In this programming assignment, you are asked to implement a program in either Java or Python. This program is supposed to download an index file to obtain a list of text file URLs and download some of these files depending on their sizes. For this programming assignment, you are not allowed to use any third party HTTP client libraries, or the HTTP specific core or non-core APIs. The goal of this assignment is to make you familiar with the internals of the HTTP protocol and using any (third party, core or non-core) API providing any level of abstraction specific to the HTTP protocol is not allowed. You must implement your program using either the Java Socket API of the JDK or the Socket package in the Python distribution. For Python, the use of any class/function from http or the requests package is prohibited. If you have any doubts about what to use or not to use, please contact **sarp.yenicesu@bilkent.edu.tr**.

Your program must be a console application (no graphical user interface(GUI) is required) and should be named as CloudDownloader (i.e., the name of the class that includes the main method for Java should be CloudDownloader). Your program should run with the

java CloudDownloader <index_file> <username>:<password>
or
python CloudDownloader.py <index_file> <username>:<password>

command where <index_file> and <username>:<password> are the command-line arguments.

The details of the command-line arguments are as follows:

- <index_file>: [Required] The URL of the index that includes the list of partial file locations and their authentication information.

- <username>:<password> [Required] The authentication information of the index server. You will use "**Basic**" authentication method of the HTTP. For more

information: https://en.wikipedia.org/wiki/Basic_access_authentication#Client_side. You can use the java.util.Base64 library in Java, and base64 module in Python when you need Base64 encoding.

When a user enters the command above, your program will send an HTTP GET request to the server to download the index file with URL <index_file>. If the index file is not found, i.e. the response is a message other than 200 OK, your program should print an error message to the command-line and exit. If the index file is found, i.e. the response is a 200 OK message, your program should continue and follow the URLs, available bytes and authentication information provided in the index file to download the complete file.

If your program successfully obtains the file or a part of the file, it saves the content under the directory in which your program runs. The name of the saved file is provided in the initial index file. A message showing which parts are downloaded from where should be printed to the command-line.

Basically, your program;

1. Sends a HTTP GET to an initial server with authentication credentials.
2. Reads the index file and sends HTTP GETs with correct range fields and authentication credentials to those URL in a sequential order.
3. Save the downloaded files into a single file.

Note that all servers have different parts of the file and some parts might be redundant such as first server containing 1-300 bytes, second server containing 200-500 bytes and finally third server containing 150-750 bytes for a 750 Byte file, in such case the program should download as much as possible from the lower indexed servers. So that it will get the 1-300 bytes from the first server, 301-500 bytes from the second server and 501-750 bytes from the third server. **Even though byte ranges for each partition file is specified, these files may contain more bytes than this range. Consider this during your implementation, be aware of redundancy.**

Therefore, your program also;

1. Calculates the range field from previously downloaded file parts.
2. Combine the files in correct order

As a **bonus assignment**, we expect you to add additional features and mechanisms to CloudDownloader. We will evaluate these features based on creativity and implementational complexity. The bonus part is not mandatory, and the grading of this part is separate from the original grading scheme. The bonus points that you will get will be added to the original grade.

Lastly, write a **report** (PDF file) in which you explain the important parts of your code. We should be able to navigate the source code just from the report. The number of pages should not exceed 5.

# Assumptions and Hints

- Please refer to W3Cs RFC 2616 for details of the HTTP messages.
- You will assume the authentication provided for all servers are correct. Note that there should be a colon ‘:’ character between the endpoints.
- You will assume each line of the index file includes one file URL followed by its authentication and followed by its byte-range all in separate lines.

- You will assume that each consecutive server stores further bytes of the file. For example, if one server has 300-500 bytes, the next server will at least have byte 501. Therefore, you must connect all the servers.
- Byte ranges in the index file always start with the correct index.
- Your program will not save the index file to the local folder.
- Your program should print a message to the command-line to inform the user about the status of the file.
- The downloaded file should be saved under the directory containing the source file CloudDownloader.[java/py] and the name of the file is given in <index_file>.
- You may use the following URLs, authentications and resulting files to test your program:

    dijkstra.cs.bilkent.edu.tr/~cs421/descriptor1.txt, cs421:bilkent,
    dijkstra.cs.bilkent.edu.tr/~cs421/descriptor2.txt, cs:421

The full version of txt files can be found in:

    dijkstra.cs.bilkent.edu.tr/~cs421/Test1.txt,

    dijkstra.cs.bilkent.edu.tr/~cs421/Test2.txt

- You will assume that all servers are accepting connections through port 80.
- Please make sure the file your program downloads is exactly the same as the test file.
- Please contact your assistant if you have any doubt about the assignment.

## Example

Let 111.111.11.11/descriptor.txt be the URL of the index file to be downloaded whose content is given as

| | |
|---|---|
| TestFile1<br>1000<br>222.222.22.22/partial1.txt<br>Hamza:4242<br>1-500<br>233.233.33.33/partial2.txt<br>Sarp:1010<br>300-1000 | Name of the file<br>Size of the file in bytes<br>URL of the first server<br>Authentication of the first server<br>The range of the available bytes in this server<br>URL of the second server<br>Authentication of the second server<br>The range of the available bytes in this server |

The username of the server 1 is ‘cs’ and password is ‘421’

**Example run 1** Let your program start with the command:

java CloudDownloader 111.111.11.11/descriptor.txt cs:421 or,
python CloudDownloader.py 111.111.11.11/descriptor.txt cs:421

```
Command--line:
URL of the index file: 111.111.11.11/descriptor.txt
File size is 1000 Bytes
Index file is downloaded
There are 2 servers in the index
Connected to 222.222.22.22/partial1.txt
Downloaded bytes 1 to 500 (size = 500)
Connected to 233.233.33.33/partial2.txt
Downloaded bytes 501 to 1000 (size = 500)
Download of the file is complete (size = 1000)
```

## Submission rules

You need to apply all the following rules in your submission. **You will lose points if you do not obey the submission rules below or your program does not run as described in the assignment above.**

- The assignment should be uploaded to the Moodle in a zip file. Any other methods (Disk/CD/DVD) of submission will not be accepted.
- The name of the submission should start with [CS421_PA1], and include your name and student ID. For example, the name must be

  [CS421_PA1]AliVelioglu20111222

  if your name and ID are Ali Velioglu and 20111222. You are **not allowed** to work in groups.
- All the files must be submitted in a **zip** file whose name is described above. The file must be a .zip file, not a .rar file or any other compressed file.
- All of the files must be in the root of the zip file; directory structures are not allowed. Please note that this also disallows organizing your code into Java packages. The archive should not contain:

  – Any class files or other executables,

  – Any third-party library archives (i.e. jar files),

  – Any text files,

  – Project files used by IDEs (e.g., JCreator, JBuilder, SunOne, Eclipse, Idea or NetBeans etc.). You may, and are encouraged to, use these programs while developing, but the end result must be a clean, IDE-independent program.
- The standard rules for plagiarism and academic honesty apply; if in doubt refer to Academic Integrity Guidelines for Students and Academic Integrity, Plagiarism & Cheating.

# Important Notes

**General Submission Rules:**

**1- )** Your submission should include source code(s) (.java/.py).

**2- )** The format of the report should be **PDF**. (Do not upload .doc, .docx or any other types). The number of pages should not exceed 5.

**4- )** Your submission should not contain any other files other than the source code(s) (.java/.py), report (.pdf) and optional README. No .txt files, no folders, no IDE related files should be included.

**5-)** Compress these files with **.zip** format. (.rar, .7z or any other compressing types will not be accepted.)

**6-)** Make sure to follow rules in the "Submission Rules" section like name of the zip file, method of the submission etc.

**For Python Submissions:**

**1-)** The code should run with the "python3 CloudDownloader.py <index_file> <username>: <password>" command.

**2-)** Python version should be **3.6 or higher**. Other versions (like Python 2) are **not accepted**.

**For Java Submissions:**

**1-)** The code should run with the following commands:

Compile: "javac *.java"

Run: "java CloudDownloader <index_file> <username>: <password>" command.

**2-)** Java version should be **8 or higher.**

**3-)** The JDK should be Oracle JDK (**not** OpenJDK).