Programming Assignment #1

125 points

Due Date/Time:

Your program is due on **Thursday, October 3rd**, at the beginning of class (8:00 A.M.).

The Project

For this assignment, you will write a linked list and use it (via composition) to build a stack and a queue class. These classes will be used for later assignments. You will write a driver program to test your classes, and also write a report with complexity analysis of the public methods in the three implementations. You will also do empirical timing tests on your classes to verify that they perform as expected. This report is an integral part of the assignment; the program is worth 100 points, and the report 25 points.

We want to segregate our data structures and separate them from any application programs. Accordingly, you must place all data structures in a package named `data_structures`. Your linked list class must implement the ListADT interface. The list is an unordered, singly linked list with both a head and tail pointer to facilitate maximum flexibility. Your project will consist of exactly the following files:

- `ListADT.java` The linked list interface (provided below)
- `LinkedListDS.java` Your implementation of the ListADT interface
- `Stack.java` Your stack class which must be built using your LinkedListDS
- `Queue.java` Your queue class which must be built using your LinkedListDS

All of the above four must go in package `data_structures`. Any driver/tester/timer programs will go in the level above the `data_structures` subdirectory. *[Sample tester/timer programs will be provided.]*

Submitting Your Assignment

Before the due date, copy your four files (listed above) into your `handin/` subdirectory. Do **not** create any folders within `handin/`; all files go directly in this folder.

You will submit a hardcopy printout of the three files you have written, plus your report at the beginning of class on the due date. Submit your report separately. That is, you will submit TWO SEPARATE THINGS. 1) The source code, and 2) the Report.

IMPORTANT: You may NOT submit multiple copies of either source code files or the report. ONE submission only. There will be a *substantial* grade penalty if we find multiple versions of either the source code or the report.

The ListADT interface:

```
package data_structures;
```

```
import java.util.Iterator;

public interface ListADT<E> extends Iterable<E> {


//  Adds the Object obj to the beginning of the list
    public void addFirst(E obj);

//  Adds the Object obj to the end of the list
    public void addLast(E o);

//  Removes the first Object in the list and returns it.
//  Returns null if the list is empty.
    public E removeFirst();

//  Removes the last Object in the list and returns it.
//  Returns null if the list is empty.
    public E removeLast();

//  Returns the first Object in the list, but does not remove it.
//  Returns null if the list is empty.
    public E peekFirst();

//  Returns the last Object in the list, but does not remove it.
//  Returns null if the list is empty.
    public E peekLast();

//  Finds and returns the Object obj if it is in the list, otherwise
//  returns null.  Does not modify the list in any way
    public E find(E obj);

//  Removes the first instance of thespecific Object obj from the list, if it exists.
//  Returns true if the Object obj was found and removed, otherwise false
    public boolean remove(E obj);

//  The list is returned to an empty state.
    public void makeEmpty();

//  Returns true if the list contains the Object obj, otherwise false
    public boolean contains(E obj);

//  Returns true if the list is empty, otherwise false
    public boolean isEmpty();

//  Returns true if the list is full, otherwise false
    public boolean isFull();

//  Returns the number of Objects currently in the list.
    public int size();

//  Returns an Iterator of the values in the list, presented in
//  the same order as the list.
    public Iterator<E> iterator();

}
```

Required methods for your Queue class are:

```
    // inserts the object obj into the queue
```

```
public void enqueue(E obj)

// removes and returns the object at the front of the queue
public E dequeue()

// returns the number of objects currently in the queue
public int size()

// returns true if the queue is empty, otherwise false
public boolean isEmpty()

// returns but does not remove the object at the front of the queue
public E peek()

// returns true if the Object obj is in the queue
public boolean contains(E obj)

// returns the queue to an empty state
public void makeEmpty()

// removes the Object obj if it is in the queue and
// returns true, otherwise returns false.
public boolean remove(E obj)

// returns an iterator of the elements in the queue.  The elements
// must be in the same sequence as dequeue would return them.
public Iterator<E> iterator()
```

Required methods for your Stack class are:

```
// inserts the object obj into the stack
public void push(E obj)

// pops and returns the element on the top of the stack
public E pop()

// returns the number of elements currently in the stack
public int size()

// return true if the stack is empty, otherwise false
public boolean isEmpty()

// returns but does not remove the element on the top of the stack
public E peek()

// returns true if the object obj is in the stack,
// otherwise false
public boolean contains(E obj)

// returns the stack to an empty state
public void makeEmpty()

// removes the Object obj if it is in the stack and
// returns true, otherwise returns false.
public boolean remove(E obj)

// returns a iterator of the elements in the stack.  The elements
// must be in the same sequence as pop() would return them.
public Iterator<E> iterator()
```

Additional Details

- The four files specified in this assignment **must** have the exact names and signatures as given.
- You will submit only these four files; no drivers/testers that you have written.
- You may import only classes needed for the Iterators. You may use any class in `java.lang` (the default package). You may not use any data structure or class in `java.util` other than those specified.
- Every class file must begin with your name and rohan class account number.
- Each method should be as efficient as possible. For example, the `push(E obj)` and `pop()` methods in class stack should be O(1) methods. A `push(E obj)` method that is O(n) is unacceptable.
- Additional details about how to perform timing tests and write the report will be given in class.
- Your project must compile and run on rohan to receive any credit for the assignment. For grading, your files will be copied from your handin folder to the grader's account. The project layout will be recreated, and then compiled and run.

Late Programs:

Late programs will be accepted with a penalty of 5% per day for seven days after the due date. The late penalty will be determined by the submission date/time of the latest submission of 1) the source code or 2) the report.

The Report

In addition to the source code for each of the three files you wrote for your project, you will also prepare a report. The report consists of two parts:

- Analysis of methods. You will analyze all of the public methods in your linkedListDS class, giving the O or θ complexity of each one, along with a precise justification for your answer.
- You will provide empirical timing tests on the following methods: `addFirst, addLast, removeFirst, removeLast, find, remove.`

Submit your report separately. That is, you will submit TWO SEPARATE THINGS. 1) The source code, and 2) the Report.

Cheating Policy:

There is a zero tolerance policy on cheating in this course. You are expected to complete all programming assignments on your own. Collaboration with other students in the course is not permitted. You may discuss ideas or solutions in general terms with other students, but you must not exchange code. (Remember that you can get help from me. This is not cheating, but is in fact enouraged.) I will examine your code carefully. Anyone caught cheating on a programming assignment or on an exam will receive an "F" in the course, and a referral to Judicial Procedures.