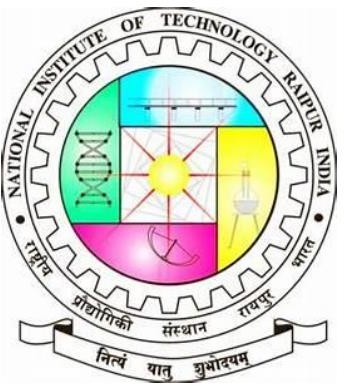
National Institute of Technology, Raipur

**Department of Computer Science and Engineering**



**Lab Report Distributed Systems Lab**

|  |  |
| --- | --- |
| **Submitted by:** | **Kethavath Sai Munishwar** |
| **Semester:** | **VII** |
| **Roll Number:** | **20115046** |

# EXPERIMENT-1

**Aim:** Implement concurrent echo client server application in java

**Theory:**

TCP: Transmission Control Protocol (TCP) is a communications standard that enables application programs and computing devices to exchange messages over a network. It is designed to send packets across the internet and ensure the successful delivery of data and messages over networks.

UDP: The User Datagram Protocol, or UDP, is a communication protocol used across the Internet for especially time-sensitive transmissions such as video playback or DNS lookups. It speeds up communications by not formally establishing a connection before data is transferred.

## Code:

**Tcp Server**

import java.io.\*; import java.net.\*; public class TcpServer {

public static void main(String[] args) throws Exception

{

ServerSocket ss=new ServerSocket(8088); System.out.println("server is ready!"); Socket ls=ss.accept();

while (true){

System.out.println("Client Port is "+ls.getPort());

//READING DATA FROM CLIENT

InputStream is=ls.getInputStream(); byte data[]=new byte[50]; is.read(data);

String mfc=new String(data);

//mfc: message from client mfc=mfc.trim();

String mfs="Hello:"+mfc;

//mfs: message from server

//SENDING MSG TO CLIENT

OutputStream os=ls.getOutputStream(); os.write(mfs.getBytes());

}

}

}

## Tcp Client

import java.net.\*; import java.io.\*; class TcpClient {

public static void main(String[] args) throws Exception

{

System.out.println("connecting to server"); Socket cs=new Socket("localhost",8088); BufferedReader br=new BufferedReader(new

InputStreamReader( System.in));

System.out.println("The Local Port "+cs.getLocalPort()+"\nThe Remote Port"+cs.getPort());

System.out.println("The Local socket is "+cs); System.out.println("Enter your name");

String str=br.readLine();

//SENDING DATA TO SERVER

OutputStream os=cs.getOutputStream(); os.write(str.getBytes());

//READING DATA FROM SERVER

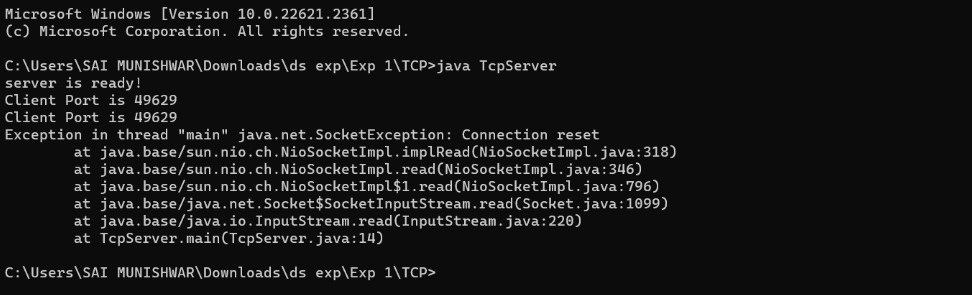
InputStream is=cs.getInputStream(); byte data[]=new byte[50]; is.read(data);

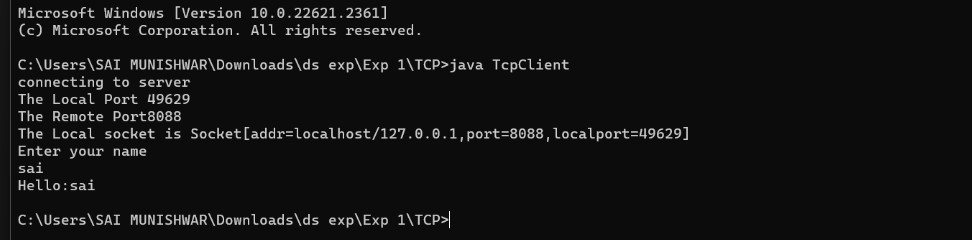
//PRINTING MESSAGE ON CLIENT CONSLOE

String mfs=new String(data); mfs=mfs.trim(); System.out.println(mfs);

}

}





## UDPServer

import java.net.\*; class UDPServer{

public static void main(String[] args) throws Exception{ byte buff[]=new byte[1024];

DatagramSocket ds =new DatagramSocket(8088); DatagramPacket p=new

DatagramPacket(buff,buff.length);

System.out.println("Server ready :"); ds.receive(p);

String msg = new String(

p.getData(),0,p.getLength()).trim();

String str = "Hello "+new String(buff); buff = str.getBytes();

ds.send(new DatagramPacket(buff,buff.length,InetAddress.getLocalHost(),808 9));

System.out.println("Msg received "+msg);

}

}

## UDPClient

import java.net.\*; import java.io.\*;

class UDPClient{

public static void main(String[] args) throws Exception

{

byte[] buff=new byte[1024];

DatagramSocket ds = new DatagramSocket(8089);

DatagramPacket p=new DatagramPacket(buff,buff.length);

BufferedReader br=new BufferedReader(new InputStreamReader(

System.in)); System.out.print("Enter your name:"); String msg = br.readLine();

buff = msg.getBytes();

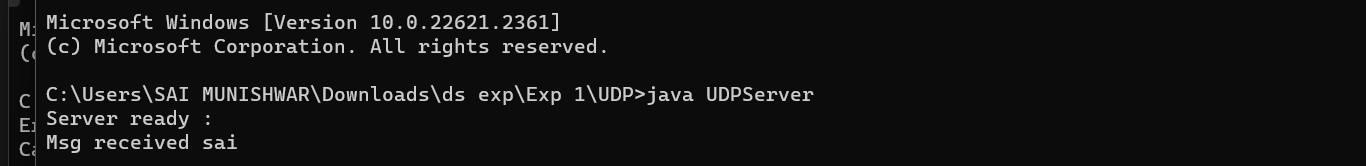
ds.send(new DatagramPacket(buff,buff.length, InetAddress.getLocalHost(),8088));

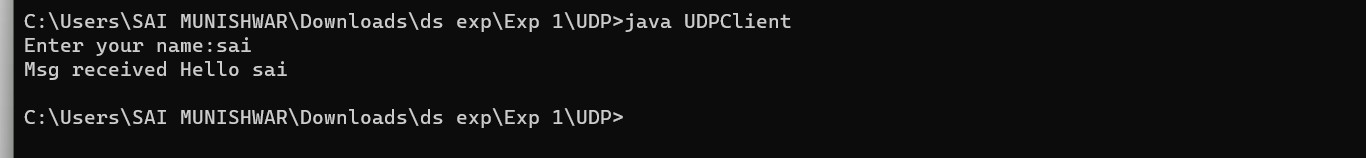
ds.receive(p); msg = new String(

p.getData(),0,p.getLength()).trim(); System.out.println("Msg received "+msg);

}

}





# EXPERIMENT-2

### **Aim:** Implement distributed Chat server using tcp sockets in java

**Theory**: A chat server is a computer dedicated to providing the processing power to handle and maintain chatting and its users. For example, there are thousands of dedicated servers set up for IRC (Internet Relay Chat), each of these servers are considered a chat server.

**Code**:

**ServerApp**

import java.io.InputStream; import java.net.ServerSocket; import java.net.Socket;

public class ServerApp implements Runnable{

/\*\*

\* @param args

\*/

public static Socket s=null; public static int i=1;

public static String clientName = "";

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

// TODO Auto-generated method stub ServerSocket ss = new ServerSocket(8089); ServerApp sa = new ServerApp();

Thread t; try{

"+i);

while(true){

System.out.println("Waiting for client

s = ss.accept(); i++;

t = new Thread(sa); t.start();

}

}catch (Exception e) {

// TODO: handle exception

}

finally{

ss.close();

}

}

@Override

public void run() {

// TODO Auto-generated method stub

try

{

}

InputStream is = s.getInputStream(); byte[] b = new byte[1024]; is.read(b);

clientName="";

clientName = new String(b).trim();

catch (Exception e)

{

e.printStackTrace();

}

new ChatGUI(s,clientName);

}

}

## ClientApp

import java.io.BufferedReader; import java.io.InputStreamReader; import java.io.OutputStream; import java.net.Socket;

public class ClientApp {

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

System.out.print("Enter your name:"); BufferedReader br = new BufferedReader(new

InputStreamReader(System.in));

String name = br.readLine();

Socket s = new Socket("localhost",8089); OutputStream os = s.getOutputStream(); os.write(name.getBytes());

new ChatGUI(s,"Admin");

}

}

## ChatGUI

import java.awt.\*;

import java.io.IOException; import java.io.InputStream; import java.io.OutputStream;

import java.net.Socket;

import java.net.SocketException; import javax.swing.\*;

public class ChatGUI extends JFrame implements ActionListener

{

private static final long serialVersionUID = 1L; Socket s;

JButton button; JTextArea ta1, ta2; String msg = "", title;

JScrollPane scrollPane1, scrollPane2; InputStream is;

OutputStream os;

ChatGUI(Socket x, String str) { s = x;

title = str;

button = new JButton("SEND"); ta1 = new JTextArea(5, 20); ta2 = new JTextArea(5, 20); ta1.setEditable(false);

scrollPane1 = new JScrollPane(ta1); scrollPane2 = new JScrollPane(ta2); setLayout(new FlowLayout()); add(scrollPane1); add(scrollPane2);

add(button); button.addActionListener(this); setSize(300, 300); setVisible(true);

setDefaultCloseOperation(DISPOSE\_ON\_CLOSE); setTitle("Messenger " + title);

try {

is = s.getInputStream();

os = s.getOutputStream();

} catch (IOException ioe) {

}

try { chat();

} catch (Exception e) {

// TODO Auto-generated catch block e.printStackTrace();

}

}

@SuppressWarnings("deprecation") public void chat() throws Exception {

while (true) { try {

byte data[] = new byte[50]; is.read(data);

msg = new String(data).trim(); ta1.append(title+": " + msg + "\n");

} catch (SocketException se) { JOptionPane.showMessageDialog(this,

"Disconnected from "+title);

this.dispose(); Thread.currentThread().stop();

}

}

}

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub msg = ta2.getText();

try {

os.write(msg.getBytes());

} catch (IOException ioe) {

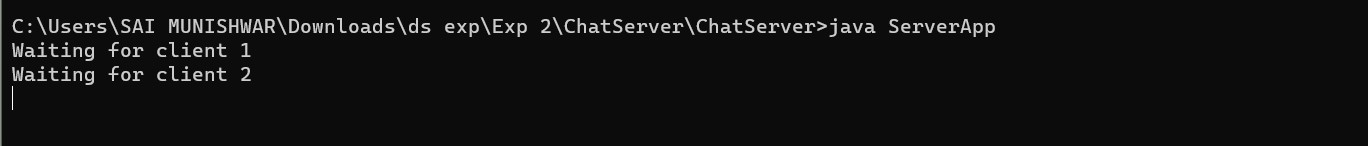
// TODO Auto-generated catch block ioe.printStackTrace();

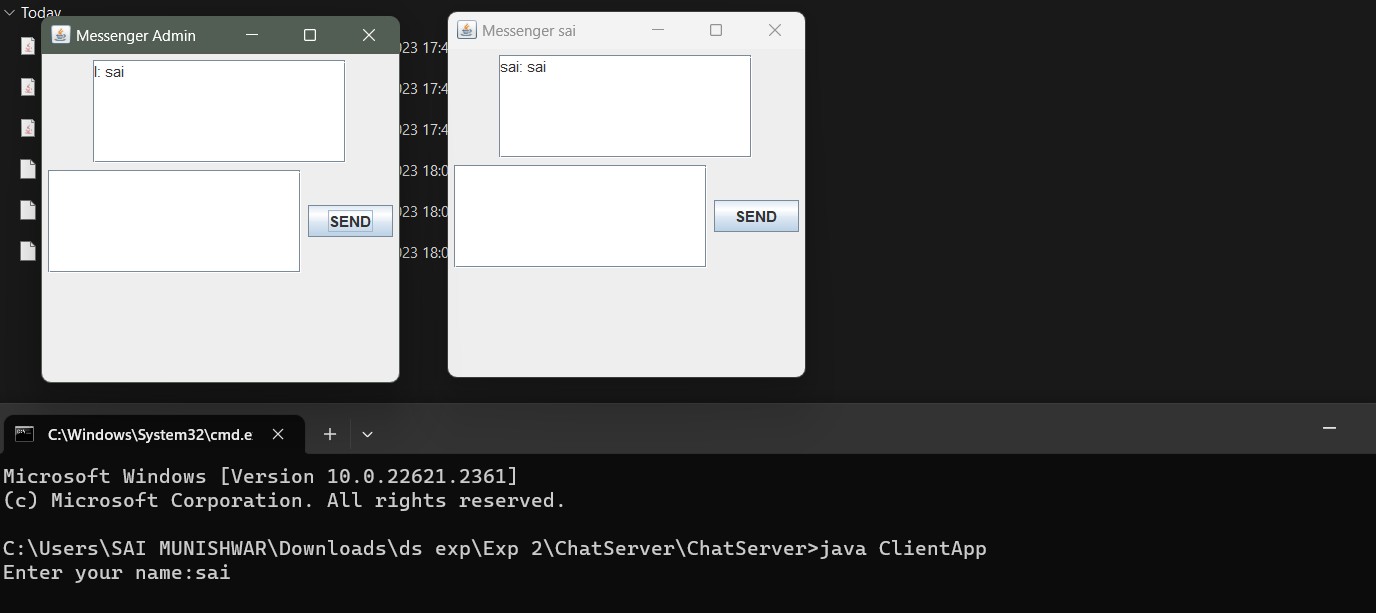
}

ta1.append("I: " + msg + "\n"); ta2.setText("");

}

}





# EXPERIMENT-3

### **Aim:** Implement concurrent Day- Time client server application in java

**Theory**: A daytime server listens for client requests on port 13. When it receives a message from a client, a daytime server replies to that client with its current date and time. A daytime client thus simply sends a message to a daytime server and then displays its reply, which will be the date and time on that server's host.

**Code:**

**Server\_DT**

import java.net.\*; import java.io.\*; import java.util.Date; public class Server\_DT

{

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

ServerSocket ss=new ServerSocket(5000); System.out.println("The server has reserved port No:"+ss.getLocalPort()+" for this Service"); Socket cs=ss.accept(); System.out.println("Client with IP

Address"+cs.getInetAddress()+"has communicated via port No:"+cs.getPort());

Date d= new Date();

String s="Current Date & Time on Server is:"+d; PrintWriter toclient =new PrintWriter(cs.getOutputStream(),true); toclient.print(s);

toclient.close(); cs.close();

ss.close();

}

}

## Client\_DT

import java.net.\*; import java.io.\*;

public class Client\_DT

{

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

Socket cs=new Socket("LocalHost",5000);

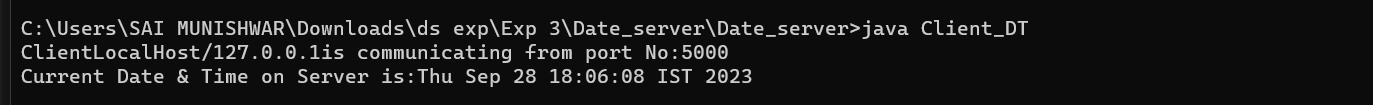
System.out.println("Client"+cs.getInetAddress()+"is communicating from port No:"+cs.getPort());

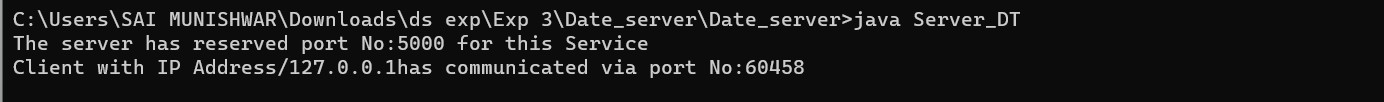
BufferedReader fromserver=new BufferedReader(new InputStreamReader(cs.getInputStream())); System.out.println(fromserver.readLine()); fromserver.close();

cs.close();

}

}





# EXPERIMENT-4

### **Aim**: Configure following options on server socket and tests them: SO\_KEEPALIV E, SO\_LINGER,SO\_SNDBUF, SO\_RCV BUF, TCP\_NODELAY

**Theory**: - SO\_KEEPALIVE:Leave received out-of-band data inline

* SO\_LINGER:Linger on close if data to send
* SO\_SNDBUF:Send buffer size
* SO\_RCV BUF:Receive buffer size
* TCP\_NODELAY:specifies whether the server disables the delay of sending successive small packets on the network

**Code**:

**Server**

import java.io.BufferedReader; import java.io.IOException; import java.io.InputStreamReader; import java.io.PrintWriter; import java.net.ServerSocket; import java.net.Socket;

import java.net.SocketException;

public class Server {

public static void main(String[] args) { try {

// Create a server socket on port 12345 ServerSocket serverSocket = new

ServerSocket(12345);

System.out.println("Server listening on port

12345...");

// Accept client connection

Socket clientSocket = serverSocket.accept(); System.out.println("Client connected: " +

clientSocket.getInetAddress().getHostAddress());

// Enable SO\_KEEPALIVE (TCP keep-alive) clientSocket.setKeepAlive(true);

seconds

// Enable SO\_LINGER with a linger time of 5 clientSocket.setSoLinger(true, 5);

// Set SO\_SNDBUF to 64KB (64 \* 1024 bytes) int sendBufferSize = 64 \* 1024;

clientSocket.setSendBufferSize(sendBufferSize);

// Get input and output streams BufferedReader reader = new BufferedReader(new

InputStreamReader(clientSocket.getInputStream())); PrintWriter writer = new

PrintWriter(clientSocket.getOutputStream(), true);

message);

// Read and send data String message;

while ((message = reader.readLine()) != null) { System.out.println("Received from client: " +

// Send response back to the client writer.println("Server response: " + message);

}

// Close the socket and server socket when done clientSocket.close();

serverSocket.close();

} catch (SocketException e) {

// Handle socket-related exceptions e.printStackTrace();

} catch (IOException e) {

// Handle IO-related exceptions e.printStackTrace();

}

}

}

## Client

import java.io.BufferedReader; import java.io.IOException; import java.io.InputStreamReader; import java.io.PrintWriter; import java.net.Socket;

import java.net.SocketException;

public class Client {

public static void main(String[] args) { try {

// Connect to the server on localhost:12345 Socket socket = new Socket("localhost", 12345);

// Enable SO\_KEEPALIVE (TCP keep-alive) socket.setKeepAlive(true);

seconds

// Enable SO\_LINGER with a linger time of 5 socket.setSoLinger(true, 5);

// Set SO\_SNDBUF to 64KB (64 \* 1024 bytes) int sendBufferSize = 64 \* 1024; socket.setSendBufferSize(sendBufferSize);

// Get input and output streams BufferedReader reader = new BufferedReader(new

InputStreamReader(socket.getInputStream())); PrintWriter writer = new

PrintWriter(socket.getOutputStream(), true);

// Send data to the server writer.println("Hello, server!");

response);

// Receive response from the server String response = reader.readLine();

System.out.println("Received from server: " +

// Close the socket when done socket.close();

} catch (SocketException e) {

// Handle socket-related exceptions

e.printStackTrace();

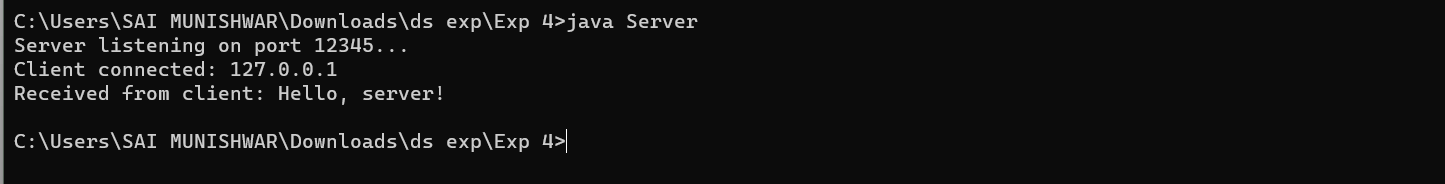
} catch (IOException e) {

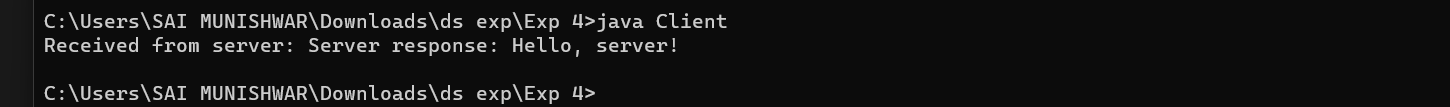
// Handle IO-related exceptions e.printStackTrace();

}

}

}





# EXPERIMENT-5

### **Aim:** Write a program to Incrementing a counter in shared memory in JAVA

**Theory**: Incrementing a counter in shared memory is a common operation in concurrent programming, particularly in scenarios where multiple threads or processes need to access and update a shared counter.

**Code:**

import java.util.Scanner;

public class Unsynchronized\_Counter{ static class Counter{

int count; void inc()

{

count=count+1;

}

int getCount()

{

return count;

}

}

static Counter counter;

static int numberofincrements;

static class IncrementerThread extends Thread

{

public void run()

{

for(int i=0; i<numberofincrements;i++)

{

counter.inc();

}

}

}

public static void main(String[] args)

{

Scanner in= new Scanner(System.in); while(true)

{

to run");

System.out.println();

System.out.println("how many threads do you want int numberofthreads=in.nextInt();

if(numberofthreads<=0) break;

do

{

System.out.println();

System.out.println("How many times should each

thread increment the counter");

numberofincrements=in.nextInt(); if(numberofthreads<=0)

{

System.out.println("No of increments should be positive");

}

}while(numberofincrements<=0); System.out.println();

System.out.println("using"+numberofthreads+"threads");

System.out.println("each thread increment the counter"+numberofincrements+"times");

System.out.println(); System.out.println("working"); System.out.println(); IncrementerThread[] workers= new

IncrementerThread[numberofthreads];

counter=new Counter();

for(int i=0; i<numberofthreads;i++) workers[i]=new IncrementerThread();

for(int i=0;i<numberofthreads;i++) workers[i].start();

for(int i=0; i<numberofthreads;i++){ try{

workers[i].join();

}

catch(InterruptedException e){

}

}

System.out.println("the final value of the counter should be"+(numberofincrements\*numberofthreads));

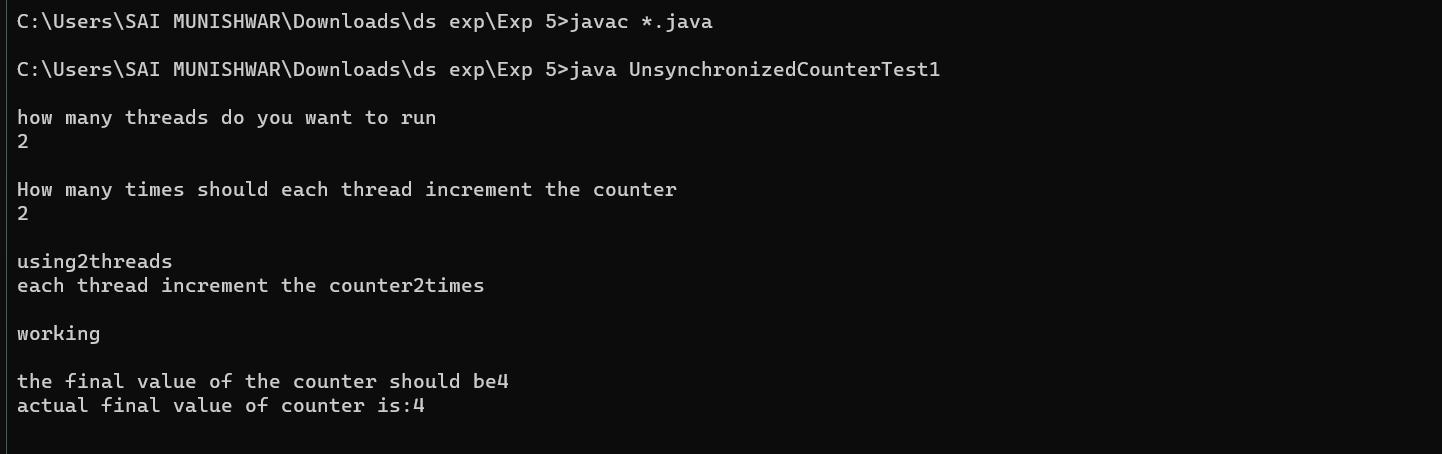
System.out.println("actual final value of counter is:" +counter.getCount());

System.out.println(); System.out.println();

}

}

}



# EXPERIMENT-7

### **Aim**: Write a program to Implement Java RMI‟ mechanism for accessing methods of remote systems”

**Theory**: Remote Method Invocation (RMI) is an application programming interface (API) in the Java programming language and development environment. It allows objects on one computer or Java Virtual Machine (JVM) to interact with objects running on a different JVM in a distributed network.

#### Code :

**AddRem :**

//define remote interface import java.rmi.\*;

public interface AddRem extends Remote

{

public int addNum(int a ,int b) throws RemoteException;

}

**AddRemImpl:**

//implementation of interface import java.rmi.\*;

import java.rmi.server.UnicastRemoteObject;

public class AddRemImpl extends UnicastRemoteObject implements AddRem

{

public AddRemImpl() throws RemoteException{} public int addNum(int a, int b)

{

return(a+b);

}

}

## AddServer :

**import** java.rmi.\*;

**import** java.net.\*;

**public class** AddServer {

**public static void** main(String args[]) {

#### try {

AddRemImpl locobj = **new** AddRemImpl();

Naming.*rebind*("rmi:///AddRem", locobj);

} **catch** (RemoteException re) { re.printStackTrace();

} **catch** (MalformedURLException mfe) { mfe.printStackTrace();

}

}

}

## AddClient :

//Client program **import** java.rmi.\*; **import** java.net.\*; **import** java.io.\*; **import** java.util.\*;

**public class** AddClient {

**public static void** main(String arges[]) { String host = "localhost";

Scanner sc = **new** Scanner(System.***in***); System.***out***.println("Enter first parameter"); **int** a = sc.nextInt(); System.***out***.println("Enter second parameter"); **int** b = sc.nextInt();

#### try {

AddRem remobj = (AddRem) Naming.*lookup*("rmi://"

+ host + "/AddRem");

System.***out***.println("Sum of two parameters : " + remobj.addNum(a, b));

} **catch** (RemoteException re) { re.printStackTrace();

} **catch** (NotBoundException nbe) { nbe.printStackTrace();

} **catch** (MalformedURLException mfe) { mfe.printStackTrace();

}

}

}

