EECS 2011: Assignment 2

Due: as set in *eClass.*

May be done in groups of up to three students (from either of the sections M and Z)

Motivation

The purpose of this assignment is to implement a binary heap-based priority queue, apply it to implement a heapsort array-sorting algorithm, and to compare the performance of that heapsort to the performance achievable with the sorting methods that are already available in Java.

You are also encouraged to compare your priority queue performance to the one in the `java.util` package[1].

Introduction

In this assignment you will implement a class that extends an `AbstractQueue`[2] class, which in itself implements a `Queue`[3] interface. Note that the names of the methods in that interface differ from the ones used for priority queues in the textbook. Also note that each of the queue methods exists in two version: a version returning a special value, and a version that throws an exception.

Like with any queue implementation utilizing a binary heap data structure, the performance of insertions and removals is going to be $O(\log n)$, while examining the value will be done in constant time. While there are other data structures that are more efficient in theory, real-world performance of binary heaps is often superior, especially when no additional functionality is required (e.g., changing the keys).

In addition to the lecture notes, you might find the following resources useful:

Priority Queues chapter of the Algorithms textbook
https://algs4.cs.princeton.edu/24pq/

Implementation

Name your class `PriorityQueue2011` and put it in the *default package*.

Your implementation should use an array representation of the binary heap. In addition, you should not use the first position of the array, just like described in the link above. The array should also grow automatically, as needed (similar to how it's implemented in ArrayList, for example). Automatic *shrinking* is not required.

---

[1] https://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html

[2] https://docs.oracle.com/javase/8/docs/api/java/util/AbstractQueue.html

[3] https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html

Part 1

Implement the following public methods:

1. `boolean  offer(E e)`
2. `boolean  add(E e)`
3. `E  poll()`
4. `E  remove()`
5. `E  peek()`
6. `E  element()`
7. `int  size()`
8. `String  toString()` (described below)
9. `String  toTree()` (described below)

One public constructor should exist in your implementation: the one that takes no parameters and creates an empty queue when the class is instantiated. No other methods need to be implemented or overridden (e.g., the iterator() method is not required).

The behaviour of the methods in your implementation should be *equivalent* to that of Java Standard Library's classes (i.e., `java.util.PriorityQueue`; please refer to the class API online).

`toString()` should simply return a String with the contents of the array:

```
[null, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

The array looks sorted, as the items were inserted sequentially, from 1 to 12. When the same are inserted in the reverse order, the resulting array will be:

```
[null, 1, 3, 2, 6, 4, 7, 8, 12, 9, 10, 5, 11]
```

`toTree()` method should return a String with the tree representation of the heap, as follows, for the same two examples.

```
                1
        2               3
    4       5       6       7
  8   9  10  11  12


                1
        3               2
    6       4       7       8
 12   9  10   5  11
```

The following reference might be useful:
https://docs.oracle.com/javase/tutorial/essential/io/formatting.html

Of course, you are free to implement any private or protected methods and classes as you see fit. However, you should not have any public methods other than the ones mentioned (or the ones present in the interface or in the class' superclass).

Part 2

Name your class `PQSort` and put it in the *default package*. Inside, create the following two public methods:

```
static <E extends Comparable <? super E>> void heapSort(E[] a)
```

```
static <E extends Comparable <? super E>> void heapSort2011(E[] a)
```

Both of these methods should implement a heap sort algorithm using priority queues (not-in-place kind). The first method should rely on the `java.util.PriorityQueue` implementation, and the second one – on the priority queue you implemented in Part 1.

NB: These methods can literally be implemented in three lines each. Test the methods to ensure they work correctly (and produce the same result). The order is *non-decreasing*.

Part 3

Nothing needs to be submitted for this part.

Create some tester class and use it with your queue implementation to investigate its running time complexity as the number of items you insert in the queue increases. Try using values from 10 to 1,000,000 while measuring the running time. Confirm that the running time follows the expected logarithmic behaviour for both `add` and `remove` methods.

NOTES:

1. Do not use *package*-s in your project (put your classes in the `default` package). Using packages will cost you a 10 % deduction from the assignment mark.

2. Some aspects of your code will be marked automatically (e.g., how it handles boundary cases and error conditions), using a custom tester class. It is also imperative you test your classes. If any of the java files that you submit do not compile, the whole submission will be given a grade of zero, regardless of how trivial the compiler error is. The code you submit should compile using

```
javac *.java
```

command in either Windows, Linux, or MacOS.

3. Your code should include Javadoc comments.

4. Using any `java.util` implementations of `Collection`[4] or `Map`[5] is not allowed (importing the `java.util.AbstractQueue` interface is of course fine).

[4] https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html

[5] https://docs.oracle.com/javase/8/docs/api/java/util/Map.html

Submission

Find all the `java` files in your project and submit them electronically via eClass (do not zip, tar, rar, or 7z them). Only two files are expected, but you may write other classes, if you deem them necessary.

If working in a group, **make only one submission** per group and include a **`group.txt`** file containing the names and the student numbers of the group members. The deadline is firm. Contact your instructor *in advance* if you cannot meet the deadline explaining your circumstances.

Grading

The assignment will be graded using *the Common Grading Scheme for Undergraduate Faculties*[6]. We look at whether the code passes the unit tests, satisfies the requirements of this document, and whether it conforms to the code style rules.

Academic Honesty

Academic honesty will be strictly enforced in this course. Specifically, direct collaboration (e.g., sharing code or answers) is not permitted, and plagiarism detection software will be employed. You are, however, allowed to discuss the questions, ideas, or approaches you take.

Cite all sources you use (online sources – including web sites, old solutions, books, etc.) in your code comments; using textbook examples is allowed, but these must be cited as well.

E.g.,

```
1) /**
     * Constructs an empty list with the specified initial capacity.
     *
     * @param   initialCapacity   the initial capacity of the list
     * @exception IllegalArgumentException if the specified initial
capacity
     *            is negative
     * uses code from www.xxx.yyy.edu/123.html for initialization
     *
     */


2) //formula based on Jane Smart's thesis, page XX, available at
https://...
```

[6] https://secretariat-policies.info.yorku.ca/policies/common-grading-scheme-for-undergraduate-faculties/

```
a = b + c;
```

Although using outside sources is allowed – with proper citing, if the amount of non-original work is excessive, your grade may be reduced. You might find the following article useful to have an idea what kind of collaboration is appropriate: https://en.wikipedia.org/wiki/Clean_room_design

You may not post the contents of this assignment, nor solicit solutions on any external resources.