Program #5

Due: **12/8/18 at 10:00 PM**
Late Collections: **12/9/18 at 10:00PM**
12/10/18 at 10:00 PM
Program Files (3): **DoublyLinkedNode.java, DoublyLinkedList.java, PriorityQueue.java**

In this assignment we will explore an alternative to singly linked lists and use it to implement an alternative version of a priority queue. The first data structure you will be creating is a generic doubly linked list. This structure consists of a DoublyLinkedNode object that is linked to its previous and next neighbors in the list. Like a regular linked list, this structure can grow without the need to resize an array. However, unlike a typical linked list, you can move either forward or backward through the list.

DoublyLinkedNode<E>	
-element:E	The content of the node
-previous:DoublyLinkedNode<E>	A reference to the previous node in the list, null if it is the first Node
-next:DoublyLinkedNode<E>	A reference to the next node in the list, null if it is the last Node
+DoublyLinkedNode(element:E)	Sets the element field to the parameter and initializes the previous and next fields to null.
+getPrevious():DoublyLinkedNode<E>	Returns the previous node that this node is connected to.
+getNext():DoublyLinkedNode<E>	Returns the next node that this node is connected to.
+setPrevious(node: DoublyLinkedNode<E>): void	Sets the previous node for this node.
+setNext(node: DoublyLinkedNode<E>): void	Sets the next node for this node.
+getElement():E	Returns the content of this node.
+setElement(value:E):void	Sets the node's content.

DoublyLinkedList<E>	
-head:DoublyLinkedNode<E>	The first node in the list.
-tail:DoublyLinkedNode<E>	The last node in the list.
-size:int	The number of elements in the list.
+DoublyLinkedList()	Creates an empty DoublyLinkedList.
-getNode(index:int): DoublyLinkedNode<E>	Returns the node with the given **index**, where the first node has index=0. This node should be found by traversing as few nodes as possible. If the index is <= size/2, then it locates the element by moving from the head towards the tail, otherwise it moves from tail towards head.
+add(element:E):void	Adds a DoublyLinkedNode containing **element** to the end of the list.

+add(index: int, element:E):void	Inserts a DoublyLinkedNode at **index** containing element. The method should traverse as few nodes as possible to locate the insertion point. If the index is less than 0 or greater than the **size**, throws a java.lang.IndexOutOfBoundsException.
+get(index: int):E	Returns the value of the node at **index** in the list. This element should be found by traversing as few nodes as possible. If the index is invalid, throws an IndexOutOfBoundsException.
+indexOf(element:E):int	Returns the index of the first node containing "element", or -1 if the element is not in the list.
+remove(index:int):E	Removes the Node at the given **index** and return its **element**. The method should traverse as few nodes as possible to locate the node to be removed. If the index is invalid, throws an IndexOutOfBoundsException.
+remove(element:E):boolean	Removes the first Node that contains **element**. Returns true if the element was successfully removed, or false if no such element is in the list.
+size():int	Returns the number of Nodes in the list.

Once you have created the Doubly Linked List we are going to use that to build a Priority Queue. A Priority Queue is a collection of elements, each of which has a priority. When elements are removed, they are removed in order of priority, instead of the first-in, first-out policy of a typical queue. For our purposes, the element with highest priority is the least element as determined by the compareTo method (this is in contrast to the book, which assumes items with higher values have higher priority). The book provides an implementation of a PriorityQueue using a Heap. In this assignment, you will create one using a DoublyLinkedList.

PriorityQueue<E extends Comparable<E>>	
-elements:DoublyLinkedList	
+PriorityQueue()	Constructor that instantiates the elements list.
+enqueue(element:E):void	Adds an element to the priority queue. Elements should be added in sorted order to the list. Remember, we assume that the lowest element represents the one with highest priority, and this should be at the front of the list.
+peek():E	Returns the element at the front of the priority queue (the one with the highest priority) but does not remove it. If the queue is empty, returns null.

+dequeue():E	Returns the element at the front of the priority queue (the one with the highest priority) and removes it. If the queue is empty, returns null.
+clear():void	Removes all elements from the priority queue.
+size():int	Returns the number of elements in the priority queue.
+main(args:String[]):void	The main method. See below for details.

The main() method:
PriorityQueue needs to have a main method that will open a file that is passed in as a single command line argument. The program should terminate gracefully if an incorrect number of command line arguments are given, or if it is unable to open or read from the input file. Each line of this file will be treated as a String that should be added to the priority queue. After all the elements are added it should print out all the elements from the priority queue, one per line.

Comments:
At a minimum provide a comment that summarizes each class, and comments that explain what each method does. Include additional comments for sections of code that are especially complicated. At the top of each file, include a comment with the following form:

```
/*
CSE 17
Your name
Your Lehigh e-mail
Tutor: name (e-mail)   [only include if a tutor aided you]
Program #5      DEADLINE: December 8, 2018
Program Description: Doubly Linked List and Priority Queue
*/
```

Submission:
Make sure that your program precisely follows the UML specification, as we will write test programs that rely on this API. You should thoroughly test all methods, as the main method will only make use of some of the methods you have written. Once the program compiles, runs, and has been tested to your satisfaction, upload all of the **.java** files to Course Site. To do so, click on the name of the assignment in the Course Site page, and then press the "Add submission" button at the bottom of the next page. Drag and drop each file into the area under "File submissions". If necessary, you can update your submission at any time before the deadline passes.

Sample Output:
Assume that the input file **list.txt** has the following contents:

```
Adam
Zach
Mark
Dave
Jennifer
Sarah
Tiffany
Abigail
```

Then the output of the following command would be:

```
> java PriorityQueue list.txt
Abigail
Adam
Dave
Jennifer
Mark
Sarah
Tiffany
Zach
```