Assignment 3.1: Linked Lists

Due: 8am, January 21, 2008

Directions

The problems for this assignment are described below. You will submit this assignment by emailing it do Mr. Wulsin. The file will be named "Assignment 3.1-LastName.zip", where [LastName] is obviously your last name. All of your submitted assignments will be named in this fashion. The Zip file will contain a folder of the same name (so, "Assignment 3.1-LastName"), which will contain a folder for each of the subsequent problems. Each of these subfolders will have the name of its particular problem. Each of these problem subfolders will contain all of the necessary files for that particular problem. It's very important that you follow all of these naming conventions exactly as specified.

Problems

1) Doubly Linked List

You will create two classes, `DblListNode` and `DblLinkedList`, to implement a doubly linked list. Your text has examples of implementations of singly linked lists, which you may use to base your doubly linked list classes off of, but if you *do* use them, you MUST cite them in your code. Your `DblLinkedList` will implement the `DblLinkedListI` interface (which is basically an abstract class definition, for more on Interfaces, see Section 11.7). You are provided with the `DblLinkedListI` interface, its parent interface, `LinkedListI`, and a fully fuctional tester class, `LLTester`. The `DblLinkedListI` and `LinkedListI` interfaces will tell you what methods you need to create in your DblLinkedList class. I recommend creating shells (fully defined methods but with nothing between the curly brackets, or returning `null` if a return is necessary) of each method at the start so that you can test the methods as you write them. If your DblLinkedList implementation works correctly, the output of the `LLTester` class should be

```
29
[I@6af62373
Third Quarter
2009
Hi there.
e
----
The element value at index 2 is: Third Quarter
The new value at index 2 is: Second Semester
```

You will also incorporate throwing exceptions in this problem. If a user passes an invald (out of bounds) index to the `setNode` or `getNode` methods, you will throw an `IndexOutOfBoundsException` using the following code:

```
throw new IndexOutOfBoundsException(‘‘Put your error message here’’);
```

Make sure to test these exceptions in your `LLTester` class, which doesn't do it automatically.

2) Reduction in Force (AB only)

The tough economic times have forced the investment bank for which you work to lay off all but one of the analysts in your group. Instead of trying to choose the smartest and hardest working of the bunch, your boss has decided to select the one by the following method. You will all get into a circle, with someone designated the start of the circle. You give the first person a paintball gun and tell them to shoot the person directly clockwise of them in the chest, effectively *laying them off*. The person who shot the guy to his left then hands the paintball gun to the next person (the guy originally two people to his left), who then does the same thing to the person to *his* left. This process continues around and around the circle until only one person is left standing, the lucky one who gets to keep his job.

Since you don't want to lose your job (finding new finance jobs isn't very easy these days), you decide to write a progam that will figure out what position you should stand in so that you will remain given the number of people in the group. For example, if there are 5 people in the circle, you want to be at position 2; 10 people, position 4; 100 people, position 72.

You will approach this programming problem through a circular (doubly-) linked list. Each node in the list will represent a person and have an `int` value representing its position in the circle. As each person is eliminated, their node is removed from the list, until there is just one node left.

Your tester method (which you will create) will specify a circle of a certain size, process the eliminations, and then print out the position of the last person standing.

For simplicity, you can just the the `DblListNode` class you created in the previous problem. You will create a new class, `CircLinkedList`, which may be very similar to the `DblLinkedList` you already created except that there is only a first element (no last, since the last element can be gotten through first.getPrevious()). It will implement the `DblLinkedListI` interface as well. You may copy and paste as much as you want from your `DblLinkedList` class (indeed, the code will be very similar) but make sure to adjust it accordingly for a circular linked list. Note, your implementation will not have a header node (as discussed on pg. 519 of your book).