CSC 310 Data Structures

Homework 4: Queues and Linked Lists

CSC310_Hw4_1

```
class MyCircularDeque{
    int[] deque;                        //array to store
    int f;                              //the front spot
    int r;                              //the rear spot
    int max;                            //the max size of the queue
    int size;                           //the size of queue

    //constructor
    MyCircularDeque(int k){
        deque = new int[k];             //sets the array size
        f=0;                            //sets front to 0
        r=0;                            //sets rear to 0
        max = k;                        //sets the max to the array's size
        size =0;                        // keeps track of the the size of queue
    }


   //inserts element in the front of the queue
   public boolean insertFront(int n){
       if(isFull())             //checks if it is full and returns that op is false
           return false;
       try{
           if(f==0)             //checks if the number is either the start or end of array
               cycleNum(f);     //changes number is so
           else
               f-=1;
           deque[f]=n;          //sets the new element
           size++;              //increases size
           return true;
       }catch(Exception e){     //catches exceptions and return a fail
           return false;
       }
   }

   //inserts element in the rears of the queue
   public boolean insertLast(int n){
       if(isFull())             //checks if it is full and returns that op is false
           return false;
       try{
           deque[r]=n;          //sets the new element
           if(r==max-1)         //checks if the number is either the start or end of array
               cycleNum(r);     //changes number is so
           else
               r+=1;
           size++;              //increases size
           return true;
       }catch(Exception e){     //catches exceptions and return a fail
           return false;
       }
   }
```

This is a class for circular queues. I needed a wat to store the data so I chose to use an array. I also needed to keep up with the front, rear, the max size, and size. The constructor is need to set the initial values.

For the insert for the front and rear I had to check if its full or not. To catch exceptions I use the try and catch to catch the fails of the operations. I checked if the number would have to cycle to the other end of the array for the front I had to change the front number first before setting the value while the last is the opposite. If it goes through it returns true.

```
   //deletes the front element (just moves the f var)
   public boolean deleteFront(){
       if(isEmpty())            //checks if is empty and send fail if it is
           return false;
       try{
           if(f==max-1)         //checks if need to cycle if so it does
               cycleNum(r);
           else
               f+=1;
           size--;              //decease size
           return true;
       }catch(Exception e){     //catches fails
           return false;
       }
   }

   //deletes the front element (just moves the f var)
   public boolean deleteLast(){
       if(isEmpty())            //checks if is empty and send fail if it is
           return false;
       try{
           if(isFull()){        //checks if it is full for special case of full
               size--;          //decreease size
               return true;     //sends a success
           }
           if(r==0)             //checks if it need to cycle and does so
               cycleNum(r);
           else
               r-=1;
           size--;              //reduce size to match
           return true;
       }catch(Exception e){     //catches fail
           return false;
       }
   }
```

The delete I had to check if it is empty to be able to delete and if it is it returns a false. I used a try and catch to catch exceptions and it returns a false. For the last you have to check if the array is full to see if it just has to change the size. Both of them it has to check if it needs to be cycle the numbers and change the var and reduce the size.

```
89      //gets the first element of queue
90      public int getFront(){
91          if(isEmpty())
92              return -1;
93          return deque[f];
94      }
95
96      //gets the last element of queue
97      public int getRear(){
98          if(isEmpty())
99              return -1;
100         if(r==0)
101             return deque[max-1];
102         return deque[r-1];
103     }
104
105     //checks if queue is empty
106     public boolean isEmpty(){
108         if(size==0)
109             return true;
110         return false;
111     }
112
113     //checks if the queue is full
115     public boolean isFull(){
116         if(max==size)
117             return true;
118         return false;
119     }
```

The get functions must check if the que is empty and returns a -1 and returns the value in the array the rear has to be r-1 to get the actual rear value. For the empty and full it returns the size compared to 0 and the max of the array.

For the class to work with an array it has to have a way to switch from 0 to max and back.

```
119     //cycles number if it is the max value of array or at 0
120     private void cycleNum(int num){
121         if(num == f){           //for the front
122             if(f == 0)
123                 f=max-1;
124             else
125                 f=max-1;
126         }else{                  //for rear
127             if(r == 0)
128                 r=max-1;
129             else
130                 r=max-1;
131         }
132     }
```

The example code

```
136     public static void main(String[] args) {
137         ///*
138         MyCircularDeque cd = new MyCircularDeque(3);
139         System.out.println(cd.insertLast(1));       //true
140         System.out.println(cd.insertLast(2));       //true
141         System.out.println(cd.insertFront(3));      //true
142         System.out.println(cd.insertFront(4));      //false
143         System.out.println(cd.getRear());           //2
144         System.out.println(cd.isFull());            //true
145         System.out.println(cd.deleteLast());        //true
146         System.out.println(cd.insertFront(4));      //true
147         System.out.println(cd.getFront());          //4
148         //*/
149         /*
150         MyCircularDeque cd = new MyCircularDeque(3);
151         System.out.println(cd.insertLast(1));       //true
152         System.out.println(cd.insertLast(2));       //true
153         System.out.println(cd.insertFront(3));      //true
154         System.out.println(cd.insertFront(4));      //false
155         System.out.println(cd.getRear());           //2
156         System.out.println(cd.isFull());            //true
157         System.out.println(cd.deleteFront());       //true
158         System.out.println(cd.insertFront(4));      //true
159         System.out.println(cd.getFront());          //4
160         */
```

```
tput - CSC310_Hw4_1 (run)
run:
true
true
true
false
2
true
true
true
4
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
161         /*
162         MyCircularDeque cd = new MyCircularDeque(3);
163         System.out.println(cd.insertLast(1));       //true
164         System.out.println(cd.insertLast(2));       //true
165         System.out.println(cd.insertFront(3));      //true
166         System.out.println(cd.deleteFront());       //true
167         System.out.println(cd.deleteFront());       //true
168         System.out.println(cd.deleteFront());       //true
169         System.out.println(cd.deleteFront());       //false
170         */
171         /*
172         MyCircularDeque cd = new MyCircularDeque(3);
173         System.out.println(cd.deleteFront());       //false
174         System.out.println(cd.deleteLast());        //false
175         */
176     }
177 }
```

CSC310_Hw4_2

```java
import java.util.*;

class LinkedQueue{
    private Node head;              //Keeps track of the head of the list
    private Node tail;              //Keeps track of the tail of the list
    private int size;               //Keeps track of the size of the list

    /////////Nested Node Class/////////
    private class Node{             //Keeps the element and the next elements location
        private int element;
        private Node next;

        Node(int e, Node n){        //sets the nodes element and next one
            element=e;
            next=n;
        }
    }

    //constructor
    public LinkedQueue(){
        head=null;                  //sets head to a null
        tail=null;                  //sets tail to a null
        size=0;                     //sets size to 0
    }
```

For this program it needs a class for linked list and a nested node for the linked list to keep track of the elements and next node. The linkedQueue needs a constructor to set the values to start as null.

```java
    public int len(){               //returns the size
        return size;
    }

    public boolean isEmpty(){       //returns if list is empty
        return size==0;
    }

    public int first(){             //returns the fist element w/o removing
        if(isEmpty())
            throw new NoSuchElementException();

        return head.element;
    }

    public int last(){              //returns the last element w/o removing
        if(isEmpty())
            throw new NoSuchElementException();

        return tail.element;
    }
```

the assignment required the len function and the size that I kept track of is easy to return. Need to check if the list is empty or not. Must test if the list is empty first before returning the first element. The last is the same but for the tail node.

```java
    public int dequeue(){           //returns the first element while removing
        if(isEmpty())
            throw new NoSuchElementException();

        int out = head.element;     //stores element so it can be removed
        head = head.next;           //sets head as next node
        size--1;                    //reduce size

        if(isEmpty())               //makes sure if the list is empty that the tell is null
            tail=null;

        return out;
    }

    public void enqueue(int e){     //places element in the new tail postion
        Node newN=new Node(e,null);

        if(isEmpty())
            head=newN;
        else
            tail.next=newN;
        tail=newN;
        size+=1;
    }
```

for both of the functions you have to check if the list is empty for the deque to throw exception. And for the enqueue to set the new node for the head else set it tell.

```java
//function to merges two lists into new lists
public LinkedQueue mergeQueues(LinkedQueue a, LinkedQueue b){
    LinkedQueue c = new LinkedQueue();      //creates a new list

    //keeps the merge goint until the lists are empty
    while((!a.isEmpty())||(!b.isEmpty())){
        if((!a.isEmpty()) && (!b.isEmpty()))//for checking which one is lower
            if(a.first()<= b.first())       //for when the first a is less then b first
                c.enqueue(a.dequeue());
            else                           //for when its not
                c.enqueue(b.dequeue());
        else if(a.isEmpty())               //for when or or the other is more
            c.enqueue(b.dequeue());
        else
            c.enqueue(a.dequeue());
    }
    return c;
}
}
```

For this assignment it needed a merge. For the merge theyboth have to be not empty at first then I have compare the first of each and when one is empty you have to just put the rest of the other list in the new lists.

Examples:

```java
public class CSC310_Hw4_2 {
    public static void main(String[] args) {
        LinkedQueue a= new LinkedQueue();
        a.enqueue(1);
        System.out.print("["+a.last()+", ");
        a.enqueue(2);
        System.out.print(a.last()+", ");
        a.enqueue(4);
        System.out.println(a.last()+"]");

        LinkedQueue b= new LinkedQueue();
        b.enqueue(1);
        System.out.print("["+b.last()+", ");
        b.enqueue(3);
        System.out.print(b.last()+", ");
        b.enqueue(4);
        System.out.println(b.last()+"]");

        a=a.mergeQueues(a, b);
        while(!a.isEmpty())
            System.out.println(a.dequeue());
    }
```

```
itput - CSC310_Hw4_2 (run)
run:
[1, 2, 4]
[1, 3, 4]
1
1
2
3
4
4
BUILD SUCCESSFUL (total time: 0 sec
```

CSC310_Hw4_3

The only difference between CSC310_Hw4_2 and CSC310_Hw4_3 in the class is the final method in CSC310_Hw4_3 it has a method of search instead of merge.

```java
//searches the list for key and returns if its in or not
public boolean search(int key){
    Node c = head;
    while(c!=null){
        if(c.element==key)
            return true;
        c=c.next;
    }
    return false;
}
}
```

Example output:

```
89  public class CSC310_Hw4_3 {
90      public static void main(String[] args) {
91          LinkedQueue a= new LinkedQueue();
92          a.enqueue(1);
93          a.enqueue(2);
94          a.enqueue(3);
95          a.enqueue(4);
96          a.enqueue(5);
97          a.enqueue(6);
98          System.out.println(a.isEmpty());              //false
99          System.out.println(a.search(1));              //true
100         System.out.println(a.search(2));              //true
101         System.out.println(a.search(3));              //true
102         System.out.println(a.search(4));              //true
103         System.out.println(a.search(5));              //true
104         System.out.println(a.search(6));              //true
105         System.out.println(a.search(7));              //false
106         a.dequeue();
107         System.out.println(a.search(1));              //false
108         a.enqueue(1);
109         System.out.println(a.search(1));              //true
110         System.out.println("size: "+a.len());         //6
111         a.dequeue();
112         System.out.println("size: "+a.len());         //5
113         System.out.println("First: "+a.first());      //3
114         a.dequeue();
115         System.out.println("First: "+a.first());      //4
116         System.out.println(a.dequeue());              //4
117         System.out.println(a.dequeue());              //5
118         System.out.println(a.dequeue());              //6
119         System.out.println(a.dequeue());              //1
120         System.out.println(a.isEmpty());              //true
121     }
```

```
tput - CSC310_Hw4_3 (run)
run:
false
true
true
true
true
true
true
false
false
true
size: 6
size: 5
First: 3
First: 4
4
5
6
1
true
BUILD SUCCESSFUL (total time: 0 sec
```