Assignment 2
Computer Science 441
Due: 23:55, Friday October 21, 2022
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 an HTTP client program to download a web object specified by its URL.

2 Specification

2.1 Overview

In this assignment, you will implement an HTTP client called `HttpClient` from scratch. You have to write code to establish a 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.

2.2 Implementation

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

Program 1 `HttpClient`

Input Parameter: `url`

```
1: Parse the input url to extract server address and object path
2: Establish a TCP connection with the server
3: Send a GET request for the specified object
4: Read the server response status and header lines
5: if response status is OK then
6:     Create the local file with the object name
7:     while not end of input stream do
8:         Read from the socket and write to the local file
9:     end while
10: end if
11: Clean up (e.g., close the streams and socket)
```

The client program `HttpClient` 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 URL with the following syntax:

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

where,

- `[:port]` is an optional part which specifies the server port. If no `port` is specified, use the default port 80

- `[pathname]` is an optional part which specifies the object path on the server. If no `pathname` is specified, use the default name `index.html`

2.3 Sending GET Request

To send an HTTP GET request, you should establish a TCP connection to the server on the specified port number. Then create a request using the `pathname` 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 HTTP 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://ict746x.cs.ucalgary.ca/files/medium.pdf
```

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

2.4 Constructing GET Request

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

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

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

2.5 Reading Server Response

You can use method `InputStream.read()` on the socket input stream to read an array of bytes that can be converted to a String object for parsing. Remember that each line of the response header is terminated with character sequence `"\r\n"`. Once you have read the entire header, you know the length of the body of the response, and subsequently can use method `InputStream.read(byte[])` to read the body of the response.

3 Software Interfaces

3.1 Method Signatures

The required method signatures for class `HttpClient` are provided to you in the source file `HttpClient.java`. There is only one method that you need to implement, namely method

`get()`. 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 `HttpDriver` is provided on D2L to demonstrate how we are going to run your code. Read the inline documentation in the source file `HttpDriver.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 with a web server on `ict746x.cs.ucalgary.ca`, although you should be able to use any HTTP 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.

3.5 Design Document

Prepare and submit a design document (one page) to describe how you read and parse the server response. Explain what stream you used and how you read header lines and the object body. Follow the design document formatting requirements described on D2L.

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.