Lab 10 – Heap-Based Priority Queues

Work with your newly assigned partner on this lab.

Preliminaries

Part of today's lab will focus on implementing a heap-based priority queue using an *ArrayList*-like class called *ZHExtensibleArrayStructure*. Recall that a heap is a binary tree (*not* a binary *search* tree) such that every element in the tree is less than (or in some implementations greater than) or equal to both its left and right child elements (if they exist). Every level of the tree is *complete* except possibly for the bottom level which is filled in from the left to the right with no gaps allowed. We could implement a heap using binary tree nodes, but it's actually easier to implement a heap using a linear structure.

Instead of using a raw array for the elements, we'll use the *ZHExtensibleArrayStructure* class, which is sort of a primitive *ArrayList*. It works like an array, but it supplies methods required by the *ZHCollection* interface along with other useful methods such as `get`, `set`, `findIndex`, `addLast` and `removeLast`; it also automatically expands the structure when necessary, so you don't have to check for the need to reallocate. In other words, using this class, we can concentrate on how the heap works and not on the details of how to use the array. Use the online documentation for the package to study the *ZHExtensibleArrayStructure* and *ZHArrayStructure* classes as well as the *ZHQueue* and *ZHCollection* interfaces:
http://www.users.csbsju.edu/~csweb/ZHStructures/index.html

Copy the lab directory into your *CS200/labs*. Keep the two test files in your lab directory but move file *<FOO>HeapPriorityQueue.java* into your structures directory and rename it with your initials in place of *<FOO>*. Open the three files and change all instances of *<FOO>*and *<foo>* to the appropriate initials. When done, close the test classes but keep your *<FOO>HeapPriorityQueue.java* file open.

For the 2nd and 4th parts of this lab, you will be modifying your bank simulation program from Lab 8 to use your own priority queue implementations. If you have not already done so, complete the simulation program from Lab 8, the bank simulation using priority queues. If neither you nor your current programming partner is close to finishing that lab, arrange to get a copy from another student who has it completed, but be sure that every file you copy from another person includes that person's name as author.

Problem 1: Building and testing your own heap-based priority queue

Complete the two constructors to create the `elements` instance variable using the suitable constructors from class *ZHExtensibleArrayStructure*. Throw proper exceptions when needed. Next, consider the `peek` method; the first element in the queue (if it exists) will always be at position 0. Have the `peek` method return the element at that position if available (without removing it). Throw any specified exceptions.

Complete the remaining methods except dequeue and enqueue. All these methods are essentially calls to the corresponding methods of the `elements` instance variable.

Start working toward implementing the dequeue and enqueue methods by developing helper methods for them. The first three helper methods need to be able to find parent and child positions in the heap. Use your class notes to determine formulas for the position relationships between parent and child nodes in the heap. Add the following three methods and use those formulas to complete the code for these methods:

```
        protected int parentPosition(int childPosition) { ... }
        protected int leftChildPosition(int parentPosition) { ... }
        protected int rightChildPosition(int parentPosition) { ... }
```

Next, add the `leastPosition` helper method that determines the position of the smallest element among *currentPosition*, left child of *currentPosition* (if it exists) and right child of *currentPosition* (if it exists). Note that if the left child doesn't exist then neither does the right child, and if the right child does exist then so does the left child since the heap is always a complete binary tree. (In this and the two following helper methods, you can assume that *currentPosition* will always be within the heap, since we wouldn't call it otherwise.)

```
        protected int leastPosition(int currentPosition) { ... }
```

Finally, add the following two *recursive methods* using the algorithms discussed in class:

```
        protected void checkUpward(int currentPosition) { ... }
        protected void checkDownward(int currentPosition) { ... }
```

Recall that the `checkUpward` method works as follows: if parameter `currentPosition` is zero (i.e., we're at root), it is done; otherwise, it compares the element at `currentPosition` with its parent element (if it exists). If the current element is less than the parent element, it swaps them and calls itself recursively with the parent position passed as the new value of `currentPosition`.

Method `checkDownward` is similar, but works from top to bottom: it first calls `leastPosition(currentPosition)`; if that call returns `currentPosition` it is done. Otherwise, it swaps the current element with the element at the position returned by `leastPostition` and calls itself recursively with that position as the new value of `currentPosition`.

Once you have these methods, you can complete the `enqueue` and `dequeue` methods. The `enqueue` method uses method `addLast` to append the new element to the end of the heap and method `checkUpward` to move the element into the correct position. Method `dequeue` is very similar but uses methods `removeLast`, `set` and `checkDownward` instead. You should be able to figure out the logic on your own.

Use the supplied *HeapPriorityQueueTest.java* test program to conduct your tests. Show your fully-commented class and test results to the lab instructor or TA.

Problem 2: Modifying the simulation program to use your own priority queue
Copy all files from your lab 08 folder into today's lab folder.

In this problem, you are asked to make all the necessary changes to your simulation in order to use your own priority queue implementation in place of Java's priority queue used earlier. Run the simulation a number of times to make sure that you're getting similar results as before. Make sure that customers with the same priority are served by arrival time.

Demo the working version of the simulation to the lab instructor or TA and explain all changes made.

Problem 3: Creating and testing a max-heap-based priority queue
Make a copy of your final *<FOO>HeapPriorityQueue.java* from problem 1 and rename it as *<FOO>**MAX**HeapPriorityQueue.java* using your initials in place of *FOO.* Make all necessary changes to class *<FOO>MAXHeapPriorityQueue* so that it orders elements from highest to lowest values. This is known as a *max-heap* as opposed to the *min-heap* you built earlier.

Use the *MAXHeapPriorityQueueTest.java* program to conduct your tests. Show your *fully commented* class and test results to the lab instructor or TA.

Problem 4: Modifying the simulation program again to use your max-heap priority queue

Create copies of your programs from problem 2 and rename them appropriately to distinguish them from those modified in problem 2. Make all the necessary changes to the new copies in order to replace your earlier min-heap-based priority queue with your new max-heap-based priority queue implementation.

Run the simulation a number of times to see how the results may have changed. Does your simulation now serve customers with higher priority values first? What about customers with the same priority, do you get served according to arrival time (earlier arrivals served first)?

Demo the working version of the simulation to the lab instructor or TA and explain all changes made.