Diego Sabaris Montes: S21859.

Points 1, 2, and 3 were done from the project “System ściągania i udostępniania plików TORrent”**.**

Description:

When the Main.class file is executed, two directories are created: D:\\TORent\_0 and D:\\TORent\_1 containing one .txt example file on each. Each app ( 0 or 1 ) is client and server in the same time, running on 2 separate threads. The client socket of the app 0 is connected to the server Socket of app 1 and the other way round. In each app, both client and server sides are constantly reading from a stream from the socket connection and that is how the exchange of information takes place (TCP protocol).

The script **00Compilingscript.bat**  will compile the program. (used jdk1.8.0\_271.)

The script **0MainConsoleOverview.bat** runs two applications (app 0 and app 1).

The script **1PullFile.bat** runs two applications (app 0 and app 1) and calls the pullFile() method on the client of the receiver, which pulls an example file (file nr 0) by app 1 from app 0. The description of how this method works can be found in the section “More details” in this document.

The script **2FilesandMD5infoExhange.bat** runs two applications (app 0 and app 1) and calls the checkFiles() method on both applications, which exchanges between them the files which are available on each directory along with their respective MD5 checksums. The description of how this method works can be found in the section “More details” in this document.

The script **3PushFile.bat** runs two applications (app 0 and app 1) and calls the pushFile() method on the server of the sender, which pushes an example file (file nr 0) from app 1 to app 0. The description of how this method works can be found in the section “More details” in this document.

More details:

Every functionality explained below is applicable both ways <->. This means that the receiver can be a sender and vice versa.

This is how the **pushFile** method works (Server.java, line 39. Parameters: String):

If we take parameter “exitexit” we are closing the connection.

If we take a file path, then:

App 0 (the sender) sends the path file that is being transferred to app 1 (receiver) and then this latter saves the file name using .getFileName() method. (These are the instances of the classes that are implemented in the exact order from both ends connected via Socket: BufferWriter > OutputStreamWriter > OutputStream **>** InputStream > InputStreamReader > BufferReader. )

After that, the sender writes the binary information from the file which is being sent into a byte array and sends it directly to the OutputStream. The content of this latter arrives to the receiver, who will then use it to create a copy of that file in the destination folder, using the file name that was saved before( FileInputStream > OutputStream > InputStream **>** FileOutputStream . ).

This is how the **pullFile** method works (Client.java, line 49, Parameters: String):

if we are taking as a parameter a file number, the app 1(receiver) sends the file number that wants to download from app 0(the sender)( BufferWriter > OutputstreamWriter > OutputStream > InputStream > InputStreamReader > Bufferreader ).

Finally, the sender checks to which index of the file list the number corresponds ( every server is storing the file paths on an ArrayList<String> ) and performs the “pushFile” method explained before, using the path that was at the index indicated.

This is how the **checkFiles** method works(Client.java, line 58, No parameters):

The app 0 requests the files to app 1, and this latter will send to app 0 the list of files found on the directory D:\TORrent\_1 ordered from 0 to N. (BufferWriter > OutputstreamWriter > OutputStream > InputStream > InputStreamReader > Bufferreader.)