NapServer Report

Project 2 - GV-NAP File Sharing System

Professor El-Said
CIS 457 – 10
20 November 2017

Project Group Members:

Louis Sullivan
Javier Ramirez-Moyano
Matthew Schuch
Brendon Murthum

Project completion percentage: 100%

Table of Contents

p. 2 Introduction to Napster Redesign Project

p. 2 Report Overview

p. 2 Project Features

p. 3 Descriptions of Classes

p. 3 Problems and Resolution

p. 4 Project Summary

p. 6 Screen Captures of Complete Session

p. 12 Code Samples – CentralServer.java

p. 13 Code Samples – ClientHandler.java

p. 18 Code Samples – GUI.java

p. 29 Code Samples – Host.java

p. 34 Code Samples – HostedDescriptions.java

p. 36 Code Samples – HostServer.java

Napster Redesign

The system that we have designed is a loosely mirrored example of the Napster system--a peer-to-peer file-sharing system. The basic architecture is that of a main-server that handles connections of clients that send the main-server their information of which files they are making available to other clients. This allows other clients to be able to search the main-server for available files, to then connect to the other client individually to manage downloading the files from that client. It is important to note that the main-server acts simply as a middle-man that handles simple tables of user meta-data so that the users can act accordingly.

The follow are some critical notes on Napster, orginally "Rhapsody." Here's a quote from Wikipedia on Napster.

> Initially, Napster was envisioned as an independent peer-to-peer file sharing service by Shawn Fanning. The service operated between June 1999 and July 2001.[10] Its technology allowed people to easily share their MP3 files with other participants.[11] Although the original service was shut down by court order, the Napster brand survived after the company's assets were liquidated and purchased by other companies through bankruptcy proceedings.
>
> Later companies and projects successfully followed its P2P file sharing example such as Gnutella, Freenet, Kazaa, BearShare, and many others.

Report Overview

More specifically this document serves to explain the functionality of the system with full detail. Our document will do this by (1) showing screen captures of example run-throughs, (2) having discussion of the basic logic of the system and its individual modules, and (3) further reflecting on issues found and resolved over the course of the project.

Features

The following are notable features that we have implemented to this system:

- Our *NapServer* features a GUI interface generated with javax.swing.
- The Client-GUI allows the user to connect with a chosen user-name, which is constrained by the Central-Server to be unique.
- The Central-Server stores an ever-changing list of available files based on current connected users.
- The connections between Central-Server and its Clients are setup to have full availability of error-handling (if desired in future updates, we could allow the Client-GUI to fully show disconnects and errors, number of connected users, total amount of available data).
- The Client-GUI constrains the user to have a properly-named XML document.

Description of Classes

CentralServer

This class handles the full initiation of the server that handles multiple hosts connecting to be able to search for files. This is done with multithreading to be able to have concurrent connections. This class directs each new client to the ClientHandler Class to be handled more directly.

ClientHandler

This class, on the server-side, handles the commands taken from the client and manages the data on the main server.

GUI

This class is the initiated class to run the client program. It takes in user data and initiates the appropriate methods and instantiates the proper objects.

Host

This class manages, more directly, the internal methods that connect the client to the Central-Server. It also handles parsing the data sent-to and retrieved-from the central server.

HostedDescriptions

This class creates the main object on the Central-Server. It provides methods internally to handle new clients, leaving clients, and queries of available files.

HostServer

This class handles having each client also simultaneous acting as an FTP-server to other clients. This class is an extension of Thread for the sake of it running concurrently with the GUI. Within this class, it welcomes new connections to send those to their own threads as well.

Problems and Resolution

In a certain way its hard to see the problems as problems once they are resolved. The following is a reflection on a few of the resolved (and unresolved) problems that came up in development.

Making Sure the Code is Understood

There was a subtle difficulty in making sure the entire group had full understanding of the code at a given moment. This may have given friction to allow for individuals to feel that they could work on it on their own. With frequent, critical, updates--that were also mostly the brainchild of any one developer at a given time--it was difficult to be sure that everyone was on the page at the same time. To handle this,

there were lots of in-code comments and an implemented area to track developer developments on the Github page.

Finding Time to Meet for Coding

The timing of this project in the semester had it competing for attention with projects from other classes. with the added social pressure of any group-project, we certainly all found the time to meet and regularly inquired on new times to meet-up through email. Even though I (Brendon) found our meet-ups fruitful and regular enough to be productive, it still seemed like there was some difficulty in getting everyone together at one time in crucial moments.

Finding Proper Dynamic to Resolve Design Conflicts

First, I should say any conflicts were resolved and that all the involved developers were incredibly inclined to reduce tensions and find resolve. It was a wonderful group.

There were a couple dilemmas in design choice that surfaced. A notable one was how to implement a "Quit" function. This may have been instructed as an allocated creative choice, but by the writing and sample image of the assignment document it was unclear between us if a "Disconnect" button needed to exist, or if a typed "Quit" in the input-command area of the FTP window should disconnect the user from the Central-Server. With full intention of following the required functionality, this was forced as an issue. Some more resolve may have come from directly inquiring from the professor, but this was feared to come with a "I told you so" attached feeling. We came to decide to have the only way to disconnect from the entire system was by exiting the main GUI.

Complications in Use of so Many Classes

I should note that a proper UML-diagram early in development to solidify developer's potential choices in code may have allowed for more individual-work to happen due to a knowledge of their conductivity with the center head-space. It was hard to foresee the variability in coding styles and knowledge of classes between developers, noting that one may be hesitant to put themselves in a position to admit lack of knowledge.

The many interacting classes in parallel with the many interacting systems made it difficult to code and have fruitful communication. Both developers in any conversation had to have full understanding of the entire link to be able to debug well.

Summary

This project has been a wondrous exercise of several things:

- In classes and objects in Java.
- In small-group code-management with Github, email-chains, and code checkpoints.
- In early team project-design.
- In handling ports, streams, and multi-threading.
- In many many cases, writing much-needed error-handling.

There were many long afternoons spent with two to three guys typing out code and debugging time away. This project took much longer than expected. I am confident in saying that each of us have come to be much more comfortable with the code and the needed-head-space to complete projects at this scope.

Screen Captures of Complete Session

This example runs a sample user, “Alice,” through a typical session.

The initial state is that both clients are running and the server is running, but that nobody is connected (Figure 1). Alice then connects to the main system (Figure 2). Then Bob connects to the main system (Figure 3). Alice requests a keyword search of “Image” (Figure 4), then of “Comedy,” (Figure 5, Figure 6) then Alice connects to Bob directly to download a file (Figure 7, Figure 8). Alice downloads the file, “fortytwo.txt” (Figure 9). Alice then disconnects from Bob (Figure 10), and disconnects from the main server (Figure 11).

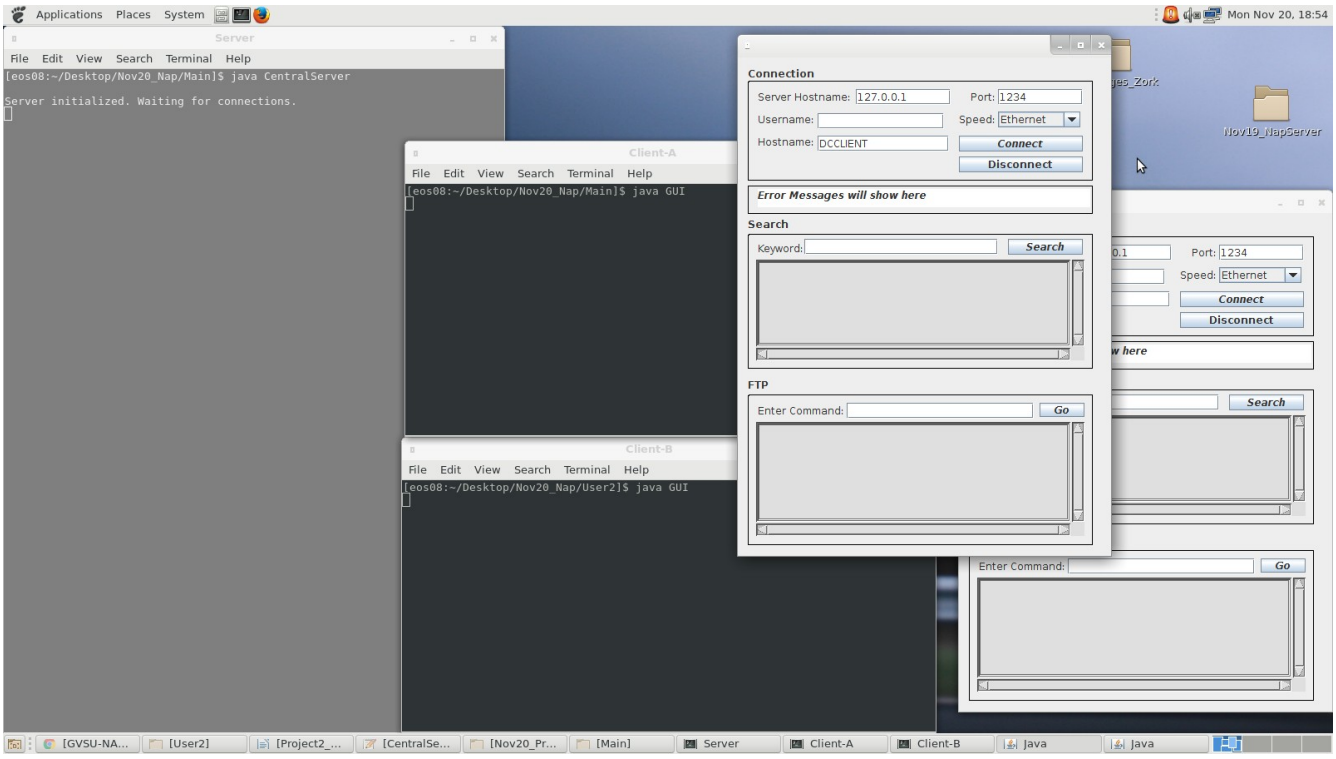

Figure 1: Initial State. The server is running. No users are connected.

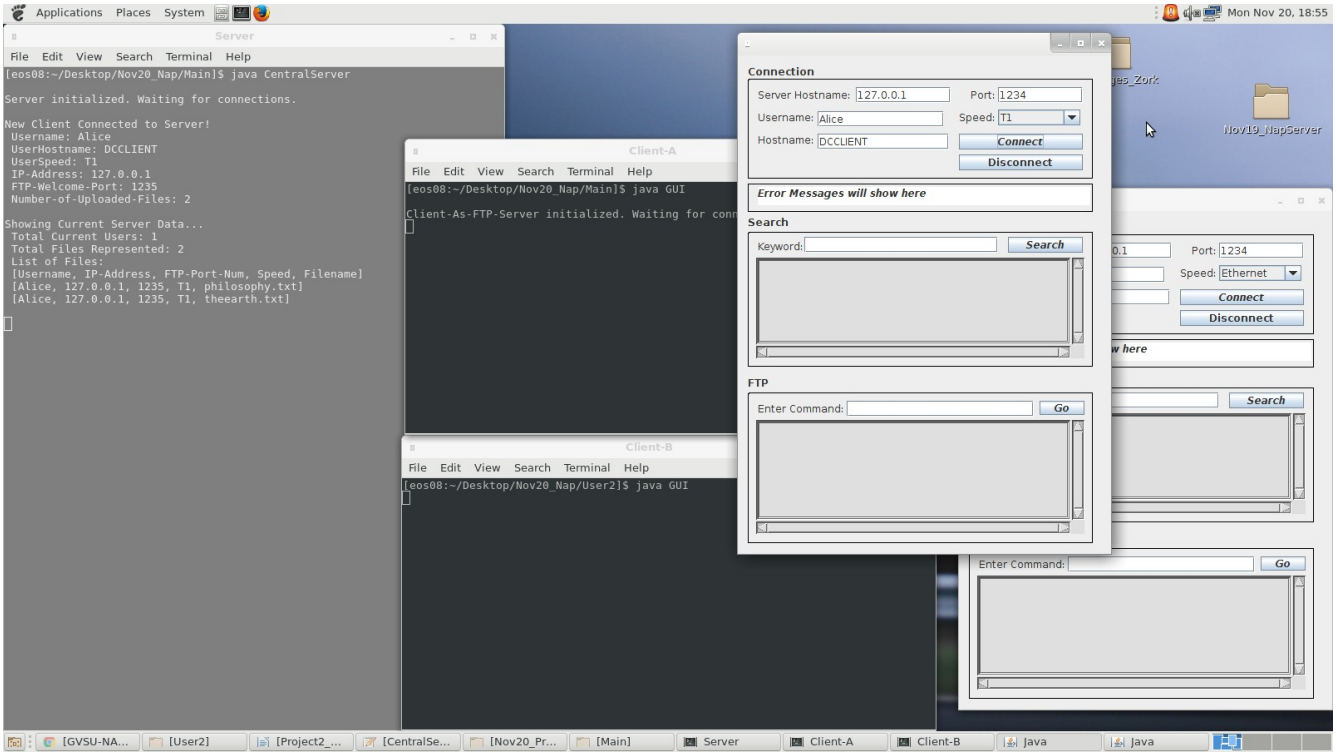

Figure 2: Alice hits connect. The server shows in terminal the now-available files.

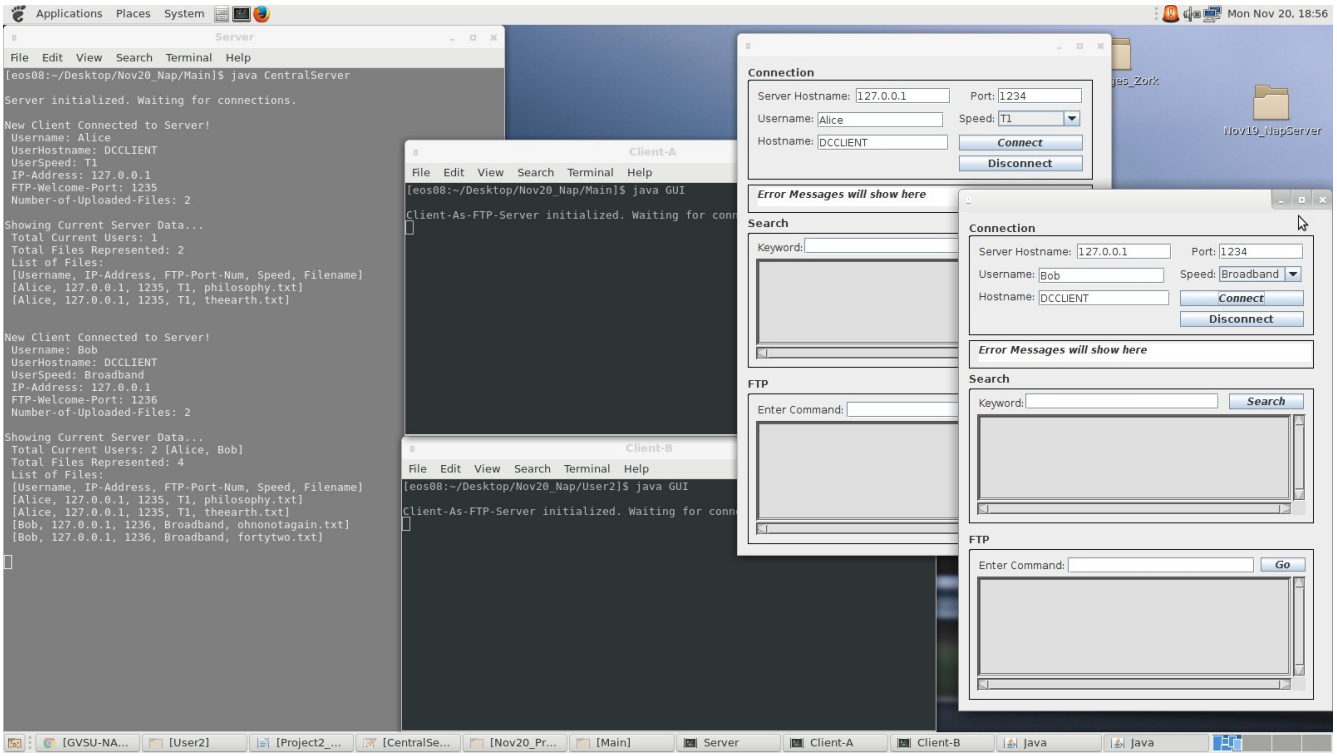

Figure 3: Bob hits connect. The server updates, now showing four files available.

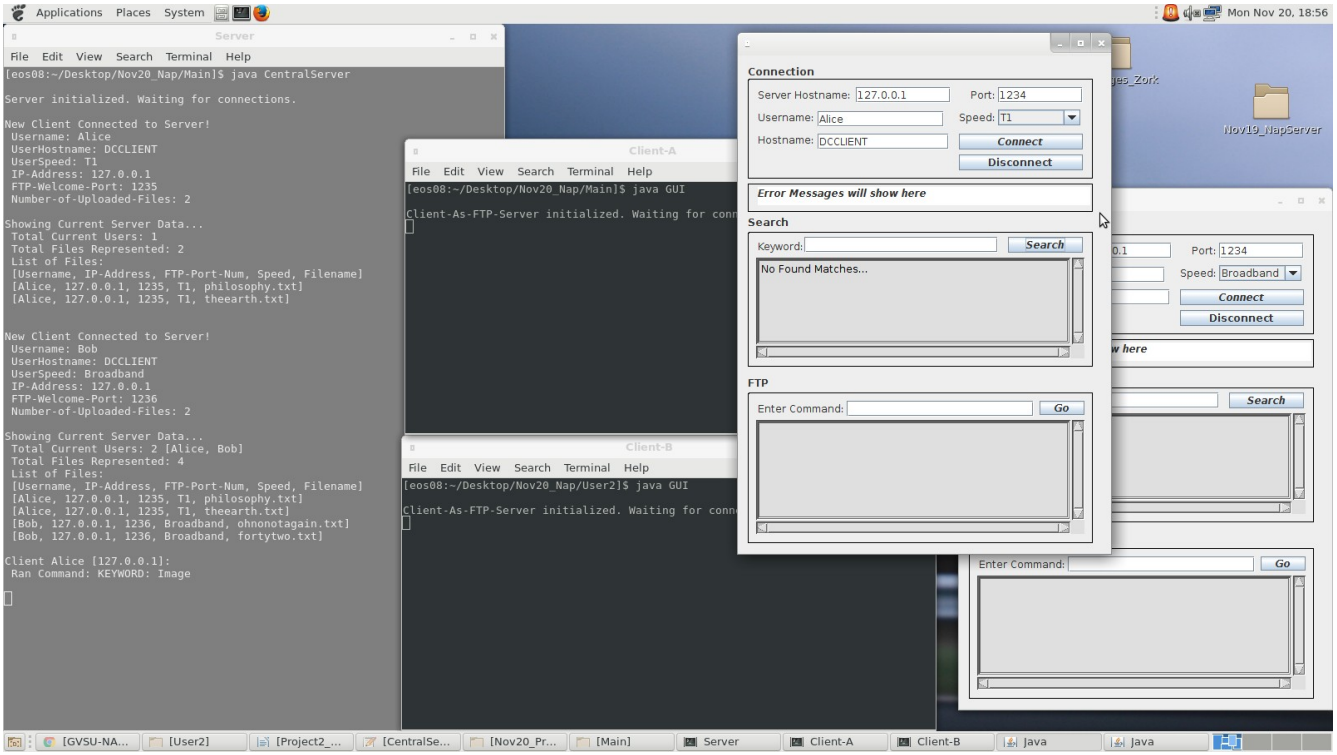

Figure 4: Alice searches for files. This key, “Image,” returns no available files (Success).

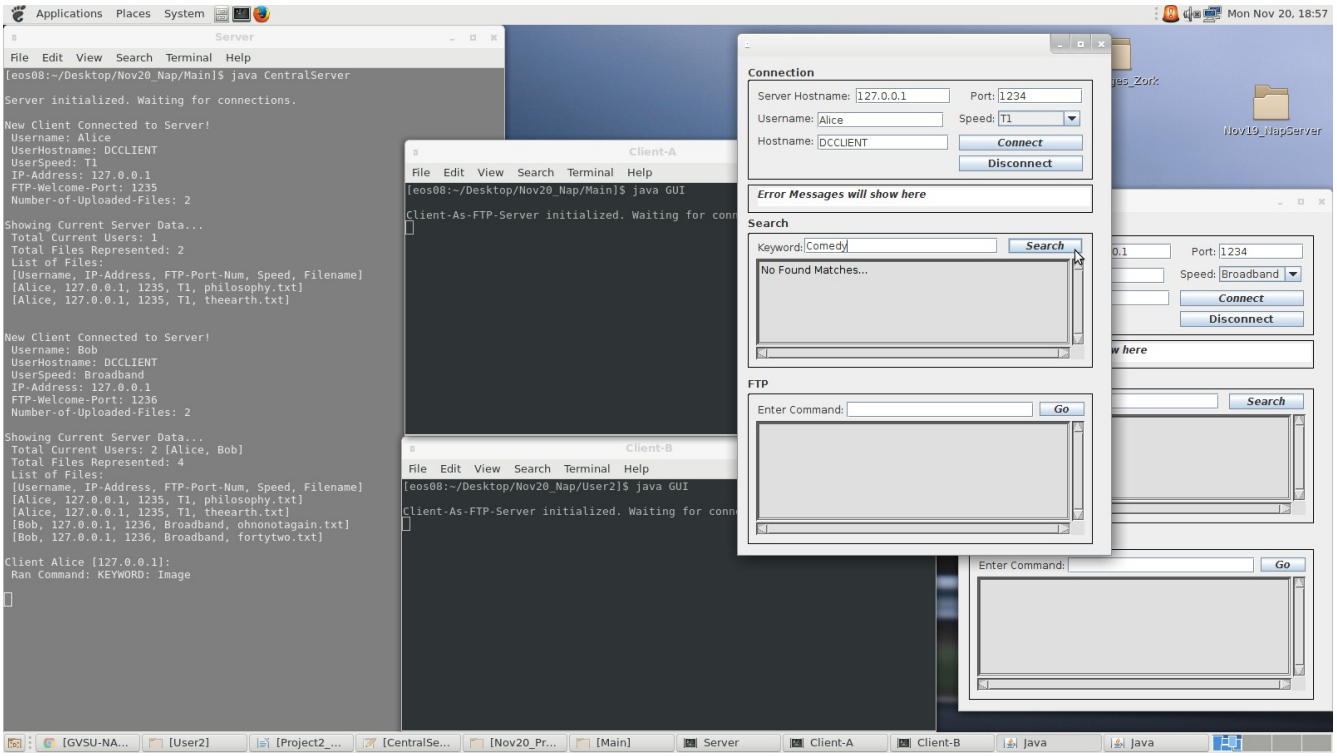

Figure 5: Alice is about to search for the key, “Comedy.”

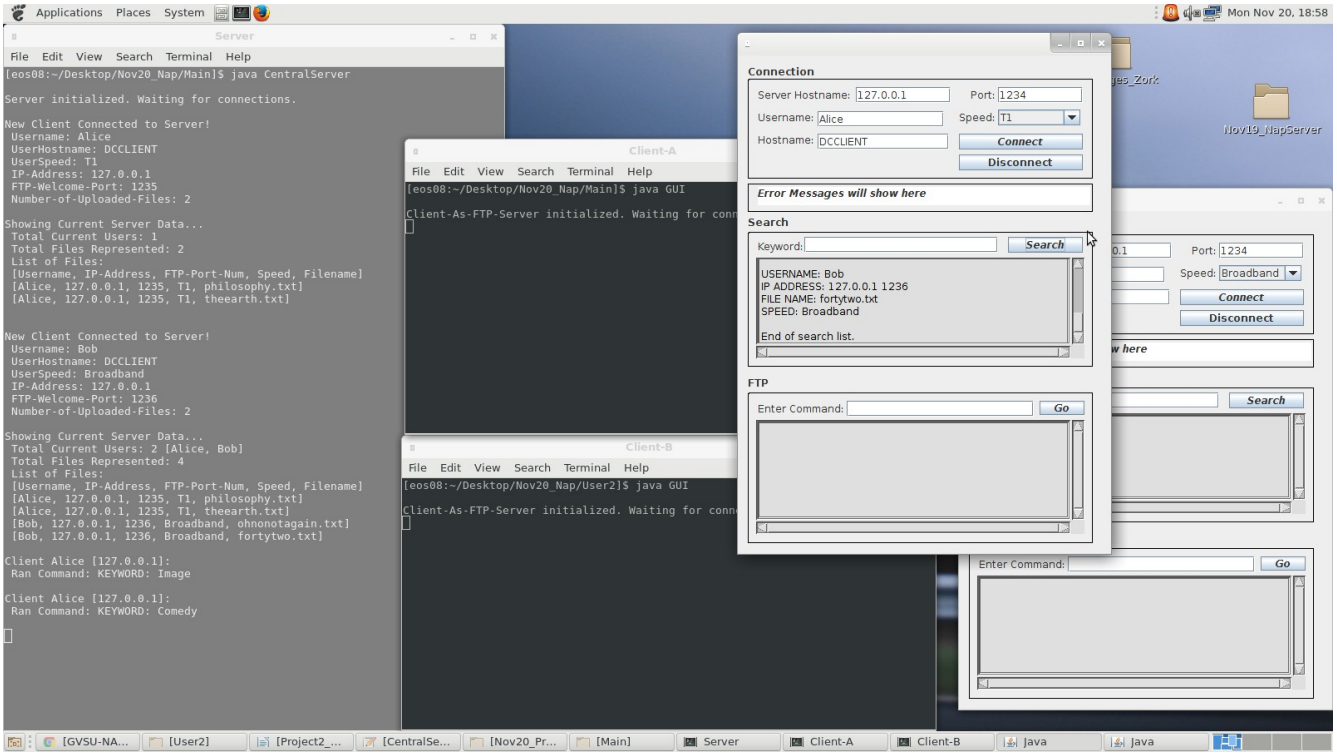

Figure 6: Alice submits the key search for “Comedy.” This returns a list of files on the GUI.

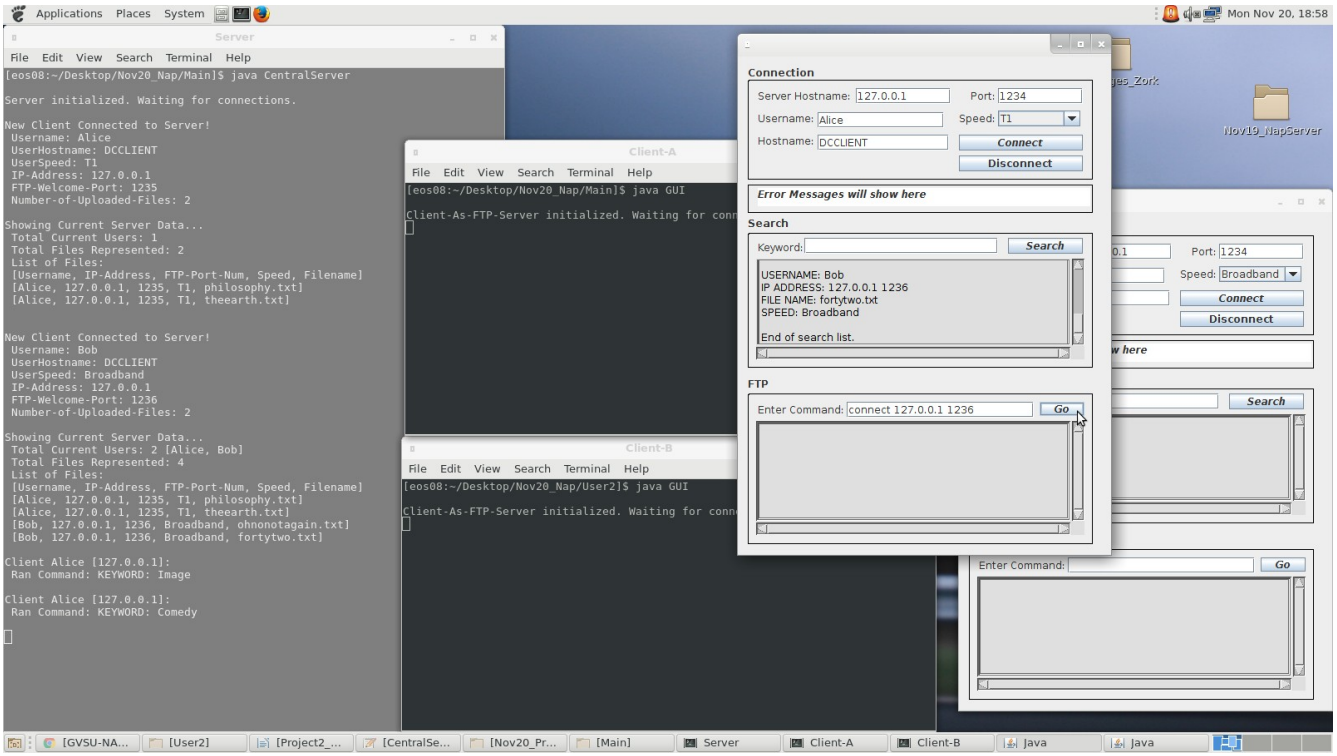

Figure 7: Alice is about to connect to Bob directly.

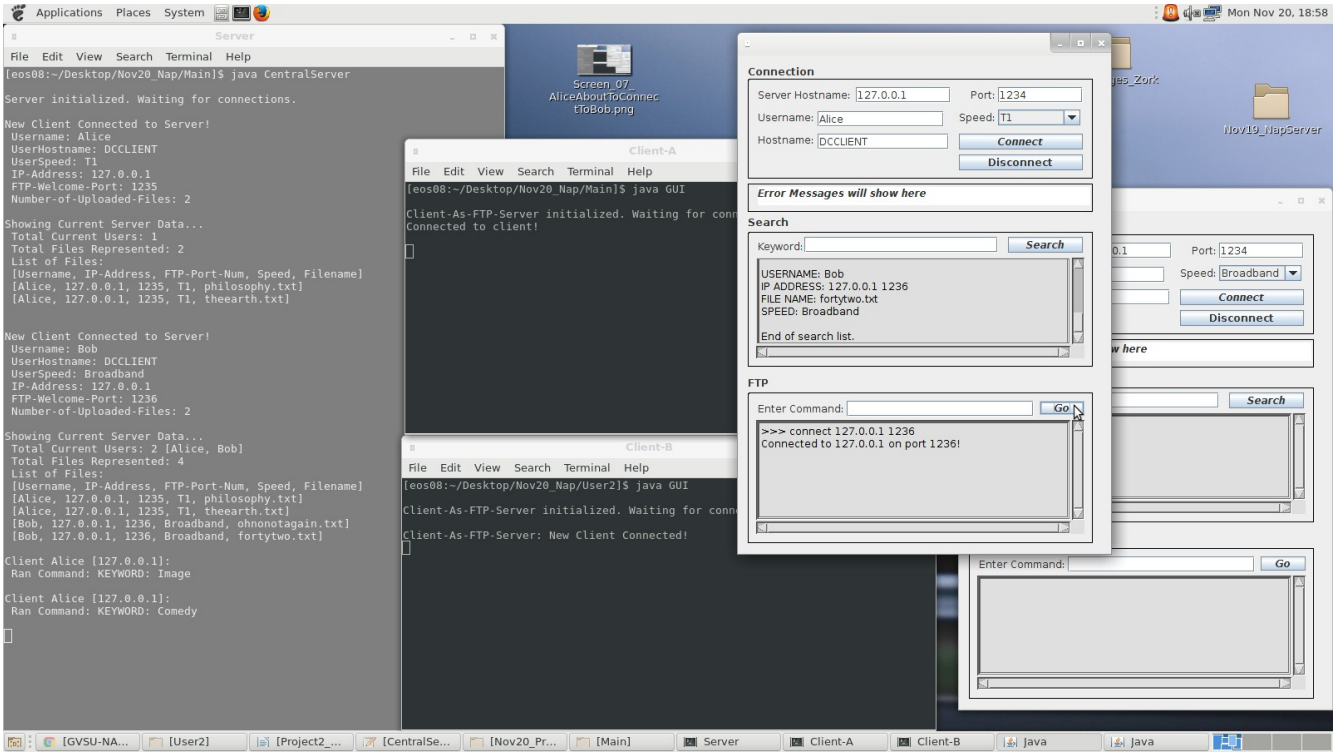

Figure 8: Alice is now connected to Bob directly.

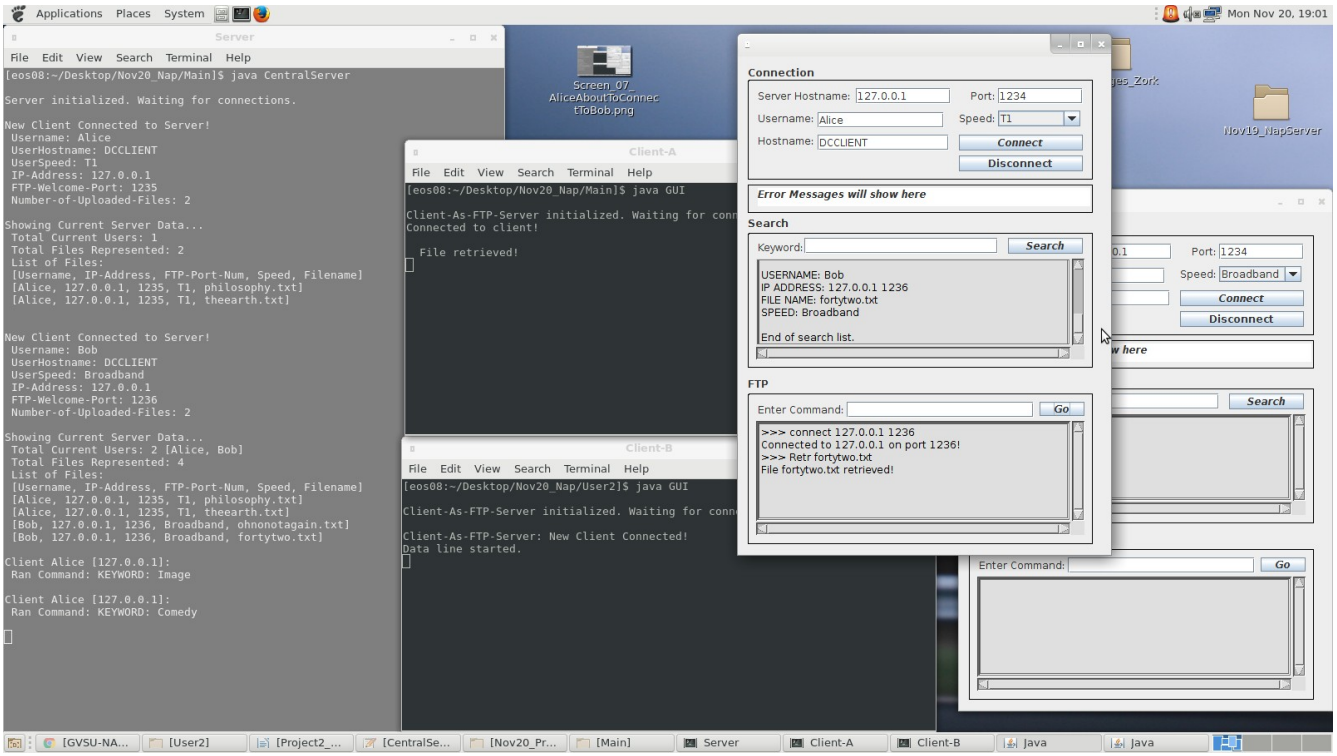

Figure 9: Alice retrieves a file, “fortytwo.txt,” from Bob.

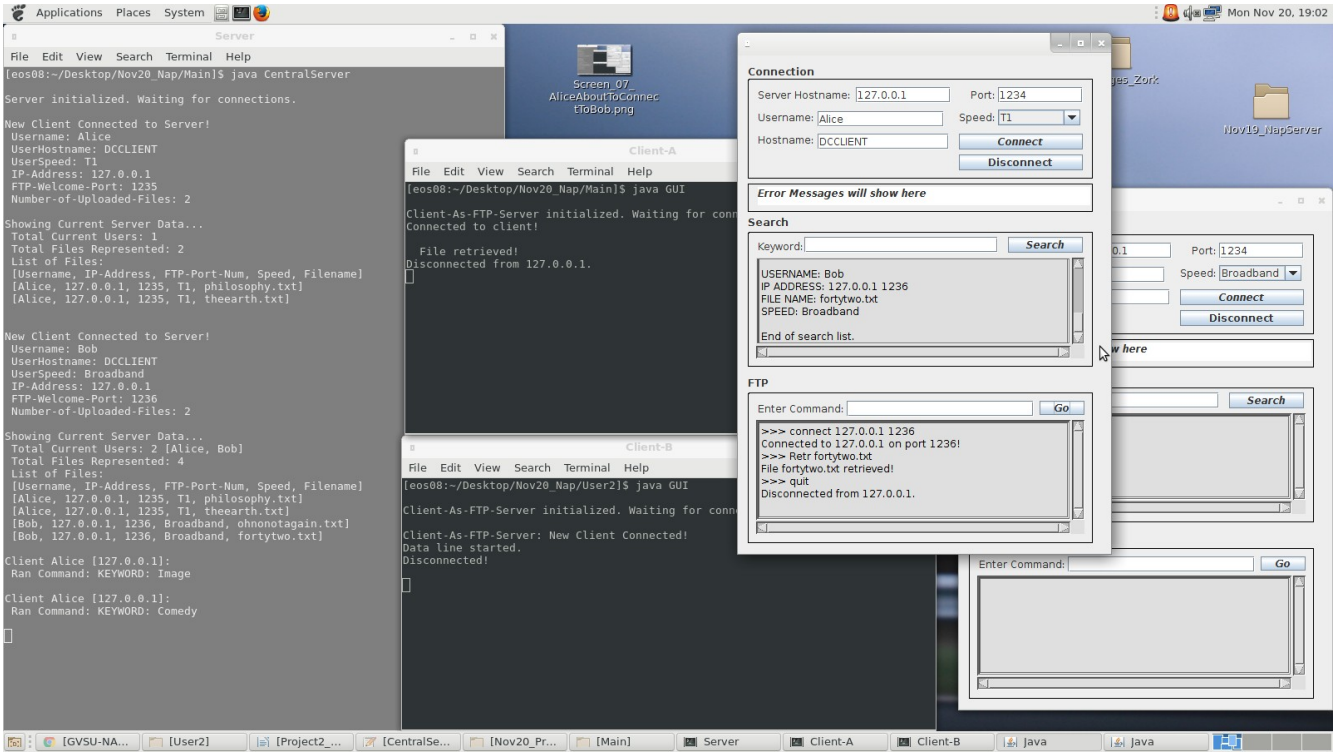

Figure 10: Alice disconnects from Bob.

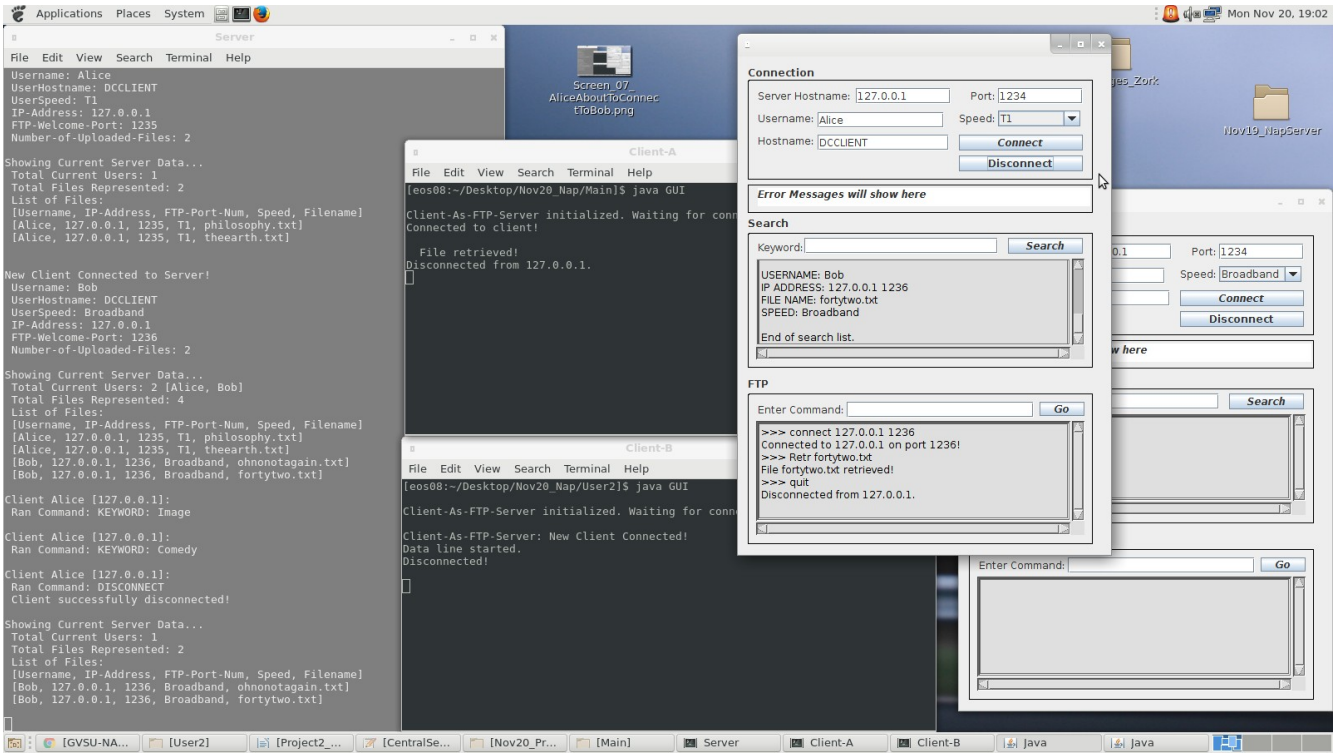

Figure 11: Alice disconnects from the Central-Server. This leaves only Bob in the system.

Code Samples – CentralServer.java

```
import java.net.*;
import java.util.*;
import java.io.*;
import java.awt.*;


/**
 *
 * @author                  Brendon Murthum
 *                      Javier Ramirez-Moyano
 *                      Matthew Schuch
 *                      Louis Zullivan
 *
 */

public class CentralServer {
       private static HostedDescriptions totalHostedDescriptions = new HostedDescriptions();

       private static final int welcomePort = 1234;
    private static ServerSocket welcomeSocket;

    public static void main(String[] args) throws IOException {

        /* Show server starting status */
        System.out.println(" ");
        System.out.println("Server initialized. Waiting for " +
                           "connections.");

        /* Initialize the welcome socket */
        try {
            welcomeSocket = new ServerSocket(welcomePort);
        }
        catch (IOException ioEx) {
            System.out.println(" ");
            System.out.println("  ERROR: Unable to set up port!");
            System.exit(1);
        }

        /* Perform a loop to wait for new connections */
        do {
            /* Wait for client... */
            Socket connectionSocket = welcomeSocket.accept();

            System.out.println("\nNew Client Connected to Server!");

            /* For debugging */
            // System.out.println("  DEBUG: New Connection's IP: " +
            //     connectionSocket.getInetAddress());

            /*
             Create a thread to handle communication with this client and
             pass the constructor for this thread a reference to the
             relevant socket and user IP.
            */
            ClientHandler handler = new ClientHandler(connectionSocket,
                       totalHostedDescriptions);

            /* Start a new thread for this client */
            handler.start();

        } while (true);


    /* End of main() */
    }
/* End of public class CentralServer */
}
```

Code Samples – ClientHandler.java

```
import java.net.*;
import java.util.*;
import java.io.*;
public class ClientHandler extends Thread{

       /** This handles the stream from the command-line of client */
    private Scanner inFromClient;

    /** This allows for identification of specific users in streams */
    private String remoteIP;

    /** This socket takes commands from client */
    private Socket controlListen;

    /** This handles the output stream by command-line to client */
    private PrintWriter outToClient;

    /** This is used to grab bytes over the data-line */
    private String recvMsg;

    /** This is for thread control */
    private boolean endThread = false;

    /** This takes the user information from the host     */
    private String userInformation;
    private String UserName;
    private String UserHostName;
    private String UserSpeed;
    private String UserFTPWelcomePort;
    private String tmpFileName, tmpKeyWords;

    private HostedDescriptions allHostedDescriptions;

       /*****************************************************************
    *
    * Beginning of thread.
    * This constructor marks the beginning of a thread on the server.
    * Things here happen once, exclusively with THIS connected client.
    *
    *****************************************************************/
   public ClientHandler(Socket controlListen, HostedDescriptions totalHostedDescriptions) {

       /* Make this all... point to the total... */
       allHostedDescriptions = totalHostedDescriptions;

       try {
           /* Setting up a threaded input control-stream */
           inFromClient = new Scanner (
               controlListen.getInputStream());
           /* Setting up a threaded output control-stream */
           outToClient =
               new PrintWriter(controlListen.getOutputStream());
           /* For error handling */
           /* Get IP from the control socket for future connections */
           remoteIP = controlListen.getInetAddress().getHostAddress();

           /* For debugging */
           // System.out.println("  DEBUG: A new thread was successfully setup.");
           // System.out.println("");

       } catch(IOException ioEx) {
           ioEx.printStackTrace();
           System.out.println("  ERROR: Could not set up a " +
                       "threaded client input and output stream.");
       }
   }
```

```
/*****************************************************************
*
* Beginning of main thread code.
* This method marks the threaded area that this client receives
* commands and handles. When this receives "QUIT" from the client,
* the thread closes.
*
*****************************************************************/
 public void run(){

    /* Keeps tracks of when to close the thread */
     boolean stayAlive = true;
     /** Gets user information from the host      */
     StringTokenizer userTokens;

     try {
         /* This grabs the string of data with filenames and keys */
            userInformation = inFromClient.nextLine();

            /* For Debugging. This shows what is read from control-line. */
            // System.out.println("  DEBUG: Read-In: " + userInformation);

            /* Make the string parseable by tokens */
            userTokens = new StringTokenizer(userInformation);

            /* Initialize and display the new user's shown username */
         UserName = userTokens.nextToken();
         UserName = UserName.replaceAll("@@", " ");

         /*
             We can error handle multiple users on the same IP (two people
             from the same house) by checking THIS username against the
             stored ones. On User-SUCCESS, of choosing a unique one,
             continue as planned. On User-ERROR, of choosing a taken one,
             never show the main table of info to the user, and break the
             current thread.
             A failure to the user will look as nothing happened, while
             success will look as a table generated.
            */

         /* If username taken, throw error, return a message, and stop thread. */
            if (allHostedDescriptions.isUsernameTaken(UserName)) {
             System.out.println("  ERROR: \""+ UserName + "\" [" + remoteIP + "] must" +
                                " pick another username to connect!");

             /* Send notification to client to rechoose a username */
             outToClient.println("BAD-USERNAME");
             outToClient.flush();

             endThread = true;
             throw new EmptyStackException();
            }
            else {
                /* Send confirmation of username choice */
                outToClient.println("GOOD-USERNAME");
             outToClient.flush();
            }

         System.out.println(" Username: " + UserName);

         /*
          TODO - Change UserHostName
          Brendon - NOV 11, 2017
          The User's IP is grabbed directly from the stream by our
          methods. Right now, the variable remoteIP equals the user's
          actual IP regardless of what the user typed in. I propose we
          change UserHostName to be strictly the User's
          typed-computer-name, to have our program paste the ACTUAL IP
          to the end. Currently, a User could type in a false IP to
          later have another user ping the wrong address.
         */
```

```
            /* Initialize and display the new user's hostname and IP */
            UserHostName = userTokens.nextToken();
            UserHostName = UserHostName.replaceAll("@@", " ");
            System.out.println(" UserHostname: " + UserHostName);

            /* Initialize and display the new user's speed */
            UserSpeed = userTokens.nextToken();
            System.out.println(" UserSpeed: " + UserSpeed);

            /* Display new user's IP */
            System.out.println(" IP-Address: " + remoteIP);

            /* Display new user's FTP Welcome Port */
            UserFTPWelcomePort = userTokens.nextToken();
            System.out.println(" FTP-Welcome-Port: " + UserFTPWelcomePort);

            /* Variables used in parsing the crazy string */
            String tmp_a_Token, tmpFilename, tmp_c_Token;
            try {
                /* First "Filename" */
                tmp_a_Token = userTokens.nextToken();
                /* First "Image.jpg" */
                tmpFileName = userTokens.nextToken();
            }
            catch (Exception e) {
                /* There seems to be no uploads from this user */
                // System.out.println("  DEBUG: No uploads from user.");
            }

            String tmpKeywordList = "";
            int numberOfFilesDescriptions = 0;
            while (true) {
                /* If there are still tokens left */
                try {
                    /* Should mostly be keys */
                    tmp_c_Token = userTokens.nextToken();
                } catch (Exception e) {
                    /* The last submit, as if "Filename" found again */

                    /* Add file description to the main collection */
                    allHostedDescriptions.addValues(UserSpeed, UserHostName, tmpFileName,
tmpKeywordList, UserName, remoteIP, UserFTPWelcomePort);

                    /* For debugging */
                    // System.out.println("Submit: " + tmpFileName + tmpKeywordList);

                    /* To show number of uploads at user entry */
                    numberOfFilesDescriptions++;

                    break;
                }
                if (tmp_c_Token.equals("FILENAME")) {

                    /* Add file description to the main collection */
                    allHostedDescriptions.addValues(UserSpeed, UserHostName, tmpFileName,
tmpKeywordList, UserName, remoteIP, UserFTPWelcomePort);

                    /* For debugging */
                    // System.out.println("Submit: " + tmpFileName + tmpKeywordList);

                    /* To show number of uploads at user entry */
                    numberOfFilesDescriptions++;

                    tmpKeywordList = "";
                    /* "Image.jpg" */
                    tmpFileName = userTokens.nextToken();
                }
                else {
                    tmpKeywordList = tmpKeywordList + " " + tmp_c_Token;
                }
            }
```

```
    /*
     Once the file-descriptions are parsed, display a success
     report of number of uploads to output on the server.
     */
    System.out.println(" Number-of-Uploaded-Files: " + numberOfFilesDescriptions);

} catch (Exception e) {
    /* Host did not supply user information  */
    System.out.println("  ERROR: Host did not supply valid information");
    endThread = true;
}

/* End the thread */
if (endThread == true) {
    System.out.println("  ERROR: Ending user's thread");

    /* Remove all rows with this username */
    allHostedDescriptions.remove(UserName);

    /* Show the server data on user-leave */
    System.out.println(" ");
    allHostedDescriptions.showData();
    System.out.println(" ");

    return;
}

/* For show. To separate user-logins. */
System.out.println(" ");

/* For debugging */
// System.out.println("  DEBUG-01: User's thread running");

/* For show. View the server's main data points on each new entrance! */
allHostedDescriptions.showData();

/* For show. To separate information. */
System.out.println(" ");

/* The controlling loop that keeps the user alive */
while (stayAlive) {

       /* For debugging */
    // System.out.println("  DEBUG-02: User's thread running");

    /* This reads the command from the client */
    try {
        recvMsg = inFromClient.nextLine();

    } catch (Exception e) {
        /* Client left early, or otherwise */
        System.out.println("Client " + UserName + " [" + remoteIP + "] left early!");

        /* Remove all rows with this username */
        allHostedDescriptions.remove(UserName);

        /* Show the server data on user-leave */
        System.out.println(" ");
        allHostedDescriptions.showData();
        System.out.println(" ");

        break;
    }

    /* For token handling */
    String returnedString = "";
    String currentToken;
    StringTokenizer tokens = new StringTokenizer(recvMsg);

    /* Client command, "UPDATE" or another. */
    String commandFromClient = tokens.nextToken();
```

```
/* For variable handling */
String key, fileName, keyWords;
String totalKeys = "";

/* While checks for more user input in the token */
while (tokens.hasMoreTokens()) {

    /* Client opened dataport, "1079," for the server to
       connect to */
    /* NOTE - This "clientDataPort" may be changed later. It was added to
     * have continuity with the FTP-server that we developed. Oct 25.
     */

    try {

        /* This commands initiates the keyword-search */
        if(commandFromClient.equals("KEYWORD")){
            /*Passes the last arguement into the key */
            key = tokens.nextToken();

            /* For debugging */
            // System.out.println("Key " + key);
            totalKeys = totalKeys + " " + key;
        }
        /* This command updates the files the host have in their root directory */
        else if(commandFromClient.equals("UPDATE")){

            /* Passes the last arguement into the key */
            fileName = tokens.nextToken();
            System.out.println("Filename: " + fileName);

            keyWords = tokens.nextToken();
            System.out.println("keyWords " + keyWords);
        }
    }
    catch (Exception e) {
    /* Was there a token error? */

        /* For debugging */
        // System.out.println(" DEBUG: End of Parsing Tokens");
    }

   /* End of hasMoreTokens loop */
}

/* For server log printing */
System.out.println("Client " + UserName + " [" + remoteIP + "]: ");

/* For server log printing */
if (commandFromClient.equals("QUIT")) {
    System.out.println("Client " + UserName + " [" + remoteIP + "]
                disconnected!");

    /* Remove all rows with this username */
    allHostedDescriptions.remove(UserName);

    stayAlive = false;
}
else if (commandFromClient.equals("KEYWORD")) {
    System.out.println(" Ran Command: KEYWORD:" + totalKeys);

    /*
     HERE, do the methods for figuring which things to send
     back to the client.
    */
    String toSendToClient = allHostedDescriptions.getKeywordData(totalKeys);

    /* For debugging */
    // String tempstring = "This works!";

    /* Send the query to the server! */
```

```
                outToClient.println(toSendToClient);
                       outToClient.flush();

                /* For debugging */
                // System.out.println("  DEBUG: Succesfully Sent: " + toSendToClient);
            }
            else if (commandFromClient.equals("UPDATE")) {
                System.out.println(" Ran Command: UPDATE");
            }
            else if (commandFromClient.equals("DISCONNECT")) {
                System.out.println(" Ran Command: DISCONNECT");

                /* Client left early, or otherwise */
                System.out.println(" Client successfully disconnected!");

                /* Remove all rows with this username */
                allHostedDescriptions.remove(UserName);

                /* Show the server data on user-leave */
                System.out.println(" ");
                allHostedDescriptions.showData();
                System.out.println(" ");

                break;
            }


            /* For server log printing */
            System.out.println(" ");


        /* End of the other while loop */
        }

    /* End of threading run() */
    }

/* End of the thread class */
}
```

Code Samples – GUI.java

```
import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.BorderLayout;
import javax.swing.BoxLayout;
import javax.swing.JLabel;
import javax.swing.border.LineBorder;
import java.awt.Color;
import javax.swing.JButton;
import java.awt.Font;
import javax.swing.JTextField;
import javax.swing.JList;
import javax.swing.AbstractListModel;
import javax.swing.JComboBox;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JScrollPane;
import javax.swing.ScrollPaneConstants;
import java.awt.Scrollbar;
import java.awt.TextArea;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.net.*;
import java.util.*;
import java.io.*;

/* For tokens */
import java.util.*;
import javax.swing.JTextArea;
```

```
public class GUI {

    /* Lots of GUI setup */
       private JFrame frame;
       private JTextField textFieldServerHostname;
       private JTextField textFieldUsername;
       private JTextField textFieldHostname;
       private JTextField textFieldPort;
       private JTextField keywordSearchField;
       private JTextField ftpCommandField;
       private JComboBox comboBoxSpeed;

       /* Initializing helper variables to empty */
       private String commandHistory = "";
       private String ipAddress = "";
       private String portNum = "";

       /** This handles connection to the Central-Server */
       private Host host = new Host();

       /** This handles client being server to other FTP-connections */
       private HostServer hostServer;

       /** This allows for FTP-connection with another host */
       private Socket controlSocket;

       /** Stores whether the user is currently connected to a host */
       private boolean isConnectedToOtherHost = false;

       /* This handles the control-line out stream */
    PrintWriter outToHost = null;

    /* This handles the control-line in stream */
    Scanner inFromHost = null;

       /* This is used as a helper in initial connection to Central-Server */
       private String hostFTPWelcomeport;
       /* Holds the condition of whether Host-as-Server is setup */
       private boolean alreadySetupFTPServer = false;
       /* Holds the condition of whether connected to the Central-Server */
       private boolean isConnectedToCentralServer = false;

       /**
        * Launch the application.
        */
       public static void main(String[] args) {
              EventQueue.invokeLater(new Runnable() {
                     public void run() {
                            try {
                                   GUI window = new GUI();
                                   window.frame.setVisible(true);
                            } catch (Exception e) {
                                   e.printStackTrace();
                            }
                     }
              });
       }

       /**
        * Create the application.
        */
       public GUI() {
              initialize();
       }

       /**
        * Initialize the contents of the frame.
        */
       private void initialize() {
              frame = new JFrame();
              frame.setBounds(100, 100, 451, 628);
```

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.getContentPane().setLayout(null);

JPanel panelConnection = new JPanel();
panelConnection.setBorder(new LineBorder(new Color(0, 0, 0), 1, true));
panelConnection.setBounds(12, 28, 415, 120);
frame.getContentPane().add(panelConnection);
panelConnection.setLayout(null);

JLabel lblServerHostname = new JLabel("Server Hostname:");
lblServerHostname.setFont(new Font("Dialog", Font.PLAIN, 12));
lblServerHostname.setBounds(12, 12, 133, 15);
panelConnection.add(lblServerHostname);

JLabel lblUsername = new JLabel("Username:");
lblUsername.setFont(new Font("Dialog", Font.PLAIN, 12));
lblUsername.setBounds(12, 39, 119, 15);
panelConnection.add(lblUsername);

JLabel lblPort = new JLabel("Port:");
lblPort.setFont(new Font("Dialog", Font.PLAIN, 12));
lblPort.setBounds(268, 12, 39, 15);
panelConnection.add(lblPort);

JLabel lblNewLabel = new JLabel("Hostname:");
lblNewLabel.setFont(new Font("Dialog", Font.PLAIN, 12));
lblNewLabel.setBounds(12, 66, 68, 15);
panelConnection.add(lblNewLabel);

JLabel lblSpeed = new JLabel("Speed:");
lblSpeed.setFont(new Font("Dialog", Font.PLAIN, 12));
lblSpeed.setBounds(254, 39, 49, 15);
panelConnection.add(lblSpeed);

JButton btnConnect = new JButton("Connect");
btnConnect.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent e) {

		String serverHostname = textFieldServerHostname.getText();

		String username = textFieldUsername.getText();
		username = username.replaceAll(" ", "@@");

		String port = textFieldPort.getText();

		String hostname = textFieldHostname.getText();
		hostname = hostname.replaceAll(" ", "@@");

		String speed = comboBoxSpeed.getSelectedItem().toString();

		if (alreadySetupFTPServer == true) {
		    /* For debugging */
		    System.out.println("  DEBUG-06: FTP-Server is already running.");

		}

		/* TODO - For displaying to the user */
		// IF NO USERNAME
		// THEN DISPLAY "Please enter a USERNAME"

		/* Makes the host a FTP server */
		if (!username.equals("") && alreadySetupFTPServer == false) {
	    try {
		    /*
		     This starts a new thread for FTP-as-Server-Handling
		        This allows for the GUI to act separately from the
		        actions of the FTP part of the client
		       */
		       hostServer = new HostServer();
		       hostFTPWelcomeport = hostServer.getFTPWelcomePort();

		       /* For debugging */
```

```
                                // System.out.println("  DEBUG-05: WelcomePort: " +
                                //    hostFTPWelcomeport);

                                /* Initiate the thread */
                                hostServer.start();

                                /*
                                Next "Connect" click will now NOT try this
                                again. This important so that a multitude of
                                actions aren't being opened on the same port.
                                */
                                alreadySetupFTPServer = true;

                                /* For debugging */
                             // System.out.println("DEBUG-01: Initializing Host as FTP Server!");

                            } catch(Exception f) {
                                   System.out.println("  ERROR-03: Failure to setup" +
                                    " client as a FTP-Server.");
                            }
                        }
                        else {
                            System.out.println("  DEBUG-03: Did not attempt FTP-Server setup.");
                        }

                           /* Initialize connection to the Central-Server! */
                           if (!username.equals("") && isConnectedToCentralServer == false) {
                                /*
                                 This internally handles the user clicking "connect"
                                 multiple times.
                                */
                                try {
                                    /* NOV17 - This may need to behave differently to validate
usernames */
                                    int report = host.connectToServer(serverHostname, port, username,
hostname, speed, hostFTPWelcomeport);


                                    if (report == -2 || report == -3 || report == -4) {
                                        /* Taken Username and other errors */
                                        isConnectedToCentralServer = false;
                                    }
                                    else if(report == 1) {
                                        isConnectedToCentralServer = true;
                                    }
                                    else {
                                        /* For debugging */
                                        System.out.println("  ERROR: General failure in connecting to
Central-Server.");

                                        isConnectedToCentralServer = false;
                                    }
                                }
                                catch (Exception g) {
                                    System.out.println("  ERROR-03: Issue connecting to Central-
Server!");
                                    isConnectedToCentralServer = false;
                                }
                            }
                            else {
                                /* For debugging */
                                System.out.println("  DEBUG-07: Did not attempt CS-Connection.");
                            }



                    }
                });
                btnConnect.setFont(new Font("Dialog", Font.BOLD | Font.ITALIC, 12));
                btnConnect.setBounds(254, 66, 149, 19);
                panelConnection.add(btnConnect);
```

```
                textFieldServerHostname = new JTextField();
                textFieldServerHostname.setText("127.0.0.1");
                textFieldServerHostname.setBounds(130, 10, 114, 19);
                panelConnection.add(textFieldServerHostname);
                textFieldServerHostname.setColumns(10);

                textFieldUsername = new JTextField();
                textFieldUsername.setText("");
                textFieldUsername.setBounds(84, 39, 152, 19);
                panelConnection.add(textFieldUsername);
                textFieldUsername.setColumns(10);

                textFieldHostname = new JTextField();
                textFieldHostname.setText("DCCLIENT");
                textFieldHostname.setBounds(84, 66, 158, 19);
                panelConnection.add(textFieldHostname);
                textFieldHostname.setColumns(10);

                textFieldPort = new JTextField();
                textFieldPort.setText("1234");
                textFieldPort.setBounds(301, 10, 102, 19);
                panelConnection.add(textFieldPort);
                textFieldPort.setColumns(10);

                comboBoxSpeed = new JComboBox();
                comboBoxSpeed.setModel(new DefaultComboBoxModel(new String[] {"Ethernet", "Broadband",
"T1", "T3"}));
                comboBoxSpeed.setFont(new Font("Dialog", Font.PLAIN, 12));
                comboBoxSpeed.setBounds(301, 37, 99, 19);
                panelConnection.add(comboBoxSpeed);

                JButton btnDisconnect = new JButton("Disconnect");
                btnDisconnect.setBounds(254, 91, 149, 19);
                panelConnection.add(btnDisconnect);
                btnDisconnect.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                            /*
                             Actions to happen on click of "Disconnect"
                            */
                            try {
                                host.disconnectFromServer();

                                /* Reset the boolean for connectedness */
                                isConnectedToCentralServer = false;
                            }
                            catch (Exception h) {
                                System.out.println("  ERROR: Failure in disconnect()...");

                            }
                        }
                });

                JPanel panelSearch = new JPanel();
                panelSearch.setBorder(new LineBorder(new Color(0, 0, 0), 1, true));
                panelSearch.setBounds(12, 212, 415, 162);
                frame.getContentPane().add(panelSearch);
                panelSearch.setLayout(null);

                TextArea keywordSearchArea = new TextArea();
                keywordSearchArea.setEditable(false);
                keywordSearchArea.setBounds(10, 31, 395, 121);
                panelSearch.add(keywordSearchArea);

                JLabel lblKeyword = new JLabel("Keyword:");
                lblKeyword.setBounds(12, 10, 63, 15);
                lblKeyword.setFont(new Font("Dialog", Font.PLAIN, 12));
                panelSearch.add(lblKeyword);

                keywordSearchField = new JTextField();
                keywordSearchField.setBounds(68, 6, 233, 19);
                panelSearch.add(keywordSearchField);
                keywordSearchField.setColumns(10);
```

```
JButton btnSearch = new JButton("Search");
btnSearch.addActionListener(new ActionListener() {

	/* The following occurs upon click of "Search" */
	public void actionPerformed(ActionEvent e) {

	    String keySearch = keywordSearchField.getText();

	    if(!keySearch.equals("")) {

	        if (alreadySetupFTPServer == true &&
	            isConnectedToCentralServer == true) {

	            /* This may hold values to paste to GUI boxes */
	            String results;

	            /* Query the server for the User's string of keywords */
	            String returnedFileData = host.queryKeywords(keySearch);

	               /* For debugging */
	               keywordSearchArea.setText( returnedFileData );

	               /* For initial textarea */
	               keywordSearchArea.setText("");

	               String stringForTextArea = "";

	               /* This contains all the FILES as tokens to be parsed */
	               StringTokenizer tokens = new StringTokenizer(returnedFileData);

	               /*
	               If there was no files == "NOFOUNDMATCHES"
	               If there was files == "FILE"
	               If there was error == "ERROR"
	               */
	               String firstToken = "";
	               try {
	                  firstToken = tokens.nextToken();
	               }
	               catch (Exception g) {
	                  System.out.println("  DEBUG: First token threw error!");
	               }

	               String currentToken;
	               if (firstToken.equals("ERROR")) {
	                  System.out.println("  DEBUG-10: Error in parsing the returned
string!");

	                  /* Print in GUI "There was an error, try again..." */
	                  keywordSearchArea.setText("There was an error...");
	               }
	               else if (firstToken.equals("NOFOUNDMATCHES")) {

	                  /* For debugging */
	                  // System.out.println("  DEBUG: No found matches!");

	                  /* Print in GUI "no matches" */
	                  keywordSearchArea.setText("No Found Matches...");
	               }
	               else if (firstToken.equals("FILE")) {
	                      /*
	                      For each filedescription, for each row, on the
	                      GUI. On each iteration in the while(), these
	                      values change.
	                      */
	                      String tempUserPort = "";
	                      String tempFileName = "";
	                      String tempHostName = "";
	                      String tempUserName = "";
	                      String tempUserIP = "";
	                      String tempSpeed = "";
	                      String notUsed = "";
```

```
                                        /* Could be a TRY CATCH problem */
                                        try {
                                            while (tokens.hasMoreTokens()) {
                                               // "Alice"
                                               tempUserName = tokens.nextToken();

                                               // "127.1.1.2"
                                               tempUserIP = tokens.nextToken();

                                               // "1235"
                                               tempUserPort = tokens.nextToken();

                                               // "Apples.jpg"
                                               tempFileName = tokens.nextToken();

                                               // "Ethernet"
                                               tempSpeed = tokens.nextToken();

                                               /* Bandaid of an issue in the GUI */
                                               tempSpeed
                                                = tempSpeed.replaceAll("FILE","");

                                               stringForTextArea = stringForTextArea
                                                   + "USERNAME: "
                                                   + tempUserName + "\nIP ADDRESS: "
                                                   + tempUserIP + " " + tempUserPort
                                                   + "\nFILE NAME: " + tempFileName
                                                   + "\nSPEED: " + tempSpeed + "\n\n";

                                               // FILE - NOT USED, maybe should be...
                                               // notUsed = tokens.nextToken();

                                            }
                                        }
                                        catch (Exception h) {
                                            System.out.println("  ERROR: while() threw an
error!");
                                        }

                                        stringForTextArea = stringForTextArea +
                                                            "End of search list.";
                                        keywordSearchArea.setText(stringForTextArea);
                                    }

                                /* End of if-server-is-connected */
                            }
                            else {

                                /* For debugging */
                                // System.out.println("  DEBUG-04: User not " +
                                //              "connected to Central-Server!");
                            }

                        /* End of if-user-typed-anything */
                        }
                        else {

                            /* For debugging */
                            // System.out.println("  DEBUG: No current keyword. " +
                            //                 "Did not send request!");
                        }

                    keywordSearchField.setText("");

                    }
                });

                    btnSearch.setBounds(313, 6, 90, 19);
                    btnSearch.setFont(new Font("Dialog", Font.BOLD | Font.ITALIC, 12));
                    panelSearch.add(btnSearch);
```

```
JPanel panelFTP = new JPanel();
panelFTP.setBorder(new LineBorder(new Color(0, 0, 0), 1, true));
panelFTP.setBounds(12, 405, 415, 181);
frame.getContentPane().add(panelFTP);
panelFTP.setLayout(null);

TextArea ftpTextArea = new TextArea();
ftpTextArea.setBounds(10, 33, 395, 137);
panelFTP.add(ftpTextArea);
ftpTextArea.setEditable(false);

JLabel lblEnterCommand = new JLabel("Enter Command:");
lblEnterCommand.setFont(new Font("Dialog", Font.PLAIN, 12));
lblEnterCommand.setBounds(12, 12, 125, 15);
panelFTP.add(lblEnterCommand);

ftpCommandField = new JTextField();
ftpCommandField.setBounds(119, 10, 225, 19);
panelFTP.add(ftpCommandField);
ftpCommandField.setColumns(10);

JButton btnGo = new JButton("Go");
btnGo.addActionListener(new ActionListener() {

	/* These actions are performed on the click of "Go" */
	public void actionPerformed(ActionEvent e) {

		String toAdd = "";

		if (alreadySetupFTPServer == true
		    && isConnectedToCentralServer == true) {

			String commandLine = ftpCommandField.getText();
			commandHistory = commandHistory + ">>> "
			 + commandLine + "\n";
			ftpTextArea.setText(commandHistory);

			StringTokenizer cmdTokens
			 = new StringTokenizer(commandLine);
			String firstToken = cmdTokens.nextToken();
			firstToken = firstToken.toLowerCase();

			if(!firstToken.equals("")) {
				if (firstToken.equals("connect")){

					/* For debugging */
					// System.out.println("  DEBUG: Inside "
					//  + "connect function.");

					/* Grabbing the data from the typing */
					try {
						ipAddress = cmdTokens.nextToken();
						portNum = cmdTokens.nextToken();

						toAdd = "Connecting to " + ipAddress +
						    " on port " + portNum + "\n";
					}
					catch (Exception q) {
						System.out.println("ERROR: Improper "
						 + "or invalid arguments!");
						toAdd = "ERROR: Improper or invalid "
						 + "arguments!\n";
					}

					/* Connect to other user's HostServer */
					try {
						connect(ipAddress, portNum);
						toAdd = "Connected to " + ipAddress
						+ " on port " + portNum + "!\n";

					}
					catch (Exception q) {
```

```
                                        System.out.println("ERROR: Failed to "
                                        + "connect to server!");
                                        toAdd = "ERROR: Failed to connect"
                                        + " to server!\n";
                                }

                        }
                        else if (firstToken.equals("retr")) {

                                /* For debugging */
                                // System.out.println("  DEBUG: Inside retr
function.");

                                try {
                                        String fileName = cmdTokens.nextToken();

        /** The "Welcome socket" for the data-link */
ServerSocket dataListen;

        /* This socket handles data-links with
        the server */
Socket dataConnection = null;

/* This value handles the file transfer */
int recvMsgSize;

        /* For sending and retrieving file */
byte[] byteBuffer = new byte[32768];

        /** The port number to send files across */
        int dataPort = 1240;

                                try {
    /* Send the request over the control line */
    String toSend = "RETR" + " " + dataPort + " " + fileName;


    outToHost.println(toSend);
    outToHost.flush();

    /* Connect to server and establish variables */
    dataListen = new ServerSocket(dataPort);


    dataConnection = dataListen.accept();
    InputStream inFromServer_Data =
        dataConnection.getInputStream();



    /* Below this handles sending the file itself */
    String fileRequestedString =
        new String(fileName.getBytes());
    while ((recvMsgSize =
            inFromServer_Data.read(byteBuffer)) != -1) {
        try {
            File fileRequested =
                new File(fileRequestedString);
            FileOutputStream fileOutputStream =
                new FileOutputStream(fileRequested);

            fileOutputStream.write(byteBuffer,
                                        0, recvMsgSize);
            fileOutputStream.close();

            Scanner scanner = new Scanner(fileRequested);
            String errorCheck = scanner.nextLine();
            if(errorCheck.equals(
                                "File does not exist.")) {
                System.out.println(
                  "File does not exist.");
                fileRequested.delete();
```

```
                                        } else {
                                            System.out.println("  File retrieved!");
                                        }
                                    } catch (Exception f) {
                                        System.out.println("Error trying to " +
                                         "retrieve file.");
                                    }
                                }

                                /* Close the data connection */
                                try {
                                    dataListen.close();
                                    dataConnection.close();
                                    toAdd = "File " + fileName + " retrieved!\n";
                                } catch (Exception h) {
                                    System.out.println("ERROR: Could not close data" +
                                        " connection");
                                }

                            } catch (Exception g) {
                                System.out.println("ERROR: Failure in " +
                                    "file retrieval.");
                            }



                                        }
                                        catch (Exception q) {
                                                System.out.println("ERROR: Did not give " +
                                                    "arguement to RETR.");
                                                toAdd = "ERROR: Did not give " +
                                                    "arguement to RETR.\n";
                                        }

                                        // Attempt to retrieve file from selected user.

                                    }
                                    else if (firstToken.equals("quit")){

                                        /* For debugging */
                                        // System.out.println("  DEBUG: Inside quit
function.");


                                        if (isConnectedToOtherHost != false){
                                                disconnect();
                                                toAdd = "Disconnected from " + ipAddress +
".\n";
                                                System.out.print(toAdd);
                                        }else {
                                                toAdd = "Not connected to a host.\n";
                                                System.out.println("Not connected to
host.");
                                        }
                                    }else {
                                            toAdd = "Invalid command!\n";
                                    }
                                }
                            }else {
                                    String commandLine = ftpCommandField.getText();
                                    commandHistory = commandHistory + ">>> " + commandLine + "\n";
                                    ftpTextArea.setText(commandHistory);

                                    toAdd = "Not connected to server!\n";
                            }
                            commandHistory = commandHistory + toAdd;
                            ftpTextArea.setText(commandHistory);
                            ftpCommandField.setText("");
                        }
                });
                btnGo.setFont(new Font("Dialog", Font.BOLD | Font.ITALIC, 12));
                btnGo.setBounds(351, 10, 54, 17);
```

```
        panelFTP.add(btnGo);

        JLabel lblConnection = new JLabel("Connection");
        lblConnection.setBounds(12, 12, 80, 15);
        frame.getContentPane().add(lblConnection);

        JLabel lblSearch = new JLabel("Search");
        lblSearch.setBounds(12, 193, 70, 15);
        frame.getContentPane().add(lblSearch);

        JLabel lblFtp = new JLabel("FTP");
        lblFtp.setBounds(12, 386, 70, 15);
        frame.getContentPane().add(lblFtp);

        JPanel panel = new JPanel();
        panel.setBorder(new LineBorder(new Color(0, 0, 0)));
        panel.setBounds(12, 154, 415, 34);
        frame.getContentPane().add(panel);
        panel.setLayout(null);

        JTextArea errorTextArea = new JTextArea();
        errorTextArea.setFont(new Font("Dialog", Font.BOLD | Font.ITALIC, 12));
        errorTextArea.setBackground(Color.WHITE);
        errorTextArea.setBounds(12, 4, 400, 22);
        errorTextArea.setEditable(false);
        errorTextArea.setText("Error Messages will show here");
        panel.add(errorTextArea);
    }

    /**********************************************************************
    * Connect is intended to set up a connection between host A and host B.
    **********************************************************************/
    private void connect(String ipAddress, String portNum){

         /* Connect to server's welcome socket */
         try {
             controlSocket = new Socket(ipAddress,
                                 Integer.parseInt(portNum));
             boolean controlSocketOpen = true;
         }catch(Exception p){
             System.out.println("ERROR: Did not find socket!");
         }

         // Set-up the control-stream,
         // if there's an error, report the non-connection.
         try {
             inFromHost =
                new Scanner(controlSocket.getInputStream());
             outToHost =
                new PrintWriter(controlSocket.getOutputStream());
             isConnectedToOtherHost = true;
             System.out.println("Connected to client!");
             System.out.println(" ");
         }
         catch (Exception e) {
             System.out.println("ERROR: Did not connect to " +
                 "client!");
             isConnectedToOtherHost = false;
        }
    }

    private void disconnect(){
         /* Disconnect from server's welcome socket */
        outToHost.println("quit");
        outToHost.flush();

        boolean controlSocketOpen = false;
         inFromHost.close();
         outToHost.close();
        isConnectedToOtherHost = false;
    }
}
```

Code Samples – Host.java

```
import java.net.*;
import java.util.*;
import java.io.*;
import java.awt.*;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;

/**********************************************************************
This class handles the main client-side functions that interact with
the Central-Server


**********************************************************************/

public class Host {

        /* To handle connecting to Central-Server */
        static Socket server;

        /* Central-Server's IP */
        private String serverHostname;

        /* Central-Server's Port */
        private String port;

        /* User's username */
        private String username;

        /* User's name of computer... and IP */
        private String hostname;

        /* User's submitted internet speed */
        private String speed;

        /* User's FTP port for others to connect to for downloading */
        private String hostFTPWelcomePort;

    /* This handles the control-line out stream */
    PrintWriter outToServer_Control = null;

    /* This handles the control-line input stream */
    Scanner inFromServer_Control = null;

        /* Checks for the user being already connected */
    private boolean isConnected = false;

        @SuppressWarnings("resource")
    public static void main(String[] args) throws Exception {}

        public void disconnectFromServer() {

            outToServer_Control.println("DISCONNECT");
         outToServer_Control.flush();

            isConnected = false;
        }

        /**
         * This method, intiated by the GUI, handles connecting to the
         * Central-Server.
         *
         * @param serverHostname
         * @param port
         * @param username
         * @param hostname
```

```
 */
public int connectToServer( String serverHostname, String port,
                            String username, String hostname,
                            String speed,
                            String hostFTPWelcomePort) {

       /* Taking the parameters from the GUI */
       this.serverHostname = serverHostname;
       this.port = port;
       this.username = username;
       this.hostname = hostname;
       this.speed = speed;
       this.hostFTPWelcomePort = hostFTPWelcomePort;

    /* This socket handles communicating main commands to server */
    Socket controlSocket = null;

    /* To catch if there was errors. Limits sending. */
    boolean hadConnectErrors = false;

    /* Only DO connect if NOT already connected */
    if (isConnected == false) {
       /* Establish a TCP connection with the server */
           try {
               controlSocket = new Socket(serverHostname,
                                   Integer.parseInt(port));
               boolean controlSocketOpen = true;
           } catch (Exception p) {
               System.out.println("  ERROR: Did not find socket!");
               hadConnectErrors = true;
           }

           /* Set-up the control-stream. Handle errors. */
           try {
               inFromServer_Control =
                  new Scanner(controlSocket.getInputStream());
               outToServer_Control =
                  new PrintWriter(controlSocket.getOutputStream());
               isConnected = true;

               /* For debugging */
               // System.out.println("  DEBUG: Connection stream intiated!");
               // System.out.println(" ");
           }
           catch (Exception e) {
               System.out.println("  ERROR: Did not connect to " +
                   "server!");
               hadConnectErrors = true;
               isConnected = false;
           }

           /*
            The following prepares the string "totalDescription" to be
            sent over the wire. This string contains all the keywords and
            filenames of the files included within the client's XML
            document that should be within their source folder.

            TODO: This CAN validate that each of those files exist in
            the directory before sending any list to the Central-Server.
            There could be a case that the XML document has misspellings
            and causes issue with retrieval of files by another host.

            NOTE: The following website was a great help to this function
            of parsing from XML.
            https://www.mkyong.com/java/how-to-read-xml-file-in-java-dom-parser/
           */
              String allFilenamesAndKeys = "";
           boolean errorReadingXML = false;
           try {
              /* Prepare the document for XML extraction */
                  File fXmlFile = new File("HostedFiles.xml");
                  DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
```

```
                        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
                        Document doc = dBuilder.parse(fXmlFile);

                           //optional, but recommended
                        //read this - http://stackoverflow.com/questions/13786607/normalization-in-
dom-parsing-with-java-how-does-it-work
                        doc.getDocumentElement().normalize();

                        NodeList nList = doc.getElementsByTagName("File");

                        /* For each "object," or, each file-description iterate */
                        /* Each iteration would send a string to the central-server */
                        /* TO-DO: (1) Look over this. (2) Implement the server listening for these
strings. */


                /* Loop through each "<File id='0001'>"... */
                for (int temp = 0; temp < nList.getLength(); temp++) {

                            Node nNode = nList.item(temp);
                                   if (nNode.getNodeType() == Node.ELEMENT_NODE) {

                             /* The target element. Ex: "<File id='0001'>" */
                                         Element eElement = (Element) nNode;

                                         /*
                                          NOTE - If later we want to include the
                                          display-name of the file...

                                          String DisplayName =
                                           eElement.getElementsByTagName("DisplayName").item(0);
                                         */

                                         /* Grab the filename */
                                         String FileName =
eElement.getElementsByTagName("FileName").item(0).getTextContent();
                                         FileName = FileName.replaceAll(" ", "");
                                         String KeyWords = "";

                                         /* How many keys are there? */
                                         int tmpLength =
eElement.getElementsByTagName("Key").getLength();

                                         /* For debugging */
                                         // System.out.println("  DEBUG: Filename: " + FileName);
                                       // System.out.println("  DEBUG: Number of keys: " +
tmpLength);

                                         /* Wrap all the keys to one string */
                                         String tempKey;
                                         String totalKeysForCurrentFile = "";
                                         for(int i = 0; i < tmpLength; i++) {
                                             tempKey =
eElement.getElementsByTagName("Key").item(i).getTextContent();
                                             totalKeysForCurrentFile = totalKeysForCurrentFile + " "
+ tempKey;
                                             /* For debugging */
                                             // System.out.println("  DEBUG: tempKey: " + tempKey);
                                         }

                                         /* For debugging */
                                         // System.out.println("  DEBUG: File's Keys: " +
totalKeysForCurrentFile);
                        // System.out.println(" ");

                        /* Attach those keys to the whole sending-string */
                        allFilenamesAndKeys = allFilenamesAndKeys + "FILENAME " + FileName + " " +
totalKeysForCurrentFile + " ";
                                   }
                            }

                  } catch (Exception e) {
```

```
        System.out.println("  ERROR: Error in reading XML " +
                           "document!");
        hadConnectErrors = true;
        errorReadingXML = true;
        return (-6);
    }

    /*
     If there are no errors to this point and the client is
     connected to the Central-Server, then send data.
     */
    if (isConnected == true && hadConnectErrors == false) {
        /*
         Send userName, HostName, speed, and each filename and
         its associated keys to the Central-Server
         */
        String newUserInformation = username + " " + hostname + " "
                                    + speed + " " + hostFTPWelcomePort + " "
                                    + allFilenamesAndKeys;

        /* For debugging. View the sent filenames and keys. */
        // System.out.print("  DEBUG: Sent String To Connect: "
        //                  + newUserInformation);

        outToServer_Control.println(newUserInformation);
        outToServer_Control.flush();

        String recvMsg;
        try {
            /* Wait for response from server */
            recvMsg = inFromServer_Control.nextLine();

            if (recvMsg.equals("GOOD-USERNAME")) {

                /* For debugging */
         //System.out.println("  DEBUG: Good username!");

         isConnected = true;
                return (1);
            }
            else if (recvMsg.equals("BAD-USERNAME")) {

                /* For debugging */
         // System.out.println("  DEBUG: Bad username!");

         isConnected = false;
                return (-2);
            }
            else {

                /* For debugging */
         // System.out.println("  DEBUG: Corrupted response from server!");
         isConnected = false;
                return (-3);
            }
}
catch (Exception e) {

    /* For debugging */
    // System.out.println("  DEBUG: Failed response from server!");

    isConnected = false;
    return (-4);
}

        /* TODO -
         Wait for response from server to validate that THAT
         username is valid. Username may be the best unique
         identifiers for end-users. It may also help in backend.
        */

    }
```

```
                else if (isConnected == true && hadConnectErrors == true && errorReadingXML == true)
{
                    /* Send command to end thread because of client issues. */

                    /* Which error message to send */
                    String sendErrorMessage = "XML-READ-ERROR";

                    /* For debugging. View the sent filenames and keys. */
                    System.out.print("  DEBUG: Sent String: "
                                    + sendErrorMessage);

                    /* Send the error message */
                    outToServer_Control.println(sendErrorMessage);
                    outToServer_Control.flush();

                    /* Reset the connected status to later reconnect */
                    isConnected = false;
                }
                else {
                    /*
                     Return to user that there was an error in connecting.
                     This could be by the GUI. It could be several options of
                     issue: (1) an already taken username, (2) no XML document,
                     (3) a missing file that was called by the XML document,
                     (4) no connection to available to the called IP of
                     Central-Server, (5) refused connection by the "CS" at
                     application-level.
                    */

                    /* For debugging */
                    System.out.println("  DEBUG: Error in connecting...");
                    return (-5);
                }

            /* Ends of if(isConnected == false) */
        }
        else {
            /* For debugging */
                System.out.println("  DEBUG-04: Already connected to Central" +
                "-Server!");
                return (-7);
        }

        return (0);
    /* End of connectToServer() */
    }

/* This allows the GUI to send a keyword search to the Central-Server */
public String queryKeywords(String keySearch) {

    /* Prepare the command for the central-server */
    String toSend = "KEYWORD" + " " + keySearch;

    /* Send the query to the server! */
    outToServer_Control.println(toSend);
   outToServer_Control.flush();

   /* Listen for Server Response */
   /* This would have all the files that have the key involved */
   try {
          String recvMsg;
          recvMsg = inFromServer_Control.nextLine();

          /* For debugging */
          // System.out.println("  DEBUG: Received message: " + recvMsg);

          return (recvMsg);

   }catch (Exception e) {
          System.out.println("  DEBUG: Connection broke with server while waiting for response.");
          return ("ERROR");
   }
```

```
        /* For debugging */
         // System.out.println("  DEBUG: Sending: " + toSend);
         // System.out.println("  DEBUG: End of queryKeywords()");

    /* End of queryKeywords() */
    }


/* End of FTP client */
}
```

Code Sample – HostedDescriptions.java

```
import java.util.ArrayList;
/* For tokens */
import java.util.*;

public class HostedDescriptions {

        private ArrayList<String> speeds;
        private ArrayList<String> hostnames;
        private ArrayList<String> filenames;
        private ArrayList<String> keywords;
    private ArrayList<String> usernames;
    private ArrayList<String> userIPs;
    private ArrayList<String> userPorts;

        public HostedDescriptions() {
            userIPs = new ArrayList<String>();
            userPorts = new ArrayList<String>();
            usernames = new ArrayList<String>();
                speeds = new ArrayList<String>();
                hostnames = new ArrayList<String>();
                filenames = new ArrayList<String>();
                keywords = new ArrayList<String>();
        }

        public void addValues (String addThisToSpeeds, String addThisToHostname, String
addThisToFilename, String addThisToKeywords, String addThisToUsernames, String addThisToUserIPs, String
addThisToUserPorts) {
                this.speeds.add(addThisToSpeeds);
                this.hostnames.add(addThisToHostname);
                this.filenames.add(addThisToFilename);
                this.keywords.add(addThisToKeywords);
                this.usernames.add(addThisToUsernames);
                this.userIPs.add(addThisToUserIPs);
                this.userPorts.add(addThisToUserPorts);
        }

        /* Removes all elements that belong to the client with a specific username */
        public void remove (String username) {
                for(int i = 0; i < usernames.size(); i++) {
                        if(usernames.get(i).equals(username)) {
                            usernames.remove(i);
                                hostnames.remove(i);
                                speeds.remove(i);
                                filenames.remove(i);
                                userIPs.remove(i);
                                keywords.remove(i);
                                userPorts.remove(i);
                                i--;
                        }
                }
        }

        /* Returns true if the username is already taken */
        public boolean isUsernameTaken(String username) {
            if (usernames.contains(username)) {
                 return true;
```

```
            }
            else {
                return false;
            }
        }

        /* Displays some information about the current state of the main tables */
        public void showData() {
            ArrayList<String> variousUsernames = new ArrayList<String>();
            int numberOfUsers;
            try {
             variousUsernames.add( usernames.get(0) );
             for(int i = 0; i < usernames.size(); i++) {
                 if (!variousUsernames.contains( usernames.get(i) )) {
                     variousUsernames.add( usernames.get(i) );
                 }
                }
                numberOfUsers = variousUsernames.size();
            }
            catch (Exception e) {
                numberOfUsers = 0;
            }

            System.out.println("Showing Current Server Data...");
            System.out.print(" Total Current Users: " + numberOfUsers);
            if (numberOfUsers > 1) {
                System.out.print(" [");
                for(int j = 0; j < variousUsernames.size() - 1; j++) {
                    System.out.print( variousUsernames.get(j) + ", ");
                }
                System.out.print( variousUsernames.get(numberOfUsers - 1) + "]");
            }
            System.out.print("\n");

            if (numberOfUsers != 0) {
                System.out.println(" Total Files Represented: " + usernames.size() );
             System.out.print(" List of Files: \n");
             System.out.println(" [Username, IP-Address, FTP-Port-Num, Speed, Filename]");
             for(int k = 0; k < filenames.size(); k++) {
                 System.out.print(" [" + usernames.get(k) + ", " + userIPs.get(k) + ", " +
userPorts.get(k) + ", " + speeds.get(k) + ", " + filenames.get(k) + "] \n");
             }
         }

    /* End of showData() */
        }

        public String getKeywordData(String keywordsList) {
               /* keywordsLIST = "image apple banana tom cruise"

               /* Construct the thing to return */
               /* Ex: "FILE username userIP userPort filename speed */
               String stringToReturn = "";



               StringTokenizer tokens = new StringTokenizer(keywordsList);

               String grabbedFiles = "";

               while (tokens.hasMoreTokens()) {
                      try {
                             /* currentToken equals one key from the client's string */
                             /* Ex: "apple" */
                             String currentToken = tokens.nextToken();

                             /* Loops through all the files in search for a keyword */
                             for(int i= 0; i < keywords.size(); i++) {

                                 /* If a file has this keeyword inside */
                                 if ( keywords.get(i).contains(currentToken)
                                        && !grabbedFiles.contains(filenames.get(i) + usernames.get(i)
```

```
                                        ) ) {
                                            /* To build the string to send back */
                                            /*stringToReturn = stringToReturn + " FILE " +
usernames.get(i) + " " +
                                            userIPs.get(i) + " " + userPorts.get(i) + " " +
                                            filenames.get(i) + " " + speeds.get(i); */

                                            stringToReturn = stringToReturn + "FILE " +
usernames.get(i)
                                            + " " + userIPs.get(i)
                                            + " " + userPorts.get(i)
                                            + " " + filenames.get(i)
                                            + " " + speeds.get(i);

                                            /* Allows for uniqueness of files with users */
                                            String identifier = filenames.get(i) + usernames.get(i);

                                            /* For checking if the file was already appended */
                                            grabbedFiles = grabbedFiles + " " + identifier;



                                            /* For debugging */
                                            // System.out.println("  DEBUG-09: File has "
                                            // + "key (" + currentToken + "): "
                                            // + filenames.get(i) );
                                    }
                                }
                        }catch (Exception e) {
                        System.out.println("  ERROR-08: Caught nextToken error!");
                        }
                }

                if (stringToReturn == "") {
                        return ("NOFOUNDMATCHES");
                }
                return stringToReturn;
        }


/* End of class HostedDescriptions */
}
```

Code Samples – HostServer.java

```
import java.net.*;
import java.util.*;
import java.io.*;

/**********************************************************************
 *
 * Host Server
 *
 * This class is a part of the host in this project, being initiated by
 * it. This part of the host, keeps and open connection for other host
 * to connect and request files.
 *
 * @author Javier Ramirez
 * @version November 20, 2017
 *
 **********************************************************************/
public class HostServer extends Thread {

        /** Used in handling incoming messages */
        private String recvMsg;

    /** The port number that accepts new FTP-connects */
        private static int welcomePort = 1235;

        /** The socket that listens for new FTP-connects */
        private static ServerSocket welcomeSocket;
```

```
/** This keeps track of success and failure in connecting */
private boolean successfullySetFTPPort = false;

/*****************************************************************
* Constructor initializes the socket that listens for new FTP
* connections. It also flexibly sets up the port-number based on
* a realization of which ones are available to be used. This allows
* us avoid issues with multiple FTP-hosts. It logs the connection
* having been setup.
*
* @exception EmptyStackException if no port was established to
* listen for new FTP-connections.
*****************************************************************/
public HostServer() {

	/* Show server starting status */
	System.out.println(" ");
	System.out.println("Client-As-FTP-Server initialized. Waiting" +
				" for connections...");
	for(int i = 0; i < 5; i++){

		/* Initialize the welcome socket */
		try {
			welcomeSocket = new ServerSocket(welcomePort);
			successfullySetFTPPort = true;

			/* For debugging */
			// System.out.println("  DEBUG: FTP-Welcome Port: " + welcomePort);

			/* Stop the loop if found a good port */
			break;
		}
		catch (IOException ioEx) {

			/*
			 Setting a port for another host to connect to.
			 This is setup to handle the case of multiple users on
			 one computer. Without this loop and catching, the
			 second user immediately breaks.
			*/
			welcomePort = welcomePort + 1;

			/* For debugging */
			// System.out.println("  DEBUG: Changing the " +
			//                      "port number.");
		}
	}

	/* If unable to setup a port for later FTP connections */
	if (successfullySetFTPPort == false) {

		/* Throw exception to the GUI */
		throw new EmptyStackException();
	}
}

/*****************************************************************
* This is the continuously running part of the class. It welcomes
* new connections, then sets them up to their own thread to allow
* for simultaneous other processes.
*
* @exception Throws an error if there was trouble in setting up a
* new thread for our new connection to be handled within.
*****************************************************************/
public void run(){

	/* Perform a loop to wait for new connections */
	try {
		do {

			/* Wait for client... */
```

```
                        Socket connectionSocket = welcomeSocket.accept();

                /* Display to terminal when new client connects */
                        System.out.println("\nClient-As-FTP-Server: New Client Connected!");

                        /* For debugging */
                        // System.out.println("  DEBUG: New Connection's IP: " +
                        //     connectionSocket.getInetAddress());

                        /*
                         Create a thread to handle communication with this
                         client and pass the constructor for this thread a
                         reference to the relevant socket and user IP.
                        */
                        FTPClientHandler handler
                                        = new FTPClientHandler(connectionSocket);

                        /* Start a new thread for this client */
                        handler.start();
                    } while (true);
                }
                catch (Exception e) {
                        System.out.println("ERROR: Failure in setting up a " +
                                        "new thread.");
                }
        }

        /**********************************************************************
         * Once the host-FTP-server is setup, we know it's welcomePort. THEN
         * this needs to be passed to the Central-Server to allow others to
         * download files from this host. This function helps in this
         * process.
         * @return String - the string of the port number that this host is
         * listening on.
         **********************************************************************/
        public String getFTPWelcomePort() {
                return Integer.toString(welcomePort);
        }

/* End of Entire HostServer Class */
}

/**************************************************************************
 * This class handles allows for threading and handling the various
 * clients that are connected to server simultaneously.
 **************************************************************************/
class FTPClientHandler extends Thread {

        /** This sets the packet size to be sent across to this size */
        private static final int BUFSIZE = 32768;

        /** Port for the commands to be sent across */
        private static final int controlPort = 1078;

        /** This socket takes commands from client */
        private Socket controlListen;

        /** This socket takes data from client */
        private Socket dataSocket;

        /** This handles the stream from the command-line of client */
        private Scanner inScanFromClient;

        /** This is used for handling the buffering of files over /
            the data stream */
        private int recvMsgSize;

        /** This is used to grab bytes over the data-line */
        private String recvMsg;

        /** This allows for identification of specific users in streams */
        private String remoteIP;
```

```
/** Output Stream for Control */
private OutputStream outToClient;

/*************************************************************
*
* Beginning of thread.
* This constructor marks the beginning of a thread on the server.
* Things here happen once, exclusively with THIS connected client.
*
*************************************************************/
public FTPClientHandler(Socket controlListen) {
    try {
            /* Setting up a threaded input control-stream */
            inScanFromClient = new Scanner (
                controlListen.getInputStream());
            /* Setting up a threaded output control-stream */
            outToClient = controlListen.getOutputStream();
            /* Get IP from the control socket for future connections */
            remoteIP = controlListen.getInetAddress().getHostAddress();
            /* For debugging */
            // System.out.println("  DEBUG: A new thread was successfully setup.");
            // System.out.println("");
    } catch(IOException ioEx) {
            ioEx.printStackTrace();
            System.out.println("ERROR: Could not set up a " +
                       "threaded client input and output stream.");
    }
}

/*************************************************************
*
* Beginning of main thread code.
* This method marks the threaded area that this client receives
* commands and handles. When this receives "QUIT" from the client,
* the thread closes.
*
*************************************************************/
public void run(){

        /* For sending and retrieving file */
        int BUFSIZE = 32768;
        byte[] byteBuffer = new byte[BUFSIZE];

        Socket dataConnection;
        boolean stayAlive = true;

        while (stayAlive) {
                /* Try to get input from client */
                try {
                        recvMsg = inScanFromClient.nextLine();

                        /* For debugging */
                        // System.out.println("RECEIVED MESSAGE: " + recvMsg);
                } catch (Exception e) {
                        System.out.println("");

                        /* For debugging */
                        // System.out.println("NO INPUT FROM CLIENT");

                        break;
                }

                /* Set up a tokenizer */
                StringTokenizer tokens = new StringTokenizer(recvMsg);
                String commandToken = tokens.nextToken();

                if(commandToken.toLowerCase().equals("retr")) {

                        /*
                         From the string read from the connection with the other
                         host...
```

```
            (1) Grab the dataport to connect back to.
            (2) Grab the filename to retrieve.
           */
           String dataPort = tokens.nextToken();
           String fileName = tokens.nextToken();

           InputStream inFromClient_Data;
           OutputStream outToClient_Data;

           try {

                  /* Data connection socket */
                  dataConnection = new Socket(remoteIP,
                  Integer.parseInt(dataPort));

                  /* Initiate data Input/Output streams */
                  inFromClient_Data =
                      dataConnection.getInputStream();
                  outToClient_Data =
                      dataConnection.getOutputStream();
                  System.out.println("Data line started.");

                  /* Handle the sending of file over the line */
                  File myFile = new File(fileName);
                  if (myFile.exists()) {
                         try {
                                /* Declare variables for converting
                                    file to byte[] */
                                FileInputStream fileInputStream =
                                    new FileInputStream(myFile);
                                byte[] fileByteArray =
                                    new byte[(int) myFile.length()];
                                // Grabs file to memory to then be passed
                                fileInputStream.read(fileByteArray);
                                fileInputStream.close();

                                // Write to client over DATA line
                                outToClient_Data.write(fileByteArray);
                                outToClient_Data.flush();
                         } catch (Exception e) {
                                e.printStackTrace();
                         }
                  } else {
                         // Write error message to client over DATA line
                         String errMsg = new String("File does " +
                             "not exist.");

                         try {
                                outToClient.write(errMsg.getBytes());
                                System.out.println("Writing data.");
                         } catch (Exception e) {
                                System.out.println("ERROR: Input/out " +
                                           "stream failure.");
                         }

                         try {
                                outToClient.flush();
                                System.out.println("Data sent.");
                         } catch (Exception e) {
                                System.out.println("ERROR: Input/out " +
                                           "flush failure.");
                         }
                  }

                  /* Close after the data is written to client */
                  dataConnection.close();
           }
           catch (Exception j) {
               System.out.println("  DEBUG: Catch-all error..");
           }
    }
    else if(commandToken.toLowerCase().equals("quit")) {
```

```
                                try {
                                    inScanFromClient = null;
                                    outToClient = null;
                                    remoteIP = "";
                                    System.out.println("Disconnected!");
                                }
                                catch (Exception e) {
                                    System.out.println("  ERROR: Closing connection error");
                                }
                        }
                        else {
                                System.out.println("HOST SERVER: WRONG INPUT");
                        }

                /* End of the other while loop */
                }
        /* End of threading run() */
        }
/* End of the thread class */
}
```