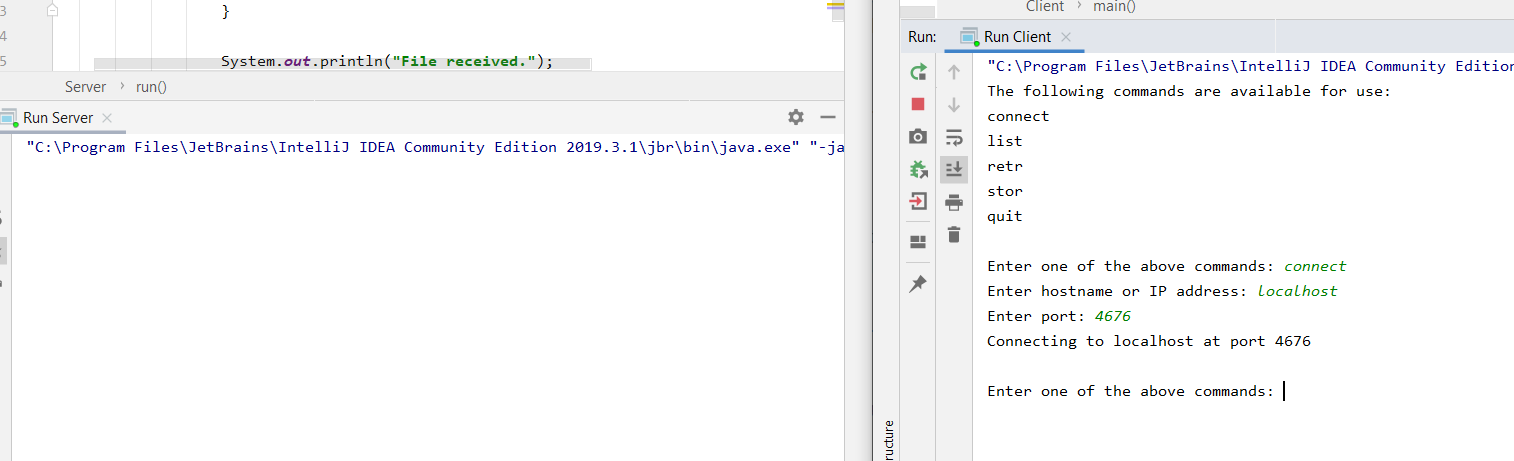
**Demonstration:**

Connect:



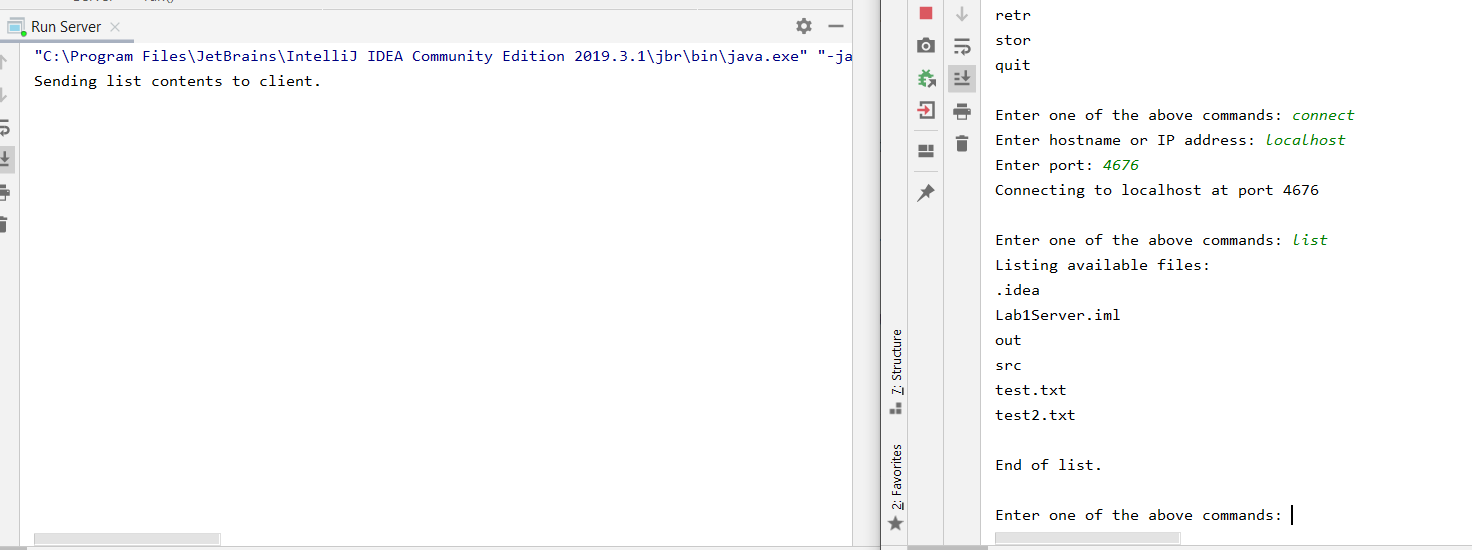
Client Side:

Prompts user to enter hostname and port (4676) it then creates the socket and attempts to connect to the server. Once it has done this, it initializes a new writer.

Server Side:

The server attempts to connect to a client at port 4676. If it connects to one, it sends that socket to a new thread and then waits for another connection at that port.

List:



Client Side:

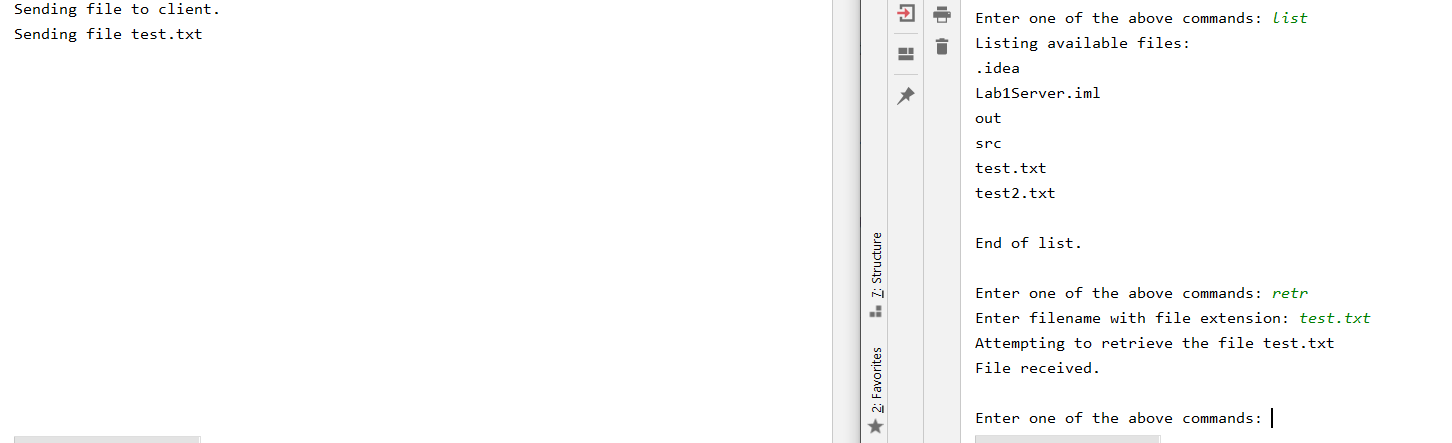
The client sends the word “list” to the server and then attempts to accept a connection at port 1 in order to receive the list. It receives the list by looping through a do while loop until the client receives the text “End of List.” It then closes all port 1 connections.

Server Side:

Server Side:

Upon receiving the word “list”, the server creates a socket at port 1 and attempts to connect to the client. Upon connecting to the client, the server gets all files located in the current folder it is in and sends them to the client. When it sees no more files, it sends the text “End of List.” and closes the sockets.

Retr:



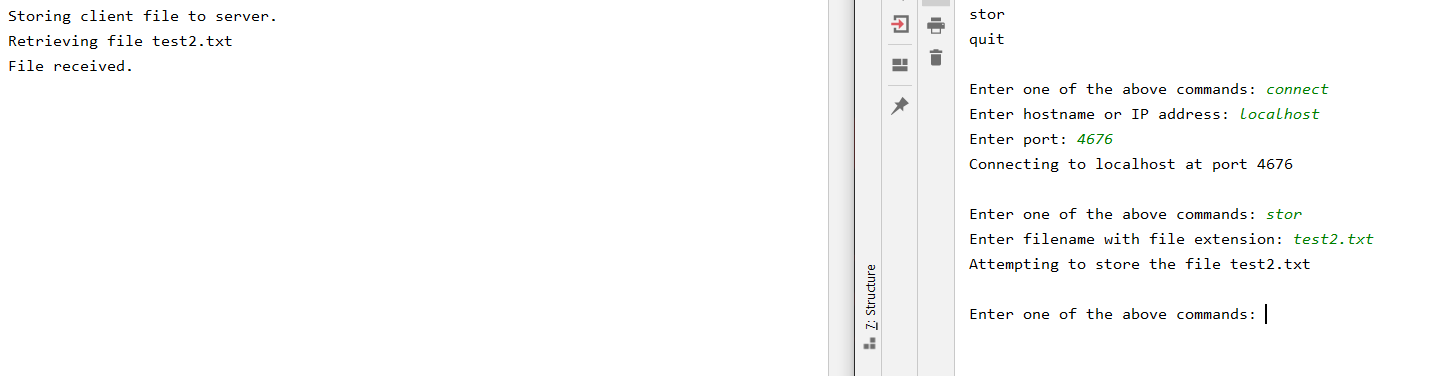
Client Side:

After prompting the user to name an existing file, the client sends the word “list” to the server and then attempts to accept a connection at port 1 in order to receive the file. This time, the file is received in byte format and is then converted back to file format.

Server Side:

Upon receiving the word “retr”, the server creates a socket at port 1 and attempts to connect to the client. Upon connecting to the client, the server takes the file and converts it to byte format and sends it to the client. Upon sending all of the info, the server closes the socket, which causes the server and client in await of the next command.

Stor:



Client Side:

After prompting the user to name an existing file, the client sends the word “stor” to the server and then attempts to create a socket connection at port 1 in order to send the file. It then takes the file, converts it to byte format and sends it across to the server. Once it has done this, it closes the socket so the server and client can await the next command.

Server Side:

Upon receiving the word “stor”, the server attempts to accept a socket at port 1. Upon connecting to the client, the server receives the file in byte format which is then converted back to file format in which it is stored in the server’s directory. The socket is then closed after this happens.

Quit:



Client Side:

Upon receiving this command, the client sends the word “quit” to the server. After doing this, all sockets on the client side are shut down and the client stops running.

Server Side:

Upon receiving the word “quit” the server shuts down the socket connecting to the client and the thread is killed. However, the client is still up and running in await of a new client connection.

The server will not shut down unless it is manually shut down.

**Implementation:**

The overall logic of the client and server is quite simple, though I did struggle a bit on some of the more detailed aspects of it. The client simply creates a socket and attempts to connect to the server using said socket. Once it is connected, this main socket is only used to send and receive commands. Upon receiving a command that is not quit, the server or socket creates a new sub socket which is only open during the execution of the command.

In regard to the server, when the server connects to a new command socket via a client, the server sends this new socket into a thread in which all of the other processes take place. This allows for multiple clients to connect to the same server and issue commands to it at the same time.

Problems:

I will admit I was a little rusty in regard to file I/O, so I had to brush up on that.

My first real issue that I came across came from me not closing the socket upon sending file contents to the server or client, which allowed the program to continue. The issue with this was that I was using the command socket to send the file, which caused major issue when attempting to send another command to the server. In order to fix this, I used sub sockets to send and receive data, and this worked like a charm.

My final issue was implementing multithreading, but the notes and some java tutorials helped me fix this issue.