CSC 115: Fundamentals of Programming II
Assignment # 4: Emergency Room Triage

Due date

Monday, April 9, at 9:00 am via submission to conneX.

How to hand in your work

Submit the requested files through the Assignment #4 link on the CSC 115 conneX site. Please make sure you follow all the required steps for submission (including confirming your submission).

Learning outcomes

When you have completed this assignment, you are expected to be able to:

- Implement the Heap ADT
  - Use the java.util.ArrayList as the underlying data structure.
  - Handle generic types that implement the Comparable interface.
- Create a specific PriorityQueue that simulates the ordering of patients presenting to the triage desk of an Emergency Room.
  - Use the heap as the underlying data field,
- Continue to apply good programming practice in
  - proper documentation, and
  - testing and debugging problems with code.

Hospital Admission Information System Description.

The local hospital Emergency Room (ER) admits and treats patients that present themselves for medical treatment. Each patient is assessed by a triage nurse, who performs a preliminary assessment to determine the priority level of the patient's presenting complaints. If the patient needs immediate attention, the triage nurse provides enough intervention to stabilize them. An ER patient record is initialized, containing the patients's name and complaint. A priority level is added to the record by the triage nurse.

Patients are released from "triage" into the ER proper once there is an available Emergency Room Physician who can provide further assessment, treatment, diagnosis and referrals as needed.

ER physicians treat patients in the order of the triage queue, which manages an order determined firstly by the triage nurse's assessment and secondly by the amount of time that has passed since the patient registered.

Problem description
We are asked to design a system for the triage area. The triage requirements for insert (admission to triage) and remove (when the patient is released from triage to the ER physician's care) are familiar. They describe the requirements for a PriorityQueue ADT. We expect that every entry into the queue will also be removed from the queue, so are aware that both inserts and removals should be equally efficient. The Heap data structure is a perfect data field for this triage queue.

Programming requirements

You will write the source code for the following classes, which have been started for you:

- *Heap.java.* This implementation is for a standard heap data structure that contains generic objects that implement the *Comparable* interface. Using generics allows us to re-use this program for a variety of systems and elements, while ensuring that elements implement *Comparable* allows the heap to use the method *compareTo* to determine the ordering property of the elements.
- *EmergencyTriage.java.* This is basically a priority queue for Emergency patient records. Records are released out of the queue once the patient is released from triage. This program uses a heap data structure (implemented in this assignment) as its main data field.

The following fully-developed Java files are provided to support the code you write:

- *NoSuchCategoryException.* The triage nurse has a list of categories of priority based on the urgency of the patient's condition. If he enters a category that does not exist, this the exception that is thrown.
- *ERPatient.java.* This is the information that is contained with the patient's record in triage. Along with the name, it includes the triage priority number and an ordering number that indicates how long a patient is waiting in triage.

Implementation details

All the specifications documentaion can be accessed via the following link. (Make sure you are logged onto conneX when accessing the link directly.)

To get you started, we recommend the following:

1. Download the files associated with this assignment into a fresh working directory (csc115/assn4 for example).
2. Read through all the source code. Keep a copy of the specifications close by when developing the code that you are writing.
3. The Heap has been started for you. Finish the minimal class, copying and pasting the required specified methods.
    a) The main data field is an java.util.*ArrayList* data structure that contains generic values that implement *Comparable.* This makes sense in that the heap imposes an ordering, which every *Comparable* object handles with its

required *compareTo* method. Read the Java Documentation available for this class.

b) The reason we use the ArrayList, instead of the basic array we are used to implementing, is that Java has no intuitive way to handle generics and arrays. The ArrayList gives us all the functionality of the arrays, but has some oddities that we can guide you through as you continue reading this document.

c) We've started the constructor for you. The java.util.ArrayList only expands as values are inserted. Because the *Heap* methods have cases where elements at given index positions are *updated, or replaced,* we recommend that you **not** add or remove elements in the ArrayList (we explain the reason in 'd'). The methods *set* and *get* are preferred. To allow us to limit ourselves to only these methods, we initialized the list by filling it to capacity with *null* values. We also recommend that you maintain your own *size* variable that keeps track of the number of elements in the list.

d) Note that there are many methods in ArrayList, not all of them helpful for our current needs. The only ones you need to use are *get* and *set*. The following examples demonstrate how to accomplish the array handling by using the ArrayList (`arraylist`) for equivalent statements using the basic array (`array`)

```
array[i] = someElement;   <=>  arraylist.set(i, someElement);

someElement = array[i];   <=>  someElement = arraylist.get(i);
```

4. Compile each of the files.
5. Refer to the textbook's description of the various parts of the Heap ADT when you start to fill in the methods.
6. Test the heap using the main method.
7. Implement the EmergencyTriage class, using a Heap<ERPatient> as its main data field; this one is also started for you.
    - Let all the methods in this class call the heap data field's methods.

You are encouraged to use any number of extra *helper* methods to make your code easier to understand and handle. You may not create helper classes.

Internal testing:

The EmergencyTriage, since it uses all the Heap methods, can simply register a random list of patients, and then admitToER, printing out the results. If the result is sorted by lastname, then you can be assurred that this class is working as it should.

Since most of the work occurs in Heap, it will need to be tested thoroughly. Note that since the Heap will handle any *Comparable* element, you can test it with something simple, like Integer or String objects. Make sure that you test each of the methods you write, either by calling it from another method, or by setting up test cases and calling from the *main* method.

The following output from a smaller version of the author's Heap internal testing (using String elments) may help you design your own testing:

```
demo$ java Heap
Heap has 5 elements
1:alpha
2:beta
3:gamma
4:delta
5:epsilon
Emptying the heap:
Removing alpha
Heap has 4 elements
1:beta
2:delta
3:gamma
4:epsilon
Removing beta
Heap has 3 elements
1:delta
2:epsilon
3:gamma
Removing delta
Heap has 2 elements
1:epsilon
2:gamma
Removing epsilon
Heap has 1 elements
1:gamma
Removing gamma
Heap has 0 elements
Checking error handling
Exception handling works
demo$
```

The files to submit:

1. *Heap.java*
2. *EmergencyTriage.java*

Grading

- Submissions are expected to follow all of the requirements for the submitted files in the specification documents.
- Evidence of internal testing must be present: you may comment out any testing the main methods if you wish.
- The coding conventions, specifed in the codingConventions.pdf document on conneX, must be followed.
- Please post questions to the forum set up for this assignment; any clarifications posted there are part of the overall specifications. If there are any major changes, this will be announced and you will receive an email.