Umar Janjua

CS 331 - Computer Networks

05 December 2022

Course Project

Introduction

My course project is to create a terminal based peer to peer chat program using Java. Each device on the network is to act as a client and server. The main goal of my project was to have the two programs run on separate devices. Messages are not only exchanged between the two peers but the messages are also to be stored in each program for the peer to refer back to as well as the time the messages were received. The two peers can also test their connections and receive an acknowledgement to ensure that the connection is stable and fine. With these in mind let's take a look at the code and see it in action between two devices.

Imports

```
import java.io.*;
import java.net.*;
import java.util.Scanner;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.ArrayList;
```

- Java.io for inputs/outputs.
- Java.net for the use of connecting the devices.
- Scanner for user input.
- SimpleDateFormat and Calendar to print and save message times.
- ArrayList to save the messages and the times they were received.

Variables

```
//variable declaration
Scanner kbd = new Scanner(System.in);
String UserInput = "";
String PrintData = "";
int msgnumber = 0;
ArrayList<String> msglist = new ArrayList<>();
ArrayList<String> msgtimelist = new ArrayList<>();
Calendar date = null;
SimpleDateFormat msgtimeformat = new SimpleDateFormat("E MM/dd/yyyy hh:mm:ss a");
String msgtime = "";
String ack = "Here is an acknowledgement proving the connection is stable.";
```

There are a few things that needed to be initialized before proceeding with the connection those being:

- A scanner to actually allow the user to interact with the program.

- A string for user input that allows the program to take the user input and send it to the peer.

- A string for printing data received from the peer.

- An integer msgnumber for the msglist and msgtimelist ArrayLists that hold the message and the time they were received.

- A calendar and date format to print the date and time of messages.

- A string msgtime to be used with the msgtimelist ArrayList.

- A string ack to be used for testing the overall connection.

The Connections

```
//reserve port for peer to connect to you
System.out.println("Reserving port 6667 to read data from Peer.Java...");
ServerSocket ReadSocket = new ServerSocket(6667);

//peer connects to this program
Socket ReadConnection = ReadSocket.accept();
System.out.println("Peer.Java is connected to you!");

//connect to peer
System.out.println("Connecting to Peer.Java to write data to Peer.Java...");
Socket WriteSocket = new Socket("127.0.0.1", 6666);
System.out.println("Connected!");
```

```
//connect to peer
System.out.println("Connecting to HostPeer.Java to write data to HostPeer.Java...");
Socket WriteSocket = new Socket("127.0.0.1", 6667);
System.out.println("Connected to HostPeer.Java!");

//reserve port for peer to connect to you
System.out.println("Reserving port 6666 to read data from HostPeer.Java...");
ServerSocket ReadSocket = new ServerSocket(6666);

//peer connects to this program
Socket ReadConnection = ReadSocket.accept();
System.out.println("HostPeer.Java is now connected to you!");
```

As for the connections the host reserves port 6667 and waits for the peer to connect. Once the connection is established, the peer that is connected will reserve port 6666 and wait for a connection from the host peer.

Reading and Writing Data

```
//read and write data from peer
BufferedReader Input = new BufferedReader(new InputStreamReader(ReadConnection.getInputStream()));
DataOutputStream Output = new DataOutputStream(WriteSocket.getOutputStream());

//acknowledgement data on this output stream
DataOutputStream Ack = new DataOutputStream(WriteSocket.getOutputStream());
```

There is one BufferedReader named Input which will read data from the peer. There are two DataOutputStreams. One named Output to send user input to the peer and the other named Ack for testing the connection and sending an acknowledgement.

Keeping the Connection Established

To keep the connection established between the two programs I used a while loop.

```
while (!UserInput.equals("exit"))
{
    if(UserInput.equals("inbox"))
    {

        System.out.println("You have: " + msglist.size() + " message(s)" +
                           "\nInput a number for the message you would like to see."
                           + "\nYour first message will be under '0'");

        msgnumber = kbd.nextInt();
        System.out.println("On [" + msgtimelist.get(msgnumber) +"] " +"Peer said: " + msglist.get(msgnumber));

    }
    else if (PrintData.equals("test"))
    {
        System.out.println("A test request has been received please press enter twice to start the connection test.");
        UserInput = kbd.nextLine();
        Ack.writeBytes(ack);
        System.out.println("Connection test successful and stable.");
    }
    else
    msgtime = msgtimeformat.format(Calendar.getInstance().getTime());
    System.out.println("[" + msgtime + "]" +" Peer: " + PrintData);
    msglist.add(PrintData);
    msgtimelist.add(msgtime);
    UserInput = kbd.nextLine();
    Output.writeBytes(UserInput + "\n");
    PrintData = Input.readLine();
}
```

This while loop is what keeps the connection established and has all of the commands within the loop. The connection stays established unless the user types “exit”.

The commands

The commands are executed by placing them in if, else if, and else statements within the while loop. Let's take a look at the code when the commands aren't used.

```
else
msgtime = msgtimeformat.format(Calendar.getInstance().getTime());
System.out.println("[" + msgtime + "]" +" Peer: " + PrintData);
msglist.add(PrintData);
msgtimelist.add(msgtime);
UserInput = kbd.nextLine();
Output.writeBytes(UserInput + "\n");
PrintData = Input.readLine();
```

The first thing that happens when a message is received is the Input BufferedReader will be converted to string PrintData. PrintData will run through the if statement to check it matches any of them. When PrintData does not match any of the if statements the string msgtime will get the current time of when the loop was executed in that moment. After that a print statement will contain the time and message that was received from the peer. Then the PrintData string is stored into the msglist ArrayList and the msgtime string is stored into the msgtime ArrayList. The user is then prompted to enter their message which is stored into string UserInput a writeBytes statement is then used through the Output DataOutputStream to send the user's message to the peer. The second program is built almost exactly like this. The only difference is that PrintData = Input.readLine(); is placed at the beginning of the while loop. Now let's take a look at the "inbox" command.

```
//if user wants to view inbox
if(UserInput.equals("inbox"))
{

    System.out.println("You have: " + msglist.size() + " message(s)" +
                       "\nInput a number for the message you would like to see."
                        + "\nYour first message will be under '0'");

    msgnumber = kbd.nextInt();
    System.out.println("On [" + msgtimelist.get(msgnumber) +"] " +"Peer said: " + msglist.get(msgnumber));

}
```

The inbox command is when the user types “inbox”. When this happens a print statement will display and show how many messages are stored in their inbox which in this case is the msglist ArrayList. The user is then prompted to select which message they would like to see with the first message being stored in zero as that is the first value in an ArrayList. After the user picks which message to see another print statement will display that contains not only the message but the time that very message was received stored in the msgtimelist ArrayList. With the “inbox” command covered, it leaves the next command, “test”.

```
//when test request is received
else if (PrintData.equals("test"))
{
    System.out.println("A test request has been received please press enter twice to start the connection test.");
    UserInput = kbd.nextLine();
    Ack.writeBytes(ack);
    System.out.println("Connection test successful and stable.");
}
```

The test command is when string PrintData is equal to test. This means that the peer uses this command to text the connection. When the peer says “test” the peer who receives it will receive a print statement that says a request to test connection has been received and the user must press enter twice. The user will do so and when this happens the DataOutputStream Ack will execute a writeBytes statement that contains String ack which says, “Here is an acknowledgement proving the connection is stable.” Which will print on the peer’s end that requested the test. With “test” covered it leaves the last command.

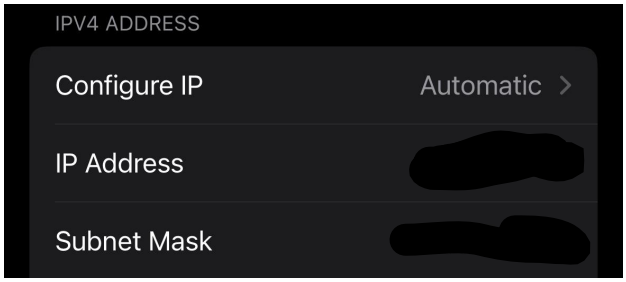
```
while (!UserInput.equals("exit"))
```

```
System.out.println("Closing all sockets and connections.");
Output.close();
Input.close();
WriteSocket.close();
ReadSocket.close();
kbd.close();
```

The final command “exit”. Does what the name suggests. It will exit the loop, close all sockets and end the program.

Testing and Output

Though my two programs work as intended, my main goal with this project is to have two separate devices communicating with each other in a LAN which I am successfully able to do. To demonstrate this and the commands, I will be using two things. Command Prompt on my laptop and Pico Compiler for iOS which is a Java IDE and Compiler that runs Java code. I would use another computer, but unfortunately I do not have one so this is the next best thing. Now, before even executing the code two things are needed. The local ip address of my laptop and the local ip address of my phone. Both of which are easy to acquire. Writing the command “ipconfig” in the command prompt on my laptop. As for my phone I can acquire the ip address by going into settings, then Wi-Fi, pressing the ‘i’ symbol on my connected Wi-Fi. In this case the IP Addresses are:



Now with this information we simply need to put the ip addresses into the code. My phone will act as the host so the first thing that will be done is to put my phone's local ip address into the Peer that is connecting to the host which will look like this.

```
//connect to peer
System.out.println("Connecting to IP x.x.x.x");
Socket WriteSocket = new Socket("[redacted]", 6667);
System.out.println("Connected!");
```

Now the host peer must connect back to the peer so in the code it will look like this.

```
40        //connect to peer
41          System.out.println("Connecting to IP x.x.x.x");
42          Socket WriteSocket = new Socket("[redacted]",
6666);
```

NOTE: The iPhone keeps its local time zone as GMT 0 and calculates the displayed time based on the user's location. So the time zone displayed on my phone when receiving messages is GMT 0. When running the programs between two computers the time zone should be fine.

Here is a basic chat:

```
Hello HostPeer!
[Mon 12/05/2022 08:22:57 PM] Peer: Hello Peer!
How are you?
[Mon 12/05/2022 08:23:23 PM] Peer: I am good, yourself?
I'm doing alright myself. Thanks for asking!
[Mon 12/05/2022 08:23:59 PM] Peer: No problem!
```

[Tue 12/06/2022 02:22:29 AM] Peer: Hello HostPeer!
Hello Peer!
[Tue 12/06/2022 02:23:08 AM] Peer: How are you?
I am good, yourself?
[Tue 12/06/2022 02:23:45 AM] Peer: I'm doing alright myself. Thanks for asking!
No problem!

Inbox command:

inbox
You have: 10 message(s)
Input a number for the message you would like to see.
Your first message will be under '0'
2
On [Tue 12/06/2022 02:23:45 AM] Peer said: I'm doing alright myself. Thanks for asking!

```
inbox
You have: 13 message(s)
Input a number for the message you would like to see.
2
On [Mon 12/05/2022 08:23:23 PM] Peer said: I am good, yourself?
```

Test command:

```
test
[Mon 12/05/2022 08:24:16 PM] Peer: Here is an acknowledgement proving the connection is stable.
```

A test request has been received please press enter twice to start the connection test.

Connection test successful and stable.
[Tue 12/06/2022 02:23:45 AM] Peer: test

test
[Tue 12/06/2022 02:54:53 AM] Peer: Here is an acknowledgement proving the connection is stable.

```
A test request has been received please press enter twice to start the connection test.

Connection test successful and stable.
[Mon 12/05/2022 08:54:42 PM] Peer: test
```

Exit command:

```
exit
Closing all sockets and connections.
```

[Tue 12/06/2022 02:59:49 AM] Peer: exit
exit
Closing all sockets and connections.

Conclusion

All in all the project was a success as I was able to accomplish all of my goals. I was able to get two different devices to connect and have a chat which was my main goal. Being able to store the message and time through ArrayLists and having a command dedicated to acknowledgments allowed for the real time implementation of a peer to peer network as each device in a peer to peer network must act as a client and server. Which is exactly what I was able to accomplish.