Jack Wong - Project Portfolio

PROJECT: ClinicIO

1. Overview

This portfolio serves the purpose of documenting my contributions to the project - ClinicIO. ClinicIO is a command-oriented desktop software for clinic use with the aim of making the lives of receptionists, doctors and patients easier. Command line is the primary mode of input due to the assessment constraints. Nonetheless, the visual output is graphical.

Our team consists of three primary developers, an assistant team lead and a team lead. As an assistant team lead, I assist the team lead in the overall project coordination. Furthermore, I am responsible for the logic component of the software.

In this project, I have designed and developed some crucial features of the software such as *assigning a patient into the queue*, *removing a particular patient from the queue* and *showing all the patients in the queue*.

I worked mainly on *designing the queue* and the *algorithm of the queueing system*. This system aims to achieve fairness, efficiency and flexibility.

Besides queue, I have *designed the model for a patient* so that the data associated with patients can be stored efficiently in our database.

Lastly, I am the main tester of ClinicIO. I *ensure all the components are integrated properly and thoroughly tested* before the software is deployed.

2. Summary of contributions

> *This section outlines the major and minor enhancements I made to ClinicIO. In addition, other relevant contributions made to the project and my team as a whole are also included.*

Major enhancement 1 - added the ability to assign a patient into the queue:

- **What it does:** It allows the user to assign a patient into the queue based on the index of the patient in the patient display list.
- **Justification:** This feature improves the product significantly because a user - typically a receptionist - needs to routinely place patients into the queue.
- **Code snippet for enqueuing a patient:**
  ```
  model.enqueue((Patient) patientToEnqueue);
  ```

Major enhancement 2 - added the ability to remove a patient from the queue:

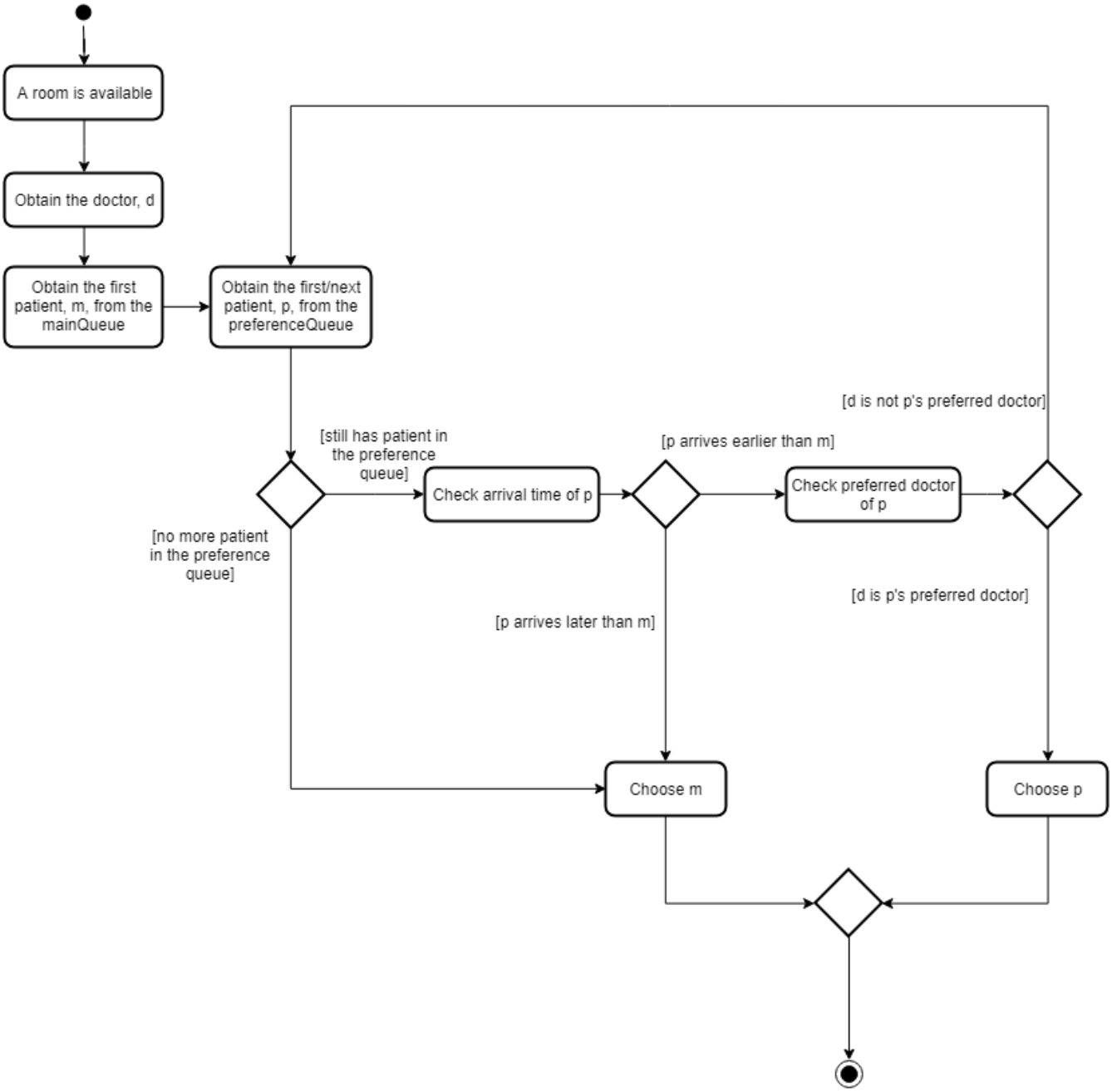
- **What it does:** It allows the user to remove a patient from the queue based on the patient's

index.

- **Justification:** This feature improves the product significantly because the user can carry out his/her daily routine more efficiently without affecting existing patients in the queue.
- **Code snippet for dequeuing a patient:**
  `model.dequeue((Patient) patientToDequeue);`

Major enhancement 3 - designed the preference-based queuing algorithm:

- **What it does:** It assigns a patient's queuing priority based on his/her preference for any doctor.
- **Justification:** This feature serves the need of many patients who opt for a specific doctor for consultation. It ensures fairness by prioritising patients who arrive earlier.
- **Activity diagram for selecting a patient for consultation:**

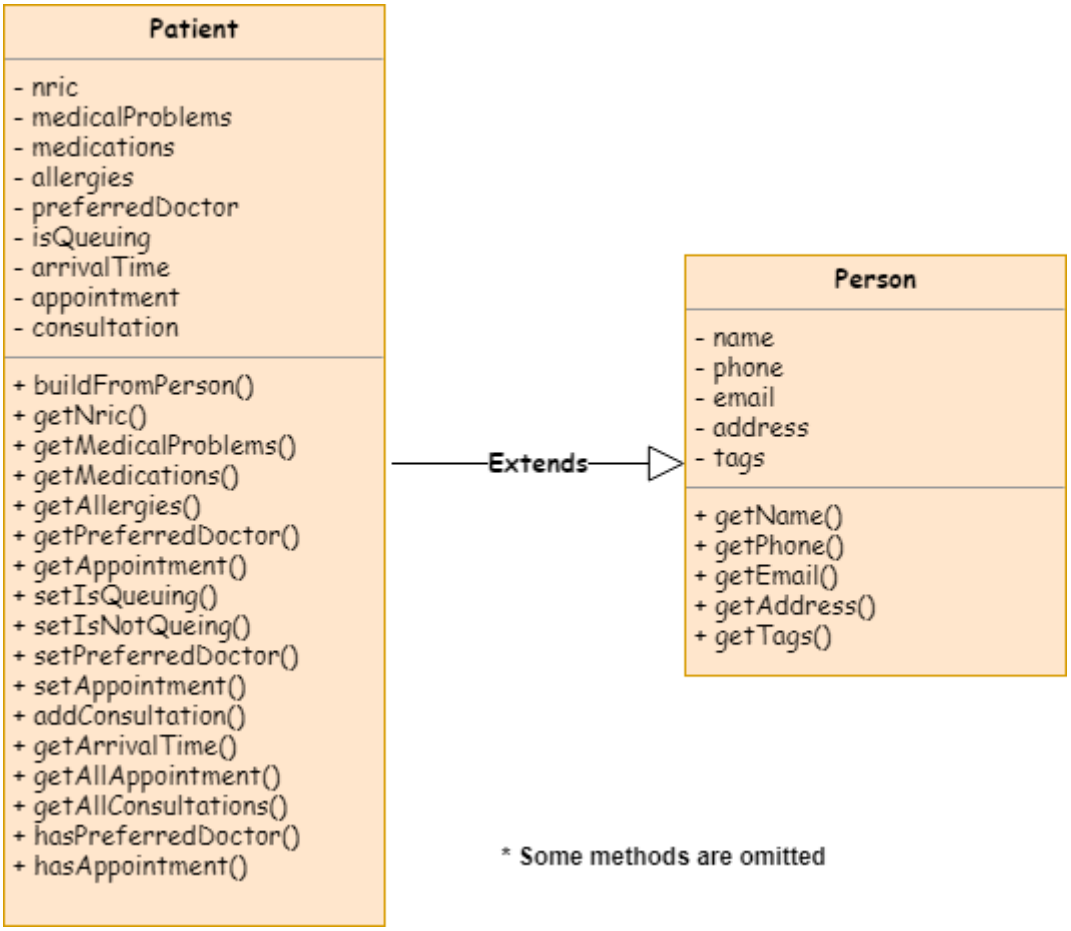


Major enhancement 4 - added the ability to show all patients currently in the queue:

- **What it does:** It allows the user - typically a receptionist - to display all patients currently in the queue.
- **Justification:** This feature is useful because patients sometimes would enquire the time it takes for their turn to come. The receptionist can use this simple feature to display all patients in the queue and relay necessary information (how many patients are in front of the asking patient) to the patients.
- **Code snippet for showing the queue:**
  `model.updateFilteredPersonList(model.PREDICATE_SHOW_ALL_PATIENTS_IN_QUEUE);`

Major enhancement 5 - designed the Patient model:

- **What it does:** It represents the patient in digital format.
- **Justification:** This feature is crucial because it allows the receptionist to store all relevant information associated with a patient into our database. This allows the receptionist to easily query a patient given some information about the patient.
- **Class diagram for a patient:**



Codes contributed:

This subsection provides the link to the relevant source codes I am responsible for writing.

- Url link to **all codes** I have written (RepoSense dashboard): https://nus-cs2103-ay1819s1.github.io/cs2103-dashboard/#=undefined&search=iamjackslayer&sort=displayName&since=2018-09-30&until=2018-11-11&timeframe=day&reverse=false&repoSort=true

Other contributions:

This subsection lists some aspects of the entire project I am responsible for contributing. These contributions reflect my role as an assistant team lead for the project and also my technical responsibility as the main tester for the software. Furthermore, this subsection lists some of my contributions beyond our team.

- Project management:
  - Managed releases `v1.3`(Practical Exam Dry Run) on GitHub.
  - Managed ClinicIO Board - A digital equivalent of whiteboard that displays tasks and issues belonged to team members. I also created a column which contains articles on miscellaneous topics such as how to collaborate effectively on Github.
    Url link to ClinicIO Board: https://github.com/CS2103-AY1819S1-W14-1/main/projects/1
- Quality assurance:
  - Wrote additional tests for existing features.
  - Reviewed all test cases written by team members.
    - Example review: https://github.com/CS2103-AY1819S1-W14-1/main/pull/106
- Documentation:
  - Updated the class diagram for Model component in the Developer Guide:
    - Url link: https://github.com/CS2103-AY1819S1-W14-1/main/blob/master/docs/DeveloperGuide.adoc#24-model-component
- Community:
  - Reviewed PRs (Pull Requests) of team members:
    - Url link: https://github.com/CS2103-AY1819S1-W14-1/main/pull/105
    - Url link: https://github.com/CS2103-AY1819S1-W14-1/main/pull/106
  - Contributed to a forum discussion:
    - Url link: https://github.com/nus-cs2103-AY1819S1/forum/issues/127
  - Reported bugs and suggestions for another team in the class:
    - Team T16-4: https://github.com/CS2103-AY1819S1-T16-4/main/issues/171
    - Team T16-4: https://github.com/CS2103-AY1819S1-T16-4/main/issues/181
    - Team T16-4: https://github.com/CS2103-AY1819S1-T16-4/main/issues/165

3. Contributions to the User Guide

This section delineates my contributions to the User Guide. They showcase my ability to write documentation targeting the end users.

Queue Commands

Assign a patient into the queue : `enqueuepatient`

Assigns the patient based on the index number used in the displayed patient list.
Format: `enqueuepatient INDEX`

- Assigns the patient into the queue (first in first out manner)
- The index refers to the index number shown in the displayed patient list.
- The index **must be a positive integer** `1, 2, 3, …`

The command is typically used in combination with other commands.
Examples:

- `list`
  `enqueuepatient 7`
  Selects the 7th patient in the displayed list of patients resulting from the `list` command and assigns the patient into the queue.
- `find Logan`
  `enqueuepatient 1`
  Assigns the 1st patient in the displayed list of patients whose names contain **Logan** (case insensitive) resulting from the `find Logan` command and assigns the patient into the queue.

Show all patients in the queue : `showqueue`

Lists all patients in the queue. Format: `showqueue`

Example:

- `showqueue`

Removing a patient from the queue: `dequeuepatient INDEX`

Pulls out the patient based on the index number used in the displayed patient list.

Examples:

- `dequeuepatient 9`
  Removes the 9th patient in the ClinicIO record from the queue.

The command can also be used in combination with other commands such as `list` and `find`.

Examples:

- `list`
  `dequeuepatient 3`
  Removes the 3rd patient in the displayed list of patients from the queue.
- `find Cassandra`
  `dequeue 1`
  Selects the 1st Cassandra as displayed in the list resulting from the `find Cassandra` command and removes her from the queue.

4. Contributions to the Developer Guide

This section delineates my contributions to the Developer Guide. They showcase my ability to write technical documentation and the technical depth of my contributions to the project.

Queue feature

This feature allows the user to perform operations related to the queue. In particular, it provides the functionality to assign a patient into the queue, remove a patient from the queue and show the list of patients in the queue.

Current Implementation

The Queue feature contains several operations to indirectly manipulate the two queues underlying the `patientQueue`. The two queues are `mainQueue` and `preferenceQueue`, both of which are hidden from the end user. To the end user, there exists only one queue. When a patient gets assigned to the queue without any preferred doctor, the patient will be inserted into the `mainQueue`. In the scenario where a patient has a preferred doctor, the patient will be inserted into the `preferenceQueue`.

When a room is available, the system will look for the first patient in the `preferenceQueue` whose preferred doctor is in the room. It will then compare this patient with the frontmost patient from the `mainQueue` on their arrival time. The patient who arrives earlier will get to consult the doctor.

Both queues are composed of `java#ArrayList`. This allows the system to handle the situation where a patient in the middle of the queue decides to leave the queue.

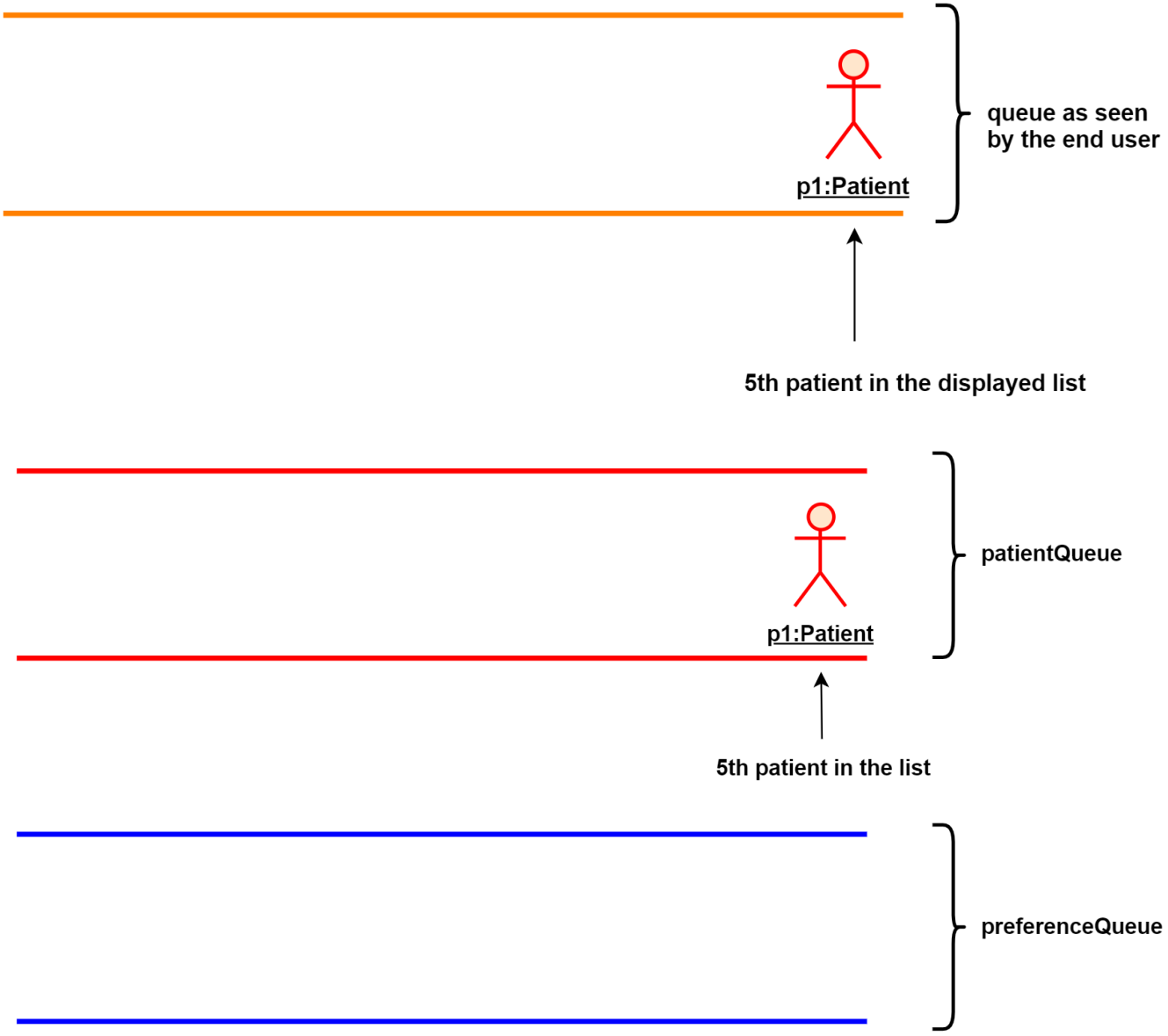
The implemented operations for Queue are:

- `enqueuepatient` Command - Assigns a patient into the queue.
- `dequeuepatient` Command - Removes a patient from anywhere in the queue.
- `showqueue` Command - Shows a list of all patients in the queue.

Given below is an example usage scenario and how the queue-related operations behave at each step.

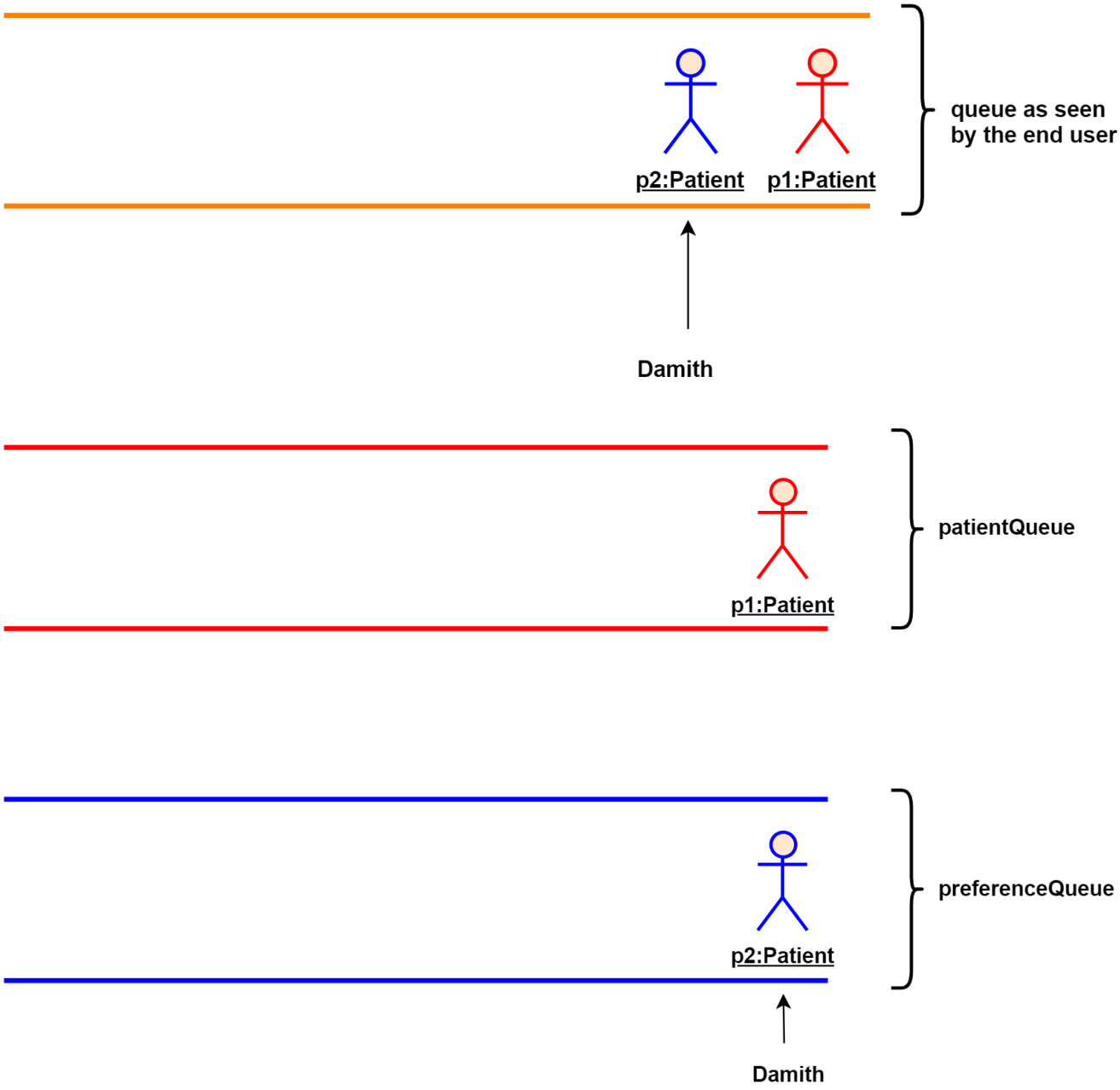
Step 1: The user lists all the patients using the `list` command. All patients in the ClinicIO record are be displayed.

Step 2: The user executes `enqueuepatient 5` command to assign the 5th patient in the list into the queue. This patient has no preferred doctor. Now the queue has 1 patient. Underlying the queue, the patient is assigned into `mainQueue`. The `preferenceQueue` remains empty:

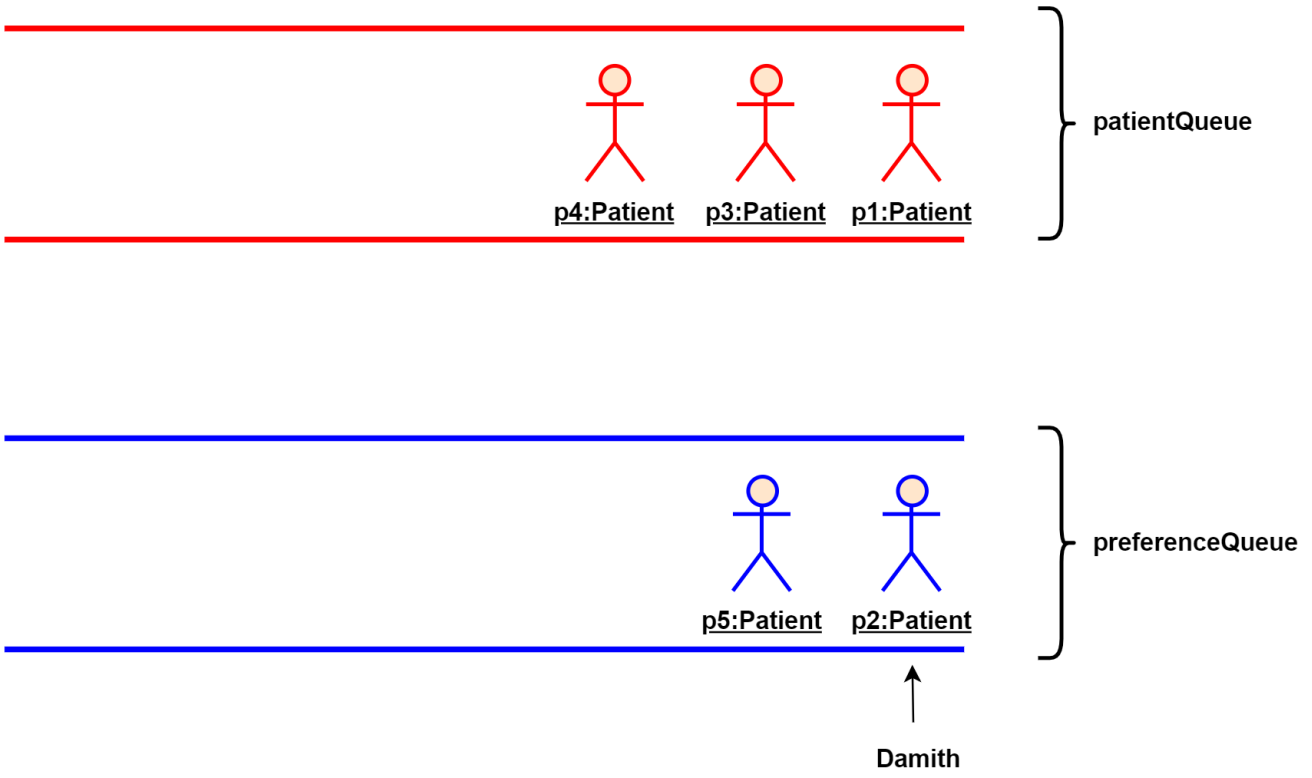


Step 3: The user finds all patients with names `Damith` using `find Damith` command. All patients with names `Damith` in the ClinicIO record will be displayed.

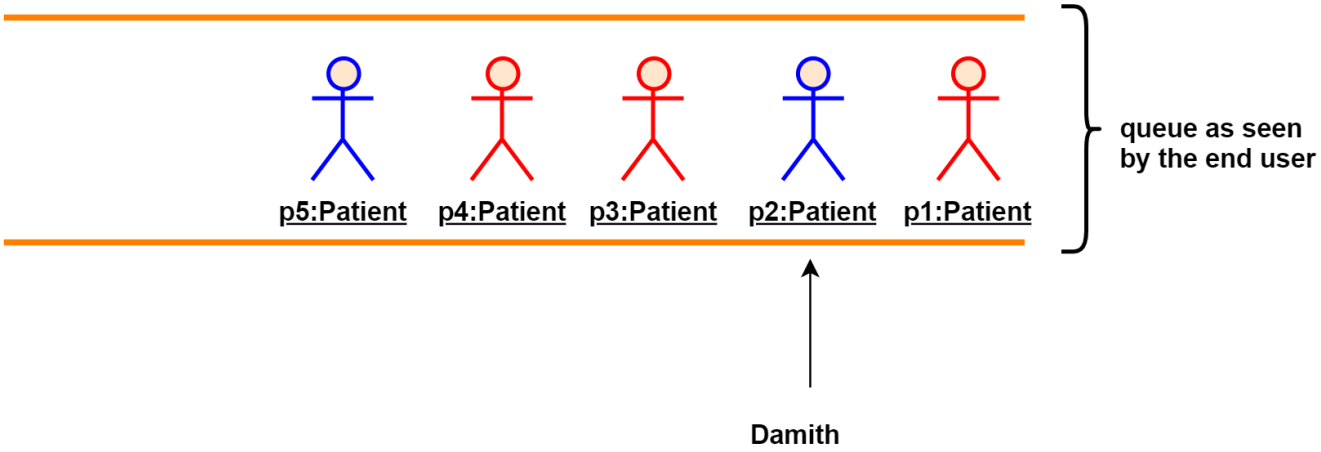
Step 4: The user executes `enqueuepatient 2` to assign the 2nd patient whose name contains `Damith` into the queue. This patient has a preferred doctor. Now the queue has 2 patients. Underlying the queue, this patient is assigned into `preferenceQueue`:



Step 5: The user keeps adding patients until there are 5 patients in the queue. The `mainQueue` and `preferenceQueue` look like this:



The end user only sees one queue:



Step 6: The user lists all patients in the queue by using `showqueue` command. All patients currently waiting in the queue are displayed.

Design Considerations

Aspect: How enqueuepatient command executes

- **Alternative 1 (current choice):** Inserts the patient into one of the two underlying queues.
  - Pros: Easier to implement. Slightly faster than the other alternative.
  - Con: May have performance issue in terms of memory usage.
- **Alternative 2:** Inserts the patient into only one queue.
  - Pro: Uses less memory as only one data structure is needed.
  - Con: Worse time complexity than the current implementation.

Aspect: How dequeuepatient command executes

- **Alternative 1 (current choice):** Looks for the queue (mainQueue or preferenceQueue) from which the patient is to be removed. Then searches for the patient and removes from the queue.
  - Pro: Slightly faster than the other alternative.
  - Con: May have performance issue in terms of memory usage.
- **Alternative 2:** Naively looks for the patient in the queue, assuming Alternative 2 of `enqueuepatient` command is used (only one underlying queue).
  - Pros: Uses less memory. Easy to implement as only one naive search is required.
  - Con: Worse time complexity than the current implementation.

Aspect: How showqueue command executes

- Shows a list of patients filtered according to `Patient#isQueuing()` which is basically the queuing status of the patient.

Aspect: Data structures to support the queue-related commands

- **Alternative 1 (current choice):** Uses two ArrayLists to store the patients.
  - Pros: Easier to implement. Provides more functionalities compared to Queues/LinkedList.
  - Con: Uses more memory than using only one ArrayList.
- **Alternative 2:** Uses one ArrayList to store the patients.
  - Pro: Uses less memory than Alternative 1.
  - Con: Worse time complexity when looking for a particular patient.
- **Alternative 3:** Uses Queue/LinkedList
  - Pro: Easy to implement. Very fast operation for popping the frontmost patient.
  - Con: Limited functionalities. Removing a patient from the middle of the data structure requires extra codes.

This is the end of my project portfolio