Web Image Scraper Assignment – Supplemental Information

How Are Web Pages Structured?

Web pages are written in HTML, and are made up of a series of potentially nested elements. **Elements** are indicated in HTML using text **tags**. You've probably seen this if you've ever clicked "view source" in your web browser. Here's a small example:

```
<html>
   <head><title>Hi World!</title></head>
   <body>
      <h1>This is a heading</h1>
      <img src="picture.jpg"/>
      <p>This is a paragraph with <a href="page2.html">a link</a> inside it!</p>
   </body>
</html>
```

Many tags (like h1 headings) have both begin and end tags to indicate grouping—sort of like the { } characters in Java. Some tags like img (for images) do not have end tags.

The string that immediately follows the `<` character is known as the "**tag name**". Thus `html, head, title, h1, img, p`, and `a` are the tags used in this example. The two most important tags for this assignment are the image tag (img) and the anchor tag (a). Image tags contain information about the image filenames used on a website. Anchor tags are used to create links to other HTML pages.

Some tags also have **attributes**, which are a bit like instance variables and values in object oriented programming. Attributes are denoted inside the begin tag for an element using the `attributeName=attributeValue` format. For image tags the `src` attribute specifies the filename for the image to be loaded. For anchor tags, the `href` attribute specifies the destination page for the link text inside of the anchor tag.

This information should suffice for what you need on the assignment, but in case you'd like to explore how HTML works in a bit more detail, see: https://www.w3schools.com/html/

How Can We Access Web Pages in Java?

As part of the starter code, we have provided a small library of classes that will help you access the HTML for a web page. It has some handy abstractions that allow you to perform queries on a website's elements without worrying about how they might be nested inside each other. You'll find these classes documented in the `scraper.utils` package.

Example 1: How many h1 elements are on a page?

```
Document page = new Document();
page.loadPageFromURL("http://www.unomaha.edu");
int numH1s = page.getElementsByTag("h1").size();
System.out.println("Number of H1 headings = " + numH1s);
```

Example 2: What is link text for first three anchors on the page?

```
Document page = new Document();
page.loadPageFromURL("http://www.unomaha.edu");
Elements anchors = page.getElementsByTag("a");
for (int i = 0; i < 3; i++)
   System.out.printf("Link %d text: %s\n", i, anchors.getNextElement().getInnerHTML());
```

To complete this assignment, you'll need to experiment with and use the methods of `scraper.utils.Document`, `scraper.utils.Elements`, and `scraper.utils.Element`. We encourage you to write some small programs with them first to understand how they work while looking at the HTML source of a website you're querying.

So What's the Strategy for Crawling for Images?

As described in the main assignment documentation, your solution will start by visiting an initial page specified by the client program. In the GUI application, the string entered in the text box will be used as this initial page value. You'll also specify a page depth that algorithm will explore to.

When you explore a page, you should first extract all of the image URLs directly found on that page in img tags. To this list you'll then add entries for any images found at the pages referred to in anchor tags on this page. You must avoid adding the same URL for a picture twice and you should not explore the same page more than once while you recurse.

For example, consider these three hypothetical pages:

http://cool.story.com/page1.html

```
<html>
  <head><title>Page1</title></head>
  <body>
     <img src="pic1a.png"/>
     <img src="pic1b.jpg"/>
     <a href=" http://cool.story.com/page2.html">Page 2</a>
     <img src="pic1c.png"/>
     <img src="pic1d.gif"/>
  </body>
</html>
```

http://cool.story.com/page2.html

```
<html>
  <head><title>Page2</title></head>
  <body>
     <a href=" http://cool.story.com/page3.html">Page 3</a>
     <img src="pic2a.png"/>
     <img src="pic2b.jpg"/>
     <img src="pic2c.png"/>
  </body>
</html>
```

http://cool.story.com/page3.html

```
<html>
  <head><title>Page3</title></head>
  <body>
     <img src="pic3a.jpg"/>
     <a href=" http://cool.story.com/page1.html">Page 1</a>
     <img src="pic3b.png"/>
     <a href=" http://cool.story.com/page2.html">Page 2</a>
     <img src="pic1d.gif"/>
     <a href=" http://cool.story.com/page4.html">Page 4</a>
     <img src="pic3f.png"/>
  </body>
</html>
```

Crawling at a depth of 0 means that you should not explore any further links found on the page, and only return the images found here. Thus, crawling page 1 at depth of 0 should generate this list of images:

http://cool.story.com/pic1a.png
http://cool.story.com/pic1b.jpg
http://cool.story.com/pic1c.png
http://cool.story.com/pic1d.gif

Crawling at a depth of 1 means that we should visit links found at the base page. Thus, crawling page 1 at a depth of 1 would generate the list below because page 2 is linked from page 1. Page 3 is not explored from page 2 because we are only supposed to go one level down in the list of page 1's links.

http://cool.story.com/pic1a.png	(everything from page 3)
http://cool.story.com/pic1b.jpg	
http://cool.story.com/pic1c.png	
http://cool.story.com/pic1d.gif	
http://cool.story.com/pic2a.png	(then everything from page 2)
http://cool.story.com/pic2b.jpg	
http://cool.story.com/pic2c.png	

If we crawl all the pages starting from page 3 with a depth of 1, we would visit every page since they are directly linked. We'd get a list in this order:

http://cool.story.com/pic3a.jpg	(everything from page 3)
http://cool.story.com/pic3b.png	
http://cool.story.com/pic1d.gif	
http://cool.story.com/pic3f.png	
http://cool.story.com/pic1a.png	(then everything from page 1, except for pic1d.gif since it was on page 3)
http://cool.story.com/pic1b.jpg	
http://cool.story.com/pic1c.png	
http://cool.story.com/pic2a.png	(then everything on page 2)
http://cool.story.com/pic2b.jpg	
http://cool.story.com/pic2c.png	
	(followed by nothing from page 4, since it doesn't actually exist)

In order to help you understand the process, try writing the list of images that would be extracted by the following:

- Explore at a depth of 2 starting at page2.html
- Explore at a depth of 0 starting at page3.html
- Explore at a depth of 3 starting at page1.html

Once you've got a handle on these examples, you should get started with predicting the output for some test cases based on the live demo webpages found online here:

http://loki.ist.unomaha.edu/~bdorn/csci1620/hw3testpage/

Use your browser's "View source" feature to have a look at the source code of these small sample pages.