In this lab, you’ll begin using the Java API for getting webpages. This API greatly simplifies the task as we saw it in CS 226: get IP address, open connection, make query, and read response.

Web browser redux

One of the assignments in CS 226 was a primitive web browser that displayed the HTML of the webpage. It turns out that Java has an API that greatly simplifies this task. Here are the key classes and methods:

- `java.net.URL` is a class that represents a URL. You can create a URL from a string such as my webpage’s “http://faculty.knox.edu/dbunde”, extract the hostname (i.e. the “faculty.knox.edu” part) using `getHost`, and actually open a connection using `openConnection`.

- `java.net.URLConnection` is what you get back from `openConnection`. It represents a connection to query for that URL. For static webpages, you get query it about the header and/or use `getInputStream` to get an `InputStream` object from which to read the body of the response.

Use these classes to read a webpage and print it to the screen. Note that `InputStream` is the same type as `System.in` so you can create a `Scanner` to simplify reading from it. Even though I have pointed out the main features above, you should still refer to the Java API because these methods can throw a variety of exceptions based on many things that can go wrong while trying to read the contents of a URL.

Once you get the basic version done, see if you can identify links in the text, separating the URL and link text out from the message body. For example, the link

```
<a href="http://cs.knox.edu">Department of Computer Science</a>
```

corresponds to URL “http://cs.knox.edu” and link text “Department of Computer Science”. Store these in a list as they are found. Then, after printing the page text, print a list of these links and allow the user to select a number, going to the link corresponding to that number. Note that the two argument constructor for `URL` will help you with relative links (those that give a path relative to the current directory).