# A P2P FILE SHARING SYSTEM

## MANUAL

## PROJECT FILES - OVERVIEW

The  project mainly comprises of two interface files - ITFCentralIndexingServer.java and ITFPeer.java - which define the behavioral capabilities each server and peer implementing class should provide.

There is single implementing class for the server - CentralIndexingServer.java - which is responsible for both implementing the server indexing interface as well as providing the server application main which boots up the server for use in the network.

There are, however, *'two'* implementing Peer classes – Peer.java in the *'server_peer'* folder and a Peer.java in the *'client_peer'.* The file in the server_peer/Peer.java, in addition to the interface implementations, contains the peer application main for the peers who will only act as registering peers not initiating any downloads from the network. On the other hand, the client_peer/Peer.java file contains, in addition to the code present in server_peer/Peer.java code which is used to develop a simple test case for a file download. ( All the performance tests were also done by manipulating this application template ).

There are additionally two policy files - server.policy & client.policy - which are used to facilitate the working of the security managers within all applications. There is also a 1MB test text file called '1000kb.txt' for simulating a download.

The communicating protocol is RMI which supports inherent concurrency if required, therefore all the code was written with thread safety in mind, without any explicit multithreading.

## COMPILING THE PROJECT FILES

1) Create in in 3 different directories folder pertaining to each – The Central Indexing Server, A registering peer and a downloading peer.
2) Start by copying the interface files, the server class implementation file, the server.policy file and the server_peer/Peer.java file to your *'server'* folder.

3) In a terminal window, reach your server directory and then type the following commands in the order they are mentioned: -
    a) *javac *.java* – This will compile all the java files inside the folder.
    b) *rmic CentralingIndexingServer*
    c) *rmic Peer*

    These will create the stub class which will serve as a proxy for their respective class objects in JVMs other than their own. As you must have guessed it is these stub classes only through which remote method invocations are expressed.
4) Repeat steps (2) & (3) for the registering peer folder as well with the exception of using client.policy file instead of server.policy. Also copy the testfile.txt to this folder.
5) Repeat steps (2) & (3) for the downloading peer with the exception that instead of the file server_peer/Peer.java use the file client_peer/Peer.java for the entire process and instead of server.policy use client.policy.

    Note that the stub classes for both the peers are exactly the same, as the stub classes only implements the methods defined in the corresponding interfaces. Instead of individual compilation certain files could also be copied after the first compilation.

## RUNNING THE PROJECT FILES

1) Open a terminal window and cd to the directory folder of the server.
2) Fire up the rmi registry by typing : -
    a. *rmiregistry &* - Linux
    b. *start rmiregistry - Windows*
3) Boot up the server by typing : -
    a. *java -Djava.security.policy=server.policy CentralIndexingServer*
    b. A bound confirmation message should now display on screen.

1) Open another terminal and cd to the folder of the registering client.
2) Boot up the registering peer by typing : -
    a. *java -Djava.security.policy=client.policy Peer "Peer_Name"*
    b. Where "Peer_Name" is a user input name it wants to give to the peer.

c. Note that the name argument is not optional.

d. A registration confirmation message should now be displayed.

1) Open another terminal and cd to the folder of the downloading client
2) Boot up the downloading peer by typing : -
    a. *java -Djava.security.policy=client.policy Peer “Peer_Name” “1000kb.txt”*
    b. Where “Peer_Name” is a user input name it wants to give to the peer & the second argument implies the full name of the file ( including the extension txt ) which the client requires downloading, here “testfile.txt”
    c. Note that the peer_name & and the file_name arguments are not optional.
    d. A download confirmation should be displayed.