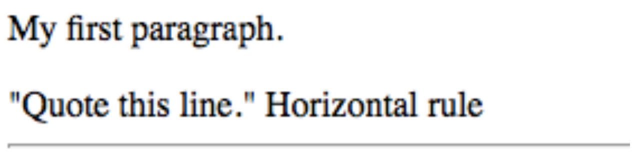
HTMLUtilities.java - Part 1

Objective: To use file IO and string manipulation to create a simple HTML tokenizer.

Background:
HTML, or Hypertext Markup Language, is the code behind every web page. HTML is made up of "tags", or formatting commands, inserted into a text document that specify how text and graphics will be formatted. Here is a simple example:

```
<html>
<body>
<p>My first paragraph.</p>
<!-- A comment -->
<q>Quote this line.</q>
Horizontal rule
<hr>
</body>
</html>
```



On the left is the HTML code and on the right is how it looks on a browser. The tags in the code are denoted by triangular brackets "<" and ">". Notice that tags, like `<html>` and `<p>`, are nested with closing tags `</p>` and `</html>`.

The job of the browser is to take the HTML code and turn it into the formatted page. This process takes several steps. The first step is to break the code into *tokens* that will be used in future steps. In this project we will create a simple tokenizer or *lexer* that will break the HTML code into an array of token **String**s.

To get started, we need to discuss what text makes a token. We are creating a "simple" tokenizer, so we can define it in much more constrained way than what is actually used. Even with a simple tokenizer, you will get a sense of what a real tokenizer will do.

Discussion:
If we use the simple example above and break the code on the left into an array of **String** tokens, this is what it would look like:

```
{ "<html>", "<body>", "<p>", "My", "first", "paragraph", ".", "</p>", "<q>", Quote", "this",
"line", ".", "</q>", "Horizontal", "rule", "<hr>", "</body>", "</html>" }
```

The first thing to notice is that spacing and line feeds are not tokenized. The next thing to notice is that tags, words, and punctuation are separate tokens. These tokens are fed into a page formatter which formats the page we see in the browser.

The input format of the HTML code does not influence the output format. For example, I could rewrite the code in the previous figure like this:

```
<html><body><p>My first paragraph.</p>
<!-- A comment -->
<q>Quote this line.</q> Horizontal rule <hr></body></html>
```

The tokenizer would gather the same tokens and the page formatter would produce the *same formatted output*. Since the HTML code is just a series of tokens in a text file, a tokenizer is fairly simple to explain and moderately challenging to write. This is a perfect project for our APCS A class!

The best way to approach a tokenizer is to use an algorithm that performs a character-by-character evaluation. Tokenizers collect characters, one at a time, and with the use of a "look-ahead" character it assembles each token. For example, let's consider the input string "Hello Good-bye". The tokenizer reads each character 'H', 'e', 'l', 'l', 'o' and assembles them into a token "Hello". The look-ahead will see a space character, determine the token is now complete, then add the token “Hello” to the token array. The tokenizer continues character-by-character until it gets the next alpha character ‘G’, then begins to assemble the next token of contiguous alphabetic characters. Once it has “Good-bye” (notice the hyphen is part of the word), it detects it is at the end of the input string and adds “Good-bye” to the token array. The token array {“Hello”, “Good-bye”} is returned.

Assignment:
Download **HTMLUtilities1.zip** from Mr Greenstein’s web site and unzip the file. It will create an **HTMLUtilities** directory in which you do all your work. The zip file contains the tester class **HTMLTester.java**, a starter **HTMLUtilities.java** file, and four HTML example files. **HTMLTester** requires the **FileUtils** class, so copy that file (or Geany “Save As…”) into the directory.

The majority of your work will be in the **HTMLUtilities** class and the **tokenizeHTMLString** method. You may also need to create private helper methods to perform tokenization.

1. Tokenize tags: In the **tokenizeHTMLString** method, add functionality to tokenize just the HTML tags. All tags start with a “<“ character and end with a “>” character. Ignore all text outside of the tags. For example, the tokenizer receives the input string:

   ```
   “<q>Quote this line.</q> Horizontal rule <hr></body></html>”
   ```

   and returns the array:

   ```
   { “<q>”, “</q>”, “<hr>”, “</body>”, “</html>” }
   ```

   Test your code using **HTMLTester** and input file **example1.html**. A sample run is below. Tokenized tags are highlighted in **bold**.

   ```
   java HTMLTester example1.html

   <!DOCTYPE html>

     [token 0]: <!DOCTYPE html>

   <html>

     [token 0]: <html>

   <body>

     [token 0]: <body>

   <p>My first paragraph.</p>

     [token 0]: <p> [token 1]: </p>

   <!-- A comment -->

     [token 0]: <!-- A comment -->

   <q>Quote this line.</q>

     [token 0]: <q> [token 1]: </q>

   <br>

     [token 0]: <br>

   Horizontal rule


   <hr>

     [token 0]: <hr>
   ```

```
</body>

  [token 0]: </body>

</html>

 [token 0]: </html>
```

2. Tokenize words: In addition to tags, add functionality to the **tokenizeHTMLString** method to tokenize words. A "word" is any contiguous group of alphabet characters (a to z or A to Z) including a single hyphen surrounded by alpha characters (like "Hello-World" below). Test your code using **HTMLTester** and input file **example2.html**. A sample run is below. Tokenized words are highlighted in **bold**.

```
java HTMLTester example2.html

<!DOCTYPE html><html><body>This is a paragraph: Hello-World.<br>

  [token 0]: <!DOCTYPE html> [token 1]: <html> [token 2]: <body> [token 3]: This [token 4]: is
  [token 5]: a [token 6]: paragraph [token 7]: Hello-World [token 8]: <br>

This is the second sentence like the first except longer.<br>

  [token 0]: This [token 1]: is [token 2]: the [token 3]: second [token 4]: sentence
  [token 5]: like [token 6]: the [token 7]: first [token 8]: except [token 9]: longer
  [token 10]: <br>

. . .
```

3. Tokenize punctuation: Add functionality to the **tokenizeHTMLString** method to tokenize punctuation characters. It will be helpful if you create an **isPunctuation** method to detect a punctuation character. Here is a list of possible punctuation characters:

```
 '.', ',', ';', ':', '(', ')', '?', '!', '=', '&', '~', '+', '-'
```

Often, punctuation is at the beginning or ending of words or tags. A sample run with **HTMLTester** and specific lines from **example3.html** are below. Tokenized punctuation are highlighted in **bold**.

```
<body>We like the number for pi 3.1415926 and so on.

  [token 0]: <body> [token 1]: We [token 2]: like [token 3]: the [token 4]: number
  [token 5]: for [token 6]: pi [token 7]: . [token 8]: and [token 9]: so
  [token 10]: on [token 11]: .


Stand-alone punctuation ; : . - can cause problems.

  [token 0]: Stand-alone [token 1]: punctuation [token 2]: ; [token 3]: : [token 4]: .
  [token 5]: - [token 6]: can [token 7]: cause [token 8]: problems [token 9]: .
```

Notice that numeric decimal points are tokenized as punctuation. We will handle numbers in the next step.

4. Tokenize numbers: Add functionality to the **tokenizeHTMLString** method to tokenize numbers. Numbers will appear in different forms:

53 3.1245 987.34 4e-2 6.0221409e23

There is also the negative form of each of these numbers:

-53 -3.1245 -987.34 -4e-2 -6.0221409e23

A sample run with **HTMLTester** and specific lines from **example4.html** are below. Tokenized numbers are highlighted in **bold**.

```
53        3.1245       987.34       4e-2         6.0221409e23

  [token 0]: 53 [token 1]: 3.1245 [token 2]: 987.34 [token 3]: 4e-2 [token 4]: 6.0221409e23

-53       -3.1245             -987.34             -4e-2        -6.0221409e23

  [token 0]: -53 [token 1]: -3.1245 [token 2]: -987.34 [token 3]: -4e-2 [token 4]: -6.0221409e23
```

Note: A hyphen ‘-‘ with a space before the number, like “- 2.34”, will be interpreted as two tokens: a hyphen and a number.