CSC 115: Fundamentals of Programming II
Assignment #5: The Emergency Room

Due date:

Wednesday, December 5, 2018 at 23:55 pm via submission to conneX.

Objectives: Upon completion of this assignment you need to be able to:

- Create a binary tree using a *Vector*, which is Java's version of a resizable array.
- Create a version of the *PriorityQueue* ADT, using a Heap.
- Apply *data abstraction*, hiding a complex data type inside a simpler ADT.
- Experience the alternative to generics in Java, by casting the Object class.
- Continue to apply principles of inheritance, encapsulation and modularity.
- Continue to apply good programming practice in
  - designing programs
  - proper documentation,
  - testing and debugging problems with code.

Resources:

- CSC 115 Java Coding Conventions.pdf (found on conneX, under Resources)
- Textbook Chapter 12
- The specification pages provided.

Introduction:

A group of patients are sitting in the local hospital's Emergency waiting room, when the attending ER physician arrives for her shift. The triage nurse has already assessed patients by their main complaint and provides an electronic device whereby the physician can touch the screen and the next patient chart is provided. The ordering of the charts is determined by the patient's priority and then time of check-in. You have been selected to write part of the software that stores the patient information and prioritizes the list for the ER physician. The project leader has elected to use a priority queue based on an array-based heap to store the *ER_Patient* objects.

Quick Start:

1) If you haven't done so already, download all the necessary documents for this assignment from conneX into a specific folder for CSC115 assignment five, assn5 is a recommended name. You may want to separate the documentation files into a separate directory.
2) All of the specification documentation for public class and public methods can be found in the appropriate *.html files.

3) The following classes are already complete:
    - ER_Patient.java
    - NoSuchCategoryException.java
4) Familiarize yourself with the *ER_Patient* class, provided. Note that it implements *Comparable*, but unlike the previous assignment, has no specific generic markings. Because of this, the *compareTo()* method takes an Object parameter and must make sure that the other Object is actually subclassed as an *ER_Patient* before doing the comparison.
5) **Complete** the *Heap* class. Note that we are not using generics in this class and the warnings have been suppressed. The *Heap* only handles objects of type *Comparable*. (Note that since an *ER_Patient* implements the *Comparable* interface, any object of type *ER_Patient* is by inheritance an object of type *Comparable*.)
6) **Complete** the *PriorityQueue* class, noting that the *Heap* object is the main data field, and can handle most of the work the priority queue needs to do.

Detailed Instructions:

Chapter 12 of the textbook provides most of the background on the array-based heap and the *PriorityQueue* ADT. Note that within the *PriorityQueue*, the *Heap* is completely hidden from the user, invoking the **Information Hiding** aspects of O-O programming.

Detailed steps to completing the Heap class:

1. The shell is provided for this class. Fill in the comments above each of the public methods and write the code that makes these methods work.

2. Add additional data fields as necessary. Add additional private methods that help maintain both the structure and ordering of the Heap array during inserts and removals. A private print out of the array is always recommended, as are separate methods for *bubbling* an item upwards or downwards.

3. You **must not** change the provided method headers or the given data fields. There are two reasons for this requirement:
    i. You are a programmer on a team; others are expecting their code to plug into your code.
    ii. You are a student in CSC115, doing an exercise that enhances your skills as a programmer. To obtain adequate feedback, the marker(s) are counting on easy access to your implementation.

4. Why are we using a *java.util.Vector* instead of basic array? Arrays in Java were originally copied from the C programming language. They behave as Objects but do not follow the standard rules for proper O-O programming.

Once Java introduced generics, the array was left behind. There is no way to create a basic array that handles generic objects. The *Vector* class has all the functionality of a resizable array and it can handle generic objects.

5. **You must test every method internally**. The more rigorously you test, the more robust your code. Do not move to the next step until you are confident that the Heap works as expected.
    - Note that the *ER_Patient* internal class uses *Thread.sleep()* to spread a single second between admit times. You may use this technique or choose the constructor to create your own admit times. For the sake of the marker's sanity, **DO NOT** use *Thread.sleep()* in any method other than main.

Detailed steps to completing the PriorityQueue class:

1. The shell is provided for this class. Fill in the comments above each of the public methods and write the code that makes these methods work.
2. Let the hidden *Heap* data structure do all the work in each of the *PriorityQueue* methods.
3. Test the *PriorityQueue*. If the *Heap* has been thoroughly tested, it is sufficient to insert a few patients and then dequeue and print until the queue is empty. A sorted print-out indicates success.

A Note about the Heap vs. the PriorityQueue:

A *PriorityQueue* can use a linked data structure, an un-ordered array, an ordered array, or a tree as its underlying data structure. We use the *Heap* in this exercise, because the *Heap* data structure is so beautifully similar to the *PriorityQueue* and takes O(log n) for both insert and remove operations. However, the *Heap* is a complete binary tree data structure and is not necessarily bounded by the *PriorityQueue* ADT. For instance, it is allowed to have an iterator and it is allowed to have a size method. There are other tree-related methods such as *isInternal()* that are not part of the *PriorityQueue* ADT. There is also nothing stopping a *Heap* from deleting an item in the middle of its structure, although there are better data structures for that kind of operation.

The *PriorityQueue* ADT is much more limiting. For that reason, you want to make sure that the user does not ever have access to the *Heap* itself. It is desirable to have the *PriorityQueue* hide any extra functionality of the *Heap*.

Submission

Submit the following completed files to the Assignment folder on conneX:

- Heap.java
- PriorityQueue.java

Please make sure that conneX has sent you a confirmation email. Do not send [.class] (the byte code) file, or any another file for this assignment. Also make sure you submit your assignment, not just save a draft. All draft copies are not available to the instructors, so we cannot mark them.

Note about Academic Integrity: It is OK to talk about your assignment with your classmates, and you are encouraged to design solutions together, but each student must implement (code) their own solution. We will be using plagiarism detection software on your assignment submissions.

Grading

All submitted java files must compile. The marker will be looking for:

- Proper programming style as per the code conventions on CSC115 conneX Resources.
- Source code follows the specifications and instructions.
- Good modularity: well-defined helper methods.
- Internal testing, using the main method as a test harness.
- You will receive no marks for any Java files that do not compile.

Requirement	Marks
Proper programming style as per the code conventions on CSC115 conneX Resources	2
Pass tests for Heap class (7 tests)	
Constructor (1 test)	1
isEmpty(), size() and getRootItem() methods (2 tests)	2
insert() method (3 tests)	3
removeRootItem() method (1 test)	2
Pass tests for PriorityQueue class (6 tests)	
Constructor (1 test)	1
isEmpty() and peek() methods (4 tests)	2
enqueue() and dequeue() methods (1 test)	2
Total	**15**