

## ASSIGNMENT 9 : LO4

Aim:

To implement Socket Programming using UDP in Java - UDPServer, UDPClient

Theory:

UDP (User Datagram Protocol) is a connectionless protocol in the Transport layer of the OSI model. Unlike TCP (Transmission Control Protocol), UDP does not establish a connection before sending data and does not guarantee message delivery. It's faster than TCP because it avoids the overhead of connection management and acknowledgment, but it may lose packets or deliver them out of order. In Java, we can implement UDP socket programming using the DatagramSocket and DatagramPacket classes.

Key Components of UDP Socket Programming in Java

1. DatagramSocket:

- This class is used for both sending and receiving UDP packets.
- A DatagramSocket listens on a specified port and sends/receives datagrams (packets).

2. DatagramPacket:

- This class represents a packet of data that can be sent or received. It contains information such as:

- The data (in the form of a byte array).
- The length of the data.
- The destination address (IP address).
- The port number on the destination machine.

Key Steps in UDP Socket Programming in Java

1. Create a DatagramSocket:

- On the server side, you create a DatagramSocket that listens for incoming packets on a specific port.
- On the client side, you create a DatagramSocket to send the packets to the server.

2. Create a DatagramPacket:

- On the client side, you create a DatagramPacket that contains the data (e.g., a message or number) you want to send, along with the destination address and port.

On the server side, you create a DatagramPacket to receive incoming data.

3. Send and Receive Data:

- The client sends a packet to the server using DatagramSocket.send().
- The server receives the packet using DatagramSocket.receive().
- The server then processes the data and sends a response back to the client.

4. Process Data:

- The server processes the received data, such as performing a mathematical operation (e.g., addition or multiplication) or simply printing

the data.

- The server sends a response back to the client, which can be another packet.

5. Close the DatagramSocket:

- Both the client and server should close their DatagramSocket objects when they are done to release system resources.

When to Use UDP:

UDP is generally used when:

- Speed is more important than reliability (e.g., streaming video, live audio, online gaming).
- Real-time communication is required, and occasional data loss can be tolerated.
- Broadcasting or multicasting is needed, such as in network management or IPTV.

When to Avoid UDP:

UDP should generally be avoided when:

- Data reliability is critical (e.g., file transfers, financial transactions).
- Packet ordering is important (e.g., database synchronization, ordered messaging systems).
- Error handling and retransmission are required by the application.

Program:

Server:

```
import java.net.*;

public class UDPServer {
    public static void main(String[] args) {
        DatagramSocket socket = null;
        try {
            // Create a DatagramSocket to listen on port 9876
            socket = new DatagramSocket(9876);
            byte[] receiveData = new byte[1024];
            System.out.println("Server is listening on port 9876...");
            while (true) {
                // Receive the packet from the client
                DatagramPacket receivePacket = new DatagramPacket(receiveData,
                    receiveData.length);
                socket.receive(receivePacket);
                // Extract data from the packet
                String message = new String(receivePacket.getData(), 0,
                    receivePacket.getLength());
                String[] parts = message.split(",");

                if (parts.length == 2) {
                    // Parse numbers for multiplication
                    try {
                        int num1 = Integer.parseInt(parts[0].trim());
                        int num2 = Integer.parseInt(parts[1].trim());
                        // Perform multiplication
                        int result = num1 * num2;
```

```

// Send the result back to the client
String responseMessage = "Multiplication Result: " + result;
byte[] sendData = responseMessage.getBytes();
InetAddress clientAddress = receivePacket.getAddress();
int clientPort = receivePacket.getPort();
DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.length, clientAddress, clientPort);
socket.send(sendPacket);
} catch (NumberFormatException e) {
System.out.println("Error: Invalid numbers received.");
}
} else {
System.out.println("Error: Invalid message format.");
}
}
} catch (Exception e) {
e.printStackTrace();
} finally {
if (socket != null && !socket.isClosed()) {
socket.close();
}
}
}
}
}

```

Client:

```

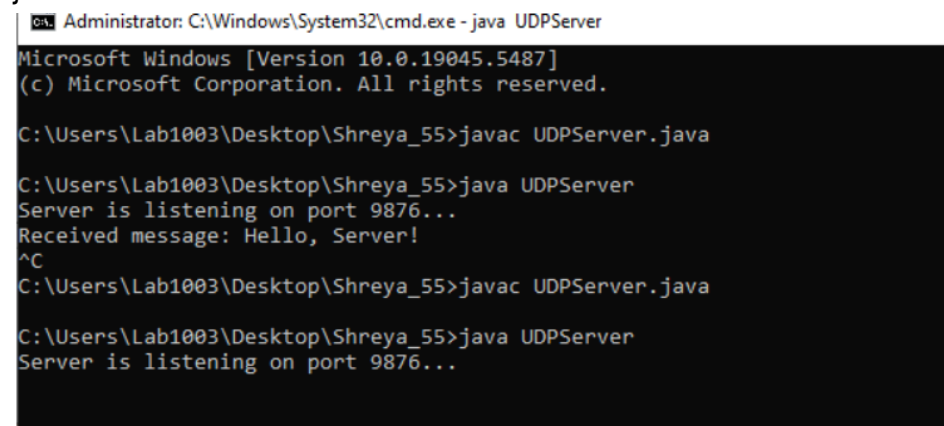
import java.net.*;
public class UDPClient {
    public static void main(String[] args) {
        DatagramSocket socket = null;
        try {
            // Create a DatagramSocket
            socket = new DatagramSocket();
            // Numbers to send for multiplication
            int num1 = 6;
            int num2 = 7;
            // Format the message as "num1,num2"
            String message = num1 + "," + num2;
            byte[] sendData = message.getBytes();
            // Specify the server address and port
            InetAddress serverAddress = InetAddress.getByName("localhost");
            int serverPort = 9876;
            // Create a DatagramPacket to send the message
            DatagramPacket sendPacket = new DatagramPacket(sendData,
            sendData.length, serverAddress, serverPort);
            socket.send(sendPacket);
            System.out.println("Message sent to server: " + message);
            // Receive the response from the server
            byte[] receiveData = new byte[1024];

```

```

DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
socket.receive(receivePacket);
String responseMessage = new String(receivePacket.getData(), 0,
receivePacket.getLength());
System.out.println("Received response from server: " + responseMessage);
} catch (Exception e) {
e.printStackTrace();
} finally {
if (socket != null && !socket.isClosed()) {
socket.close();
}
}
}
}
}

```



```

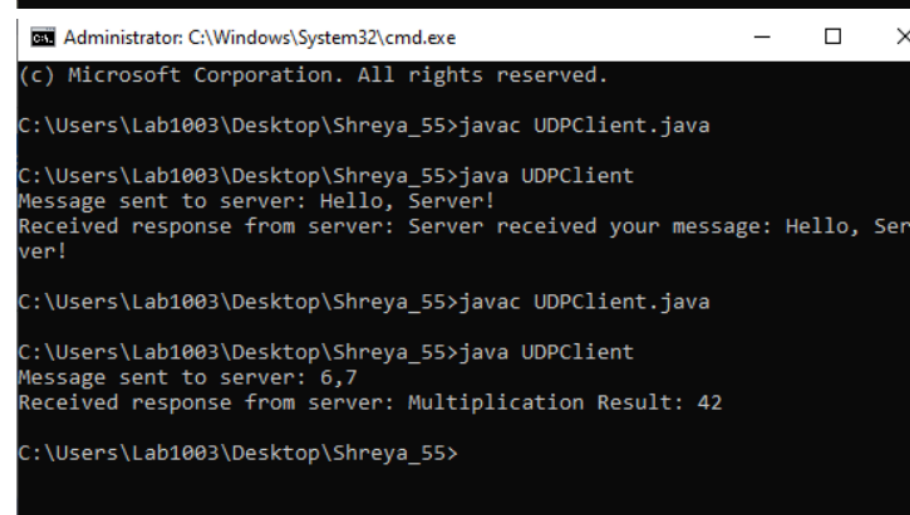
Administrator: C:\Windows\System32\cmd.exe - java UDPServer
Microsoft Windows [Version 10.0.19045.5487]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Lab1003\Desktop\Shreya_55>javac UDPServer.java

C:\Users\Lab1003\Desktop\Shreya_55>java UDPServer
Server is listening on port 9876...
Received message: Hello, Server!
^C
C:\Users\Lab1003\Desktop\Shreya_55>javac UDPServer.java

C:\Users\Lab1003\Desktop\Shreya_55>java UDPServer
Server is listening on port 9876...

```



```

Administrator: C:\Windows\System32\cmd.exe
(c) Microsoft Corporation. All rights reserved.

C:\Users\Lab1003\Desktop\Shreya_55>javac UDPClient.java

C:\Users\Lab1003\Desktop\Shreya_55>java UDPClient
Message sent to server: Hello, Server!
Received response from server: Server received your message: Hello, Ser
ver!

C:\Users\Lab1003\Desktop\Shreya_55>javac UDPClient.java

C:\Users\Lab1003\Desktop\Shreya_55>java UDPClient
Message sent to server: 6,7
Received response from server: Multiplication Result: 42

C:\Users\Lab1003\Desktop\Shreya_55>

```

### Conclusion:

UDP is a highly efficient and fast protocol suitable for scenarios where speed is more important than reliability. However, because it does not guarantee data delivery or order, and lacks mechanisms like flow control or error recovery, it is best suited for applications that can handle potential data loss or require real-time data transmission, like video streaming, gaming, or VoIP. For applications that require guaranteed delivery, such as file transfer or sensitive transactions, TCP would be a better choice