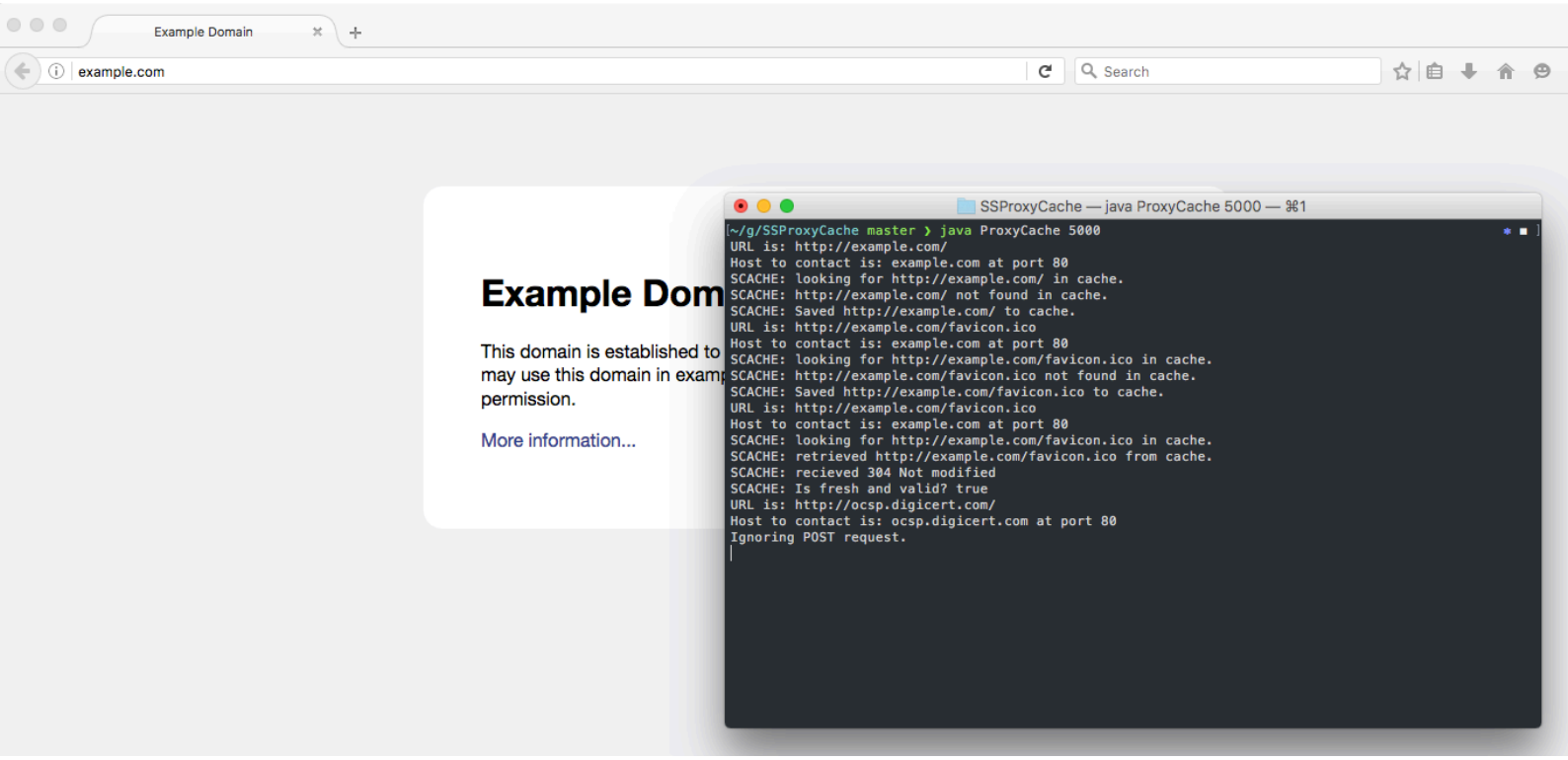
SSProxyCache

https://github.com/publicarray/SSProxyCache



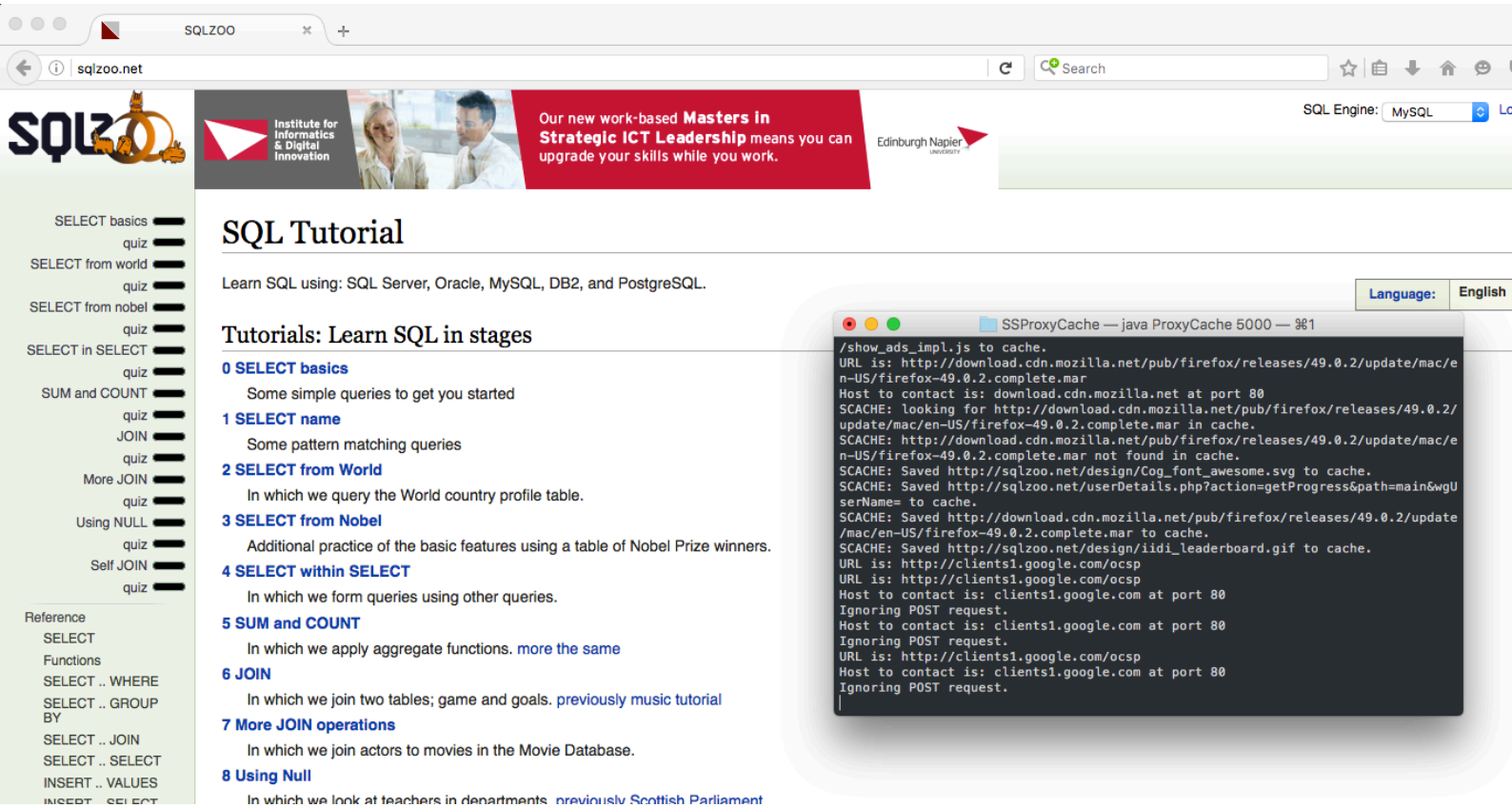


Table of Contents

1. Introduction

The SSProxyCache is designed to run a local system to cache the responses from a http server. The cache is a key value memory cache for increased performance. If an item is in the cache than the cached item is checked for freshness by asking the origin server.

2. Install instructions

The proxy is invoked and compiled using the command line.

2.1. Compiling the program

First get a copy of the program with git if you don't have one already.

```
git clone https://github.com/publicarray/SSProxyCache &&cd SSProxyCache
```

On a Linux or macOS system I recommend to compile the program with Make.

```
make
```

On a Windows system I recommend to compile with javac.

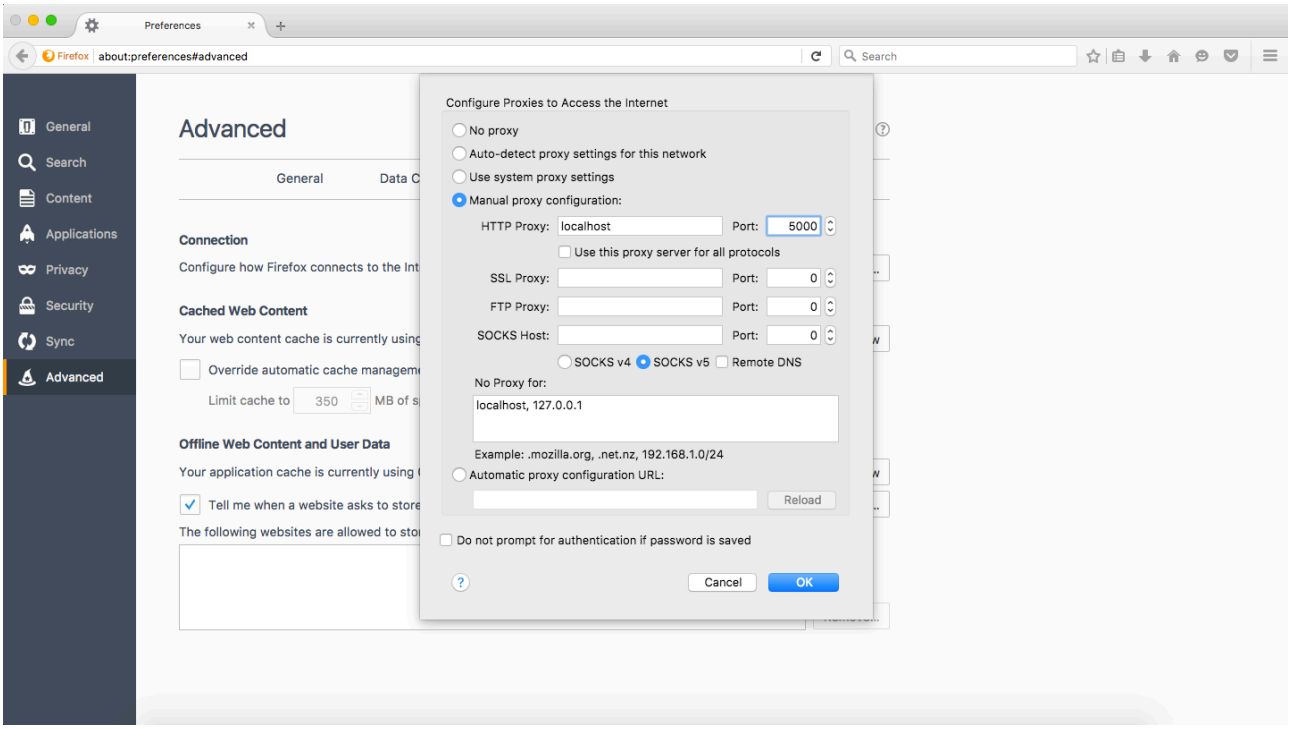
```
javac -Xlint:deprecation ProxyCache.java HttpRequest.java
HttpResponse.java SCache.java SSHelpers/Util.java
```

2.2. Setup

2.2.1 Firefox

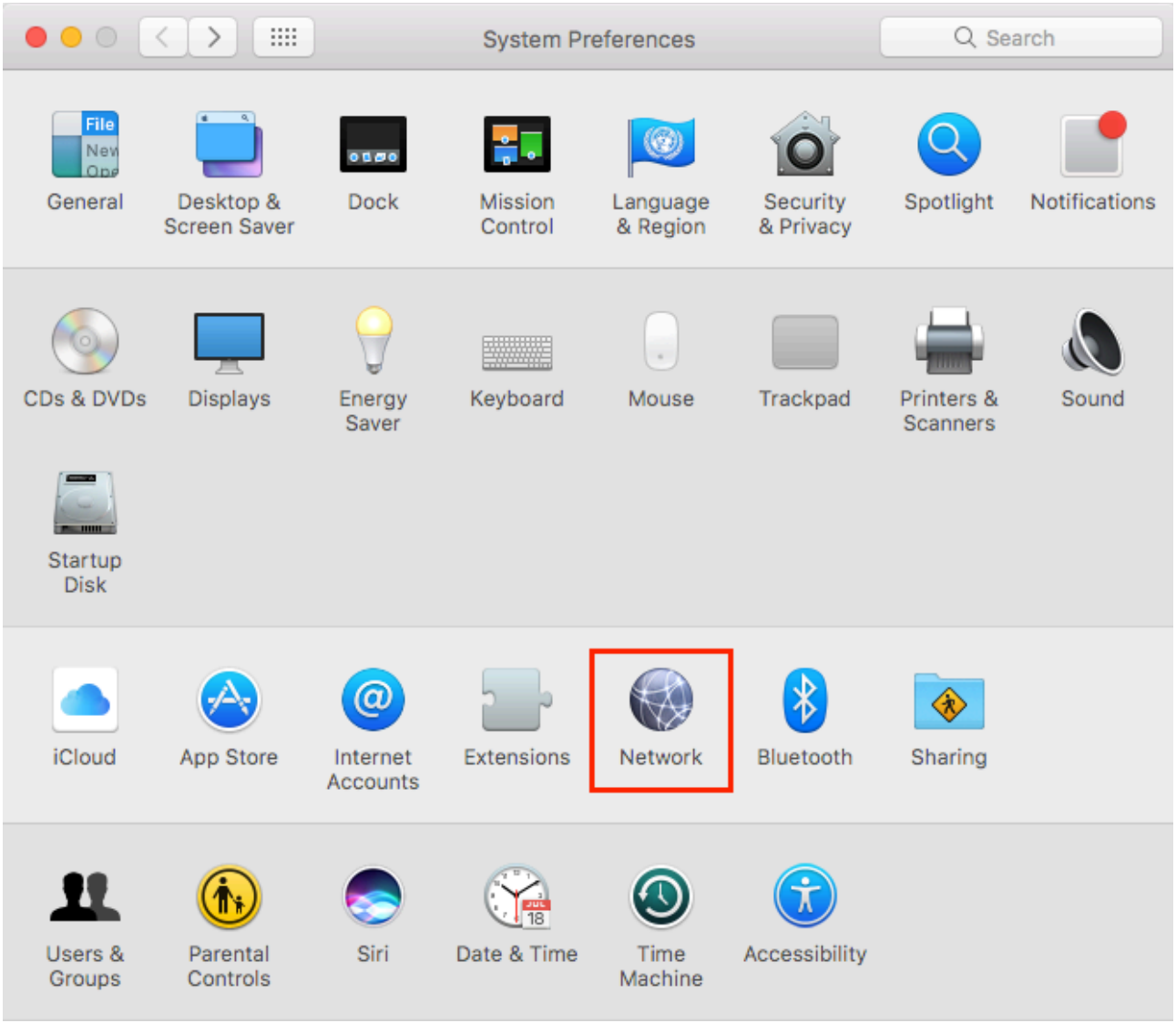
In Firefox navigate to `about:preferences#advanced` and open the network tab.

Under the Connection heading click on the Settings... Button. In the following window enter localhost as the HTTP Proxy and any port number higher than 1024 for the Port.

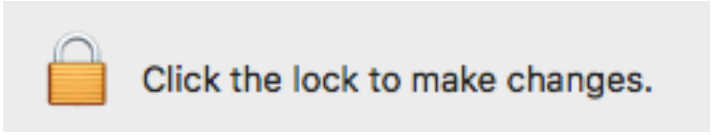


2.2.2. macOS

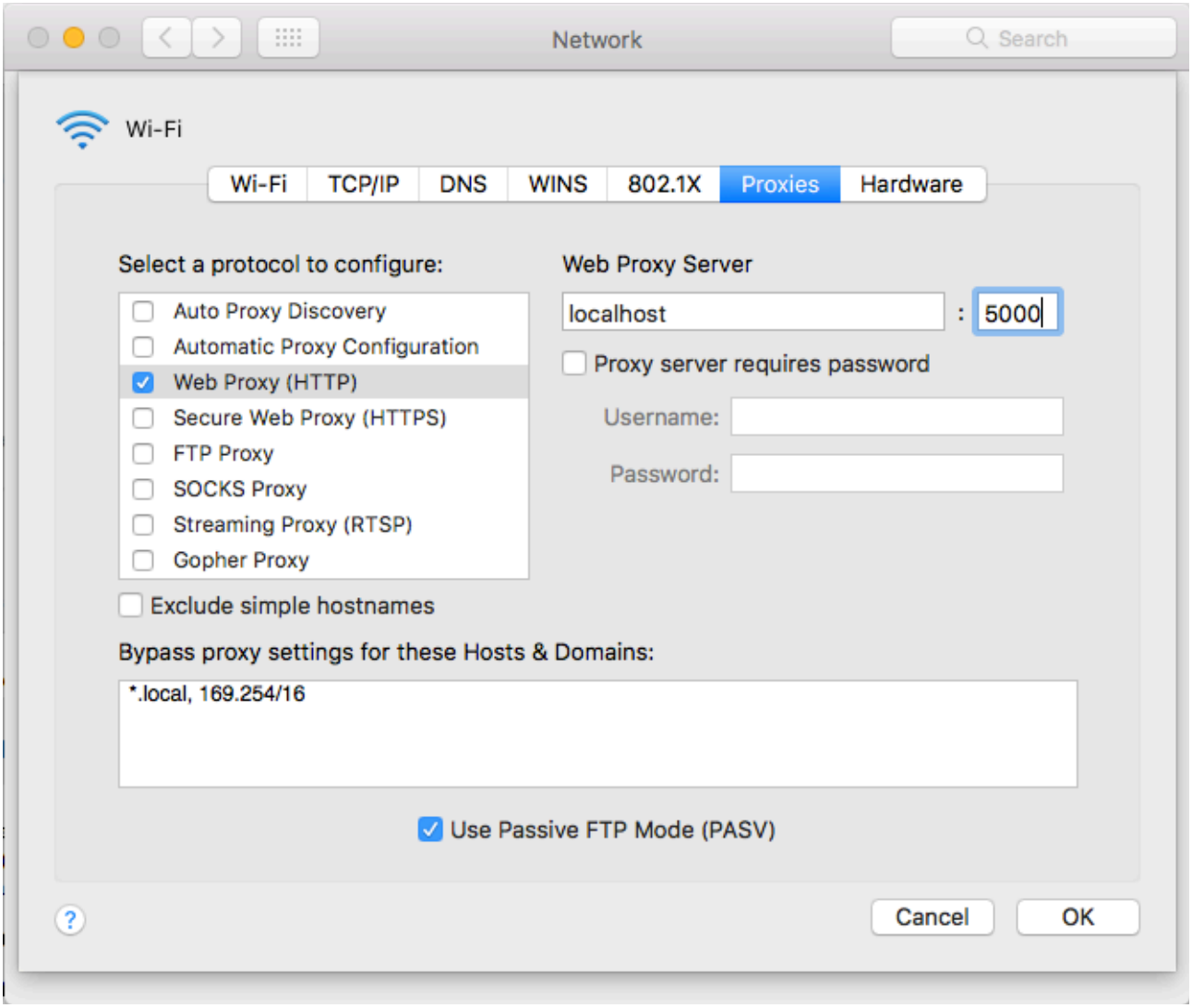
Open your System preferences by opening the preferences app. Next open your network preferences:



If the lock icon is closed, click on it and enter your password to unlock the network settings.

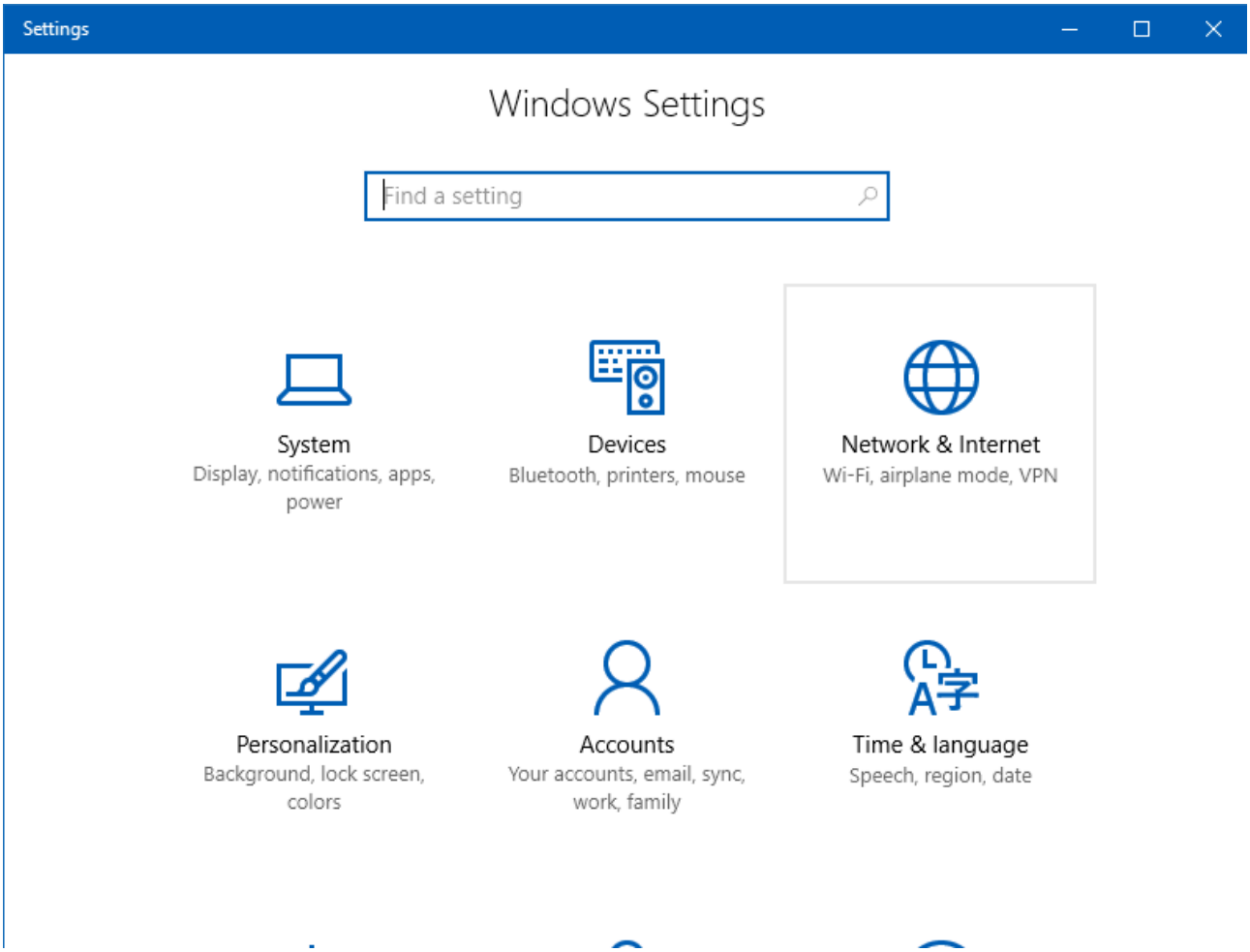


Next open the Advanced… preferences and open the Proxies tab.
Check the Web Proxy (HTTP) checkbox and enter localhost as the Web Proxy Server and any port number higher than 1024 for the Port (after the ":"). Click on the OK and than the Apply button.

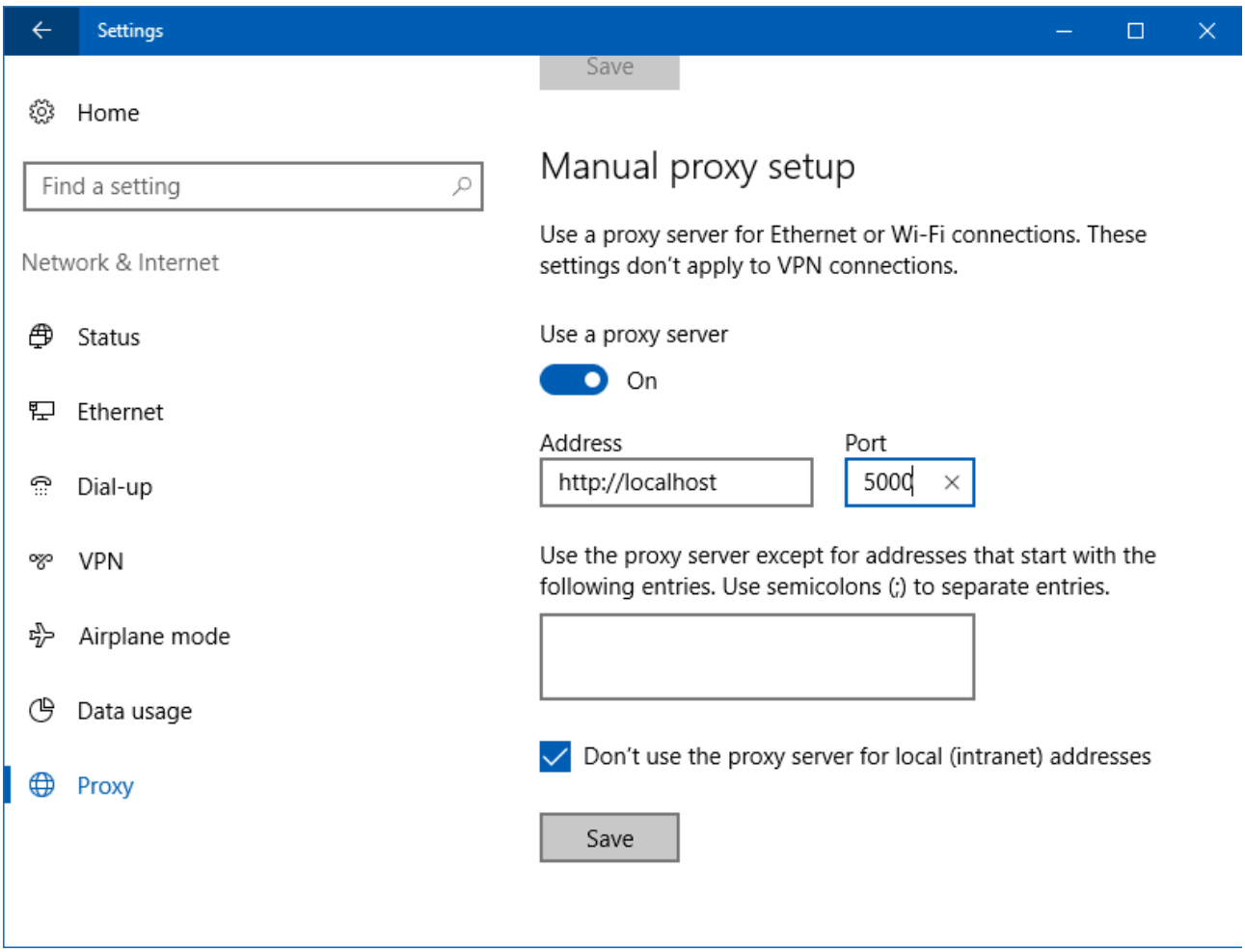

Network
Search
Wi-Fi
Wi-Fi
TCP/IP
DNS
WINS
802.1X
Proxies
Hardware
Select a protocol to configure:
Auto Proxy Discovery
Automatic Proxy Configuration
Web Proxy (HTTP)
Secure Web Proxy (HTTPS)
FTP Proxy
SOCKS Proxy
Streaming Proxy (RTSP)
Gopher Proxy
Web Proxy Server
localhost
:
5000
Proxy server requires password
Username:
Password:
Exclude simple hostnames
Bypass proxy settings for these Hosts & Domains:
*.local, 169.254/16
Use Passive FTP Mode (PASV)
Cancel
OK

2.2.3. Windows

Open the control panel and select the Network & Internet settings.



Next select the proxy tab and toggle on “Use a proxy server”. Enter the address as localhost and a port number grater than 1024. Next make sure the “Don’t use the proxy server for local (intranet) addresses is checked. Finally hit the save button.


Settings
Save
Home
Find a setting
Network & Internet
Status
Ethernet
Dial-up
VPN
Airplane mode
Data usage
Proxy
Manual proxy setup
Use a proxy server for Ethernet or Wi-Fi connections. These settings don't apply to VPN connections.
Use a proxy server
On
Address
http://localhost
Port
5000
Use the proxy server except for addresses that start with the following entries. Use semicolons (;) to separate entries.
Don't use the proxy server for local (intranet) addresses
Save

3. Usage

```
java ProxyCache <Port Number> [Arguments]
```

The Port number should be the same number you have entered in the setup step.
in our example above you can run the proxy with:

```
java ProxyCache 5000
```

Available arguments are:

-v, --verbose	Verbose output (more logging to the console)
-s, --secure	Proxy HTTPS/TLS (experimental, can use 100% CPU)
-e, --expires	Check expires header when checking the freshness of the cached object.

For example you can use the following command to proxy HTTPS/TLS traffic with the proxy listening on port 5000.

```
java ProxyCache 5000 –s
```

Note: to proxy HTTPS/TLS traffic you need to change your proxy settings in the setup step in 2.2 above to include HTTPS, TLS or SSL.

4. Appendix

4.1. ProxyCache.java

```
import java.net.*;
import java.io.*;
import java.util.*;

public class ProxyCache {
    private static boolean secure = false;
    public static boolean expires = false;
    public static boolean verbose = false;

    /** Port for the proxy */
    private static int port;
    /** Socket for client connections */
    private static ServerSocket socket;
    // Memory Cache
    private static SCache cache;

    /** Create the ProxyCache object and the socket */
    public static void init(int p) {
        port = p;
        cache = new SCache();
        try {
            socket = new ServerSocket(p);
        } catch (IOException e) {
            System.out.println("Error creating socket: " + e);
            System.exit(-1);
        }
    }

    public static void verbose(String message) {
        if (verbose) {
            System.out.println(message);
        }
    }

    public static void handle(Socket client) {
        int errorCode = 0;
        Socket server = null;
        HttpRequest request = null;
        HttpResponse response = null;

        /* Process request. If there are any exceptions, then
simply
         * return and end this request. This unfortunately means
the
         * client will hang for a while, until it timeouts. */
```

```
        /* Read request */
        try {
            BufferedReader fromClient = new BufferedReader(new
InputStreamReader(client.getInputStream()));
            verbose("\n**** New Request ****");
            request = new HttpRequest(fromClient);
            verbose("---> Request --->\n" + request.toString());
            if (request.getError() != 0) {
                new HttpResponse(request.getError()).send(client);
                return;
            }
        } catch (IOException e) {
            System.out.println("Error reading request from client:
" + e);
            new HttpResponse(400).send(client);

        }

        if (request.method.equals("GET")) { // Handle GET requests
            // Check if key is in cache
            if ((response = cache.get(request.getURL())) == null)
{ // get item from cache
                // if the item is not in the cache.

                /* Send request to server */
                response = request.send();

                /* Read response and forward it to client */
                cache.put(request.getURL(), response);// Save
response to cache
            }

            verbose("<--- Response <--- \n" +
response.toString());
            try {
                /* Write response to client. First headers, then
body */
                DataOutputStream toClient = new
DataOutputStream(client.getOutputStream());
                toClient.writeBytes(response.toString()); //
headers
                response.body.writeTo(toClient); // body
                client.close();
            } catch (IOException e) {
                System.out.println("Error writing response to
client: " + e);
            }
        }

        // http://stackoverflow.com/questions/16358589/
implementing-a-simple-https-proxy-application-with-java
```

```
        // http://stackoverflow.com/questions/18273703/tunneling-
two-socket-client-in-java#18274109
        else if (secure && request.method.equals("CONNECT")) {
            System.out.println("CONNECT found! attempt tunnel
HTTPS connection.");
            InputStream fromClient;
            OutputStream toServer;
            try {
                server = new Socket(request.getHost(),
request.getPort());
                fromClient = client.getInputStream();
                toServer = server.getOutputStream();
                new HttpResponse(200).setMessage("Connection
established").setVersion("HTTP/1.0").send(client);
            } catch (UnknownHostException e) {
                System.out.println("Unknown host: " +
request.getHost());
                new HttpResponse(404).send(client);
                return;
            } catch (IOException e) {
                System.out.println("Error establishing connection:
" + e);
                new HttpResponse(500).send(client);
                return;
            }


            try {
                client.setSoTimeout(3000);
                server.setSoTimeout(3000);
                byte[] buffer = new byte[2048];
                int rc = 0;

                // http://stackoverflow.com/questions/31365522/
java-proxy-tunnelling-https
                Thread servertToClient = new TunnelThread(server,
client);
                new Thread(servertToClient).start(); // from
server to client
                while (!client.isClosed() && !server.isClosed() &&
!client.isInputShutdown() && !server.isOutputShutdown()) {
                    while ((rc = fromClient.read(buffer)) != -1) {
                        toServer.write(buffer, 0, rc);
                        toServer.flush();
                    }
                }
                server.close();
                client.close();
            } catch (SocketTimeoutException e) {
                System.out.println("Connection Timed out " + e);
                // EXIT HERE
                return;
```

```
            } catch (IOException e) {
                System.out.println("Error tunnelling request: " +
e);
            }
        } else { // e.g POST, HEAD or DELETE requests
            System.out.println("Ignoring " + request.method + "
request.");
            new HttpResponse(501).send(client);
            return;
        }
    }

/* ------------------------------------------------- */


    /** Read command line arguments and start proxy */
    public static void main(String args[]) {
        int myPort = 0;

        try {
            myPort = Integer.parseInt(args[0]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Usage: ProxyCache <Port Number>
[args]\nArguments:\n  -v, --verbose    Verbose output (more
logging) \n  -s, --secure    Proxy HTTPS/TLS (experimental, can
use 100% CPU)\n  -e, --expires   Check expires header");
            System.exit(-1);
        } catch (NumberFormatException e) {
            System.out.println("Please give port number as
integer.");
            System.exit(-1);
        }

        init(myPort);

        for (String argument: args) { // http://java.about.com/od/
javasyntax/a/Using-Command-Line-Arguments.htm
            if(argument.equals("-v") || argument.equals("--
verbose")) {
                // Verbose output (more logging)
                verbose = true;
                System.out.println("Verbose logging=true");

            }
            if(argument.equals("-s") || argument.equals("--
secure")) {
                // proxy secure HTTPS/TLS connections
(experimental as it can course 100% CPU usage)
                secure = true;
                System.out.println("Proxy HTTPS/TLS connections.
(experimental as it can course 100% CPU usage)");
```

```
            }
            if(argument.equals("-e") || argument.equals("--
expires")) {
                // when validating the cache also check the
expires header, this can reduce the number of requests out to the
web.
                expires = true;
                System.out.println("Check expires header when
checking the freshness of an object in the cache.");
            }
        }

        /** Main loop. Listen for incoming connections and spawn a
new
         * thread for handling them */
        Socket client = null;

        while (true) {
            try {
                client = socket.accept();
                new Thread(new ProxyThread(client)).start();
            } catch (IOException e) {
                System.out.println("Error reading request from
client: " + e);
                /* Definitely cannot continue processing this
request,
                 * so skip to next iteration of while loop. */
                continue;
            }
        }

    }
}



class ProxyThread extends Thread {
    Socket client;

    ProxyThread(Socket client) {
        this.client = client;
    }

    public void run() {
        ProxyCache.handle(client);
    }
}

class TunnelThread extends Thread {
    Socket server;
    Socket client;
```

```
    TunnelThread(Socket server, Socket client) {
        this.server = server;
        this.client = client;
    }

    public void run() {
        int rs = 0;
        byte[] buffer = new byte[2048];
        try {
            OutputStream toClient = client.getOutputStream();
            InputStream fromServer = server.getInputStream();

            while (!server.isClosed() && !client.isClosed() && !
server.isInputShutdown() && !client.isOutputShutdown()) {
                while ((rs = fromServer.read(buffer)) != -1) {
                    toClient.write(buffer, 0, rs);
                    toClient.flush();
                }
            }
            System.out.println("thread ended normally");
        } catch (IOException e) {
            System.out.println("Thread: IOError: " + e);
            interrupt();
            return;
        }
    }
}
```

4.2. HttpRequest.java

```
import java.io.*;
import java.net.*;
import java.util.*;
import java.text.SimpleDateFormat;
import static SSHelpers.Util.*;

/**
 * This class contains the needed information (host, port number
and all Http headers) for a typical Http Request.
 *
 * @author      Modified by: Sebastian Schmidt
 * @version     1.0 2016-10-18
 * @since       1.0
 */

public class HttpRequest {
    /** Help variables */
    final static String CRLF = "\r\n";
    final static int HTTP_PORT = 80;
    /** Store the request parameters */
    String method;
    String URL;
    String version;
    String headers = "";
    /** Server and port */
    private String host;
    private int port;

    private int error;

    /**
     * Creates a HttpRequest by reading it from the client socket.
     * The constructor attempts to parse the content form the
client.
     * If there is a error in parsing the client will receive a
     * 400 (Bad request) error indicating that the request was
malformed.
     * @param  from The content stream from a socket connection.
     */
    public HttpRequest(BufferedReader from) {
        error = 0;
        String firstLine = "";
        try {
            firstLine = from.readLine();
        } catch (IOException e) {
            error = 400;
            System.out.println("Error reading request line: " +
e);
        }
```

```
        String[] tmp = firstLine.split(" ");
        method = tmp[0];
        URL = tmp[1];
        version = tmp[2];

        System.out.println("URL is: " + URL);




        try {
            String line = from.readLine();
            while (line.length() != 0) {
                headers += line + CRLF;
                /* We need to find host header to know which
server to
                 * contact in case the request URL is not
complete. */
                if (line.startsWith("Host:")) {
                    tmp = line.split(" ");
                    if (tmp[1].indexOf(':') > 0) {
                        String[] tmp2 = tmp[1].split(":");
                        host = tmp2[0];
                        port = Integer.parseInt(tmp2[1]);
                    } else {
                        host = tmp[1];
                        port = HTTP_PORT;
                    }
                }
                line = from.readLine();
            }

            // TODO: get POST parameters

        } catch (Exception e) {
            error = 400;
            System.out.println("Error reading from client socket:
" + e);
            return;
        }
        System.out.println("Host to contact is: " + host + " at
port " + port);
    }

    /**
     * Constructs a HttpRequest given the required variables
     * @param  method  The http method E.g GET or POST
     * @param  url     The location of the resource E.g. /
example.html
     * @param  version The Http protocol version, usually it is
HTTP/1.1
```

```
     * @param  headers Any optional http headers. If you don't
want to add any use an empty string ("").
     * @param  host    The hostname of the server where the
request is send to E.g. example.com
     * @param  port    The port number used for the webs server
E.g. 80
     */
    public HttpRequest(String method, String url, String version,
String headers, String host, int port) {
        this.method = method;
        this.URL = url;
        this.version = version;
        this.headers = headers;
        this.host = host;
        this.port = port;
        this.error = 0;
    }

    public HttpResponse askServerIfvalid(Date lastModified, String
etag) {
        if (lastModified == null || etag == null ||
etag.isEmpty()) {
            return null;
        }

        // User-Agent: SSProxyCache/1 (https://github.com/
publicarray/SSProxyCache)
        // headers = "User-Agent: SSProxyCache/1.0\r\n";
        headers = "";

        if (!etag.isEmpty()) {
            // If-None-Match:
"07bb743de42c7918a1a47fd61c6aa9c7:1469533683"\r\n
            headers += "If-None-Match: " + etag + CRLF;
        } if (lastModified != null) {
            // If-Modified-Since: Thu, 13 Oct 2016 00:36:43
GMT\r\n
            headers += "If-Modified-Since: " +
dateToString(lastModified) + CRLF;
        }

        headers += "Host: " + host + CRLF;
        return send();
    }

    public int getError() {
        return error;
    }

    /** Return host for which this request is intended */
    public String getHost() {
        return host;
```

```
    }

    public void setHost(String host) {
        this.host = host;
    }

    /** Return port for server */
    public int getPort() {
        return port;
    }

    public void setPort(int port) {
        this.port = port;
    }

    /** Return URL to connect to */
    public String getURL() {
        return URL;
    }

    // Return the method (GET, PUT, POST, etc.)
    public String getMethod() {
        return method;
    }

    /**
     * Convert request into a string for easy re-sending.
     */
    public String toString() {
        String req = "";

        req = method + " " + URL + " " + version + CRLF;
        req += headers;
        /* This proxy does not support persistent connections */
        req += "Connection: close" + CRLF;
        req += CRLF;

        return req;
    }

    HttpResponse send() {
        // send message/request to server
        Socket server;
        try {
            server = new Socket(getHost(), getPort());
            DataOutputStream toServer = new
DataOutputStream(server.getOutputStream());
            toServer.writeBytes(toString());
        } catch (UnknownHostException e) {
            System.out.println("Unknown host: " + getHost());
            return new HttpResponse(404);
        } catch (IOException e) {
```

```
            System.out.println("HttpRequest.java - Error writing
request to server: " + e);
            return new HttpResponse(500);
        }

        // handle response from server
        try {
            DataInputStream fromServer = new
DataInputStream(server.getInputStream());
            HttpResponse response = new HttpResponse(fromServer);
            server.close();
            return response;
        } catch (IOException e) {
            System.out.println("HttpRequest.java - Error reading
response from server: " + e);
            return new HttpResponse(520);
        }
    }
}
```

4.3. HttpResponse.java

```
import java.io.*;
import java.net.*;
import java.util.*;
import static SSHelpers.Util.*;

public class HttpResponse {
    final static String CRLF = "\r\n";
    /** How big is the buffer used for reading the object */
    final static int BUF_SIZE = 8192;



    /** Reply status and headers */
    String version; // HTTP version, part of statusLine
    int status; // HTTP status code, part of statusLine
    String statusMessage; // part of statusLine
    String statusLine = "";
    String proxy = "SSProxy"; // self advertising + it helps to
identify requests that have passed through the proxy
    String headers = ""; // all headers, excluding the statusLine
    Date lastModified;
    Date expires;
    String etag;
    /* Body of reply */
    ByteArrayOutputStream body = new ByteArrayOutputStream();

    /** Read response from server. */
    public HttpResponse(DataInputStream fromServer) {
        /* Length of the object */
        int length = -1;
        boolean gotStatusLine = false;

        /* First read status line and response headers */
        try {

            String line = fromServer.readLine();
            // Parse HTTP headers
            while (line != null && line.length() != 0) { // line !
= null to medigate java.lang.NullPointerException
                if (!gotStatusLine) {
                    statusLine = line;
                    String[] tmp = line.split(" ", 3);
                    if (tmp.length == 3) {
                        this.version = tmp[0];
                        this.status = Integer.parseInt(tmp[1]);
                        this.statusMessage = tmp[2];
                    }
                    gotStatusLine = true;
                } else {
                    headers += line + CRLF;
```

```
                }

                /* Get length of content as indicated by
                 * Content-Length header. Unfortunately this is
not
                 * present in every response. Some servers return
the
                 * header "Content-Length", others return
                 * "Content-length". You need to check for both
                 * here. */
                if
(line.toLowerCase(Locale.ROOT).startsWith("content-length")) {

                    String[] tmp = line.split(" ");
                    length = Integer.parseInt(tmp[1]);
                }
                else if
(line.toLowerCase(Locale.ROOT).startsWith("last-modified")) {
                    String[] tmp = line.split(" ", 2);
                    lastModified = toDate(tmp[1]);
                } else if
(line.toLowerCase(Locale.ROOT).startsWith("expires")) {
                    String[] tmp = line.split(" ", 2);
                    expires = toDate(tmp[1]);
                }  else if
(line.toLowerCase(Locale.ROOT).startsWith("etag")) {
                    String[] tmp = line.split(" ", 2);
                    etag = tmp[1];
                }

                line = fromServer.readLine();
            }
        } catch (IOException e) {
            System.out.println("Error reading headers from server:
" + e);
            return;
        }

        try {
            int bytesRead = 0;
            byte buf[] = new byte[BUF_SIZE];
            boolean loop = false;

            /* If we didn't get Content-Length header, just loop
until
             * the connection is closed. */
            if (length == -1) {
                loop = true;
            }

            /* Read the body in chunks of BUF_SIZE and copy the
chunk
```

```
             * into body. Usually replies come back in smaller
chunks
             * than BUF_SIZE. The while-loop ends when either we
have
             * read Content-Length bytes or when the connection is
             * closed (when there is no Connection-Length in the
             * response. */
            while (bytesRead < length || loop) {
                /* Read it in as binary data */
                int res = fromServer.read(buf);

                if (res == -1) {
                    break;
                }
                /* Copy the bytes into body. Make sure we don't
exceed
                 * the maximum object size. */
                body.write(buf, 0, res);




                bytesRead += res;
            }


        } catch (IOException e) {
            // error = 500;
            System.out.println("Error reading response body: " +
e);
            return;
        }


    }

    // constructor for HTTP status codes
    public HttpResponse(int statusCode) {
        version = "HTTP/1.1";
        status = statusCode;

        switch (status) {
            case 200:
                statusMessage = "OK";
                break;
            case 400: // when headers are not properly formatted
                statusMessage = "Bad request";
                break;
            case 404:
                statusMessage = "Not found";
                break;
            case 405:
```

```
                statusMessage = "Method not allowed";
                break;
            case 500:
                statusMessage = "Internal server error";
                break;
            case 501: // for methods that are not implemented
                statusMessage = "Not implemented";
                break;
            case 505: // for http/2
                statusMessage = "HTTP version not supported";
                break;
            case 520:
                statusMessage = "Unknown Error";
                // used by Microsoft and CloudFlare as a "catch-
all" response
                // for when the origin server returns something
unexpected
                // or something that is not tolerated/interpreted.
                break;
            default:
                System.out.println("Invalid HTTP status code");
                status = 500;
                statusMessage = "Internal server error";
        }
        statusLine = version + " " + status + " " + statusMessage;
        String bodyStr = status + " " + statusMessage;
        body.write(bodyStr.getBytes(), 0, bodyStr.length());
    }

    public HttpResponse setStaus(int status) {
        this.status = status;
        return this;
    }

    public HttpResponse setMessage(String message) {
        this.statusMessage = message;
        return this;
    }

    public HttpResponse setVersion(String version) {
        this.version = version;
        return this;
    }

    public byte[] getBody() {
        return body.toByteArray();
    }

    /**
     * Convert response into a string for easy re-sending. Only
     * converts the response headers, body is not converted to a
     * string.
```

```
     */
    public String toString() {
        String res = "";

        res = statusLine + CRLF;
        res += headers;
        res += "X-Proxy-Agent: " + proxy + CRLF;
        res += CRLF;

        return res;
    }

    // send response to a host (generally the client)
    public Socket send(Socket host) {
        try {
            // Write response to host.
            DataOutputStream toClient = new
DataOutputStream(host.getOutputStream());
            toClient.writeBytes(toString());
            ProxyCache.verbose("<--- Response <--- \n" +
toString());
        } catch (IOException e) {
            System.out.println("Error writing response to client:
" + e);
        }
        return host;
    }

    // check if response is still valid using the (now old)
expires header
    public boolean isExpired() {
        if (lastModified != null && expires != null) {
            Date now = new Date(); // get current date time
            return !(now.after(lastModified) &&
now.before(expires)); // no web request is needed
        }
        return false;
    }
}
```

4.4. SCache.java

```
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.HashMap;
import java.text.SimpleDateFormat;
import static SSHelpers.Util.*;

public class SCache {
    // private Helpers helpers = new Helpers();

    // A key value store in memory
    private Map<String, HttpResponse> cacheStore;
    // A HttpRequest used to send If-None-Match and If-Modified-
Since requests to the server
    HttpRequest request;

    /**
     * This SCache class contains the cache for the proxy.
     * It provides methods to add, validate and get items from the
cache.
     * The cache uses the URL as a key to lookup items in the
cache
     */
    public SCache() {
        // synchronised HasMap prevents data corruption when
reading while writing the Map
        cacheStore = Collections.synchronizedMap(new
HashMap<String, HttpResponse>());
        request = new HttpRequest("GET", "", "HTTP/1.1", "", "",
0);
    }

    /**
     * Adds the HttpResponse into the cache with the url as the
key.
     * @param  key   The Url
     * @param  value The HttpResponse
     * @return       The given value HttpResponse
     */
    public HttpResponse put(String key, HttpResponse value) {
        System.out.println("SCACHE: Saved " + key + " to cache.");
        return cacheStore.put(key, value);
    }

    /**
     * Retrieves the HttpResponse from cache by looking up the
url.
     * The HttpResponse is checked to make sure it is fresh.
     * @param  key The Url
     * @return     The Cached HttpResponse
```

```
     */
    public HttpResponse get(String key) {
        System.out.println("SCACHE: looking for " + key + " in
cache.");
        HttpResponse value = cacheStore.get(key);
        if (value == null) {
            System.out.println("SCACHE: " + key + " not found in
cache.");
            return null;
        }
        System.out.println("SCACHE: retrieved " + key + " from
cache.");

        boolean valid = isValid(key, value);
        System.out.println("SCACHE: Is fresh and valid? " +
valid);
        if (!valid) { // get the new value if there is a new one
            value = cacheStore.get(key);
        }
        return value;
    }

    /**
     * Determines if the cached HttpResonse is still fresh and
updates the cache when it is not.
     * It checks that the response has not expired by checking the
Last-Modified and the Expired header
     * in the response. If no expired header is present the server
is asked with If-None-Match and the
     * If-Modified-Since headers. If the server response status
code is not a 304 Not Modified response
     * than the new response updates the value in the cache.
     * @param  requestURL The key in the cache
     * @param  response   The value in the cache
     * @return            isValid? as in is the item in the cache
fresh?
     */
    public boolean isValid(String requestURL, HttpResponse
response) {
        if (ProxyCache.expires && !response.isExpired()) { //
TODO: add command line flag
            System.out.println("SCACHE: response has not
expired");
            return true;
        }

        request.setHost(getHost(requestURL));
        request.setPort(getPort(requestURL));
        request.URL = getLocation(requestURL);
```

```
        HttpResponse newResponse =
request.askServerIfvalid(response.lastModified, response.etag); //
ask severer
        if (newResponse == null) { // if error
            return true; // use the cached response if new
response has an unrecoverable error.
        }
        if (newResponse.status == 304) { // Not modified, so still
valid
            System.out.println("SCACHE: recieved 304 Not
modified");
            return true;
        } else { // 200 OK and other responses
            System.out.println("SCACHE: recieved " +
newResponse.status);
            System.out.println("SCACHE: response was modified.
Updating Cache.");
            put(requestURL, newResponse); // save the new response
            return false;
        }
    }
}
```

4.5. SSHelpers/Util.java

```
package SSHelpers;

import java.util.Date;
import java.util.TimeZone;
import java.text.ParseException;
import java.text.SimpleDateFormat;
/**
 * A utility class containing small helper methods.
 * These methods can be used my importing the class: <code>import
static SSHelpers.Util.*;</code>
 *
 * @author      Sebastian Schmidt
 * @version     1.0 2016-10-18
 * @since       1.0
 */
public class Util {

    /**
     * From a url string get the host-name
     * @param  url The Url E.g. http://example.com/foo
     * @return     The Hostname E.g. example.com
     */
    public static String getHost(String url) {
        if (url == null || url.length() <= 0) {
            return "";
        }

        int start = url.indexOf("//");
        if (start == -1) {
            return "";
        }
        start += 2;

        int end = url.indexOf('/', start);
        if (end == -1) {
            return "";
        }

        return url.substring(start, end);
    }

    /**
     * From a url string get the port number
     * @param  url The Url E.g. http://example.com:50/foo
     * @return     The Port number E.g. 50
     */
    public static int getPort(String url) {
        if (url == null || url.length() <= 0) {
            return 80;
        }
```

```
        int start = url.indexOf("//");
        if (start == -1) {
            return 80;
        }

        start += 2;

        int end = url.indexOf('/', start);
        if (end == -1) {
            return 80;
        }

        int port = url.indexOf(':', start);
        if (port == -1) {
            return 80;
        }

        if (port < end) {
            end = port;
        }

        return Integer.parseInt(url.substring(port, end));
    }

    /**
     * From a url string get the location
     * @param  url The Url  E.g. http://example.com/foo
     * @return     The location E.g. /foo
     */
    public static String getLocation(String url) {
        if (url == null || url.length() <= 0) {
            return "";
        }

        int start = url.indexOf("//");
        if (start == -1) {
            return "";
        }
        start += 2;

        int end = url.indexOf('/', start);
        if (end == -1) {
            return "";
        }

        return url.substring(end);
    }

    /**
     * Converts a string to a Java date object given the correct
format.
```

```
     * @param  dateStr The string that is parsed as a date.
     * @param  format  The format as defined in the
SimpleDateFormat <a href="https://docs.oracle.com/javase/8/docs/
api/java/text/SimpleDateFormat.html">https://docs.oracle.com/
javase/8/docs/api/java/text/SimpleDateFormat.html</a>.
     * @return         Returns a new date object. On failure it
will return null.
     */
    public static Date toDate(String dateStr, String format) {
        try {
            SimpleDateFormat inFormat = new
SimpleDateFormat(format);
            inFormat.setTimeZone(TimeZone.getTimeZone("GMT"));
            return inFormat.parse(dateStr);
        } catch (ParseException e) {
            System.out.println("Parse date error: " + e + " -- at
position:" + e.getErrorOffset());
        }
        return null;
    }
     /**
     * Converts a string to a Java date object using "E, dd MMM
yyyy HH:mm:ss z" (Mon, 13 Jun 2016 21:06:31 GMT) as the format.
     * @param  dateStr The string that is parsed as a date.
     * @return         Returns a new date object. On failure it
will return null.
     */
    public static Date toDate(String dateStr) {
        String format = "E, dd MMM yyyy HH:mm:ss z";
        return Util.toDate(dateStr, format);
    }

    /**
     * Converts a Java string object to a string with the given
format.
     * @param  date   The date to format.
     * @param  format The format as defined in the
SimpleDateFormat <a href="https://docs.oracle.com/javase/8/docs/
api/java/text/SimpleDateFormat.html">https://docs.oracle.com/
javase/8/docs/api/java/text/SimpleDateFormat.html</a>.
     * @return        The date as a string.
     */
    public static String dateToString(Date date, String format) {
        SimpleDateFormat dateFormat = new SimpleDateFormat
(format);
        dateFormat.setTimeZone(TimeZone.getTimeZone("GMT"));
        return dateFormat.format(date);
    }

    /**
     * Converts a Java string object to a string using "E, dd MMM
yyyy HH:mm:ss z" (Mon, 13 Jun 2016 21:06:31 GMT) as the format.
```

```
     * @param  date   The date to format.
     * @return        The date as a string.
     */
    public static String dateToString(Date date) {
        return Util.dateToString(date, "E, dd MMM yyyy HH:mm:ss
z");
    }
}
```