CS310 Fall 2010

Data Structures

Programming Assignment #3
Extra Credit
Priority Queues

75 points

Due Date/Time:

NOTE: This is an optional assignment. You do not have to do it. If you decide not to submit this assignment, you will NOT be at any disadvantage as the course is not graded on the curve.

Because this assignment is optional and extra-credit, special rules apply:

- You must complete the project entirely on your own. You may **NOT** get any help, either from the instructor, or the TAs. The TAs have been instructed not to provide any assistance on this assignment. You may ask general questions, but we will NOT look at your code.
- Partial projects are not acceptable. You must submit both implementations, and both must compile and run, to receive any extra-credit.
- Late assignments (including the printout) will not be accepted.
- Extra-credit project will be subjected to extra scrutiny for any evidence of cheating. Cheating on this project will result in an "F" grade for the course, not just the assignment.

Your program is due on **Tuesday, November 12th.**

On the due date, you must place the two files you will write in your `handin/` directory, and submit hardcopy of the two Java source code files at the beginning of class on the same day.

The Program:

For this assignment, you will write two implementations of a **Priority Queue**. For this ADT, removal operations always return the object in the queue of highest priority that has been in the queue the longest. That is, no object of a given priority is ever removed as long as the queue contains one or more object of a higher priority. Within a given priority FIFO order must be preserved.
Your two implementations will be:

- Ordered Linked List. *This implementation must use an ordered linked list. Thus, insertion will be O(n) and removal will be O(1). .*
- Binary Heap, *This implementation must be an array-based Min Heap, which will give O(log n) insertion and removal. Additionally, you must insure that the structure is stable. That is, for items of identical priority, FIFO behavior must be preserved--the first item in must be the first removed. Hint: Use a wrapper class.*

Both implementations must have identical behavior, and must implement the PriorityQueue interface (provided). Both implementations must have two constructors, a default constructor with no arguments that uses the DEFAULT_MAX_CAPACITY constant from the PriorityQueue.java interface, and a constructor that takes a single integer parameter that represents the maximum capacity of the PQ. Although linked lists are theoretically never full, priority queues may be full. Thus you must enforce the max size for both implementations.

You are not required to implement any resizing strategies. If the user attempts to insert a new item into a full priority queue, you will abort the operation and return false. Also, note carefully the behavior specified for each method. Your classes must pattern this behavior exactly.

The PriorityQueue interface follows:

```
/*  The PriorityQueue ADT may store objects in any order.  However,
    removal of objects from the PQ must follow specific criteria.
    The object of highest priority that has been in the PQ longest
    must be the object returned by the remove() method.  FIFO return
    order must be preserved for objects of identical priority.

    Ranking of objects by priority is determined by the Comparable<E>
    interface.  All objects inserted into the PQ must implement this
    interface.  Priorities may be of any type that supports comparison;
    a lower priority value indicates a higher priority.  i.e. given two
    objects a and b, if a < b then a has higher priority than b.

*/

package data_structures;

import java.util.Iterator;

public interface PriorityQueue<E> extends Iterable<E> {
    public static final int DEFAULT_MAX_CAPACITY = 1000;

    //  Inserts a new object into the priority queue.  Returns true if
    //  the insertion is successful.  If the PQ is full, the insertion
    //  is aborted, and the method returns false.
    public boolean insert(E object);

    //  Removes the object of highest priority that has been in the
    //  PQ the longest, and returns it.  Returns null if the PQ is empty.
    public E remove();

    //  Returns the object of highest priority that has been in the
    //  PQ the longest, but does NOT remove it.
    //  Returns null if the PQ is empty.
    public E peek();

    //  Returns the number of objects currently in the PQ.
    public int size();

    //  Returns true if the element is in the priority queue, and
    //  false if it is not (or the pq is empty).
    public boolean contains(E object);

    //  Returns an iterator of the objects in the PQ, in no particular
    //  order.  The iterator must be fail-fast.
    public Iterator<E> iterator();
```

```
    //  Returns the PQ to an empty state.
    public void clear();

    //  Returns true if the PQ is empty, otherwise false
    public boolean isEmpty();

    //  Returns true if the PQ is full, otherwise false.
    public boolean isFull();

}
```

Thus, your project will consist of the following files. You must use exactly these filenames.

- `PriorityQueue.java` The ADT interface (provided)
- `ListPriorityQueue.java` The ordered linked list implementation.
- `BinaryHeapPriorityQueue.java` Array-based binary min-heap implementation.

Additional Details:

- Your project must consists of only the three files specified (including the provided interface), no additional source code files are permitted. (Do not hand in a copy of PriorityQueue.java, as it is provided to you).
- You will need to write driver/test programs to test your project. No sample drivers will be provided.
- Your code must be as efficient as possible. ie. a O(n) size() method is unacceptable.
- You may not make any modifications to the PriorityQueue interface provided. I will grade your project with my copy of this file.
- All source code files must have your **name and class account number** at the beginning of the file.
- All of the above classes must be in a package named `data_structures`.
- You may import `java.util.Iterator`, `java.util.NoSuchElementException`, and `ConcurrentModificationException` only. If you feel that you need to import anything else, let me know. You are expected to write all of the code yourself, and you may not use the Java API for any containers.
- Your code must not print anything.
- Your code should never crash, but must handle any/all error conditions gracefully. i.e. if the user attempts to call the clear() method on an empty PQ, or remove an item from an empty PQ, the program should not crash. Be sure to follow the specifications for all methods.
- You must write generic code according to the interface provided. You may not add any public methods to the implementations, but you may add private ones, or inner classes if needed.
- Your code may generate unchecked cast warnings when compiled, but it must compile and run correctly on rohan using the default JDK to receive any credit.

Turning in your project:

To submit your project, you must copy the two Java source code files into your `handin/` subdirectory. You will submit a printout of both files in class on the due date. As with the past two assignments, do not recreate the data_structures subdirectory in the handin subdirectory--just copy your files into the directory itself. Be sure to check the Program Submission Guidelines page.

Cheating Policy

There is a zero tolerance policy on cheating in this course. You are expected to complete all programming assignments on your own. Collaboration with other students in the course is not permitted. You may discuss ideas or solutions in general terms with other students, but you must not exchange code. During the grading process I will examine your code carefully. Anyone caught cheating on a programming assignment (or on an exam) will receive an "F" in the course, and a referral to Judicial Procedures. This applies to option, extra-credit assignment as well.