

CSE 214– Homework 4

Instructor: Dr. Ritwik Banerjee

This homework document consists of 3 pages. Carefully read the entire document before you start coding. You are required to implement the `LinkedList` data structure without using any readily available classes or interfaces in Java.

1 Overview

Much like some of your earlier homework assignments, some code is already given to you. Before you start coding, you should study the structure of the given code to get an intuitive understanding of the big picture (*i.e.*, how the classes and interfaces connect to each other). The provided code is as follows, with classes in blue, and the interface in italicized green:

- *Heap*
- `PriorityQueue`
- `Treatment`
- `HealthCenter`

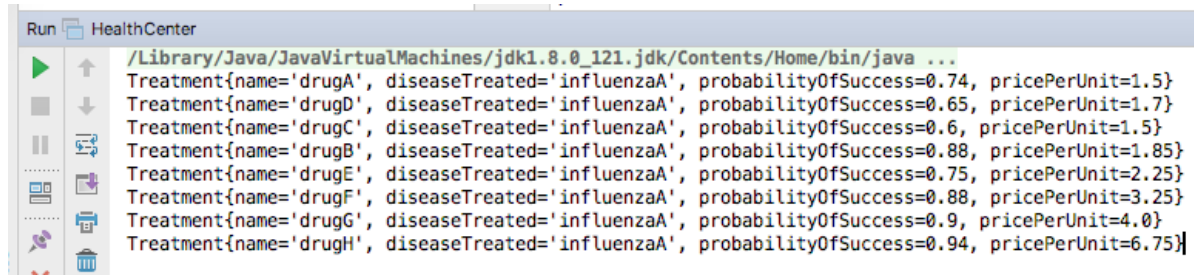
This code will be written with a healthcare application as its motivation. Suppose there is a clinic in some region afflicted by an epidemic, and a lot of people are suffering from a particular illness (recent epidemics like Ebola and Avian Influenza (H1N1) come to mind).

The local health center in the affected region has access to a finite amount of drugs for the treatment of this disease. This resource is presented to you as a file where each line consists of four comma-separated values: (i) *the drug's name*, (ii) *the disease it treats*, (iii) *the probability of success (a score between 0 and 1)*, and (iv) *price per unit of the drug*. When it comes to treating the local population, there are two factors of primary importance:

- (i) the success rate of the treatment, and
- (ii) the price of the treatment.

The first is, of course, important on humanitarian grounds. The second factor is important because in such times of crisis, treatment provisions often depend on government aid, which is limited. Therefore, the health center needs to quickly determine the best treatment option (*i.e.*, which drug should be given to the patient) based on the two factors mentioned above. And to do that, the available treatments are getting stored in priority queues.

Your task is to ensure that the entire available repository of drugs is stored correctly in a priority queue based on treatment efficacy or the drug's price. Of course, as the treatments are provided, the cache of available drugs will reduce. But at any given point, the next best option should be made available quickly to the next patient. All this can be done by correctly building the priority queue of treatments, and by correctly implementing the remove algorithm in priority queues.



```

Run HealthCenter
/Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/bin/java ...
Treatment{name='drugA', diseaseTreated='influenzaA', probabilityOfSuccess=0.74, pricePerUnit=1.5}
Treatment{name='drugD', diseaseTreated='influenzaA', probabilityOfSuccess=0.65, pricePerUnit=1.7}
Treatment{name='drugC', diseaseTreated='influenzaA', probabilityOfSuccess=0.6, pricePerUnit=1.5}
Treatment{name='drugB', diseaseTreated='influenzaA', probabilityOfSuccess=0.88, pricePerUnit=1.85}
Treatment{name='drugE', diseaseTreated='influenzaA', probabilityOfSuccess=0.75, pricePerUnit=2.25}
Treatment{name='drugF', diseaseTreated='influenzaA', probabilityOfSuccess=0.88, pricePerUnit=3.25}
Treatment{name='drugG', diseaseTreated='influenzaA', probabilityOfSuccess=0.9, pricePerUnit=4.0}
Treatment{name='drugH', diseaseTreated='influenzaA', probabilityOfSuccess=0.94, pricePerUnit=6.75}

```

Figure 1: A priority queue based on the unit price of medicines, with the given `treatments.csv` file as the source of data. Lower price is considered better.

2 Tasks

The details of these tasks, in terms of programming implementations, are described in this section. First, look at `HealthCenter.java`. It contains the main method to run this application, and demonstrates how to impose a total order on treatments, based on the price of the medicines. A sample output is shown in Fig. 1.

The Treatment Class:

The basic structure of a treatment (i.e., medication) is provided in this class. You must write the `toString`, and `equals` methods for this class, as well as any constructor(s). Additionally, the comparators used in the main method also appear in this class. That is, you will have to implement the two comparators

- `PriceBasedTreatmentComparator`, and
- `SuccessBasedTreatmentComparator`.

The Heap Interface and the Priority Queue Class:

The heap interface is already defined, with its methods explained in the Javadoc. Follow the documentation provided in this interface to complete the priority queue class.

2.1 Notes

- Do NOT change any code already provided, including the main method.**
- The printed output that you obtain must be in the exact same format as shown in Fig. 1.
- There may be methods that you will need to write, but not mentioned in the given code.
- It is a part of this assignment for you to figure out how to write the code so that the main method as it has been provided works without any change.

Points Distribution	Total: 50 points
Priority Queue Implementation	30 points
Printing a priority queue as shown in Fig. 1	6 points
Priority queue based on price comparator	7 points
Priority queue based on success rate comparator	7 points

3 Guidelines

- **Can I change the given code?**

No. You can (and must, where specified) add your code, but you cannot modify any existing code.

- **Can I add my name under the author tag?**

Yes, of course! But only if you have actually added some significant code in that class. In that case, you SHOULD add yourself as an author using the `@author` tag in Javadoc.

- **What should I submit?**

A single `.zip` file consisting of all the `.java` files (including the interfaces already provided, even though you haven't changed those). The archive must NOT have folders or subfolders within itself. Extract your archive and double-check its structure before submitting to be sure of this.

- As always, uncompileable code or code that does not conform to the above guidelines cannot be tested (and hence, cannot be graded).

Submission Deadline: Apr 21, 2017 (Friday), 11:59 pm
