Assignment 2: Ad blocker using Java Sockets

The goal of this assignment is to gain experience in application layer network programming through Java Sockets [1] and get familiarised with the basics of distributed programming. Specifically, this assignment will help to understand the Hypertext Transfer Protocol (HTTP), which is one of the widely used protocols on the Internet, and the operation of adblockers.

Assignment Description

The assignment consists of two parts to develop a program in the classic client/server paradigm. In the first part, you will implement the adblocker: an HTTP client that can request information from an HTTP server, filtering out advertising content on a webpage.

In the second part of the assignment, you will build an HTTP server that can store and serve information, such as a web page.

This exercise illustrates that correctly following protocol standards allows your programs (i.e., user and server) to interact with each other but also with other people's programs using application layer networking concepts.

Part 1 - HTTP Client

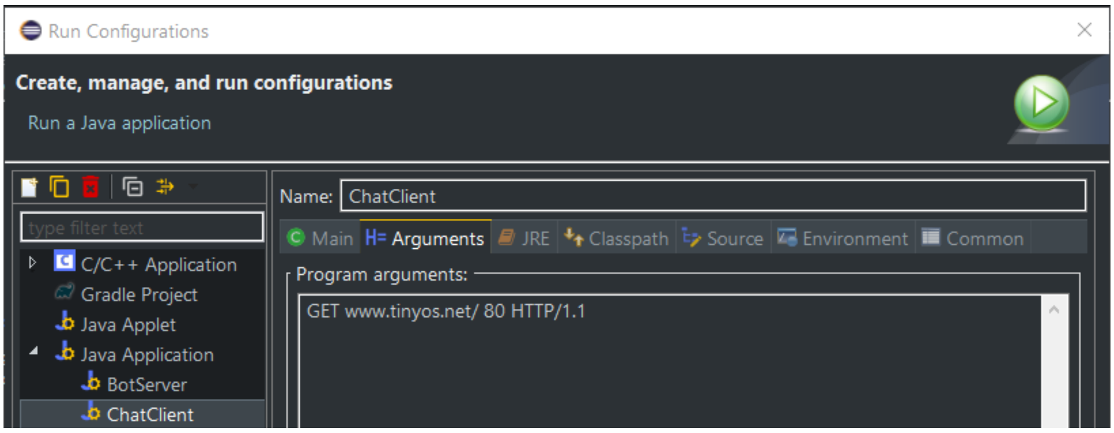
In the first part, you will build an HTTP client that communicates with a HTTP server that serves information such as web pages for end users. The HTTP client program should support HTTP version 1.1. For your assignment, you should implement a HTTP client, which you should be able to start either through the command line or in Eclipse.

For the command line, it should accepts the following arguments in the command line:

```
CommandLine$ ChatClient HTTPCommand URI Port
```

In the above line, the ChatClient denotes the name of your java executable. The HTTPCommand refers to HEAD, GET, PUT or POST. You can use any URI's such as http://www.example.com or http://www.google.com to test your client implementation. And, the default port number of 80 should be used to connect with the server.

For Eclipse, you can specify command line arguments in "Run Configurations". An example is shown below:



Name: ChatClient

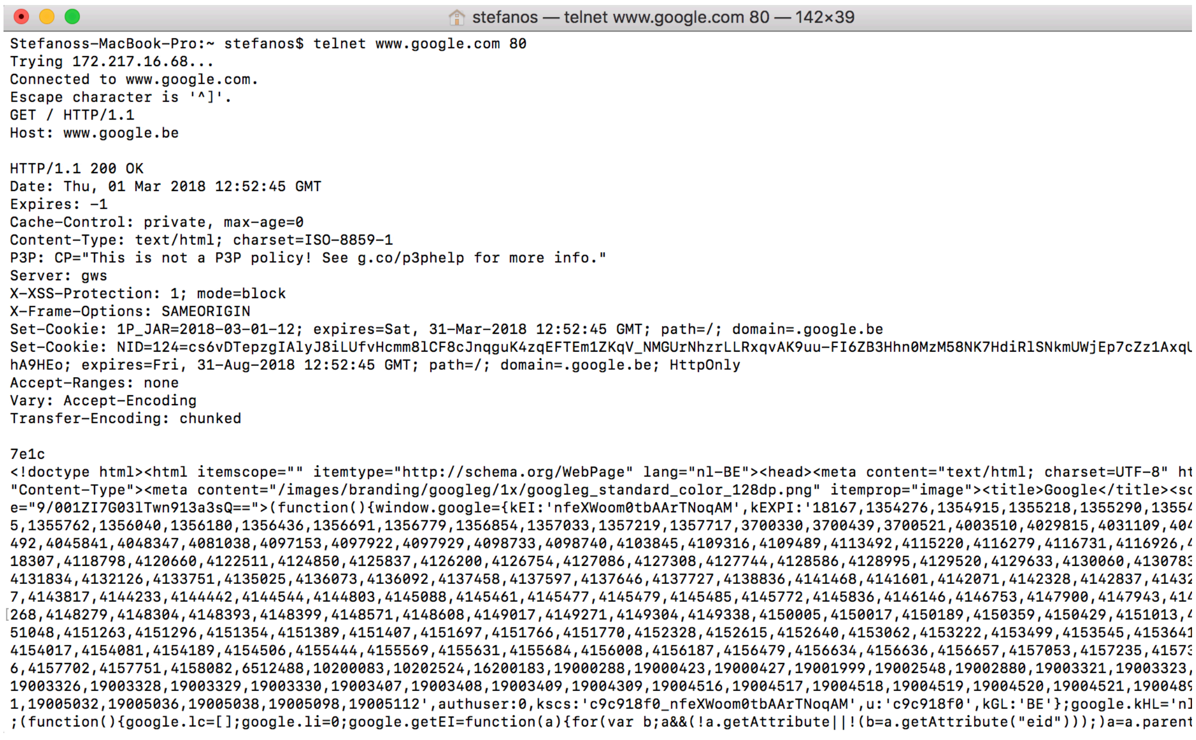
Program arguments:

```
GET www.tinyos.net/ 80 HTTP/1.1
```

After each request, your client should display the response received from the HTTP server on the terminal and also store the response in an HTML file locally.
The HTTP server can respond with either ‘Content-Length’ or ‘Transfer-Encoding: chunked’ headers. We expect your client to be able to support **both**.
When you retrieve a webpage from the server, you should scan the HTML file and check for embedded objects such as images. If you find an embedded object in the HTML file, you should use the GET command to retrieve those objects as well. In order to reduce the complexity, we do not expect you to retrieve all types of embedded objects. During the demonstration, you should at least retrieve image files from a HTTP server. The retrieved images should be stored locally.

For PUT and POST commands, your user should read a string from an interactive command prompt and send that onwards. These two commands will be tested with your HTTP server program.

For debugging purposes, we strongly recommend to use “telnet” [6]. Telnet enables you to connect to a remote server and issue HTTP commands through your terminal, where you can view the HTTP response of the server. An example is shown below:



```
Stefanoss-MacBook-Pro:~ stefanos$ telnet www.google.com 80
Trying 172.217.16.68...
Connected to www.google.com.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.google.be

HTTP/1.1 200 OK
Date: Thu, 01 Mar 2018 12:52:45 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2018-03-01-12; expires=Sat, 31-Mar-2018 12:52:45 GMT; path=/; domain=.google.be
Set-Cookie: NID=124=cs6vDTepzgIAlyJ8iLUfvHcmm8lCF8cJnqguK4zqEFTEm1ZKqV_NMGUrNhzrLLRxqvAK9uu-FI6ZB3Hhn0MzM58NK7HdiRlSNkmUWjEp7cZz1Axql
hA9HEo; expires=Fri, 31-Aug-2018 12:52:45 GMT; path=/; domain=.google.be; HttpOnly
Accept-Ranges: none
Vary: Accept-Encoding
Transfer-Encoding: chunked

7e1c
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="nl-BE"><head><meta content="text/html; charset=UTF-8" ht
"Content-Type"><meta content="/images/branding/googleg/1x/googleg_standard_color_128dp.png" itemprop="image"><title>Google</title><sc
e="9/001ZI7G03lTwn913a3sQ==">(function(){window.google={kEI:'nfeXWoom0tbAArTNoqAM',kEXPI:'18167,1354276,1354915,1355218,1355290,1355
5,1355762,1356040,1356180,1356436,1356691,1356779,1356854,1357033,1357219,1357717,3700330,3700439,3700521,4003510,4029815,4031109,40
492,4045841,4048347,4081038,4097153,4097922,4097929,4098733,4098740,4103845,4109316,4109489,4113492,4115220,4116279,4116731,4116926,
18307,4118798,4120660,4122511,4124850,4125837,4126200,4126754,4127086,4127308,4127744,4128586,4128995,4129520,4129633,4130060,413078
4131834,4132126,4133751,4135025,4136073,4136092,4137458,4137597,4137646,4137727,4138836,4141468,4141601,4142071,4142328,4142837,4143
7,4143817,4144233,4144442,4144544,4144803,4145088,4145461,4145477,4145479,4145485,4145772,4145836,4146146,4146753,4147900,4147943,41
268,4148279,4148304,4148393,4148399,4148571,4148608,4149017,4149271,4149304,4149338,4150005,4150017,4150189,4150359,4150429,4151013,
51048,4151263,4151296,4151354,4151389,4151407,4151697,4151766,4151770,4152328,4152615,4152640,4153062,4153222,4153499,4153545,4153641
4154017,4154081,4154189,4154506,4155444,4155569,4155631,4155684,4156008,4156187,4156479,4156634,4156636,4156657,4157053,4157235,4157
6,4157702,4157751,4158082,6512488,10200083,10202524,16200183,19000288,19000423,19000427,19001999,19002548,19002880,19003321,19003323,
19003326,19003328,19003329,19003330,19003407,19003408,19003409,19004309,19004516,19004517,19004518,19004519,19004520,19004521,19004
1,19005032,19005036,19005038,19005098,19005112',authuser:0,kscs:'c9c918f0_nfeXWoom0tbAArTNoqAM',u:'c9c918f0',kGL:'BE'};google.kHL='n
;(function(){google.lc=[];google.li=0;google.getEI=function(a){for(var b;a&&(!a.getAttribute||!(b=a.getAttribute("eid")));)a=a.parent
```

Part 2 - HTTP Server

Your HTTP server, should host a simple web page on your local machine. This server should be multi-threaded to support multiple clients at the same time. The server should support the following client operations: HEAD, GET, PUT and POST.

A user should be able to retrieve the web page hosted on your server. The user can either be your HTTP Client program or any third party clients such as Firefox or Chrome. If you have followed the protocol standards correctly, any user will be able to interact with your server.

For the demonstration, we expect your server to host a web page that can be retrieved using your HTTP Client program **and** one browser (i.e. Chrome, Firefox, Safari) of your choice. If your HTTP Client program retrieved the web page and the associated embedded objects correctly, then any third party clients such as Firefox or Chrome should be able to render it.

In the case of PUT and POST commands, the HTTP server should store the data received from clients in a text file, stored in the same directory.
For the PUT command, the user input should be stored in a new text file on the server.
For the POST command, the user input should be appended to an existing file on the server. If the file does not exist, then the file should be created.

As your server needs to support HTTP version 1.1, it should use persistent connections and support the *if-modified-since* HTTP header. Note that the host header field is mandatory for HTTP version 1.1.

Finally, you should at least support the following status codes on your server:

200 OK
404 Not Found
400 Bad Request
500 Server Error
304 Not Modified

Along with the status codes, the server should send the date, content type and content length headers to the client.
The server should respond with the "400: Bad Request" status code when the HTTP Client does not include the host header in its request for HTTP version 1.1.

Part 3 - Implementing Ad blocker

Ad blockers function as a blacklist which disables certain ad scripts and prevents the browser from retrieving ads by looking for specific elements in the html code (like iframe, image tags, etc.) while maintaining the original layout once the ad is removed.

In the third part of your assignment you need to extend your HTTP client by adding simple ad blocker functionality in order to filter out the ads from the retrieved webpage.

For simplicity, we have provided you with a simple html webpage with four embedded images (webpage.zip), which your HTTP server will serve to your HTTP client upon request.
Images: ads1.png, ads2.png, ads3.png are ads and should not be downloaded when requesting the webpage.

You can be creative with how you render the retrieved webpage (i.e. by adding a simple HTML element where the ad would have been or your own image).

General Guidelines

You implement this assignment in Java but should mainly use the Sockets package to complete the assignment.
You are not allowed to use the HTTPURLConnection package for this assignment, or external third-party packages.URI parsing may be done using the URI library of Java. However, you should not use the URL library to connect to a HTTP server. It is sufficient to support the URI of the format www.example.com instead of http://www.example.com.
You can choose any parser to scan the HTML files, but you are not allowed to use any API's from the parser library to retrieve the embedded objects.
The parser should be used only to detect the embedded object tags and the associated URIs.
You should use GET method to retrieve the embedded objects from the HTTP server.
We recommend the Java IDE Eclipse (https://www.eclipse.org/).
You should document your code. During the evaluation, we will go through your code and may ask you to explain how a certain method works in your code or to demonstrate certain functions.
You should be familiar with the HTTP protocol. Questions can be asked about this.
For HTTP/1.1, both the HTML content and the embedded objects should be received using the same connection.Your HTTP Server should handle multiple clients at the same time.

Create multiple tabs in your web browser to test this with your HTTP server.

Telnet[1] can be used to test and understand the HTTP commands. You can use telnet as a debugging tool during your implementation. For more information, see section 1.
You can use the following URI's to test your client program:

- www.example.com
- www.google.com (TE: chunked)
- www.tcpipguide.com
- www.jmarshall.com
- www.tldp.org
- www.tinyos.net
- www.linux-ip.net

We do not expect you to retrieve webpages from servers that only work with HTTPS. The exercises and what your program must be able to do are explained while leaving some room for interpretation. As such, creativity beyond the basic exercise is appreciated but the most important things are demonstrating working code supported by a good design and documentation. When choices are made, you should be able to motivate your choice.

Practical Information

You should either work alone or in groups of two. In the latter case, you and your partner need to belong to the same session! You should email your group details to **stefanos.peros@cs.kuleuven.be** before or during your **first session**. The subject of this email should be **CN:HTTP Group Composition**.
The body should consist of three lines:
Names: FirstName1 LastName1 , FirstName2 LastName2.
Student numbers: rXXXXXXX, rYYYYYYY.
Session Slot: Day, Start_Hour - End_Hour (i.e. Monday, 16:00 - 18:30).
Emails regarding questions for the assignment should also start with the **CN:HTTP** prefix.
You have 2 lab sessions to complete the assignment. In the 3rd lab, you should demonstrate your assignment to the teaching assistants. You will be marked based on your performance in the demonstration.
For students working in groups, both the students should be prepared to demonstrate the assignment. If you cannot explain your code, we will assume that you did not write it.
The third session is only meant for marking your assignment. Therefore, you should be ready to demonstrate your code at the start of the third practical session. You will only be marked in your assigned session.

[1] If you are having issues, try https://stackoverflow.com/questions/16619080/cant-get-the-http-response-using-telnet-from-the-command-line

References:

Java Sockets. Link: https://docs.oracle.com/javase/tutorial/networking/sockets/index.html
Very short HTTP intro:
Link: https://learn.onemonth.com/understanding-http-basics/
More in-depth:
Link: https://www.tutorialspoint.com/http/http_quick_guide.htm
From an implementer point-of-view:
HTTP made really easy [Strongly recommended].
Link: http://www.jmarshall.com/easy/http/
The HTTP Specification: HTTP 1.1 (RFC 2616).
The definition of embedded objects or MIME types (RFC 1521).
https://en.wikipedia.org/wiki/Telnet