Lab 6
Along Came a Spider¹

Due: Wed. 10/23 at 11PM (for Mon. aft. lab), Thurs. 10/24 at 5PM (for Mon. evening), or Thurs. 10/24 at 11PM (for Tues. aft.)

Your goal for this assignment will be to complete a program that collects information about the set of web pages found in the web site for CS 134 (or, with slight modification, any other such web site).

The good news: The lab handout is only 4 pages long and we’ve already written most of the code and will share it with you!

A snapshot of what the program’s interface will look like while it is running is shown below.

![Program Interface](image)

The text field near the top of the window is used to enter the URL of a web site from which the user would like the program to start searching. After entering an address in this field, the user should press the “Start At:” button. The program will then place the address entered in the text field into the “Visit http:...” button that appears two lines below the text field, enable this button, and also enable the “Fetch Pages Automatically” button.

Each time the user clicks on the “Visit http:...” button, the program will scan the HTML for the page whose address (officially called a Uniform Resource Locator or URL) is displayed in the button looking for links (i.e., `<a href=…URL… >` tags) and extracting the URLs for these links. It will maintain a list of all URLs found in this way. Once it finishes scanning the HTML for the page whose URL was displayed on the button, it will select a different URL that it has found but not yet scanned and replace the URL that had been displayed in the “Visit http:...” button with this new URL.

¹ While the title of this lab is derived from the nursery rhyme, and the name “web spider” used to describe a program that traverses the web of links that forms the World Wide Web, it seems important to note that it is also the title of a movie from 2001 in which the villain was a computer science teacher.
If the user instead presses the “Fetch Pages Automatically” button, the program will act as if the “Visit http:...” button were being pressed repeatedly, visiting pages continuously until the button is pressed again to stop this process.

When it finishes responding to a click of one of its control buttons, the program will display the information it has collected in the text areas that occupy most of its window. The upper text area will contain a list of all of the URLs that have been found. Each line will describe one page including a count of the number of links to the page the program has found. For each page whose HTML has been scanned, the program will also display a count of the number of outgoing links that were found while scanning that page. The lower text area will display a list of all of the different web server names found in the URLs the program has discovered. For each server, the program will indicate the number of pages the program has found that are stored on that server.

As a starting point this week, we will provide you with a complete Java program that you can download from the course website that already implements most of the functionality for the program we want you to submit!

What’s the catch?

We recognize that you have just finished two challenging labs and a CS 134 midterm coming up, and may have other mid-term demands. With this in mind, we have tried to design this lab so that you can focus your attention on the new programming topic we have just covered in class, recursion, while wasting as little effort as possible on other programming tasks. We have provided all the string processing code to look for URLs and the networking code to contact web servers and retrieve the HTML for a page. The program, however, depends on several recursive classes to manage lists of page URLs and server names. We have not completed all the methods for these classes. Your job is to provide this missing code. Recursion is an amazingly powerful programming technique and focusing your time and effort on understanding it will be time and effort well-spent.

Be Friendly!

For this lab, we encourage you to work with a partner on your lab exercise. We will do this for all of the remaining labs this semester. We believe that this can be valuable because most real programs are constructed by teams. Working with a partner should also decrease your workload while probably increasing the amount you learn. To make sure both partners get the most out of the lab by participating actively, we strongly suggest that from time to time you switch whose fingers are actually on the mouse and keyboard. Please refer to the Honor Code guidelines for “team programs” described on the course website, so you understand how the code applies to such assignments. If you do work with a partner, you should follow the instruction in the handout “Working With Partners in GitLab”. Doing this will allow you to share code through GitLab (evolene.cs.williams.edu) and will enable us to track who is working together so that students get appropriate credit when we grade the labs.

A Quick Tour

As explained above, to aid you in implementing this program, we will provide most of the required code. You should be able to implement the recursive classes and methods that are needed but have not been provided without understanding the code we have written precisely, but the following overview should provide you with a helpful sense of how our code functions. If you want to look at the code we have provided before you get to lab, you will find links to both the code for the individual classes on the Labs page of the course website. When you checkout your lab6 repository from evolve this week, it will contain this starter code.

First, a bit of terminology.

A web page address such as http://www.cs.williams.edu/~cs134/index.html is called a Uniform Resource Locator or URL for short. The machine to which a URL refers (www.cs.williams.edu in the example) is called the server or host. The files that describe the material that should be displayed when a web page is viewed are written using a language called Hypertext Markup Language or HTML.

To create a link to another page in the HTML that describes a page, one includes an A (for anchor) tag of the form <A HREF="some-URL">. Our program operates by scanning the HTML for one page using methods like
indexOf and substring to find URLs for other pages and then performing the same process on these other URLs. Such a program is called a web spider (or crawler) because it wanders around the links of the web.

From your point of view, the most important feature of our spider is that it maintains three lists:

- A list of all of the **pages** for which it has found URLs,
- A list of all of the **URLs** it has found for which it has not yet downloaded and scanned the HTML, and
- A list of all of the **web servers** included in the links it has found.

When the program is first run, all of these lists are empty.

When the user clicks the “Start At” button, the program adds the URL in the program’s text field to the list of all pages and the list of pages that it has not yet retrieved. It then displays the URL in the button displaying text starting with “Visit”.

Each time the “Visit” button is actually pressed, the program tries to fetch the HTML for the page referenced by the link displayed in the button. This is performed by the `visitOnePage` method of the `WebCrawler` class. First the HTML text for the web page is fetched. The retrieval of the HTML is performed by the `get` method of an object of a class we provide named `PageGrabber`. This method has been designed to handle the many forms of delays and errors that can occur when trying to fetch a document from a web server. In particular, the program may sometimes appear to be stuck for as long as a minute or two while `PageGrabber` is trying to contact a failed server, but if you are patient, it will probably recover. While it is still simple enough that you should be able to understand much of `PageGrabber`'s code, you should not need to examine this class to complete your assignment.

If the `PageGrabber` successfully retrieves a page, the program then scans the HTML searching for the string “href=” used in an HTML “A” tag to specify the URL associated with a link. The text following “href=” is extracted. If this URL is not already in the list of all URLs the program has found, it is added to that list and to the list of URLs whose pages still need to be fetched and scanned. This process is described by the `getUrls` method of the `WebCrawler` class.

When the “Automatic” button is pressed, the program creates a timer using a class provided by the Squint library named `PaceMaker`. As long as this timer is running, the program periodically executes the code in a method named “tick” that simulates the pressing of the “Visit” button. If the “Automatic” button is pressed again, the `PaceMaker` is stopped which terminates automatic page retrieval.

Three very simple classes are defined in the program to manage information about URLs, pages, and hosts.

The `Url` class is used to represent a single URL. There is actually a standard Java class named `URL` designed for this purpose, but using that class requires knowledge of exception handling, a topic we have not covered. The `Url` class we include behaves like the standard `URL` class except no exception handling is required.

Objects of the `PageInfo` class hold three pieces of information about a web page:

1. the page’s `Url`,
2. a count of the number of links to the page that have been encountered,
3. a boolean indicating whether the page’s HTML has been retrieved and scanned, and
4. a count of the number of links found within the page’s HTML when it was scanned.

Objects of the `HostInfo` class hold two pieces of information about a web server:

---

2 You might feel that we could visit more pages per second than we do. This is intentional: We want to avoid flooding the server with page requests from a program that may be malfunctioning!
1. the server’s name, and
2. the number of links referring to the server that have been encountered.

For each of the three types Url, PageInfo and HostInfo, the program also depends on an associated recursive class to maintain lists of items. These classes are cleverly named UrlList, HostList and PageList. We have provided you with a complete implementation of the UrlList class.

The PageList class included in the starter project is only partially complete. It contains definitions of all the required instance variables and constructors and of a critical method named get used to retrieve a page’s description from the list given the Url for the page. Several additional methods needed to collect the statistics the program is supposed to report about the pages found are incomplete. Templates for the missing methods are included so that the program will compile and run, but you will have to complete the bodies of these methods to complete this assignment. The comments included in the incomplete methods describe the functionality the code you write should provide.

The HostList class included in the starter project is just a template for code you must provide. We have included headers for the needed constructors and methods, but we have not included complete code for any of these methods and have not included needed instance variable declarations. As we did in PageList, we have included comments clearly indicating where the additional code you will write belongs and what it should do.

**Getting Started**

You should begin your work this week by checking out lab6 from https://evolene.cs.williams.edu in the usual manner, as described on the course webpage. Unlike previous weeks, the project you download in this way will contain code for several classes — some complete, some just templates that you must complete.

**Implementation Plan**

The code provided in this week’s starter project makes a detailed implementation plan unnecessary. We will, however, provide two suggestions.

We suspect that for most of you, it will be best to start with the task of writing the code needed to implement the HostList class and then moving on to complete the missing methods of the PageList class. At first, it might seem that working with the partially completed PageList class would be an easier undertaking than writing the HostList from scratch. Much of the code required for HostList, however, will be similar to the code we have already provided for PageList. Therefore, working on HostList will give you a better understanding of how PageList works. This understanding will prove helpful when you work on adding the missing functions to PageList.

Within PageList, we suggest you start work on the methods in the order they appear in the file we have provided: size, totalLinksTo, totalLinksFrom, and finally mostReferenced.

We will provide a working version of the completed program that you can run (but not see the code) as a link on the labs page of our web site. When you think your code is complete, you should probably compare its behavior to this sample solution.

**Finishing Up**

Make sure to take a final look through your code, checking its correctness and style. Check over the style guide accessible through the course web page and make sure you have followed its guidelines. Make sure you included your name and lab section in a comment in each class definition.

Now you should be ready to submit your program. You can find instructions describing how to do this on the “Labs” page of the course web site at

http://www.cs.williams.edu/~cs134/Labs.html