**Name : Atharva Salitri**

**Division : TY CSAI-B**

**Batch : B2**

**Roll No. : 029**

**PRN : 12310120**

**Subject : Computer Network Technology**

**Assignment No:5**

**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.

1. **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.
* **Datagram-based communication**: Each message is sent as an individual packet.
* **Best-effort delivery**: No guarantee of packet arrival or order.

**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().
5. **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. |

**Conclusion:** Hence, we have Successfully wrote a client server programs using UDP Berkeley socket primitives for wired /wireless network to say Hello to Each other and Calculator (Trigonometry).

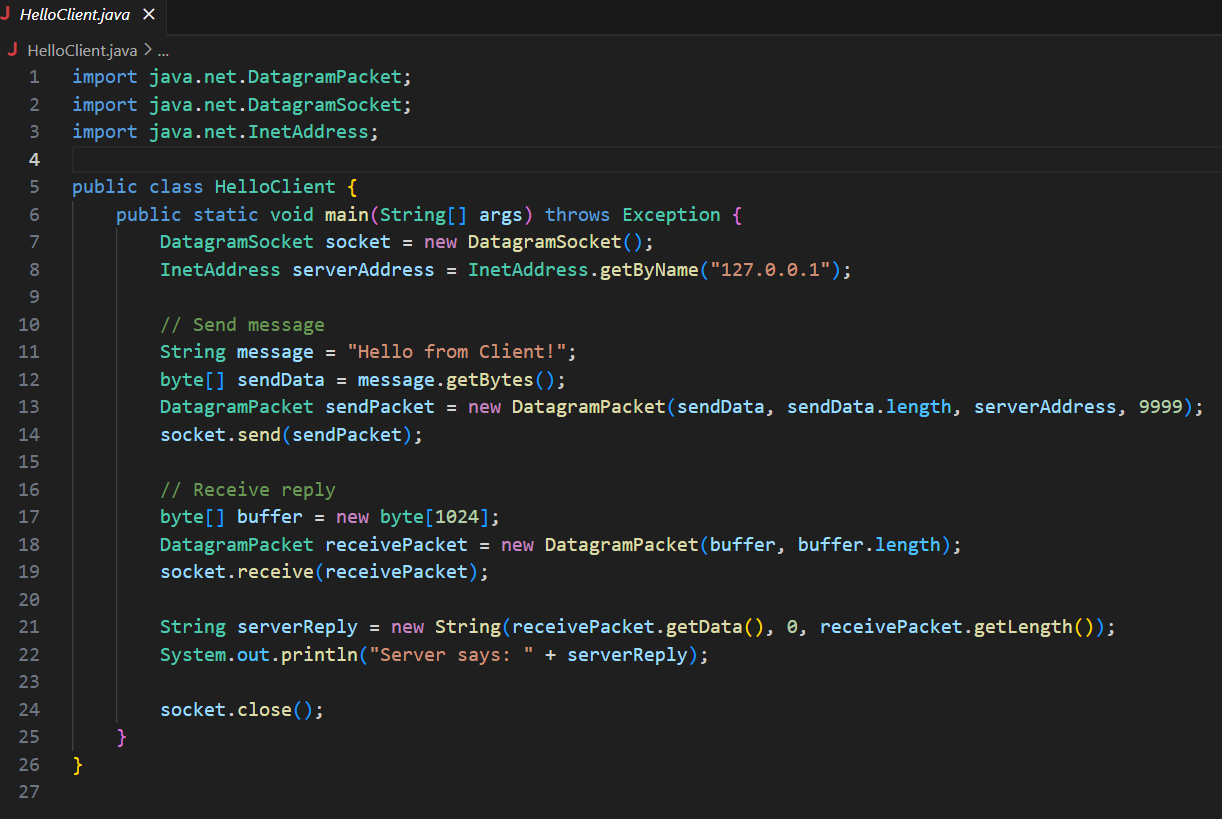
**Program:**

|  |
| --- |
| **//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Client.Java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  package ChatApplication;  import java.net.\*;  import java.util.Scanner;  public class Client {  public static void main(String[] args) {  try {  DatagramSocket clientSocket = new DatagramSocket(); // Create client socket  InetAddress serverAddress = InetAddress.getByName("localhost"); // Server IP  Scanner scanner = new Scanner(System.in);  byte[] sendData;  byte[] receiveData = new byte[1024];  while (true) {  // Ask client user to enter a message  System.out.print("You (Client): ");  String message = scanner.nextLine();  sendData = message.getBytes();  // Send message to server  DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, serverAddress, 9876);  clientSocket.send(sendPacket);  if (message.equalsIgnoreCase("bye")) {  System.out.println("You exited the chat.");  break;  }  // Receive server reply  DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);  clientSocket.receive(receivePacket);  String serverReply = new String(receivePacket.getData(), 0, receivePacket.getLength());  System.out.println("Server: " + serverReply);  if (serverReply.equalsIgnoreCase("bye")) {  System.out.println("Server exited the chat.");  break;  }  }  clientSocket.close();  } catch (Exception e) {  e.printStackTrace();  }  }  }  **//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Server.Java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  package ChatApplication;  import java.net.\*;  import java.util.Scanner;  public class Server {  public static void main(String[] args) {  try {  DatagramSocket serverSocket = new DatagramSocket(9876); // Server socket on port 9876  Scanner scanner = new Scanner(System.*in*);  byte[] receiveData = new byte[1024];  byte[] sendData;  System.*out*.println("Server is running...");  while (true) {  // Receive message from client  DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);  serverSocket.receive(receivePacket);  String clientMessage = new String(receivePacket.getData(), 0, receivePacket.getLength());  System.*out*.println("Client: " + clientMessage);  if (clientMessage.equalsIgnoreCase("bye")) {  System.*out*.println("Client exited the chat.");  break;  }  // Ask server user to type a reply  System.*out*.print("You (Server): ");  String reply = scanner.nextLine();  sendData = reply.getBytes();  // Send response to client  InetAddress clientAddress = receivePacket.getAddress();  int clientPort = receivePacket.getPort();  DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, clientAddress, clientPort);  serverSocket.send(sendPacket);  if (reply.equalsIgnoreCase("bye")) {  System.*out*.println("Server exited the chat.");  break;  }  }  serverSocket.close();  } catch (Exception e) {  e.printStackTrace();  }  }  } |
| **Output:**   |  |  | | --- | --- | | Server is running...  Client: Hello  You (Server): hi how are you  Client ended the chat.  Connection closed. | You (Client): Hello  Server: hi how are you  You (Client): bye  You ended the chat.  Connection closed. | |

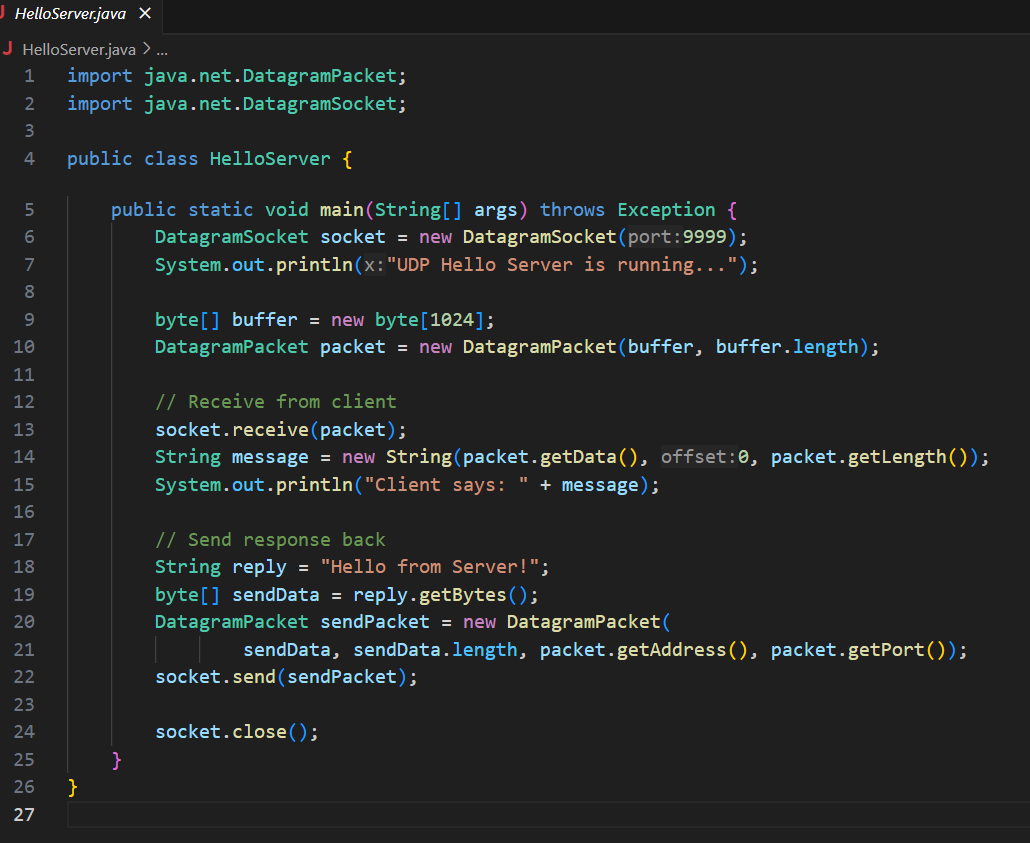
|  |
| --- |
| **//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Client.Java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  package Calculator;  import java.net.DatagramPacket;  import java.net.DatagramSocket;  import java.net.InetAddress;  import java.util.Scanner;  public class Client {  public static void main(String[] args) throws Exception {  DatagramSocket clientSocket = new DatagramSocket();  InetAddress IPAddress = InetAddress.*getByName*("localhost");  Scanner sc = new Scanner(System.*in*);  while (true) {  // Get user input  System.*out*.print("Enter operator (sin/cos/tan): ");  String op = sc.next();  System.*out*.print("Enter number: ");  double num = sc.nextDouble();  // Prepare data to send  String message = op + " " + num;  byte[] sendData = message.getBytes();  DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 9877);  clientSocket.send(sendPacket);  // Receive response  byte[] receiveData = new byte[1024];  DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);  clientSocket.receive(receivePacket);  String reply = new String(receivePacket.getData(), 0, receivePacket.getLength());  System.*out*.println(reply);  // Ask user if they want to continue  System.*out*.print("Do you want to continue? (yes/no): ");  String choice = sc.next();  if (!choice.equalsIgnoreCase("yes")) break;  }  clientSocket.close();  sc.close();  }  }  **//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Server.Java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  package Calculator;  import java.net.DatagramPacket;  import java.net.DatagramSocket;  import java.util.StringTokenizer;  public class Server {  public static void main(String[] args) throws Exception {  DatagramSocket serverSocket = new DatagramSocket(9877);  byte[] receiveData = new byte[1024];  byte[] sendData;  while (true) {  // Receive request from client  DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);  serverSocket.receive(receivePacket);  String input = new String(receivePacket.getData(), 0, receivePacket.getLength());  // Parse operation and number  StringTokenizer st = new StringTokenizer(input, " ");  String op = st.nextToken();  double num = Double.*parseDouble*(st.nextToken());  // Perform trigonometric operation  double result;  switch (op.toLowerCase()) {  case "sin": result = Math.*sin*(Math.*toRadians*(num)); break;  case "cos": result = Math.*cos*(Math.*toRadians*(num)); break;  case "tan": result = Math.*tan*(Math.*toRadians*(num)); break;  default: result = Double.*NaN*;  }  // Send result back to client  String output = "Result: " + result;  sendData = output.getBytes();  DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,  receivePacket.getAddress(), receivePacket.getPort());  serverSocket.send(sendPacket);  }  }  } |
| **Output:**  Execute Server followed by client  **Client Side:**  Enter operator (sin/cos/tan): sin  Enter number: 90  Result: 1.0  Do you want to continue? (yes/no): yes  Enter operator (sin/cos/tan): tan  Enter number: 90  Result: 1.633123935319537E16  Do you want to continue? (yes/no): no |

**Screenshots:**

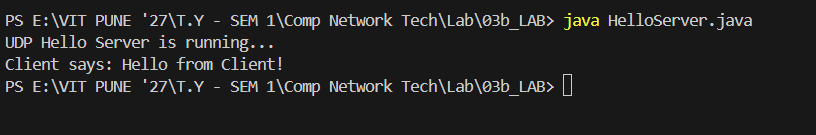
HelloClient.java

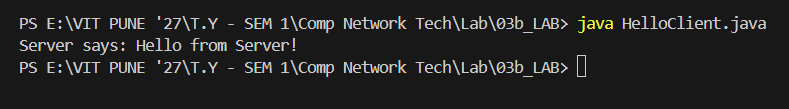


HelloServer.java

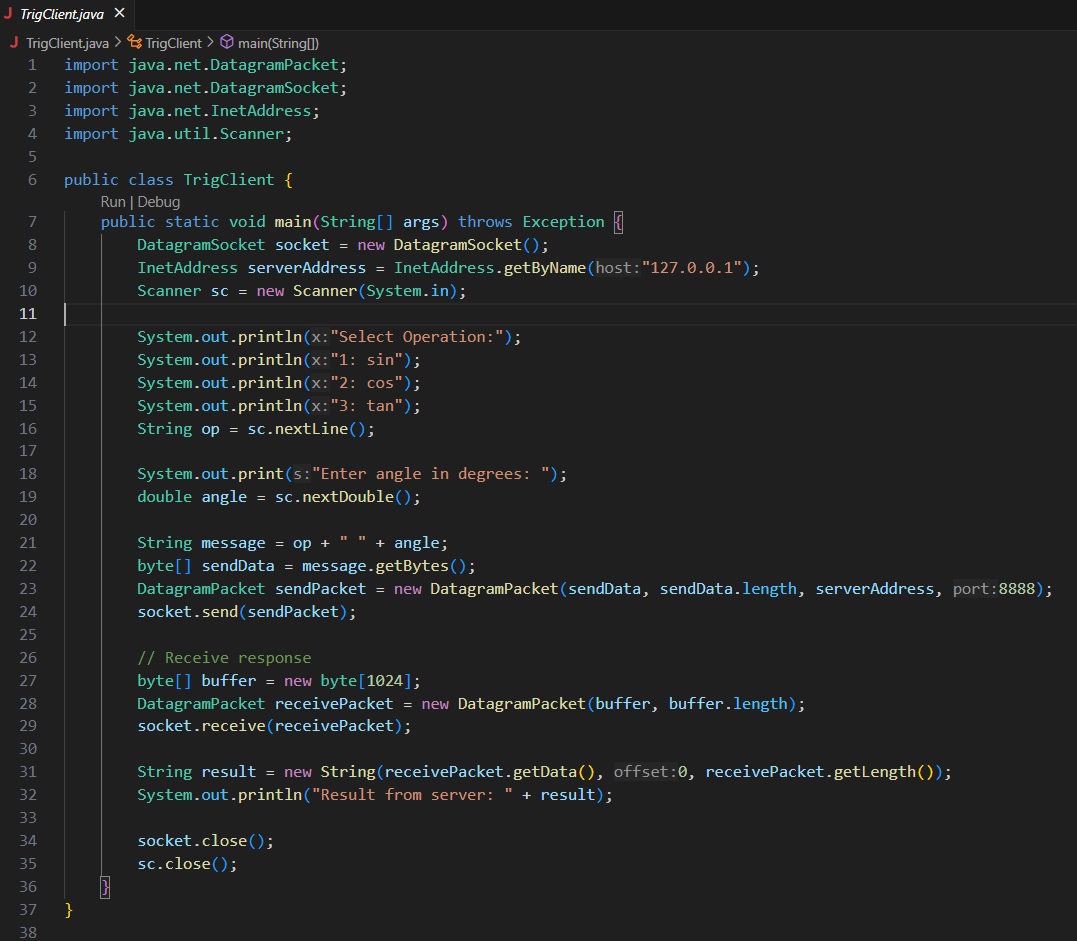


OUTPUT:

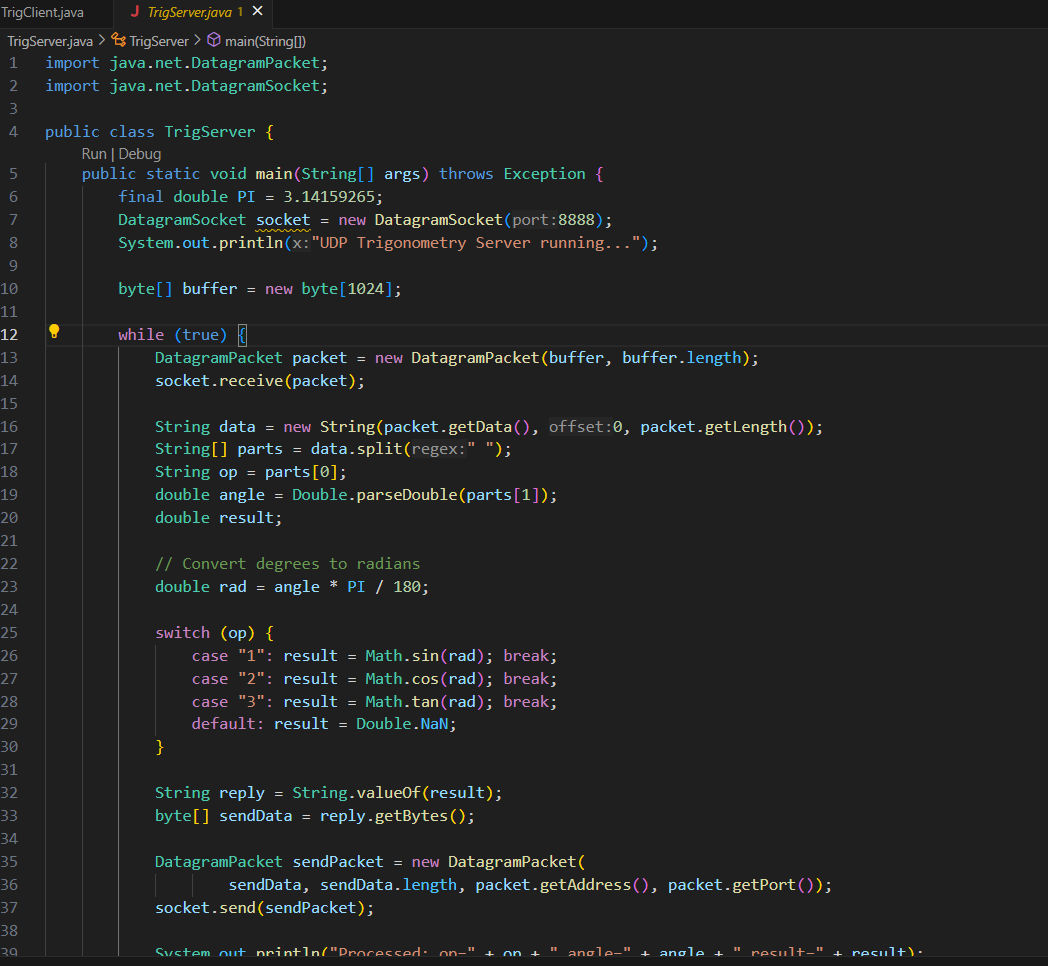




TrigClient.java



TrigServer.java



OUTPUT:

