COMP 251 Assignment 1

Prepared by Prof. Michael Langer

Posted: Thurs. Jan. 16, 2014
Due: Sun, Jan. 26, 2014 at 23:59.

General Instructions

- The T.A.s handing this assignment are Bentley Oakes and Syed Ahmed. Their office hours will be posted on the public web page. Office hours will be in Trottier 3110.

- Use the mycourses discussion boards for clarification questions. Prof. Langer will monitor the discussion boards up until Fri. Jan. 24 5 pm.

- Do not change any of the given code, and add code only where instructed.

- Do not change the package name **(a1posted)**, as this slows down the grading.

- The starter code includes a test class that you may use to examine if your methods are correct. We reserve the right to test your code using a more elaborate set of test cases than the ones provided. If you identify situations that are not covered by the given test cases, then everyone would appreciate it if you would share these on the discussion board. (But do ***not*** hand in your tester class. If you do hand it in, the grader will ignore it.)

- You must use Java naming conventions for variable names (click here).

- Comment your code so that the grader can easily follow what the code is doing. Points may be removed for poor style (non-sense variable names, non-existing or unhelpful comments).

- Submit your code using the myCourses assignment dropbox. Include your name and student ID number as a comment at the top of any file. If you have any issues that you wish the TAs to be aware of when grading, then include them as a comment at the top of your file.

- Failure to follow any of the above these instructions may result in a penalty.

- **Late assignment policy:** Late assignments will be accepted up to only 3 days late and will be penalized by 20 points per day. If you submit one minute late, this is equivalent to submitting 23 hours and 59 minutes late, etc.

Introduction

The purpose of this assignment is to give you some “hands on” experience with a heap implementation of a priority queue. It also requires you to work with common and useful Java classes that such as ArrayLists and HashMaps.

You are given:

- a class **Heap** which is a basic binary heap implementation of a priority queue. This class has only a few public methods. Clients can add priorities (keys) and can access the *min* priority. This class stores only priorities, however. It does not store the objects that have these priorities. (This is very different from the Java **PriorityQueue** class, which stores objects that can be compared to each other.) As such, the **Heap** class is not a very useful class. Removing the minimum priority doesn’t help you if you don’t know which object is associated with that priority.

- a partially implemented class **IndexedHeap** which allows a client to change the priorities of objects (see lecture 4). Here, each “node” of the heap has both a priority (a double) and an object name (a string). For example, we could use this heap to store a set of vertices of a graph. To do this, we would store each vertex name along with the priority of that vertex. This name could be “v1”, “v2”, …”vn”. Notice that this **IndexedHeap** class does not use any sort of explicit Node class. Instead, two arrays (**priorities** and **names**) are used.

  The **upHeap()** and **downHeap()** methods are provided for you. Note that these methods call a **swap** method which you will need to implement.

  There is also a hash map **nameToIndex** which is the inverse mapping of the array **names**. (The **names** array is a map from array indices to names of objects. You can think of it as a map “indexToName” as in the lecture 4 slides.) In the inverse mapping **nameToIndex,** a name is mapped to the index in the **names** array, where that name is found.

  The subtlety of the **IndexedHeap** is that you need to understand and maintain the proper relationship between the three maps: **priorities**, **names**, *and* **nameToIndex**. Specifically, **names** and **nameToIndex** should be the inverse of each other, and the **priorities** and **names** arrays (ArrayLists, in fact) should have corresponding elements. These proper relationships should be invariant to adding or removing elements, or changing a priority.

Finally, as discussed in lecture 4, one typically refers to the priorities of a heap as “keys”. However, in the case of an indexed heap (or more generally, an indexed priority queue), we also have a map from object names to heap indices and so one may be tempted to refer to these object names as keys as well. (Indeed, it is very common that we use maps to index objects by their key names.) Using the term “keys” for both of these meanings would be confusing, however. To avoid this confusion, I have avoided the term “key” in this assignment. And I would like you to avoid this term as well, if you ask questions on the discussion forum.

Your Task

Implement the following methods in the **IndexedHeap** class**.** Each is worth 25 points. There are 100 points total on this assignment.

- **swap**
- **removeMin**
- **add**
- **changePriority**

What to Submit

- A single file called **IndexedHeap.java** with your name and student ID at the top. Please read the General Instructions carefully to avoid problems with your submission.

Have fun, get started early, and good luck!