3 Homework Assignment 3

Your assignment for Homework 3 is to implement a stack to parse simplified XML input and verify that the input consitutes well formed XML. This is quite similar to parsing an algebraic expression to ensure that the open and closing parentheses have been done correctly. Indeed, many algebraic expressions are written with three different opening and closing characters: parentheses `(` and `)`, "square brackets" `[` and `]`, and "curly braces" { and }. XML just allows a user to generate an essentially infinite variety of opening and closing tokens as `<tag>` and `</tag>` with the string used for `tag` being almost any text string at all.

This programming assignment will require you to write a program to accomplish two different tasks. The first is the code for creating and managing a stack as a data structure, and the second is the code that uses the stack to parse XML.

Your main class need only declare and then invoke an `XLMParser` object. This object should contain a `checkValidXML` method that takes a `Scanner` as a parameter and then returns a `boolean`

You should use the parsing code done in Lab Assignment 3 to be able to separate the tokens in the input file into open tokens, close tokens, and data tokens.

The way to test for valid XML using a stack is this. As you parse the input file, all open tokens and data tokens should be pushed onto the stack. When you encounter a close token, you should pop the zero or more data tokens, saving them in a string, and the first open token that is popped off the stack should be the open token that matches the close token. Should you encounter the one exceptional case `<whatever />` then you should push the string `_NULL_` (opening and closing underscores intentional) onto the stack as the data element. Yes, this does mean that this particular tag cannot be used for other purposes, but that's the way that a number of such software tools work.

You should implement the stack object using the `interface IStack` given here and found on the website.

The actual `Stack` class can use your `DLL` from Homework 1. By doing this, the `push` method can simply use the existing `addAtHead` method to push a node onto the stack, and the `pop` method can simply return the head node of the DLL. (Remember to unlink the head node as well.) Creating a stack is thus the easy part of this program; the hard part of the program is

to use the stack to verify correctness of the input file.

Note that you will have to open up your `DLL` code slightly to make this work. There is a valid question that can be raised about which methods to add, delete, modify nodes should be `public` and which should be `private`. In the original code only the `add` method was `public`. But now, if we want to make the `push` method essentially synonymous with the `addAtHead` method, this last needs to be made `public` as well.

```
/*************************************************************************
 * Interface to describe a stack.
 * Copyright (C) 2011 by Duncan A. Buell.  All rights reserved.
 * @author Duncan A. Buell
 * @version 1.00 2011-01-18
**/
public interface IStack<T extends Comparable<T>>
{
/*************************************************************************
 * Method to get the <code>size</code>
 *
 * @return the <code>int</code> value of <code>size</code>
**/
  public int getSize();
/*************************************************************************
 * Method to peek at the top entry on the stack.
 *
 * @return the <code>T</code> data from the top of the stack.
**/
  public T peek();
/*************************************************************************
 * Method to pop an entry from the top of a stack.
 *
 * @return the <code>T</code> data from the top of the stack.
**/
  public T pop();
/*************************************************************************
 * Method to push an entry onto the top of a stack.
 *
 * @param node the <code>T</code> data to add.
**/
  public void push(T node);
/*************************************************************************
 * Method to <code>toString</code> a complete DLL, including head
 * and tail nodes.
**/
  public String toString();
} // public interface IStack<T extends Comparable<T>>
```

Figure 1: IStack.java