**Lebanese American University**

**Fall 2022**

****

**TCP client and server, and UDP client and server**

**CSC430:Computer Nertworks**

**Group 1**

**Omar Al Rafei**

**Toufic Al Rifaii**

**Larissa Abboud**

Table of Contents

[Specification: 3](#_Toc120752029)

[Design and testing plan: 3](#_Toc120752030)

[TCP client-server: 3](#_Toc120752031)

[UDP client-server: 5](#_Toc120752032)

[Implementation: 6](#_Toc120752033)

[Using TCP protocol: 6](#_Toc120752034)

[Using UDP protocol: 12](#_Toc120752035)

[Testing: 18](#_Toc120752036)

[TCP CLIENT-SERVER: 18](#_Toc120752037)

[UDP LCIENT-SERVER: 18](#_Toc120752038)

# Specification:

The program is a multiple client-server program that allows clients to chat with each other. This program is implemented using TCP and UDP protocols separately. With the TCP server, there is a welcoming socket which has a port number and each time a client initiates connection to the server a new socket is created. The TCP client server program is a messenger type of program where the server waits for client connection. The client will begin by connecting to the chat and sending a message which is sent to the server and broadcasted back to the chat where clients will communicate with each other. On the other hand, with the UDP server, there is no welcoming socket, and all data from different clients enter the server through one socket. The UDP client server program implements the same concept as the UDP client server program as well.

# Design and testing plan:

## TCP client-server:

For server:

·   Create a server socket for the server which takes the port number (no need for ip address) using ServerSocket (int port\_number).

·   The client accepts a socket; to handle multiple clients we open a new instance each time (handle multiple clients concurrently).

·   Begin by initializing all

o   Print writer which we get from the client to send.

o   Buffered reader which takes input stream which we get from the client.

o   Function connection handler which will deal with the client.

·   In connection handler we have the client which accepts a socket, the buffered reader (get stream from socket so when the client sends something, we want to get it from buffer reader) and print writer (want to write something to the client).

· Begin process:

o   Server sends to the client a message[out].

o   Wait for input from the client[in].

o To broadcast to all other clients [list of client (connection handlers)] loop through each client and simply send the message to the chat.

· All handlers must be run in a thread pool [ threads that can be reused since they are short lived], and the server runs in one thread all the time.

o   ExecutorService will be the thread pool

o   Initialized after beginning.

o   Executed after each connection.

· Main (): create new server and run [run connection handler for each connection]

· Summary: loop to continuously obtain input from the client with several conditions according to the commands.  And then broadcast it to the chat.

o   /nick: assigns a nickname to the client.

o   /quit: closes socket; shutdown the chat when we close the client. Close input and output streams as well.

o   Normal chatting

For client:

· 2 threads, one that receives all messages from the server and another which receives the input.

· Initializing server address details using Socket(“id\_address”,int port\_number).

· Create a client socket for the server which takes the local host and server port.

· In class client which we initialize

o   Print writer which takes output streams that we get from the client.

o   Buffered reader which takes input streams.

o   Function input Handler which will constantly ask for new console lines.

· In input handler we have buffered reader which accepts system input stream (command line input) and simple commands:

o    /quit: closes socket; shutdown the group chat when we close the client. Close input and output streams as well.

o   Messages the client wants to send.

· Create thread for class client  which takes an input handler, use start to open a separate thread unlike run().

· Summary:

o   Create client.

o   Initialize input and output stream and input handler which takes input from command line.

o   Create thread that takes the input handler created

o   Messages are read and exchanged

· Main () creates a client and begins the connection.

## UDP client-server:

For server:

· DatagramPacket and DatagramSocket are the two main classes that are used to implement a UDP client/server application. DatagramPacket is a data container and DatagramSocket is a mechanism to send and receive DatagramPackets.

· Create a DatagramPacket to be sent, so you need to specify the address and port number of the destination host.

· The parameter length specifies the amount of data in the byte array to be used, usually it is the length of the array.

. Create a DatagramSocket object to establish a UDP connection for sending and receiving datagram which takes a port number.

· Blocks until a datagram is received. And the DatagramPacket is sent to the clients.

· The server also needs to know the client's address and port to send the DatagramPacket. This information can be obtained from the DatagramPacket received from the client previously. And a String is converted to an array of bytes which then can be wrapped in a DatagramPacket.

For client:

· Sends a DatagramPacket to a server specified by hostname and port.

· Specify the server’s address and port in the DatagramPacket. So the code just sends a signal to the server

· Receives a DatagramPacket from the server.

· Once the socket is opened, receiving a packet is very simple. the byte array is converted to a String to be printed in readable format.

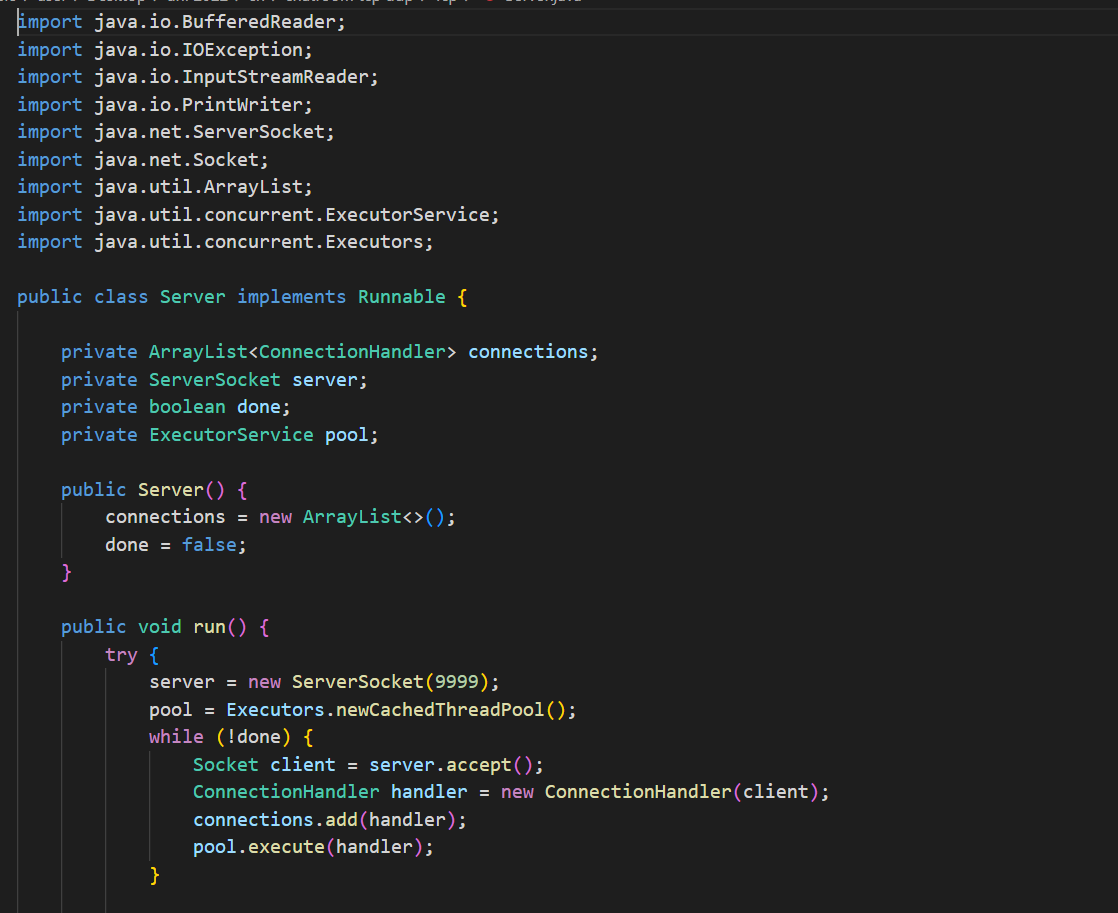
· In message receiver the packet is created and the socket receives the packet

· Then we obtain the data and display in the chat

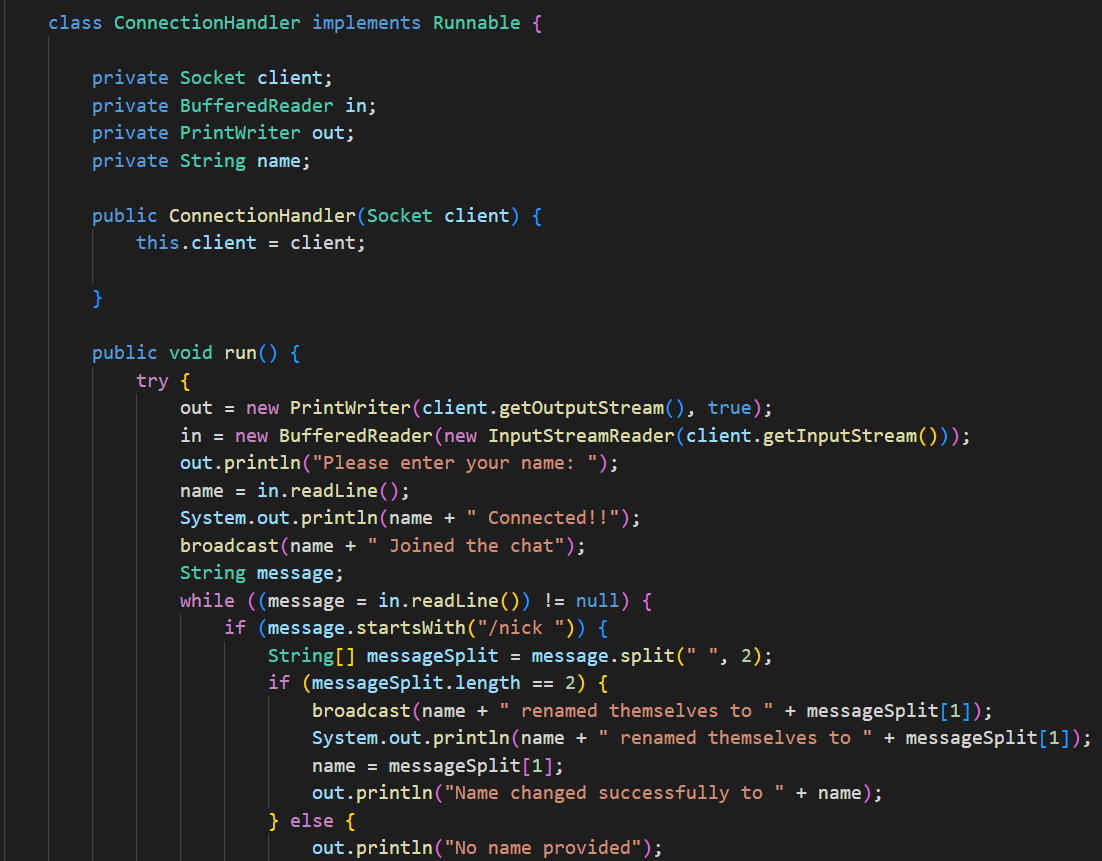
# Implementation:

## Using TCP protocol:

Class Server.java:

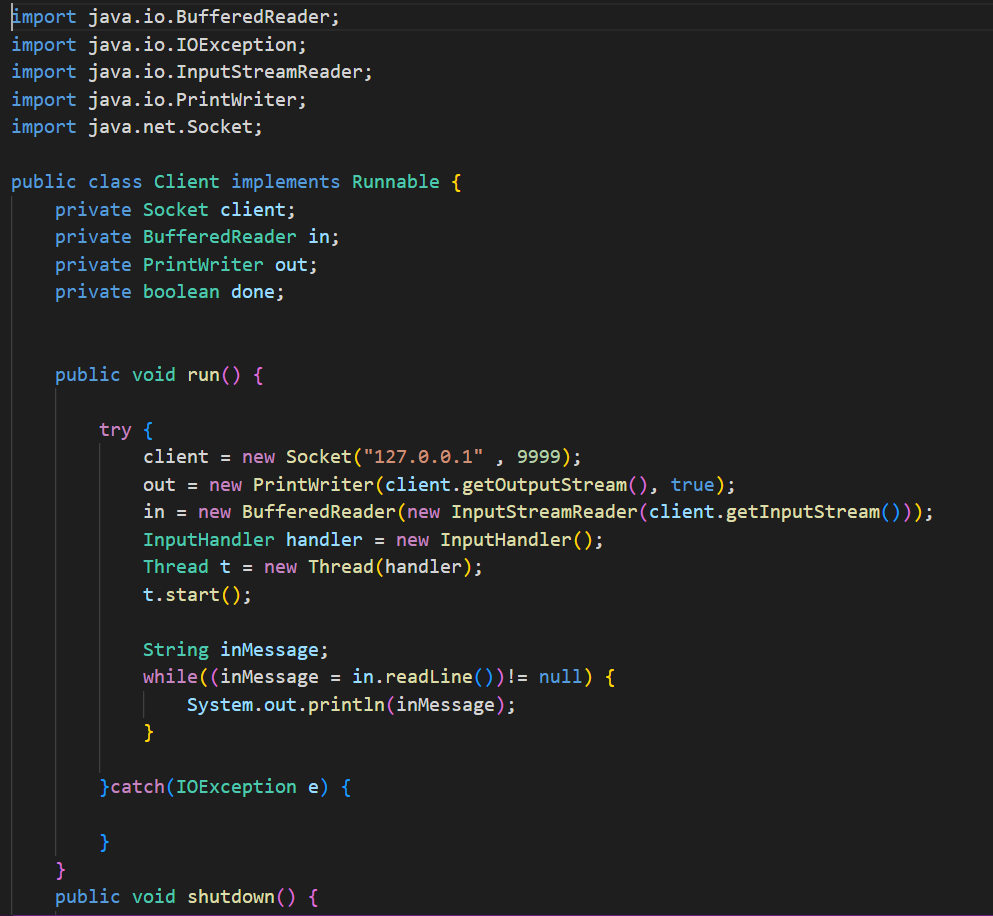




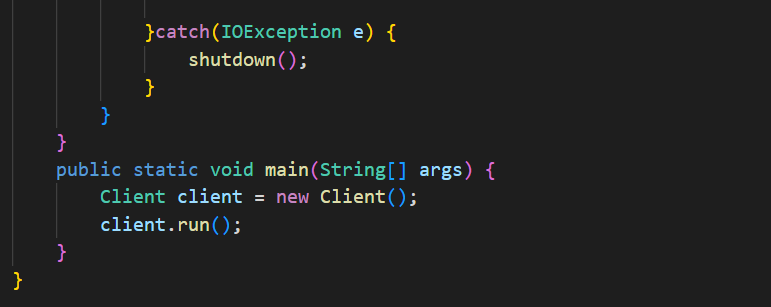




Class Client.java:

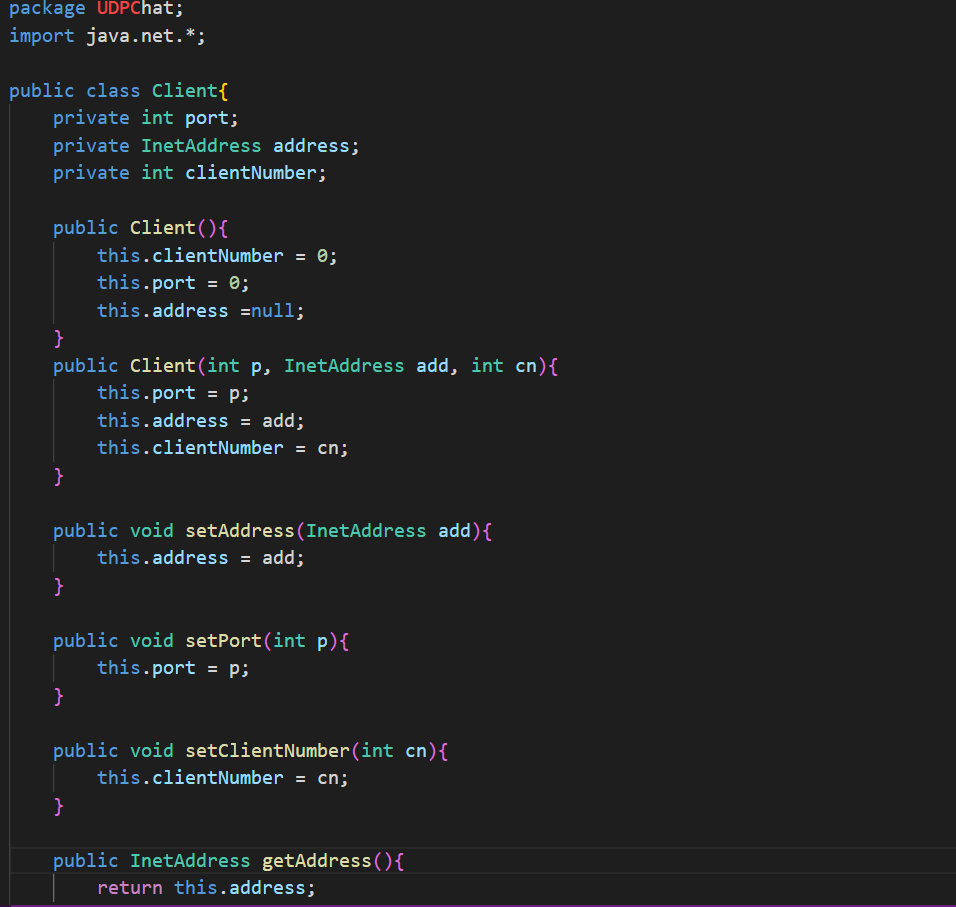


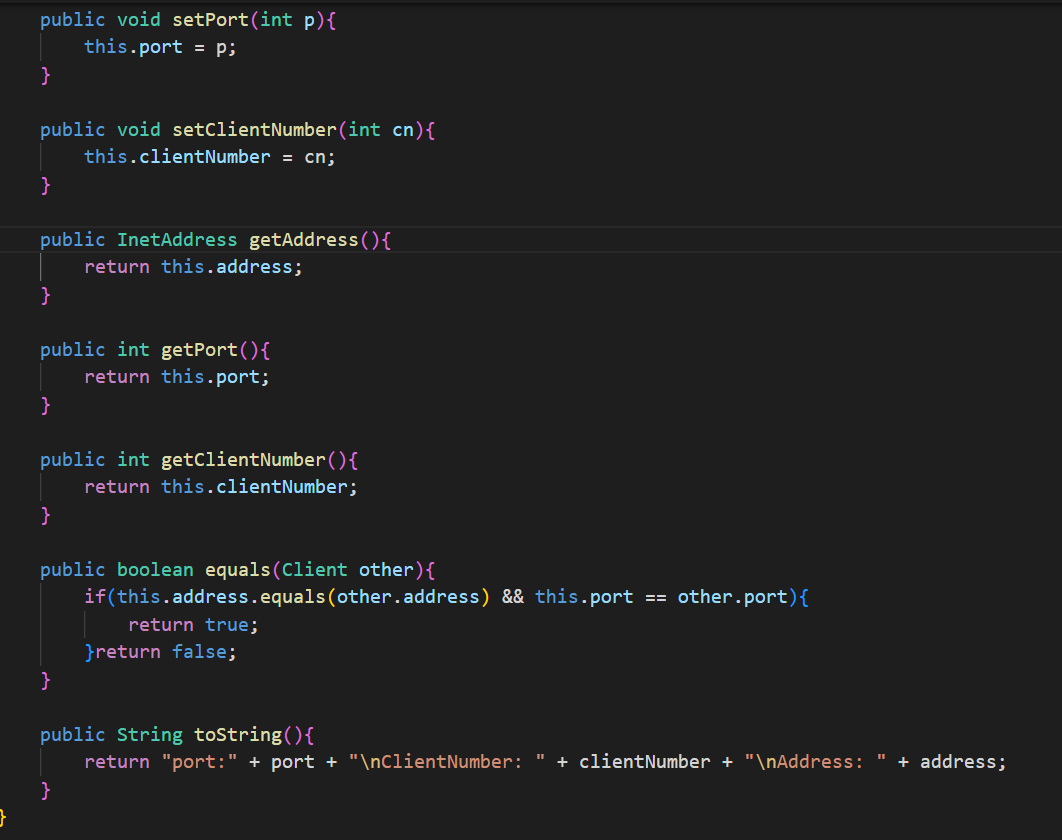




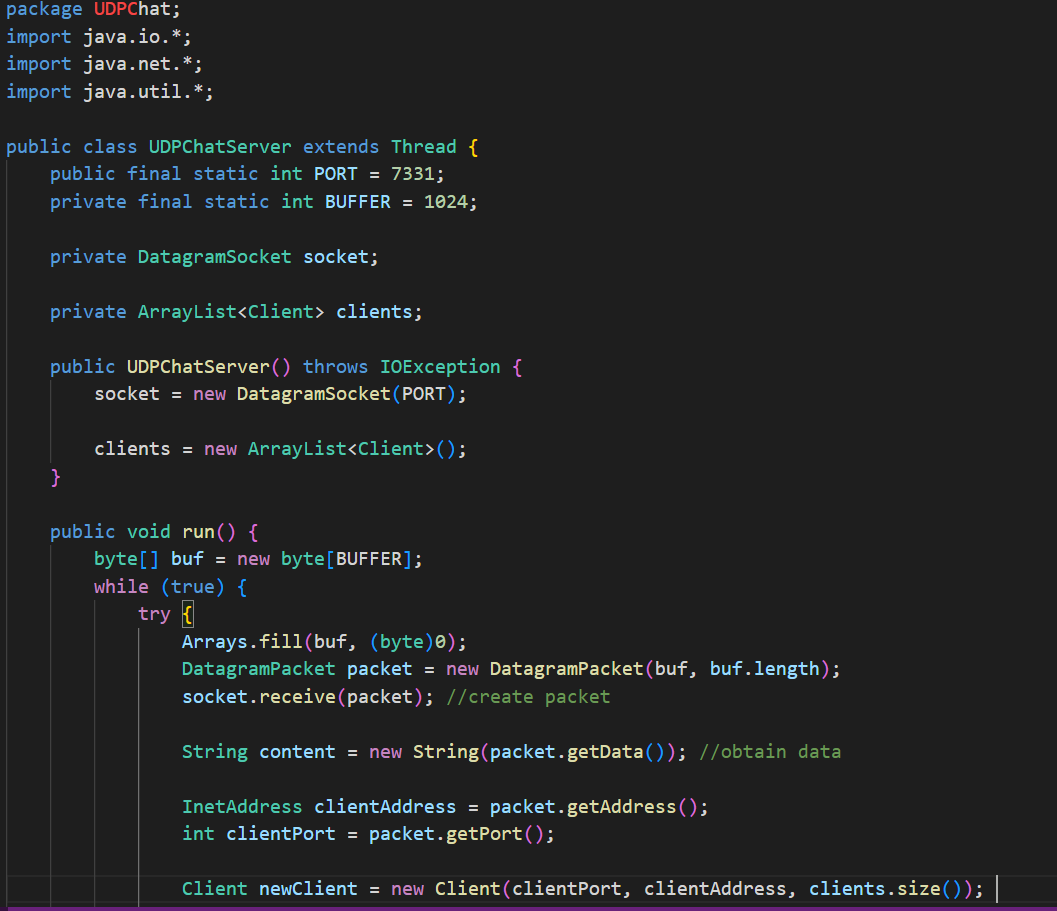
## Using UDP protocol:

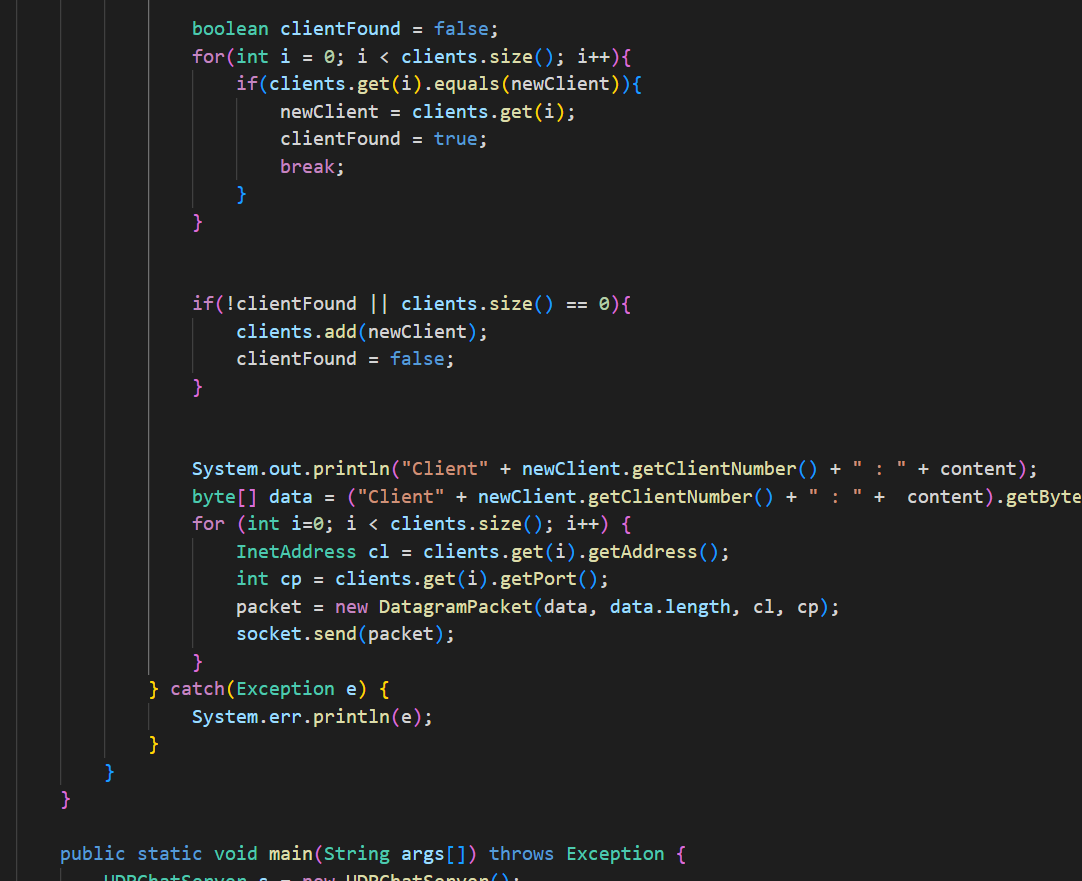
Class Client.java:

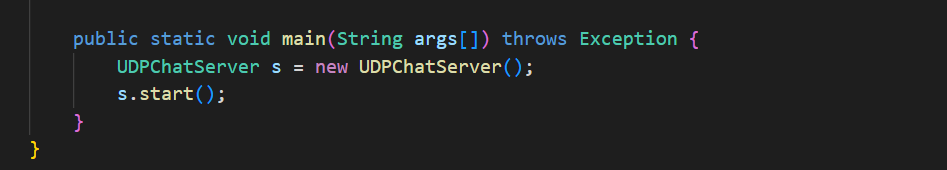




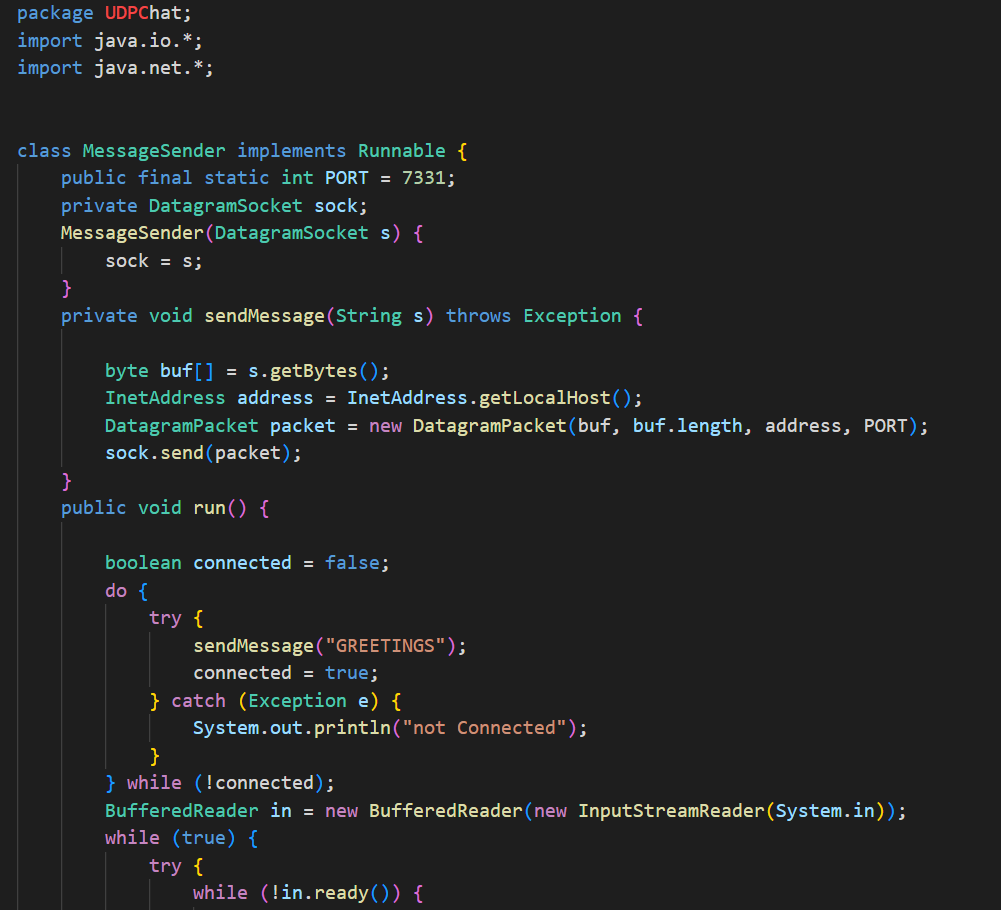
Class UDPChatServer.java:



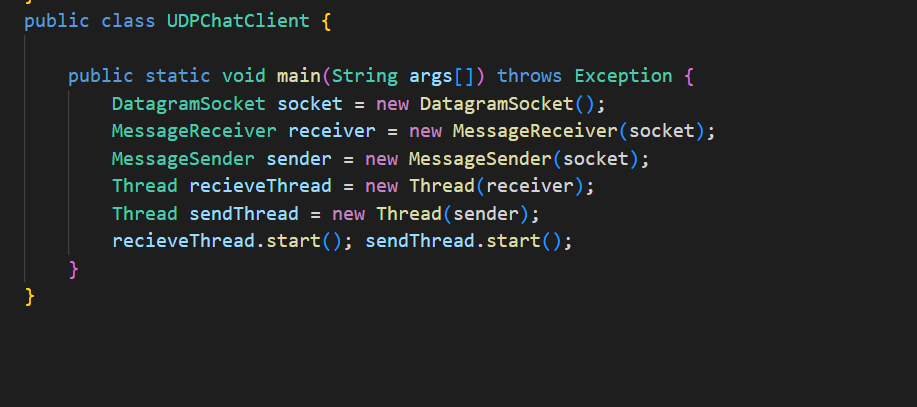




Class UDPChatClient.java:

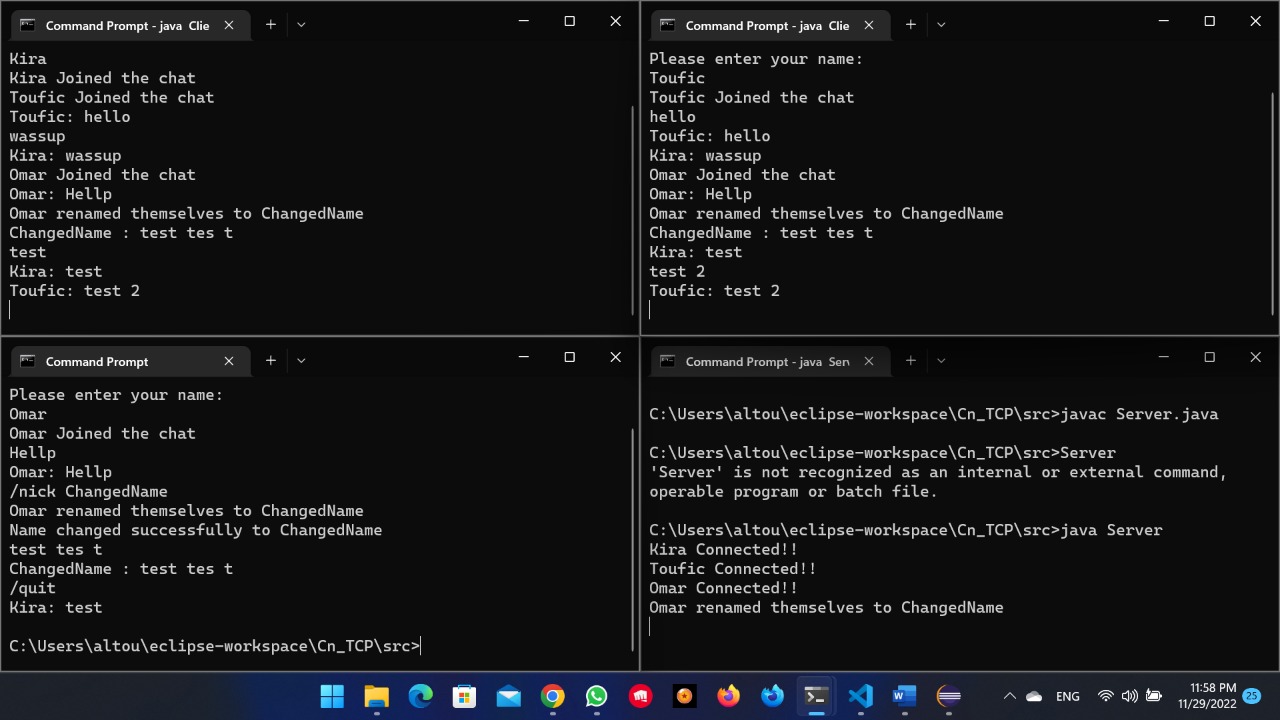




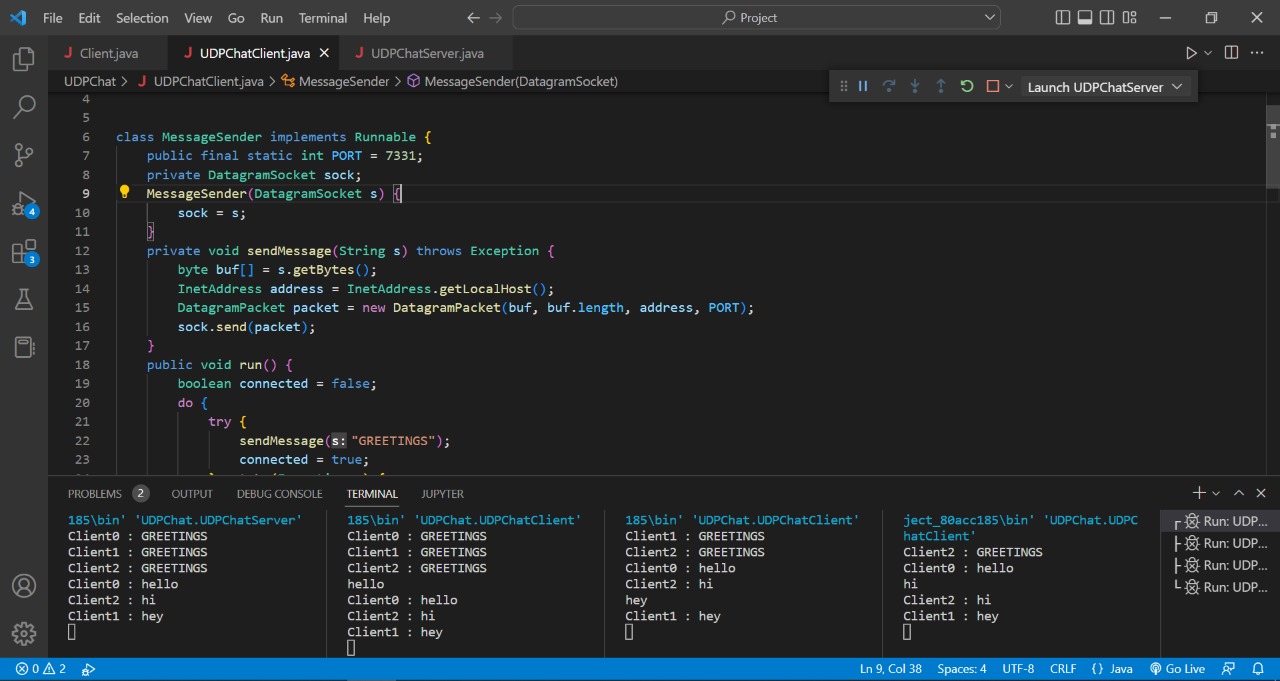


# Testing:

## TCP CLIENT-SERVER:



## UDP LCIENT-SERVER:



Reference:

[Simple TCP Chat Room in Java](https://www.youtube.com/watch?v=hIc_9Wbn704)

[GitHub - ratanak1010/Java-UDP-Chat: Simple program to chat in Java](https://github.com/ratanak1010/Java-UDP-Chat)