Assignment 2

Please read this document very carefully. Follow instructions exactly. If you have any questions please post them to MS Teams or ask during office hours.

This assignment is due Nov. 20th, by 11:59pm

In this assignment you are responsible for implementing a custom data structure called a "Priority List". The description of such a data-structure is given later in this document. You are given four .java files as starter code.

- `Customer.java`
- `MyQueue.java`
- `PriorityList.java`
- `PriorityNode.java`

Within the starter code, pay close attention to comments. Some comments say "Do note change this method/code". Changing this code will result in a loss of grades. You are responsible for submitting two .java files.

- `MyQueue.java`
- `PriorityList.java`

MyQueue

The class MyQueue is implemented using an array. Hence, the size of MyQueue is initially fixed. You will also notice that the queue is implemented in the same fashion we saw in lecture. That is, the queue views the array as circular and wraps around it as values are added/removed. You will be altering this class to make it dynamic. That is, there will be no fixed max number of customers in the queue. You are responsible for implementing two methods:

- `ResizeUp()`
- `ResizeDown()`

You should only change the code within these methods as well as complete a single line of code in remove() (see the comments in the .java file). Do not change the other methods or the implementation. You may add private "helper methods" if you wish. You have some freedom to implement these two methods as you see fit, however the following conditions must hold:

- The length of the array is never less than 10
- After resizeUp() is called, if there are n values in the queue, the length of the array must be greater than $(3/2)n$
- If there are n values in the queue, where $n > 5$, the length of the underlying array must be less than or equal to $4n$.

PriorityList

The PriorityList data-structure orders customers to be served, where different customers may have a higher priority than others. A customer's priority level is represented by an integer, where the higher the value of the integer the less that customer's priority is. For example, a customer with priority 5 has a higher priority than a customer with priority 10. The highest priority a customer can have is 1. Multiple customers can have the same priority level. If customers have the same priority level, they are served on a first come first serve basis.

For example, consider adding the following in the following order Customers to the PriorityList, where (id, p) denotes the customer's identification number and priority respectively:

(111, 5), (222, 1), (333, 4), (444, 1), (555, 4), (666, 1)

then the customers would be served in the following order:

(222, 1), (444, 1), (666, 1), (333, 4), (555, 4), (111, 5)

The PriorityList orders customers by representing each priority level as a node in a linked list. Each node of the linked list has a MyQueue which holds all the customers at that priority level. In the example above, the PriorityList would have 3 nodes total, one for each of priority level 1, 4, and 5. The first node (priority level 1) would have a MyQueue of length 3. You have four methods to implement (see below). Note, after your methods execute there should not exist a node with an empty queue; if a queue becomes empty, delete its corresponding node. You should always appropriately update numberOfCustomers;

- addCustomer(): Takes in a Customer and appropriately adds it to the PriorityList
- getNextCustomer(): Removes and returns the customer with the highest priority. You may assume this will only be called on a PriorityList with customers in it.
- deletePriorityLevel(): Takes in an integer k. Deletes the node with priority level k. If no such node exists, nothing is altered.
- truncateDownTo() Takes in an integer k. Deletes the lowest priority nodes in the PriorityList until there are less than or equal to k customers in the PriorityList.
- mergeLists(): Takes in a PriorityList and builds a new PriorityList based off the input and the object itself. The new PriorityList with have all the Customers the original two lists had and it will also maintain the priority levels. If the two original lists both have customers at the same priority level the method will interleave customers. For example, if the original object had customers at priority level 1 to be served in the order:

  (111, 1) then (221, 1) then (331, 1) then (441, 1)

  and the input PriorityList had customers in priority level 1 to be served in the order:

  (112, 1) then (222, 1)

  then the new merged list will have a priority level 1 which serves jobs in the following order:

  (111, 1) then (112, 1) then (221, 1) then (222, 1) then (331, 1) then (441, 1).

Submitting and Grading

This assignment will be submitted electronically via Avenue. This assignment is worth 15% of your final grade. Grading is done automatically. That is, a program calls your method, passes it certain arguments, and checks to see if it returns the expected output. The assignment is broken down as follows:

- resizeUp: 15%
- resizeDown: 15%
- addCustomer: 10%
- getNextCustomer: 5%
- deletePriorityLevel: 15%
- truncateDownTo: 20%
- mergeLists: 20%

Good luck!

Academic Dishonesty Disclaimer

All of the work you submit must be done by you, and your work must not be submitted by someone else. Plagiarism is academic fraud and is taken very seriously. The department uses software that compares programs for evidence of similar code.

Please don't copy. The TAs and I want you to succeed and are here to help. Here are a couple of general guidelines to help you avoid plagiarism:

Never look at another assignment solution, whether it is on paper or on the computer screen. Never show another student your assignment solution. This applies to all drafts of a solution and to incomplete solutions. If you find code on the web that solves part or all of an assignment, do not use or submit any part of it! A large percentage of the academic offenses in involve students who have never met, and who just happened to find the same solution online. If you find a solution, someone else will too.