CSC207H: Assignment 2

Verifying HTML correctness, finding URLs

You will write a complete Java program to verify one aspect of HTML correctness, beginning with a webpage at a specified URL and proceeding to check other URLs that the first webpage links to.

The part of HTML correctness your program will check is that the *tags* match. A tag in an HTML document is a code that specifies non-textual information such as font, layout, or link URLs. Here is an example:
`<em>emphasized words</em>`
In the example, the words "emphasized words" are displayed in emphasized form — usually with an italic font — because tag `<em>` means "start emphasizing here". Tag `</em>` means "stop emphasizing here": the slash, `'/'`, generally means "stop", and tags usually appear in matching start-stop pairs.

HTML tends to be produced either by humans or by faulty computer programs, so it is often imperfect. This can cause security problems or heated arguments. What your program will check is that the tags in a specified web page are properly matched — except for tags that don't require a match, and tags that match themselves. Those details are specified later.

Tags can contain other information, such as font, colour and size. These are called *attributes* of the tag. For example,
`<font color="red">`
causes the output to begin being coloured red. More interestingly,
`<a href="http://www.codinghorror.com/blog/archives/001172.html">link text</a>`
causes the words "link text" to be a link to the URL http://www.codinghorror.com/blog/archives/001172.html. The '`a`' of this tag is short for "anchor". To make life more interesting for you, there may be any number of spaces before the "`href`" (but at least one), and there may or may not be spaces before or after the '='. In real HTML, other attributes may be used in the tag, either before or after "`href`" but we won't be considering this. If you wish to have your code handle other attributes that might appear in anchor tags you may, but our test cases will not include other attributes inside anchor tags. Last, '`a`' tags may not even have an "`href`" attribute.

Those hrefs are interesting to you because, after it has finished checking the current URL for tag matching, your program must follow all the links from this URL to check those URLs for tag matching too.

Checking the tags

Program structure

You must write and use these classes, which must all be in the package `a2`:

- `Tag`, with these methods:
  - `Tag(String s)`, the only constructor, which takes one argument: the tag string from opening '<' to closing '>'.
  - `toString()`, which returns the same string as the original tag string, with the original capitalization. You should trim any whitespace from the tag name.
  - `getName()`, which returns the name part of the tag, converted to lower case. For example, if the tag is `< x y="SomeThing">`, then the name returned is "`x`". Again, you should trim the returned string of leading and trailing whitespace.
  - `getAttributes()`, which returns the parts of the tag other than the name, converted to lower case. (These are, not strictly speaking, the attributes.) In our example, the string returned would be "`y="something"`".
  - `isClosingTag()`, which returns true or false depending on whether the name begins with '`/`'.
  - `isSelfMatching()`, which returns true or false depending on whether the name ends with '`/`' or the attributes are exactly "`/`".

  There may be other methods, if you like, but we will test only those listed.
- `TagIterator`, which is somewhat like `java.util.Iterator`, but with a few differences:
  - There is no generic type parameter, and `next()` always returns a `Tag`.
  - There is no `remove()` method.
  - There is one constructor, `TagIterator(String url)`. The `url` parameter is the URL that this `TagIterator` will iterate over.
  - There is a `urlList()` method that returns a `List<String>` containing all the hrefs found in anchor tags in the URL on which this `TagIterator` is based.
  - The methods may throw exceptions if that helps you (even though your program must catch all exceptions and print or display appropriate error messages instead of showing the exceptions to the user).

  Again, you may add other methods if you like, but we will rely on these.
- `A2`, with these methods:
  - `main()`, which expects the starting URL as the first "command-line" argument.
  - `checkURL(List<String> urls)`, a static method that removes the first element in `urls` and checks it to see whether its tags are properly matched. The method returns the empty string `""` if the tags do match, or an error message if they do not.
    The `checkURL()` method also adds to the list `urls` all the links that it finds in the URL that it checks.

Details: tag matching

- While checking for matching tags, you must detect the *first* error in the HTML and display an error message about it. Do not display any further error messages for that URL, even if there are more errors.
- Even if the HTML you are examining does not have properly matching tags, you must extract all the links you can from it and check them.
- In our test cases, hrefs will appear only as part of '`a`' tags, and no other attributes will be in '`a`' tags.
- An opening tag "`<x y="z">`" is closed by "`</x>`". That is, the closing tag does not need to mention the attributes of the opening tag. Also, you can assume there is no space between

the '`/`' and the rest of the tag name. The '`/`' is considered to be part of the tag name.

- Both tags and attribute names may be capitalized arbitrarily: "em" or "EM" or "Em" or "eM"; "href" or "HREF" or "HreF" or "hrEF" and so on. You should preserve the original capitalization, but check for matching with capitalization ignored.
- Tags must nest properly. A closing tag must match the most recent opening tag. For example, this
  `<a>Paul<b>Jim</b></a>`
  is legal, but this
  `<a>Paul<b>Jim</a></b>`
  is not legal.
- Some opening tags may or may not have closing tags: <area>, <base>, <basefont>, <body>, <br>, <col>, <colgroup>, <dd>, <dt>, <frame>, <head>, <hr>, <html>, <img>, <input>, <isindex>, <li>, <link>, <meta>, <option>, <p>, <param>, <tbody>, <td>, <tfoot>, <th>, <thead>, <tr>. There is no way to tell when reading one of these opening tags whether or not it will have a closing tag.
- A tag with a '`/`' following the tag name, for example "`<br/>`" or "`<br />`", is both an opening and a closing tag, and matches itself. You can assume that the author of the HTML has correctly chosen a tag that can legally close itself. Note that the closing '`/`' may be separated from the tag name by a space.
- The comment tag "`<!-- -->`" is special. In the middle may be any text at all, including other tags, and all of it must be ignored.
- Ignore tags that are not comments but have names beginning with "`!`".
- Tags may span several lines. That is, the opening '`<`' and the closing '`>`' may be on different lines, separated by one or many newline characters. Please omit the newlines from the tags you return, but not any other characters.
- You may assume that we will not put '`>`' inside the attribute values or forget the final quotations on the end of an attribute value.

Details: visiting the URLs

- You must complete checking one URL before starting on another.
- The user will specify a single URL. Check that one first, and then the URLs it links to directly — at a "distance" of 1 — and then the URLs at a link distance of 2 from the original URL, and so on. If you think of the set of URLs as a tree, with the links as edges, then you are visiting the nodes in *breadth-first* order: the root, then its children, then its grandchildren, and so on.
- Stop after visiting five URLs. (We may change that number; watch for announcements.)
- Here's how to get a BufferedReader from a URL:

```
new BufferedReader(new InputStreamReader(new URL(url).openStream()))
```

Suggestions on algorithms and data structures

To visit nodes in breadth-first order, use a queue to save the unvisited URLs. That is, keep a list, and add newly-discovered URLs to the end of the list.

The web is not a tree. To avoid infinite recursion, you must stop yourself from re-visiting previously visited URLs. Keeping a set of visited URLs and checking it before each visit is one way to solve this problem.

To check for matching pairs of nested tags, a stack is the obvious mechanism: push the opening tag, and pop it when you see the closing tag. Self-closing tags and tags with optional closers add interest, but not a great deal of difficulty.

You may want to define your own exceptions. It's very easy: extend the class `RuntimeException`, and implement the no-argument constructor and the constructor that accepts one String argument. Each constructor should just call the parent class's constructor with the same argument list.

Make sure, however, that the user never sees your exceptions. A diagnostic message about the URL is appropriate, but not a program crash.

Implementing the TagIterator class can be a bit more interesting than you might first imagine but not because of "trick cases" where attribute values include the symbols < or >. Our test cases won't do that and your code doesn't have to handle it. What makes the design of the TagIterator interesting, is that you need to allow the user to call `hasNext()` as many times as he likes (including zero times) before calling `next()` and still have both methods work properly. You also can't keep rereading the file from the URL. So, while you need to look ahead to determine the return value for `hasNext()` you can't count on the user even calling the `hasNext()` method between every call to `next()`. Think carefully about how to design your TagIterator class before you even start it on the computer.