

COMP 123 — INTERNET SERVICES & PROTOCOLS

Kevin Jeffay

Spring 2004

Homework 2, January 14

Due: 2pm, January 21

More Baby-steps Towards the Construction of a Web Server

In this assignment you will extend the program you developed for Homework 1 to begin doing something useful with input lines that are valid HTTP GET requests. Specifically, you will extend your HW1 program to determine the file type (the “MIME extension”) of the filename specified in the Request-URL token of a GET request and to perform an action based on the type.

If the filename ends in either of the strings “.txt”, “.htm”, or “.html” then your program should attempt to open the specified file, read successive lines from the file, and output these lines to standard output (*i.e.*, the screen). (For this assignment you may assume that any file having a file name ending with any of the above extensions contains only ASCII text lines with the normal line termination character sequences.) The test for the file extension should be case insensitive and hence any uppercase or upper/lower case variant of the above file extensions is acceptable. For example, for the GET request

```
GET /foo/bar.html HTTP/1.0
```

your program would open the file `foo/bar.html` and write the contents of the file to standard output.

The file name represented by the Request-URL is to be interpreted relative to the current directory in which your program is executing. That is, for the Request-URL “`/foo/bar.html`,” your program should attempt to open the file `foo/bar.html` in the current working directory.

If the file name does not end in one of the extension strings listed above (or has no extension), the following error message should be output to standard output:

```
501 Not Implemented: <Request-URL>
```

where “`<Request-URL>`” is the Request-URL from the GET request. If the Request-URL references a file that does not exist, the following error message should be output to standard output:

```
404 Not Found: <Request-URL>
```

For all other errors encountered in reading the file, simply output to standard output the string:

```
ERROR: <Java error message>
```

where “`<Java error message>`” is the error message string generated by the Java `IOException` class.

To read the contents of a file in Java use a `BufferedReader` on an `InputStreamReader` on a `FileInputStream`. (Note that this assignment will be quite easy if you follow the good programming practice of reviewing the definitions for the classes you use.)

Remember that you are *extending* the operation of your program from Homework 1. That is, your program should continue to recognize ill-formed GET request lines and output the error messages described in Homework 1. Your program should attempt to process a Request-URL only if it has recognized a GET request as being valid. Your program should continue to output each request line and the decomposition of each valid GET request. Thus for the get request above your program should output:

```
Method = GET
Request-URL = /foo/bar.html
HTTP-Version = HTTP/1.0
<line 1 of file foo/bar.html>
<line 2>
...
<last line of file>
```

As in Homework 1, your program should terminate when it reaches end-of-file on standard input (*i.e.*, when *control-D* is typed from the keyboard under UNIX or *control-Z* on Windows). Also as in Homework 1, your program must not output any user prompts, debugging information, status messages, *etc.*

Grading

As per the instructions distributed with Homework 1, create a subdirectory named HW2 within the directory `~/comp123/submissions`, and place the final version of your program in this directory before the due date. Please give your final program the name “Echo.java.”

Since we’ve had a change of TA in the class, you need to authorize the second TA to access your files. To do this execute the following command before starting the assignment:

```
fs sa ~/comp123/submissions alok read
```

As before, your program(s) should be neatly formatted (*i.e.*, easy to read) and well documented. In general, 75% of your grade for a program will be for correctness, 25% for “programming style” (appropriate use of language features, including variable/procedure/class names), and documentation (descriptions of functions, general comments, use of invariants, pre- and post conditions where appropriate).