Project — Website Map

Introduction

The purpose of this assignment is to produce a program that will build HTML maps of websites. Your program must be a modification of A3Driver.java which continues to provide all the commands specified in the Assignment One handout. You will write an additional command called map which we will explain below.

Modelling URCLS: A Better Way

In A3 you were required to use an object to encapsulate a Title/URL pair. Each of these components was represented by a String. For this assignment you must represent the URL component by an object from the java.net.URL class.\(^1\) This is important when you want to read the contents of the webpage located at that URL. It means that you can use the openStream() method to get the InputStream ultimately required by the BufferedReader needed by LinkIterator. You will also find that you need the equals() method and the toString() method from this class.

The new command to add: map

map will take two arguments title and filename and ultimately produce an HTML webpage that represents a map of the website found at the webpage associate with title. We will refer to that webpage as the root webpage for the remainder of this handout. In the same style as the other typed commands from A1, the arguments for map will each be on a separate line in the input. Your map command does not need the URLList for anything except to determine the root webpage and does not change the URLList in any way.

A Small Example

Figure is a diagram of 5 webpages. The boxes and arrows show the links found on each page. There is other text in those pages but these are all the links.

Here is what the browser displays when you generate the map using http://a/x/ as the root webpage. Each of these should be displayed as a clickable link except the fifth one (http://a/x/dead.html) which will be discussed later.

http://a/x/
  http://a/x/b.html
    http://a/x/c/a.html
    http://a/x/c/d.html
  http://a/x/dead.html
  http://a/x/g.html

\(^1\)It is important that you read the documentation for class java.net.URL yourself in the Java APIs.
Figure 1: The Links on Five Connected Webpages.

There are lots of very important details to notice here. First we only include webpages in our map which are:

1. reachable directly from some other page already in our map and

2. *under* the URL specified by *title*

By restriction one, we mean that we include `http://a/x/c/a.html` because it is linked to by `http://a/x/c/d.html` which is included. By restriction two, we define a URL to be *under* a root URL if the second URL contains the root URL as a prefix. So, in our example all the URLs we put into our tree were required to begin with “http://a/x/”. If the map command were called specifying `http://a/x/b.html` as the home webpage then the only page in the map would be the home webpage since no other URLs begin with the String “http://a/x/b.html”.  

Secondly, notice that some of the URLs given in the links are relative. For example, inside the file `http://a/x/` the link `g.html` is a relative link to the file `http://a/x/g.html` and inside the file `http://a/x/b.html` the link `c/d.html` is a relative link as well. Because the fully qualified URL is under the root webpage these URLs are included in the map.

In A1 and A3 we placed very few restrictions on the URL’s that the user was allowed to type for the add command. For this project, the URL given for that command will be a fully specified (not relative) URL, with protocol http.  

Thirdly, notice in the example above that the parent directory was not included for `http://a/x/c/d.html` but it was included for `http://a/x/b.html`. This is because we are not building a tree of the directory

---

2 You may have realized that when a URL specifies a directory such as `http://a/x/` the actual file accessed by the browser or by `openStream is index.html` within that directory. In this case `http://a/x/index.html`. The same is true if you omit the final slash after the directory. In our case `http://a/x`. Don’t worry about this. In all our test cases we will not use `index.html` and we will include the trailing slash.

3 This means that the root webpage of your map will always have a fully specified URL.
structure. Rather we are producing a clickable list of links to all the files within that web-site. It is possible
that there is no html file at http://a/x/c/ and so we don’t want to include it in our list.

**Some Constraints on How You Accomplish the Task**

You must use a `LinkIterator` to retrieve the links from each page. You can use the one you wrote for A2
or if yours doesn’t work properly you can fix it by getting help from your TA or your professor.

You must use recursion to process the pages found by valid links. You do not need to use recursion
to iterate through the links on each individual page. You can have a loop which calls the `next()` and
`hasNext()` methods of your `LinkIterator`. Beware of cycles: It’s possible for two (or more) files to each
link to the other. Only insert and process a file if it hasn’t already been done. Notice the loops in our small
example. Be certain that your design will work on that example and not loop endlessly.

Some links will not lead to currently valid files. We call these *dead* links. For purposes of the project, a
link is dead if either the URL constructor throws a `MalformedURLException` or if the `openStream()` method
throws an `IOException`. In the final file your produce, dead links should not be clickable: just put them in
as bare text, without an anchor tag.

You should call your `main` class `Project`. We will be running them by hand, rather than with an
automarker.

**What Does the Output Look Like?**

Calling the `map` command should construct a webpage containing an HTML picture of the map (including
the dead links). It should use `<ul>` and `<li>` tags as part of the HTML. This means that for all the valid
links you must generate an `<a href="...">` tag. Use the `String` representation of the URL as the title
associated with the link.

The order in the tree is important: the URLs should appear alphabetically, as decided by `String.compareTo()`,
and subpages should be nested.

Here is the HTML fragment that produced the interesting parts of the webpage generated for our small
example.

```html
<ul>
  <li> <a href="http://a/x/">http://a/x/</a>
  </li>
  <ul>
    </li>
    <ul>
      <li> <a href="http://a/x/c/d.html">http://a/x/c/d.html</a>
      </li>
    </ul>
  </li>
  <li> http://a/x/dead.html
  <li> <a href="http://a/x/g.html">http://a/x/g.html</a>
  </li>
</ul>
```
A Big Hint

Think carefully about what data structure you can use to store this map as you are building it. Then later, once the map is completely built you can print it out in the required format. Notice that the final entries in our output are all in alphabetical order. It isn’t just the entries in a given subdirectory that are alphabetized but the entire list. 4

Testing

Test your project: for each class in your project, write a corresponding testing class with a main method. This could be huge, but we don’t intend it to be. For each class and method, demonstrate that it works in a “regular” (non-empty, non-boundary) case, and if the class contains a list or tree, include a test case demonstrating that it works when empty.

For every list or tree class, you must also test one method that throws an exception: call that method when you know it should throw an exception, and if it doesn’t happen you should report an error.

Note that this means you only need to write three tests for each class: one test case creates an empty instance and calls all the relevant methods (and checks their results), the second creates a “regular” instance and calls all the methods (and checks their results), and the third forces an exception. You can certainly include more testing if you like.

After you have testing classes written for each of the classes in the project, write a testing class called MetaTester whose main method just calls all the other main methods. With this “meta-main” class, you can choose to test individual classes just by commenting out various lines in the meta-main class, and you can also test all parts of the program at once.

In addition, you must write a document describing how you would thoroughly test the map command, including the contents of input files. For that testing document, please print your HTML input files so that several fit on a single page when printed. (Print the HTML code, not a browser view, so that we can tell which files are linked together.)

Note that since map is a typed command, your testing is not automated. The first page(s) of your testing document should contain an overview of the testing strategy: what cases are you testing? The next page(s) should contain a script that a tester would follow: it should explain exactly what they should type, what the name of the output file will be, and what the contents of that file should be. Your TA will be following this script; make it clear and concise.

JavaDoc

You must provide JavaDoc comments for every public method, including these tags: @param, @return, and @throws.

Bonuses

Please read this section on the original project handout if your code is working and you are finished your testing.

4You can use your A4 tree code to do this for you but you aren’t required to do this. In fact, because you don’t always want the parent nodes of files in subdirectories it may turn out to be harder to adapt your A4 code than to do it without using a tree at all.
Marking

This is a rough marking scheme to give you an idea of how much weight we will give each part. They are meant as a guideline only.

<table>
<thead>
<tr>
<th>Percentage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>25%</td>
<td>program design, including modularity and exception choices.</td>
</tr>
<tr>
<td>25%</td>
<td>correctness (as determined by a semi-automated marking process)</td>
</tr>
<tr>
<td>25%</td>
<td>comments (including JavaDoc and representation invariants when necessary)</td>
</tr>
<tr>
<td>25%</td>
<td>testing (including test case descriptions and the map testing document)</td>
</tr>
</tbody>
</table>

Warning about assignment size

This is as much as 2 times the size of the other assignments, although this will vary tremendously with the individual student.

What to submit

- **Electronically**: every Java file written for the project (including the testing), and the JavaDoc HTML files for all non-testing classes.
- **On paper**: printouts of your code, printouts of all the HTML produced by JavaDoc (printed from a web browser), and your testing document for map.