# Documentation of DataTransferLibrary

## CONTENT

# 1 . Introduction :

- Data Transfer Library is a Java based console application which uses a server client based program that communicates via Sockets and is capable of transferring any content across multiple machines or the same machine.
- Multiple content as it can accept and transfer any kind of file extension i.e text, PDF , MP4 , png/jpeg , cpp , java , exe , zip etc.
- For Data transfer we have used a server program and client program in which the server can send the data and the client can download the same data .
- The library will work very efficiently if both the client and the server program are working on the same computer but if multiple machine transfers are happening that is we are using two different computers then both of these computers need to be on the same network for the library to work.
- The library is a high speed, real-time, scalable, secure, reliable data transfer library which can be used by cross platform applications.

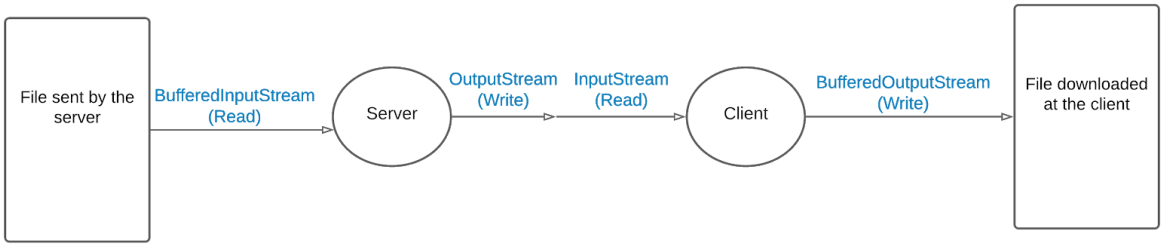



Figure : Basic working of the library

## Features -

**High Speed:**

Library supports data transfer rates of 500 mbps or more - The maximum average speed reached was 128-148 MB/s

**Real-Time:**

Library is able to transfer data within a few milliseconds to the other end .

**Scalable:**

Library is able to scale to multiple of 500 Mbps say 10 Gbps by changing library configuration - The library is using a byte[] of size 500,000 for reading the contents from the application and transferring that to the client. By changing the size of this byte array we can scale the library as required.

**Secure:**

The file that is selected from the application which is transferred to the client the filePath of that is encrypted on the server side before transmission using a random secret string key

which on the client side is decrypted using the same secret string key using AES encryption and Decryption.

Configuration file(.ini) is present on the disk for both the server side and client side .

Library does not leave any traces of data as the byte stream is flushed after each transaction

**Reliable:**

Library provides reliable data communication i.e. the data doesn't get lost due to temporary hiccups - If there is a connection loss on the server or the client side or slow connection is detected on the client side the library can deal with it very efficiently , as it stores the offset of the read contents from the application it is reading the data from and sends the contents from offset+1 when the connection resumes.If the connection loss takes place for more than 8 seconds then the program terminates with the %of transaction complete

In the event of irrecoverable loss, application is made aware of the data loss, the volume and current state (Recovered or not recovered) - as the amount of sent and downloaded data is being printed in the console for the user to see at every byte stream transaction.

At the time of connection loss if the connection resumes it prints "File can be retrieved " else "File can't be retrieved"

**Language & OS support:**

Library fully functions in java.

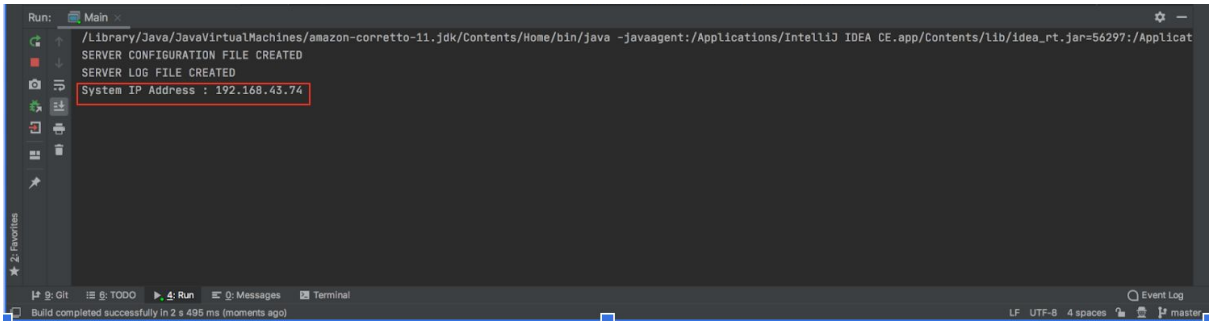
Both server and client side programs will run very efficiently for Windows & Mac operating systems (because we didn't have any linux based OS for testing).

**Additional Requirements:**

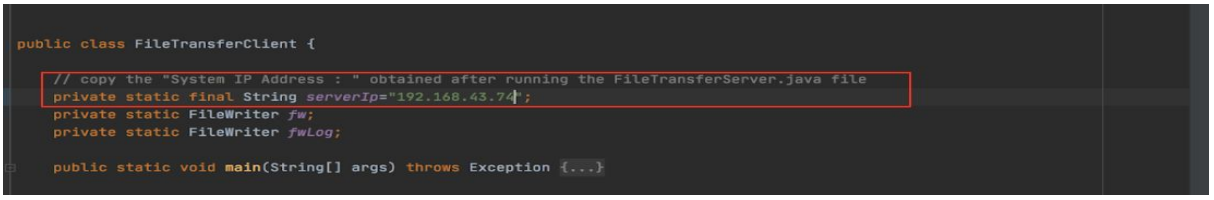
1. Statistics of data transfer
    a. Data rate + Total bytes Transferred / Received - The amount of data transferred in bytes and in percentage is printed on the console for each byte stream transferred/received.
    b. Bytes dropped if any - if the complete transaction doesn't take place the amount of successful bytes transfer completed and successful bytes received completed is printed on the console in percentage.
2. Application logs
    a. Informative logs (Configurable) + Error logs - Configuration file (.ini) and Log file (.txt) are created for server and client systems

## 2. User Manual & Troubleshooting Documents :

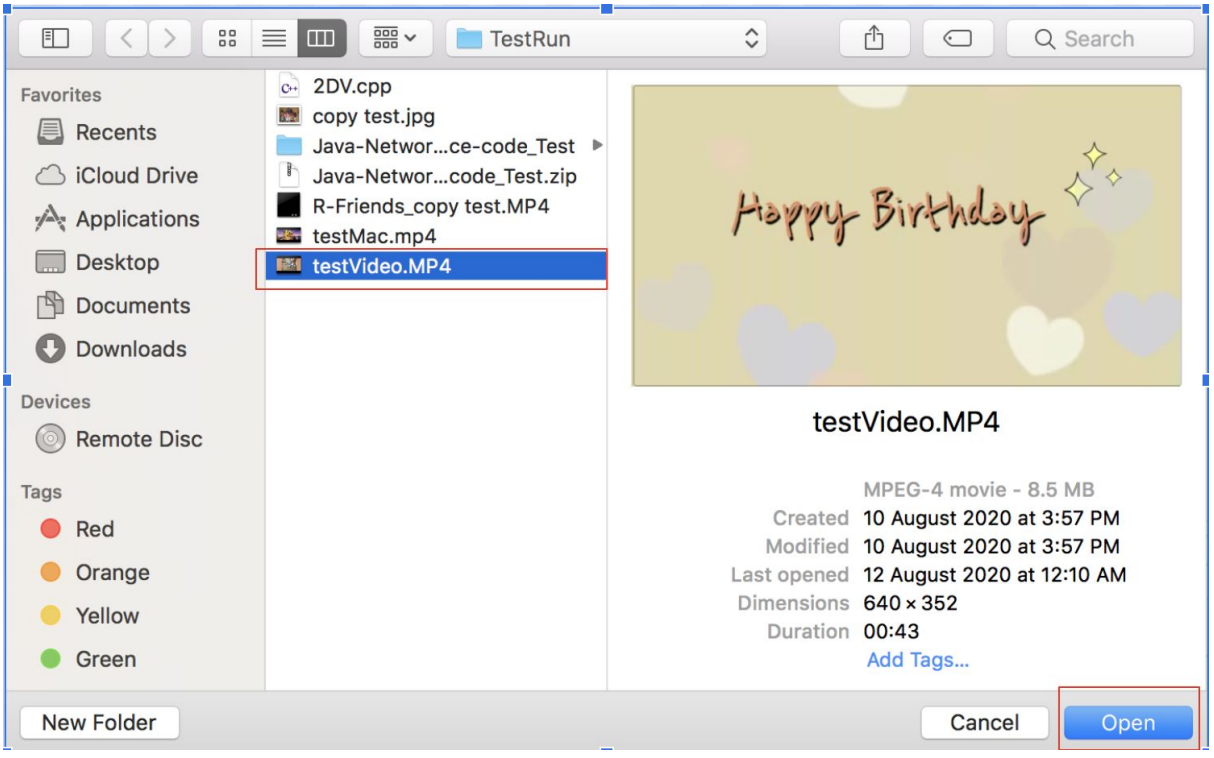
- The library supports the server to send the file and the client to download the file .
- The system that wants to send the file should run FileTransferServer.java the console will display the Ip address of the server needs to pass this to the client .




- The client side should run FileTransferClient.java with the server IP received from the server



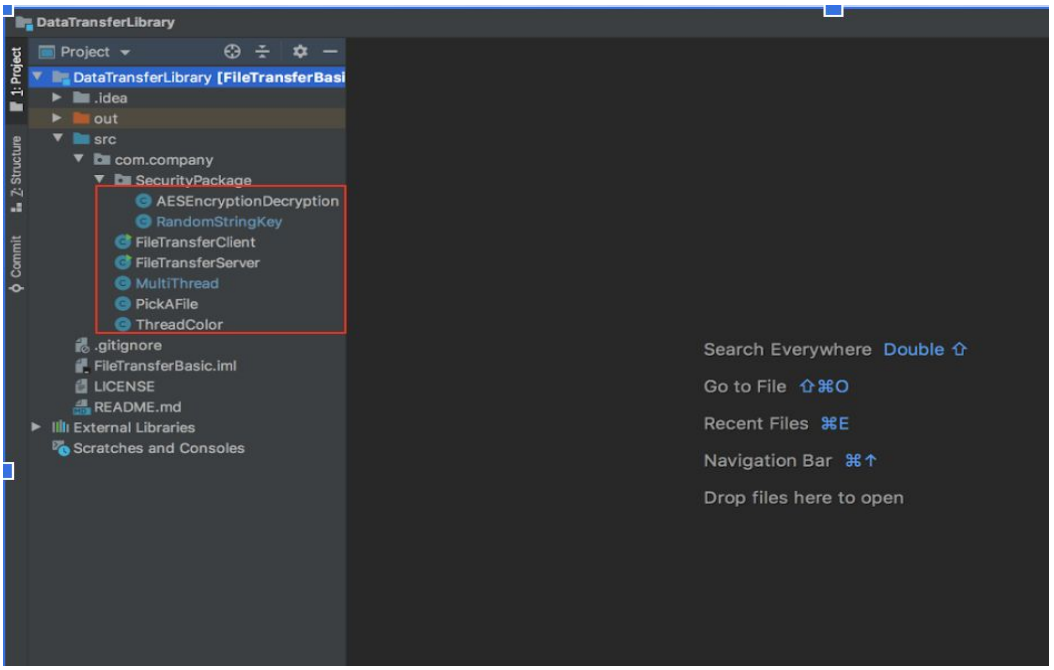

- In server system then select a file to be transferred




- Then the file will be Downloaded on the client system

## 3 . Design / Implementation description :

All the files present in the program :



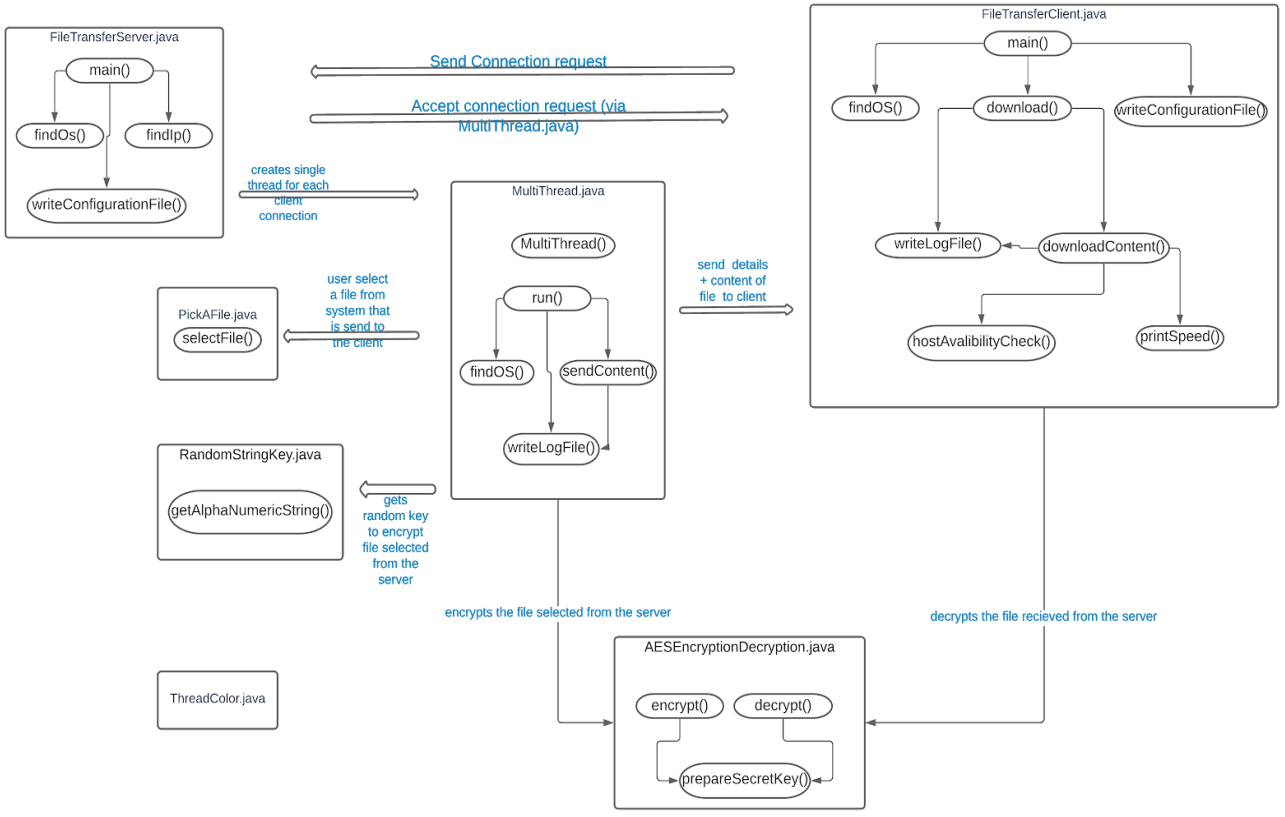




Figure : Basic layout of the library

1. FileTransferServer.java :

1. Used to Create ServerSocket @ port 5000 and accepts client connections. Multiple clients can be connected at a time FileTransferServer.java calls MultiThread.java which creates new Threads for each client.
2. Creates the Configuration and Log file of the server
3. All the other functioning of the server happens in the MultiThread.java file for each client.

Functions :

main() -

1. finds the Operating System of the server dependent on whether it is Mac or Windows creates configuration and log files in the home/Downloads/GoodProgram directory
2. finds the ip address of the server
3. Writes the server ip and port number to the configuration file
4. Calls the MultiThread.java constructor and then start the new thread created for the client

findOs() - Finds the Operating System of the server

findIp() - Find the IPaddress of the server

writeConfigurationFile() - used to write to the server configuration file

2. MultiThread.java :

1. Run each client connection on a new thread to decrease response time and increase efficiency.
2. All the functionalities of server is added in this file
3. Pick a file , encrypt this filePath , send the encrypted filePath + operating system of the server + length of the selected file to the client
4. Send the data of the selected file to the client

Functions :

run() -

1. Find the Operating System of the server
2. Calls .select() in PickAFile.java to select a file which will be send to the client
    a. If a file is selected :
        i. call getAlphanUmericKey () in RandomStringKey.java to get the random secret key for encryption
        ii. Call encrypt() in AESEncryptionDecryption.java to encrypt the filePath of the selected file
        iii. Find the length of the selected file
        iv. Send the encrypted filePath + secret key + length of file + OS of server to the client using OutputStream

v. Calls sendContent() to send the contents of the selected file to the client

b. If no file is selected

i. Send string “No String” + secret key=”Zero” + length of file =’0’ + OS of server to the client using OutputStream

3. Writes all the content/state of working to the server log file

MultiThread() - Constructor accepts client socket and object of server log file

findOS() - Finds the Operating System of the server

writeLogFile() - used to write to the server log file

sendContent() -

1. BufferedInputStreamReader is used to read the content of the selected file in a byte[] of size 50000
2. OutputStream is used to write this content to the buffer via socket

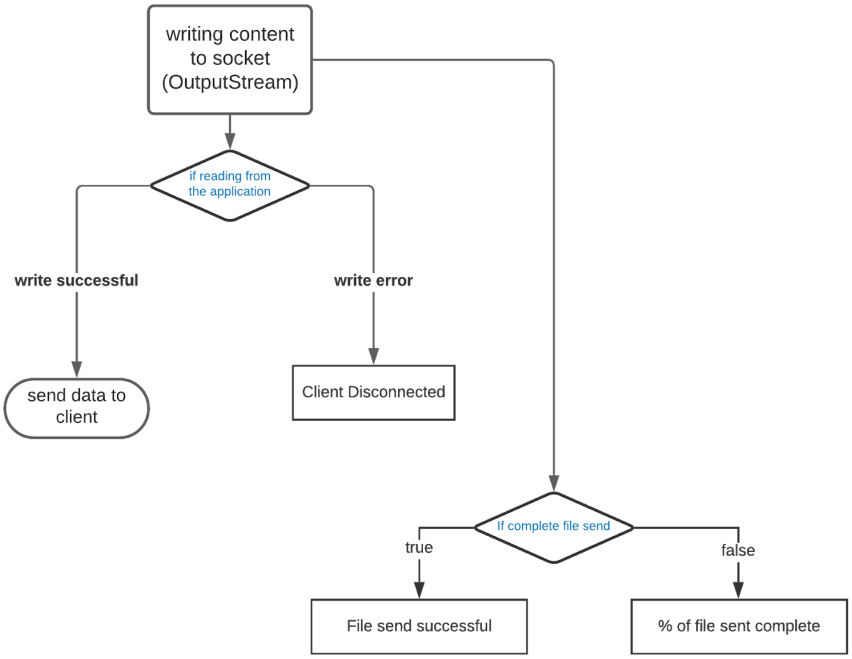



Figure : working of sendContent()

3. PickAFile.java -

1. Used to select a file from the server system via a Dialog box
2. User can chose a file or not

Function :

selectFile() -

1. user select a file from system : return the FilePath of the selected file
2. If no file chosen : return null

4. RandomStringKey.java -

1. Used to prepare the random secret key of type String and size 20

Function :
getAlphaNumericString() - creates random secret key of type String and size 20

5. AESEncryptionDecryption.java -

1. Used to encrypt and decrypt the filename of the selected file via AES technique
2. A secret key is passed to the functions that is received from RandomStringKey.java

Functions:
prepareSecretKey() - used to prepare the secret key via random secret key provided and SHA-1 and StandardCharset.UTF-8
encrypt() - used to encrypt the filename via prepared secret key
decrypt() - used to decrypt the file name via prepared secret key

6. FileTransferClient.java -

1. finds the Operating System of the client + creates configuration and log file
2. Create client socket and requests to connect with server socket via serverIP & port number here 5000 and write these details to the client configuration file
3. Receives the details of the file , decrypts the filename and downloads the file with the same name as it was in the server in home/Downloads/GoodProgram directory
4. Deals with connection loss via a server or client side
5. Finds out the average speed of the whole transaction

Functions :
main() -

1. finds the Operating System of the client dependent on whether it is Mac or Windows creates configuration and log files in the home/Downloads/GoodProgram directory
2. Asks whether the user wants to download or upload to the server.Our library only provides the client to download to the client so displays messages for upload or wrong choice chosen.
3. If client chose to download a file then download() is called which sends request to the server socket

download()-

1. Create client socket + connect to server socket(via serverIP & port number) if successful then perform below steps else “Server Not Connected”
2. Receives the encrypted fileName + OS of server + fileLength of file send + secret key from the server

a. If encrypted filename = “No String” : then no file is selected from the server print “No file selected”
b. Else : receive the details of the file and perform below steps

3. Decrypts the filePath using the secret key via AESEncryptionDecryption.java and then finds out the filename from the filePath
4. Finds the Os of the system - According to it creates the home/Downloads/GoodProgram directory and then creates the file inside this directory in the client system with the same name as it was in the server

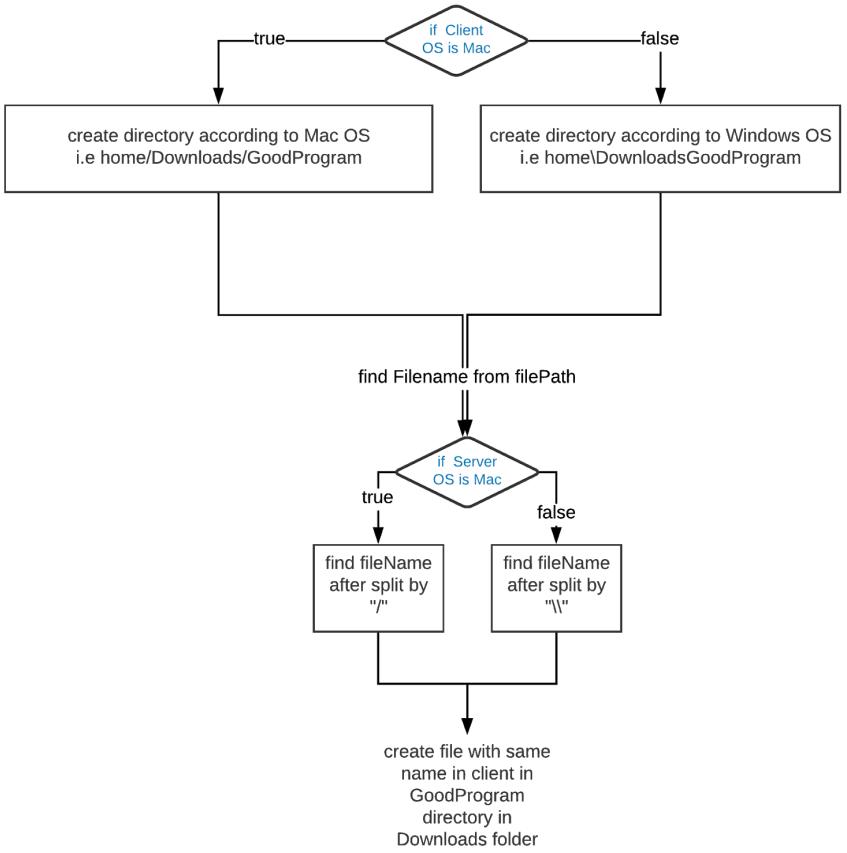



Figure : working of download()

5. Calls downloadContent() to receive/download the contents of the file from the server
6. Check if complete file transferred - “File saved successfully” ; else
   a. Check if client socket disconnected - “Client Disconnected” ; else - “Server Disconnected”
7. Writes all the content/state of working to the client log file

hostAvalibilityCheck() - ping the server i.e check whether it is reachable or not

printSpeed() - find the speed of the transaction each time the byte[] is read from InputStream

downloadContent() -

1. If Socket is connected then perform the below steps ; else Check if client socket disconnected - “Client Disconnected” ; else - “Server Disconnected”

2. Calls hostAvalibilityCheck() : to deal with connection loss or slow connection from both server and client
    a. If host is available then download data/resume download (from offset+1) else wait for host to become available (8 seconds)
    b. If host stays unavailable for long time then terminate client with the % of complete transaction

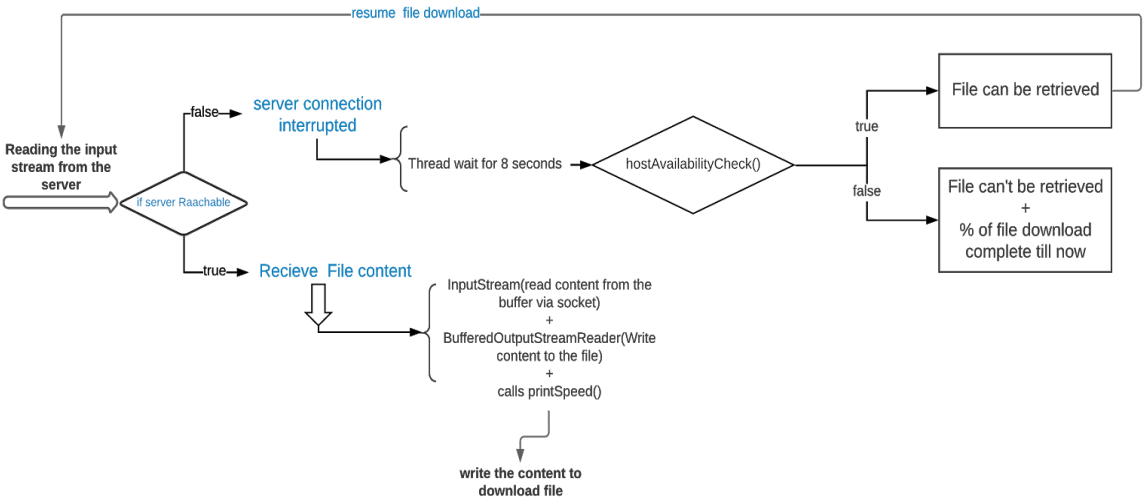



Figure : working of downloadContent()

3. Displays the Average speed with which the complete transaction take place
4. Writes all the content/state of working to the client log file

findOS() - finds out the OS of client
writeConfigurationFile() - used to write to the client configuration file
writeLogFile() - used to write to the server log file

7. ThreadColor.java -
    1. Used to provide unique colors like yellow , green , blue etc. to System.out.println threads so that the various features of the console based application can be differentiated from one another
    2. Is used by PickAFile.java , FileTransferServer.java , MultiThread.java and FileTransferClient.java

## 4 . Test cases/ Results :

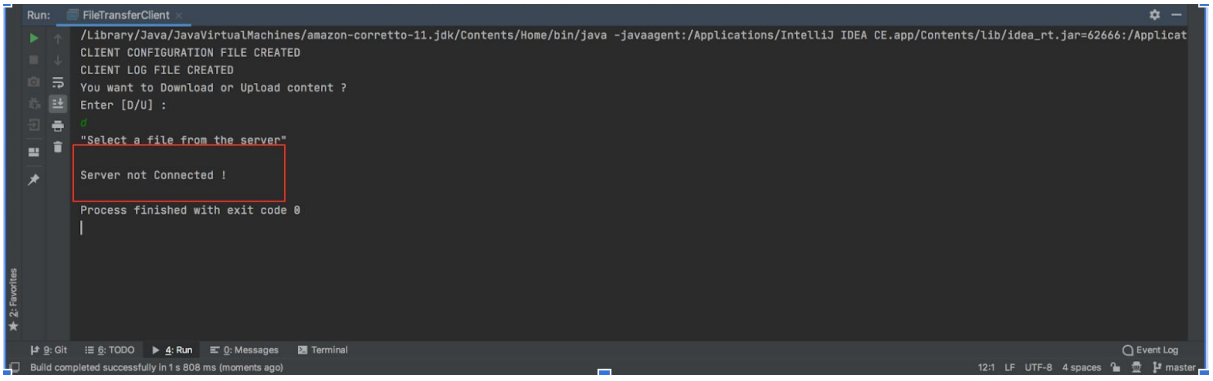



Figure : If first FileTransferClient.java is run then it will show Error “Server Not Connected”

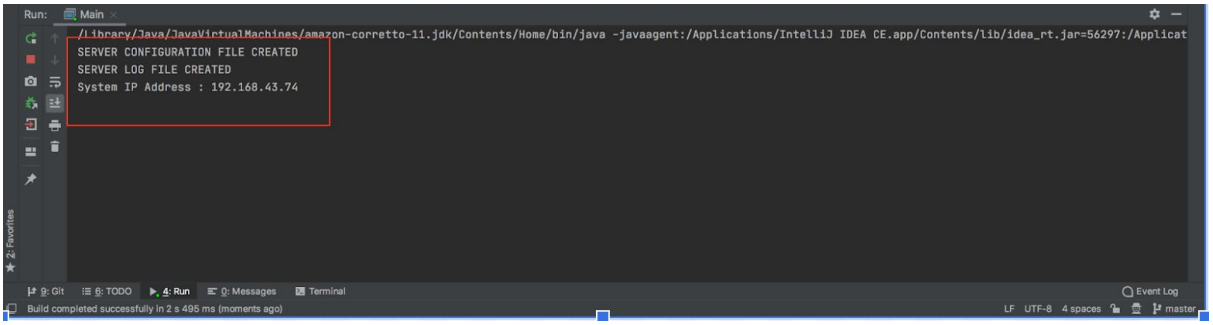



Figure : FileTransferServer.java is run and configuration & log files of server are created (according to the Os of server) + Ip Address of server is displayed

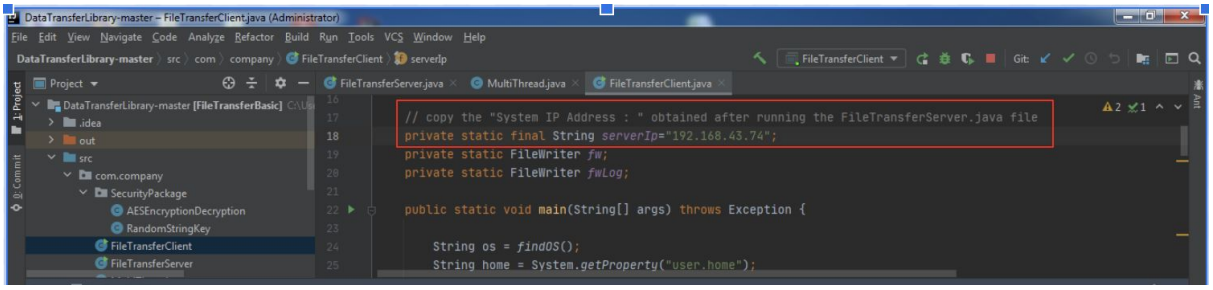



Figure : FileTransferClient.java will run with the Ip Address of the server

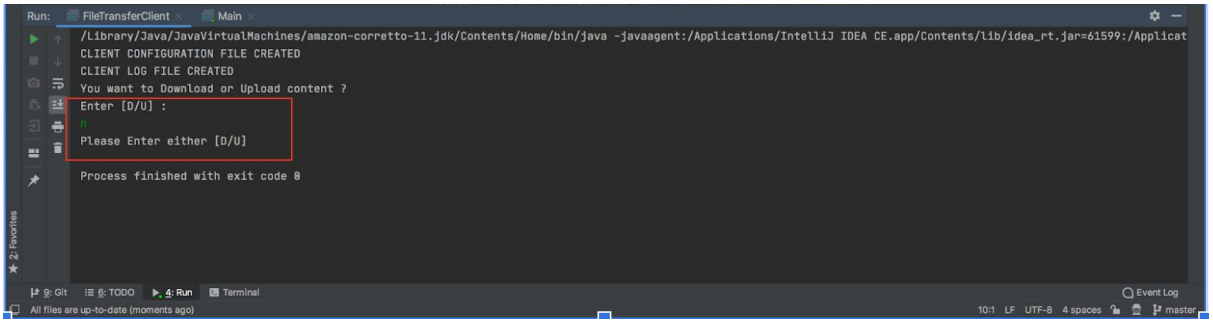



Figure : configuration & log files are created for client ; user is asked to download/upload file ; if user enters wrong choice then above message is displayed

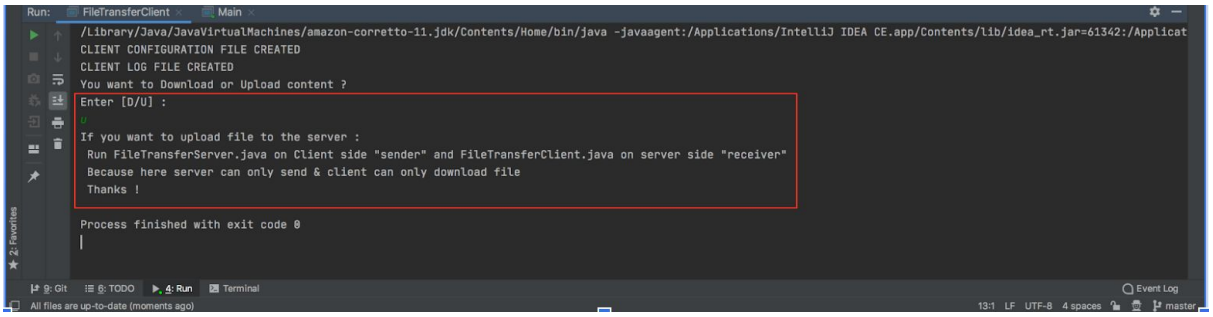



Figure : client don't support upload so message to run server in client is displayed

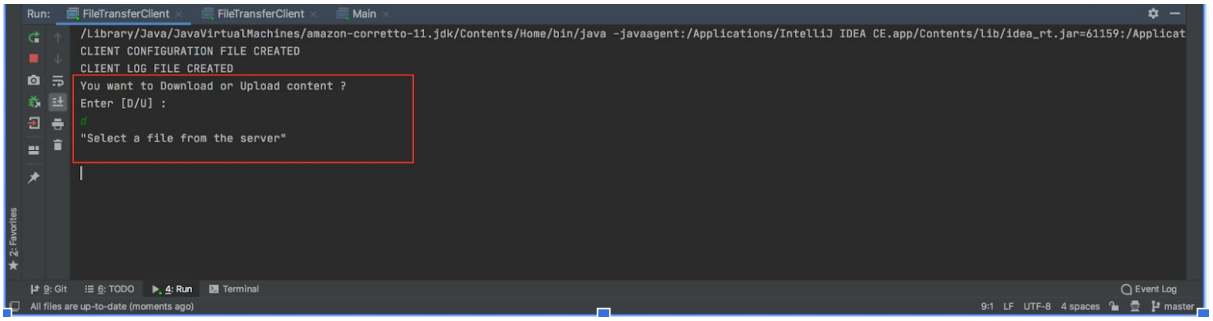



Figure : user chose to download file ; need to select a file from the server

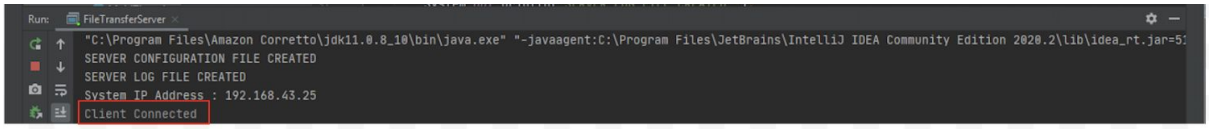



Figure : If download option is chosen by the client then "Client Connected" message will be displayed on server side as socket connection is build between client and server

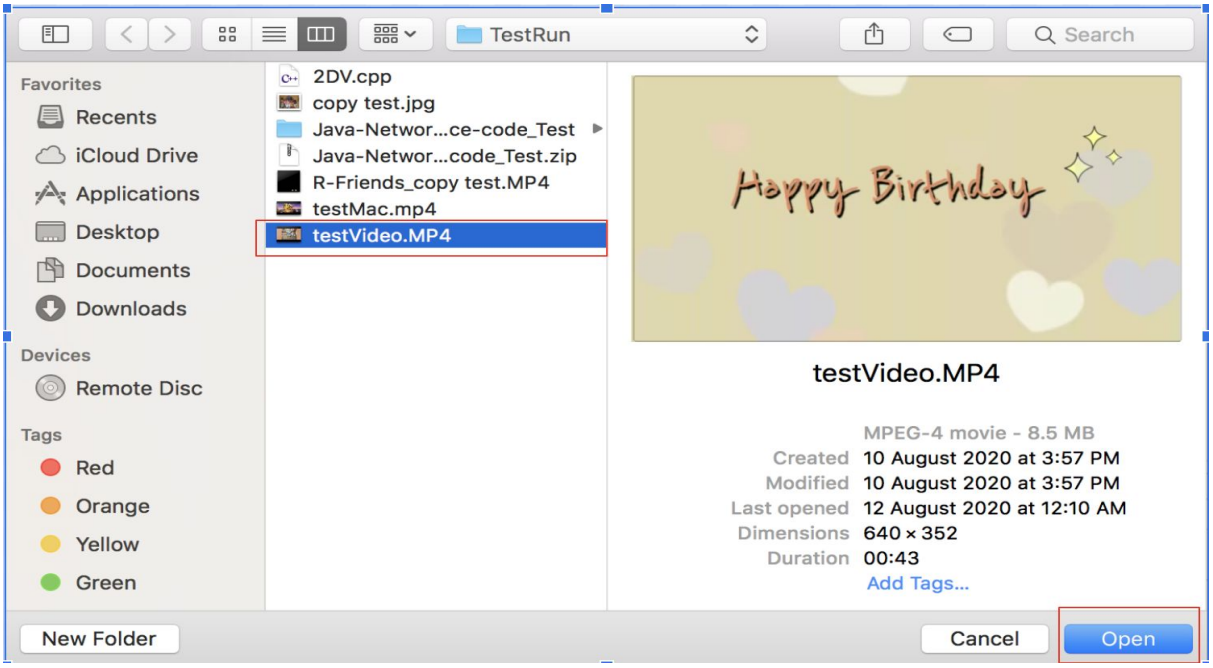



Figure : on server machine the user will select a file to transfer to the client and press open

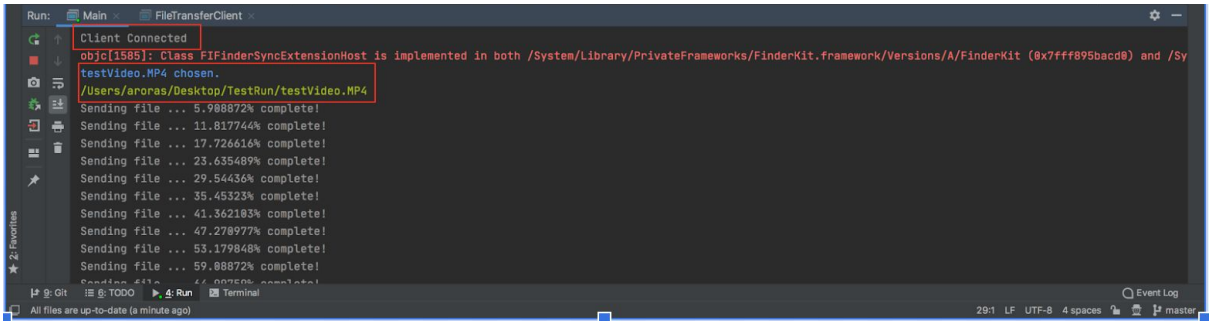



Figure : filePath of selected file to encrypt is displayed in green ; content of file will start transferring in byte[] of size 500,000 from server

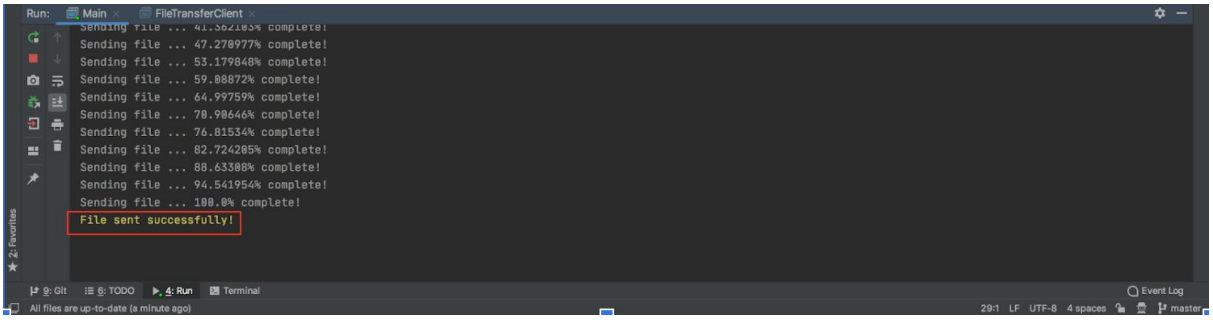



Figure : complete file send , the message “File sent successfully” is displayed

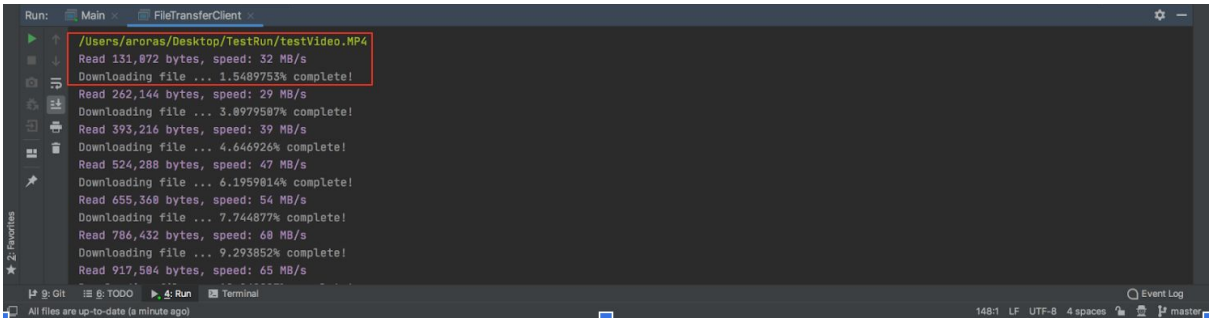



Figure : Client side will decrypt the filePath (in green after decryption) ,create download file with same name as in server and will start downloading

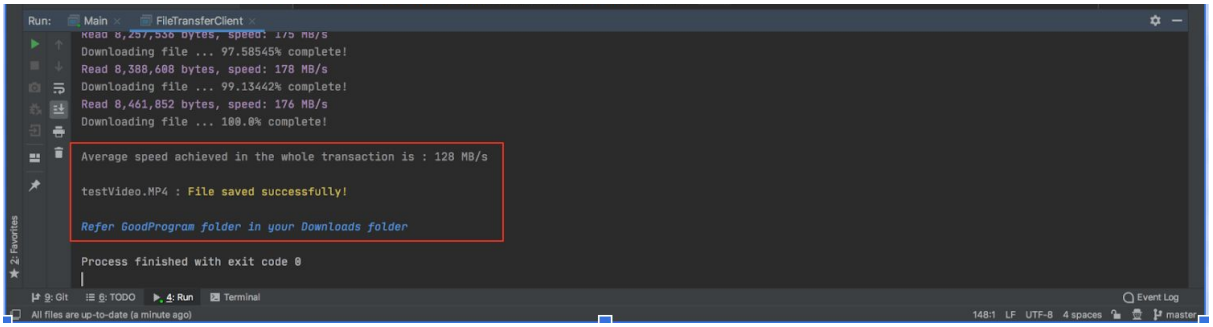



Figure : on complete download of the file “File saved successfully” message will be displayed along with the average speed achieved in the whole transaction

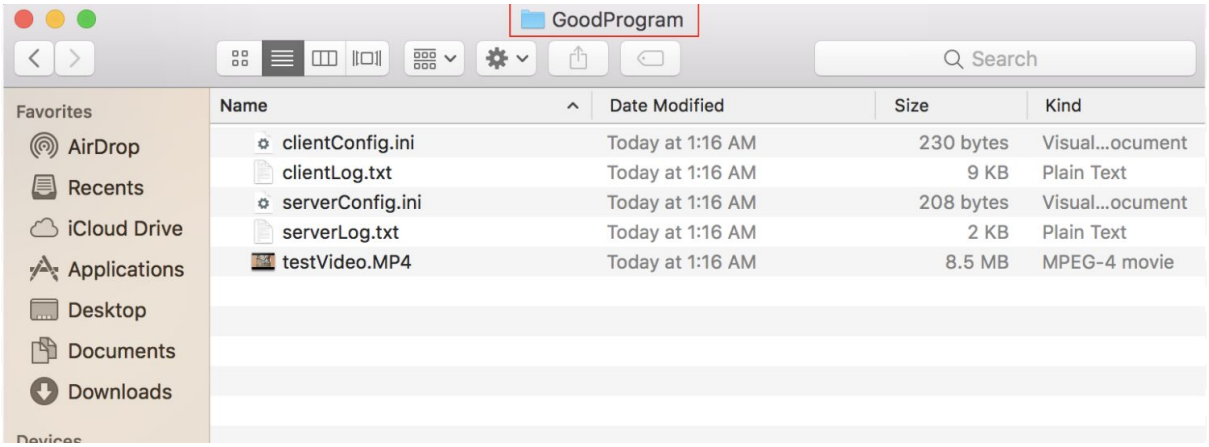



Figure : If client & server run on the same machine then GoodProgram directory will look as above

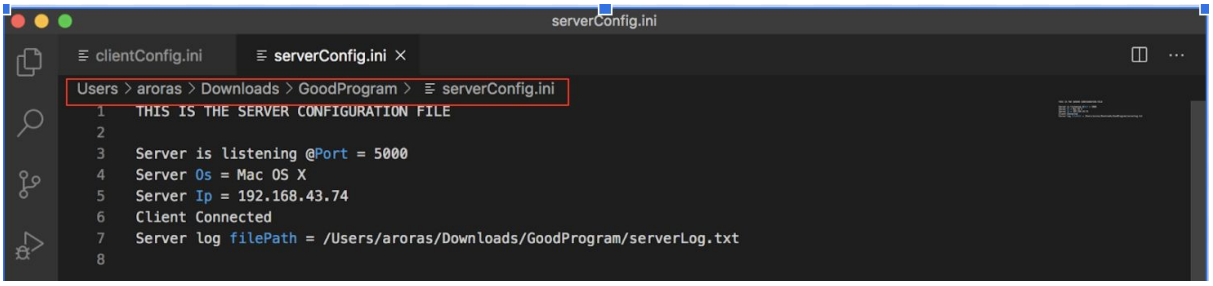



Figure : serverConfiguration file will store above server information + path of serverLog file

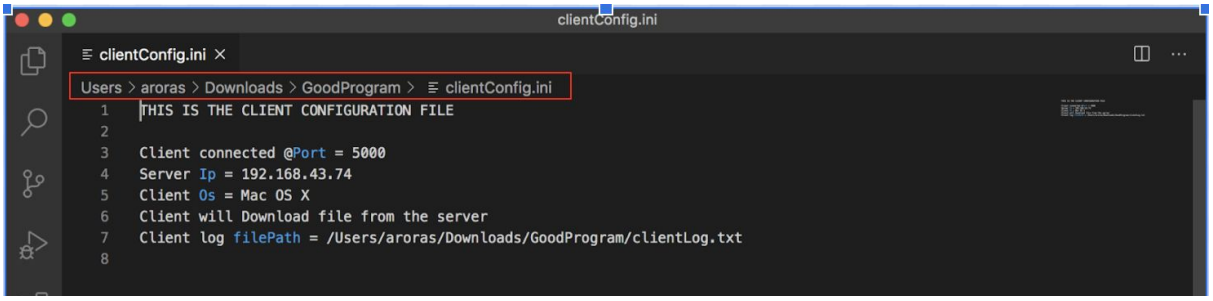



Figure : clientConfiguration file will store above client information + path of clientLog file

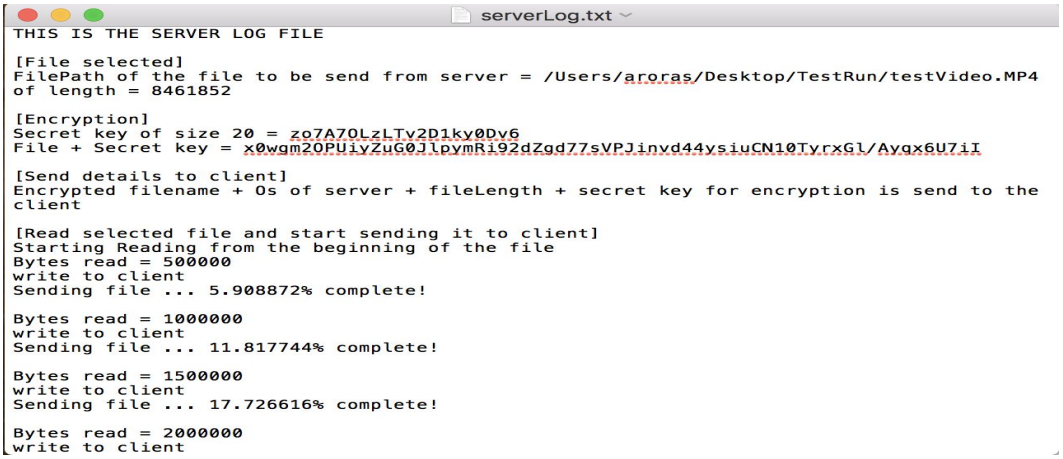


Figure : serverLog file will contain step by step data for the complete content send as above (including filePath+secret key+encrypted filePath+server OS+bytes send to client+Errors etc.)

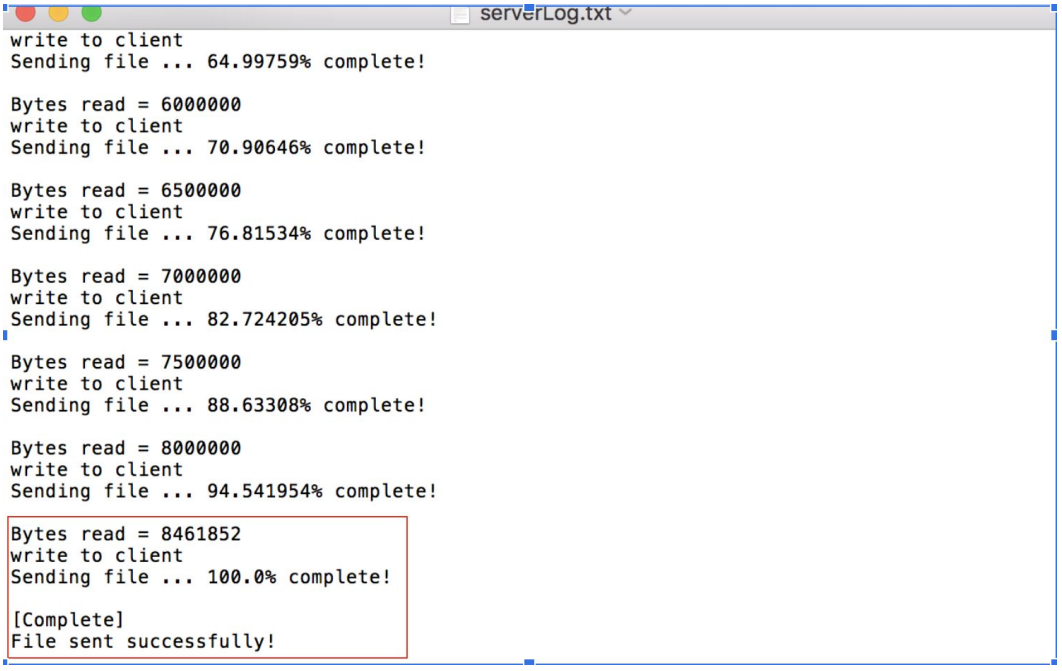



Figure : if complete file transfer is successful then serverLog will end at "File sent successfully"

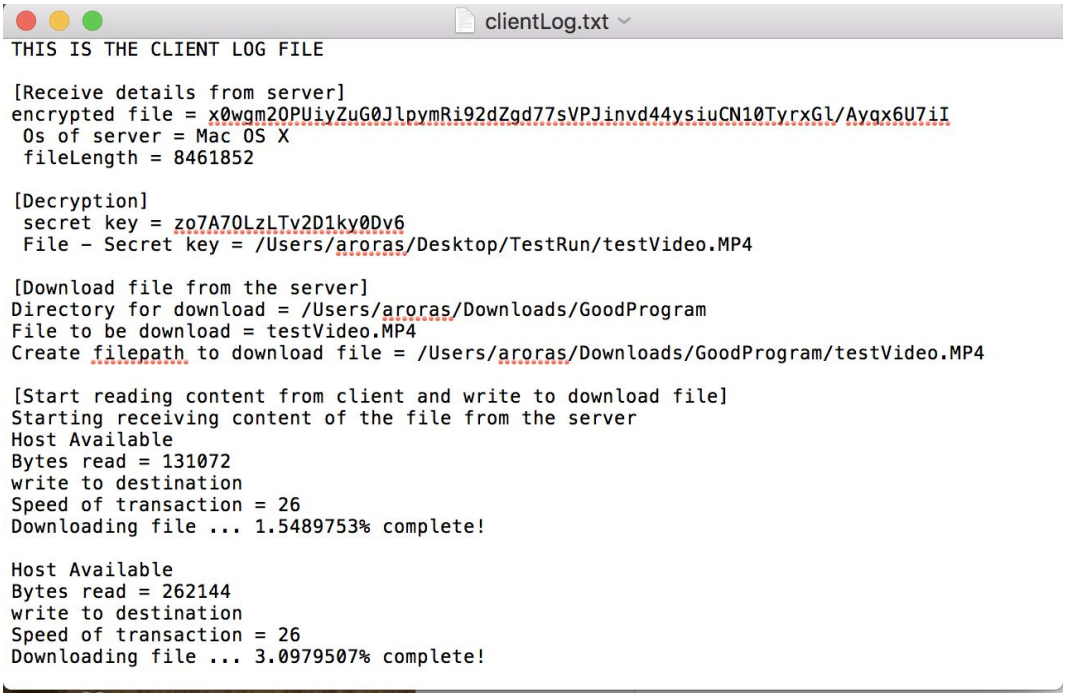


Figure : clientLog file will contain step by step data for the complete content received as above (secret key+encrypted & decrypted filePath+server & client Os+bytes received by client+Host availability+speed of transaction+Errors etc.)

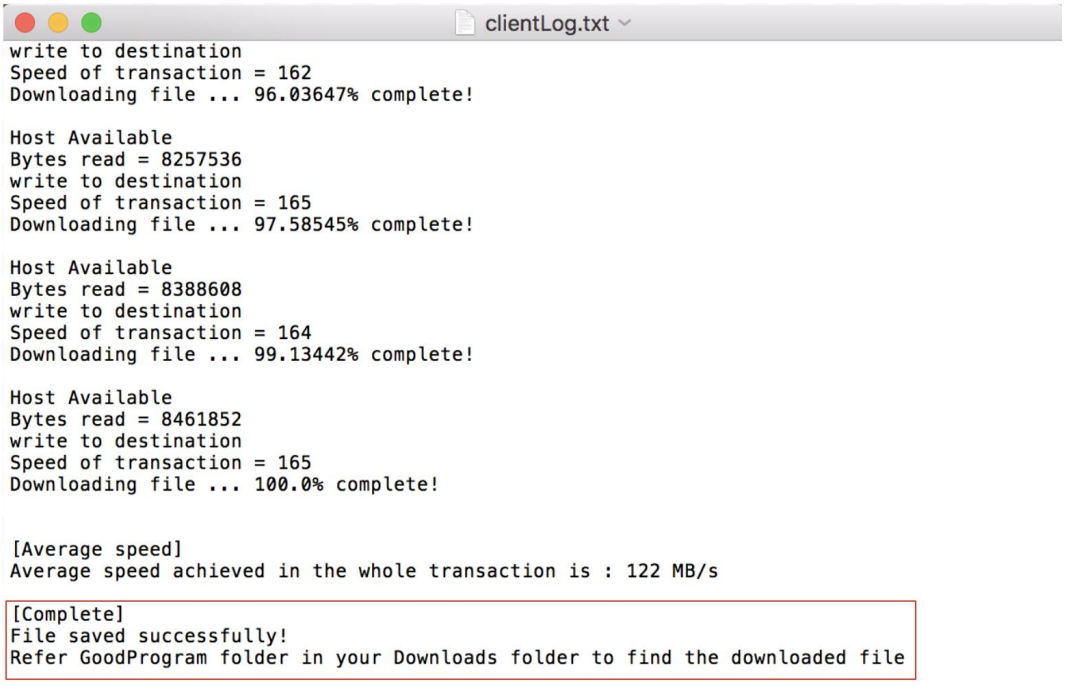



Figure : if complete file download is successful then clientLog will end at “File saved successfully” along with average speed achieved in the complete transaction

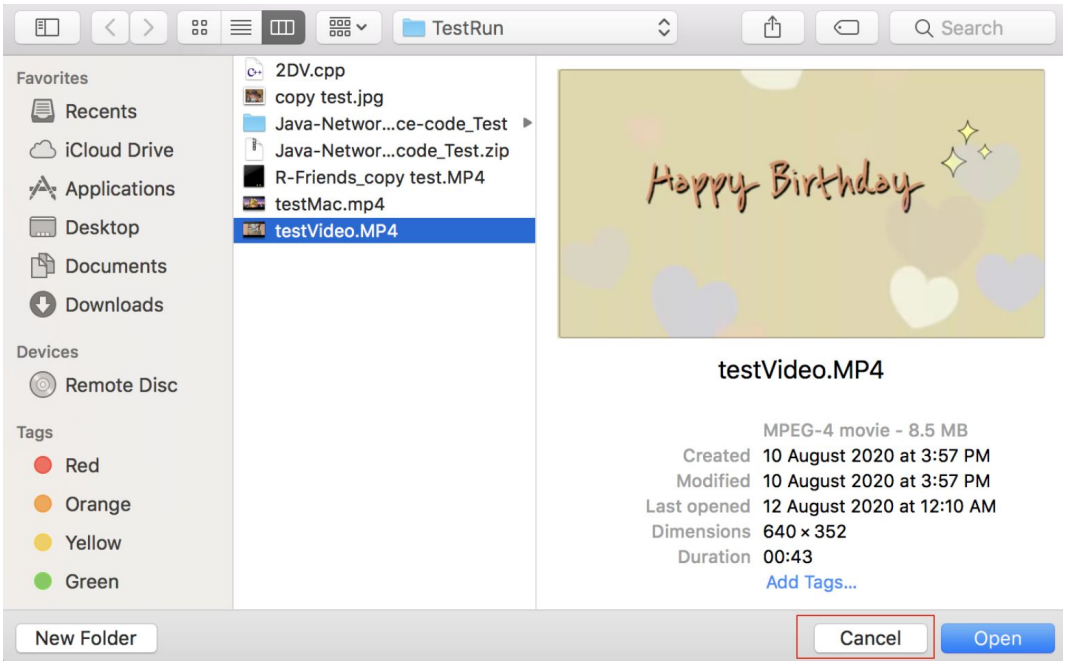



Figure : user may chose not to select any file from the server (i.e no file will be transferred to the client) and press cancel then “No file selected is send to the client”

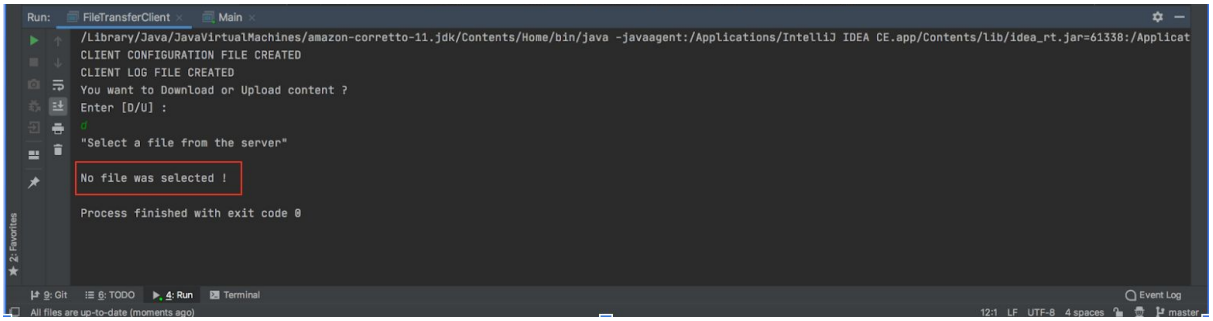



Figure : user did not choose any file so “No file was selected” messages in displayed in client

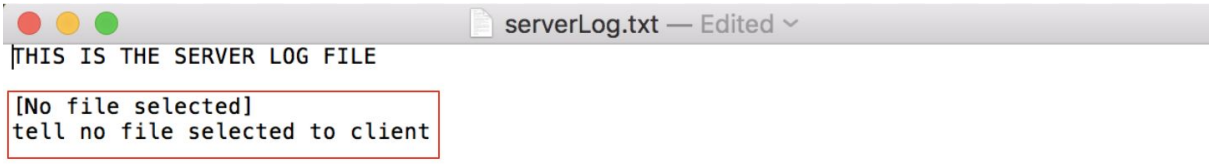



Figure : serverLog will contain “tell no file selected to client”

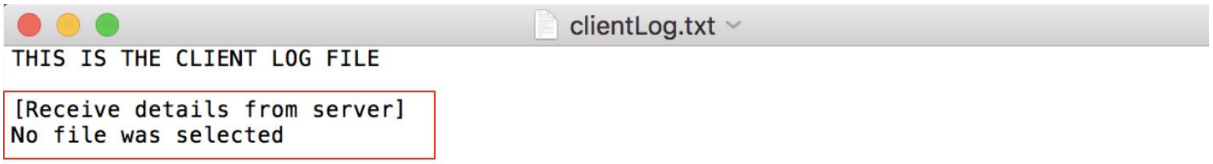



Figure : clientLog will contain “No file was selected”

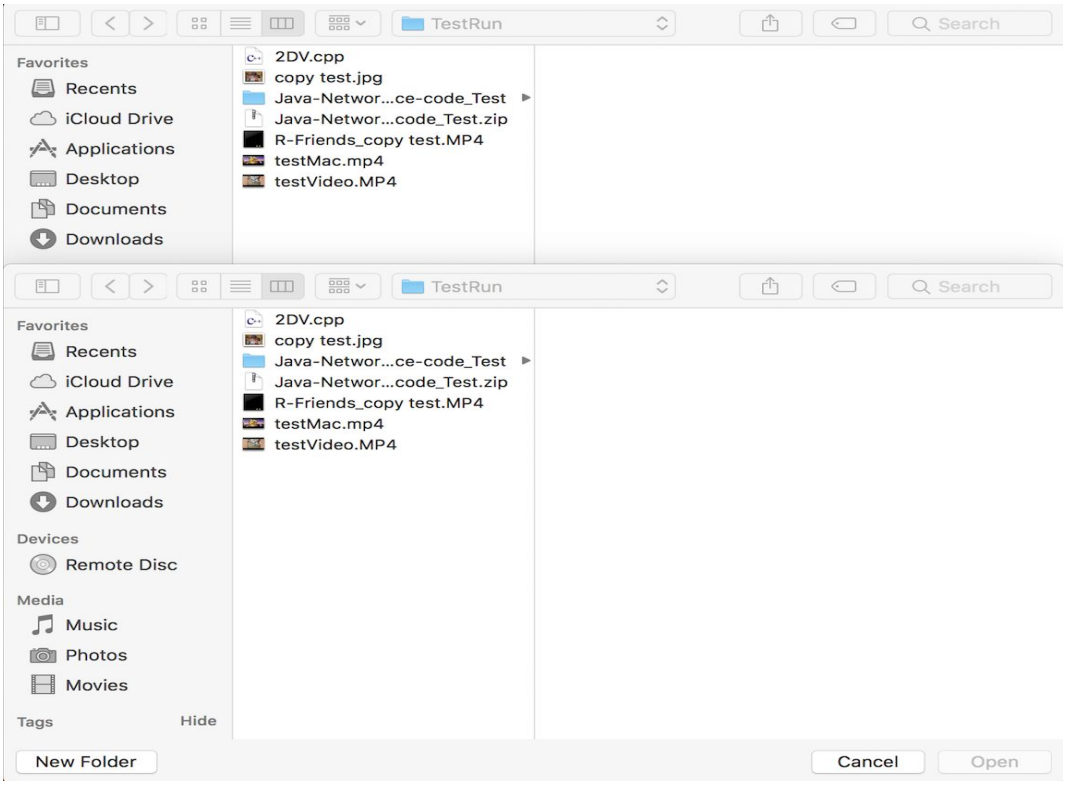



Figure : Multiple clients can access the server (and select different files) at the same time

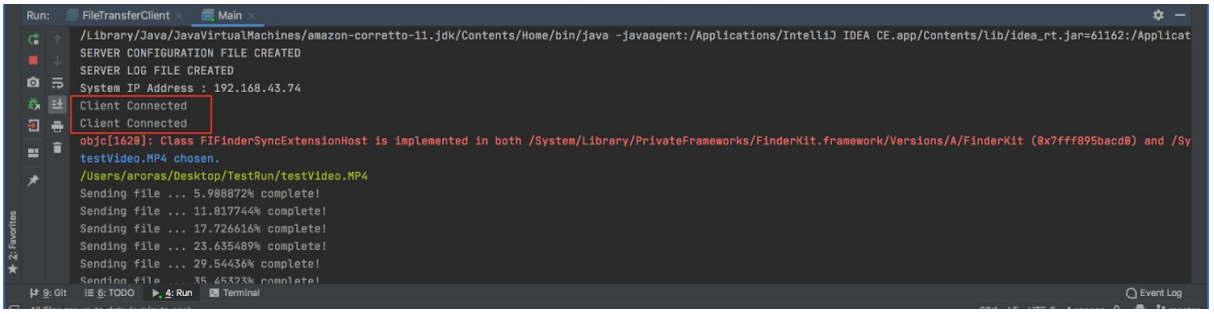



Figure : Two clients were connected(communicating with the server) at the same time therefore server side is displaying two “Client Connected” messages

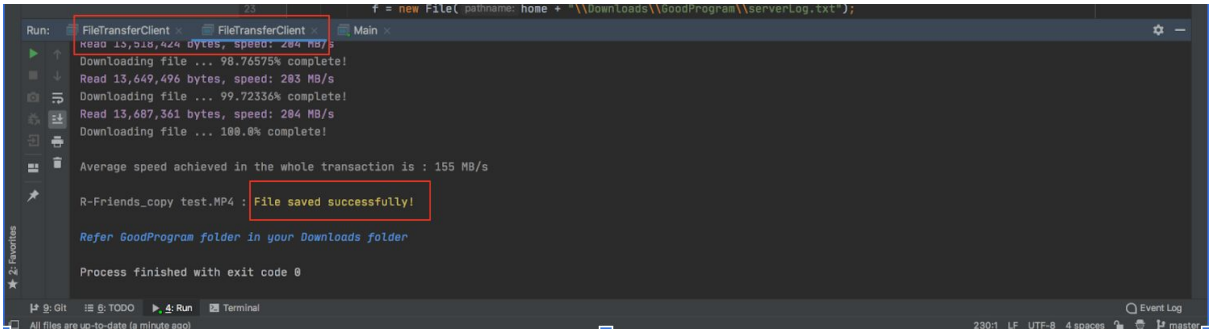



Figure : Both the clients selected two different files and download of both the files was successful at the same time

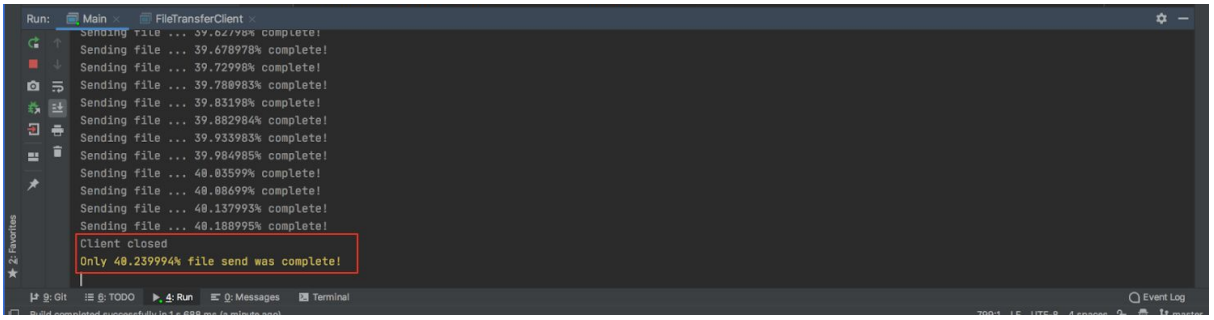



Figure : running FileTransferServer.java (server side) while server is sending the file, if client side accidentally terminates then the server will display the message “Client closed” along with the percentage(%) of file sent complete up till now

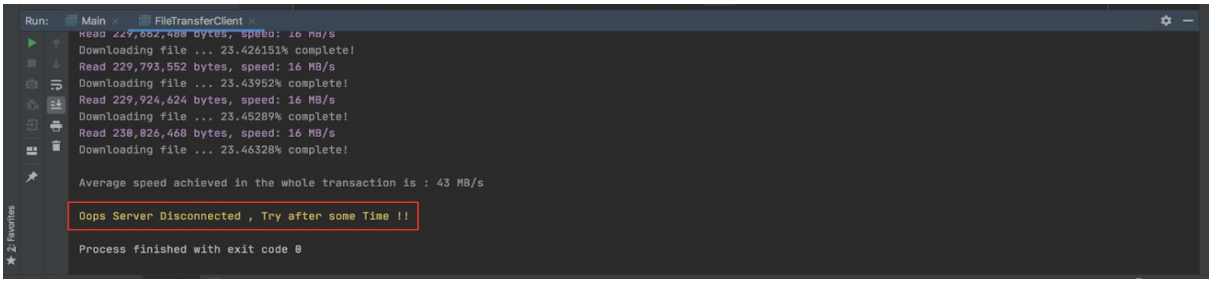


Figure :running FileTransferClient.java (client side) while client is downloading the file, if server side accidentally terminates then the client will display the message “Oops server Disconnected, Try after some time” along with the percentage(%) of file download complete and Average speed achieved in the transaction up till now

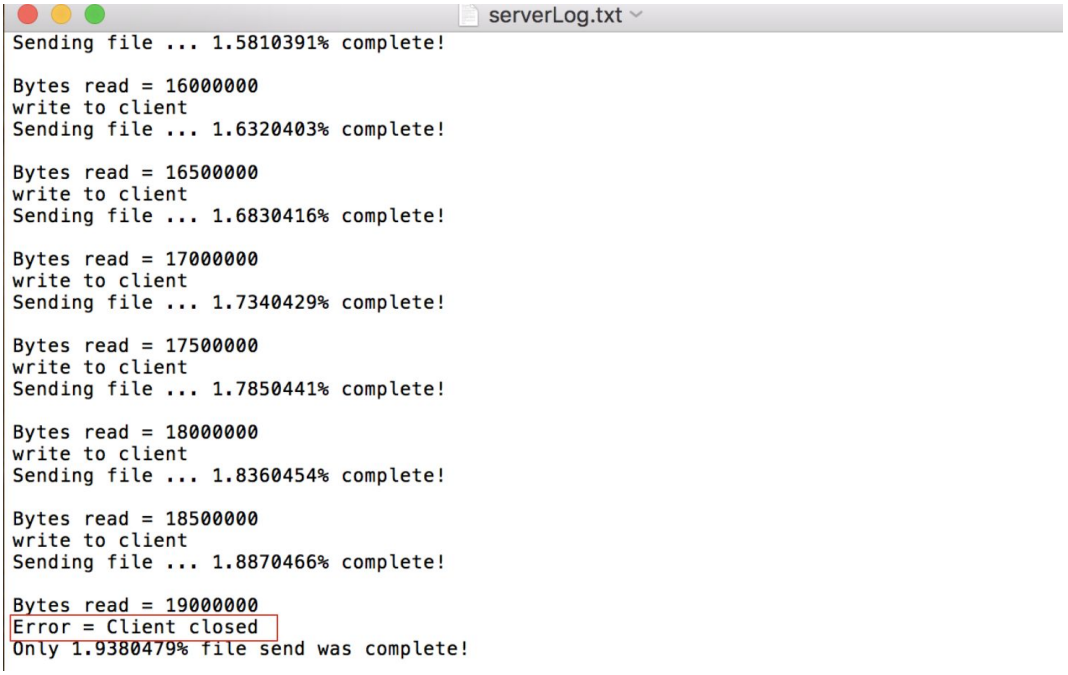



Figure : serverLog file will store the accidental client terminated as “Error = Client Closed”

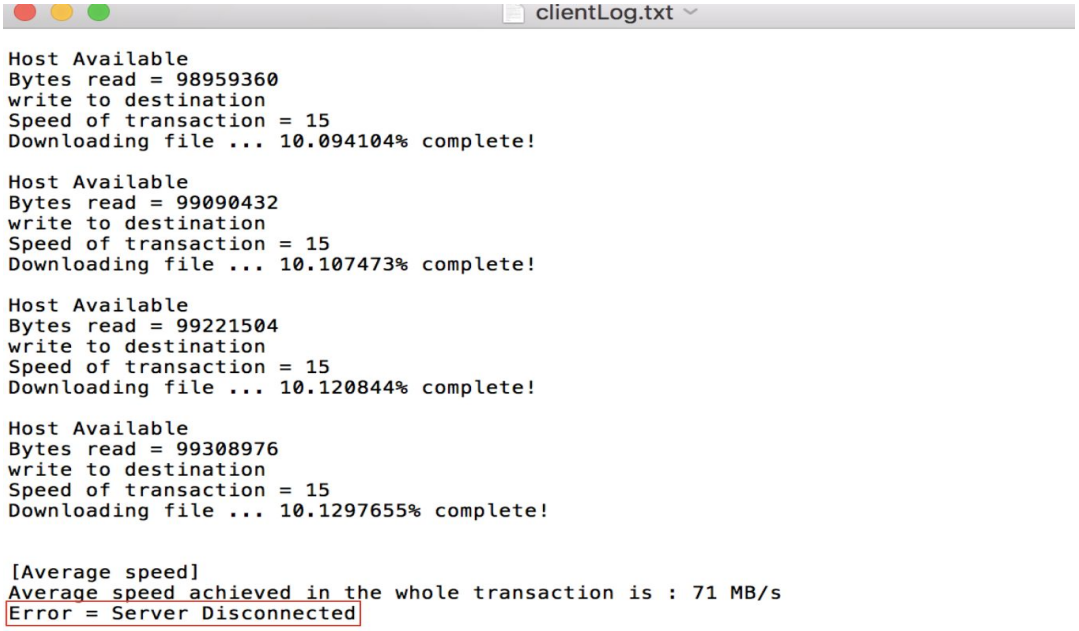



Figure : clientLog file will store the accidental server terminated as “Error = Server Disconnected”

**To test “connection Loss” or “slow connection” functionality :**

- Server running on Windows 10 OS
- Client running on Mac OS
- As tested library is creating files and working efficiently for both Mac & Windows Os

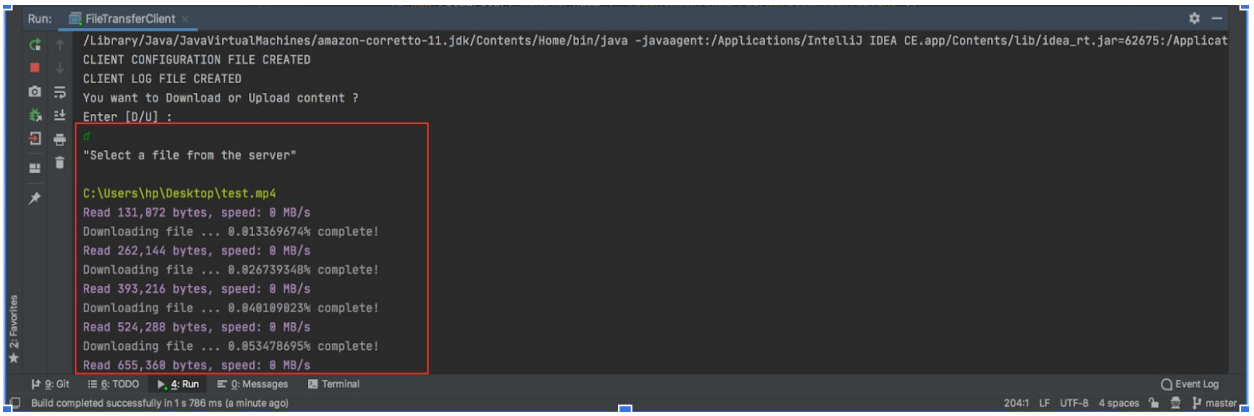



Figure : Run FileTransferClient.java to download “test.mp4” from the server

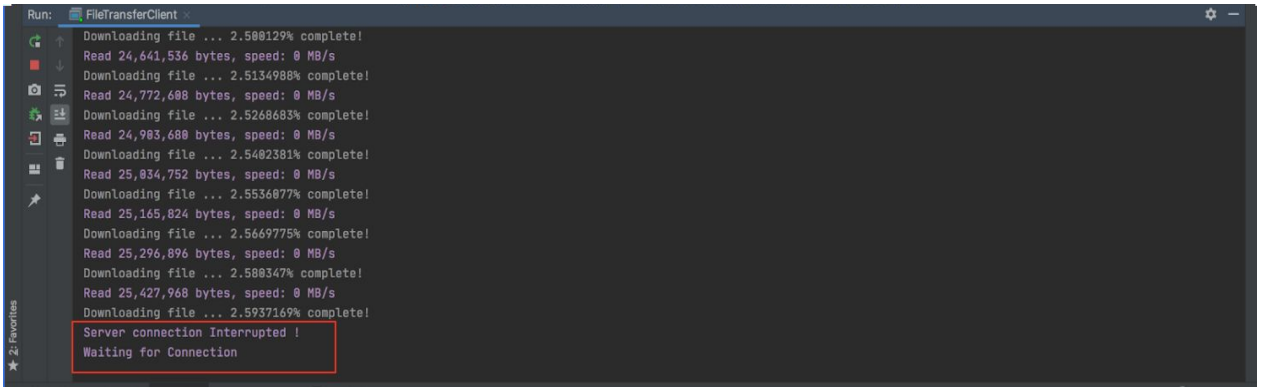



Figure : if connection Loss or slow connection happens from either client or server side then above messages will be displayed and client thread will sleep for 8 seconds (wait for connection to be available again)

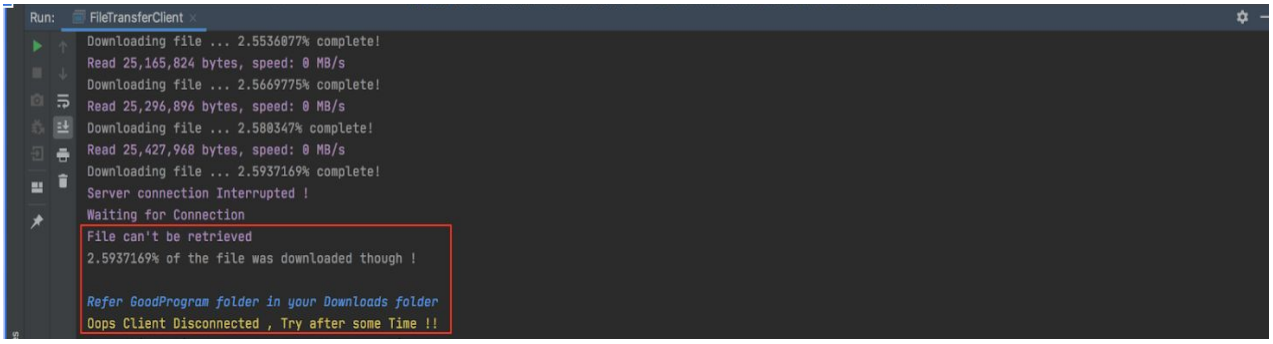



Figure : If connection isn’t available after thread awakes then client terminates with “File can’t be retrieved” message and percentage(%) of file download completed uptil now

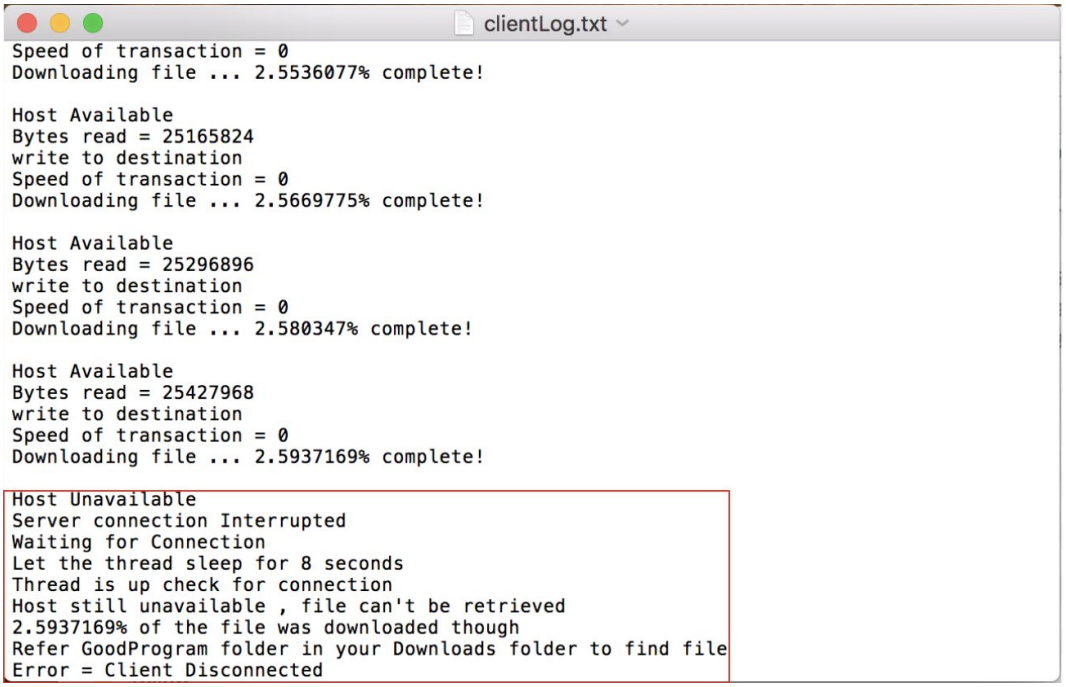



Figure : clientLog file will store the data of "Host Unavailability" + interrupted connection + thread sleep/awake + continuing unavailability +percentage(%) of file download completed uptil now ; since the server will be unreachable therefore client will terminate as "Error = Client Disconnected" and "File can't be retrieved" message

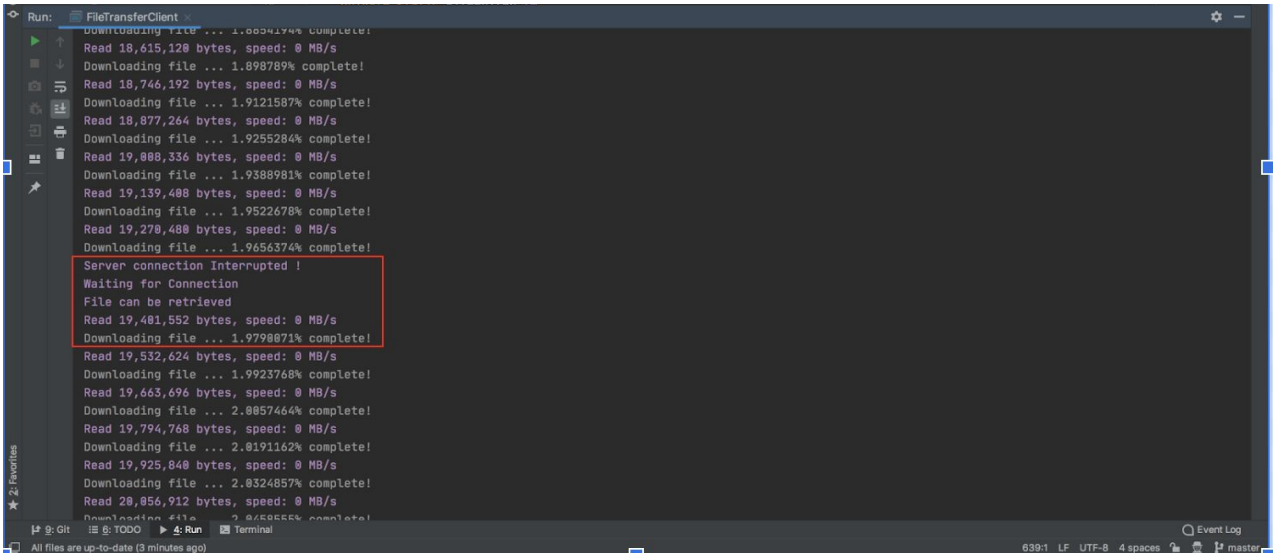



Figure : If connection becomes available after thread awakes then client will continue executing (i.e resume download of file) with "File can be retrieved" message and percentage(%) of file download completed uptil now and speed achieved

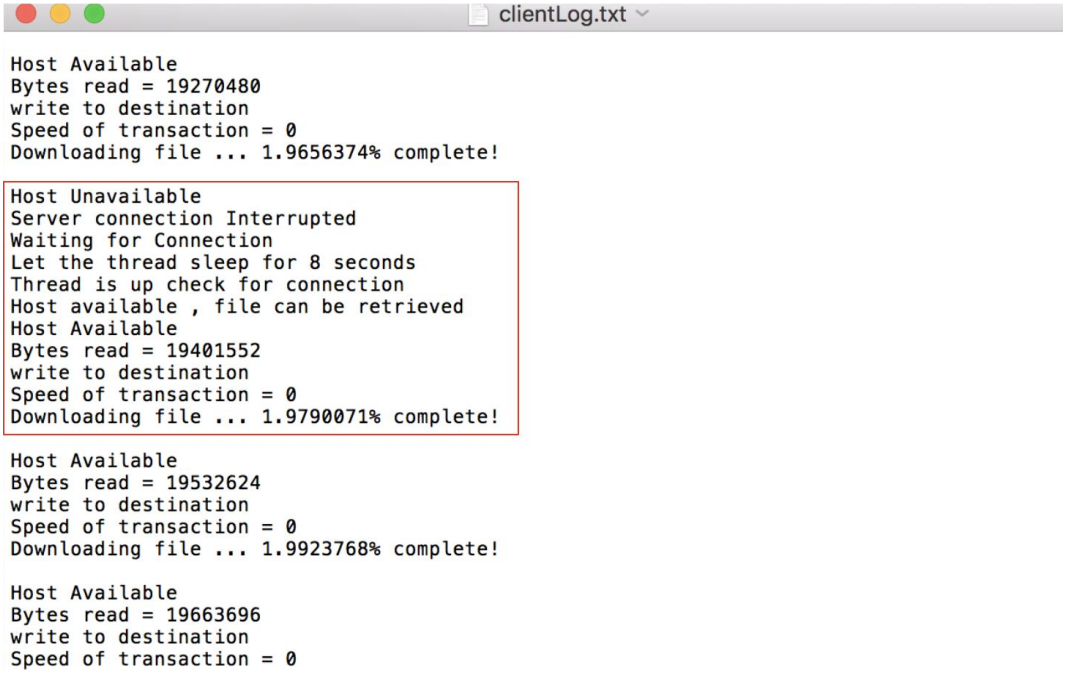



Figure : clientLog file will store the data of “Host Unavailability” + interrupted connection + thread sleep/awake + Availability of host and “File can be retrieved” +percentage(%) of file downloading +speed achieved; since the server will become reachable after thread awakes therefore client will not terminate and resume download and library will function as it was before the connection Loss or slow connection