Lab 12:
TCP Unicast vs. UDP Multicast - Two Simple Chat Programs

Objective:

In this lab you complete two Java programs that allow you to communicate with a peer on the network in a simple chat. The first Java program requires you to utilize TCP sockets between two computers, while the second one introduces you to UDP datagrams and multicasting, allowing you to chat with everyone on the segment.

This lab is meant to help you further understand the difference between the TCP and UDP delivery mechanisms and when one is a better choice than the other. TCP sockets are robust for ensuring delivery, but UDP also has its advantages. The chat program is an excellent example where UDP multicast traffic provides functionality that would be difficult to duplicate using TCP sockets.
Introduction:

To start out, you need to create a new project in Eclipse and import the following source files into your project:

- **ChatClient.java**: This is the main entry point of the application.
- **ChatGUI.java**: This creates the GUI for the chat program. 
- **IChatEngine.java**: Interface definition for a Chat Engine. 
- **IChatEngine.java**: Engine code to drive a unicast TCP chat client.
- **UnicastChatEngine.java**: Engine code to drive a multicast UDP chat client.

You will be adding code to the UnicastChatEngine and MulticastChatEngine classes to provide the functionality of the chat client. The ChatClient class will only need to be changed to indicate which ChatEngine you are invoking. The remaining classes will not need to be modified in any way.

Be sure to use the Java API to help in constructing the appropriate objects. Also, you will need to implement try {} catch {} blocks to handle IOExceptions throughout your code.

Unicast Chat Client:

This simple chat program requires that one participant be configured as a server ("waiting" for the client connection), while the other participant is a client (who "connects" to the server.) Once the connection has been established, the participants can send messages back and forth through their GUI interfaces.

Examine the UnicastChatEngine code. Notice that the constructor is already built for you. It determines at runtime whether to run as a client or server. A boolean variable called runAsServer is configured to be true if a server and false if a client. Notice that member fields have been provided to you. Use these fields to complete the run() and sendPacket() methods. Remember from previous labs involving sockets that once the connection has been made, you must create PrintWriter and BufferedReader objects to enable data to be written-to and read-from the socket.

Step 1:

In the UnicastChatEngine, complete the code in the run() method. You need to write code to function as a server and as a client (use the existing if construct).

For the SERVER: use serverSckt to listen for a connection. Then use clientSckt to obtain a connection from the client

For the CLIENT: use clientSckt to establish a connection to host on the port.

Use input and output to hold the PrintWriter and BufferedReader objects from the connection.

Loop while data read from the BufferedReader is not null, and print the message to the GUI using the appendText method of the chatGUI object.
** It is a good idea to include status messages throughout the initial setup of the connections in the GUI as well. (ie. 'waiting for connection....', or 'connection established from ....', etc.)

In the UnicastChatEngine, construct the code in the sendPacket() method. Using the string parameter passed to it, call the appropriate method to send the text to the other end of the socket.

**Step 2:**

Test a chat session with your partner as both a client and as a server. Both of you run the program ChatClient.java, one of you runs it as a server and one of you runs it as a client. (If a partner is not available, you can modify the code in ChatClient to start two GUI clients, and run one as a server and the other as a client on the same machine.)

**Question:** The UnicastChatEngine class implements a thread that "listens" for incoming messages. What benefit is derived from having this implemented as a separate thread?

**Question:** What modification would need to be made to the sendPacket() method to allow for local messages to be echoed in the chat window? (Make this modification.)

**Hand in:** Hand in a paper copy of your UnicastChatEngine.java. Also hand in “snip-shots” of both chat windows: both the server and the client. The chat windows should provide evidence that your code worked correctly.

**Multicast Chat Client:**

The multicast chat client subscribes to a multicast group. Messages sent to the "group" are received by all clients participating. Since there is no established connection between the sender and receiver, this provides an excellent opportunity to work with UDP packets.

Examine the MulticastChatEngine code. You will have to complete the constructor and code the sendPacket() and run() methods. (I changed this. I provided the correct code for the run() method. You need to provide the code for the sendPacket() method.

**Step 1:**

In the MulticastChatEngine, write the constructor code that initializes the sckt to a multicast socket on port 4446.

Using address, obtain a new InetAddress object for the multicast address 230.0.0.1.
Using the `seckt` join the group identified by `address`.

[Build the code in the `run()` method. This is the "listener" code that is spawned as a separate thread.

Create a new DatagramPacket object `receivePkt`.

Loop indefinitely while reading packets from the socket. For each packet received, you will have to construct a `String` object to hold the data from the packet. Then you will display the data on the user's GUI using the `chatGUI.appendText()` method.] – all of this was done for you.

Finally, write the `sendPacket()` method that delivers chat messages to the network. This is called by the GUI when a user enters text to send.

Create a `sendPkt` using the `data` parameter passed to the method. (Hint: use the provided `sendBuffer` byte array).

Send the resulting `sendPkt` on the network.

**Step 2:**

Test a chat session with your partner. If a partner is not available, you can modify the code in ChatClient to start two GUI clients on the same machine.

A "robot" process has been started on the segment that broadcasts a message every 20 seconds or so throughout the lab session. You should see these messages appear in your GUI if you have correctly implemented your chat client.

**Question:** The multicast chat client implements both network "send" and "listen" methods. Why don't we need to be concerned whether or not this client is acting as a client or a server at startup as we did with the UnicastChatEngine?

____________________________________________________________________

**Question:** In the UnicastChatEngine, you needed to explicitly echo local messages to the user's GUI. Why don't you need to do the same for the MulticastChatEngine to ensure the local user sees his or her own messages in the window?

____________________________________________________________________

**Question:** How could you write an application using TCP connection(s) to get the same behavior as you get with the MulticastChatEngine?

____________________________________________________________________

**Hand in:** Hand in a paper copy of your `MulticastChatEngine.java`. Also hand in “snip-shots” of three chat windows that participate in a chat. The chat windows should provide evidence that your code worked correctly, and that the chat windows captured messages from the robot.
**Extra Credit:** Modify the multicast chat client you wrote to log the chat session. Record the date/time, host name, IP address, and text of every message received. Write the log to a file. Hand in a paper copy of your code and a copy of a log file.

**Help Policy:**

Help Policy in Effect for This Assignment: **Group Project with Limited Collaboration**

In particular, you may discuss the assignment and concepts related to the assignment with the following persons, in addition to an instructor in this course: any member of your group; any St. Bonaventure Computer Science instructor; and any student enrolled in CS 254.

You may use the following materials produced by other students: **materials produced by members of your group.**