Instructions: You are to complete this work individually. You may refer to your text and notes in the preparation of your answer. Please drop your paper by my office by 5:00pm on Thursday, March 16. Be sure to write out and sign the pledge on your paper.

1. A Deque, or “double ended queue,” as described in class, behaves like a queue, but with the additional capabilities of adding an item at the front and deleting an item from the rear. Attached you will find the code for the QueueLi class. Write the two methods addToFront and removeFromRear to turn this class into a Deque. (You only have to write these two methods: you aren’t responsible for any other modifications.)
package DataStructures;

import Exceptions.*;

// QueueLi class
//
// CONSTRUCTION: with no initializer
//
// ******************PUBLIC OPERATIONS********************
// void enqueue( x ) --> Insert x
// Object getFront( ) --> Return least recently inserted item
// Object dequeue( ) --> Return and remove least recent item
// boolean isEmpty( ) --> Return true if empty; else false
// void makeEmpty( ) --> Remove all items
// ******************ERRORS********************************
// getFront or dequeue on empty queue

/**
 * List-based implementation of the queue.
 * @author Mark Allen Weiss
 */
public class QueueLi implements Queue
{
    private ListNode front;
    private ListNode back;

    /**
     * Construct the queue.
     */
    public QueueLi( )
    {
        makeEmpty( );
    }

    /**
     * Test if the queue is logically empty.
     * @return true if empty, false otherwise.
     */
    public boolean isEmpty( )
    {
        return front == null;
    }

    /**
     * Make the queue logically empty.
     */
    public void makeEmpty( )
    {
        front = null;
        back = null;
    }

    /**
     * Get the least recently inserted item in the queue.
     * Does not alter the queue.
     * @return the least recently inserted item in the queue.
     * @exception Underflow if the queue is empty.
     */
    public Object getFront( ) throws Underflow
{ 
  if( isEmpty( ) )
    throw new Underflow( "QueueLi getFront" );
  return front.element;
}

/**
 * Return and remove the least recently inserted item
 * from the queue.
 * @return the least recently inserted item in the queue.
 * @exception Underflow if the queue is empty.
 */
public Object dequeue( ) throws Underflow
{
  if( isEmpty( ) )
    throw new Underflow( "QueueLi dequeue" );

  Object returnValue = front.element;
  front = front.next;
  return returnValue;
}

/**
 * Insert a new item into the queue.
 * @param x the item to insert.
 */
public void enqueue( Object x )
{
  if( isEmpty( ) ) // Make queue of one element
    back = front = new ListNode( x );
  else // Regular case
    back = back.next = new ListNode( x );
}

package DataStructures;

// Basic node stored in a linked list

class ListNode
{
  // Friendly data; accessible by other package routines
  Object element;
  ListNode next;

  // Constructors
  ListNode( Object theElement )
  { 
    this( theElement, null );
  }

  ListNode( Object theElement, ListNode n )
  { 
    element = theElement;
    next = n;
  }
}