**CSCI 3302 Programming Assignment 04 (100 Points)**

**Due: Nov 6, 8:00 AM**

GITHUB Link: [Program 04](https://classroom.github.com/a/CJkuBqTB)

Objectives:

* Demonstrate how to define and implement an abstract data type.
* Demonstrate implementation of a Java interface.
* Implement a reference-based ADT Priority Queue.
* Demonstrate Java programming proficiency for a linked list.
* Demonstrate proficiency with Java generics.

Assignment Assistance:

* This homework assignment is due prior to the date and time specified above.
* This assignment is restricted to individual effort. You may not receive help from any other person except the instructor or the AARC (help from the AARC must be well documented!).
* Any resource used (other than Dr. Becnel or the course text) must be documented in the code (as comments) detailing the source and describing exactly what was learned and how that information was used. Submissions will be severely penalized if copied in part or in whole from any source.
* If you need help, visit your instructor during his posted office hours. If your schedule cannot accommodate any of these times, then email your instructor to schedule a different time.

Problem Description:

1. A priority queue is an abstract data type that is like a regular queue or stack data structure, but where each element has a "priority" associated with it. In a priority queue, an element with high priority is served before an element with low priority.
2. In this assignment, you will implement a class called PriorityQueue<T> from the given interface, IPriorityQueue<T>. The interface is defined as follows:

**public** **interface** IPriorityQueue<T> {

**public** **boolean** isEmpty();

**public** **void** add(**int** priority, T item);

**public** T removeTopPriority() **throws** PQException;

**public** T peekTopPriority() **throws** PQException;

}

1. The PriorityQueue<T> class will have a single private instance attribute called front which is a Node<T> object and references the first Node<T> in the linked list used to store the elements of the priority queue. No other attributes are needed in the class.
2. The Node<T> class is used to build the linked-list collection of objects stored in the PriorityQueue<T>. The code for the Node<T> class can be found at the end of this document.
   1. Notice that this version of the Node<T> class stores items of type T. Also, it contains an additional instance attribute, an itemPriority, which is a field of type int representing the priority of the associated object. **In this implementation, the lower the integer value of itemPriority, the higher the priority.**
   2. Your implementation of PriorityQueue should not rely on any modifications to the Node<T> class; that is, feel free to modify Node<T> to aid with your testing, but your submission should work without the modifications.
3. The methods that you are required to implement should, for the most part, be self-explanatory.
   1. The isEmpty method returns true if the priority queue is empty and false otherwise.
   2. The add method adds a new item to the collection.
   3. The removeTopPriority method should remove the Node<T> with the highest priority from the linked list and returns its associated item. If more than a single Node<T> has the same priority, then the removed Node<T> should be the one that has been in the priority queue for the longest amount of time (First In, First Out). The implementation of these two methods is dependent on each other.
      * One way to implement these two methods is to add the new Node<T> object in the appropriate location within the linked list based on its priority (from highest priority to lowest priority). Then, to removeTopPriority, simply remove the Node<T> object at the front of the linked list and return its corresponding item.
      * The other way is to add the new Node<T> at the front of the linked list. Then, to removeTopPriority, search through the linked list, find the Node<T> object with the highest priority (that is also the furthest back within the list), remove it, and return its corresponding item.
      * Likewise, the implementation of the peekTopPriority method would depend on the implementation of the add method.
   4. The peekTopPriority method should return the corresponding item of the Node<T> object with the highest priority without removing the Node<T> object from the linked list.
   5. To make sure that your implementation is working correctly, you will need to create a PQException class as an extension of a RuntimeException. This exception needs to be thrown if an attempt is made to remove from or peek at an empty priority queue.
   6. Your program should work in either a Windows environment or a Linux environment.
   7. You may write any private helper methods if needed.
4. For testing and debugging purposes, it is always good to have a toString() method for your classes. Here is some code for one that tells you basic information about the items and their priorities.

A screen shot of a computer code

Description automatically generated with low confidence

1. Your program should work in the GitHub codespace (Linux environment) and locally (Windows environment).

Submission:

* Review the Evaluation below to ensure you have met all the requirements.
* Commit electronic copy of PriorityQueue.java, IPriorityQueue.java, Node.java, PQException.java to GitHub. Upload a backup copy to D2L. You may have testing files in your repository, but they will not be considered for grading.

Evaluation

* 1. Project is late or not submitted at all. -100
  2. Project does not compile. -40
  3. Method signatures do not match the specification. -30
  4. Calculations/output are incorrect. -10 per method
  5. Code is not well organized or properly indented. -5
  6. Code is inadequately commented for readability. -5
  7. Code does not contain the student’s name, course section, -5

and date of submission.

* 1. Code is not submitted to the Github -15

Code for Node<T> Class:

* Your implementation should utilize the following node class to store all values in a linked-list:

**public** **class** Node<T> {

**private** **int** itemPriority;

**private** T item;

**private** Node<T> next;

**public** Node(**int** priority, T newItem) {

this.itemPriority = priority;

this.item = newItem;

this.next = **null**;

}

**public** **int** getPriority() {

**return** this.itemPriority;

}

**public** T getItem() {

**return** this.item;

}

**public** Node<T> getNext() {

**return** this.next;

}

**public** **void** setNext(Node<T> nextNode) {

this.next = nextNode;

}

}