HW2 – Linked Lists
Assigned: Saturday, May 29th
Due: Friday, June 4th, 2010 before 10:00pm

Overview
In part 1 of this assignment, you will use the provided Node class and IList interface to implement a circular doubly linked list in **CircularDoublyLinkedList.java**. Read through the javadocs in the IList interface for detailed instructions on what methods you must implement and what they should do.

Once you have completed the CircularDoublyLinkedList class, you will then use your new linked list implementation in part 2 to write **PostFixSolver.java,** which will solve arithmetic equations written in postfix notation (also known as reverse polish notation). Be sure to thoroughly read through the description of this class implementation in the part 2 requirements before attempting this.

Please read the "Remember" section at the end of this pdf for updated instructions on the javadocing/commenting requirements.

Part 1
You must implement the CircularDoublyLinkedList class based on the specifications described in IList.java. The Node class has been provided for you.

Node
This class has been provided for you and you should not edit or change it in any way. This class should be used as the nodes of your linked list in your implementation. Please read through the javadocs in the class for detailed explanations of what each method does.

IList
This interface has been provided for you. It has all of the methods that you must implement in your linked list implementation and thorough descriptions in the javadocs of what each method must do. Feel free to add any additional methods to this interface (not required) but DO NOT change or edit any of the given method signatures.

CircularDoublyLinkedList
This is the class that you must implement. This class must implement the provided IList interface (which means you must provide implementations for all of the defined methods in IList). Feel free to add any additional methods if you feel the need.

Further requirements for your implementation:

- Must contain an empty constructor which initializes an empty linked list (calling size() on the linked list after simply instantiating it should return 0).

- Must contain a private Node variable named head which always holds a reference to the first node in your list (can be null if list is empty)
- List must be a doubly linked list
- List must be a circular linked list (tails's next points to the head, head's previous points to the tail)
- A main method which thoroughly tests all of the methods (prove to us that your methods work as intended)

Pro Tip
Linked lists are very hard to work out conceptually (especially circular and doubly linked lists) so I highly recommend drawing out each operation to figure out what references need to be changed and in what order. Be sure to consider the special cases as well for each method (list is empty, head is the only item in the list).

Part 2

In this part, you will implement the **PostFixSolver** class. The purpose of this class is to solve arithmetic equations given in postfix notation (also known as reverse polish notation).

Postfix Notation
Postfix notation is simply a different way of representing common math problems. For example, 5+3 becomes 5 3 + in postfix notation. Additionally, 10 5 3 + - is equivalent to 10 - (5+3). The way to solve equations in postfix notation is the following:

1. Read in first element of the equation (must be a number).
2. If the element is a number then push it on a stack.
3. Otherwise, element must be an operator so pop 2 elements off the stack and push the result of performing the operation on those 2 elements back onto the stack.
4. Read in next element in the equation and repeat from step 2.

Let's walk through solving 10 5 3 + -.

1. We read in the first element, "10". It is a number so we push it on our stack.
2. We read in our next element, "5". It gets pushed.
3. Read in 3, push it onto the stack as well. Our stack now looks like this:

Top of Stack =>	3
	5
	10

4. We read in the next element, "+". This is an operator so we pop 2 values off the stack (first is 3, second is 5) and then push the result of performing that operation back on the stack.

stack.push(5 + 3);

Stack now looks like this:

8
10

5. Read in next element, “-”. This is an operator so we pop 2 values off the stack (first is 8, second is 10) and then push the result of performing that operation back on the stack.

   stack.push(10 - 8);

6. We have now reached the end of the equation. That means that our stack should only have 1 value in it at this point, the solution! (the solution in this case would be 2)

PostFixSolver
The PostFixSolver class requirements are as follows:

- Must have a private IList<Integer> variable named stack. This will point to an instance of your CircularDoublyLinkedList.
- Empty constructor that simply sets up the stack instance variable by pointing it to a new instance of your CircularDoublyLinkedList. The stack should be empty at this point.
- Must have a *solve* method which takes in a String representation of an equation in postfix notation. One example of a parameter would be “10 5 3 + -”. This method must solve the equation using the algorithm described above and it must use your CircularDoublyLinkedList implementation as the stack. You must support the following operators:
    - “+”
    - “-”
    - “*”

  This method should return the result of the equation as an int. You may assume that the parameter passed in will always be properly formatted.
- A main method which thoroughly tests your solve method. Some example equations to try:
    - “10 5 3 + -” => 2
    - “5 1 2 + 4 * + 3 -” => 14
    - “1 2 3 4 5 * + 6 * * +” => 277
    - “7 16 * 5 + 16 * 3 + 16 * 1 +” => 30001

What to turn in
Turn in all the files you used to write this assignment into t-square. For this assignment, please submit:

- Node.java (provided)
- IList.java (provided)
- CircularDoublyLinkedList.java
- PostFixSolver.java

By submitting this assignment, you confirm that you understand the CS1332 collaboration policy, and that your submission complies with it.

Keep in mind that you can submit the assignment as many times as you want before it is due. If you find yourself working close to the time the assignment is due then please submit often since t-square has a habit of going down at the worst possible times.

Remember
These are required, and you will lose points if you do not do any of the following:

- Javadocing and Commenting (EVERY method must be clearly and sufficiently Javadoced, even the getters and setters). Reading your javadoc should immediately tell me what your method does. Be sure to include the @param and @return tags.
- Putting your name in the @author section in the class header Java-doc for every file you created (do not do this for files we have provided to you).
- Including your collaboration statement in a comment at the top of every file you created (do not do this for files we have provided to you). For example: "I worked alone on this homework assignment."