CSC115 Assignment 5:
The Emergency Room

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.
- Continue to apply principles of inheritance, encapsulation and modularity.
- Continue to apply good programming practice in
  - designing programs,
  - proper documentation, and
  - testing and debugging code.

References:

- CSC115 Java coding conventions.
- 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 tasked with the 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) Create a fresh directory to contain this assignment: `CSC115/assn6` is a recommended name. Download this pdf file and the `.java` files to this directory.

(2) Download and extract the `javafiles.zip` and `docs.zip` in the same directory. A `docs` directory will be created to store the specifications for the `public` classes. All the document links in this document are accessible if they are stored in the `docs` directory local to this pdf file. A light blue text segment indicates a link.

(3) The following files are complete:

- `ER_Patient.java`
- `NoSuchCategoryException.html`

(4) The following files are partially done and need to be completed:

- `Heap.java`
- `PriorityQueue.java`

Descriptions of the Helper Classes

Chapter 12 of the textbook provides most of the background on the array-based heap and the priority queue 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 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:

   (a) You are a programmer on a team; others are expecting their code to plug into your code.

   (b) Your 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 sanit, 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 PriroityQueue:

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 bound by the `PriorityQueue`'s interface definitions. For instance, it is allowed to have an iterator and it is allowed to have a `size` method. 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 you have submitted the required file(s) and conneX has sent you a confirmation email. Do not send `[.class]` (the byte code) files. Also, make sure you *submit* your assignment, not just save a draft. Draft copies are not made available to the instructors, so they are not collected with the other submissions. We *can* find your draft submission,, but only if we know that it's there.

A 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 their own solution.

Grading

Marks are allocated for the following:

- Proper programming style is demonstrated, as per the coding conventions on CSC115 conneX Resources. All instruction comments must be removed from the code.
- Source code that follows the specifications and instructions.
- `Heap` uses a `Vector` as its main data field.
- Good data abstraction: well-defined Heap and PriorityQueue.

- 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.