Assignment 2

Computer Science 441

Due: 23:55, Friday March 1, 2024

Instructor: M. Ghaderi

1 Objective

The objective of this assignment is to practice network programming and learn about application layer protocols. Specifically, you will implement a program to download a web object specified by its URL .

2 Specification

2.1 Overview

In this assignment, you will implement a client program called `WebClient` to download web objects from the Internet. You have to write code to establish a (secure) TCP connection to a given server and send and receive HTTP messages. Your program should be able to download both binary and text objects. All downloaded objects are stored locally in the same directory where your program is running. `WebClient` should support URLs based on HTTP and HTTPS.

2.2 Implementation

A high-level description of the internal operation of `WebClient` is presented in Program 1.

Program 1 `WebClient`

Input Parameter: `url`

```
 1: Parse url to extract protocol, server name, and object path
 2: if protocol == HTTP then
 3:     Establish a regular TCP connection to the server
 4: else
 5:     Establish a secure TCP connection to the server
 6: end if
 7: Send a GET request for the specified object
 8: Read the server response status and header lines
 9: if response status == OK then
10:     Create a local file with the object name
11:     Read the response body from the socket and write to the local file
12: end if
13: Clean up (e.g., close the connection)
```

The client program should operate in non-persistent HTTP mode. This means that once the object is downloaded, the client closes the underlying TCP connection. The program takes as input the URL of the object to be downloaded. The input URL is a properly formatted fully qualified URL with the following syntax:

```
protocol://hostname[:port]/pathname
```

where,

- `protocol` is either HTTP or HTTPS.
- `hostname` is the name of the web server.
- `[:port]` is an optional part which specifies the server port. If no `port` is specified in the URL, use port 80 for HTTP and port 443 for HTTPS requests.
- `/pathname` specifies the path of the requested object on the web server.

Note that the `protocol` and `hostname` fields in a URL are not case sensitive, *e.g.*, both `http` and `HTTP` indicate the same protocol. Your program should consider this when checking for HTTP or HTTPS protocol in the input URL. However, the `pathname` field is case sensitive, and thus the path name used when constructing a `GET` request must exactly match the path name specified in the input URL.

2.3 Establishing TCP Connection

If the protocol is `HTTP`, a regular `Socket` can be used to establish a plain text TCP connection with the server. Otherwise, *i.e.*, if the protocol is `HTTPS`, you should establish a secure connection to the server. In Java, `SSLSocket` class can be used for this purpose. This class extends the `Socket` class, and as such provides the same API for socket programming, namely the familiar methods `getInputStream()` and `getOutpuitStream()`. Class `SSLSocket` adds security on top of the regular `Socket` by encrypting the data before sending it through the socket (among other things). The rest of the implementation of your program is independent of whether the established TCP connection is secure (via `SSLSocket`) or not (via `Socket`).

2.4 Constructing GET Request

To be compliant with what most web servers expect from a well-behaving web client, when constructing the GET request for the object, include the following header lines:

- `Host: <hostname>` – the host name of the server
- `Connection: close`

For the protocol version, you can specify `HTTP/1.1`.

2.5 Sending GET Request

To send a GET request, you should establish a TCP connection to the server on the appropriate port. Then create a request using the the object path name and send the request to the server. For writing ASCII formatted data to a socket, *e.g.*, HTTP headers, you can create a string object and then call `String.getBytes("US-ASCII")` to convert the string to a sequence of bytes that can be written to the socket. Make sure to **flush** the output stream so that the data is actually written to the socket.

Once the request is submitted, start reading the server response from the socket. The local file name of the downloaded object should match the file name specified by the URL. For example, if the input URL is

```
http://www.example.com/files/test.pdf
```

then the object should be saved in the file `test.pdf` in the working directory of the client program.

2.6 Reading Server Response

You may use method `InputStream.read()` on the socket input stream to read the status line and header lines byte-by-byte, as you are scanning for the character sequence `"\r\n"` to identify each line. Once you have read the entire status/headers, you can switch to chunk-by chunk reading using `InputStream.read(byte[])` until the socket's input stream returns an end of file. *Of course, there is no restriction in the type of the Java IO stream you can use in this assignment. Feel free to use whatever stream that better fits your design.*

3 Software Interfaces

3.1 Method Signatures

The required method signatures for class `WebClient` are provided to you in the source file `WebClient.java`. There is only one method that you need to implement, namely method `getObject()`. Refer to the Javadoc documentation provided in the source file for more information on this method.

3.2 Exception Handling

Your implementation should include exception handling code to deal with all checked exceptions in your program. Print exception messages (the stack trace) to the standard system output.

3.3 Running Your Code

A driver class named `ClientDriver` is provided on D2L to demonstrate how we are going to run your code. Read the inline documentation in the source file `ClientDriver.java` for detailed information on how to use the driver class. There are also a number of test URLs in the driver class that can be used to test the functionality of your program, although you should be able to use any web server (*e.g.*, `python -m http.server`) for testing purposes.

3.4 Console Output

Add print statements (using `System.out.println`) in your code to print the following information on the console:

1. HTTP request (request line and header lines)
2. HTTP response (status line and header lines only)

Do not directly print anything else to the console beyond exception messages and the above information. For debugging purposes, you can use the global `logger` object defined in the driver class, whose level can be set using the command line option `-v`. Refer to the driver class source file for more information. The logger can be used to write messages to console during code development.

Restrictions

- You are not allowed to modify the method signatures provided to you. However, you can implement additional methods and classes as needed in your implementation.
- You are not allowed to use package `java.net.http`, and `URL`, `URI`, `URLConnection` classes or their subclasses for this assignment. Ask the instructor if you are in doubt about any specific Java classes that you want to use in your program.