

Documentation of DataTransferLibrary



CONTENT

1. Introduction
 - a. Features
 - b. Additional Requirements
2. User manual & Troubleshooting Documents
3. Design / Implementation
4. Test cases / Results

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.

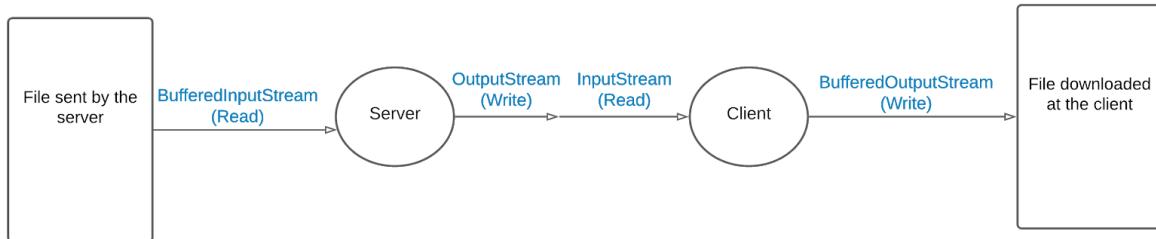


Figure : Basic working of the library

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.

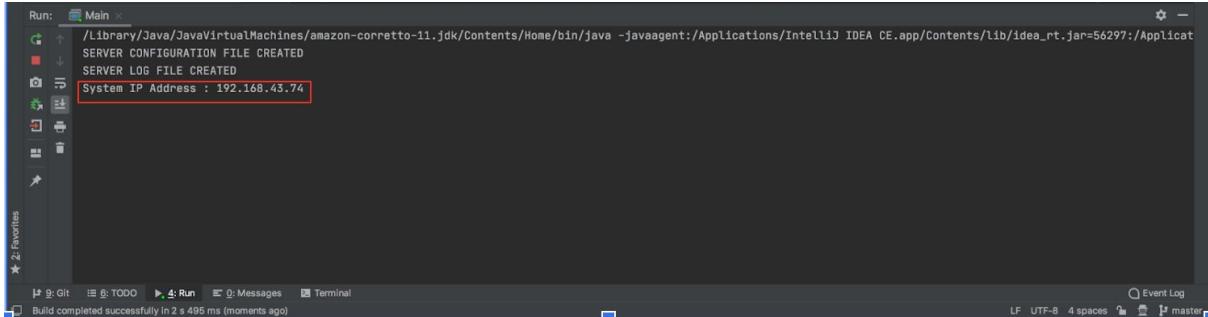
Both server and client side programs will run very efficiently for Windows & Mac operating systems (because we didn't have any linux based OS for testing).

Additional Requirements:

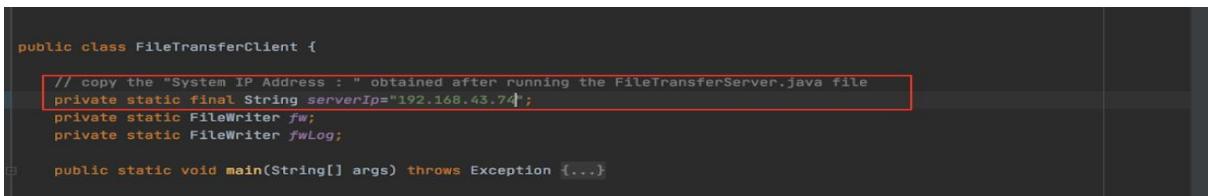
1. Statistics of data transfer
 - a. Data rate + Total bytes Transferred / Received - The amount of data transferred in bytes and in percentage is printed on the console for each byte stream transferred/received.
 - b. Bytes dropped if any - if the complete transaction doesn't take place the amount of 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 :

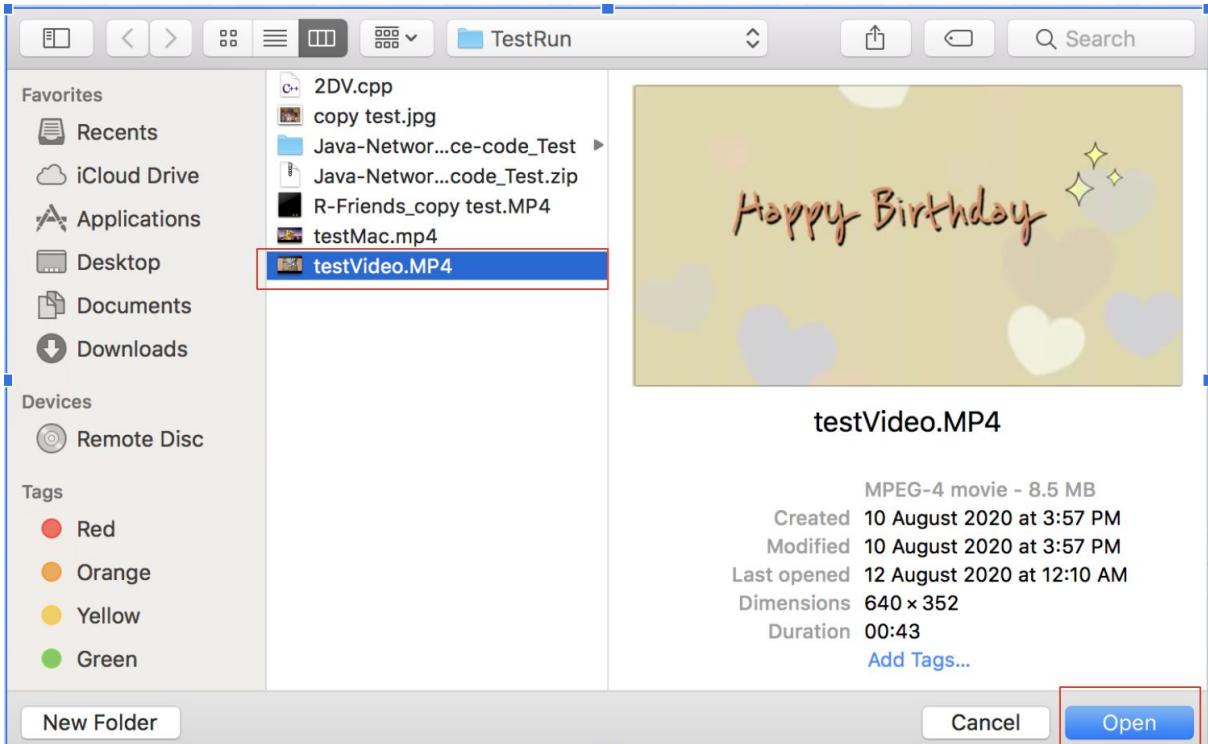
- The library supports the server to send the file and the client to download the file .
- The system that wants to send the file should run FileTransferServer.java the console will display the Ip address of the server needs to pass this to the client .



- 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 :

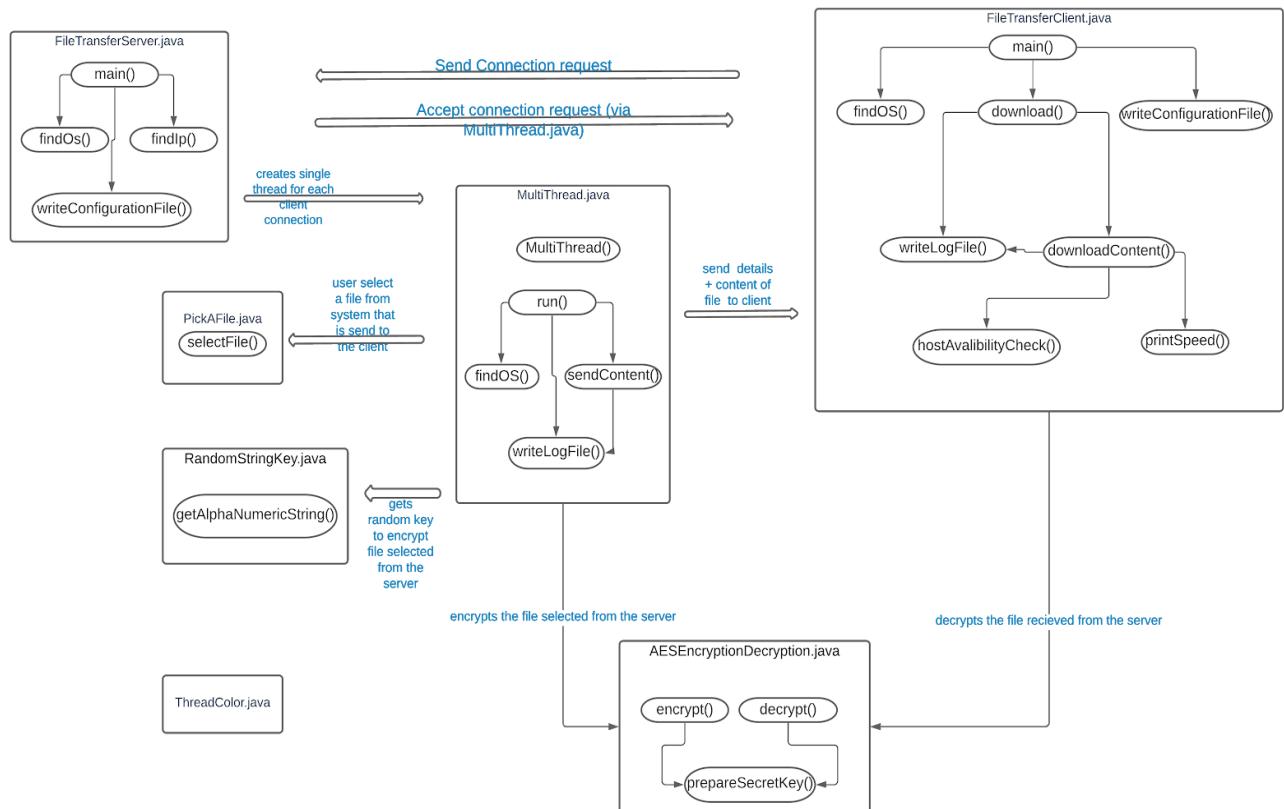
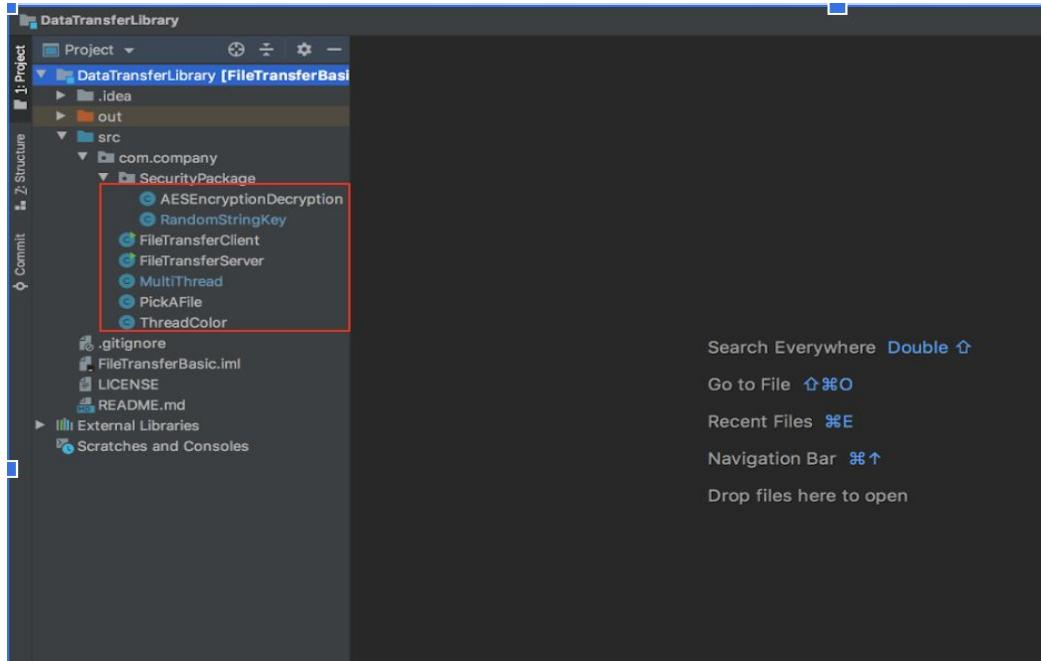


Figure : Basic layout of the library

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. BufferedReader 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

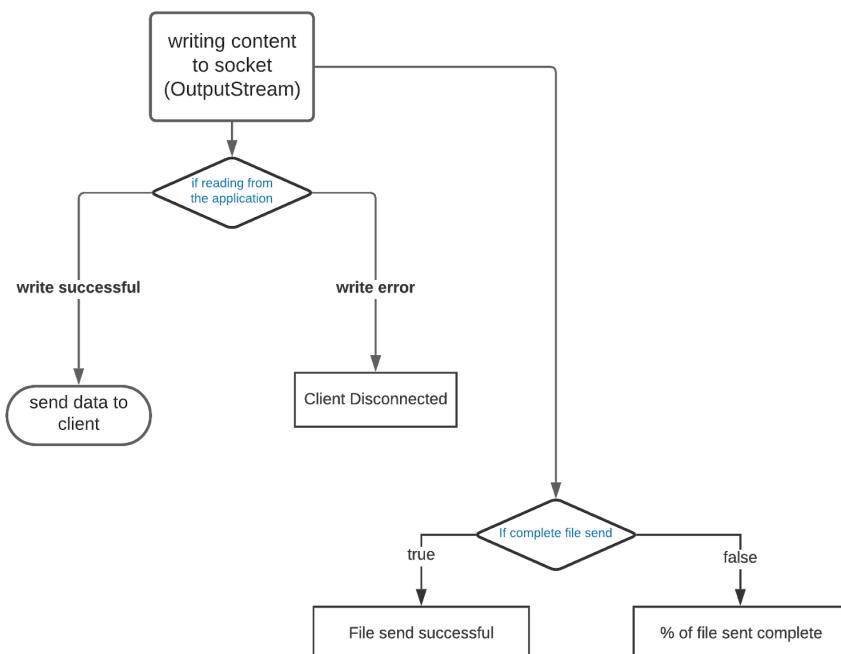


Figure : working of sendContent()

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

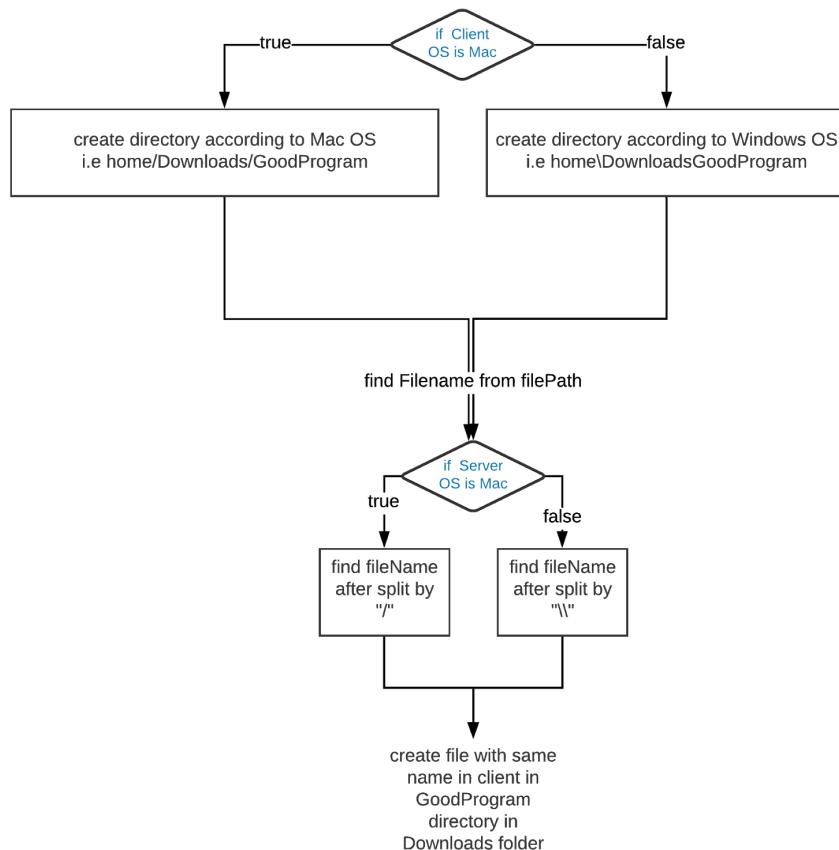


Figure : working of download()

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
- hostAvailabilityCheck() - 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 hostAvailabilityCheck() : 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

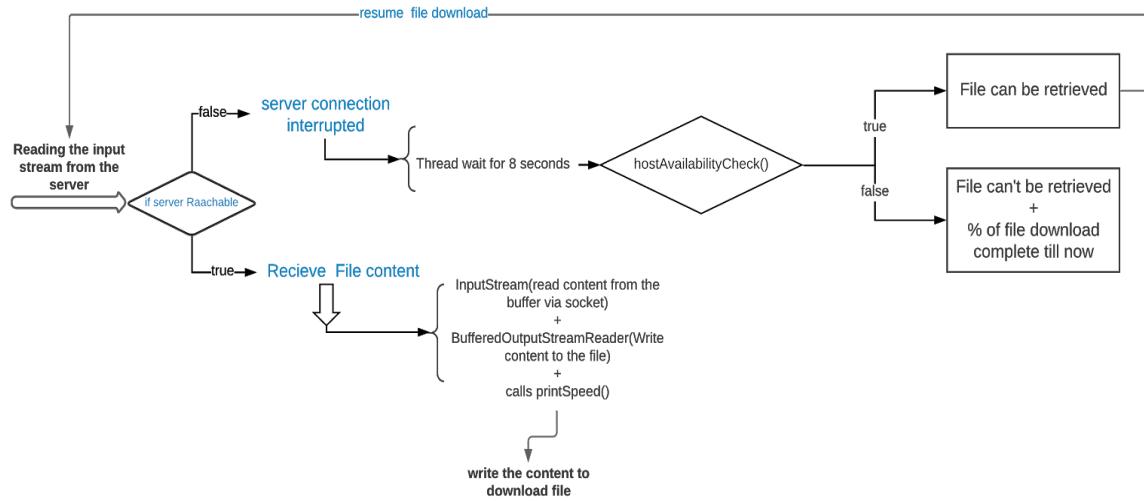


Figure : working of downloadContent()

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

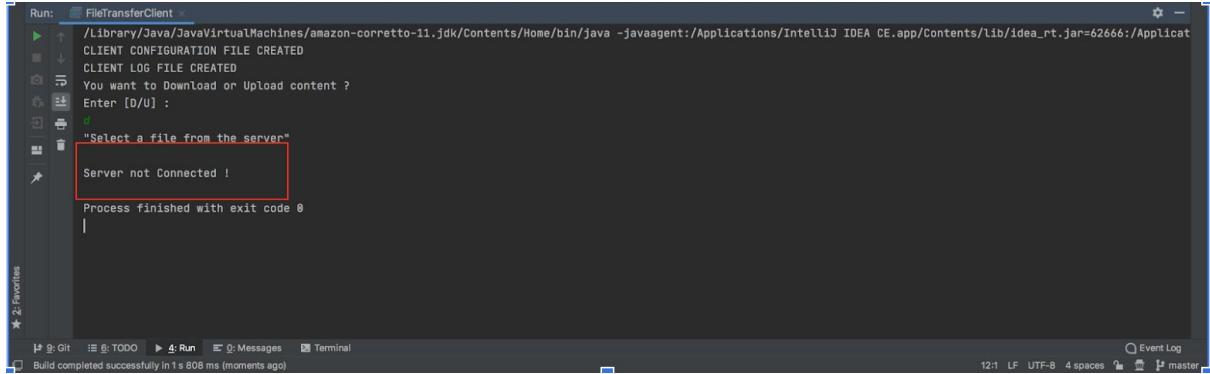
`writeConfigurationFile()` - used to write to the client configuration file

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

7. ThreadColor.java -

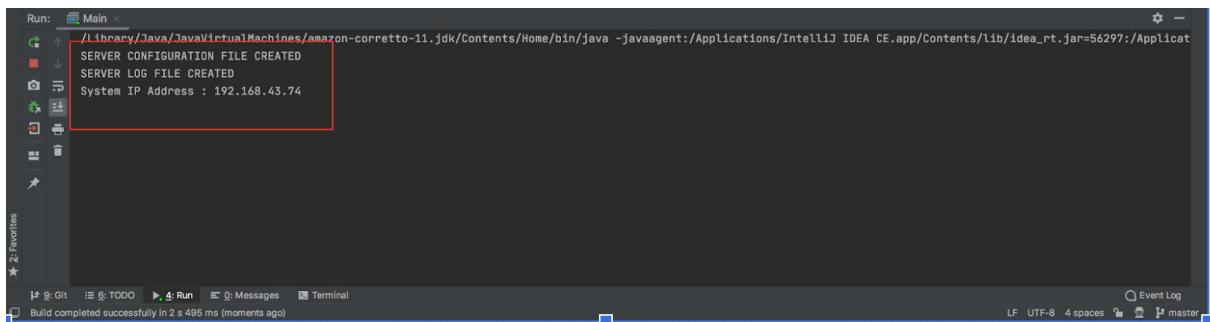
1. Used to provide unique colors like yellow , green , blue etc. to `System.out.println` threads so that the various features of the console based application can be differentiated from one another
2. Is used by `PickAFile.java` , `FileTransferServer.java` , `MultiThread.java` and `FileTransferClient.java`

4 . Test cases/ Results :



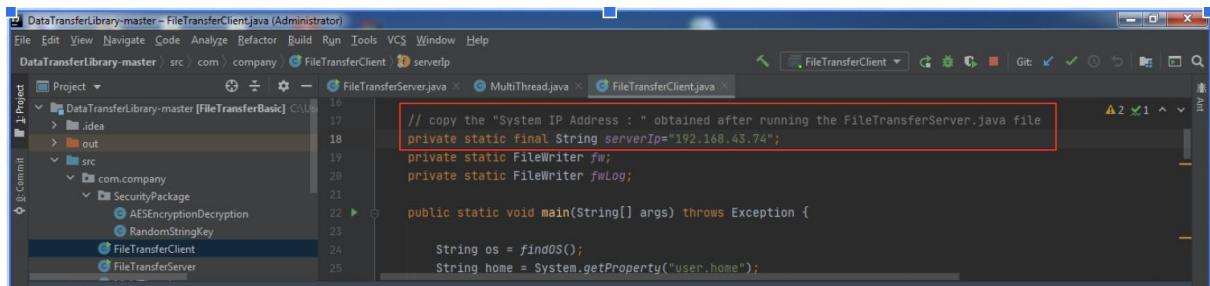
```
Run: FileTransferClient
/Library/Java/JavaVirtualMachines/amazon-corretto-11.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=62666:/Applicat
CLIENT CONFIGURATION FILE CREATED
CLIENT LOG FILE CREATED
You want to Download or Upload content ?
Enter [D/U] :
Select a file from the server
Server not Connected !
Process finished with exit code 0
```

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



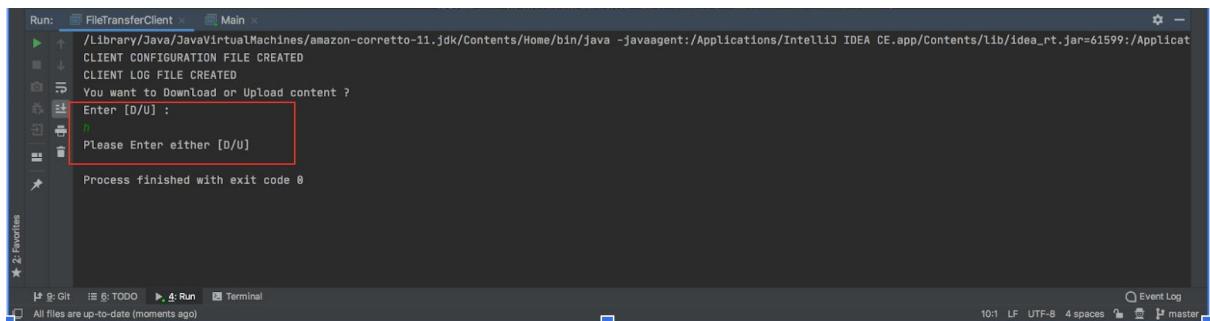
```
Run: Main
/Library/Java/JavaVirtualMachines/amazon-corretto-11.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=56297:/Applicat
SERVER CONFIGURATION FILE CREATED
SERVER LOG FILE CREATED
System IP Address : 192.168.43.74
```

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



```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
DataTransferLibrary-master / src / com / company / FileTransferClient / servedp
Project DataTransferLibrary-master [FileTransferBasic]
src
  com.company
    SecurityPackage
      AESEncryptionDecryption
      RandomStringKey
      FileTransferClient
      FileTransferServer
FileTransferClient.java MultiThread.java FileTransferClient.java
16
17
18
19
20
21
22
23
24
25
// copy the "System IP Address : " obtained after running the FileTransferServer.java file
private static final String serverIp="192.168.43.74";
private static FileWriter fw;
private static FileWriter fwLog;
public static void main(String[] args) throws Exception {
    String os = findOS();
    String home = System.getProperty("user.home");
```

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



```
Run: FileTransferClient
/Library/Java/JavaVirtualMachines/amazon-corretto-11.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=61599:/Applicat
CLIENT CONFIGURATION FILE CREATED
CLIENT LOG FILE CREATED
You want to Download or Upload content ?
Enter [D/U] :
b
Please Enter either [D/U]
Process finished with exit code 0
```

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

```

Run: FileTransferClient x Main x
/Library/Java/JavaVirtualMachines/amazon-corretto-11.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=61342:/Applicat
CLIENT CONFIGURATION FILE CREATED
CLIENT LOG FILE CREATED
You want to Download or Upload content ?
Enter [D/U] :
U
If you want to upload file to the server :
Run FileTransferServer.java on Client side "sender" and FileTransferClient.java on server side "receiver"
Because here server can only send & client can only download file
Thanks !

Process finished with exit code 0

```

Event Log
All files are up-to-date (moments ago)

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

```

Run: FileTransferClient x FileTransferClient x Main x
/Library/Java/JavaVirtualMachines/amazon-corretto-11.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=61159:/Applicat
CLIENT CONFIGURATION FILE CREATED
CLIENT LOG FILE CREATED
You want to Download or Upload content ?
Enter [D/U] :
D
>Select a file from the server"

```

Event Log
All files are up-to-date (moments ago)

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

```

Run: FileTransferServer x
C:\Program Files\Amazon Corretto\jdk11.0.8_10\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.2\lib\idea_rt.jar=5:
SERVER CONFIGURATION FILE CREATED
SERVER LOG FILE CREATED
System IP Address : 192.168.43.25
Client Connected

```

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

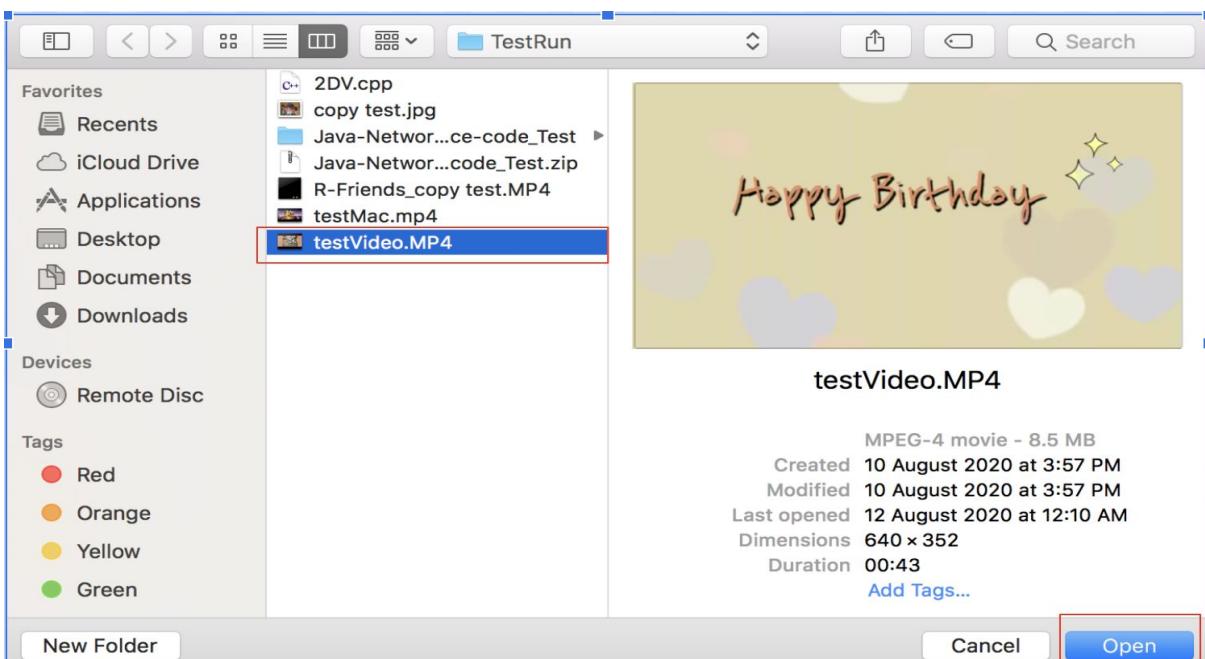


Figure : on server machine the user will select a file to transfer to the client and press open

```

Run: Main FileTransferClient
Client Connected
objc[1585]: Class FIFinderSyncExtensionHost is implemented in both /System/Library/PrivateFrameworks/FinderKit.framework/Versions/A/FinderKit (0x7fff895bacd0) and /System/Library/PrivateFrameworks/FinderSync.framework/Versions/A/FinderSync (0x7fff895bae00)
testVideo.MP4 chosen.
/Users/aroras/Desktop/TestRun/testVideo.MP4
Sending file ... 5.908872% complete!
Sending file ... 11.817744% complete!
Sending file ... 17.726616% complete!
Sending file ... 23.655489% complete!
Sending file ... 29.54436% complete!
Sending file ... 35.45323% complete!
Sending file ... 41.362183% complete!
Sending file ... 47.270977% complete!
Sending file ... 53.179848% complete!
Sending file ... 59.08872% complete!

```

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

```

Run: Main FileTransferClient
Sending file ... 41.502183% complete!
Sending file ... 47.270977% complete!
Sending file ... 53.179848% complete!
Sending file ... 59.08872% complete!
Sending file ... 64.99759% complete!
Sending file ... 70.98664% complete!
Sending file ... 76.81534% complete!
Sending file ... 82.724205% complete!
Sending file ... 88.63388% complete!
Sending file ... 94.541954% complete!
Sending file ... 100% complete!
File sent successfully!

```

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

```

Run: Main FileTransferClient
/Users/aroras/Desktop/TestRun/testVideo.MP4
Read 131,872 bytes, speed: 32 MB/s
Downloading file ... 1.5409753% complete!
Read 262,144 bytes, speed: 29 MB/s
Downloading file ... 3.0979507% complete!
Read 393,216 bytes, speed: 39 MB/s
Downloading file ... 4.646926% complete!
Read 524,288 bytes, speed: 47 MB/s
Downloading file ... 6.1959814% complete!
Read 655,360 bytes, speed: 54 MB/s
Downloading file ... 7.744877% complete!
Read 786,432 bytes, speed: 60 MB/s
Downloading file ... 9.293852% complete!
Read 917,504 bytes, speed: 65 MB/s

```

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

```

Run: Main FileTransferClient
Read 8,257,356 bytes, speed: 175 MB/s
Downloading file ... 97.58545% complete!
Read 8,388,608 bytes, speed: 178 MB/s
Downloading file ... 99.13442% complete!
Read 8,461,852 bytes, speed: 176 MB/s
Downloading file ... 100% complete!

Average speed achieved in the whole transaction is : 128 MB/s

testVideo.MP4 : File saved successfully!

Refer GoodProgram folder in your Downloads folder

```

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

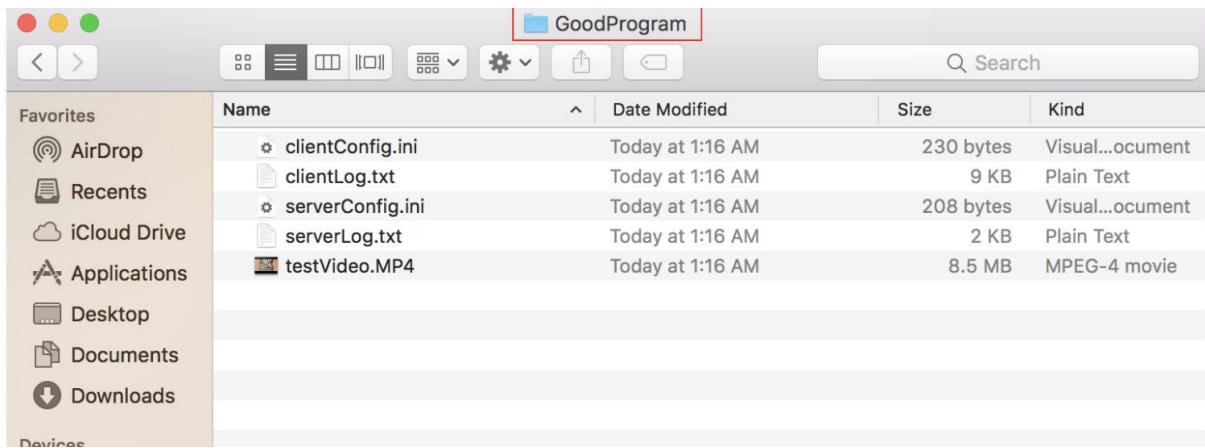


Figure : If client & server run on the same machine then GoodProgram directory will look as above



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

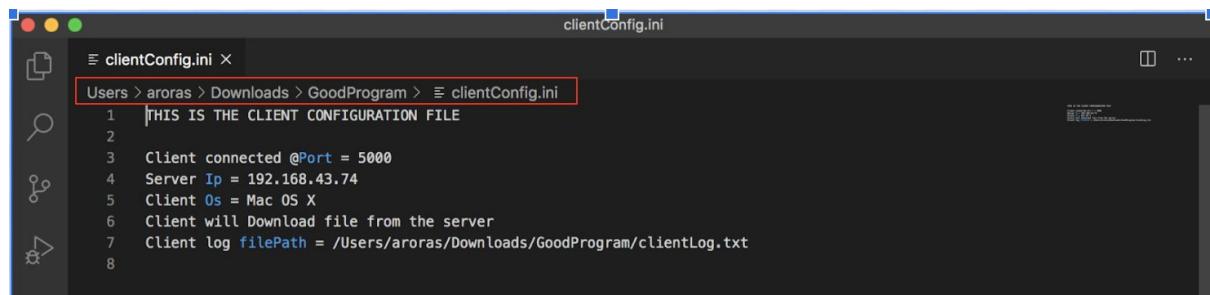


Figure : clientConfiguration file will store above client information + path of clientLog file

```

THIS IS THE SERVER LOG FILE
[File selected]
FilePath of the file to be send from server = /Users/aroras/Desktop/TestRun/testVideo.MP4
of length = 8461852

[Encryption]
Secret key of size 20 = zo7A70LzLTv2D1ky0Dv6
File + Secret key = x0wgm20PUiYzuG0JlpymRi92dZgd77sVPJinvd44ysiucN10TyrxGl/Ayax6U7iI

[Send details to client]
Encrypted filename + Os of server + fileLength + secret key for encryption is send to the
client

[Read selected file and start sending it to client]
Starting Reading from the beginning of the file
Bytes read = 500000
write to client
Sending file ... 5.908872% complete!

Bytes read = 1000000
write to client
Sending file ... 11.817744% complete!

Bytes read = 1500000
write to client
Sending file ... 17.726616% complete!

Bytes read = 2000000
write to client

```

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.)

```
serverLog.txt
write to client
Sending file ... 64.99759% complete!
Bytes read = 6000000
write to client
Sending file ... 70.90646% complete!
Bytes read = 6500000
write to client
Sending file ... 76.81534% complete!
Bytes read = 7000000
write to client
Sending file ... 82.724205% complete!
Bytes read = 7500000
write to client
Sending file ... 88.63308% complete!
Bytes read = 8000000
write to client
Sending file ... 94.541954% complete!
Bytes read = 8461852
write to client
Sending file ... 100.0% complete!
[Complete]
File sent successfully!
```

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

```
clientLog.txt
THIS IS THE CLIENT LOG FILE

[Receive details from server]
encrypted file = x0wgm20PUiyZuG0JlpymRi92dZgd77sVPJinvd44ysiucN10TyrxGl/Ayqx6U7iT
Os of server = Mac OS X
fileLength = 8461852

[Decryption]
secret key = zo7A70LzLTv2D1ky0Dv6
File - Secret key = /Users/aroras/Desktop/TestRun/testVideo.MP4

[Download file from the server]
Directory for download = /Users/aroras/Downloads/GoodProgram
File to be download = testVideo.MP4
Create filepath to download file = /Users/aroras/Downloads/GoodProgram/testVideo.MP4

[Start reading content from client and write to download file]
Starting receiving content of the file from the server
Host Available
Bytes read = 131072
write to destination
Speed of transaction = 26
Downloading file ... 1.5489753% complete!

Host Available
Bytes read = 262144
write to destination
Speed of transaction = 26
Downloading file ... 3.0979507% complete!
```

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.)

```

clientLog.txt
write to destination
Speed of transaction = 162
Downloading file ... 96.03647% complete!

Host Available
Bytes read = 8257536
write to destination
Speed of transaction = 165
Downloading file ... 97.58545% complete!

Host Available
Bytes read = 8388608
write to destination
Speed of transaction = 164
Downloading file ... 99.13442% complete!

Host Available
Bytes read = 8461852
write to destination
Speed of transaction = 165
Downloading file ... 100.0% complete!

[Average speed]
Average speed achieved in the whole transaction is : 122 MB/s

[Complete]
File saved successfully!
Refer GoodProgram folder in your Downloads folder to find the downloaded file

```

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

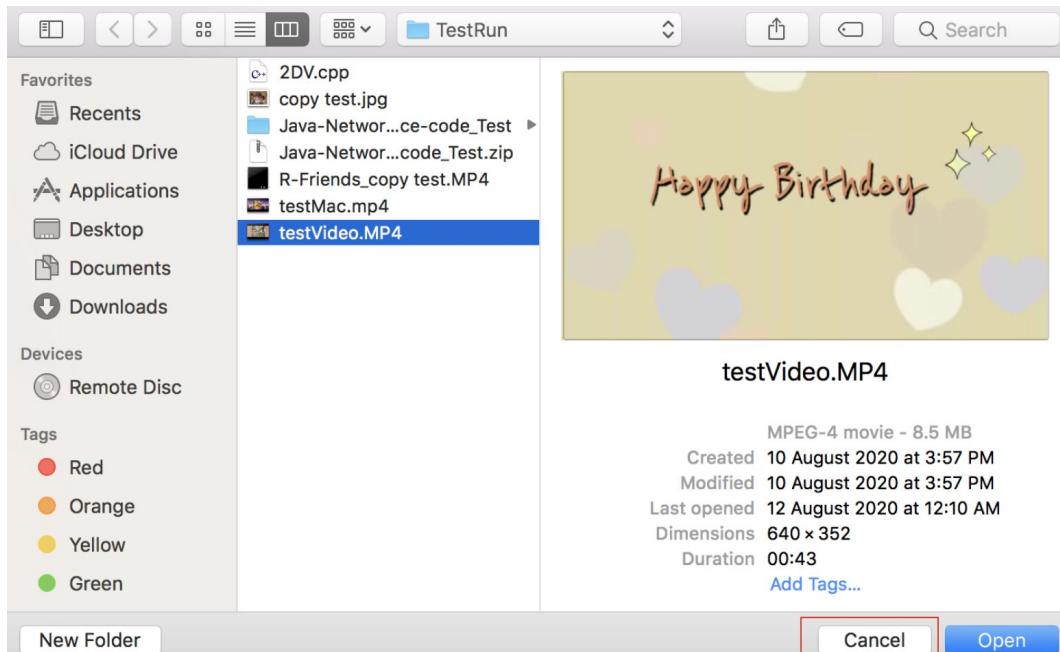


Figure : user may chose not to select any file from the server (i.e no file will be transferred to the client) and press cancel then “No file selected is send to the client”

```

Run: FileTransferClient x Main x
/Library/Java/JavaVirtualMachines/amazon-corretto-11.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=61338:/Applicat
CLIENT CONFIGURATION FILE CREATED
CLIENT LOG FILE CREATED
You want to Download or Upload content ?
Enter [D/U] :
d
>Select a file from the server
No file was selected !

Process finished with exit code 0

```

All files are up-to-date (moments ago)

Event Log
12:11 LF UTF-8 4 spaces 9e master

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

```

THIS IS THE SERVER LOG FILE
[No file selected]
tell no file selected to client

```

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

```

THIS IS THE CLIENT LOG FILE
[Receive details from server]
No file was selected

```

Figure : clientLog will contain “No file was selected”

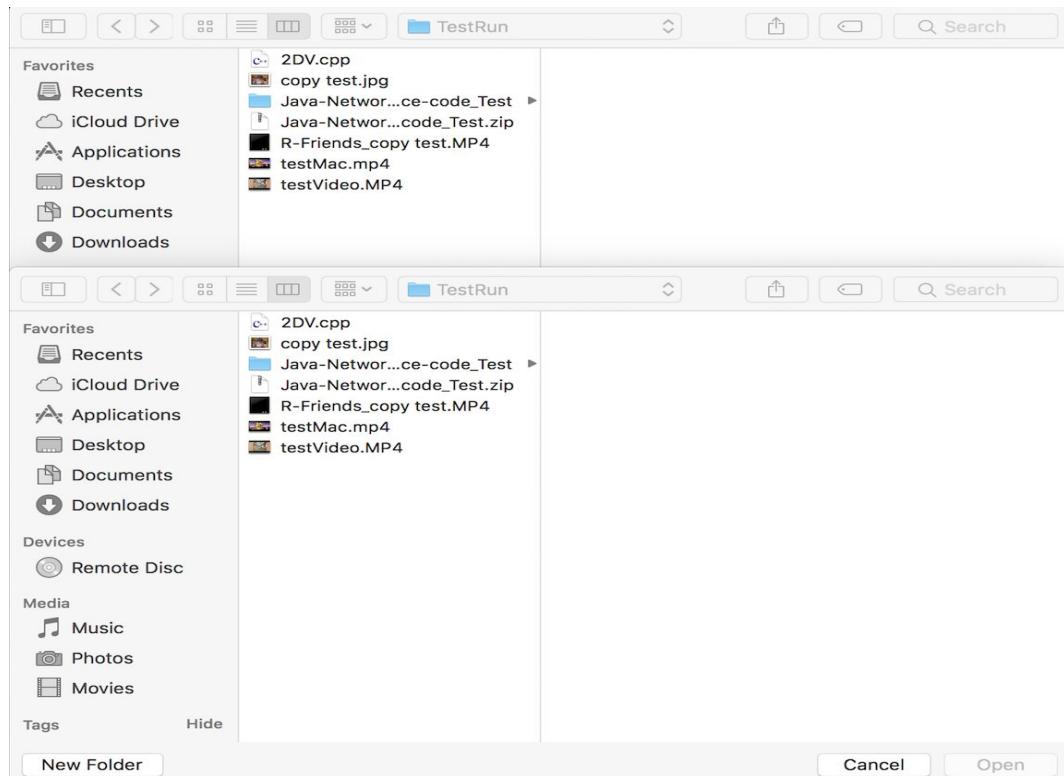


Figure : Multiple clients can access the server (and select different files) at the same time

```

Run: FileTransferClient x Main x
/Library/Java/JavaVirtualMachines/amazon-corretto-11.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=61162:/Applicat
 SERVER CONFIGURATION FILE CREATED
System IP Address : 192.168.43.74
Client Connected
Client Connected
objc[162]: Class FIFinderSyncExtensionHost is implemented in both /System/Library/PrivateFrameworks/FinderKit.framework/Versions/A/FinderKit (0x7fff895bacd0) and /Sy
testVideo.MP4 chosen.
/Users/aroras/Desktop/TestRun/testVideo.MP4
Sending file ... 5.908872% complete!
Sending file ... 11.817744% complete!
Sending file ... 17.726616% complete!
Sending file ... 23.635489% complete!
Sending file ... 29.54436% complete!
Sending file ... 35.46373% complete!

```

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

```

Run: FileTransferClient x FiletransferClient x Main x
f = new File( pathname: home + "\\\Downloads\\GoodProgram\\serverLog.txt");
read 15,518,424 bytes, speed: 284 MB/s
Downloading file ... 98.76575% complete!
Read 13,649,496 bytes, speed: 283 MB/s
Downloading file ... 99.72336% complete!
Read 13,687,361 bytes, speed: 284 MB/s
Downloading file ... 100.0% complete!

Average speed achieved in the whole transaction is : 155 MB/s
R-Friends_copy test.MP4 : File saved successfully!
Refer GoodProgram folder in your Downloads folder
Process finished with exit code 0

```

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

```

Run: Main x FileTransferClient x
Sending file ... 39.62% complete!
Sending file ... 39.678978% complete!
Sending file ... 39.72998% complete!
Sending file ... 39.780983% complete!
Sending file ... 39.83198% complete!
Sending file ... 39.882984% complete!
Sending file ... 39.933983% complete!
Sending file ... 39.984985% complete!
Sending file ... 40.03599% complete!
Sending file ... 40.08699% complete!
Sending file ... 40.137993% complete!
Sending file ... 40.188995% complete!
Client closed
Only 40.239994% file send was complete!

```

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

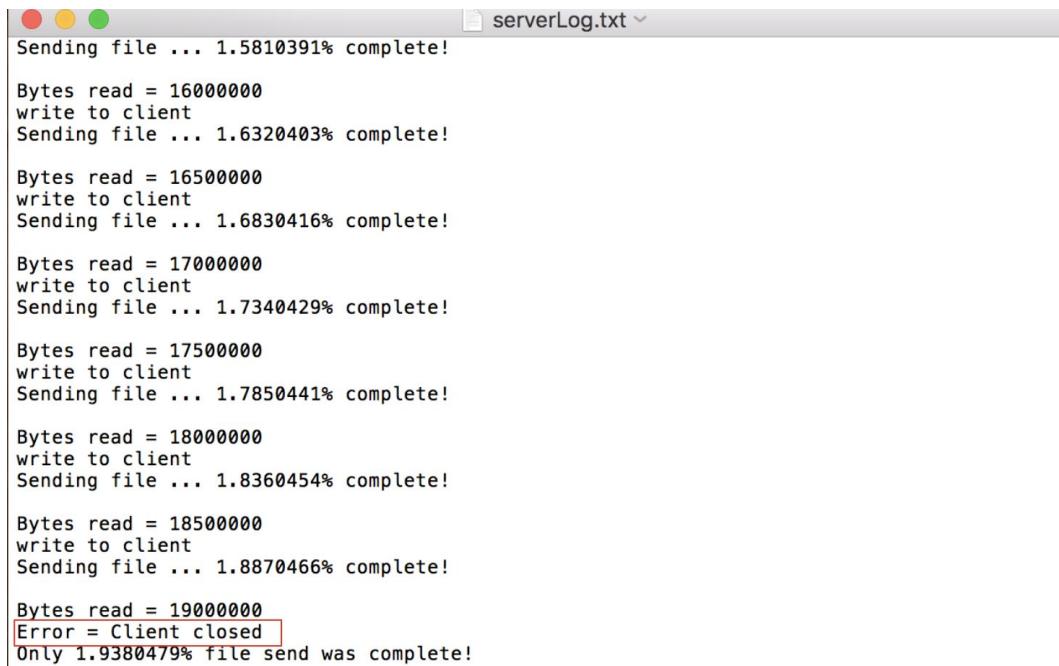
```

Run: Main x FileTransferClient x
Read 229,662,488 bytes, speed: 16 MB/s
Downloading file ... 23.426151% complete!
Read 229,793,552 bytes, speed: 16 MB/s
Downloading file ... 23.43952% complete!
Read 229,924,624 bytes, speed: 16 MB/s
Downloading file ... 23.45289% complete!
Read 230,026,468 bytes, speed: 16 MB/s
Downloading file ... 23.46328% complete!

Average speed achieved in the whole transaction is : 43 MB/s
Oops Server Disconnected , Try after some Time !!
Process finished with exit code 0

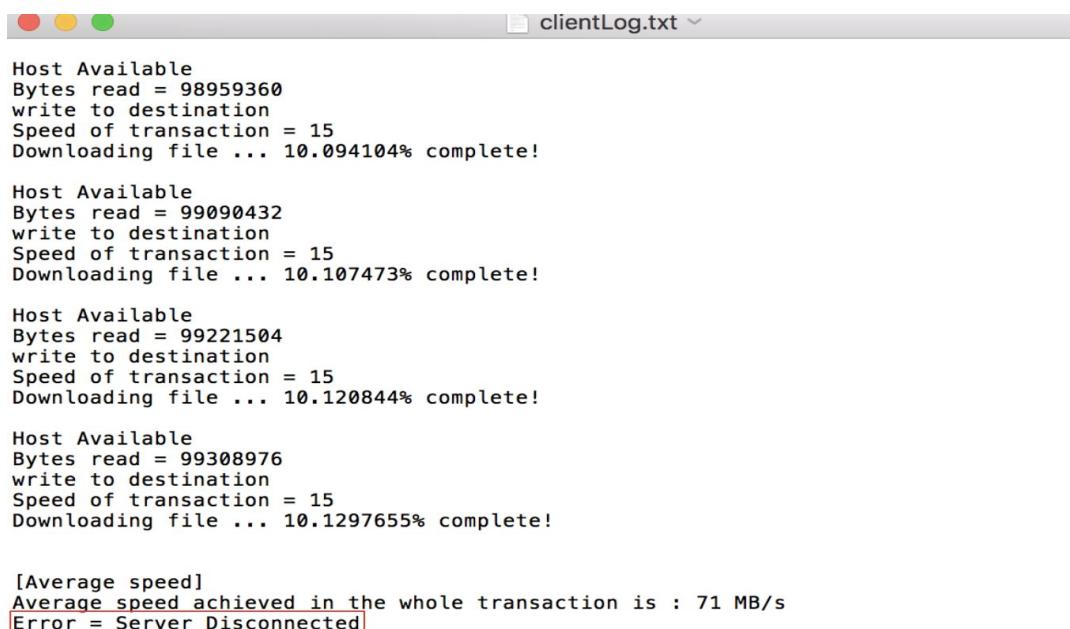
```

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



```
Sending file ... 1.5810391% complete!
Bytes read = 16000000
write to client
Sending file ... 1.6320403% complete!
Bytes read = 16500000
write to client
Sending file ... 1.6830416% complete!
Bytes read = 17000000
write to client
Sending file ... 1.7340429% complete!
Bytes read = 17500000
write to client
Sending file ... 1.7850441% complete!
Bytes read = 18000000
write to client
Sending file ... 1.8360454% complete!
Bytes read = 18500000
write to client
Sending file ... 1.8870466% complete!
Bytes read = 19000000
Error = Client closed
Only 1.9380479% file send was complete!
```

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



```
Host Available
Bytes read = 98959360
write to destination
Speed of transaction = 15
Downloading file ... 10.094104% complete!

Host Available
Bytes read = 99090432
write to destination
Speed of transaction = 15
Downloading file ... 10.107473% complete!

Host Available
Bytes read = 99221504
write to destination
Speed of transaction = 15
Downloading file ... 10.120844% complete!

Host Available
Bytes read = 99308976
write to destination
Speed of transaction = 15
Downloading file ... 10.1297655% complete!

[Average speed]
Average speed achieved in the whole transaction is : 71 MB/s
Error = Server Disconnected
```

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

```

Run: FileTransferClient
/Library/Java/JavaVirtualMachines/amazon-corretto-11.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=62675:/Appli
CLIENT CONFIGURATION FILE CREATED
CLIENT LOG FILE CREATED
You want to Download or Upload content ?
Enter [D/U] :
g
>Select a file from the server"
C:\Users\hp\Desktop\test.mp4
Read 131,072 bytes, speed: 0 MB/s
Downloading file ... 0.013369674% complete!
Read 262,144 bytes, speed: 0 MB/s
Downloading file ... 0.026739348% complete!
Read 393,216 bytes, speed: 0 MB/s
Downloading file ... 0.040109023% complete!
Read 524,288 bytes, speed: 0 MB/s
Downloading file ... 0.053478695% complete!
Read 655,360 bytes, speed: 0 MB/s

```

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

```

Run: FileTransferClient
Downloading file ... 2.580129% complete!
Read 24,641,536 bytes, speed: 0 MB/s
Downloading file ... 2.5134988% complete!
Read 24,772,688 bytes, speed: 0 MB/s
Downloading file ... 2.5268683% complete!
Read 24,903,680 bytes, speed: 0 MB/s
Downloading file ... 2.5402381% complete!
Read 25,034,752 bytes, speed: 0 MB/s
Downloading file ... 2.5536877% complete!
Read 25,165,824 bytes, speed: 0 MB/s
Downloading file ... 2.5669775% complete!
Read 25,296,896 bytes, speed: 0 MB/s
Downloading file ... 2.580347% complete!
Read 25,427,968 bytes, speed: 0 MB/s
Downloading file ... 2.5937169% complete!
Server connection Interrupted !
Waiting for Connection

```

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)

```

Run: FileTransferClient
Downloading file ... 2.5536877% complete!
Read 25,165,824 bytes, speed: 0 MB/s
Downloading file ... 2.5669775% complete!
Read 25,296,896 bytes, speed: 0 MB/s
Downloading file ... 2.580347% complete!
Read 25,427,968 bytes, speed: 0 MB/s
Downloading file ... 2.5937169% complete!
Server connection Interrupted !
Waiting for Connection
File can't be retrieved
2.5937169% of the file was downloaded though !

Refer GoodProgram folder in your Downloads folder
Oops Client Disconnected , Try after some Time !!

```

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 until now

```

clientLog.txt ▾
Speed of transaction = 0
Downloading file ... 2.5536077% complete!

Host Available
Bytes read = 25165824
write to destination
Speed of transaction = 0
Downloading file ... 2.5669775% complete!

Host Available
Bytes read = 25296896
write to destination
Speed of transaction = 0
Downloading file ... 2.580347% complete!

Host Available
Bytes read = 25427968
write to destination
Speed of transaction = 0
Downloading file ... 2.5937169% complete!

Host Unavailable
Server connection Interrupted
Waiting for Connection
Let the thread sleep for 8 seconds
Thread is up check for connection
Host still unavailable , file can't be retrieved
2.5937169% of the file was downloaded though
Refer GoodProgram folder in your Downloads folder to find file
Error = Client Disconnected

```

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

```

Run: FileTransferClient ×
↓ Downloading file ... 1.0054174% complete!
Read 18,615,120 bytes, speed: 0 MB/s
↓ Downloading file ... 1.898789% complete!
Read 18,746,192 bytes, speed: 0 MB/s
↓ Downloading file ... 1.9121587% complete!
Read 18,877,264 bytes, speed: 0 MB/s
↓ Downloading file ... 1.9255284% complete!
Read 19,008,336 bytes, speed: 0 MB/s
↓ Downloading file ... 1.9388981% complete!
Read 19,139,408 bytes, speed: 0 MB/s
↓ Downloading file ... 1.9522678% complete!
Read 19,270,480 bytes, speed: 0 MB/s
↓ Downloading file ... 1.9656374% complete!
Server connection Interrupted !
Waiting for Connection
File can be retrieved
Read 19,401,552 bytes, speed: 0 MB/s
↓ Downloading file ... 1.9798071% complete!
Read 19,532,624 bytes, speed: 0 MB/s
↓ Downloading file ... 1.9923768% complete!
Read 19,663,696 bytes, speed: 0 MB/s
↓ Downloading file ... 2.0057464% complete!
Read 19,794,768 bytes, speed: 0 MB/s
↓ Downloading file ... 2.0191162% complete!
Read 19,925,840 bytes, speed: 0 MB/s
↓ Downloading file ... 2.0324857% complete!
Read 20,056,912 bytes, speed: 0 MB/s
↓ Downloading file ... 2.0458555% complete!

```

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 upto now and speed achieved

```
clientLog.txt ▾
```

```
Host Available
Bytes read = 19270480
write to destination
Speed of transaction = 0
Downloading file ... 1.9656374% complete!

Host Unavailable
Server connection Interrupted
Waiting for Connection
Let the thread sleep for 8 seconds
Thread is up check for connection
Host available , file can be retrieved
Host Available
Bytes read = 19401552
write to destination
Speed of transaction = 0
Downloading file ... 1.9790071% complete!

Host Available
Bytes read = 19532624
write to destination
Speed of transaction = 0
Downloading file ... 1.9923768% complete!

Host Available
Bytes read = 19663696
write to destination
Speed of transaction = 0
```

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