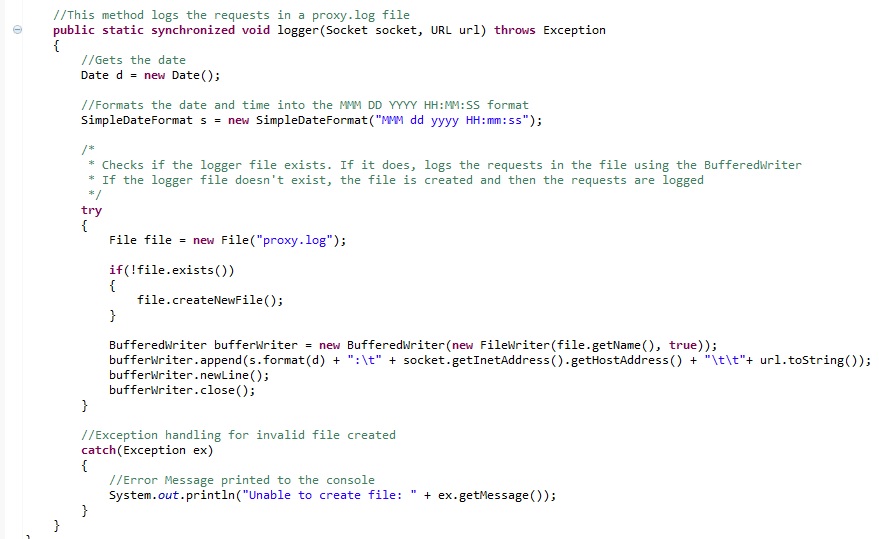
|  |
| --- |
| **Proxy Implementation Report** |
| A report of the proxy implementation for the Assignment 2 of CSCI 415 |
|  |
| This report elaborates on the proxy implementation done for the Assignment 2 for the course CSCI 415 |
|  |
| **Sharma, Akshat** |
| **10/7/2013** |
|  |

The program described below implements a HTTP proxy server that passes requests and data between web clients and web servers. The proxy server also logs requests. Caching has also been implemented where proxy server stores the returned data of past requests in local storage.

**1. Logging**

* The proxy keeps track of all requests in a log file named proxy.log



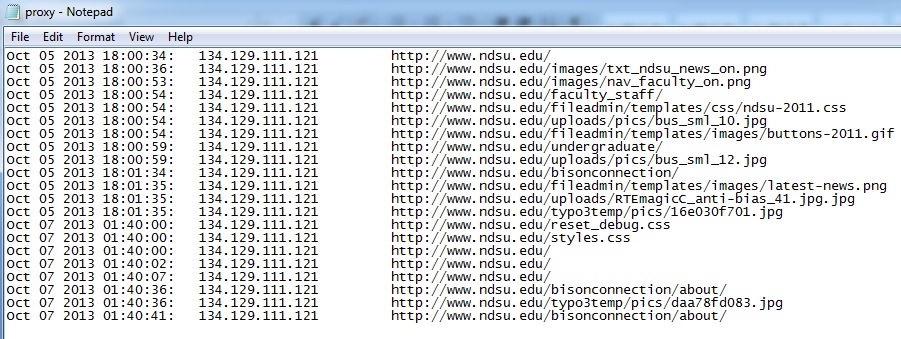
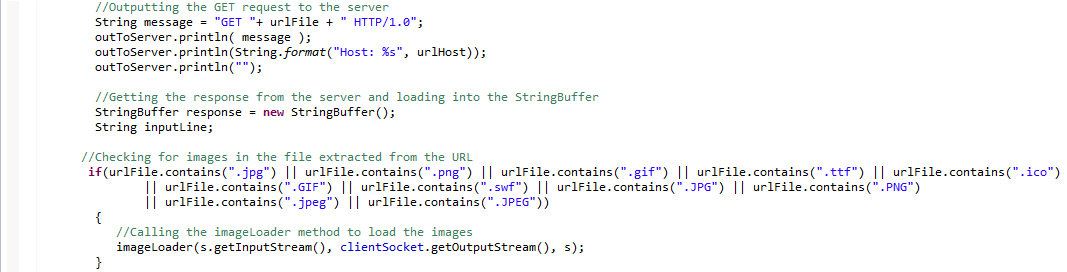
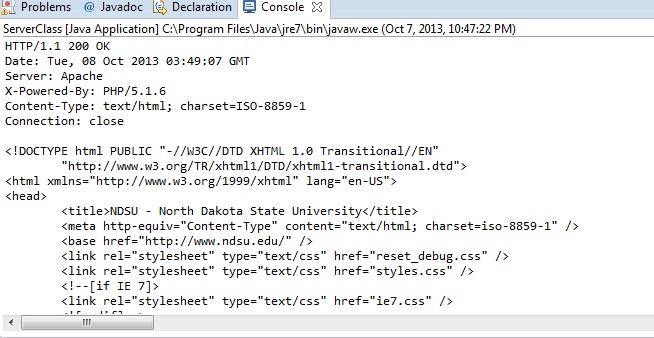


Image showing the Proxy.log file along with a few requests logged

* Here, we created a new file object naming the file proxy.log. We check if the file already existed or not. If not, then we create a new file to log the requests.
* After creating the file, we append every incoming request in a new line in the file.
* If there are any errors in creating the log file they are caught using the exception handler and printed to the user console.

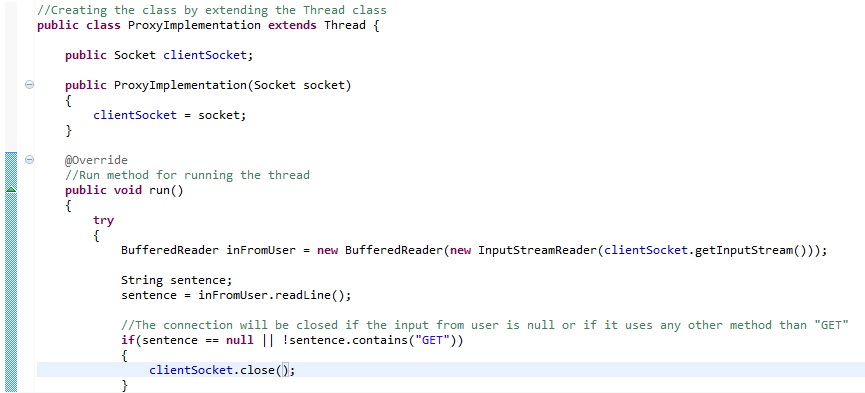
**2. GET method**





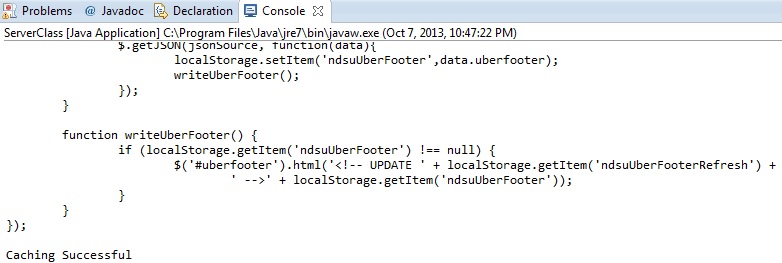
* Here we request for the URL file using the GET request and the HTTP/1.0 version.
* Then we inform the server on which host the file exist and grab the file from the host.
* The response is read by the variable inputLine line by line and each line is appended in the StringBuffer and also outputted to the browser using the PrintWriter.

**3. Multi-threading.**

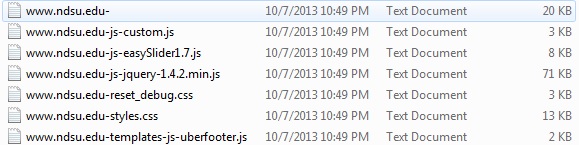


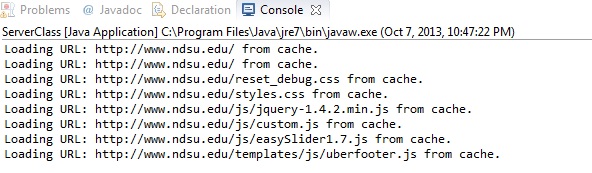
* Multithreading was implemented by extending the thread class, and then by using the run method which informs the thread to run.
* Our proxy does not process the request sequentially but multiple requests at once are processed using multithreading to make the program efficient.
* Only one thread accesses our log file at a time and the requests are synchronized using the synchronized keyword in the method and logged sequentially.

**4. Caching**



Webpage Cached

  
Cached Files

  
Webpage Loading from cache

* Our proxy server stores the returned data of past requests in local storage using caching.
* If the cached file of the web object does not exist, we load the object using the server request, and then cache the file of the web object to disk using the BufferWriter.
* Then if the URL already cached is requested again by the browser, first the cached file is read using the bufferReader, and then the cache kicks in at the proxy serve and outputs the cached data to the browser using the PrintWriter, rather than fetching it again from the server. However, since we were not able to cache the image files, the image files are always loaded from the server.
* Hence, by usage of caching, we saved a considerable amount of bandwidth
* The image sequence shows how first the webpage is cached, the cached files are created and then the cached files are loaded when the browser requests the same page again