Implementing min-heaps

Preamble

The results of this assignment, A6, are used in A7, which is an implementation of Dijkstra's algorithm to find a shortest path in a graph —what google maps uses to find the best route from one place to another.

A6 involves implementing a heap with the added functionality of being able to change a priority. A6 requires meticulous attention to detail. So that it doesn't take *too* long, we give you a complete JUnit testing program for it.

Keep track of how much time you spent on A6; we will ask for it upon submission.

Read this whole document before beginning to code.

We suggest getting A6 done *well* before the due date. The last weeks of the course can be stressful.

Collaboration policy and academic integrity

You may do this assignment with one other person. Both members of the group should get on the CMS and do what is required to form a group well before the assignment due date. Both must do something to form the group: one proposes, the other accepts.

People in a group must *work together*. It is against the rules for one person to do some programming on this assignment without the other person sitting nearby and helping. Take turns "driving" —using the keyboard and mouse.

With the exception of your CMS-registered group partner, you may not look at anyone else's code, from this semester or earlier ones, in any form, or show your code to anyone else (except the course staff), in any form.

Getting help

If you don't know where to start, if you don't understand testing, if you are lost, etc., please SEE SOMEONE IMMEDIATELY —an instructor, a TA, a consultant. Do not wait. A little in-person help can do wonders. See the course homepage for contact information.

The release code

We give you four files in zip file a6release.zip:

1. Class Heap. The methods you must write are marked with a comment "//TODO …". Leave the "//TODO …" comments in the program. Complete other stubbed-in methods *only* if you are going to use them.
2. JUnit class HeapTester, which has methods to completely test the methods you write in point 2.
3. Class HeapException —you have to throw a HeapException in certain circumstances.
4. Class ArrayHeap, which is a simple implementation of heaps in an array, with no ability to change the priority of an element in the heap. This class is here to show you a simple implementation of a heap. We suggest using it as an example for writing your methods —change these methods to fit the new context.

Place all files in the default package in a new project called a6Heaps (or whatever you want to call it). You may have to put JUnit 4 on the build path. Right click on the project -> build path -> configure build path; click Add Library and select JUnit; select JUnit 4 from the drop down; and click finish. Or, direct Eclipse to insert a new JUnit Test Case, and it will ask whether Unit 4 should be used.

What to do for this assignment

Your job is to implement the methods in class Heap that are marked "//TODO". These are: add and bubbleUp (together), peek, poll and bubbleDown (together), and finally changePriority.

Hints, guidelines, suggestions, requirements

1. You must implement the methods marked with "//TODO" *so that they have specified time bounds*. You may write additional private methods in the class, but be sure to specify them well. Several methods are stubbed in that do not have a note "//TODO ". You do not have to write these, and if you do, you can change their specifications. We will not test them. We placed them there because *we* found them useful.
2. Points may be deducted if methods you add do not have good javadoc specifications.
3. Points will be deducted if you violate the hints given in the bodies of the "//TODO" methods.
4. Class HeapTester does all necessary testing. If running HeapTester does not show an error, your class Heap should be correct.
5. We have declared fields in Heap and written a class invariant for Heap. Study the class invariant. Do not declare any other fields. Point 6 discusses the reason for the HashMap.
6. Special problem. Class Heap would be easy to write, using just an ArrayList for the heap, except for one issue. A call changePriority(v, p,) changes the priority of value v to p. This requires finding value v in the ArrayList, and without any other data structure to help, this could cost time linear in the size of the heap!

   To overcome this problem, we do two things:

   - Have a static inner class Prindex (for PRIority and INDEX) whose fields are (1) a priority and (2) an index into the ArrayList. Then, each item in the ArrayList is represented by an object of this class.
   - Have a field of class Heap that is a HashMap<E, Prindex>, which maps a value in the heap to an object of class Prindex that contains its priority and index in the ArrayList.

   Of course, when an element in the heap is moved to a different position, field index of the associated element of class Prindex has to be changed accordingly. And, when an element is removed from the heap, it must be removed from the HashMap too.

   Using this technique, the expected time for updating a priority should be O(1) for finding the corresponding element in the HashMap and then O(log n) for updating the heap. The corresponding worst-case times are O(n) and O(log n).
7. You would do well to use class ArrayHeap as a good example of how to write the required methods. But the code has to be changed (in Heap) to take into account three new things:
   1. The values in the heap are kept in an ArrayList instead of an array.
   2. The priorities are separate from the values in the heaps, and
   3. A HashMap is used to map an element in the heap to an object that contains (a) the priority of the value and (b) the index in the ArrayList where the value resides in the heap.

What to do submit

1. Remove all your println statements from class Heap; 5 points will be deducted if your code outputs anything.

2. In the comment at the top, put the hours hh and minutes mm that you spent on this assignment. Be careful changing hh and mm; don't change anything else. Put your name(s), netid(s), and write a few lines about what you thought about this assignment.

3. Submit (only) file Heap.java on the CMS.