matching test, they can update the results in ORGANice. Cross matching tests have two possible results: pass or fail.

Format: `done ic/PATIENT NRIC ic/DONOR NRIC res/[pass/fail]`

The order of the NRICs does not matter, as long as the patient and the donor must both be valid.

The following section outlines what happens based on the two results:

If the result of the cross matching is a 'pass':

1. Run the command based on the given format
2. It is up to you to schedule an organ transplant surgery between the patient and donor.
3. Mark the patient and donor as `Done`, based on the command format.

4. The patient and donor statuses will be marked as `Done`.
5. After running the command, the system removes the patient and donor from ORGANice.

If the result of the cross matching is a 'fail':

1. Run the command based on the given format
2. The donor and patient will be added back to the matching pool for match detection with other patients and donors.
3. The patient and donor statuses will be marked as `Not Processing`.
4. The donor and patient pair will not be considered a potential match anymore in future match detections.

Example:

- `done ic/S6764235G ic/S5234567D res/pass`
- `done ic/S5234567D ic/S6764235G res/fail`

{End of Third Extract from User Guide}

4. Contributions to 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.

{Start of First Extract from Developer Guide}

4.1. Processing feature

This section describes how the processing feature is implemented.

Processing is facilitated by the `ProcessingCommand` class, which extends the `Command` class.

Processing is done by accessing `Patient` and `Donor` in `ModelManager`, then creating a `ProcessingList` which acts like ArrayList to store the tasks which needs to be done.

It implements the `ProcessingCommand#isValidDonorPatientPair(Nric firstNric, Nric secondNric, Model model)` boolean operation.
This method will first determine which Nric belongs to a patient and which Nric belongs to a donor since the input allows user to give two Nric in no particular order.
It will also get the `Donor` and `Patient` objects with the NRIC given.
Then, it will check if the donor have a `ProcessingList` generated before.
If the donor have an empty `ProcessingList`, it means that the donor is newly processed.
Hence, it will mean that the donor may be able to pair with the given patient.
If the donor already have a `ProcessingList`, it means that the donor is already being processed with a patient.
Then, this method will check if the patient given by the user is the same patient being processed

together with this donor.
This is done by checking the `Patient` attribute that is tagged to the `ProcessingList` of the donor.
Finally, the method will check if the patient and donor given by the user is valid by checking if they are able to be matched or not and if they exist in ORGANice.
A code snippet of the method is shown below: This code aims to get the `Donor` and `Patient` from the given Nrics given by the user.
This method is necessary since users able to give the Nrics input in no particular order.

```
if (model.hasDonor(firstNric)) {
    donorNric = firstNric;
    donor = model.getDonor(donorNric);

    patientNric = secondNric;
    patient = model.getPatient(patientNric);
} else {
    patientNric = firstNric;
    patient = model.getPatient(patientNric);

    donorNric = secondNric;
    donor = model.getDonor(donorNric);
}
```

`Status` class is also created and it is one of the attributes of `Patient` and `Donor`. However, this cannot be edited or added by users.
Processing command make changes to the `Status` of the patient and donor.

- When a new `Patient` or `Donor` object is created, the `Status` of the new object will be set to `not processing`.
- When the `Processing` command is being executed, `Status` of the `Patient` and the `Donor` will be changed to `processing`, provided that the `Patient` and `Donor` are valid.
- When the `Done` command is being executed, `Status` of the `Patient` and the `Donor` will be changed to `done`, provided that the `Patient` and `Donor` are valid.

If the donor and patient have never been processed before, a default `ProcessingList` will be generated and it will belong to the patient and donor pair uniquely.

If the user made any changes to the `ProcessingList`, the list will be saved to the donor as one of its attribute which takes in a parameter of the patient's NRIC.

When the user execute the processing feature, the method, `ProcessingCommand#isValidDonorPatientPair(Nric firstNric, Nric secondNric, Model model)` will be called to determine if the user give a valid patient and donor input.
It will then check if there is a need to generate a new default task list or the donor already have a list tagged to itself.
Then, the `Status` of the patient and donor will be set to `processing`. It will then display the list out.

Following is a class diagram to show the processing feature

TODO diagram

4.1.1. Design considerations

This section will explain the reason for having some aspects.

Aspect: Method of storing the `ProcessingList`

- **Alternative 1 (current choice):** Use a new class, `TaskList` that works like ArrayList.
  - Pros: Able to implement more specific methods for the ProcessingList.
  - Cons: More methods and test cases needed which are harder to implement.
- **Alternative 2:** Use ArrayList to store the tasks.
  - Pros: Easy to implement as the default methods are already created.
  - Cons: Future extension on this will be difficult as the `ProcessingList` is unique to this feature.

Aspect: Using specific prefixes for Nric inputs

- **Alternative 1 (current choice):** Nrics of patient and donor can be inputted in no particular order.
  - Pros: More user friendly as users do not need to make sure the Nric must be in correct order. (For example, patient's Nric must come before donor's Nric)
  - Cons: It is harder to implement and more exceptions need to be taken care of.
- **Alternative 2:** Using specific prefixes for Nric input to differentiate the Nrics of donor and patient.
  - Pros: It is easier to implement as `icP/PATIENT NRIC` and `icD/DONOR NRIC` will be easier to parse to the Command method.
  - Cons: It is less user friendly as users need to remember more prefixes for this entire application.

{End of First Extract from Developer Guide}

{Start of Second Extract from Developer Guide}

4.2. ProcessingMarkDone feature

This section describes how the processingMarkDone feature is implemented.

ProcessingMarkDone is facilitated by the `ProcessingMarkDoneCommand` class, which extends the `Command` class.

ProcessingMarkDone is done by accessing `Patient` and `Donor` in `ModelManager`, then, it will get the `ProcessingList` from the `Donor` and then marking a task as done.

It implements the `ProcessingMarkDoneCommand#isValidDonorPatientPair(Nric firstNric, Nric secondNric, Model model)` boolean operation.

This method works similar to `ProcessingCommand#isValidDonorPatientPair(Nric firstNric, Nric secondNric, Model model)` method but with a slight difference.
This method just need to check the `Status` of the `Donor` and `Patient` object and make sure it is `processing`.
However, this method still need to determine which Nric belongs to a patient and which Nric belongs to a donor since the input allows user to give two Nric in no particular order.
A code snippet of the method is shown below: This part of the method is different from the `ProcessingCommand#isValidDonorPatientPair(Nric firstNric, Nric secondNric, Model model)` method.

```
if (model.hasPatient(patientNric) && model.hasDonor(donorNric)
        && patient.getStatus().isProcessing() && donor.getStatus().isProcessing()
        && match(donor, patient)) {
    return true;
} else {
    return false;
}
```

When the user execute the processingMarkDone feature, the method, `ProcessingMarkDoneCommand#isValidDonorPatientPair(Nric firstNric, Nric secondNric, Model model)` will be called to determine if the user give a valid patient and donor input.
It will also check if the task number given is valid.
Then, it will call `Donor#markTaskAsDone(int taskNumber)` to mark the task on the `ProcessingList` of that donor. It will then display the list out.

Following is an object diagram to show the processingMarkDone feature

TODO diagram

4.2.1. Design considerations

This section will explain the reason for having some aspects.

Aspect:

- **Alternative 1 (current choice):**
  - Pros:
  - Cons:
- **Alternative 2:**
  - Pros:
  - Cons:

Aspect: Using specific prefixes for Nric inputs

- **Alternative 1 (current choice):** Nrics of patient and donor can be inputted in no particular order.

    - Pros: More user friendly as users do not need to make sure the Nric must be in correct order. (For example, patient's Nric must come before donor's Nric)
    - Cons: It is harder to implement and more exceptions need to be taken care of.

- **Alternative 2:** Using specific prefixes for Nric input to differentiate the Nrics of donor and patient.
    - Pros: It is easier to implement as `icP/PATIENT NRIC` and `icD/DONOR NRIC` will be easier to parse to the Command method.
    - Cons: It is less user friendly as users need to remember more prefixes for this entire application.

{End of Second Extract from Developer Guide}

{Start of Third Extract from Developer Guide}

4.3. Done feature

This section describes how the done feature is implemented.

{End of Third Extract from Developer Guide}