Homework # 7

due Monday, October 29, 10:00 PM

In this homework you will implement generic Stack and Queue ADTs using the standard List ADT. You will then use these classes to implement an iterator that finds words in an HTML document. Finally, you will write formatting code so that HTML can be presented in a graphical window to implement a simple browser. You will use a home-grown XML parser.

The deadline for this assignment is unusual: it is due Monday after mid-term week. This is so that the deadline does not interfere with studying for the mid-term examinations. However, the material covered in this Homework will be relevant for the examinations. We recommend you finish it before the mid-term examinations start.

1 Concerning the Stack and Queue ADTs

A stack provides the following operations: (where “E” represents the element type):

- **isEmpty()** Return true if the stack is empty.
- **E pop()** Remove and return the element at the top of the stack, throwing an exception if the stack is empty.
- **E peek()** Return (but don’t remove) the element at the top of the stack, throwing an exception if the stack is empty.
- **push(E)** Add an element to the stack.

A queue provides the following operations: (where “E” represents the element type):

- **isEmpty()** Return true if the queue is empty.
- **E remove()** Remove and return the element at the front of the queue, throwing an exception if the queue is empty.
- **add(E)** Add an element to the queue.

Often, a queue also has a method to return the front of the queue without returning it, but we don’t need it and so left it out of this homework.

These operations are simple to implement using a collection class such as **LinkedList**, which is what we recommend you do. The Java standard library provides a **Stack** class and a **Queue** interface that use the same names (along with other operations), but we will not be using them in this Homework.

2 Concerning XML Parsing

For this homework, you will use a simple home-grown XML parser that returns either strings or instances of class **Element**. You don’t need to know how it works, but you should read the XML Primer. You will need to use the **Element** class. All these classes are provided and documented on the class homepage.
3 HTML and XHTML

HTML is now over 25 years old; XHTML is a dialect of HTML that fits the XML standard. Few web pages however are actually XHTML; they have many inconsistencies. But the basic structure is similar. XHTML documents are XML documents with a set of special tags that are understood including the following:

- **h1,h2,h3,h4**: These mark “headers”
- **p**: This marks a paragraph.
- **ol,ul**: Ordered (numbered) and unordered lists. We treat `<ol>` the same as `<ul>`.
- **li**: List items inside the previous lists.
- **b,i**: These mark bold or italic text respectively.
- **em**: This marks “emphasized” text (italic normally, unless the context is italic, in which case it is not italic).
- **font**: Change the font. For this homework, we only handle the “size” attribute. If the size is relative (+n or -n), the point size change is twice the number (+3 means six points larger). If the size is absolute (n), the number will be the new point size (not multiplied by 2). This convention is arbitrary and not very defensible, but it’s the one to follow for this homework!
- **br**: A line break.
- **script,style**: Javascript and CSS code. We will ignore this.

4 Formatting Text

We provide a simple Word ADT that consists of some text and a font in which to render it. In order to figure out whether it can fit on the current line, we need to know its size. That requires the font metrics, which requires the graphic context.

You will implement a Line ADT that accepts words until the line is full. The line can then be rendered at which point we either fill (to get right justification) or not (ragged right). Right justification is difficult to make look good. TeX has a very sophisticated algorithm developed 40 years ago; Microsoft Word has traditionally been much worse and thus people using Word usually use a ragged right margin. In this Homework, you will see some of the problems.

5 Iterating over XML

In order to prepare for formatting, we need a way to extract words (strings with an associated font, see above) from an XML document. This is made more difficult because XML elements are recursive: an element can include other elements. When iterating over an XML document, therefore, we need to remember what elements remain to be looked at when we’re done with the current one. Furthermore, each element can have a whole list of nested elements. Therefore, your iterator over the tree will need to keep a stack of iterators of elements still to be processed.

This homework has you implement a demonstration class PreorderElementIterator. This class simply traverses elements so that you get used to using stacks to remember where you are in
a hierarchical structure. When checking whether there are more elements, you should pop off any iterators which are done. We have a test suite for this class. Then when you implement the full class, you will use the same ideas. (The class itself is not used; it’s only for learning purposes.)

When you start on the HTML word iterator, you will also need to handle the font. So it will also need to keep a stack of fonts so that the font can be changed when (say) handling an `<em>` element, and then restored when we are done with it. In other words, the current font (top of the stack of fonts) changes while traversing the structure. For example if one has a `b` element, then all the words below will be in a bold font (unless otherwise indicated). And then when it returns to the outer element it will pop the bold font.

As well as changing the font, several of the tags will cause a line break. This will be handled by having the iterator return a null word. The following tags insert line breaks: `<br>`, `<p>`, `<ol>`, `<ul>`, `<li>` (and after the line break, a word containing just a bullet character, Unicode U+2022), `<h4>`, `<h3>` (two line breaks), `<h2>` (three line breaks) and `<h1>` (four line breaks).

When a string is encountered, the string is broken up into individual words (at the spaces) and returned as separate words. As seen here (and with tags such as `<h1>` and `<li>`) we often need to “return” two or more words. Since of course `next()` can only return one word at a time, the remaining word(s) will have to be returned on later calls to next. For this reason, we are going to use a queue of words as a buffer between the producer, code that may generate multiple words at once, and the consumer which wants only one at a time to be returned by `next()`.

Some of the elements should be ignored completely, such as `<script>` and `<style>`. The contents should be unused. Don’t grab words from the script to format into the window!

## 6 Simple Browser

The text formatter will be used in a simple browser. It will read an HTML page into XML elements and then extract the words using the HTML iterator. The words will be placed in lines until they fill at which point they are drawn on the window. Normally, one would use margins, but for this Homework, for simplicity, we simply fill the entire window width. If the window is resized, the text is reformatted to fit the new size.

The browser will require the web page to be proper XHTML. Few web pages are compliant. For example, in the past the UWM main page had some mismatched element closing tags.

## 7 Files

We provide the following files in the Homework repository, among others:

```
src/\{TestStack.java,TestQueue.java,TestHTMLIterator.java\}  JUnit test suites for three of the four classes you are to write.

src/TestLine_OS.java  JUnit test suite for the Line ADT for operating system OS. Use the test suite appropriate to your OS.

src/uwm/cs351/SimpleBrowser.java  The main application. Run it (use “Run Configurations” and place the argument in the “Program Arguments” box) on file:lib/test.html or http://www.cs.uwm.edu/classes/cs351/index.html. You will need to set the “Working directory” option to be `${project_loc}` on the “Arguments” tab of the “Run Configuration” for SimpleBrowser.

lib/test.html  Test file.
```
8 What You Need To Do

You need to complete the following classes:

- edu.uwm.cs351.util.Stack, edu.uwm.cs351.util.Queue
- edu.uwm.cs351.Line, edu.uwm.cs351.PreorderElementIterator
- edu.uwm.cs351.HTMLWordIterator

The first two classes are very easy. The last class is the hardest to implement, but will be easier if you have done the preorder element iterator first.

9 Sample Output

Using Arguments = --height=600 file:lib/test.html

Test

First heading
A paragraph of words, some **bold**, others *italic*. We also have some *bold italic*. *Important: emphasis nests* unusually even **bold emphasis**. Sometimes however, the font becomes smaller, although **large** words can occur and we can temporarily **increase** the size or make it *very small*.

**Note:**
We have the following points:

- **Bullets can be italic**
- A plain text stands out.
- *But an italic test does not.*
- **Bold** text is obvious.

- Unfortunately numbered lists are hard to handle.
- So just treat them as though un-numbered.
- **Bold** text does not stand out.