Day 26 – Scanner Lab

Today, we are acquiring baseline knowledge which will be a foundational portion of the Final Project.

This work you do today, culminating in the code you write, needs to be functioning. It's a necessary first step toward completing the Final Project.

First, recall that we've used Java's Scanner class, in most of the code we've used in class. Mostly, to read user input.

For example, from Lab 01, when we input one floating point number at a time, and note that we prompt the user for what to input:

```
Scanner keyboard = new Scanner (System.in);
System.out.println ("Enter the first number: ");
x = Float.parseFloat( keyboard.nextLine() );
```

The input above was one number per line, so `x` is assigned after return is pressed.

Here is an example, from Lab 02, when we typed an entire line, but only processed one token at a time:

```
Scanner keyboard = new Scanner(System.in);
String[] words = new String[20];
int n = 0;
System.out.println("Enter a line of words
                then use ^D or ^Z to flag the end");
while (keyboard.hasNext())
{
    words[n++] = keyboard.next();
    // other lines of computation
}
```

In both of the examples above, the `Scanner` constructor is sent `System.in` as a parameter, meaning the console in Eclipse is where we type.

But if you check the documentation for `Scanner`, https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html, and scroll to the bottom, you will note that there are other possible types of input, not just the console, including `File`, `InputStream`, and `String`. At the top of that documentation, note that `Scanner` is in the `java.util.Scanner` library, which you need to import in order to use it.

Ultimately, we want to submit a URL to be scanned. While you type in a URL, ultimately the file that is processed by your browser is an ASCII text file, usually with an extension like `.html` or `.htm`.

How do we ask Java to process file as input, particularly one that isn't necessarily saved locally, but as a URL?

That would be using Java's `URL` class, https://docs.oracle.com/javase/7/docs/api/java/net/URL.html,

So in reading this, we will need two Scanner objects. The first is associated with the keyboard, where we will prompt for a starting URL. That input will be read as a `String`. This `String` will then be the input parameter to the constructor for a `URL` object. At the top of the documentation, note the library the `URL` class is in, which you will need to import. Then, scroll almost to the bottom of the page, looking for the `openStream` method.

What is the description of `openStream` ?

Opens a connection to this `URL` and returns an `InputStream` for reading from that connection.

Since this is a pretty common operation, the `openStream` method is a shorthand for the more general:
`openConnection().getInputStream()`

Finally, that brings us to the `InputStream` class, https://docs.oracle.com/javase/7/docs/api/java/io/InputStream.html.

In considering the documentation, how would you declare and instantiate an `InputStream` Object? What parameter would you send to the constructor?

It looks like it's a default constructor.

So it will look something like:

```
InputStream name-to-call-your-url-input-stream =
    new name-of-your-url-object.openStream();
```

Then this `InputStream` object will be sent to the constructor of your second `Scanner` object.

Finally, you can set up a while loop to iterate through each token.

The condition on the while loop can use `.hasNext()`, and then in the loop you can use `.next()` to read and then simply output each token.

At end of your `main` method, you will want to `.close()` this Scanner object.

If you view the page source for https://jcsites.juniata.edu/faculty/kruse/, you will see the first few lines are:

```
<!DOCTYPE doctype PUBLIC "-//w3c//dtd html 4.0
transitional//en">
<html>
<head>

  <meta http-equiv="Content-Type"
 content="text/html; charset=iso-8859-1">
```

And comparing it to the first few lines of output from your working Scanner lab code, you'll see each token:

```
token from the URL: <!DOCTYPE
token from the URL: doctype
token from the URL: PUBLIC
token from the URL: "-//w3c//dtd
token from the URL: html
token from the URL: 4.0
token from the URL: transitional//en">
token from the URL: <html>
token from the URL: <head>
token from the URL: <meta
token from the URL: http-equiv="Content-Type"
token from the URL: content="text/html;
token from the URL: charset=iso-8859-1">
```

So, each token is just a set of characters, and they are delimited by spaces.