Emergency Room Simulation

Shuwan Huang, Northeastern University

Goal and plan

The goal of this program is to simulate the treatment of patients in a hospital emergency room. At the end of simulation, a conclusion about optimal number of examination rooms can be made.

The duration of patient generation and the number of examination rooms are provided by the user. Each patient has three properties with it—the arrival time, urgency level, and treatment time. Patients are treated according to their urgency level; patients with equal urgency level are treated in order of arrival. There are three queues in the simulator:

1) Arrival queue stores the patients waiting for treatment;
2) Room queue stores the available examination rooms;
3) Examination queue store the patients currently being treated.

When a patient arrives, it will firstly be added to the arrival queue. The simulator constantly checks if any patient finishes treatment at current local time. When a patient finishes examination, it will be removed from examination queue, and the corresponding room will be added to the room queue. If the arrival queue is not empty, the program will pick the patient with highest priority and start examination on that patient. To be specific, the patient will be moved to examination queue, and the available room with least busy time will be removed from room queue and assigned to that patient.

The simulation is complete when all patients are done with treatment.

Implementation

There are five major classes in this project—ERSimulator, Patient, ExaminationRoom, PatientGenerator, and PriorityQueue.

ERSimulator.java

The ERSimulator class is where we run the simulation described in the above section. PriorityQueue is the data structure used for the arrival queue, room queue and examination queue. At the beginning of simulation, user is asked to input the length of simulation (duration of patient generation) and the number of examination rooms. An ERSimulator object will be created with given number of rooms. The user will also input the max pause time between patient generation (milliseconds) and max treatment length of patient (minutes). By calling the method runSimulation(LocalDateTime start, Duration simulationTime, int maxPause, int maxTreatment) with start time the current local time, the simulator will start the simulation. During the given length of simulation, patients are generated by a PatientGenerator. (Notes:

the length of simulation mentioned here is not the actual overall simulation time.) The simulator will constantly check and update the three queues. At the end of simulation, the ERSimulator will analyze and print to console the average wait of patients for treatment and the usage information of examination rooms.

The *ERSimulatorConstants* interface provides the constants shared by the classes:

SIMULATION_MAX_TIME - the max duration that user can set for patient generation
MAX_PAUSE_ADD_PATIENT – the max number of milliseconds to pause between patient generation
MAX_TREATMENT_MINUTES – the max number of minutes for patient treatment
MIN_URGENCY_LEVEL – the min urgency level of patient
MAX_URGENCY_LEVEL – the max urgency level of patient

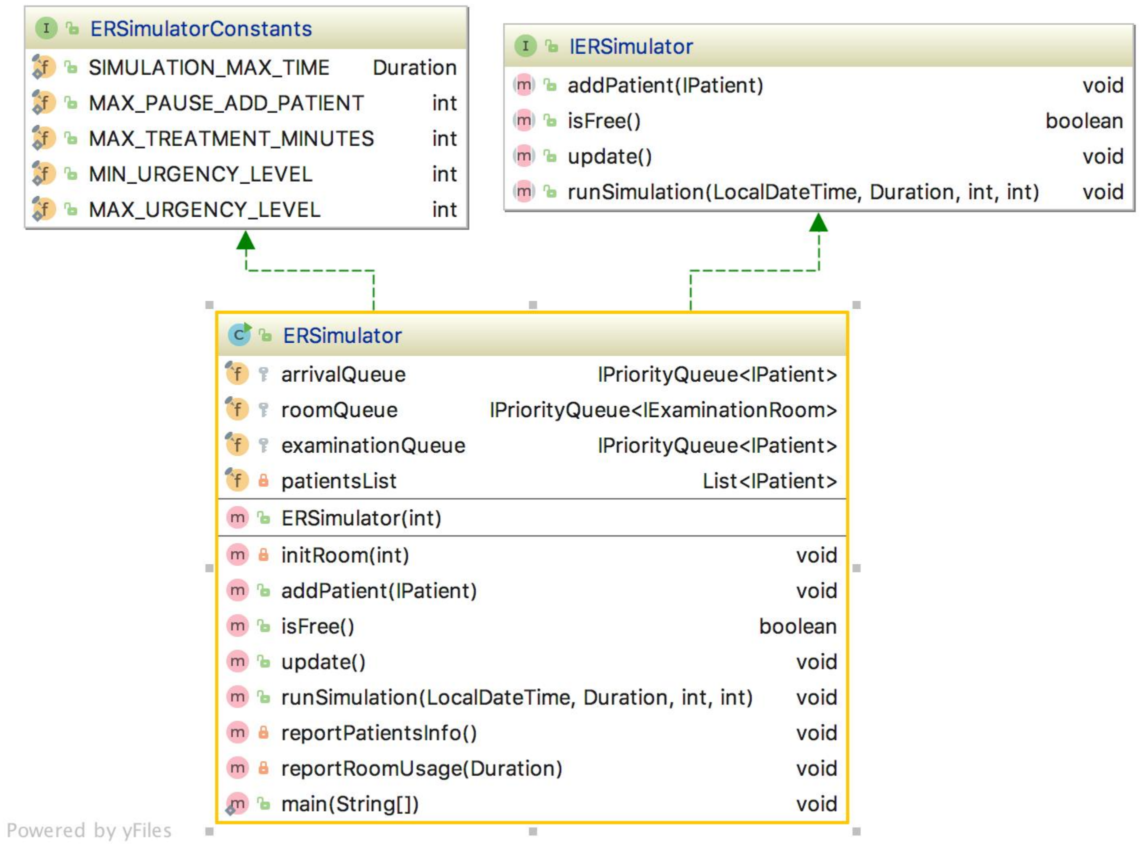

ERSimulatorConstants
SIMULATION_MAX_TIME Duration
MAX_PAUSE_ADD_PATIENT int
MAX_TREATMENT_MINUTES int
MIN_URGENCY_LEVEL int
MAX_URGENCY_LEVEL int
IERSimulator
addPatient(IPatient) void
isFree() boolean
update() void
runSimulation(LocalDateTime, Duration, int, int) void
ERSimulator
arrivalQueue IPriorityQueue<IPatient>
roomQueue IPriorityQueue<IExaminationRoom>
examinationQueue IPriorityQueue<IPatient>
patientsList List<IPatient>
ERSimulator(int)
initRoom(int) void
addPatient(IPatient) void
isFree() boolean
update() void
runSimulation(LocalDateTime, Duration, int, int) void
reportPatientsInfo() void
reportRoomUsage(Duration) void
main(String[]) void
Powered by yFiles

Fig 1. UML diagram of ERSimulator

<u>Patient.java</u>
The Patient class provides a constructor to create a new Patient object –

Patient(LocalDateTime arrivalTime, int urgencyLevel, Duration treatmentDuration, int id);

The first three parameters are described in the above section. The 4th parameter (int id) is used to refer to this patient when reporting patient info to console. The id is assigned to patient at its generation. We start examination on a patient by calling the method startExamination(IExaminationRoom room, LocalDateTime startTime), where an examination room and the start time are assigned to this patient. Once the start time is given, the departure time (= start time + treatment duration) and wait time (= start time – arrival time) can be calculated accordingly. There are getters in this class to obtain departure time, wait time of this patient, as well as the examination room where patient gets treated, and the basic information such as id, urgency level and duration of treatment. The natural order of patients is by urgency levels and arrival time (see Goal and Plan section). The Patient class also provides a Comparator (BY_DEPARTURE_TIME) for alternate ordering by departure time, which is used to order patients in examination queue.

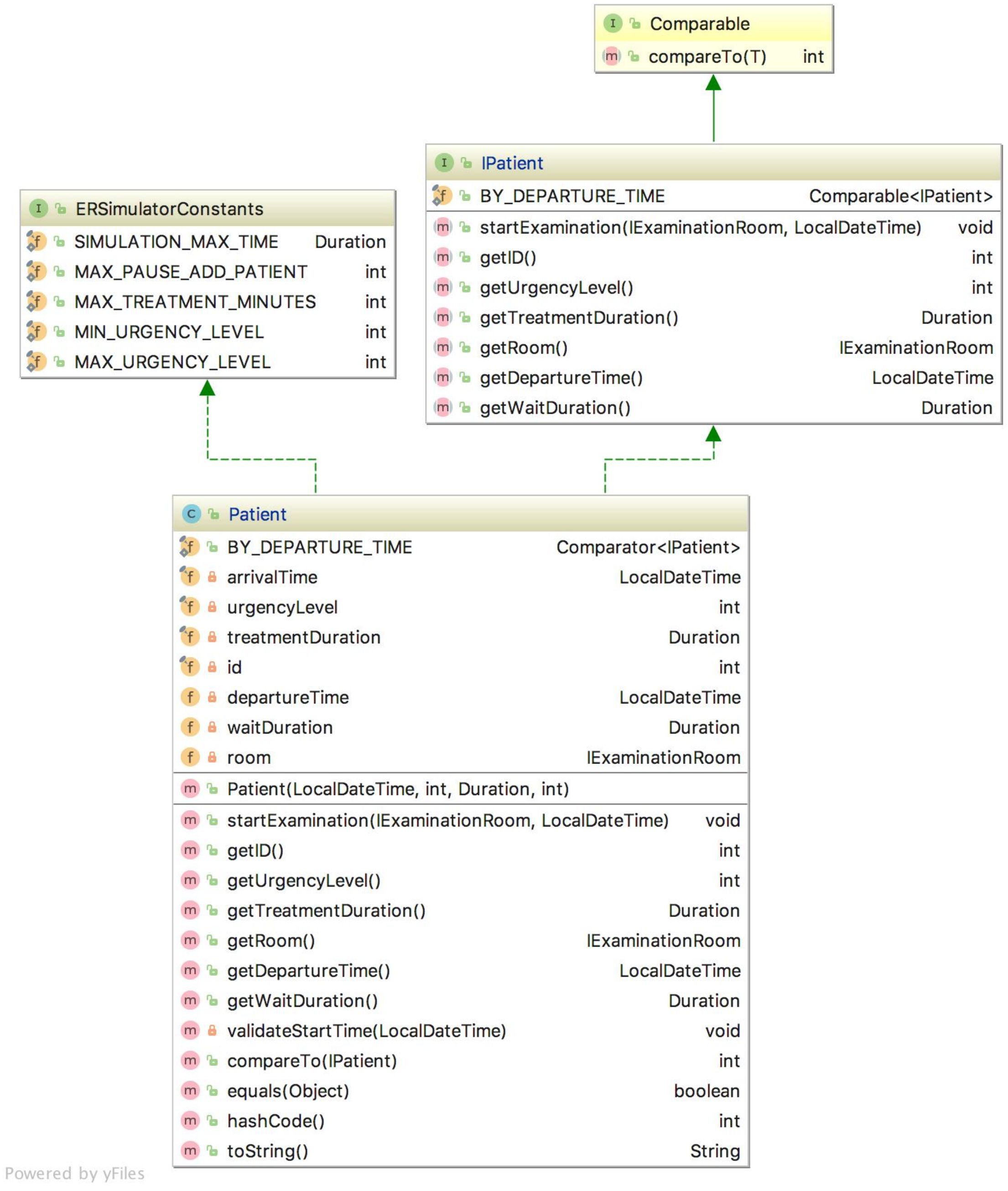

Comparable
compareTo(T) int
IPatient
BY_DEPARTURE_TIME Comparable<IPatient>
startExamination(IExaminationRoom, LocalDateTime) void
getID() int
getUrgencyLevel() int
getTreatmentDuration() Duration
getRoom() IExaminationRoom
getDepartureTime() LocalDateTime
getWaitDuration() Duration
ERSimulatorConstants
SIMULATION_MAX_TIME Duration
MAX_PAUSE_ADD_PATIENT int
MAX_TREATMENT_MINUTES int
MIN_URGENCY_LEVEL int
MAX_URGENCY_LEVEL int
Patient
BY_DEPARTURE_TIME Comparator<IPatient>
arrivalTime LocalDateTime
urgencyLevel int
treatmentDuration Duration
id int
departureTime LocalDateTime
waitDuration Duration
room IExaminationRoom
Patient(LocalDateTime, int, Duration, int)
startExamination(IExaminationRoom, LocalDateTime) void
getID() int
getUrgencyLevel() int
getTreatmentDuration() Duration
getRoom() IExaminationRoom
getDepartureTime() LocalDateTime
getWaitDuration() Duration
validateStartTime(LocalDateTime) void
compareTo(IPatient) int
equals(Object) boolean
hashCode() int
toString() String
Powered by yFiles

Fig 2. UML of Patient

PatientGenerator.java

During the simulation, patients are generated by a PatientGenerator. The main purpose of PatientGenerator is to create patients with random parameters by calling the next() method. The constructor of PatientGenerator has two parameters. One is the max pause (ms) between generation of patients, the other is the max duration of treatment (min). The urgency level will be set randomly from 1 to 10. The duration of treatment can be up to maxTreatmentMin of minutes. The pause between patient generation is also randomly set, which can be up to maxPauseMs of milliseconds. The arrival time is the current local time when PatientGenerator generates the patient. We use the field 'lastTime' to record the time that last patient was generated. The id of patient is 1 initially and will increment by 1 every time a patient is generated. All the random numbers mentioned here are chosen from the stated range with perfect uniformity.

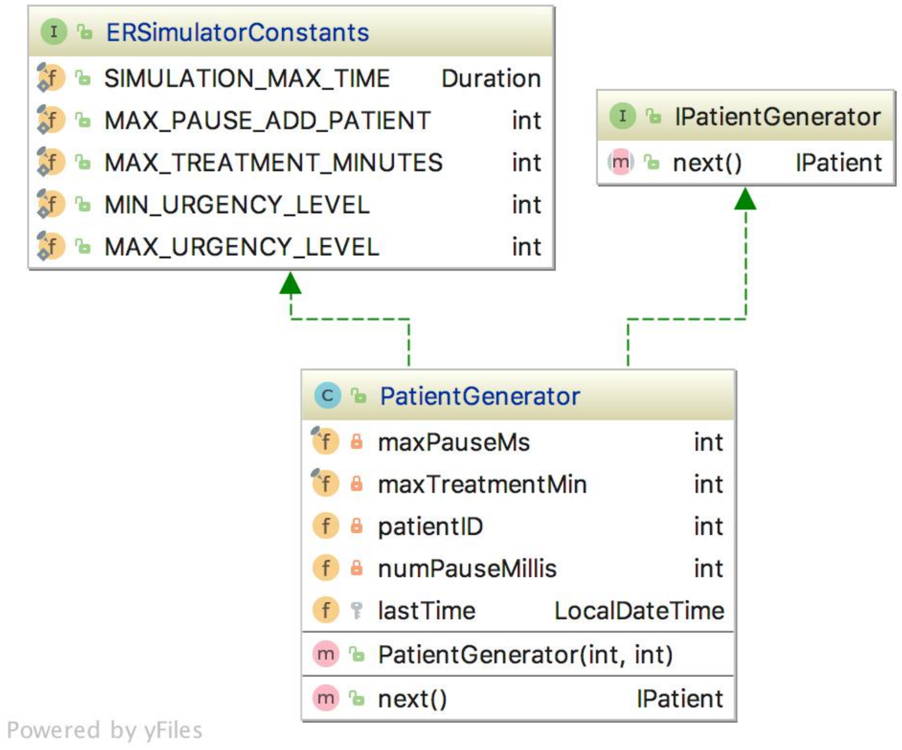


Fig 3. UML of PatientGenerator

ExaminationRoom.java

The examination rooms are represented by ExaminationRoom objects. The natural order of examination rooms is by the busy time. The room with less busy time has higher priority. When the examination room takes in a patient, the method addBusyTime(Duration busytime) will be called. This will add the treatment time of patient to the room's busy time, and the number of patients treated in this room will increment by 1. The getter getBusyTime() returns the busy time of this room. This method will help us calculate the busy percentage ($\frac{busy\ time}{overall\ simulation\ time} \times 100\%$) at the end of simulation. The toString() returns a string containing the busy time and number of patients of this room.

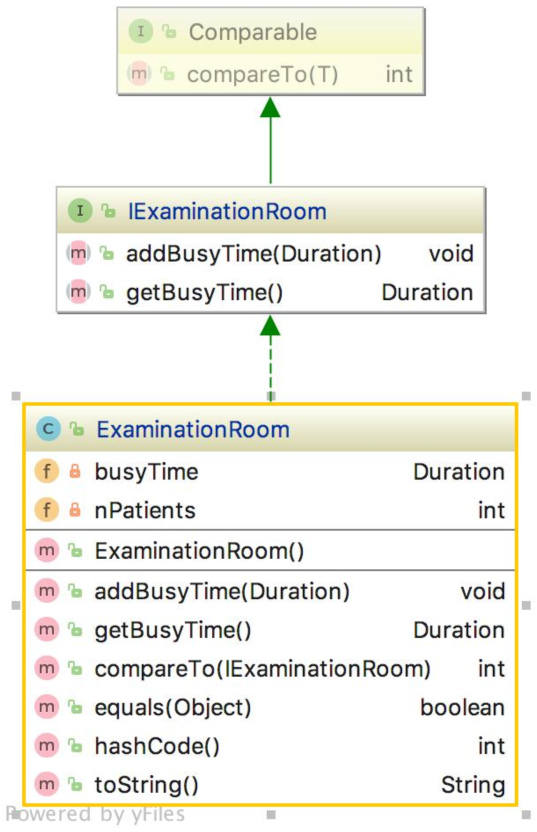


Fig 4. UML diagram of ExaminationRoom

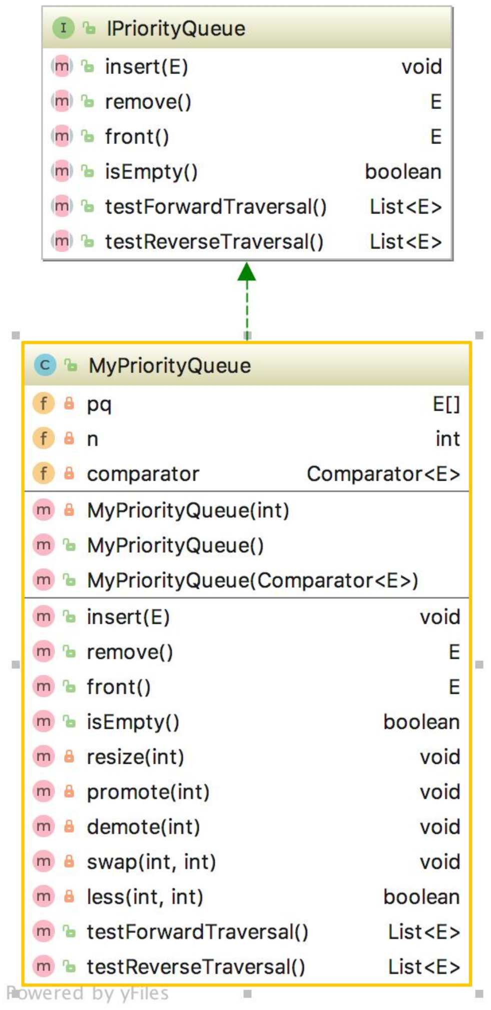


Fig 5. UML diagram of MyPriorityQueue

MyPriorityQueue.java

The last major class is the MyPriorityQueue<E extends Comparable<E>>. The objects are added to priority queue according to their priority. The main characteristic of PriorityQueue is that we can get the highest priority element in O(1). The implementation of PriorityQueue is based on a binary heap[1]. It provides two public constructors where user can choose to use a comparator or the natural order. The front() and isEmpty() take O(1) time, while the insert(E) and remove() take O(lg N) time. This class is used to model the event lists in ERSimulator.

The UML diagram of the entire package is shown on page 6.

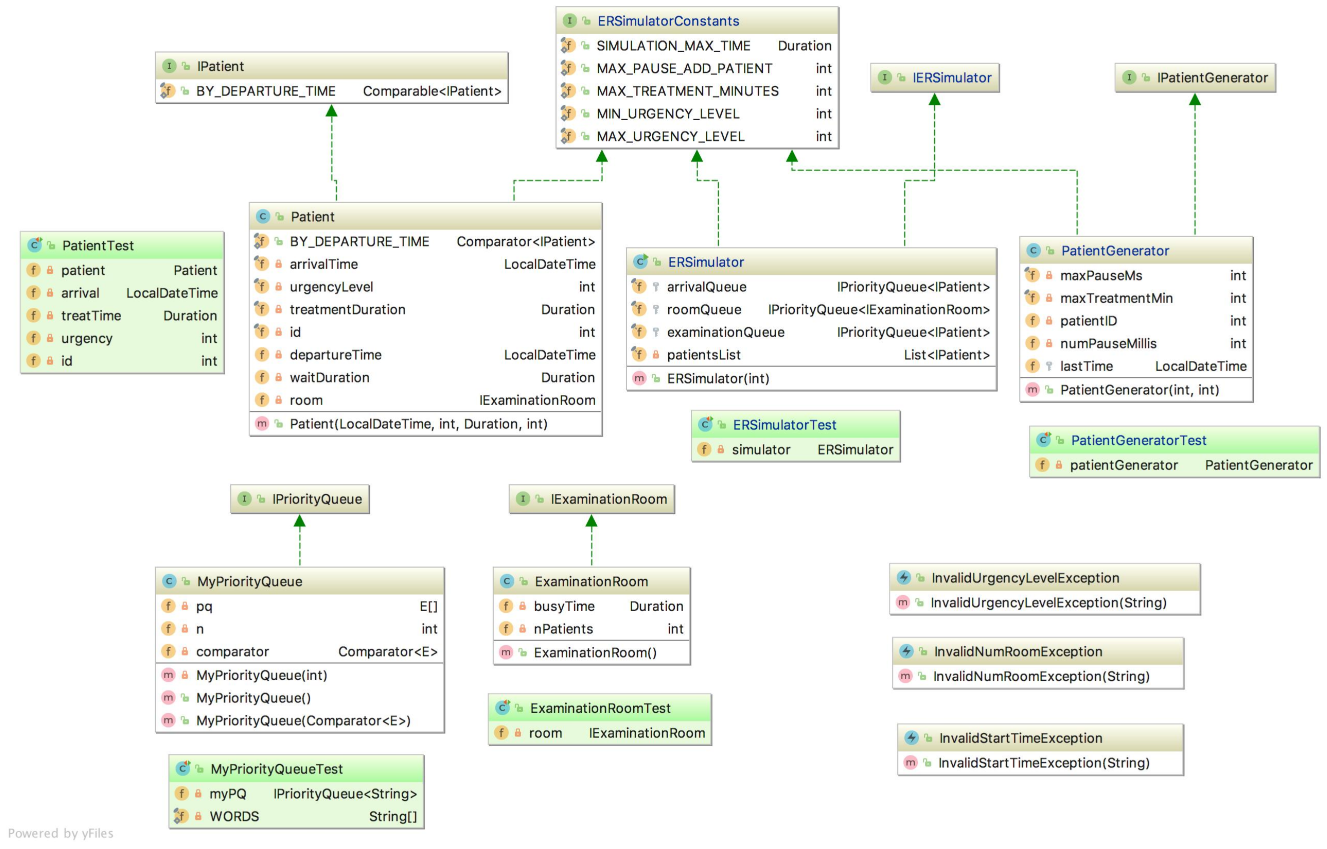


Fig 6. UML of entire package

Testing

All the classes mentioned above are tested using JUnit tests. Please refer to the java documentation for details in testing.

<u>MyPriorityQueueTest.java</u>
The MyPriorityQueue is tested using String and Patient as the object type. After inserting the elements to priority queue, we test that front(), remove() and isEmpty() return correct results. And the testForwardTraversal() and testReverseTraversal() also return lists in which objects are ordered by priority.

<u>PatientTest.java</u>
The Patient class is tested by creating a new Patient object. Makes sure that getters return correct info about this patient. Tests the natural order of patients by calling compareTo() method. Tests that Patient throws proper exceptions for null parameters, invalid urgency levels, invalid treatment time, and invalid start time. The equals(), hashcode(), and toString() methods are also tested.

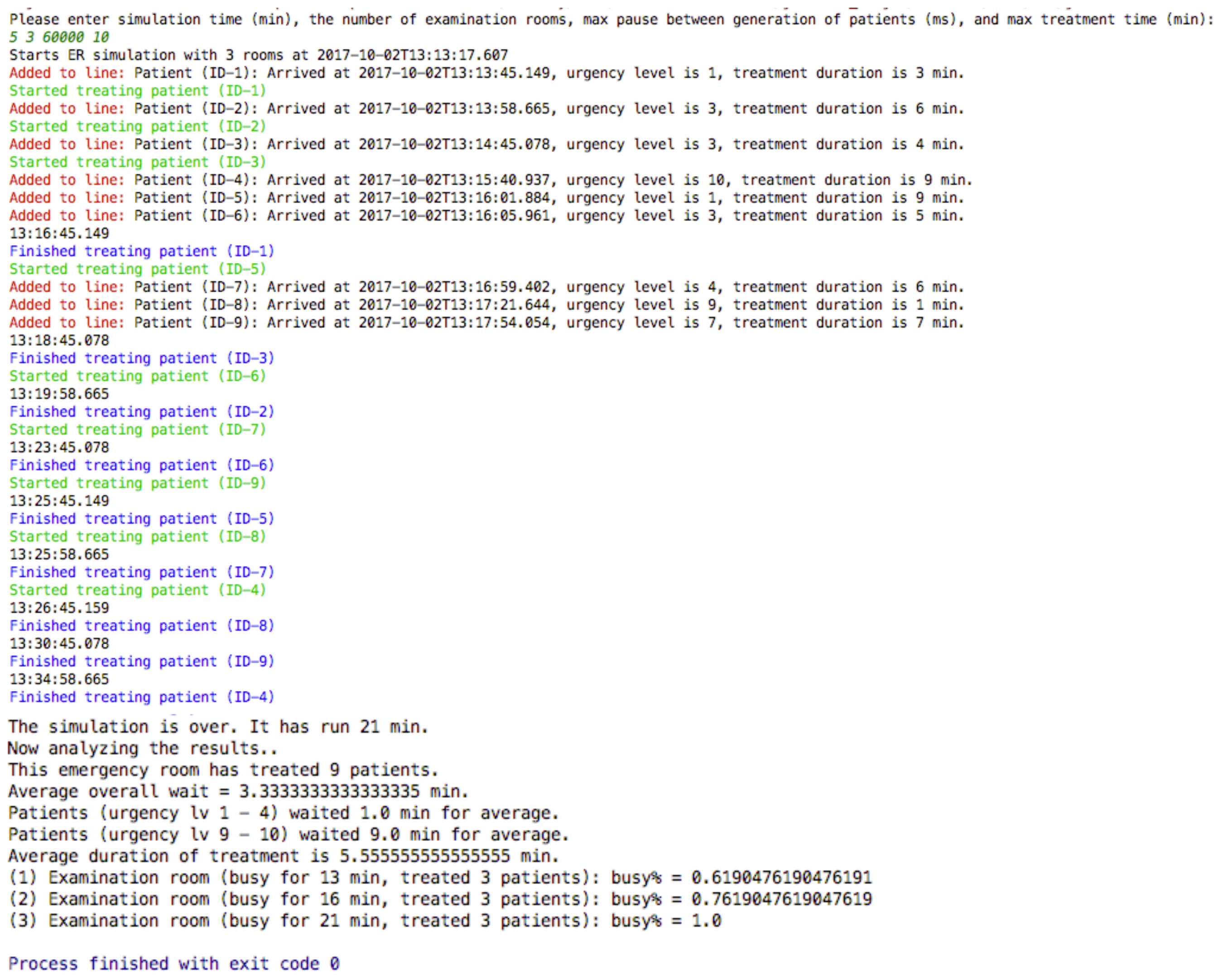
<u>ExaminationRoomTest.java</u>

The ExaminationRoom class is tested by creating a new ExaminationRoom object. Adds busy time to this room and tests that getBusyTime() returns correct amount of busy time. The natural order is also tested by calling compareTo() with other ExaminationRoom. We also test that toString() returns correct info about busy time and number of patients treated in this room.

PatientGeneratorTest.java
The PatientGenerator class is tested by creating a new PatientGenerator object. To help testing, the field lastTime in PatientGenerator is set to be protected so that it can be accessed by this PatientGeneratorTest. Sets the lastTime to be much earlier than current local time, and tests that PatientGenerator generates new patients with correct id when next() is called.

ERSimulatorTest.java
This test won't be able to cover >90% of code because some methods in ERSimulator print message directly to console and it is hard to test the output message in unit tests. Another reason is that part of the simulation is done in main(). As a result, we only did simple testing here by adding a sequence of preset patients and tests that patients in arrival queue are in the order of priority. We also test that room queue and examination queue are empty/busy respectively. To do the test, the roomQueue, arrivalQueue and examinationQueue are set protected so that they can be accessed by this test class. To carefully test that simulation is running as expected, we run the simulation with a few different inputs (simulation time, number of examination rooms), and check the output message on console. An example is given below:

User inputs: simulation time (duration of patient generation) = 5 min;
number of examination rooms = 3;
max pause between patient generation = 60,000 milliseconds;
max treatment duration = 10 min.
Message printed to console: (cont'd on next page)



```
Please enter simulation time (min), the number of examination rooms, max pause between generation of patients (ms), and max treatment time (min):
5 3 60000 10
Starts ER simulation with 3 rooms at 2017-10-02T13:13:17.607
Added to line: Patient (ID-1): Arrived at 2017-10-02T13:13:45.149, urgency level is 1, treatment duration is 3 min.
Started treating patient (ID-1)
Added to line: Patient (ID-2): Arrived at 2017-10-02T13:13:58.665, urgency level is 3, treatment duration is 6 min.
Started treating patient (ID-2)
Added to line: Patient (ID-3): Arrived at 2017-10-02T13:14:45.078, urgency level is 3, treatment duration is 4 min.
Started treating patient (ID-3)
Added to line: Patient (ID-4): Arrived at 2017-10-02T13:15:40.937, urgency level is 10, treatment duration is 9 min.
Added to line: Patient (ID-5): Arrived at 2017-10-02T13:16:01.884, urgency level is 1, treatment duration is 9 min.
Added to line: Patient (ID-6): Arrived at 2017-10-02T13:16:05.961, urgency level is 3, treatment duration is 5 min.
13:16:45.149
Finished treating patient (ID-1)
Started treating patient (ID-5)
Added to line: Patient (ID-7): Arrived at 2017-10-02T13:16:59.402, urgency level is 4, treatment duration is 6 min.
Added to line: Patient (ID-8): Arrived at 2017-10-02T13:17:21.644, urgency level is 9, treatment duration is 1 min.
Added to line: Patient (ID-9): Arrived at 2017-10-02T13:17:54.054, urgency level is 7, treatment duration is 7 min.
13:18:45.078
Finished treating patient (ID-3)
Started treating patient (ID-6)
13:19:58.665
Finished treating patient (ID-2)
Started treating patient (ID-7)
13:23:45.078
Finished treating patient (ID-6)
Started treating patient (ID-9)
13:25:45.149
Finished treating patient (ID-5)
Started treating patient (ID-8)
13:25:58.665
Finished treating patient (ID-7)
Started treating patient (ID-4)
13:26:45.159
Finished treating patient (ID-8)
13:30:45.078
Finished treating patient (ID-9)
13:34:58.665
Finished treating patient (ID-4)
The simulation is over. It has run 21 min.
Now analyzing the results..
This emergency room has treated 9 patients.
Average overall wait = 3.3333333333333335 min.
Patients (urgency lv 1 - 4) waited 1.0 min for average.
Patients (urgency lv 9 - 10) waited 9.0 min for average.
Average duration of treatment is 5.555555555555555 min.
(1) Examination room (busy for 13 min, treated 3 patients): busy% = 0.6190476190476191
(2) Examination room (busy for 16 min, treated 3 patients): busy% = 0.7619047619047619
(3) Examination room (busy for 21 min, treated 3 patients): busy% = 1.0

Process finished with exit code 0
```

In this example, we can see the patients in arrival queue are ordered by their natural order. The simulator starts the examination on patient when another patient finishes the treatment. The average wait for patient (urgency level 9-10) is greater than patient (urgency level 1-4), which is as expected.

More testing with different parameters have been performed to confirm the correctness of simulation.

Evaluation

This project is a good practice on the study of interfaces. It helps us understand the role of interfaces in the program. We have also learned how to deal with exceptions, and when is a good time to write our own exceptions. The practice of JUnit tests has also improved my debugging skill.

Conclusion

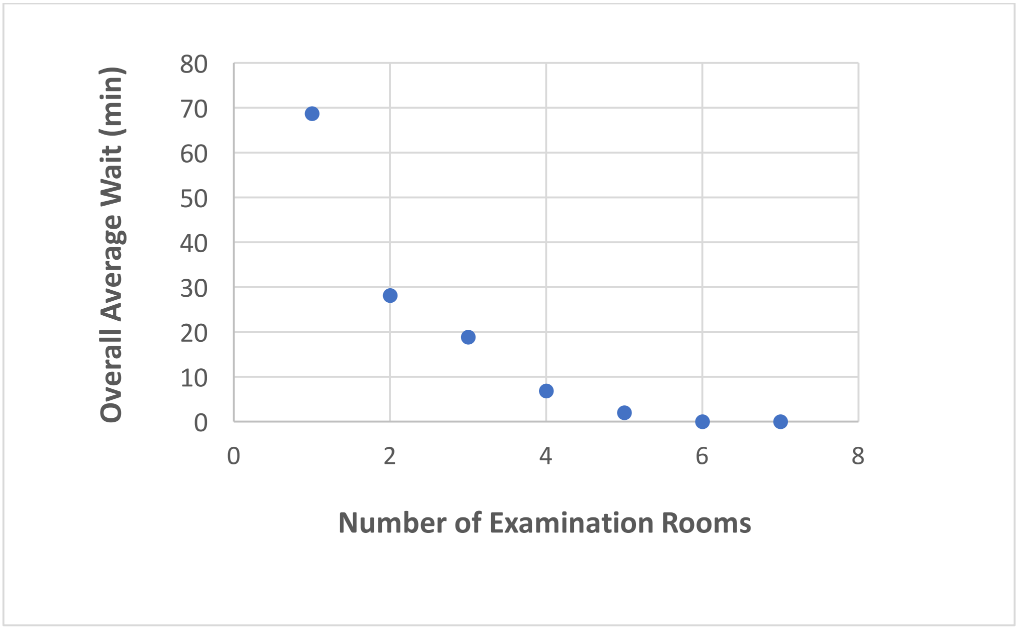


Fig 7. The overall average wait vs. number of examination rooms

(Fig 7 settings: max treatment of patient = 10 min; max pause between patient generation = 2 min; simulation time = 30 min. Notes: this setting is different from the example given in Testing section)

Based on the testing performed, we can see that the more the examination rooms, the less the average wait. When there are more than 6 examination rooms, the average wait is almost zero. When there are 5 rooms, the average wait is about 2 min; the average wait becomes ~7 min when there are only 4 rooms. Compare to the max treatment time of patient, which is set to be 10 min, we can conclude that 4~5 is the optimal number of rooms. In average, patients need to wait the same amount of time as their expected treatment time to get treated.

References

[1] Algorithms, 4th edition by Robert Sedgewick and Kevin Wayne.