Debra Cooperman
cs5010 Assignment 2: ADTs and Object-Oriented Design

ERSimulator Program

Planning and implementing the program:

I planned and organized my program objects based on the key actor in the simulation scenario (**Patient**), the makeup of the **EmergencyRoom** (consisting of a **PatientArrivalQueue** where patients arrive, are triaged, and sent to an **ExamRoom**, an **ExamRoomQueue** that holds the **ExamRoom**s that exist in the simulation, and a **PatientExaminationQueue** that holds Patients assigned to **ExamRooms** until they are finished with their **PatientExams** and exit the simulation). The queues all extend **MyPriorityQueue**, which implements the **IPriorityQueue** interface.
The main characteristic of a priority queue is it always ensures that the most important element is at the front (beginning) of the queue. I built my priority queue based on a minheap, which means that important/smaller (e.g., urgency 1 is higher than urgency 3, and is a smaller number) elements will never have less important elements in front of them.

The **ERSimulator** program works as follows:

Setting up the Simulation and EmergencyRoom:

- Through the command line, the user types in the number of exam rooms that should exist in the **EmergencyRoom,** and how many minutes the simulation should last.
- The program sets the simulation start time to now, and sets the projected end time as now plus the number of minutes typed in by the user.
- The program then creates an **ERStatisticsGatherer,** which is assigned the role of collecting all the statistics on patient visits, patient wait time, exam room visits/utilization, etc.
- An **EmergencyRoom** is created with the **ERStatisticsGatherer** and the number of exam rooms. The **EmergencyRoom** constructs a new **PatientArrivalQueue**, **ExamRoomQueue**, and **PatientExaminationQueue**.
- Each of the queues above is assigned a **QueueComparator** upon creation, which dictates what elements in each queue are considered most important when compared with other elements in the queue.
    - For **PatientArrivalQueue**, Patients with higher (smaller number) urgency are put in front of Patients with lower urgency. If two Patients have the same urgency, they are put in order of arrival (earliest first).
    - For **ExamRoomQueue**, available **ExamRooms** come first. If two **ExamRooms** are both available, the one with the lowest utilization (calculated based on total room visits/total patient visits) comes first.

- For **PatientExaminationQueue**, Patients are sorted by earliest departure time first.

Running the simulation:

- Once the simulation is set up, the **PatientGenerator** creates new patients with different ages and symptoms (temperature & blood pressure measurements). These symptoms are assigned to the Patient's current **Visit**, which is added to this Patient's **PatientHistory**. Urgency is calculated based on a combination of age (risky or not), temperature (high for age or not), and blood pressure (high for age or not), and that urgency is assigned to the patient visit and is used to sort the **Patient**s in the **PatientArrivalQueue**.

- The **Patient** in the front of the **PatientArrivalQueue** is matched up with an available **ExamRoom** in the front of the **ExamRoomQueue**. When a Patient finds an available **ExamRoom**, the **ExamRoom** is marked as unavailable and re-sorted with the other ExamRooms in the queue, the Patient is removed from the **PatientArrivalQueue** and sent to the PatientExaminationQueue, and the **PatientExam** is kicked off in a separate thread. A Patient Exam updates all the needed statistics and waits until the projected patient duration time has elapsed (based on patient urgency). Then it removes the **Patient** from the **PatientExaminationQueue**, makes itself available again, re-sorts the ExamRoomQueue, and reports back that it is done.

- If a **Patient** at the head of the **PatientArrivalQueue** finds that no ExamRooms in the **ExamRoomQueue** are available, it just waits until one of the ExamRooms becomes available and reports back. I used a CompletionService to manage the threads because a CompletionService will report back as each thread finishes vs. waiting for them all to finish.

- When the simulation time ends, the CompletionService will wait for all of the current PatientExam threads to finish before shutting down.

- After all the PatientExam threads finish, the ERStatisticsGatherer prints out a report containing all of the efficiency statistics (One such report is shown below).

Testing the program:

I spent a lot of time making sure that **1)** the queues were working correctly in terms of how the **Patients or ExamRooms** were sorted and **2)** finding an efficient way **for ExamRooms** to report back when they became available **3)** Finding a way **to update the ERStatistics in a** threadsafe way. The first I was able to test using unit tests **but** the latter two required me to run multiple simulations from multiple minutes up to multiple hours. Some of the bugs I ran into were silly, for example at first I was kicking off a thread every time a **Patient** couldn't find an available **ExamRoom—and** had that thread wait the shortest amount of time until an **ExamRoom** freed up. Because new **Patient**s were getting created a lot more quickly than **ExamRoom**s were getting freed up, so many threads were created that the system ran out of memory pretty quickly. I later realized that I should be managing the threads based on when an **ExamRoom** gets filled. Then the maximum number of threads is equal to the maximum number of **ExamRooms**, which are going to be far fewer than number of **Patient**s, and when a room is unavailable the Patient can use the take() method from the **CompletionService** to be notified when the next **ExamRoom** thread completes and the **ExamRoom** becomes available. With regard to the **ERStatistics**, I realized that since different threads could be updating the statistics at the same time, I needed to use atomic variables to increment so the statistics would be valid. Also I needed to synchronize methods such as insert and remove, since the queue needs to be re-sorted after those operations to maintain the min-heap property and keep the **Patient** and **ExamRoom** elements in the right order.

That said, although I'm sure I can be more efficient, I was able to implement all the functionality in the assignment. In terms of what is the most efficient number of **ExamRooms,** I was **never able to get above more than 20% utilization. 15 ExamRooms seem about right for achieving 10 to 15% utilization for each room.**

Simulation Reports: Two 8 hour runs (user typed in 8 hours)

```
******************ER SIMULATION REPORT*****************************

Total number of examination rooms in the system: 10

Total simulation duration (in hours): 9.6

Total number of patients treated: 1422

 Average wait time for all patients (in minutes): 53.2
        Average wait time for high urgency patients (in minutes): 53.4
        Average wait time for medium urgency patients (in minutes): 60.8
        Average wait time for low urgency patients (in minutes): 52.3

Average treatment duration (in minutes):            4
 Total number of patients treated & percentage utilization by examination room:

        Room Number: 5
        Total patients treated: 58
        Percentage utilization: 4.7%
        Room Number: 6
        Total patients treated: 64
        Percentage utilization: 5.1%

        Room Number: 8
        Total patients treated: 96
        Percentage utilization: 7.3%

        Room Number: 2
        Total patients treated: 82
        Percentage utilization: 6.6%

        Room Number: 10
        Total patients treated: 157
        Percentage utilization: 12.6%

        Room Number: 9
        Total patients treated: 174
        Percentage utilization: 13.3%

        Room Number: 7
        Total patients treated: 110
        Percentage utilization: 9.1%

        Room Number: 4
        Total patients treated: 234
        Percentage utilization: 18.5%

        Room Number: 3
        Total patients treated: 225
        Percentage utilization: 18%

        Room Number: 1
        Total patients treated: 222
        Percentage utilization: 15.6%


Process finished with exit code 0

*******************ER SIMULATION REPORT******************************

Total number of examination rooms in the system: 10
Total simulation duration (in hours): 9.9

Total number of patients treated: 1533

Total high urgency patients: 947

Total medium urgency patients: 43

Total low urgency patients: 543

Average wait time for all patients (in minutes): 55.5
        Average wait time for high urgency patients (in minutes): 56.4
        Average wait time for medium urgency patients (in minutes): 54.6
        Average wait time for low urgency patients (in minutes): 54.1

Average treatment duration (in minutes):            3.9
Total number of patients treated & percentage utilization by examination room:

        Room Number: 6
        Total patients treated: 70
        Percentage utilization: 5.4%


        Room Number: 4
        Total patients treated: 94
        Percentage utilization: 6.3%


        Room Number: 9
        Total patients treated: 77
        Percentage utilization: 6.8%


        Room Number: 3
        Total patients treated: 108
        Percentage utilization: 7.3%


        Room Number: 10
        Total patients treated: 135
        Percentage utilization: 9.1%


        Room Number: 8
```

```
Total patients treated: 212
Percentage utilization: 16.5%


Room Number: 7
Total patients treated: 220
Percentage utilization: 14.4%


Room Number: 2
Total patients treated: 161
Percentage utilization: 13.4%


Room Number: 5
Total patients treated: 158
Percentage utilization: 10.6%


Room Number: 1
Total patients treated: 298
Percentage utilization: 20%
```