GIT Department of Computer Engineering
CSE 222/505
Spring 2014
Homework 06

Heap Implementation
Due date: May 5th 2014 17:00

In this homework, you will implement a GITPriorityQueue class that implements the PriorityQueue interface of the textbook.

TABLE 8.5
Methods of the `PriorityQueue<E>` Class

Method	Behavior
`boolean offer(E item)`	Inserts an item into the queue. Returns **`true`** if successful; returns **`false`** if the item could not be inserted.
`E remove()`	Removes the smallest entry and returns it if the queue is not empty. If the queue is empty, throws a `NoSuchElementException`.
`E poll()`	Removes the smallest entry and returns it. If the queue is empty, returns **`null`**.
`E peek()`	Returns the smallest entry without removing it. If the queue is empty, returns **`null`**.
`E element()`	Returns the the smallest entry without removing it. If the queue is empty, throws a `NoSuchElementException`.

Your GITPriorityQueue class will implement the exact same methods of the class KWPriorityQueue class.

TABLE 8.6
Design of KWPriorityQueue<E> Class

Data Field	Attribute
`ArrayList<E> theData`	An `ArrayList` to hold the data.
`Comparator<E> comparator`	An optional object that implements the `Comparator<E>` interface by providing a `compare` method.
Method	**Behavior**
`KWPriorityQueue()`	Constructs a heap-based priority queue that uses the elements' natural ordering.
`KWPriorityQueue (int cap, Comparator<E> comp)`	Constructs a heap-based priority queue with an initial capacity of `cap` and that uses the `compare` method of `Comparator comp` to determine the ordering of the elements.
`private int compare(E left, E right)`	Compares two objects and returns a negative number if object `left` is less than object `right`, zero if they are equal, and a positive number if object `left` is greater than object `right`.
`private void swap(int i, int j)`	Exchanges the object references in `theData` at indexes `i` and `j`.

However, your GITPriorityQueue class will not use the array based tree implementatition. Your class will extent the BinaryTree class of the textbook defined by

TABLE 8.1
Design of the BinaryTree<E> Class

Data Field	Attribute
`protected Node<E> root`	Reference to the root of the tree.
Constructor	**Behavior**
`public BinaryTree()`	Constructs an empty binary tree.
`protected BinaryTree(Node<E> root)`	Constructs a binary tree with the given node as the root.
`public BinaryTree(E data, BinaryTree<E> leftTree, BinaryTree<E> rightTree)`	Constructs a binary tree with the given data at the root and the two given subtrees.
Method	**Behavior**
`public BinaryTree<E> getLeftSubtree()`	Returns the left subtree.
`public BinaryTree<E> getRightSubtree()`	Returns the right subtree.
`public E getData()`	Returns the data in the root.
`public boolean isLeaf()`	Returns **`true`** if this tree is a leaf, **`false`** otherwise.
`public String toString()`	Returns a `String` representation of the tree.
`private void preOrderTraverse(Node<E> node, int depth, StringBuilder sb)`	Performs a preorder traversal of the subtree whose root is `node`. Appends the representation to the `StringBuilder`. Increments the value of `depth` (the current tree level).
`public static BinaryTree<E> readBinaryTree(BufferedReader bR)`	Constructs a binary tree by reading its data from stream `bR`.

In other words, you will use an inner Node class to keep your data.

Test your priority class with at least 3 different driver methods to show the correct results and include all your design documents (EA drawings) in your submission.