CS310 Fall 2015

Data Structures

Programming Assignment #2

75 points

Due Date/Time

Your program will be due on Tuesday, October 13th at the beginning of class.

The Assignment

For this assignment, you will implement the interface from program #1 (LinearListADT) but with a doubly linked list rather than an array. Your implementation must be named `LinearList`. Additionally, you will write a Stack and a Queue, which you will build with your LinearList class via composition.

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 `LinearList` class must implement the `LinearListADT` interface. Your project will consist of exactly the following four files, all of which **must** be in a `data_structures/` package:

- `LinearListADT.java` The linear list interface (provided below)
- `LinearList.java` Your implementation of the this interface
- `Stack.java` Your stack implementation that uses `LinearList`.
- `Queue.java` Your queue implementation that uses `LinearList`.

Submitting Your Assignment

Since this is programming assignment #2, your source code must go in `handin/prog2`. Please review the Program Submission Guidelines page carefully. You are responsible for following the directions given on this page. Place the three files you have written in this location to submit them for grading. You will submit hard copy of the three files on the due date.

Before the due date, copy your `LinearList.java, Stack.java, Queue.java` source code file into your `handin/prog2` subdirectory. Do not put the LinearListADT.java interface in handin/prog2. I will use my copy of the interface to compile your program. **IMPORTANT: Do not create any data_structures folder anywhere within `handin/`; your source code files go directly in the `handin/prog2` folder**.

You will submit a hardcopy printout of the three files you have written at the beginning of class on the due date.

IMPORTANT: Once you have submitted the assignment, you may NOT resubmit. Under no circumstances may you submit multiple hard copy of programming assignments. ONE submission only. There will be a *substantial* grade penalty if I find that you have submitted more than one printout of your source code.

The LinearListADT interface (same as program #1):

```
/*  Your name
```

```
    Your masc account number
*/

package data_structures;

import java.util.Iterator;
import java.util.NoSuchElementException;

public interface LinearListADT<E> extends Iterable<E> {
    public static final int DEFAULT_MAX_CAPACITY = 100;

//  Adds the Object obj to the beginning of list and returns true if the list is not full.
//  returns false and aborts the insertion if the list is full.
    public boolean addFirst(E obj);

//  Adds the Object obj to the end of list and returns true if the list is not full.
//  returns false and aborts the insertion if the list is full..
    public boolean addLast(E obj);

//  Removes and returns the parameter object obj in first position in list if the list is not empty,
//  null if the list is empty.
    public E removeFirst();

//  Removes and returns the parameter object obj in last position in list if the list is not empty,
//  null if the list is empty.
    public E removeLast();

//  Removes and returns the parameter object obj from the list if the list contains it, null otherwise.
//  The ordering of the list is preserved.  The list may contain duplicate elements.  This method
//  removes and returns the first matching element found when traversing the list from first position.
//  Note that you may have to shift elements to fill in the slot where the deleted element was located.
    public E remove(E obj);

//  Returns the first element in the list, null if the list is empty.
//  The list is not modified.
    public E peekFirst();

//  Returns the last element in the list, null if the list is empty.
//  The list is not modified.
    public E peekLast();

//  Returns true if the parameter object obj is in the list, false otherwise.
//  The list is not modified.
    public boolean contains(E obj);

//  Returns the element matching obj if it is in the list, null otherwise.
//  In the case of duplicates, this method returns the element closest to front.
//  The list is not modified.
    public E find(E obj);

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

//  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 underlying order of the list. (front first, rear last)
    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 behavior of your `LinearList` must be identical to the ArrayLinearList from program #1.
- As with program #1, the `addFirst/addLast/removeFirst/removeLast` methods must be O(1), which means a doubly linked list is required.
- Your `LinearList` class will have only a no-argument constructor, since linked lists are never 'full'.
- 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. You will need java.util.Iterator, and java.util.NoSuchElementException.
- Every class file must begin with your name and rohan class account number.
- Each method should be as efficient as possible. For example, your size() method should not loop down the linked list and count the elements.
- Your project must compile and run on rohan to receive any credit for the assignment. For grading, your file will be copied from your `handin/prog2` folder to my account. The project layout will be recreated, and then compiled and run. Watch your package structure! Any project that fails to compile will receive a zero.

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 timestamp of your file on rohan. Your printout is due at the next class session after your program file is placed in handin. Late penalties will be assigned for late printouts using the same rules. A two-day late printout will result in a 10% penalty.

The goal here is not to be needlessly punitive. Rather, it is to enforce the simple reality that all projects will have due dates, both in school and in industry. Many professors do not accept late work. Period. I will allow late work, but with the penalties stated above. Best practice is to turn in your work on time, everytime.

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. Nor may you copy code from former CS310 students, nor the Internet. You must be able to produce code on your own--otherwise no one will ever hire you. (Remember that you can get help from the TAs or 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.