NAME: PRANJAL SIRASKAR

ROLLNO: 33

CLASS: C

BATCH: B2

PROJECT TITLE: UDP based Multithread Chat Client-Server

**Program:**

**Chat Message Class: -**

import java.io.\*;

/\*

\* This class defines the different type of messages that will be exchanged between the

\* Clients and the Server.

\* When talking from a Java Client to a Java Server a lot easier to pass Java objects, no

\* need to count bytes or to wait for a line feed at the end of the frame

\*/

public class ChatMessage implements Serializable {

// The different types of message sent by the Client

// WHOISIN to receive the list of the users connected

// MESSAGE an ordinary text message

// LOGOUT to disconnect from the Server

static final int WHOISIN = 0, MESSAGE = 1, LOGOUT = 2;

private int type;

private String message;

// constructor

ChatMessage(int type, String message) {

this.type = type;

this.message = message;

}

int getType() {

return type;

}

String getMessage() {

return message;

}

}

**Server Side: -**

import java.io.\*;

import java.net.\*;

import java.util.\*;

// the server that can be run as a console

public class Server {

// a unique ID for each connection

private static int uniqueId;

// an ArrayList to keep the list of the Client

private ArrayList<ClientThread> al;

// the port number to listen for connection

private int port;

// to check if server is running

private boolean keepGoing;

// notification

private String notif = " --- ";

//constructor that receive the port to listen to for connection as parameter

public Server(int port) {

// the port

this.port = port;

// an ArrayList to keep the list of the Client

al = new ArrayList<ClientThread>();

}

public void start() {

keepGoing = true;

//create socket server and wait for connection requests

try

{

// the socket used by the server

ServerSocket serverSocket = new ServerSocket(port);

// infinite loop to wait for connections ( till server is active )

while(keepGoing)

{

display("Server waiting for Clients .......... ");

// accept connection if requested from client

Socket socket = serverSocket.accept();

// break if server stoped

if(!keepGoing)

break;

// if client is connected, create its thread

ClientThread t = new ClientThread(socket);

//add this client to arraylist

al.add(t);

t.start();

}

// try to stop the server

try {

serverSocket.close();

for(int i = 0; i < al.size(); ++i) {

ClientThread tc = al.get(i);

try {

// close all data streams and socket

tc.sInput.close();

tc.sOutput.close();

tc.socket.close();

}

catch(IOException ioE) {

}

}

}

catch(Exception e) {

display("Exception closing the server and clients: " + e);

}

}

catch (IOException e) {

String msg = " Exception on new ServerSocket: " + e + "\n";

display(msg);

}

}

// to stop the server

protected void stop() {

keepGoing = false;

try {

new Socket("localhost", port);

}

catch(Exception e) {

}

}

// Display an event to the console

private void display(String msg) {

String time = msg;

System.out.println(time);

}

// to broadcast a message to all Clients

private synchronized boolean broadcast(String message) {

// add timestamp to the message

//String time = sdf.format(new Date());

// to check if message is private i.e. client to client message

String[] w = message.split(" ",3);

boolean isPrivate = false;

if(w[1].charAt(0)=='@')

isPrivate=true;

// if private message, send message to mentioned username only

if(isPrivate==true)

{

String tocheck=w[1].substring(1, w[1].length());

message="Personal message :"+w[0]+w[2];

String messageLf = message + "\n";

boolean found=false;

// we loop in reverse order to find the mentioned username

for(int y=al.size(); --y>=0;)

{

ClientThread ct1=al.get(y);

String check=ct1.getUsername();

if(check.equals(tocheck))

{

// try to write to the Client if it fails remove it from the list

if(!ct1.writeMsg(messageLf)) {

al.remove(y);

display("Disconnected Client " + ct1.username + " removed from list.");

}

// username found and delivered the message

found=true;

break;

}

}

// mentioned user not found, return false

if(found!=true)

{

return false;

}

}

// if message is a broadcast message

else

{

String messageLf = message + "\n";

// display message

System.out.print(messageLf);

// we loop in reverse order in case we would have to remove a Client

// because it has disconnected

for(int i = al.size(); --i >= 0;) {

ClientThread ct = al.get(i);

// try to write to the Client if it fails remove it from the list

if(!ct.writeMsg(messageLf)) {

al.remove(i);

display("Disconnected Client " + ct.username + " removed from list.");

}

}

}

return true;

}

// if client sent LOGOUT message to exit

synchronized void remove(int id) {

String disconnectedClient = "";

// scan the array list until we found the Id

for(int i = 0; i < al.size(); ++i) {

ClientThread ct = al.get(i);

// if found remove it

if(ct.id == id) {

disconnectedClient = ct.getUsername();

al.remove(i);

break;

}

}

broadcast(notif + disconnectedClient + " has left the chat room." + notif);

}

public static void main(String[] args) {

// start server on port 1500 unless a PortNumber is specified

int portNumber = 1500;

// create a server object and start it

Server server = new Server(portNumber);

server.start();

}

// One instance of this thread will run for each client

class ClientThread extends Thread {

// the socket to get messages from client

Socket socket;

ObjectInputStream sInput;

ObjectOutputStream sOutput;

// my unique id (easier for deconnection)

int id;

// the Username of the Client

String username;

// message object to recieve message and its type

ChatMessage cm;

// timestamp

//String date;

// Constructor

ClientThread(Socket socket) {

// a unique id

id = ++uniqueId;

this.socket = socket;

//Creating both Data Stream

System.out.println("Thread trying to create Object Input/Output Streams");

try

{

sOutput = new ObjectOutputStream(socket.getOutputStream());

sInput = new ObjectInputStream(socket.getInputStream());

// read the username

username = (String) sInput.readObject();

broadcast(notif + username + " has joined the chat room." + notif);

}

catch (IOException e) {

display("Exception creating new Input/output Streams: " + e);

return;

}

catch (ClassNotFoundException e) {

}

//date = new Date().toString() + "\n";

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

// infinite loop to read and forward message

public void run() {

// to loop until LOGOUT

boolean keepGoing = true;

while(keepGoing) {

// read a String (which is an object)

try {

cm = (ChatMessage) sInput.readObject();

}

catch (IOException e) {

display(username + " Exception reading Streams: " + e);

break;

}

catch(ClassNotFoundException e2) {

break;

}

// get the message from the ChatMessage object received

String message = cm.getMessage();

// different actions based on type message

switch(cm.getType()) {

case ChatMessage.MESSAGE:

boolean confirmation = broadcast(username + ": " + message);

if(confirmation==false){

String msg = notif + "No user avaialable" + notif;

writeMsg(msg);

}

break;

case ChatMessage.LOGOUT:

display(username + " disconnected with a LOGOUT message.");

keepGoing = false;

break;

case ChatMessage.WHOISIN:

writeMsg("List of the users connected at \n");

// send list of active clients

for(int i = 0; i < al.size(); ++i) {

ClientThread ct = al.get(i);

writeMsg((i+1) + ") " + ct.username );

}

break;

}

}

// if out of the loop then disconnected and remove from client list

remove(id);

close();

}

// close everything

private void close() {

try {

if(sOutput != null) sOutput.close();

}

catch(Exception e) {}

try {

if(sInput != null) sInput.close();

}

catch(Exception e) {};

try {

if(socket != null) socket.close();

}

catch (Exception e) {}

}

// write a String to the Client output stream

private boolean writeMsg(String msg) {

// if Client is still connected send the message to it

if(!socket.isConnected()) {

close();

return false;

}

// write the message to the stream

try {

sOutput.writeObject(msg);

}

// if an error occurs, do not abort just inform the user

catch(IOException e) {

display(notif + "Error sending message to " + username + notif);

display(e.toString());

}

return true;

}

}

}

**Client Side: -**

import java.net.\*;

import java.io.\*;

import java.util.\*;

//The Client that can be run as a console

public class Client {

// notification

private String notif = " --- ";

// for I/O

private ObjectInputStream sInput; // to read from the socket

private ObjectOutputStream sOutput; // to write on the socket

private Socket socket; // socket object

private String server, username; // server and username

private int port; //port

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

/\*

\* Constructor to set below things

\* server: the server address

\* port: the port number

\* username: the username

\*/

Client(String server, int port, String username) {

this.server = server;

this.port = port;

this.username = username;

}

/\*

\* To start the chat

\*/

public boolean start() {

// try to connect to the server

try {

socket = new Socket(server, port);

}

// exception handler if it failed

catch(Exception ec) {

display("Error connectiong to server:" + ec);

return false;

}

String msg = "Connection accepted " + socket.getInetAddress() + ":" + socket.getPort();

display(msg);

/\* Creating both Data Stream \*/

try

{

sInput = new ObjectInputStream(socket.getInputStream());

sOutput = new ObjectOutputStream(socket.getOutputStream());

}

catch (IOException eIO) {

display("Exception creating new Input/output Streams: " + eIO);

return false;

}

// creates the Thread to listen from the server

new ListenFromServer().start();

// Send our username to the server this is the only message that we

// will send as a String. All other messages will be ChatMessage objects

try

{

sOutput.writeObject(username);

}

catch (IOException eIO) {

display("Exception doing login : " + eIO);

disconnect();

return false;

}

// success we inform the caller that it worked

return true;

}

/\*

\* To send a message to the console

\*/

private void display(String msg) {

System.out.println(msg);

}

/\*

\* To send a message to the server

\*/

void sendMessage(ChatMessage msg) {

try {

sOutput.writeObject(msg);

}

catch(IOException e) {

display("Exception writing to server: " + e);

}

}

/\*

\* When something goes wrong

\* Close the Input/Output streams and disconnect

\*/

private void disconnect() {

try {

if(sInput != null) sInput.close();

}

catch(Exception e) {}

try {

if(sOutput != null) sOutput.close();

}

catch(Exception e) {}

try{

if(socket != null) socket.close();

}

catch(Exception e) {}

}

public static void main(String[] args) {

// default values if not entered

int portNumber = 1500;

String serverAddress = "localhost";

String userName = "Anonymous";

Scanner scan = new Scanner(System.in);

System.out.println("Enter the username: ");

userName = scan.nextLine();

// create the Client object

Client client = new Client(serverAddress, portNumber, userName);

// try to connect to the server and return if not connected

if(!client.start())

return;

System.out.println("\nHello.! Welcome to the chatroom.");

System.out.println("Instructions:");

System.out.println("1. Simply type the message to send broadcast to all active clients");

System.out.println("2. Type '@username<space>yourmessage' without quotes to send message to desired client");

System.out.println("3. Type 'WHOISIN' without quotes to see list of active clients");

System.out.println("4. Type 'LOGOUT' without quotes to logoff from server");

// infinite loop to get the input from the user

while(true) {

System.out.print(">>>> ");

// read message from user

String msg = scan.nextLine();

// logout if message is LOGOUT

if(msg.equalsIgnoreCase("LOGOUT")) {

client.sendMessage(new ChatMessage(ChatMessage.LOGOUT, ""));

break;

}

// message to check who are present in chatroom

else if(msg.equalsIgnoreCase("WHOISIN")) {

client.sendMessage(new ChatMessage(ChatMessage.WHOISIN, ""));

}

// regular text message

else {

client.sendMessage(new ChatMessage(ChatMessage.MESSAGE, msg));

}

}

// close resource

scan.close();

// client completed its job. disconnect client.

client.disconnect();

}

/\*

\* a class that waits for the message from the server

\*/

class ListenFromServer extends Thread {

public void run() {

while(true) {

try {

// read the message form the input datastream

String msg = (String) sInput.readObject();

// print the message

System.out.println(msg);

System.out.print(">>> ");

}

catch(IOException e) {

display(notif + "Server has closed the connection: " + e + notif);

break;

}

catch(ClassNotFoundException e2) {

}

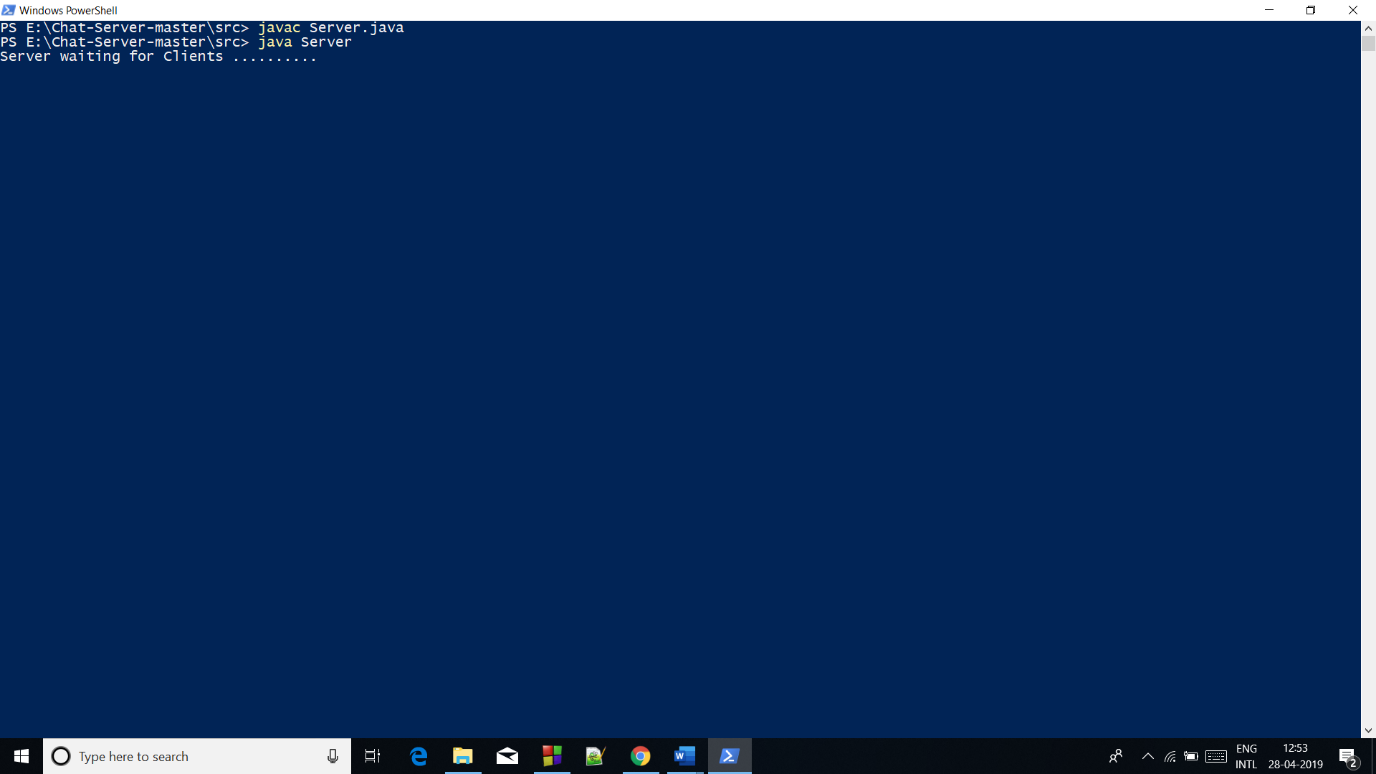
}

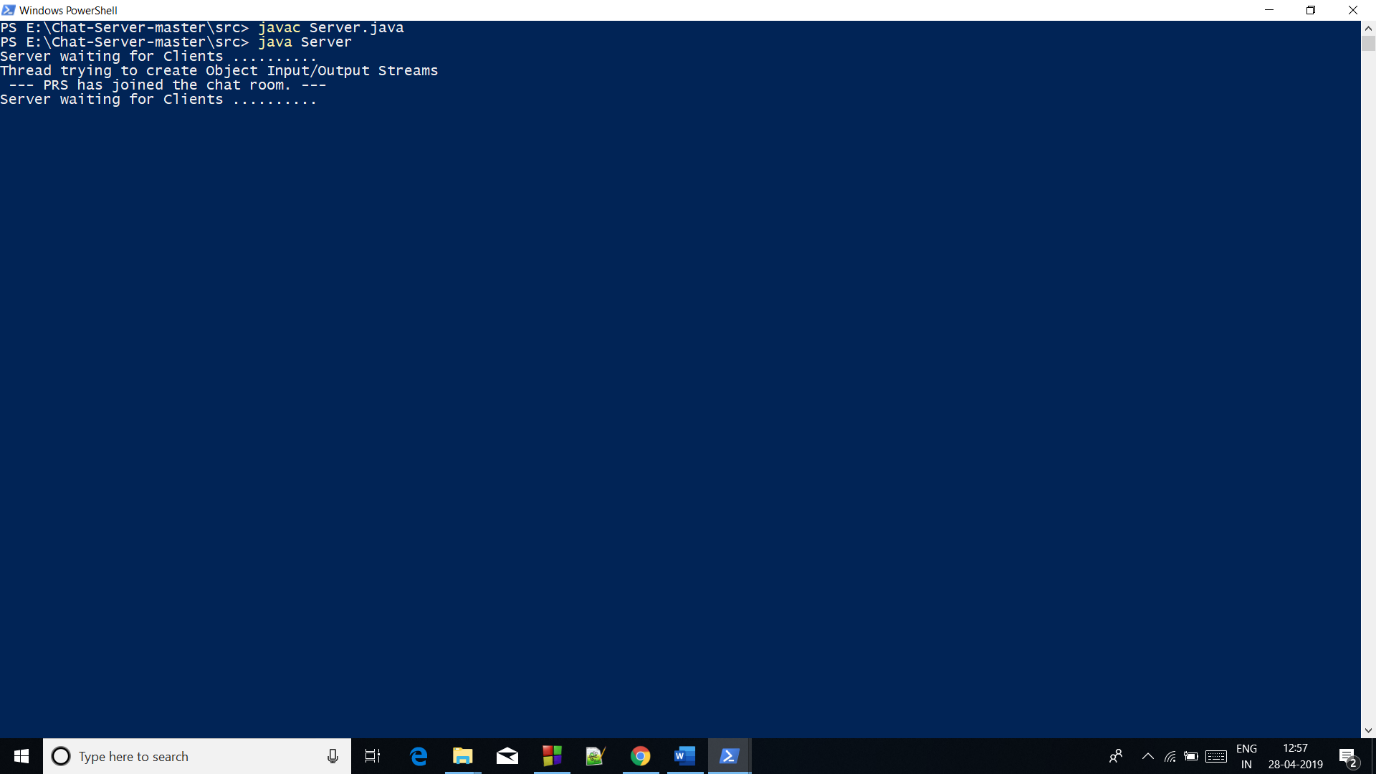
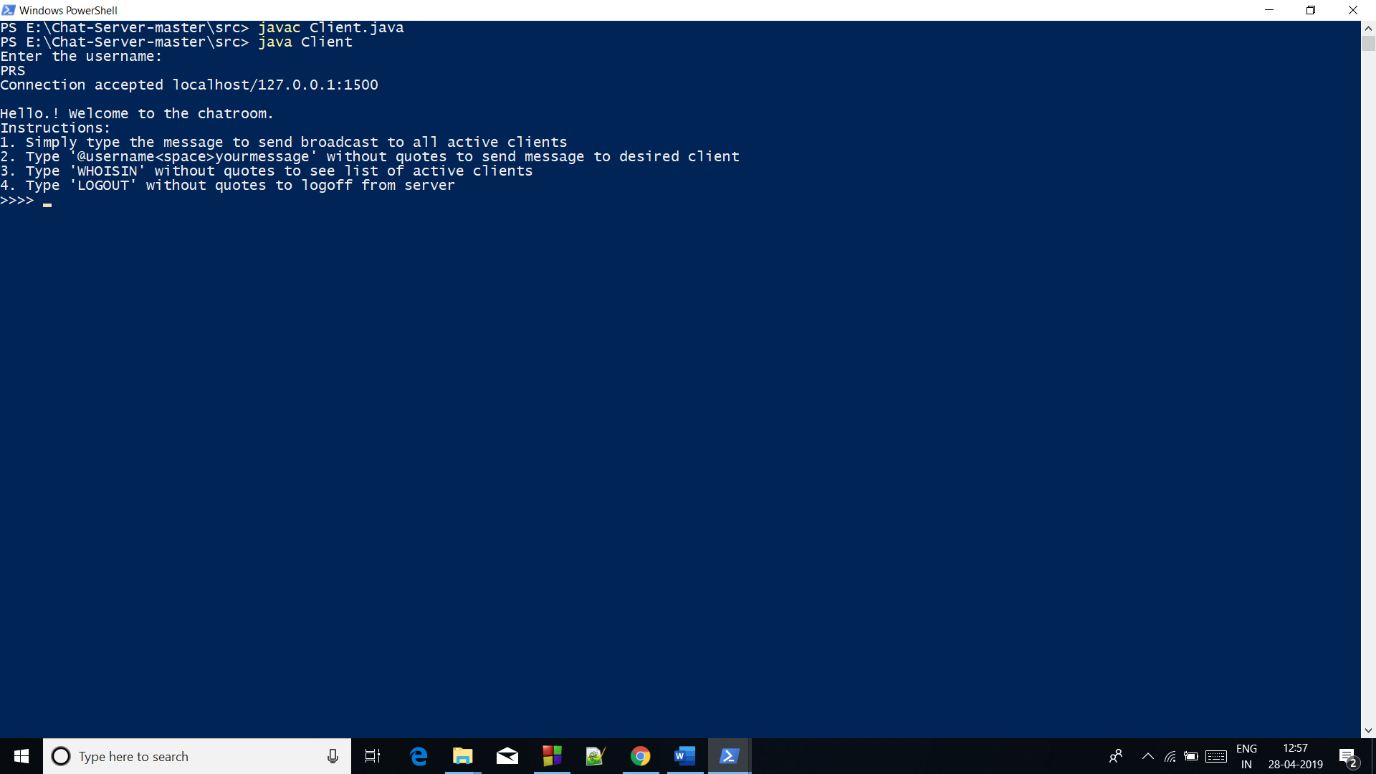
}

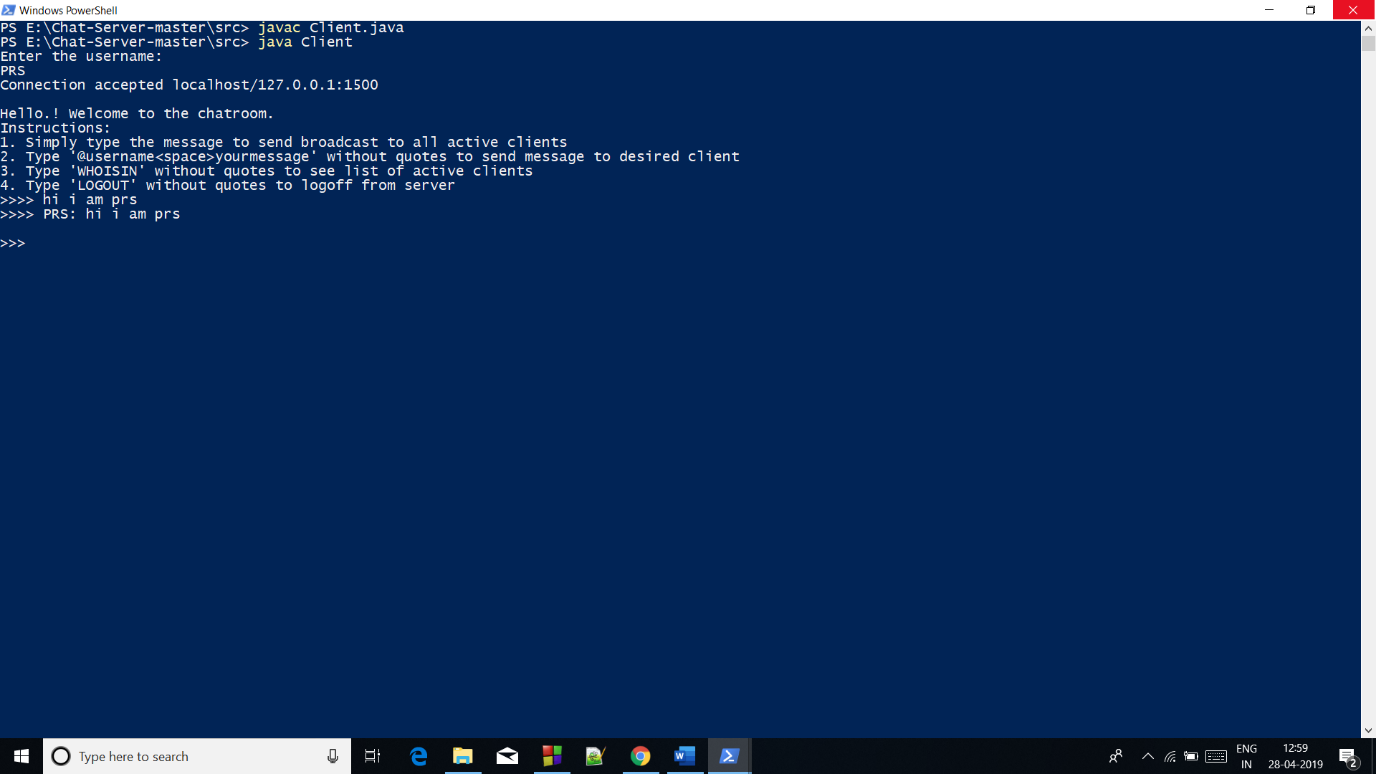
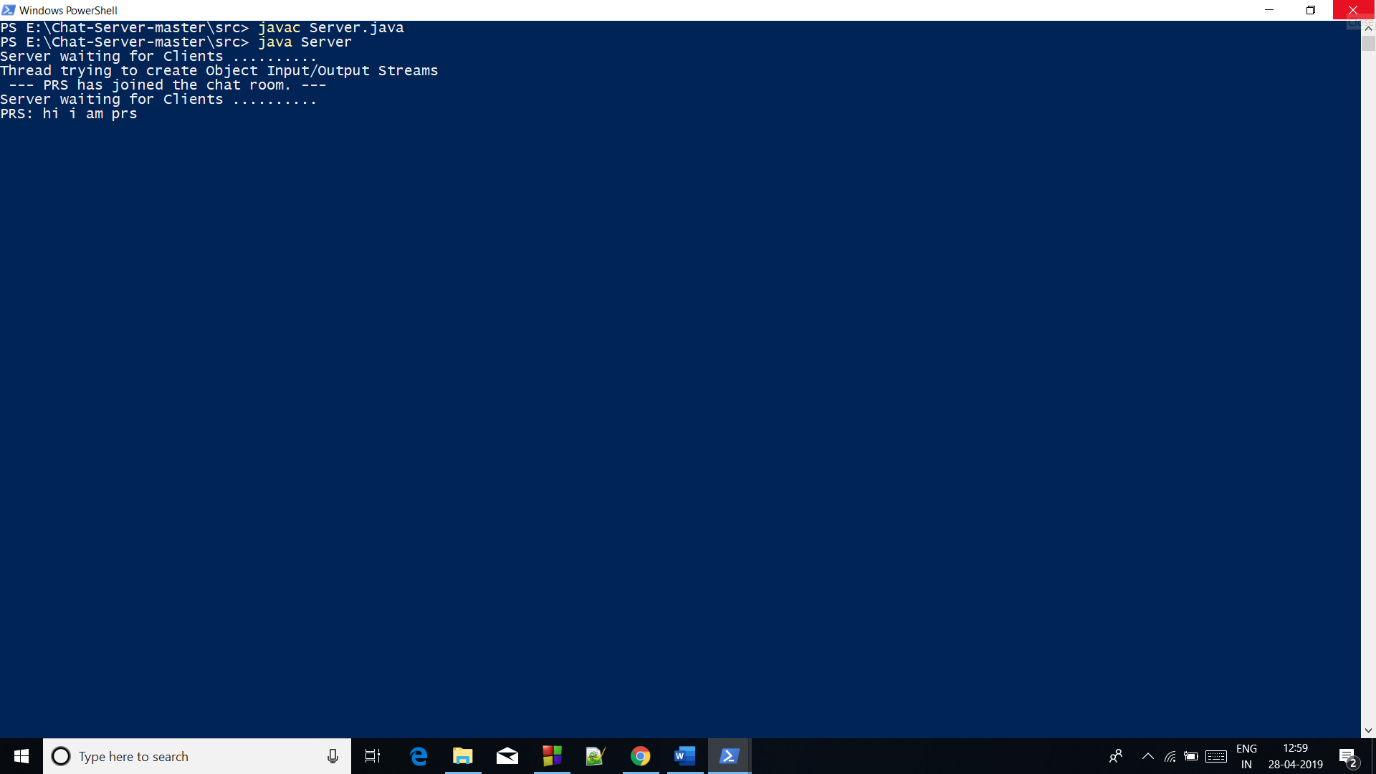
}

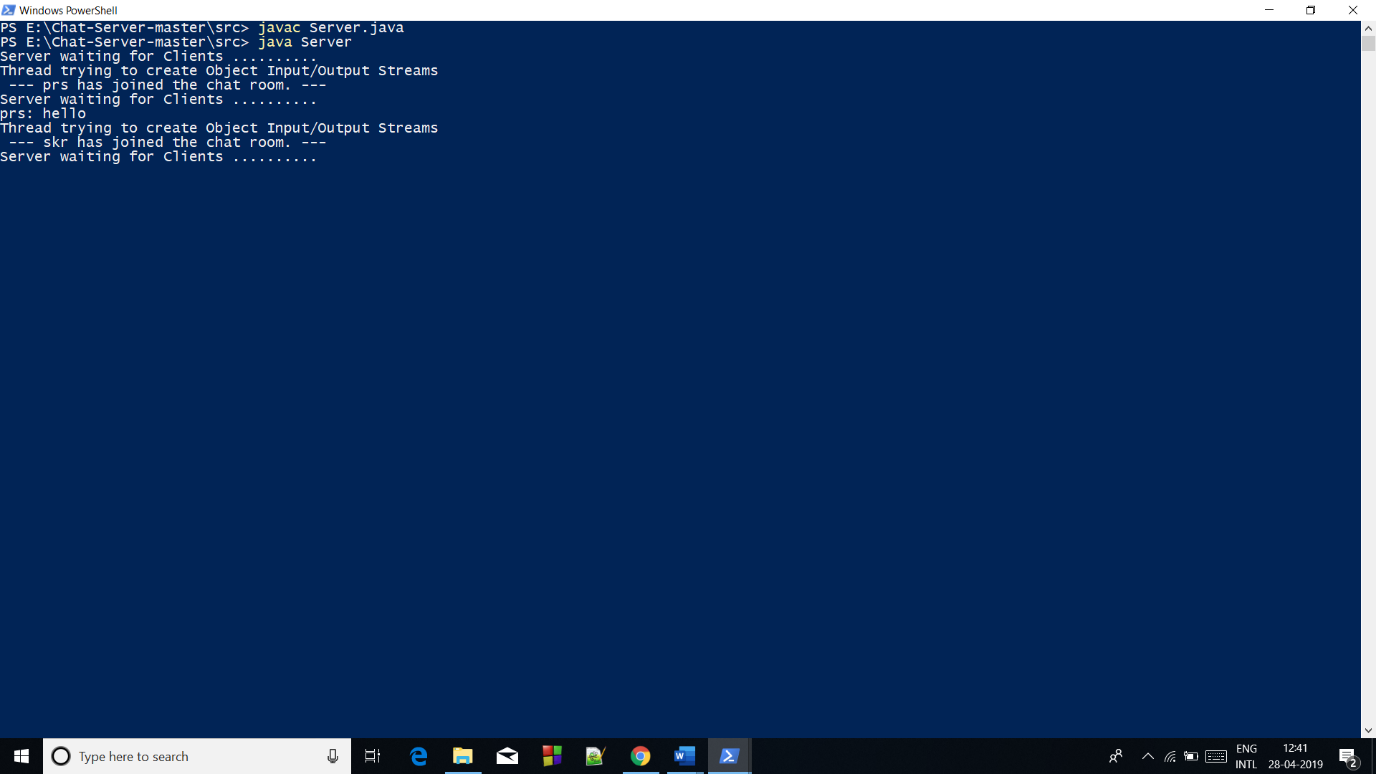
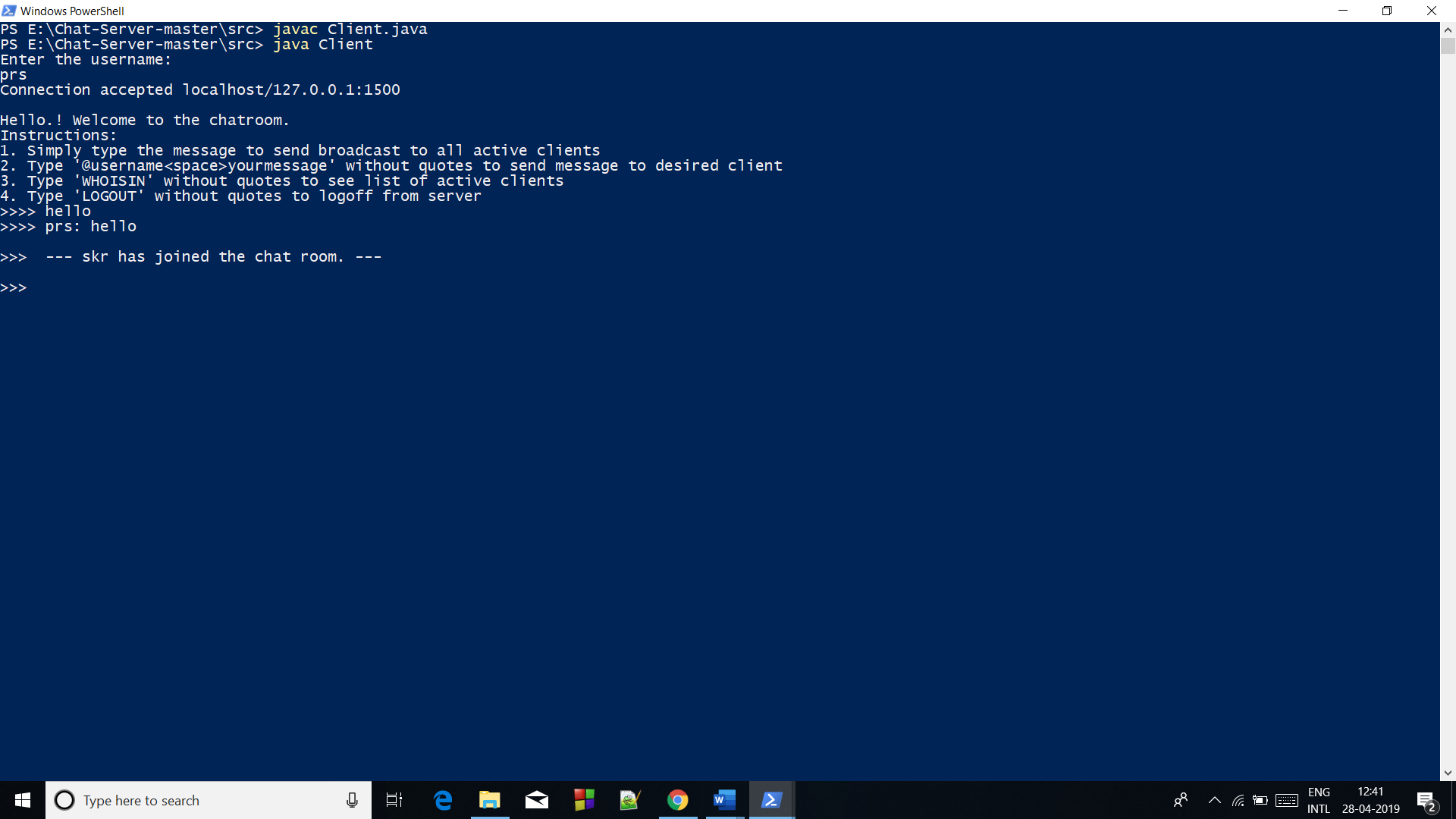
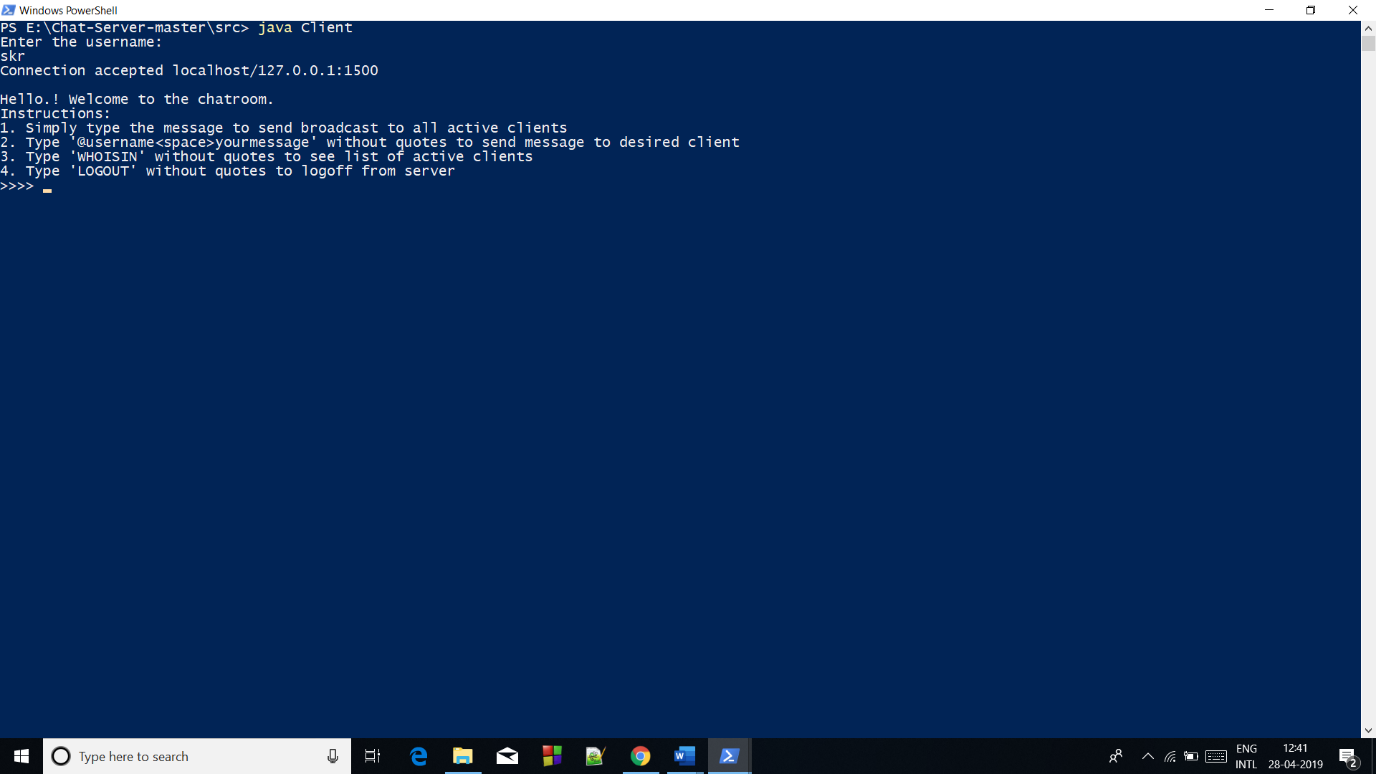
}

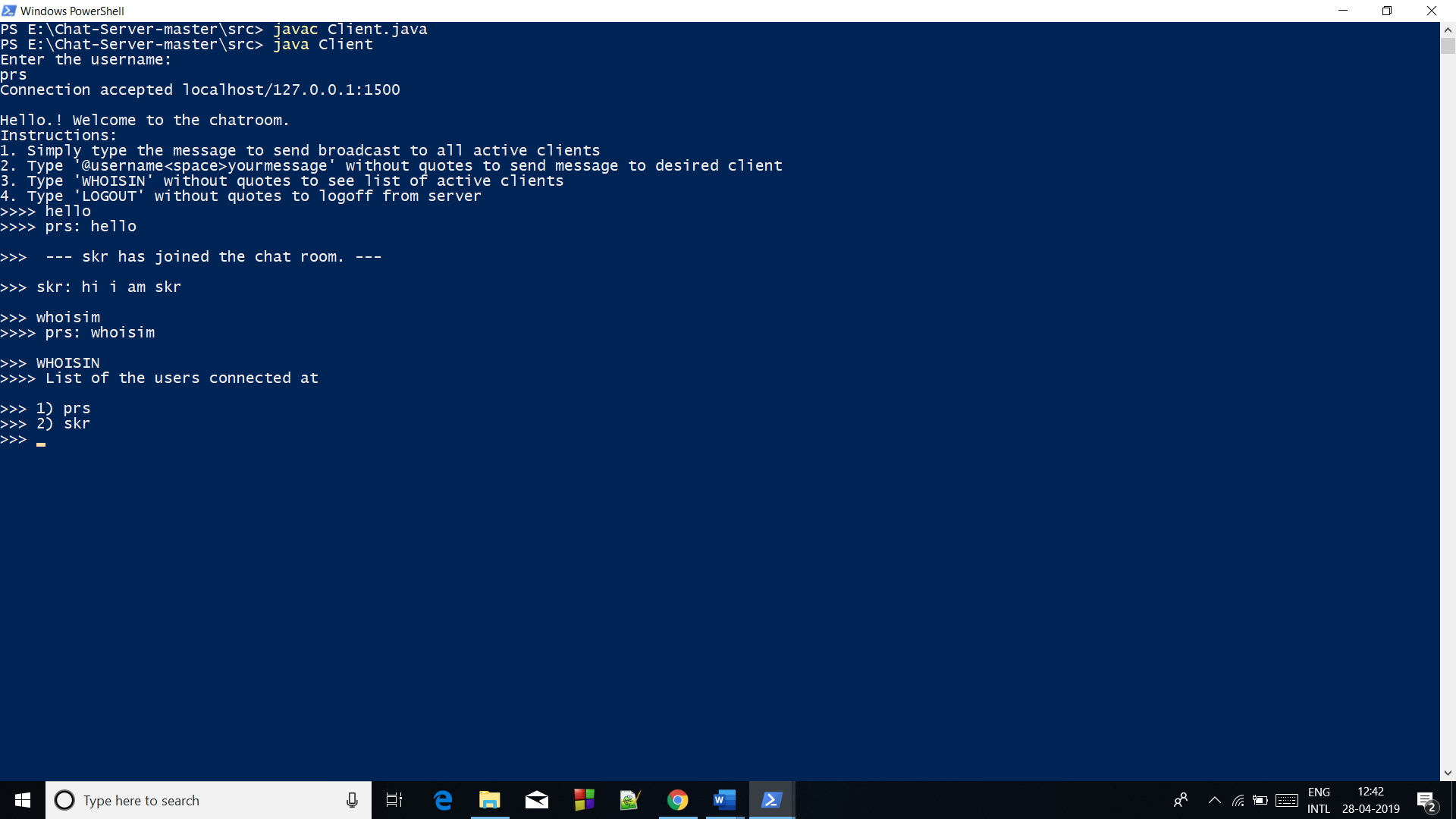
**SCREENSHOTS:**

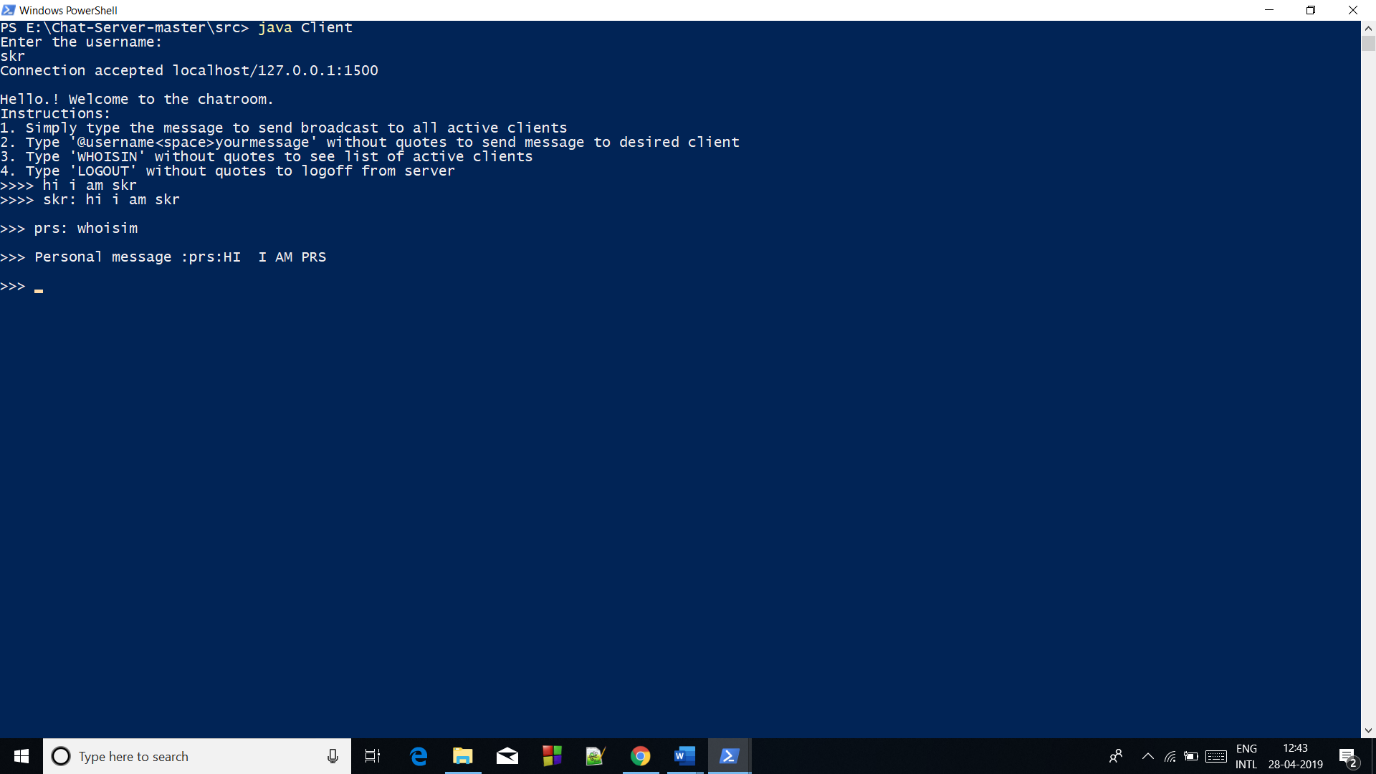
Server Created: -

Client PRS joined: -

Client Broadcast Message: -

Another Client Joined Server: -

Client Command for getting List of User connected: -

Client Sent Personal Message to another Client: -

Client Logout: -