CSE 143, Winter 2009
Programming Assignment #2: HTML Validator (20 points)
Due Thursday, January 22, 2009, 11:30 PM

This program focuses on using `Stack` and `Queue` collections. Turn in three files named `HtmlValidator.java`, `test.html`, and `output.txt` from the Homework section of the course web site. You will also need the support files named `HtmlTag.java` and `TestValidator.java` from the web site; place them in the same folder as your program.

`HtmlValidator.java` is your program, and `test.html` and `output.txt` are a test case you will create and its output.

Program Description:

In this assignment you will write a class that examines HTML files to figure out whether they represent "valid" sets of tags. You will use stacks and queues to examine the tags and figure out whether they match appropriately.

Pages on the World Wide Web are written in a language called Hypertext Markup Language, or *HTML*. An HTML file consists of text surrounded by special markings called *tags*. Tags give special information to the text, such as formatting (bold, italic, font size) or layout information (paragraph, table, bulleted list). Some tags also specify comments or information about the document itself (header, page title, document type).

A tag consists of a named *element* between less-than `<` and greater-than `>` symbols. For example, the tag for making text bold uses the element `b` and is written as `<b>`. Many tags apply to a range of text, in which case a pair of tags is used: an *opening* tag indicating the start of the range and a *closing* tag indicating the end of the range. A closing tag has a `/` slash after its `<` symbol, such as `</b>`. To make some text bold on a page, one would put the text to be bold between opening and closing `b` tags, `<b>`**like this**`</b>`. Tags can be nested to combine effects, `<b><i>`***like this***`</i></b>`.

Some tags, such as the `br` tag for inserting a line break, do not cover a range and are considered to be self-closing. Self-closing tags do not need a closing tag; for a line break, only a tag of `<br>` is needed. Some web developers write self-closing tags with an optional `/` before the `>`, such as `<br />`.

One problem on the web is that many developers make mistakes in their HTML code. All tags that cover a range must eventually be closed, but some developers forget to close their tags. Also, whenever a tag is nested inside another tag, `<b><i>`***like this***`</i></b>`, the inner tag (`i` for italic, here) must be closed before the outer tag is closed. So the following is not valid HTML: `<b><i>`***bold and italic text***`</b></i>`

Here is an example of a valid HTML file, with its tags in bold:

```
<!doctype html public "-//W3C//DTD HTML 4.01 Transitional//EN">
<!-- This is a comment -->
<html>
  <head>
    <title>Marty Stepp</title>
    <meta http-equiv="Content-Type" content="text/html">
    <link href="style.css" type="text/css" rel="stylesheet" />
  </head>

  <body>
    <p>My name is Marty Stepp.  I teach at
       <a href="http://www.washington.edu/">UW</a>.
    </p>
    <p>I was at the University of Arizona from 1997 - 2003.
       Here is a picture of my cat:
       <img src="images/kitteh.jpg" width="100" height="100">
    </p>
  </body>
</html>
```

You don't need to learn HTML to do this assignment. But there are many HTML tutorials and links on the Web, such as:

- http://www.w3schools.com/html/
- http://www.cs.washington.edu/190m/

Implementation Details:

For this program you will write a class named `HtmlValidator`. You will have to use Java's `Stack` and `Queue` from `java.util`. Your class must have the following constructors and methods. It must be possible to call these methods any number of times on your object, in any order, and get the same expected results each time.

`public` **`HtmlValidator`**`(Queue<HtmlTag> tags)`

In this constructor you should initialize your validator for examining the given queue of HTML tags representing the entire contents of an HTML file. For example, the queue for the page shown previously would contain the following tags:

front `[<!doctype>, <!-->, <html>, <head>, <title>, </title>, <meta>, <style>, </style>, </head>, <body>, <p>, <a>, </a>, </p>, <p>, <img>, </p>, </body>, </html>]` *back*

If the queue passed to your constructor is `null`, you should throw a `NullPointerException`.

`public void` **`setTags`**`(Queue<HtmlTag> tags)`

In this method you should set the validator to use the given queue of tags. Future calls to `getTags` and `validate` should use this new queue in place of the previous one. If the queue is `null`, you should throw a `NullPointerException`.

`public Queue<HtmlTag>` **`getTags`**`()`

In this method you should return your validator's queue of HTML tags. You must always return the same queue contents that were passed in to the constructor or `setTags`. If one of your other methods manipulates the queue while processing it, you must put the queue back to its original state before that method is finished running.

`public boolean` **`validate`**`()`

In this method you should print a textual representation of the HTML tags in your queue. Each tag displays on its own line. Every opening tag that requires a closing tag increases the level of indentation of following tags by four spaces until its closing tag is reached. The output for the HTML file on the first page would be:

```
<!doctype>
<!-->
<html>
    <head>
        <title>
        </title>
        <meta>
        <link>
    </head>
    <body>
        <p>
            <a>
            </a>
        </p>
        <p>
            <img>
        </p>
    </body>
</html>
```

To generate the output for this method, analyze your queue of tags with a `Stack`. The basic idea of the algorithm is that when you see an opening tag that requires a closing tag, you should push it onto a stack and increase your indentation. When you see a closing tag, you should pop the top element from the stack and decrease your indentation. You may use a single `Stack` and a single `Queue` (in addition to the queue passed to your validator's constructor) to help you compute the result. You may not use any other collections, arrays, etc., though you can create as many simple variables as you like.

Your method should also return `true` if the page has valid HTML and `false` if not. A valid queue of HTML tags is defined to be one where every opening tag that needs a closing tag has one, and where every closing tag closes the most recently opened tag that requires a closing tag.

For example, the following HTML is valid:

```
<p><b>bold text <i>bold and italic text</i> just bold again</b> <br/> more </p>
```

The following HTML is *not* valid, because the </b> appears before the </i>:

```
<p><b> bold text <i>bold and italic text</b> just italic</i> neither</p>
```

The following HTML is *not* valid, because the `<html>` tag is never closed:

```
<html><body> <b><i>bold italic</i></b> normal text</body>
```

Error Handling:

Your `validate` method should print error messages if you encounter either of the following conditions in the HTML file:

- A closing tag that does not match the most recently opened tag (or if there are no open tags at that point).
- Reaching the end of the HTML input with any tags still open that were never closed.

For example, suppose the previous short HTML file were modified to add several errors, as follows: an added unwanted `</!doctype>` tag, a deleted `</title>` tag, an added second `</head>` tag, and a deleted `</body>` tag:

```
<!doctype html public "-//W3C//DTD HTML 4.01 Transitional//EN">
</!doctype>

<!-- This is a comment -->
<html>
  <head>
    <title>Marty Stepp
    <meta http-equiv="Content-Type" content="text/html">
    <link href="style.css" type="text/css" rel="stylesheet" />
  </head>
  </head>

  <body>
    <p>My name is Marty Stepp.  I teach at
       <a href="http://www.washington.edu/">UW</a>.
    </p>
    <p>I attended the University of Arizona from 1997 - 2003.
       Here is a picture of my cat:
       <img src="images/kitteh.jpg" width="100" height="100">
    </p>
</html>
```

The resulting output for this invalid file should be the following:

```
<!doctype>
ERROR unexpected tag: </!doctype>
<!-->
<html>
    <head>
        <title>
            <meta>
            <link>
ERROR unexpected tag: </head>
ERROR unexpected tag: </head>
            <body>
                <p>
                    <a>
                    </a>
                </p>
                <p>
                    <img>
                </p>
ERROR unexpected tag: </html>
ERROR unclosed tag: <body>
ERROR unclosed tag: <title>
ERROR unclosed tag: <head>
ERROR unclosed tag: <html>
```

The reason that there are two error messages for `</head>` are because neither `</head>` tag seen matches the most recently opened tag at the time, which is `<title>`. The four unclosed tags at the end represent the fact that those four tags didn't have a closing tag in the right place (or, in some cases, no closing tag at all).

Because of the simplicity of our algorithm, a single mistake in the HTML can result in multiple error messages. Near the end of the file is a `</html>` tag, but this is not expected because `body`, `title`, and `head` were never closed. So the algorithm prints many errors, including that the `html` tag is unclosed, though really the problem is that the `html` tag was closed in the wrong place. Also notice that an unexpected closing tag does not change the indentation level of the output.

The revised algorithm is the following: Examine each tag from your queue, and if it is an opening tag, push it onto a stack. If it is a closing tag, examine the top of your stack. If the two tags match, pop the tag from the top of the stack. If they don't match, it is an error. When you have exhausted the queue contents, print errors for any tags remaining on the stack.

Provided Files:

- `HtmlTag.java`: Objects that represent HTML tags for you to process.
- `TestValidator.java`: A testing program to run your `HtmlValidator` code and display the output.

An `HtmlTag` object corresponds to an HTML tag such as `<p>` or `</table>`. You don't ever need to construct `HtmlTag` objects in your validator, but you will process a queue of them that is passed to you. Each has the following methods:

`public String getElement()`
Returns this HTML tag's element name, such as `"table"`.

`public boolean isOpenTag()`
Returns `true` if this tag is an opening tag, such as `<p>` or `<html>`.

`public boolean matches(HtmlTag other)`
Returns `true` if this tag and the given tag have the same element but opposite types, such as `<body>` and `</body>`.

`public boolean requiresClosingTag()`
Returns `true` if this element requires a closing tag; usually this is `true`, except for a few elements such as `br` and `img`.

`public String toString()`
Returns a string representation of this HTML tag, such as `"<p>"` or `"</table>"`.

Submitting a Test Case:

In addition to your `HtmlValidator.java`, you will also create a test case of your own that helps verify your validator. Create a file `test.html` that contains any (non-empty) contents you like, along with a file `output.txt` containing the output you see when you run the validator's `validate` method on that HTML file's tags. The `test.html` can be a file you create from scratch, or you can go to an existing web page and save its contents to a file. The purpose of this exercise is to make you think about testing. If you prefer, you may instead write a Java test program for your validator and submit it as `MyValidatorTest.java`. You may share your testing code with other students on the course message board.

Development Strategy and Hints:

We suggest the following development strategy:

1. Create the class and declare every method. Leave every method's body blank, or make methods return a value like `null` or `false` every time. Get your code to run in the testing program (though the output will be incorrect).
2. Implement the bodies of the constructor, `getTags`, and `setTags` methods. Verify them in the testing program.
3. Write an initial version of `validate` that assumes the page is valid and does not worry about errors. Get the overall algorithm, output, and indentation working for valid HTML.
4. Add the `validate` code that looks for errors and prints appropriate error messages.

Style Guidelines and Grading:

Part of your grade will come from appropriately utilizing stacks and queues. You may only use their methods shown in lecture and section; it is forbidden to use them in ways that are not stack/queue-like. For example, you may not call any `Stack` methods that accept index parameters. You also may not examine a stack or queue using a "for each" loop.

Redundancy is always a major grading focus; avoid redundancy and repeated logic as much as possible in your code.

Properly encapsulate your objects by making any data fields in your class `private`. Avoid unnecessary fields; use fields to store important data of your objects but not to store temporary values only used within a single call to a method.

You should follow good general style guidelines such as: appropriately using control structures like loops and `if`/`else` statements; avoiding redundancy using techniques such as methods, loops, and `if/else` factoring; properly using indentation, good variable names, and proper types; and not having any lines of code longer than 100 characters.

Comment descriptively in your own words at the top of your class, each method, and on complex sections of your code. Comments should explain each method's behavior, parameters, return, pre/post-conditions, and any exceptions thrown.

For reference, our solution is around 90 lines long including comments, though you do not have to match this exactly.