```
##########################################

Student Name = Jitong Ding

##########################################



CSE017 Grading sheet for Jitong Ding

Homework Assignment PriorityQueue

Total points                          maximum: 100

Completeness: All class/methods included (40)   [ 40 ]

Compilation: Program compiles (20)               [ 20 ]

Execution: Program executes properly (30)        [ 30 ]

Style: Program obeys style rules (10)            [ 7  ]

Subtotal                                           97

Late Penalty                                     [ 0  ]

Total Points                                       97


##########################################
##########################################
```

```
/*
CSE 17
Jitong Ding
jid221
Program #5 DEADLINE: December 8, 2018
Program Description: Doubly Linked List and Priority Queue
*/

import java.lang.IndexOutOfBoundsException;

public class DoublyLinkedList<E>{

  /** Private data filed */
  private DoublyLinkedNode<E> head;
  private DoublyLinkedNode<E> tail;
  private int size;

  /** Construct a new empty DoublyLinkedList. */
  public DoublyLinkedList(){
    head = null;
    tail = null;
    size = 0;
  }

  /** A method to return the node with the given index*/
  private DoublyLinkedNode<E> getNode(int index){
    if(index <= size/2){
      int i = 0;
      DoublyLinkedNode<E> current = head;
      while(i < index){
        current = current.getNext();
        ++i;
      }
      return current;
    }
    else if(index > size/2 && index < size){
      int j = size-1;
      DoublyLinkedNode<E> current = tail;
      while(j > index){
        current = current.getPrevious();
        j--;
      }
      return current;
    }
    return null;
  }

  /** A method to add a DoublyLinkedNode containing element to the end of the list.*/
  public void add(E element){
    DoublyLinkedNode<E> newNode = new DoublyLinkedNode<E>(element);  /** Create a new DoublyLinkedNode<E> with element.*/
    if(tail == null){
      head = tail = newNode;  /** The element is the only one in the list.*/
    }
    else{
      tail.setNext(newNode);    /** Link the new node with the last node.*/
      newNode.setPrevious(tail);
      tail = tail.getNext(); /** Tail now points to the last node.*/

    }
    size++;
  }

  /** A method to insert a DoublyLinkedNode at index containing element.*/
  public void add(int index, E element){
    if(index < 0 || index > size){
      throw new IndexOutOfBoundsException();
    }
    else{
      if(index == size){
        add(element);
      }
```

Missing javadoc comments above classes. This should explain what each class is

-1 style

add(E) and remove(int) don't call getNode()

-2 style


```
      else if(index <= size/2){
        int m = 0;
        DoublyLinkedNode<E> current = head;
        while(m < index){
          current = current.getNext();
          ++m;
        }
        DoublyLinkedNode<E> temp = current;
        current = new DoublyLinkedNode<E>(element);
        if(index == 0){
          current.setNext(temp);
          temp.setPrevious(current);
          head = current;
        }
        else{
          current.setNext(temp);
          current.setPrevious(temp.getPrevious());
          temp.setPrevious(current);
          current.getPrevious().setNext(current);
        }
        size++;
      }
      else if(index > size/2){
        int n = size-1;
        DoublyLinkedNode<E> current = tail;
        while(n > index){
          current = current.getPrevious();
          --n;
        }
        DoublyLinkedNode<E> temp = current;
        current = new DoublyLinkedNode<E>(element);
        current.setNext(temp);/** Connect the new node's next with temp.*/
        current.setPrevious(temp.getPrevious());/** Connect the new node's previous with temp's previous.*/
        temp.setPrevious(current);/** Connect the temp with the new node .*/
        current.getPrevious().setNext(current);
        size++;
      }
    }
  }

  /** A method to return the value of the node at index in the list.*/
  public E get(int index){
    return getNode(index).getElement();
  }

  /** A method to return the index of the first node containing element*/
  public int indexOf(E element){
    DoublyLinkedNode<E> current = head;
    for(int m = 0; m < size; ++m){
      if(current.getElement().equals(element)){
        return m;
      }
      current = current.getNext();
    }
    return -1;
  }

  /** A method to remove the Node at the given index and return its element.*/
  public E remove(int index){
    if(index < 0 || index > size){
      throw new IndexOutOfBoundsException();
    }
    else{
      if(index <= size/2){
        int m = 0;
        DoublyLinkedNode<E> current = head;
        while(m < index){
          current = current.getNext();
          ++m;
        }
        DoublyLinkedNode<E> temp = current;
        if(index == 0){
```

```
            head = current.getNext();
          }
          else{
            temp.getPrevious().setNext(temp.getNext());
            temp.getNext().setPrevious(temp.getPrevious());
          }
          size--;
          return temp.getElement();
        }
        else{
          int n = size -1;
          DoublyLinkedNode<E> current = tail;
          while(n > index){
            current = current.getPrevious();
            n--;
          }
          DoublyLinkedNode<E> temp = current;
          if(index == size-1){
            tail = temp.getPrevious();
          }
          else{
            temp.getPrevious().setNext(temp.getNext());
            temp.getNext().setPrevious(temp.getPrevious());
          }
          size--;
          return temp.getElement();
        }
      }
    }

    /** A method to remove the first Node that contains element.*/
    public boolean remove(E element){
      DoublyLinkedNode<E> current = head;
      for(int i = 0; i< size; ++i){
        if(current.getElement().equals(element)){
          DoublyLinkedNode<E> temp = current;
          if(i == 0){
            head = current.getNext();
          }
          else if(i == size-1){
            tail = temp.getPrevious();
          }
          else{
            temp.getPrevious().setNext(temp.getNext());
            temp.getNext().setPrevious(temp.getPrevious());
          }
          size--;
          return true;
        }
        current = current.getNext();
      }
      return false;
    }

    /** A method to return the DoublyLinkedList's size*/
    public int size(){
      return size;
    }
  }
```

```
/*
CSE 17
Jitong Ding
jid221
Program #5 DEADLINE: December 8, 2018
Program Description: Doubly Linked List and Priority Queue
*/
public class DoublyLinkedNode<E>{

  /** Private data filed */
  private E element;
  private DoublyLinkedNode<E> previous;
  private DoublyLinkedNode<E> next;

  /** Construct a new DoublyLinkedNode with element and initialize the previous
and next fields to null. */
  public DoublyLinkedNode(E element){
    this.element = element;
    previous = null;
    next = null;
  }

  /** A method to return previous*/
  public DoublyLinkedNode<E> getPrevious(){
    return previous;
  }

  /** A method to return next*/
  public DoublyLinkedNode<E> getNext(){
    return next;
  }

  /** A method to set the previous node for this node.*/
  public void setPrevious(DoublyLinkedNode<E> node){
    this.previous = node;
  }

  /** A method to set the next node for this node.*/
  public void setNext(DoublyLinkedNode<E> node){
    this.next = node;
  }

  /** A method to return element.*/
  public E getElement(){
    return element;
  }

  /** A method to set the nodes content.*/
  public void setElement(E value){
    this.element = value;
  }
}
```

```
/*
CSE 17
Jitong Ding
jid221
Program #5 DEADLINE: December 8, 2018
Program Description: Doubly Linked List and Priority Queue
*/

import java.io.File;
import java.util.Scanner;
import java.io.FileNotFoundException;

public class PriorityQueue<E extends Comparable<E>>{
  /** Private data filed */
  private DoublyLinkedList<E> elements;

  /** Construct a new PriorityQueue. */
  public PriorityQueue(){
    elements = new DoublyLinkedList<E>();
  }

  /** A method to add elements in sorted order to the list.*/
  public void enqueue(E element){
    int size = elements.size();
    if(size == 0){   /** If the queue is empty, just add the element.*/
      elements.add(element);
    }
    else if(element.compareTo(elements.get(size-1)) > 0){  /** If the element is bigger than thelast element from the list and just add the element to the last. */
      elements.add(element);
    }
    else{
      for(int m = 0; m < size; ++m){
        if(element.compareTo(elements.get(m)) < 0){
          elements.add(m,element);
          break;
        }
      }
    }
  }

  /** A method to return the element at the front of the priority queue (the one with the highest priority) but does not remove it..*/
  public E peek(){
    if(elements.size()==0){  /** If the queue is empty, returns null.*/
      return null;
    }
    else{
      return elements.get(0);
    }
  }

  /** Returns the element at the front of the priority queue (the one with the highest priority) and removes it.*/
  public E dequeue(){
    if(elements.size() == 0){  /** If the queue is empty, returns null.*/
      return null;
    }
    else{
      return elements.remove(0);
    }
  }

  /** A method to remove all elements from the priority queue.*/
  public void clear(){
    int length = elements.size();
    for(int i = 0; i< length; ++i){
      elements.remove(0);
    }
  }

  /** A method to return the number of elements in the priority queuet.*/
  public int size(){
    return elements.size();
  }

  /** The main method.*/
  public static void main(String[] args){
    if(args.length != 1){ /** Check whether there is only one command line argument.*/
      System.out.println("Usage: Please enter one command line argument");
      System.out.println("Usage: java PriorityQueue filename");
      System.exit(0);
    }

    File file = new File(args[0]);
    PriorityQueue<String> queue = new PriorityQueue<>();
    if(!file.canRead()){ /** Check the file can be opened and read.*/
      System.out.println("Usage: Please enter a vaild file");
      System.out.println("Usage: java PriorityQueue filename");
      System.exit(0);
    }

    try{    /** Try-catch block to deal with the FileNotFoundException*/
      Scanner scan = new Scanner(file);
      while(scan.hasNextLine()){
        String line = scan.next();
        queue.enqueue(line);
      }
    }
    catch(FileNotFoundException ex){
      System.out.println("Usage: Please enter a vaild file");
      System.out.println("Usage: java PriorityQueue filename");
      System.exit(0);
    }

    for(int m = 0; m < queue.elements.size(); ++m){
      System.out.println(queue.elements.get(m));
    }
  }
}
```

```
##########################################
##########################################

############ Compiled Result ###############

Source Code Compilation:

Test Case Compilation:



##########################################

##########################################
############ Execution Result ##############
##########################################

##########################################
Test1 output - testOutput1.txt

Command: java PriorityQueue input1.txt >> testOutput1.txt 2>&1

Abigail
Adam
Dave
Jennifer
Mark
Sarah
Tiffany
Zach

##########################################
Test2 output - testOutput2.txt

Command: java PriorityQueue input2.txt >> testOutput2.txt 2>&1

a
b
c
d
e
f
g
h
j
k
l
m
n
o
p
q
r
s
t
u
v
w
```

Output is good


```
x
y
z


##########################################
Test3 output - testOutput3.txt


Command: java PriorityQueue list.txt >> testOutput3.txt 2>&1
This test case is used to test the command-line arguments.

Usage: Please enter one command line argument
Usage: java PriorityQueue filename


##########################################
###########Test Code Result################
##########################################
Test4 output - testOutput4.txt


Command: java LinkedListTest >> testOutput4.txt 2>&1

add() tests...
1.a) Add A,B,C: A B C
1.b) Add D at 3: A B C D
1.c) Add S at 1: A S B C D
1.d) Add T at 4: A S B C T D
1.e) Add X to front: X A S B C T D

indexOf() tests...
2.a) Index of A: 1 [should be 1]
2.b) Index of X: 0 [should be 0]
2.c) Index of D: 6 [should be 6]

remove() tests...
3.a) Removed S: X A B C T D
3.b) Removed index=4 (T): X A B C D
3.c) Removed index=0 (X): A B C D
3.d) Removed remaining items:

4) Adding to an Integer list using add(int,E)...
Expected:       1 2 3
Result:         1 2 3
```