```java
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.StringTokenizer;

public class Centralized_Server {
    // Socket that awaits client connections.
    private static ServerSocket welcomeSocket;

    // Holds all client UserNames that have connected to the server.
    public static ArrayList<ClientHandler> users = new ArrayList<ClientHandler>();
    public static ArrayList<ClientData> clientData = new ArrayList<ClientData>();

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

        try {
            welcomeSocket = new ServerSocket(3158); // ServerPort
            System.out.println("Server UP!");
        } catch (Exception e) {
            System.err.println("ERROR: Server could not be started.");
        }

        try {
            while (true) {

                // Waits for a client to connect.
                Socket connectionSocket = welcomeSocket.accept();

                // Set up input and output stream with the client to send and receive messages.
                BufferedReader dis = new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));
                DataOutputStream dos = new DataOutputStream(connectionSocket.getOutputStream());

                // Creates a clientHandler object with the client.
                ClientHandler client = new ClientHandler(connectionSocket, dis, dos);

                // Adds the client to the arrayList of clients.
                users.add(client);

                // Makes a thread to allow the client and clientHandler to interact.
                Thread t = new Thread(client);
                t.start();
            }

        } catch (Exception e) {
            System.err.println("ERROR: Connecting Client");
            e.printStackTrace();

        } finally {
            try {
                // Close the Socket in the event of an error.
                welcomeSocket.close();
                System.out.println("Server socket closed.");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

    }
}

/******************************************************************************************
```

```
***
 *
 * Handles the client.
 *
 ******************************************************************************************
 **/
class ClientHandler implements Runnable {

    Socket connectionSocket;
    String fromClient;
    String clientName;
    String hostName;
    int port;
    String speed;
    BufferedReader dis;
    DataInputStream is;
    DataOutputStream dos;
    boolean loggedIn;

    /****
     *
     * Sets up the ClientHandler object/
     *
     ****/
    public ClientHandler(Socket connectionSocket, BufferedReader dis, DataOutputStream
    dos) {

        this.connectionSocket = connectionSocket;
        this.dis = dis;
        this.dos = dos;
        this.loggedIn = true;

    }

    /****
     *
     * Allows multiple clients to interact with the server.
     *
     ****/
    @Override
    public void run() {

        String connectionString;
        String fileList;

        int listSize;

        try {

            // Sets the first string received as the UserName, hostName and speed for the
            // client.
            is = new DataInputStream(connectionSocket.getInputStream());
            connectionString = is.readUTF();

            // Client sends a String filled with information about the client.
            StringTokenizer tokens = new StringTokenizer(connectionString);
            this.clientName = tokens.nextToken();
            this.hostName = tokens.nextToken();
            this.speed = tokens.nextToken();
            this.port = Integer.parseInt(tokens.nextToken());

            System.out.println(clientName + " has connected!");

            // Reads in whether or not the client has files available for download.
            fileList = is.readUTF();

            // If the client has no files to offer the fileList will be '505'
            if (!fileList.equals("505")) {
```

```java
                tokens = new StringTokenizer(fileList);
                String data = tokens.nextToken();

                if (data.startsWith("200")) {

                    // Number of files the client has to offer.
                    data = tokens.nextToken();
                    listSize = Integer.parseInt(data);

                    for (int i = 0; i < listSize; i++) {

                        // Read in the first String of file Information.
                        String fileInfo = is.readUTF();

                        tokens = new StringTokenizer(fileInfo);
                        String fileName = tokens.nextToken("$");
                        String fileDescription = tokens.nextToken();

                        // Creates a clientData object with the information about the
                        file.
                        ClientData cd = new ClientData(this.clientName, this.hostName,
                        this.port, fileName,
                                fileDescription, this.speed);
                        Centralized_Server.clientData.add(cd);
                    }
                }
            }

        } catch (IOException e1) {
            e1.printStackTrace();
        }

        try {

            // Do while conditional.
            boolean hasNotQuit = true;

            // Breaks down the messages received by the client into a command.
            do {

                // Waits for data.
                fromClient = is.readUTF();

                if (fromClient.equals("QUIT")) {

                    hasNotQuit = false;

                    // If the message is not a command then it is assumed the client is
                    trying to
                    // send a message.
                } else {

                    for (int i = 0; i < Centralized_Server.clientData.size(); i++) {
                        if
                        (Centralized_Server.clientData.get(i).fileDescription.contains(fr
                        omClient)) {
                            ClientData cd = Centralized_Server.clientData.get(i);
                            String str = cd.speed + " " + cd.hostName + " " + cd.port +
                            " " + cd.fileName + " "
                                    + cd.hostUserName;
                            dos.writeUTF(str);
                            System.out.println(cd.fileName);
                        }
                    }

                    dos.writeUTF("EOF");
                }

            } while (hasNotQuit);

            // Set the online status to offline.
            this.loggedIn = false;

            for (int i = 0; i < Centralized_Server.clientData.size(); i++) {
                if (Centralized_Server.clientData.get(i).hostName == this.hostName) {
                    Centralized_Server.clientData.remove(i);
                }
            }

            // Close the Socket.
            this.connectionSocket.close();
            System.out.println(clientName + " has disconnected!");

        } catch (Exception e) {
            System.err.println(e);
            System.exit(1);
        }
    }
}

/*****************************************************************************************
 *
 * Handles the clients files that are available for download.
 *
 *****************************************************************************************/
class ClientData {

    public String hostName;
    public String hostUserName;
    public String fileName;
    public String fileDescription;
    public String speed;
    public int port;

    /****
     *
     * Holds all the information of the file.
     *
     ****/
    public ClientData(String hostUserName, String hn, int port, String fn, String fd,
    String sp) {
        this.hostUserName = hostUserName;
        this.hostName = hn;
        this.port = port;
        this.fileName = fn;
        this.fileDescription = fd;
        this.speed = sp;
    }
}

```