Programming Assignment 3

DOM Tree

Posted Tue, Mar 3

Due Mon, Mar 23, 11:00 PM (WARNING!! NO GRACE PERIOD)

Extended deadline (with ONE time extension pass): Thu, Mar 26, 11:00 PM (NO GRACE PERIOD)

Worth 100 points (10% of your course grade)

In this assignment you will implement an HTML Document Object Model (DOM) Tree.

You will work on this assignment individually. Read DCS Academic Integrity Policy for Programming Assignments - you are responsible for abiding by the policy. In particular, note that "**All Violations of the Academic Integrity Policy will be reported by the instructor to the appropriate Dean**".

- You get ONE extension pass for the semester, no questions asked. There will be a total of 5 assignments this semester, and you may use this one time pass for any assignment except the last one which will NOT permit an extension. **A separate Sakai assignment will be opened for extensions AFTER the deadline for the regular submission has passed.**

- **IMPORTANT - READ THE FOLLOWING CAREFULLY!!!**

Assignments emailed to the instructor or TAs will be ignored--they will NOT be accepted for grading.
We will only grade submissions in Sakai.

If your program does not compile, you will not get any credit.

Most compilation errors occur for two reasons:

1. You are programming outside Eclipse, and you delete the "package" statement at the top of the file. If you do this, you are changing the program structure, and it will not compile when we test it.
2. You make some last minute changes, and submit without compiling.

To avoid these issues, (a) START EARLY, and give yourself plenty of time to work through the assignment, and (b) Submit a version well before the deadline so there is at least something in Sakai for us to grade. And you can keep submitting later versions (up to 10) - we will accept the LATEST version.

- Grading

Summary

You will write an application to build a Document Object Model (DOM) for a given HTML file, and process it with a set of given input commands that will transform the tree.

Document Object Model

The Document Object Model (DOM) is a *platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page.* (This is quoted from the W3C Document Object Model specification web page.)

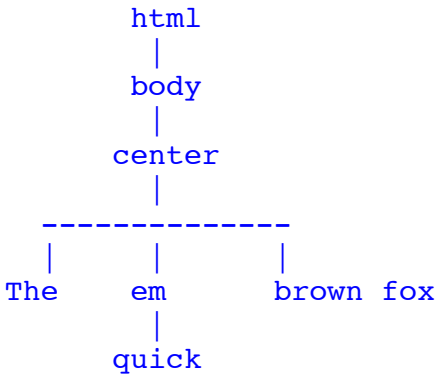
When you access a web page, the browser builds a DOM tree structure for the HTML content in the page. Consider the following HTML example:

```
<html>
<body>
<center>The <em>quick</em> brown fox</center>
</body>
</html>
```

which is rendered by a browser as:

The *quick* brown fox

The DOM tree for this document would look like this:



Note: There is a space character after "The", and before "brown", that is not visible.
In other words, the tree actually stores `"The "` and `" brown fox"` at those nodes.

You can see the DOM tree for any HTML document in the browser. In Firefox, go to Tools --> Web Developer --> Inspector. In Chrome, go to View --> Developer --> Developer Tools --> Elements. For the html example above, you will notice that under the root html node, in addition to the body node, there is also a head node (with nothing under it), which corresponds to the head tag. The head tag is not used in the example above, and is automatically supplied

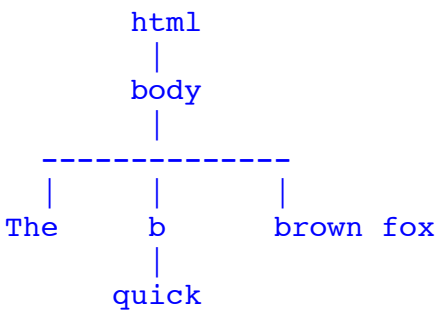
by the browser. For this assignment you can ignore the head tag.

The DOM tree helps developers build "smart" HTML pages by embedding, say, Javascript code in HTML, and making queries on the tree to extract information about the structure of the document, or transform it by changing its shape. For example, you may have a table in an HTML page, whose individual columns can be shown or hidden upon user request via Javascript code.

In the example above, you could replace the `em` tag with the `b` tag, and remove the `center` tag:

```
<html>
<body>
The <b>quick</b> brown fox
</body>
</html>
```

for which, the DOM tree would look like this:



which would be rendered as:

The **quick** brown fox

Bring up this sample try.html in your browser. This page has javascript code embedded in HTML, that replaces `em` with `b` or vice versa when you click a button. It does this by changing the DOM tree representation of the page. (Compare the DOM trees before and after the transformation.)

Restricted HTML

For this assignment, you will work with a restricted set of HTML tags. These are:

`html`	top level
`body`	second level
`p`	paragraph
`em`	emphasis (italics)
`b`	boldface
`table`	table
`tr`	row (within `table`)
`td`	column (within `tr`)
`ol`	ordered (numbered) list

`ul`	unnumbered list
`li`	list item (within `ol` or `ul`)

Moreover, the format of the HTML file itself will be restricted to the following:

- Every tag will appear on a line by itself. There will not be any tags not defined in the above list.
- There will not be any spaces between the tag name and the surrounding < and > characters
- Every HTML file will mandatorily have the `html` and `body` tags.
- None of the tags will have any attributes. For instance, there will NOT be tags like these

  `<p id="007"> or <table class="tbl">`
- The HTML will be well-formed, in that every tag will have a closing counterpart. For instance, every tag construct of the form `<b>` will be closed by `</b>`
- If there is a `table` tag in the document, there will only be one such tag. A table can have any number of rows, and a row can have any number of columns (but all rows will have the same number of columns).
- Nesting of the *same type* of tags is only permitted with `ol` and `ul` tags. That is, an `ol` tag may have another `ol` tag nested within it, and a `ul` tag may have another `ul` nested within it. But you may NOT have a `b` tag within a `b` tag, etc.

Examples

Following are some sample HTML pages. Click on a link to see the page, and view the page source to see the underlying HTML:

ex1.html	`p`, `em`, and `table` tags
ex2.html	`ol` and `ul` tags
ex3.html	Nested `ol` and `ul` tags

Tree Structure

Since the nodes in the DOM tree have varying numbers of children, the structure is built using linked lists in which each node has three fields: the **tag** (which can be an HTML tag or plain text), the **first child** (which is null if the tag is plain text), and the **sibling**, which is a pointer to the next sibling.

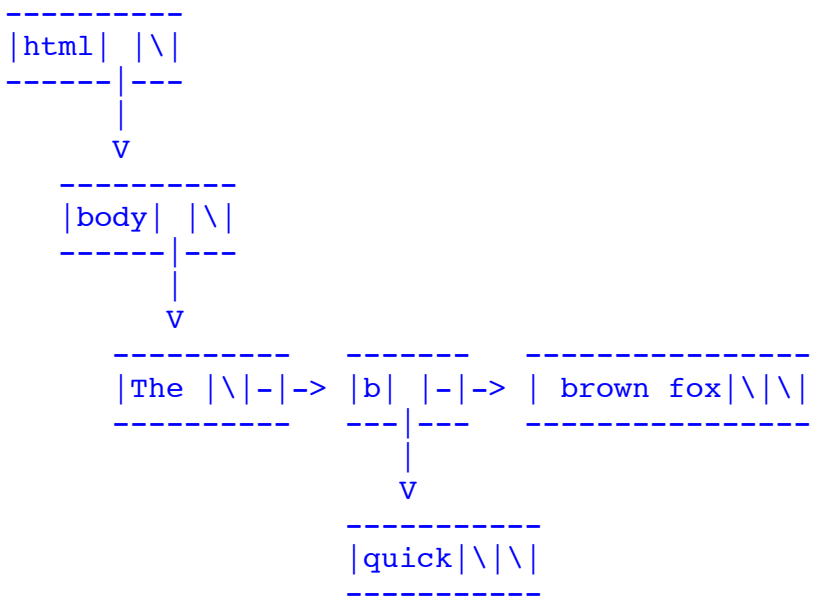
As an example, consider the following input HTML:

```
<html>
<body>
The
<b>
quick
</b>
 brown fox
</body>
```

```
</html>
```

(The third line is actually `"The "` - blank space after "The")

The DOM tree implementation for the above input HTML would look like this:



Note: Tree nodes containing tags do NOT include angle brackets with the tags.
So, if a node stores `em`, it stores the string "em", NOT the string "".
Also, closing tags ("/em") are NOT be stored in the tree.

Implementation

Download the attached `domtree_project.zip` file to your computer. DO NOT unzip it. Instead, follow the instructions on the Eclipse page under the section "Importing a Zipped Project into Eclipse" to get the entire project into your Eclipse workspace.

You will see a project called `DOM Tree Assignment` with the following classes:

- `structures.TagNode`
- `structures.Tree`
- `structures.Stack`
- `apps.DOM`

There are also a number of sample test files directly under the project folder (see the **Examples** section that follows.)

Note: You are not required to use the included `Stack` class. But if you do use a stack, you MUST use this class (not your own). When we grade your program, we will keep this `Stack` class in the project in case you are using it.

You will implement the following methods in the `Tree` class:

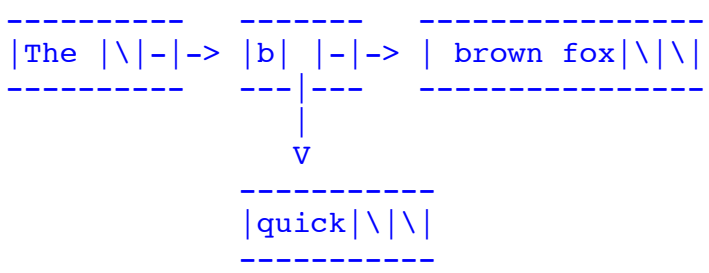
- **(20 pts)** `build`: Builds the tree from an input HTML file

- **(15 pts)** `replaceTag`: Transforms the tree by replacing *all* occurrences of a tag with another (e.g. replace `b` with `em`, or `ol` with `ul`. Only sensible replacements will be expected. For instance, `ol` cannot be replaced with `em`.)

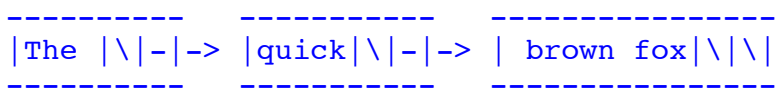
- **(30 pts)** `removeTag`: Transforms the tree by deleting *all* occurrences of a tag. There are two categories of tags that can be deleted:
  - Category 1: `p`, `em` and `b` tags: All occurrences of such a tag are deleted from the tree.

    Note: Text nodes should NOT be merged if the removal of a tag results in an unbroken string of text.

    For instance, removing tag `b` in this:

    should give this:

    In other words, the three nodes above should NOT be merged into a single text node.

  - Category 2: `ol` or `ul` tags: All occurrences of such a tag are deleted from the tree. In addition, all the `li` tags immediately under the removed tag are converted to `p` tags.

- **(15 pts)** `boldRow`: Transforms the tree by boldfacing a given row of a table. **This is done by bold facing the content of every column in the given row.** (It should NOT add a single bold face tag for the entire row.) Also see the header comment for this method in the code.

  You may assume that the row will not already have been boldfaced. Also note that the first row is row number 1 (not zero), the second row is row number 2, etc.

- **(20 pts)** `addTag`: Transforms the tree by adding a tag around all instances of a taggable word (case insensitive match). The added tag can be either `em` or `b`.

  A taggable word is a sequence of alphabetic letters that is not a part of another word, and may have one of the following punctuations AS THE LAST CHARACTER: period '.', comma ',', question mark '?', exclamation point '!', colon ':' and semicolon ';'.

  Note that the taggable word may be embedded within a longer piece of text (such as tagging "quick" in the "The quick brown fox"), and may occur multiple times in the containing text. In the following example the word "around" is tagged with `em`:

  ```
  What goes <em>around</em> comes <em>around!</em>
  ```

As another example, suppose you were asked to add a bold tag around "cow" in the following cases:

- `cow`
- `cow!`
- `cow?`
- `cow.`
- `cow;`
- `cow:`
- `COW`
- `cOw`

This would be the result after tagging, for each (the last two are case insensitive matches):

- `<b>cow</b>`
- `<b>cow!</b>`
- `<b>cow?</b>`
- `<b>cow.</b>`
- `<b>cow;</b>`
- `<b>cow:</b>`
- `<b>COW</b>`
- `<b>cOw</b>`

But you would NOT tag "cow" in these words: "cows", "cowabunga" (in both cases, not a word by itself, but part of a bigger word), "?cows", (not just letters, punctuation not last character) "cow?!!", (only one punctuation character at the end is acceptable), "cow's" (apostrophe is not a letter).

Examples of Transformations

Here are examples of applying transformations to some HTML pages, and the resulting HTML (look inside the source files as well):

Input	Transformation	Result
ex1.html	replace `em` with `b`	ex1tr1.html
ex1.html	delete `em`	ex1tr2.html
ex2.html	delete `ol`	ex2tr1.html
ex3.html	delete `ol`	ex3tr1.html
ex1.html	boldface 2	ex1tr3.html
ex2.html	add `em` around item	ex2tr2.html

Observe the following rules while working on `Tree.java`:

- You may NOT add any `import` statements to the file.
- You may NOT add any fields to the `Tree` class.
- You may NOT modify the headers of any of the given methods.
- You may NOT delete any methods.
- You MAY add helper methods if needed, as long as you make them `private`.

Submission

Submit your **Tree.java** file ONLY.

Grading

Since every method you will implement results in building or modifying the DOM tree structure, we will grade by examing the tree structure that results at the end of each of your methods, and comparing it with the expected structure.

We will NOT be looking at the HTML printout of the tree. (It is possible to get the same printout with an incorrect tree structure.) The HTML printout is only for your convenience, but you cannot rely on it to make sure the tree has been constructed correctly--for that, you may want to use the Eclipse debugger. There is comprehensive information on how to use the debugger in Eclipse under Help -> Help Contents -> Java development user guide -> Tasks -> Running and Debugging, and Help -> Help Contents -> Java development user guide -> References -> Views -> Variables View. Essentially, you need to know how to set breakpoints and examine variable values when the program stops at a breakpoint.