Assignment No:5

Name:Prathmesh Pattewar

PRN:12420193

Title: Client server programs using UDP Berkeley socket primitives.

Problem Statement: Write the client server programs using UDP Berkeley socket primitives for wired /wireless network for following a. to say Hello to Each other b. Calculator (Trigonometry).

Course Objective: To learn transport layer and application layer protocols used in the Internet.

Course Outcome: Develop Client-Servers by the means of correct standards, protocols and technologies.

Tools Required: Eclipse, Java.

Theory:

1. Introduction

The term *network programming* refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.

The java.net package of the J2SE APIs contains a collection of classes and interfaces that provide the low-level communication details, allowing you to write programs that focus on solving the problem at hand.

The java.net package provides support for the two common network protocols:

- **TCP:** TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.
- **UDP:** UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.

2. What is UDP?

UDP (User Datagram Protocol) is a connectionless, lightweight transport protocol that allows applications to send and receive datagrams without establishing a connection. It is faster but less reliable than TCP, as it does not guarantee delivery, ordering, or error checking.

2.1 Berkeley Sockets Model

The Berkeley Sockets API is a low-level network communication interface originally developed for UNIX systems. Java's java.net package provides a high-level abstraction of Berkeley Sockets. In UDP communication, Berkeley Sockets provide:

- **No connection establishment**: Send and receive packets independently

2.3 UDP Communication Flow in Java

Server Side:

1. Create DatagramSocket on a fixed port.
2. Wait for incoming packets using receive().
3. Read data from DatagramPacket.
4. Optionally send a response using send().

Client Side:

1. Create DatagramSocket.
2. Create DatagramPacket with message and server details.
3. Send message using send().
4. Wait for reply using receive().

3. Java Classes Used in UDP (Berkeley Sockets Model)

Class Name	Package	Description
DatagramSocket	java.net	Used to send and receive UDP packets. It acts as the communication endpoint.
DatagramPacket	java.net	Represents a UDP datagram packet. Carries data to be sent or received.
InetAddress	java.net	Represents an IP address (IPv4 or IPv6). Used to specify sender/receiver host.
IOException	java.io	Handles input/output errors that may occur during socket communication.
UnknownHostException	java.net	Thrown when IP address of a host cannot be determined (usually on client side).
SocketException	java.net	Thrown when there is an error creating or accessing a DatagramSocket.

Program:

//**********trignoclient.Java**********

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

public class trignoclient {
    public static void main(String[] args) {
        try {
            DatagramSocket socket = new DatagramSocket();
```

```
            InetAddress serverAddress =
InetAddress.getByName("localhost");
            Scanner sc = new Scanner(System.in);
            byte[] sendBuffer = new byte[1024];
            byte[] receiveBuffer = new byte[1024];

            System.out.println("Enter operation (sin, cos, tan) or
'exit' to quit:");
            String operation = sc.nextLine();
            sendBuffer = operation.getBytes();
            DatagramPacket opPacket = new
DatagramPacket(sendBuffer, sendBuffer.length, serverAddress,
9876);
            socket.send(opPacket);

            if (!operation.equalsIgnoreCase("exit")) {
                System.out.println("Enter angle in degrees:");
                String angle = sc.nextLine();
                sendBuffer = angle.getBytes();
                DatagramPacket anglePacket = new
DatagramPacket(sendBuffer, sendBuffer.length, serverAddress,
9876);
                socket.send(anglePacket);

                DatagramPacket receivePacket = new
DatagramPacket(receiveBuffer, receiveBuffer.length);
                socket.receive(receivePacket);
                String result = new
String(receivePacket.getData(), 0, receivePacket.getLength());
                System.out.println("Server says: " + result);
            }

            socket.close();
            sc.close();
        } catch (Exception e) {
            e.printStackTrace();
```

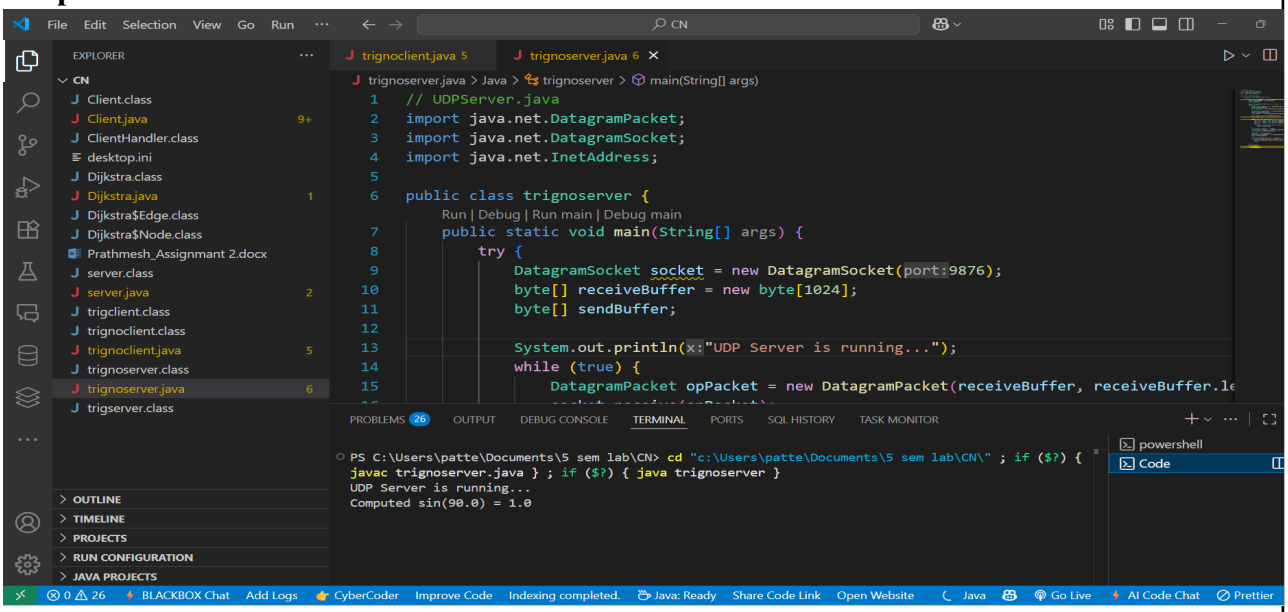
```
        }
    }
}
```

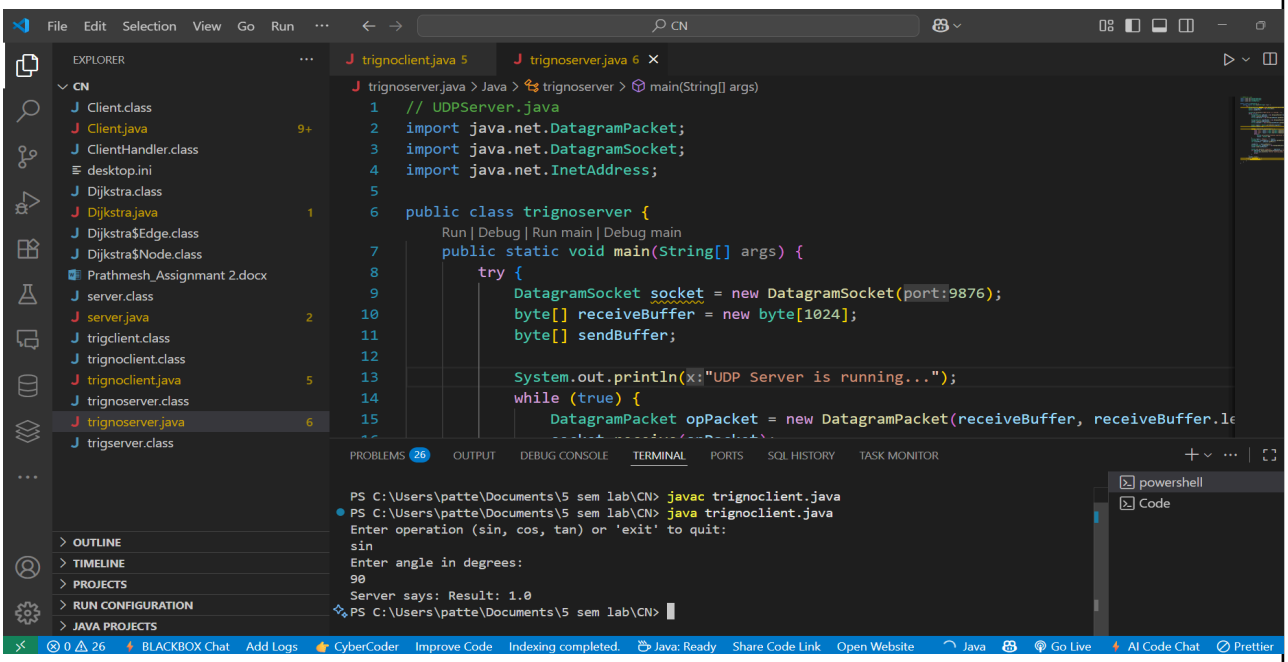
//**********trignoserver.Java**********

```
// UDPServer.java
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class trignoserver {
    public static void main(String[] args) {
        try {
            DatagramSocket socket = new DatagramSocket(9876);
            byte[] receiveBuffer = new byte[1024];
            byte[] sendBuffer;

            System.out.println("UDP Server is running...");
            while (true) {
                DatagramPacket opPacket = new
DatagramPacket(receiveBuffer, receiveBuffer.length);
                socket.receive(opPacket);
                String operation = new String(opPacket.getData(),
0, opPacket.getLength()).trim();

                DatagramPacket anglePacket = new
DatagramPacket(receiveBuffer, receiveBuffer.length);
                socket.receive(anglePacket);
                String angleStr = new
String(anglePacket.getData(), 0, anglePacket.getLength()).trim();

                double angle = Double.parseDouble(angleStr);
                double result = 0;

                switch (operation.toLowerCase()) {
                    case "sin": result =
```

```
Math.sin(Math.toRadians(angle)); break;
                    case "cos": result =
Math.cos(Math.toRadians(angle)); break;
                    case "tan": result =
Math.tan(Math.toRadians(angle)); break;
                    default:
                        System.out.println("Unknown operation
received: " + operation);
                        result = Double.NaN;
                }
                String reply = "Result: " + result;
                InetAddress clientAddress = opPacket.getAddress();
                int clientPort = opPacket.getPort();

                sendBuffer = reply.getBytes();
                DatagramPacket sendPacket = new
DatagramPacket(sendBuffer, sendBuffer.length, clientAddress,
clientPort);
                socket.send(sendPacket);

                System.out.println("Computed " + operation + "(" +
angle + ") = " + result);
                if (operation.equalsIgnoreCase("exit")) {
                    System.out.println("Server shutting down...");
                    break;
                }
            }
            socket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Output:





Conclusion: The client-server programs implemented with UDP Berkeley socket primitives illustrate the characteristics and usage of connectionless communication in computer networks.Unlike TCP, UDP does not establish a dedicated connection, enabling faster but less reliable message exchanges, which is well-suited for simple and time-sensitive applications like greeting messages and calculator services.