In this assignment, you will use the stack and queue implementations from the Java Collections API in order to build a program that determines whether an HTML page is well formatted.

In completing this assignment, you will:

- Become familiar with the methods in the java.util.Stack class and java.util.Queue interface
- Work with an abstract data type (specifically, queues) by using only the interface of an implementation
- Apply what you have learned about how stacks and queues work

Debugging/Error Note:

If you run into errors/bugs/don't understand the output that Codio is giving you, please post in the Discussion Forum and a TA will assist you! Please do NOT email Codio as they will not review any errors you are getting.

Background

Web pages are written in Hypertext Markup Language (HTML). An HTML file is composed of text surrounded by tags, where the tags "mark up" the text by specifying its format, layout, or other information. Tags can be nested as well.

Here is a simple example, with the tags highlighted in bold:

```
<html>

<head><title>Sample HTML page</title></head>

<body>

This is some <b>HTML text!</b>

</body>
```

```
</html>
```

The exact meanings of the tags are not important (though if you'd like to know more, you can sign up for our "Programming for the Web with JavaScript" course as part of this edX series!)but tags such as **<body>** and **<b>** are known as "open tags" because they indicate the start of some formatting, and tags such as **</body>** and (with the forward slash before the word) are known as "close" tags because they indicate the end of the formatting.

In theory (though not often in practice), well formatted HTML requires that the tags are "balanced," i.e. that open tags are matched by their corresponding close tag in the correct order.

For instance, if we ignore whitespace and the text between the tags, we end up with this:

```
<html><head><title></title></head><body><b></b></body></html>
```

Note that there is some symmetry in the HTML tags, in that whenever we close a tag, it matches the most recent (unclosed) open tag.

For instance, if we highlight the "title" tags, we see that a close tag matches the last open tag:

```
<html><head><title></title></head><body><b></b></body></html>
```

And in this case, the close "body" tag matches the open "body" tag, which is the most recently opened tag that has not yet been closed (since the "b" tag is already closed):

```
<html><head><title></title></head><body><b></b></body></html>
```

Some HTML tags are "self-closing" and do not rely on a matching closing tag. For instance, here the "br" tag closes itself:

```
<html><head>head><body><b><br/></b></body></html>
```

A self-closing tag is one that ends with the forward slash character, as opposed to a closing tag, which starts with one.

It is easy to make mistakes in HTML code! Most commonly, people forget to close tags or close nested tags in the wrong order, e.g. something like this:

```
<html><head><title></title><body><b></body></b></html>
```

In this case, there is no close “head” tag, and the “body” tag is closed in the wrong order: it should come after the close “b” tag.

In this assignment, you will write a method that determines whether an HTML file is well formatted using a stack. Every time your code encounters an open tag, it should push it onto the stack; when it encounters a close tag, it should pop the tag off the top of the stack, and if they don’t match, then you’ll know the file is not well formatted. More examples and explanation are provided below.

Getting Started

Download htmlreader.java and htmltag.java, which contain code that you can use in this assignment. You should not change either of these implementations for this assignment.

HtmlTag.java represents information about a single HTML tag. Methods that may be useful to you:

- **getElement()** Gets the element name (String) specified in this tag.
- **isOpenTag()** Checks whether this is the opening tag. If the tag is either the closing tag or self-closing (e.g. **
** is a line break tag that doesn’t need any accompanying text), **isOpenTag** will return **false**.
- **isSelfClosing()** Checks whether a tag is self-closing (e.g. **
**)

- **matches(HtmlTag other)** Checks whether an HtmlTag **other** is the matching open/close tag to itself (e.g. **** and **** or vice versa).>

In **HtmlReader.java** you will find a method called **getTagsFromFile** that reads in the path to an HTML file and separates it into tokens. The output is a representation of the HTML file as a Queue of *HtmlTags* in the order in which they were encountered. You may edit this code if you'd like, but please do not modify *HtmlTag.java*.

Also download htmlvalidator.java, which contains the unimplemented method for the code that you will write in this assignment.

Activity

In **HtmlValidator.java**, implement the **isValidHtml** method. **isValidHtml** should take as input a Queue of HtmlTags and return a Stack of HtmlTags that verifies the correctness of the tag structure, according to the specification described below.

The method should be implemented as follows:

If the HTML file is well formatted, the method should return an empty Stack. For example:

```
<html><body><h1>heading</h1><p>paragraph</p></body></html
>
```

In this case, the closing tags match the opening tags, so the HTML is valid. When you get to the end of the file/Queue, the Stack is empty.

If the HTML file is not well formatted, the method should return the Stack in its current state (i.e., with its current values) at the time the code determined that the tags were not balanced.

Here are some example cases to consider:

Example #1: Tags closed in incorrect order

<html><body> Sentence here** **</body></html>

In this case, you would push all opening tags onto the Stack so that it looks like this:

<body>

and, upon encountering a closing tag in the Queue, you would want to check that Stack to see if the correct match is present. The first closing tag you encounter is ** **; however, the last opening tag (at the top of the Stack) is . That's bad. As soon as you determine that the HTML file is not valid, **return the Stack of opening tags** without popping off the mismatched opening tag. In this case, the expected output would be a Stack containing (going from bottom to top): <html><body>

Example #2: Closing tag with no opening tag
<html><body>Correct
Sentence
here** **</body></html>

In this case, the first closing tag that you encounter (**** New Roman',serif; color: #222222; background: white;">) does match its opening tag, but the next one (** **) does not, so the expected output would be a Stack containing (going from bottom to top): **<html><body>**

Note that the **
** tag is self-closing and should not be placed on the Stack!

Example #3: Opening tag never closed

<html><body>This is some text

In this case, the method reaches the end of the file/Queue and there are still items on the Stack, since those opening tags were never closed. The expected output would be a Stack containing (going from bottom to top):

<html><body><b>

Example #4 (the tricky part!): Closing tag with no opening tag, everything okay until then

`<html><body><p>Hello, world!</p></body></html>`**</p>**

This is similar to Example #2 except that now when you encounter the closing tag that has no opening tag, the Stack is empty since everything before then is matched. However, returning an empty Stack means that the file *is* well formatted! In this case, though, you need to return **null** to indicate that the file is not well formatted. Think about how you can tell the difference between when to return null and when to return an empty Stack.

Please do not change the signature of the *isValid* method (its parameter list, name, and return value type). Also, do not create any additional .java files for your solution, and please do not modify *HtmlTag.java*. If you need additional classes, you can define them in *HtmlValidator.java*. Last, be sure that your *HtmlValidator* class is in the default package, i.e. there is no "package" declaration at the top of the source code.

Helpful Hints

Documentation about the methods in the Stack class and Queue interface in the latest version of Java are available at:

- https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html
- https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html

Refer to this documentation if you need help understanding the methods that are available to you.

Keep in mind that that your *HtmlValidator.isValidHtml* method should only use methods in the Queue interface, even though the Queue is implemented using a LinkedList.

It is okay if your *isValidHtml* method modifies the contents of the Queue that is passed as input, e.g. by removing elements.

Before You Submit

Please be sure that:

- your *HtmlValidator* class is in the default package, i.e. there is no "package" declaration at the top of the source code
- your *HtmlValidator* class compiles and you have not changed the signature of the *isValidHtml* method
- you have not created any additional .java files and have not made any changes to *HtmlTag.java* (you do not need to submit this file or *HtmlReader.java*)

1. Download the JUnit distribution at junit-dist.jar

   Follow this steps to add the library.

2. https://intellij-support.jetbrains.com/hc/en-us/community/posts/360009909039-How-do-I-add-a-build-Path-to-a-class-folder-
3. Download the tests at **homework2-tests.jar** and add it to the Eclipse project's build path as above.
4. Also download the test input files from homework2-files.zip Unzip this file on your computer and copy the seven .html files into your IntelliJ project; you should be able to drag and drop them right into IntelliJ. Make sure you put them in your project's root directory, as you did with the two .jar files.
5. Now run the tests by right-clicking *homework2-tests.jar* in IntelliJ to get the pop-up/context menu and selecting "Run As -->" and then "Java

Application." You should see the tests run in the console and it should tell you your score for this assignment, or "Great job!" if your score would be 100%.

Alternatively, if you would like to run the autograder from the command line, put the two .jar files and your .class files for this assignment in the same directory along with the .html files that you downloaded, and run:

Mac/Linux: java -cp .:junit-dist.jar:homework2-tests.jar Homework2Grader

Windows: java -cp .;junit-dist.jar;homework2-tests.jar Homework2Grader

This will add *junit-dist.jar* and *homework2-tests.jar* to the classpath and then run Java with *Homework2Grader* as the main class.