Name: ZOHAIB HASSAN SOOMRO

RollNo#: 19SW42

Subject: DSA

Queue Assignment

- Create an algorithm and a method in both ArrayQueue and LinkedQueue which reverses the order of the queue.
  1) LinkedQueue:

     a) Algorithm:
        i) Find total number of elements in queue that will be swapped.

           i.e If Total elements=7 then 1st element swaps with 7th, 2nd with 6th, 3rd with 5th and 4th does not needs to be changed as it's middle element so total swaps is 3.

           So the number of elements for swapping=nos= (Number of total elements)/2.

        ii) Create a Node variable(say start) and point it to First element of Queue(i.e head.next).
        iii) Create another Node Variable(say end) and point it to last element of Queue(i.e head.previous).
        iv) Repeat step (v) to (ix) till nos becomes zero, decrementing it by 1 in each iteration.
        v) Create an Object variable(say temp) and store in it the value of element that is pointed by start.
        vi) Store value of element that is pointed by end, in element that is pointed by start.
        vii) Store value of temp in element that is pointed by end.
        viii) Point start to the element which is next to the element that is pointed by start(i.e start=start.next).
        ix) Point end to the element which is previous to the element that is pointed by end(i.e end=end.previous).

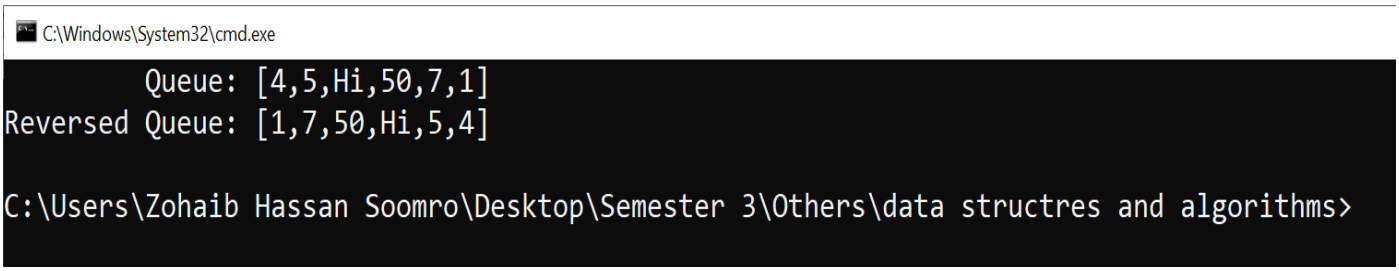
     a) Method(In purple color written below in this below class):

```
public class LinkedQueue implements Queue {

private int size;
private Node head=new Node(null);

private class Node {

private Object object;
private Node previous=this;
private Node next=this;

public Node(Object obj) {
   this.object = obj;
}

public Node(Object object, Node previous, Node next) {
   this.object = object;
```

```
    this.previous = previous;
    this.next = next;
}

}

@Override
public void add(Object obj) {
    head.previous = head.previous.next = new Node(obj, head.previous,
head);
    size++;
}

@Override
public Object first() {
    if (this.isEmpty()) {
        throw new IllegalStateException("Queue is empty!");
    }
    return head.next.object;
}

@Override
public Object remove() {
    if (this.isEmpty()) {
        throw new IllegalStateException("Queue is empty!");
    }
    Object firstElement = head.next.object;
    head.next = head.next.next;
    head.next.previous=head;
    size--;
    return firstElement;
}


//////////////////Method to Reverse the queue
public boolean reverse(){
     if (this.isEmpty())
        return false;
    Node start=head.next;
```

```
    Node end=head.previous;
    int till= size()/2;   //Calculating that how many values will be
swapped
    while (till--!=0) {
     Object temp=start.object;
     start.object=end.object;
     end.object=temp;

     start=start.next;
     end=end.previous;
    }
    return true;
}

@Override
public int size() {
    return size;
}

@Override
public String toString(){
     if (this.isEmpty())
       return "[]";
    String buffer="[";
    Node p=head.next;
    while(p!=head){
     buffer+=p.object+",";
     p=p.next;
    }
    return (buffer+"\b]");
}

@Override
public boolean isEmpty() {
    return size == 0;
}

public static void main(String[] args) {
    LinkedQueue queue= new LinkedQueue();
```

```
        queue.add(4);
        queue.add(5);
        queue.add("Hi");
        queue.add(50);
        queue.add(7);
        queue.add(1);
        System.out.println("        Queue: "+queue);
        queue.reverse();
        System.out.println("Reversed Queue: "+queue);}
    }
```



2) ArrayQueue:

b) Algorithm:

i) Create two integer variables(say start & end) and initialize them with 0(lower bound of array) and (size of queue-1) respectively.

ii) Repeat Step (iii) to (v) till start is less than end.
For example Size of queue is 6 then start=0 & end=6-1=>end=5 ,
So the 1st element swaps with 6th(start=0,end=5), 2nd with 5th(start=1,end=4), 3rd with 4th(start=2,end=3). So next time start=3 & end=2 and here we do not have more elements to swap that's why Loop condition should be (start<end).

iii) Create an Object variable(say temp) and store in it the value of element that is at index start.

iv) Store value of element that is at index end in element that is at index start and then increment start by 1.

v) Store value of temp in the element that is at index end and then decrement end by 1.

b) Method(In purple color written below in this below class):

```
public class ArrayQueue implements Queue {

private int size;
private Object array[];
```

```
public ArrayQueue(int capacity) {
    array = new Object[capacity];
}

@Override
public Object first() {
    if (this.isEmpty()) {
        throw new IllegalStateException("Queue is empty!");
    }
    return array[0];
}

@Override
public Object remove() {
    if (this.isEmpty()) {
        throw new IllegalStateException("Queue is empty!");
    }
    Object obj = array[0];
    System.arraycopy(array, 1, array, 0, size);
    array[--size] = null;
    return obj;
}

@Override
public void add(Object obj) {
    if (size == this.array.length) {
        resizeArray();
    }
    array[size++] = obj;
}

@Override
public int size() {
    return size;
}

@Override
public boolean isEmpty() {
    return size == 0;
```

```
}

public void resizeArray() {
   Object[] array2 = this.array;
   this.array = new Object[2 * size];
   System.arraycopy(array2, 0, this.array, 0, array2.length);
}

@Override
public String toString(){
   if (this.isEmpty())
      throw new IllegalStateException("Queue is empty!");
   String buffer="[";
   for (int i=0;i<size;i++) {
      buffer+=array[i]+",";
   }

   return (buffer+"\b]");
}


//////////////////Method to Reverse the queue
public boolean reverse(){
   if (this.isEmpty())
      return false;
   int start=0,end=size()-1;

   while (start<end) {          //condition for elements that will be
swapped
      Object temp=array[start];
      array[start++]=array[end];     //swapping elements
      array[end--]=temp;
   }
   return true;
}

public static void main(String[] args) {
   ArrayQueue queue = new ArrayQueue(2);
   queue.add(5);
```

```
        queue.add(51);
        queue.add("Hello!");
        queue.add(2);
        queue.add(3);
        queue.add(7);
        System.out.println("        Queue: "+queue);
        queue.reverse();
        System.out.println("Reversed Queue: "+queue);
    }
    }
```