Lab 8

CSE 274

In this lab we develop a priority queue data structure which can queue objects of the class `Packet`:

```
class Packet
{
      public String Data;
      public int Priority;

      Packet(){}
}
```

I. Implementing Priority Queues by Arrays

The `PriorityQueue` class stores packets in an `InternalArray`:

```
class PriorityQueue
{
      int number=0;
      private int maxSize;
      private Packet[] InternalArray;

      PriorityQueue(int size)
      {
            maxSize=size;
            InternalArray=new Packet[maxSize];

            for (int j=0;j<maxSize;j++)
                  InternalArray[j]=new Packet();
      }

      public String deQueue()
      {
            String temp= InternalArray[number-1].Data;
            number=number-1;
            return temp;
      }

      public void displayQueue()
      {
            for (int j=0;j<number;j++)
                  System.out.println(InternalArray[j].Data);
      }

      public boolean isEmpty()
      {return number==0;}
}
```

As it can be seen above, each entry of `InternalArray` is a reference to an object of `Packet` class. In `PriorityQueue` class, `maxSize` is the size of `InternalArray`. Also, `number` is the number of non-empty packets in the `InternalArray`. In fact, the packets at `index=number` to `index=maxSize` are considered empty. The constructor of the class initializes the size of `InternalArray` and allocates memory to objects.

II. The enQueue Method

Develop the following method for the `PriorityQueue` class:

```
public void enQueue(String newData, int newPriority)
```

- The method receives a `newData` and a `newPriority`.
- The method searches for a place to insert `newData` and `newPriority` in the `InternalArray`. The search starts at the first index of the `InternalArray`.
- The search stops at an index `j` if we have `newPriority<=InternalArray[j].Priroty`. If this condition is never satisfied, the search stops at index `j=number-1`, where `number` is the number of non-empty packets that are already in the `InternalArray`.
- The method creates an empty space at index `j` of the `InternalArray` to insert `newData` and `newPriority` at index `j`. Toward this goal, the method makes a one-entry forward shift to all the entires from `index=j` to `index=number-1` of the `InternalArray`.
- The method performs actual insertion of `newData` and `newPriority` at index `j` of the `InternalArray` by running the following commands:

  ```
  InternalArray[j].Data=newData;
  InternalArray[j].Priority=newPriority;
  ```

- The method increases `number` by one.

Test the developed method using the below lines of code:

```
PriorityQueue myQueue = new PriorityQueue(10);
PriorityQueueLinkedList();

myQueue.enQueue("A", 1);
myQueue.enQueue("B", 3);
myQueue.enQueue("C", 2);
myQueue.enQueue("D", 5);
myQueue.enQueue("E",4);

while (!myQueue.isEmpty())
      System.out.println(myQueue.deQueue());
```

The expected output is printed below:

```
D
E
B
C
A
```

III. Implementing Priority Queues using Linked Lists

The limitation of the `PriorityQueue` class that we developed in previous sections is that the priority queues created by the class can have to up `maxSize` packets. To remove this limitation, in this section we implement priority queues using linked lists. We add new variables to the `Packet` class so that objects of these class can be linked to each other and create a doubly linked list:

```
class Packet
{
      public String Data;
      public int Priority;
      Packet next;
      Packet previous;

      Packet(){}
}
```

Please notice that the `PriorityQueue` class of the previous section can still use the edited `Packet` class. Nevertheless, the code for the `PriorityQueueLinkedList` is below:

```
class PriorityQueueLinkedList
{
      private Packet first;
      private Packet last;

      PriorityQueueLinkedList(){}
}
```

The `PriorityQueueLinkedList` has the following `deQueue` method:

```
public String deQueue()
{
      Packet temp=first;

      first=first.next;

      if (first==null)
      {
            last=null;
            return temp.Data;
      }
      first.previous=null;

      return temp.Data;
}
```

The `PriorityQueueLinkedList` has the following `isEmpty` method:

```
public boolean isEmpty()
{
      return first==null;
}
```

The `PriorityQueueLinkedList` has the following `displayQueue` method:

```
public void displayQueue()
{
      Packet current = first;

      System.out.print("Data: ");
      while (current != null) {
            System.out.print(String.format("%10s", current.Data));
            current = current.next;
      }

      System.out.println("\n");
      current = first;

      System.out.print("Next: ");
      while (current != null) {

            if (current.next != null)
                  System.out.print(String.format("%10s", current.
                     next.Data));
            else
                  System.out.print(String.format("%10s", current.
                     next));

            current = current.next;
      }

      System.out.println("\n");
      current = first;

      System.out.print("Previous:");
      while (current != null) {
            if (current.previous != null)
                  System.out.print(String.format("%10s", current.
                     previous.Data));
            else
                  System.out.print(String.format("%10s", current.
                     previous));

            current = current.next;
      }

      System.out.println("\n");

}
```

IV. The enQueue method

Develop the following method for `PriorityQueueLinkedList` class:

```
public void enQueue(String newData, int newPriority)
```

For developing the above method, use the below logic/pseudo-code:

```
Packet newPacket=new Packet()
newPacket.Data=newData
newPacket.Priority=newPriority

If the linked list of packets is empty:

      Add the newPacket as the first link of the linked list

      return

If newPacket.Priority <=last.Priority:

      add the newPacket as the last link of the linked list

      return

Packet current=last
while current is not null:

      if (newPriority <=current.Priority):
            break

      current=current.previous

if current is null:

      Add the newPacket as the first link of the linked list

      return

Add the newPacket as a new link right before "current", so that
   newPacket would be closer to "first" link of the linked list.

return
```

Use the following lines of code to test the developed method:

```
PriorityQueueLinkedList myQueue = new PriorityQueueLinkedList();

myQueue.enQueue("A", 1);
myQueue.enQueue("B", 3);
myQueue.enQueue("C", 2);
myQueue.enQueue("D", 5);
myQueue.enQueue("E",4);

myQueue.displayQueue();
```

The expected output is printed below:

```
Data:         D   E   B   C   A

Next:         E   B   C   A   null

Previous:  null   D   E   B   C
```

V. Submitting the Assignment

When submitting your response to the assignment, keep the above lines of code into the body of the `main` method.