Lab Assignment 2

In preparation for future homework assignments, you are to write code to parse text input to identify opening and closing XML tags and data tokens that are not XML tags.

The input data can be found on the website. This input is the XML input data to be used in a project in the USC Center for Digital Humanities in order to put on the web a book authored by Dr. Ward Briggs. This is a book of biographical information on North American scholars in the classics. Part of the input data is as follows.

```
<text>
  <body>
    <div>
      <listPerson>
        <person>
          <persName>
            <surName>
NEMIAH
            </surName>
            <foreName>
Royal
            </foreName>
            <addName>
Case.
            </addName>
          </persName>
          <born>
21 May 1891, Hartford, CT, to James William &amp;amp; Carrie Delcina Case N.
          </born>
          <married>
Ruth Louise Comes, 5 Apr. 1917; Elly Olesen, 1 July 1950; Dr. Eugenie Bilfinger, 3 Sept.
          </married>
          <education>
A.B. Yale, 1912; Ph.D. 1916; study at Gttingen, 19124.
          </education>
```

An opening XML (for this assignment) token has a single tag contained within angle brackets, with no spaces. A closing token has the same structure

except that the character immediately following the opening angle bracket is a forward slash.

To be valid XML under modern rules it is necessary that an XML document have properly nested opening and closing tags, just as you would nest parentheses in an algebraic expression. In a later homework assignment you will be asked to parse this file (and other files) using a stack to verify that the nesting is correct, that the document is valid XML, and to extract the data.

ASSUMPTIONS AND CAVEATS

1. You may assume that all the blank spaces are significant and that there are no blank spaces that are not supposed to be there. That is, when you read a token using the `next()` method for a `Scanner`, you will be returned one of three things:

   (a) a token with an opening angle bracket AND a closing angle bracket AND a forward slash for the second character (subscript 1), which means that this token is a closing token;

   (b) a token with an opening angle bracket AND a closing angle bracket AND something other than a forward slash for the second character, which means that this token is an opening token;

   (c) a token whose first character is not an opening angle bracket OR whose last character is not a closing angle bracket, and this token will be for data.

2. To be honest, real XML tags can have "atttributes" inside the angle brackets as well as the tag, just as there can be more inside the angle brackets in the HTML of a web page. But that complicates the parser substantially, so we won't allow anything inside the angle brackets except the tag.

Your program should correctly parse the input file, strip off the angle brackets and forward slashes, and identify an input token as an open or close tag or just data. The first few lines of my output are the following:

In your homework assignment you will be required to match up the tags, so it will be necessary for you to be able to extract the tag itself from inside the tokens that also have the angle brackets and forward slashes.

You will probably want to implement at least three different methods to simplify the code that parses the file. Two of these will be testing methods to return a `boolean` as to whether the `String` in question is an opening token or a closing token (note that for this assignment, if it's not one of these two, then it is by definition "data"). The third method you will probably want will take either an opening or closing token and strip away the angle brackets and forward slashes to return only the tag.

Strategy

This assignment has relatively little to do with data structures *per se*, but it has a lot to do with programming and with manipulation of `String` data. The body of the main program for this assignment is essentially only two lines long:

1. Create a `new` instance of a `ParseTokens` object.

   You will need only one additional `ParseTokens` class (although you may wish to use more classes including the `FileUtils` class).

2. Write a `parseTokens` method in the `ParseTokens` object to read the data file and print a running progress report on the data as read. The `parseTokens` method should be passed in a `Scanner` that is linked to the input file, so you can read the input data.

   I recommend that in the very early stages you copy the first five or six lines of the input file and use only that much data for testing your code. You are going to need to verify that you can distinguish open tags, close tags, and data tokens that are just data tokens. If you can do this with the first half-dozen lines of input, you probably have a program that will work, and with only half a dozen lines, the tracing information that you (might want to) include in your code won't produce so much output that you can't look at it carefully. When you think you can read the first several lines correctly, then try reading the entire file.

   Strategy for writing `parseTokens`: First, just make sure you can read the data, line by line. Do this perhaps with a loop

```
while(infile.hasNext())
{
  token = infile.nextLine();
  System.out.printf("token read was '%s'%n",token);
}
```

Once you know you can read the file, then start writing the methods that will manipulate the data you are reading into the `token` variable.

I recommend that you have a separate method returning a `boolean` for `isOpenTag(token)` and for `isCloseTag(token)`. The `isOpenTag(token)` will return `true` if the `String` value of `token` has a less-than angle bracket as its lead (zero-th) character and a greater-than angle bracket as its trailing character and the character at subscript 1 *is not* a slash character. The `isCloseTag(token)` will return `true` if the `String` value of `token` has a less-than angle bracket as its lead (zero-th) character and a greater-than angle bracket as its trailing character and the character at subscript 1 *is* a slash character.

If neither of these happens to be true, then the token is simply data and should be treated as such.