

## DATA STRUCTURE:

My System Implements my PING and PONG in the form of each Node sending a Unique ID back and forth between each other.

An ID will look like this: **{TCPPORTNUMBER}-{Number of files available}-Localhost**

An example of that will look like this: 6341-4-localhost

6341 is the port number and 4 is the number of files available to that node, localhost is where everything is hosted. The port number is randomly generated at the startup of the Node. *TCPPORTNUMBER* and *Number of files Available* are Integers.

A Query in my system will look like this:

**Query-{TCPPORTNUMBER of source of Query}-{Number of files available}-localhost-{filename.type}-{ID of Request}-{TTL}-**

An example of that will look like this: Query-6341-4-localhost-file.txt-7106-5-

The file we are requesting is file.txt, the requesting node is the one with port number 6341.

Once a Query is found the Requesting Node gets back a string that that allows to get ready for the download of the file.

Response back to requesting Node: **file:{filename.type}found at {ID of node that has file}.**

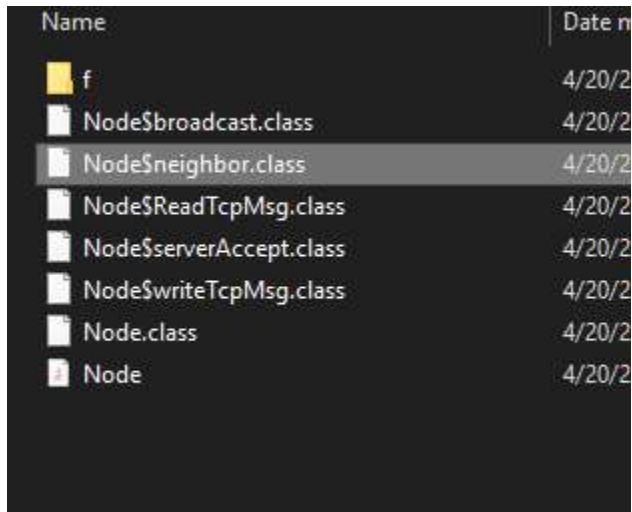
Example: file:file.txtfound at 4000-6-localhost

**File Download LIMIT: 1MB**

**NODE Limit: Testing has shown me that it works with 5 Node and more but it ultimately will depend on memory limit.**

## HOW TO USE:

The Way that everything works, is that each node has a folder called f that would be the shared folder with files to share



Name	Date n
f	4/20/2020
Node\$broadcast.class	4/20/2020
Node\$neighbor.class	4/20/2020
Node\$ReadTcpMsg.class	4/20/2020
Node\$serverAccept.class	4/20/2020
Node\$writeTcpMsg.class	4/20/2020
Node.class	4/20/2020
Node	4/20/2020

Figure 1

Figure 1 shows an example of a node directory, we you have the program code and a folder called f, anything in that f folder is subject to sharing, that folder must be made before the start of the node. Once that is made and things that we want to possibly share are in f. we have start the seed Node.

**-The first Node is the SEED Node, starting a seed Node is done simply by running `java Node`**

If another node wants to start it has to be **`java NODE 4000 {filename.type to request}`**

Let say we want to launch a node to request file.txt from whomever has it that Node has to be launched for the terminal this way: **`java Node 4000 file.txt`**

## MODULE designs:

Each Node's ID contains the portNumber for it's own TCP connection, this is how every node knows and connects to other nodes, this way even if the seeding node stops, once all the wanted nodes are started the other nodes will still be able to communicate with their neighbors but no new nodes will be able to join the network unless we know one of the running Nodes Port. Also the Nodes will be able to maintain and choose another Neighbor by just using the other nodes who are connected to its own ID.

Main Method: the main method checks if user is starting a Node or a seed Node. If it is not a seed node then we connect to seed node and start our own network, if it is a seed node then a network is started with the seed port number.

I have two methods to handle this

```
ConnecttoNewtwork(port);
```

```
StartNetwork(port);
```

ConnectNetwork() joins an existing TCP Network and takes in a port number, and StartNetwork() starts a TCP network.

## Threads:

We have 4 threads ReadTcpMsg(), broadcast(), writeTcpMsg() and serverAccept()

**serverAccept()** : this thread is launched when StartNetwork() is called, this thread is responsible for accepting new nodes to a specific node, this is in the effort to decentralize the communication.

```
while (true) {  
  
    //Accept new Node  
    //launch ReadTcpMsg() thread to read from incoming Node.  
    //launch writeTcpMsg() thread to send id of NODE accepting  
connection to connecting NODE (Ping and Pong)  
    //launch broadcast() thread to tell Incoming node about  
neighbors  
  
}
```

**WriteTcpMsg():** This thread is used whenever in the code I want to send any message to a socket, Node. It has a handler function `sendmsgTcp()` which opens the TCP output stream and writes to it. This thread can either be run forever or onetime depending on the parameter passed to it.

**ReadTcpMsg():** this thread is launched to read messages from incoming Nodes. with handler function `readDataTcp()`. this thread is important to my implementation because of this function, I handle all the actions to download and send a file here to process query.

```
private static boolean readDataTcp(boolean donotprint, InputStream input, BufferedReader reader)
    throws IOException {

    ///if incoming msg has keyword "found" then we need to download file
    ///readDataTcp returns true to ReadTcoMsg() thread and recvFile()
    /// helper function to receive file is called
    ///file is downloaded to f folder

    ///if incoming msg has "Query" and we have not seen
    ///this request before then we call our helper function
    ///handleQuery() if Node has the file sendFile() is called
    ///file is sent to requesting Node f folder.

    ///if I get a node ID that I have not seen before I connect
    /// Using ConnectNetwork() to that ID.
    ///the ID is basically the TCP port number

    ///if something other then it is a ping and pong, print them
    ///only print that and not file data bytes

}
```

**broadcast():** whenever `serverAccept()` gets a new Node this is called to tell that node about our node neighbors by iterating through our list of known nodes and sending their IDs to the incoming node by the `WriteTcpMsg()` thread. Once the incoming Node receives the other neighbors IDs it will connect to them which was explained above in the `ReadTcpMsg()`