

## Intro

The object of this project was to build a multi-threaded, text-based, FTP server. The FTP server allows for multiple client connections at the same time. The main features of the application include listing the files on the server's directory, retrieving files from the server directory and storing them on the client directory, and retrieving files from the client, and storing them on the server. The application makes use of 2 TCP connections – one for the control messages (which are the commands) and another for the data transfer. The language-of-choice for this application was Java because it easily handles multi-threading, socket programming, and input and output streams.

The program has two main Java files – FTPClient.java and FTPServer.java. Another file, FileClass.java, is used to contain reusable code between the client and server (determine if a text file exists on a given directory). The FTP client is a basically a main method that creates a control connection, gets user input commands, and creates and closes data connections as necessary. The FTP server continuously accepts client connections. For each client, a client handler thread is created. The **run** method of the thread is where most of the server logic resides.

The following commands are implemented:

**list:** The client sends this command to the server, and receives a list of the files on the server directory.

**retr:** <filename> The client sends this command with a file name to the server, and it reads the file from the server, line-by-line, over a data connection.

**stor:** <filename> The client sends this command with a file name to the server, and the server reads the file, line-by-line, over the data connection.

**quit:** The client sends this to the server, which allows the client to disconnect from the server.

## Program Logic and Implementation

### 1. Client

#### a. User Arguments

When the client program starts, it waits for the user to enter input to connect to the server. The user must enter the **connect** argument, followed by the correct **IP address** and port number the server is listening on. A method was created to handle these arguments. The method checks if the user enters three arguments, and if so, if the argument **connect** is followed by another string (the IP address) and then by an integer greater than 1024. If the user enters invalid command arguments, then the user must continue trying until a valid set of arguments are given.

#### b. Connection to Server

If valid arguments are given, the IP address and port number are extracted from the user input string, and a connection socket is created using the valid IP address of the server and port that it is listening on. Using this socket, a data output stream is set up to send commands to the server.

#### c. Client commands

A loop is created that allows the client to enter commands to send to the server as long as the command is not **quit**. If the user enters **list:**, the message is sent to the

server along with the port number, and the data connection starts listening on port 7173 (7171 + 2). A Buffered Reader is used to read data over the data connection, and it reads each file from the server over the data connection until it reaches and **EOF**. If the message starts with **retr:** or **stor:**, a message is sent over the control connection using the port, message and file name. If the message is **retr:**, it creates a data connection, creates a Buffered Reader, and if the file is found on the server (200 message from server), then each line is read from the server. A Buffered Writer is created, and as the file is read in, it writes to a new file, line-by-line, in the client directory. If the command is **stor:**, a data connection is created, and A File Reader reads the file, if it exists, line-by-line from the client directory. It then writes the message 200, followed by the file, line-by-line to the server over the data connection. If the user enters **quit**, the client sets the loop variable to **false**, and writes that message to the server. The control connection is then closed.

## 2. Server

### a. Server Class

The server application starts by creating a TCP server socket that listens on port 7171. Within a loop, the server continuously waits for clients to connect to the server. Once a connection is accepted, the server creates an instance of a **ClientHandler**, which creates a control connection between the server and client. It creates and runs a new thread that handles this connection given the client socket and client number.

### b. Client Handler

The ClientHandler class creates a connection and sets up the input reader for the control connection between the server and client.

### c. Client Handler Run Method

The main server logic is done in the **run** method. Here, the client message is parsed to get the port number. The client IP address is also obtained. If the client message is **list**, **retr:**, or **stor:**, the server creates a data connection by connecting to the client's IP address and port number. If the client message is **list**, then the server creates a File object based on the server directory, and it iterates through the folder, writing the name of each file to the client via the data connection. If the message is **retr:** **<filename>**, the server checks to see if the file exists in the server directory. If it is found, it reads the file line by line using a File Reader object. The server writes **200**, followed by the file contents, followed by **EOF**, and the File Reader is closed. Otherwise, the error message **550** is sent to the client. If the client message is **stor:** **<filename>**, the data connection must use an input stream to get data from the client. To store the file on the server, the server must first check if the file already exists. If the file does NOT already exist, and the client sends a message of 200 (file found), the server creates a File Writer. It reads the client data from the input stream until it reads an EOF. The File Writer and Input Stream is closed. Once a command is finished, the data connection is closed. If the server receives **quit** from the client, the loop variable is set to **false**, and the server control connection is closed.

## Resolving Issues

This project has successfully implemented all the requirements. It allows for multi-threading with multiple clients. Each of the commands work as well. According to RFC 959, when the commands `retr:` or `stor:` is implemented, the file that is being written (to the client or server directory) overwrites any files with the same name. The project was implemented in small intervals, completing one small task at a time. Some of the code for the multi-threaded application obtained from past labs.

One of the problems that occurred was when the client or server was attempting to write to the data connection. The problem caused the client program to stand still when attempting to write to the control connection. The problem was that the data connection was being written to after the client created a data connection and waited for the server to connect. However, the server was also waiting to receive input for the next command, so there was a bottleneck.

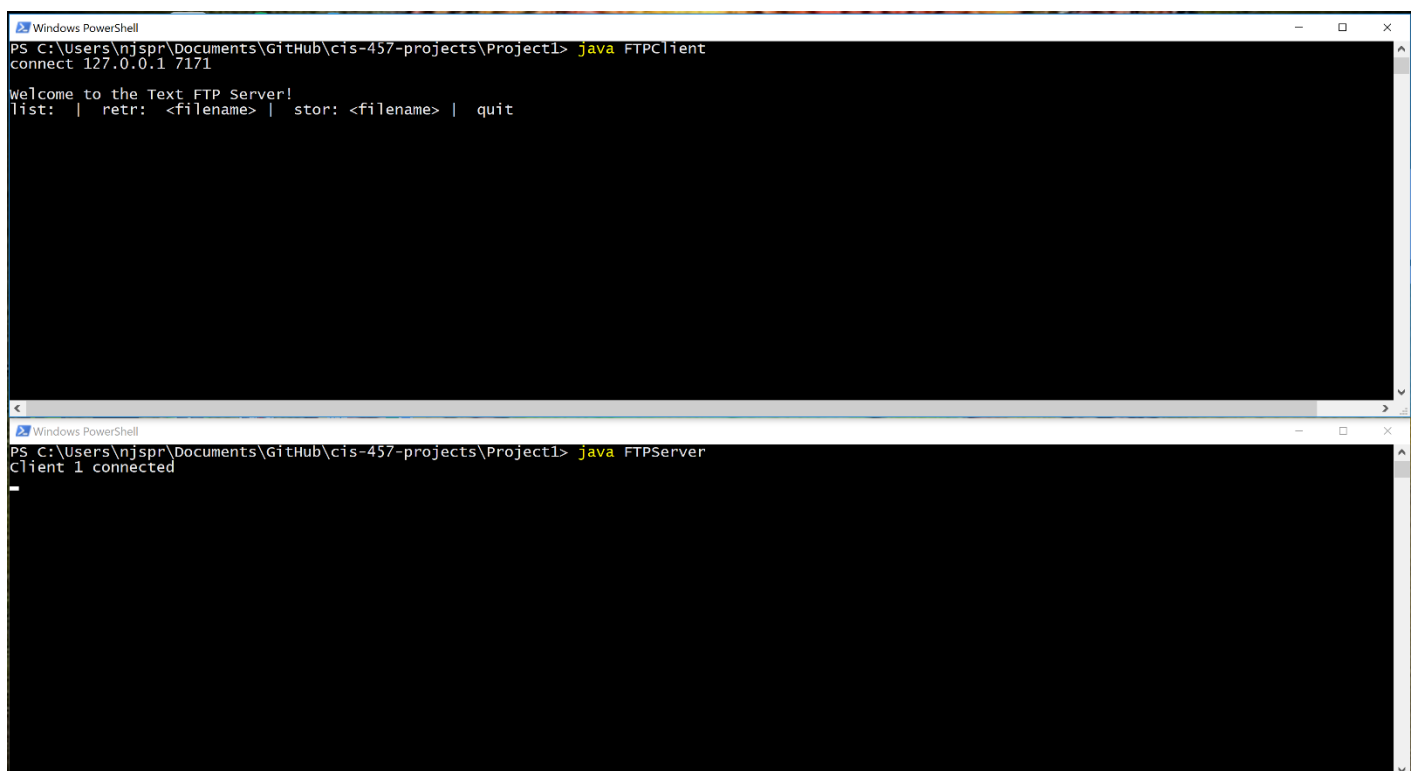
Another problem that occurred was not closing the input, output, or file streams or sockets. This caused several IO Exceptions, and it created problems when trying to re-created data connections in the main client loop.

One final problem was when trying to store files to the server. I forgot to use the codes 200 and 550 to first communicate to the server whether to read the file or not. This created some trouble, until I realized I could use these codes just like I did in the previous **retr:** command.

Overall, the project took roughly fifteen hours to complete, including time to research Java classes and methods via the API, writing the report, and writing and testing the code.

## Screenshots

### Connect to server:

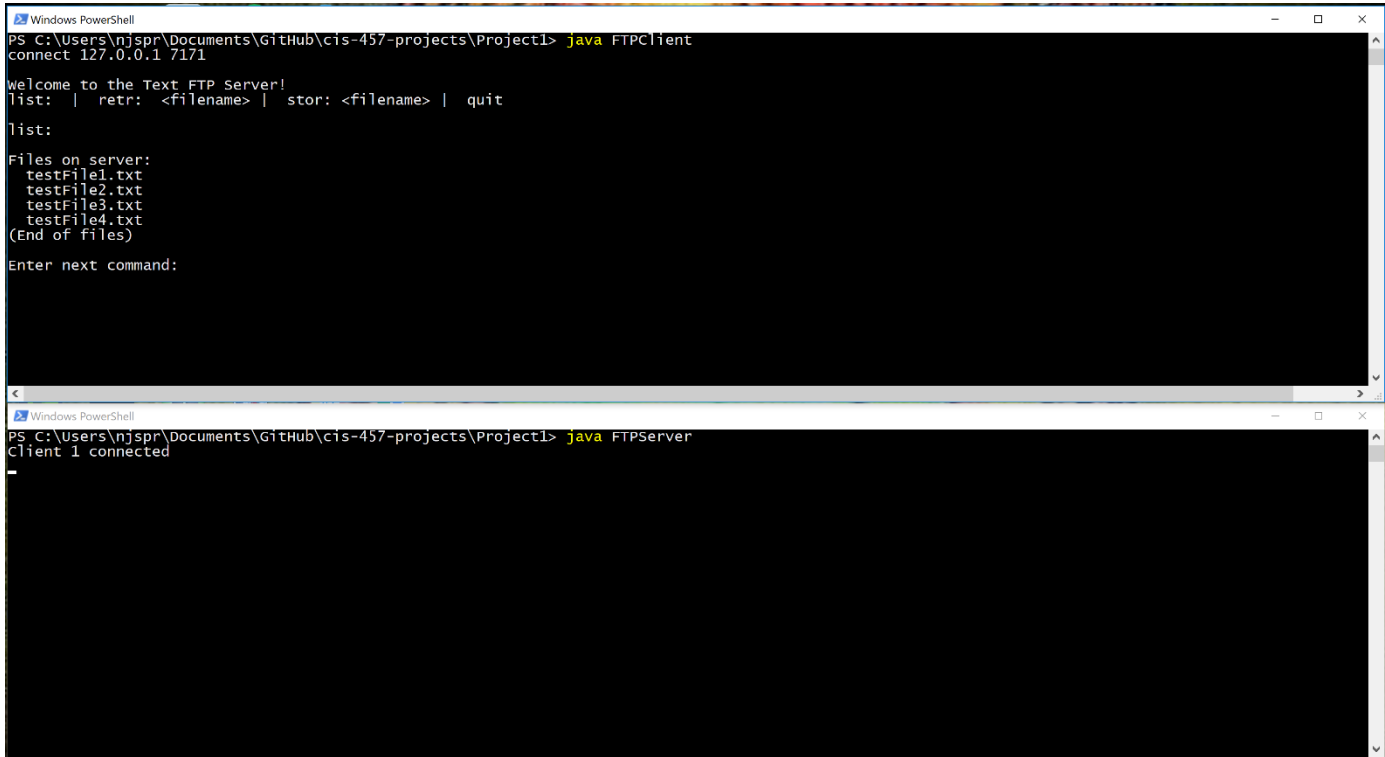


```
Windows PowerShell
PS C:\Users\njspr\Documents\GitHub\cis-457-projects\Project1> java FTPClient
connect 127.0.0.1 7171

welcome to the Text FTP Server!
list: | retr: <filename> | stor: <filename> | quit

Windows PowerShell
PS C:\Users\njspr\Documents\GitHub\cis-457-projects\Project1> java FTPServer
client 1 connected
```

## List files on server:



The image shows two Windows PowerShell terminal windows. The top window is running the FTPClient application, which connects to 127.0.0.1 7171. It displays a welcome message and a list of files on the server: testFile1.txt, testFile2.txt, testFile3.txt, and testFile4.txt. The bottom window is running the FTPServer application, which shows 'Client 1 connected'.

```
PS C:\Users\njspr\Documents\GitHub\cis-457-projects\Project1> java FTPClient
connect 127.0.0.1 7171

Welcome to the Text FTP Server!
list: | retr: <filename> | stor: <filename> | quit

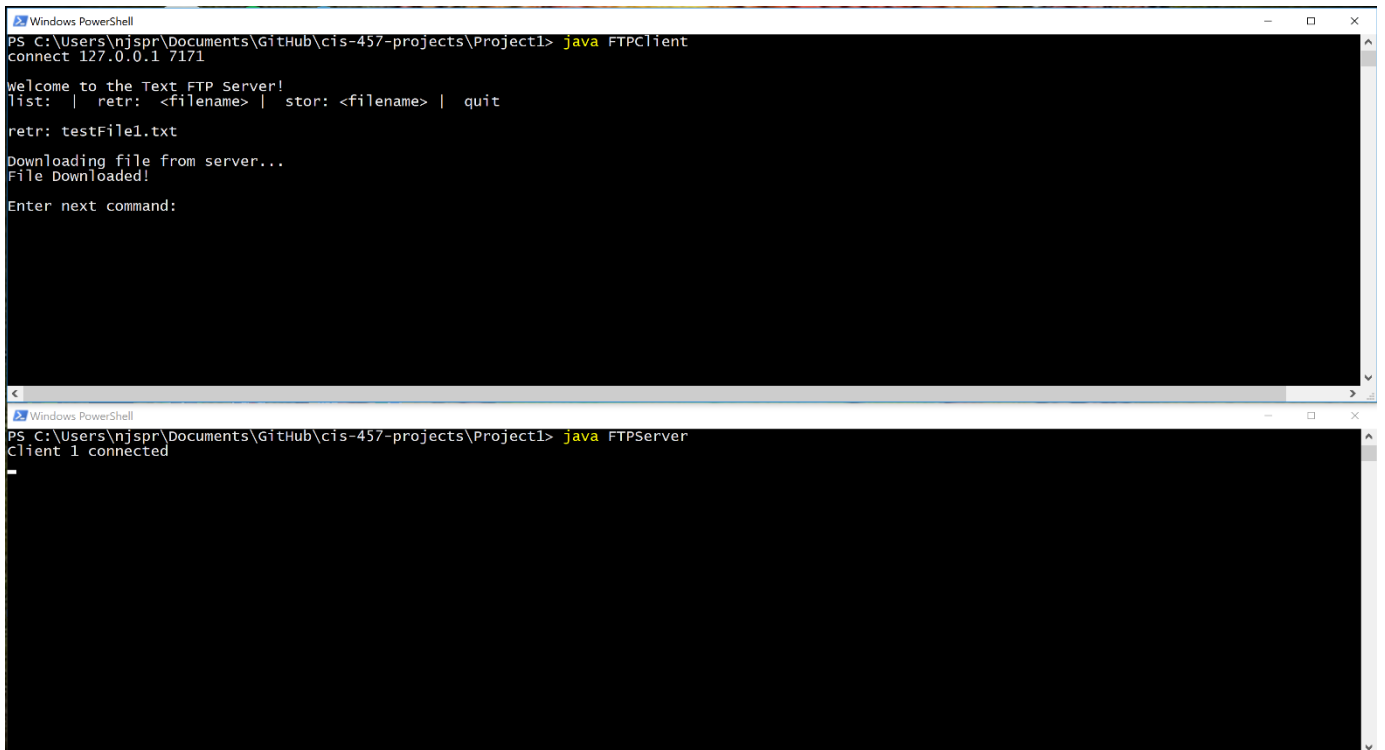
list:

Files on server:
  testFile1.txt
  testFile2.txt
  testFile3.txt
  testFile4.txt
(End of files)

Enter next command:

PS C:\Users\njspr\Documents\GitHub\cis-457-projects\Project1> java FTPServer
Client 1 connected
```

## Retrieve file from server:



The image shows two Windows PowerShell terminal windows. The top window is running the FTPClient application, which connects to 127.0.0.1 7171. It displays a welcome message and a list of files on the server. The user enters the command 'retr: testFile1.txt', and the server responds with 'Downloading file from server...' and 'File Downloaded!'. The bottom window is running the FTPServer application, which shows 'Client 1 connected'.

```
PS C:\Users\njspr\Documents\GitHub\cis-457-projects\Project1> java FTPClient
connect 127.0.0.1 7171

Welcome to the Text FTP Server!
list: | retr: <filename> | stor: <filename> | quit

retr: testFile1.txt

Downloading file from server...
File Downloaded!

Enter next command:

PS C:\Users\njspr\Documents\GitHub\cis-457-projects\Project1> java FTPServer
Client 1 connected
```

## Store file from client to server:

```
Windows PowerShell
PS C:\Users\njspr\Documents\GitHub\cis-457-projects\Project1> java FTPClient
connect 127.0.0.1 7171

Welcome to the Text FTP Server!
list: | retr: <filename> | stor: <filename> | quit

stor: testFile4.txt

Uploading file to server...
File uploaded!

Enter next command:

Windows PowerShell
PS C:\Users\njspr\Documents\GitHub\cis-457-projects\Project1> java FTPServer
Client 1 connected
```

## Client quits:

```
Windows PowerShell
PS C:\Users\njspr\Documents\GitHub\cis-457-projects\Project1> java FTPClient
connect 127.0.0.1 7171

Welcome to the Text FTP Server!
list: | retr: <filename> | stor: <filename> | quit

stor: testFile4.txt

Uploading file to server...
File uploaded!

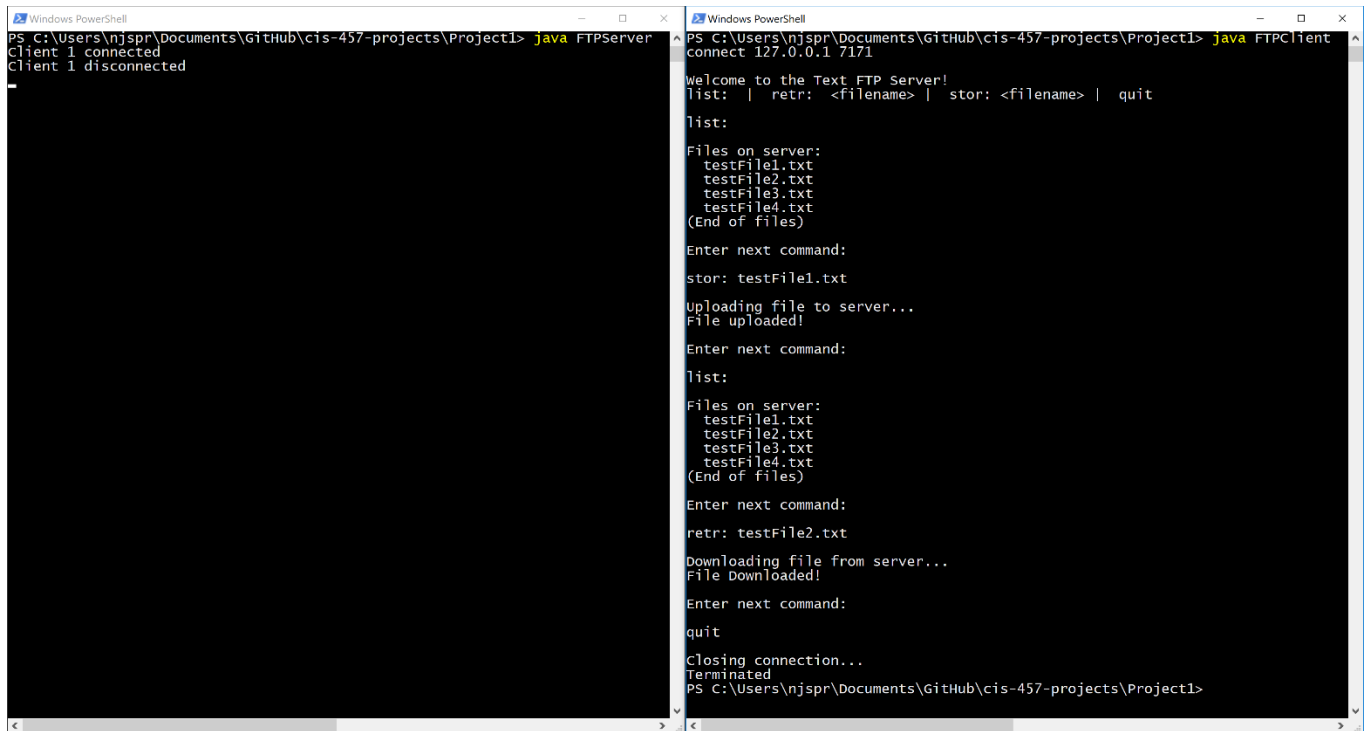
Enter next command:

quit

Closing connection...
Terminated
PS C:\Users\njspr\Documents\GitHub\cis-457-projects\Project1>

Windows PowerShell
PS C:\Users\njspr\Documents\GitHub\cis-457-projects\Project1> java FTPServer
Client 1 connected
Client 1 disconnected
```

## All commands:



The image shows two side-by-side Windows PowerShell windows. The left window runs the `java FTPServer` command, showing 'Client 1 connected' and 'Client 1 disconnected'. The right window runs the `java FTPClient` command, showing a 'Welcome to the Text FTP Server!' message, a list of files on the server, and the successful upload and download of `testFile1.txt` and `testFile2.txt` respectively. The session ends with 'Closing connection...' and 'Terminated'.

```
PS C:\Users\njspr\Documents\GitHub\cis-457-projects\Project1> java FTPServer
Client 1 connected
Client 1 disconnected

PS C:\Users\njspr\Documents\GitHub\cis-457-projects\Project1> java FTPClient
connect 127.0.0.1 7171

Welcome to the Text FTP Server!
list: | retr: <filename> | stor: <filename> | quit

list:

Files on server:
  testFile1.txt
  testFile2.txt
  testFile3.txt
  testFile4.txt
(End of files)

Enter next command:

stor: testFile1.txt

Uploading file to server...
File uploaded!

Enter next command:

list:

Files on server:
  testFile1.txt
  testFile2.txt
  testFile3.txt
  testFile4.txt
(End of files)

Enter next command:

retr: testFile2.txt

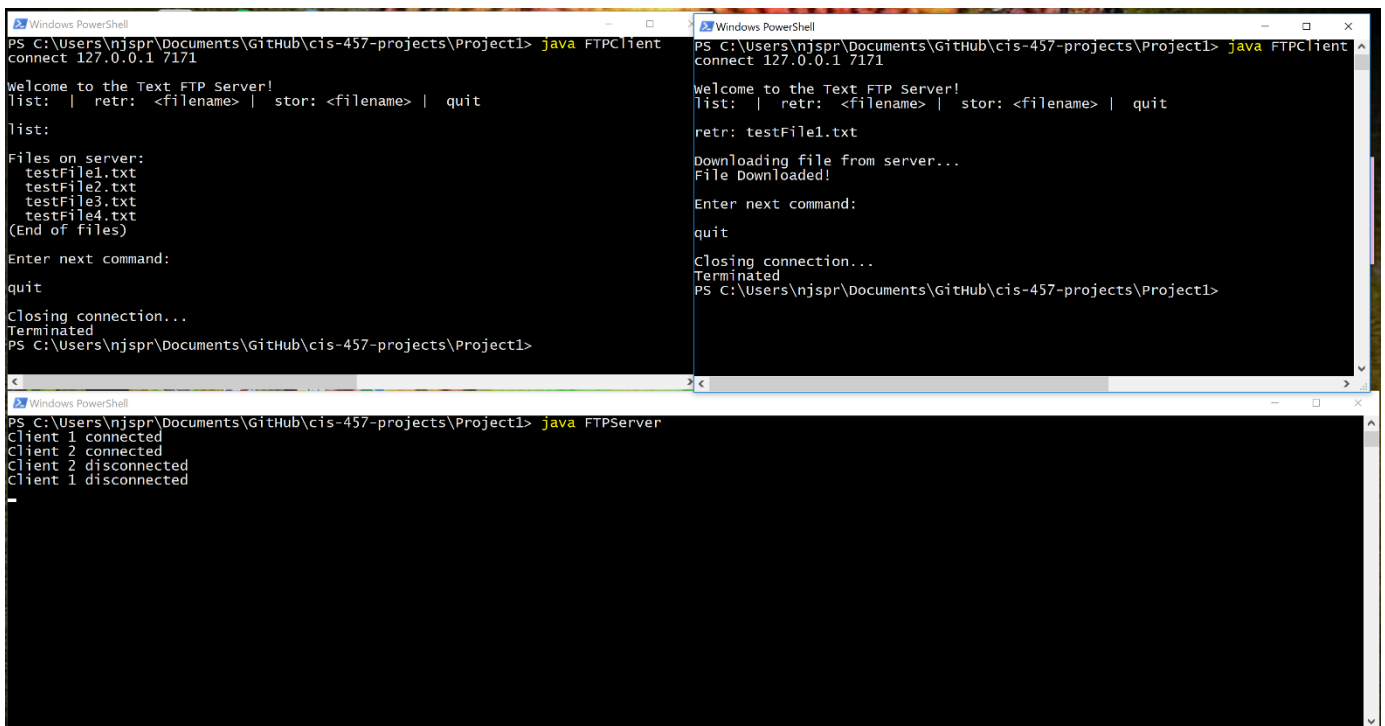
Downloading file from server...
File Downloaded!

Enter next command:

quit

Closing connection...
Terminated
PS C:\Users\njspr\Documents\GitHub\cis-457-projects\Project1>
```

## Multiple clients:



The image shows three Windows PowerShell windows. The top-left window runs `java FTPClient` and shows a 'Welcome to the Text FTP Server!' message, a list of files, and the successful download of `testFile1.txt`. The top-right window runs `java FTPClient` and shows a 'Welcome to the Text FTP Server!' message, a list of files, and the successful download of `testFile1.txt`. The bottom window runs `java FTPServer` and shows 'Client 1 connected', 'Client 2 connected', 'Client 2 disconnected', and 'Client 1 disconnected'.

```
PS C:\Users\njspr\Documents\GitHub\cis-457-projects\Project1> java FTPClient
connect 127.0.0.1 7171

Welcome to the Text FTP Server!
list: | retr: <filename> | stor: <filename> | quit

list:

Files on server:
  testFile1.txt
  testFile2.txt
  testFile3.txt
  testFile4.txt
(End of files)

Enter next command:

quit

Closing connection...
Terminated
PS C:\Users\njspr\Documents\GitHub\cis-457-projects\Project1>

PS C:\Users\njspr\Documents\GitHub\cis-457-projects\Project1> java FTPClient
connect 127.0.0.1 7171

Welcome to the Text FTP Server!
list: | retr: <filename> | stor: <filename> | quit

retr: testFile1.txt

Downloading file from server...
File Downloaded!

Enter next command:

quit

Closing connection...
Terminated
PS C:\Users\njspr\Documents\GitHub\cis-457-projects\Project1>

PS C:\Users\njspr\Documents\GitHub\cis-457-projects\Project1> java FTPServer
Client 1 connected
Client 2 connected
Client 2 disconnected
Client 1 disconnected
```