Alexander Dicharry
Professor Nur
CSCI-4311: Computer Networks & Telecommunications
16 October 2020

Programming Assignment 1

Figure 1:

Client.java

Client.java | Server.java

```java

public class Client {

    static boolean active = true;

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

        // making a new scanner
        Scanner scanner = new Scanner(System.in);

        // making a socket
        Socket socket = new Socket("localhost", 5000);

        DataInputStream dataInputStream = new DataInputStream(socket.getInputStream());
        DataOutputStream dataOutputStream = new DataOutputStream(socket.getOutputStream());

        // output for the Server
        // asking for the username, ackowledging the connection was successful, and welcoming the user
        System.out.println("Enter username: ");
        String username = scanner.nextLine();
        System.out.println("");
        System.out.println("Connection successful " + socket);
        System.out.println("Welcome " + username);

        try {
            dataOutputStream.writeUTF(username);
        } // end of the try

        catch (IOException e) {
        } // end of the catch

        Thread send = new Thread(new Runnable()
        {

            @Override
            public void run() {
                while (active) {
                    try {
                        String msg = scanner.nextLine();
                        if(msg.equals("bye")) {
                            active = false;
                            dataOutputStream.writeUTF("bye");
                            socket.close();
                            return;
                        } // if statement

                        dataOutputStream.writeUTF(msg);
                    } // end of the try
```

Line 31, Column 103 Tab Size: 4

This is the client class. Once this is compiled and ran in the terminal, it won't acknowledge a connection until opening and running the server. After completing the communication, by typing "bye" the program will end.

Figure 2:

Client.java

```java
                    try {
                        String msg = scanner.nextLine();
                        if(msg.equals("bye")) {
                            active = false;
                            dataOutputStream.writeUTF("bye");
                            socket.close();
                            return;
                        } // if statement

                        dataOutputStream.writeUTF(msg);
                    } // end of the try

                    catch (IOException e) {
                    } // end of the catch

                } // while loop

            } // end of override run method in the send thread

        } // end of the send thread
        );

        Thread recieve = new Thread(new Runnable() {
            @Override
            public void run() {
                while(active) {
                    try {
                        String msg = dataInputStream.readUTF();
                        System.out.println(msg);
                    } // end of the try

                    catch (IOException e) {
                    } // end of the catch

                } // end while loop

            } // end of Override run method

        }); // end of the receive thread

        send.start();
        recieve.start();


    } // end main method

} // end class Client
```

Line 31, Column 103 Tab Size: 4

This is to continue from Figure 1. But here is the receiving threads in the client class.

Figure 3:

Server.java

Client.java | Server.java

```java
import java.io.*;

public class Server {
    static ArrayList<ClientHandler> clientList = new ArrayList();

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

        // creating a new server socket
        ServerSocket serverSocket = new ServerSocket(5000);

        // formatting the time
        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("HH:mm:ss");

        Socket socket;

        while(true) {
            socket = serverSocket.accept();

            DataInputStream dataInputStream = new DataInputStream(socket.getInputStream());
            DataOutputStream dataOutputStream = new DataOutputStream(socket.getOutputStream());

            ClientHandler clientHandler = new ClientHandler(socket, dataInputStream, dataOutputStream);


            LocalDateTime now = LocalDateTime.now();

            System.out.println(">>" + dtf.format(now)+ " Welcome: " + clientHandler.username);
            Thread thread = new Thread(clientHandler);

            for(ClientHandler arr: clientList) {
                now = LocalDateTime.now();
                if(arr.active)
                    arr.dataOutputStream.writeUTF(">>" + dtf.format(now)+" Welcome: " + clientHandler.username);
            }

            clientList.add(clientHandler);

            thread.start();
            } // end while loop

        } // end main method

        public static char[] listOfClients(DataInputStream dataInputStream) {
            return null;
        }

    } // end class server
```

Line 11, Column 19 Tab Size: 4 Java

Here is the Server class. In the beginning we are creating and making new server sockets and clientHandlers for input and output streams. We also implement output to the client by welcoming him/her along with formatting the time that each message was sent and/or received by the client and server. The port number is set at 5000. This was used after looking at the “Socket Programming Example” you gave to us in class when first assigned.

Figure 4:

```java
    } // end class server

    // clientHandler implementing Runnable
    class ClientHandler implements Runnable{

        // building a scanner
        Scanner scanner = new Scanner(System.in);

        public String username;
        public Socket socket;

        final DataInputStream dataInputStream;
        final DataOutputStream dataOutputStream;
        final String time;

        // creatinga boolean named active, setting it to true
        boolean active = true;

        // here I create the dating format on how to display the date and time of the responses
        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("HH:mm:ss");
        DateTimeFormatter dtf1 = DateTimeFormatter.ofPattern("EEE, dd MMM  HH:mm:ss 'CST' yyyy", Locale.US);

        // here is the constructor setting up the multiple instantiations, and setting our parameters
        // also throwing an IOException
        public ClientHandler(Socket socket, DataInputStream dataInputStream, DataOutputStream dataOutputStream) thro

            this.dataInputStream = dataInputStream;
            this.dataOutputStream= dataOutputStream;
            this.socket = socket;
            this.username =  dataInputStream.readUTF();
            LocalDateTime now = LocalDateTime.now();
            this.time = dtf1.format(now);
        }

        // getter method to get the time
        public String getTime() {
            return time;
        }

        // getter method to get the username
        public String getUsername() {
            return username;
        }

        public boolean getActive() {
            return this.active;
        }
```

In this photo we have an inner class within the Server class that is implementing Runnable. This is what is running where we ask for the username, time and output stream for the client. As well here, we format the time we show for the messages.

Figure 5:

Server.java

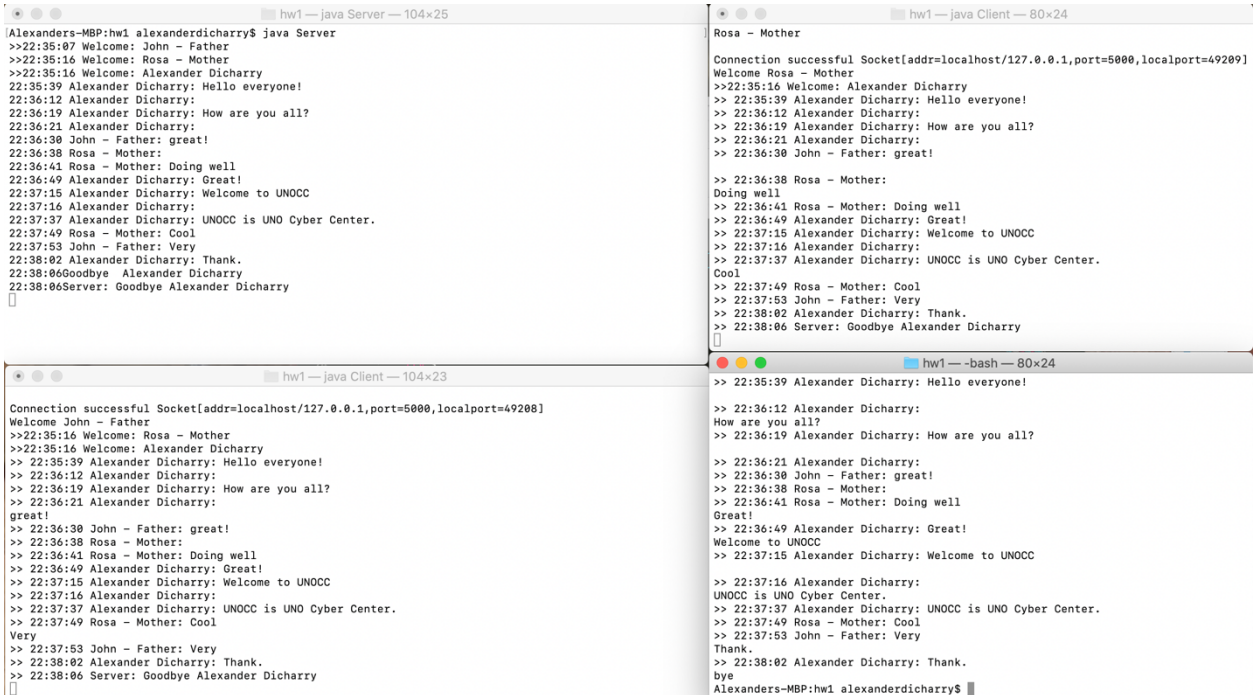
Client.java | Server.java

```java
    // here we override the run method
    @Override
    public void run() {
        String received;
        while(true) {
            try {
                received = dataInputStream.readUTF();
                if(received.equals("bye")) {
                    LocalDateTime now = LocalDateTime.now();
                    System.out.println( dtf.format(now) + "Goodbye  " + username);
                    System.out.println( dtf.format(now) + "Server: Goodbye " + username);

                    active = false;
                    for(ClientHandler clientHandler: Server.clientList) {
                        if( clientHandler.active == true)
                            clientHandler.dataOutputStream.writeUTF(">> " + dtf.format(now) + " Server: Goodbye " +
                        } // for loop

                    break;
                } // end if statement

                else if(received.equals("AllUsers")){

                    LocalDateTime now = LocalDateTime.now();
                    dataOutputStream.writeUTF( "Users: " + dtf.format(now));
                    for(int i = 0 ; i < Server.clientList.size(); i++) {
                        if(Server.clientList.get(i).getActive()) {
                            dataOutputStream.writeUTF(">> " + i+") " + Server.clientList.get(i).getUsername() + " si
                        } // end if-statement

                    } // for-loop

                } // end else

                else {
                    LocalDateTime now = LocalDateTime.now();

                    System.out.println( dtf.format(now) + " "+ username +": "+ received);

                    for(ClientHandler clientHandler: Server.clientList) {
                        if( clientHandler.active == true)
                            clientHandler.dataOutputStream.writeUTF(">> " + dtf.format(now) + " "+ username +": "+ re
                    } // for-loop

                } // end else

            } // end of the try
```

Line 11, Column 19 Tab Size: 4 Java

In this photo, this is where we write out to the user “goodbye” if the you want to end the program and tell the clients that the application is basically over. If the message “bye is received by the client, the server will end the program.

Figure 6:



hw1 — java Server — 104×25

```
Alexanders-MBP:hw1 alexanderdicharry$ java Server
>>22:35:07 Welcome: John - Father
>>22:35:16 Welcome: Rosa - Mother
>>22:35:16 Welcome: Alexander Dicharry
22:35:39 Alexander Dicharry: Hello everyone!
22:36:12 Alexander Dicharry:
22:36:19 Alexander Dicharry: How are you all?
22:36:21 Alexander Dicharry:
22:36:30 John - Father: great!
22:36:38 Rosa - Mother:
22:36:41 Rosa - Mother: Doing well
22:36:49 Alexander Dicharry: Great!
22:37:15 Alexander Dicharry: Welcome to UNOCC
22:37:16 Alexander Dicharry:
22:37:37 Alexander Dicharry: UNOCC is UNO Cyber Center.
22:37:49 Rosa - Mother: Cool
22:37:53 John - Father: Very
22:38:02 Alexander Dicharry: Thank.
22:38:06Goodbye  Alexander Dicharry
22:38:06Server: Goodbye Alexander Dicharry
```

hw1 — java Client — 80×24

```
Rosa - Mother

Connection successful Socket[addr=localhost/127.0.0.1,port=5000,localport=49209]
Welcome Rosa - Mother
>>22:35:16 Welcome: Alexander Dicharry
>> 22:35:39 Alexander Dicharry: Hello everyone!
>> 22:36:12 Alexander Dicharry:
>> 22:36:19 Alexander Dicharry: How are you all?
>> 22:36:21 Alexander Dicharry:
>> 22:36:30 John - Father: great!

>> 22:36:38 Rosa - Mother:
Doing well
>> 22:36:41 Rosa - Mother: Doing well
>> 22:36:49 Alexander Dicharry: Great!
>> 22:37:15 Alexander Dicharry: Welcome to UNOCC
>> 22:37:16 Alexander Dicharry:
>> 22:37:37 Alexander Dicharry: UNOCC is UNO Cyber Center.
Cool
>> 22:37:49 Rosa - Mother: Cool
>> 22:37:53 John - Father: Very
>> 22:38:02 Alexander Dicharry: Thank.
>> 22:38:06 Server: Goodbye Alexander Dicharry
```

hw1 — java Client — 104×23

```
Connection successful Socket[addr=localhost/127.0.0.1,port=5000,localport=49208]
Welcome John - Father
>>22:35:16 Welcome: Rosa - Mother
>>22:35:16 Welcome: Alexander Dicharry
>> 22:35:39 Alexander Dicharry: Hello everyone!
>> 22:36:12 Alexander Dicharry:
>> 22:36:19 Alexander Dicharry: How are you all?
>> 22:36:21 Alexander Dicharry:
great!
>> 22:36:30 John - Father: great!
>> 22:36:38 Rosa - Mother:
>> 22:36:41 Rosa - Mother: Doing well
>> 22:36:49 Alexander Dicharry: Great!
>> 22:37:15 Alexander Dicharry: Welcome to UNOCC
>> 22:37:16 Alexander Dicharry:
>> 22:37:37 Alexander Dicharry: UNOCC is UNO Cyber Center.
>> 22:37:49 Rosa - Mother: Cool
Very
>> 22:37:53 John - Father: Very
>> 22:38:02 Alexander Dicharry: Thank.
>> 22:38:06 Server: Goodbye Alexander Dicharry
```

hw1 — -bash — 80×24

```
>> 22:35:39 Alexander Dicharry: Hello everyone!

>> 22:36:12 Alexander Dicharry:
How are you all?
>> 22:36:19 Alexander Dicharry: How are you all?

>> 22:36:21 Alexander Dicharry:
>> 22:36:30 John - Father: great!
>> 22:36:38 Rosa - Mother:
>> 22:36:41 Rosa - Mother: Doing well
Great!
>> 22:36:49 Alexander Dicharry: Great!
Welcome to UNOCC
>> 22:37:15 Alexander Dicharry: Welcome to UNOCC

>> 22:37:16 Alexander Dicharry:
UNOCC is UNO Cyber Center.
>> 22:37:37 Alexander Dicharry: UNOCC is UNO Cyber Center.
>> 22:37:49 Rosa - Mother: Cool
>> 22:37:53 John - Father: Very
Thank.
>> 22:38:02 Alexander Dicharry: Thank.
bye
Alexanders-MBP:hw1 alexanderdicharry$
```

In comparison to your prompt, I opened multiple terminals for multiple clients and one for the server. I ran one as “Alexander Dicharry”, one as “John-Father”, and one as “Rosa-Mother”. And the terminal in the top left was the server. I tried a couple of messages in each client to communicate with everyone and the output came out in each terminal for each client and server to see the chat. By me saying “bye” in the bottom left terminal, the program ended the application.