IT22512 – DATA COMMUNICATION AND NETWORKING Laboratory

EX NO :

DATE :

INTRODUCTION TO SOCKET PROGRAMMING

QUESTION: Study the Socket programming and Client - Server Architecture.

AIM:

To stude the Socket programming and Client-Server Architecture.

STUDY:

Socket programming in Java allows two computers to communicate with each other over a network. It's a fundamental aspect of network programming, enabling the creation of client-server applications where one application (the client) requests services from another application (the server). Here's an introduction to socket programming in Java, covering basic concepts and an example.

Socket:

A socket is an endpoint for communication between two machines. It's essentially a combination of an IP address and a port number.

Client-Server Model:

Server: The server listens on a specific port for incoming client connections. It runs indefinitely, waiting for a client to make a connection request.

Client: The client initiates the connection to the server by specifying the server's IP address and port number.

TCP vs. UDP:

TCP (Transmission Control Protocol): Provides reliable, ordered, and error-checked delivery of a stream of data between applications. It's connection-oriented.

UDP (User Datagram Protocol): Provides a simpler, connectionless communication model with minimal protocol overhead. It's suitable for applications where speed is more critical than reliability.

Blocking Operations:

Methods like `accept()` and `readLine()` block the execution until a connection is

established or data is received, respectively.

Streams: Communication between the client and server happens through input and output streams.

Port Numbers: The server listens on a specific port, and the client connects to that port.

Steps in Socket Programming:

1. Server Side:

- Create a `ServerSocket` object to listen for client requests on a specific port.
- Use the `accept()` method of `ServerSocket` to block and wait for a connection from a client.
- Once a connection is established, communicate with the client through `InputStream` and `OutputStream` objects.

2. Client Side:

- Create a `Socket` object to connect to the server using the server's IP address and port number.
- Use `getInputStream()` and `getOutputStream()` methods of `Socket` to communicate with the server.

Example of a Simple Client-Server Application

Server Code:

```
import java.io.*;
import java.net.*;

public class SimpleServer {
    public static void main(String[] args) {
        try {
            // Create a server socket listening on port 5000
            ServerSocket serverSocket = new ServerSocket(5000);
            System.out.println("Server is waiting for client on port 5000...");

            // Accept a client connection
            Socket socket = serverSocket.accept();
            System.out.println("Client connected!");

            // Get input and output streams for communication with the client
            InputStream input = socket.getInputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader(input));
```

```
            OutputStream output = socket.getOutputStream();
            PrintWriter writer = new PrintWriter(output, true);

            // Read a message sent by the client
            String clientMessage = reader.readLine();
            System.out.println("Received from client: " + clientMessage);

            // Send a response to the client
            writer.println("Hello from server!");

            // Close the socket connection
            socket.close();
            serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Client Code:

```
import java.io.*;
import java.net.*;

public class SimpleClient {
    public static void main(String[] args) {
        try {
            // Connect to the server at localhost on port 5000
            Socket socket = new Socket("localhost", 5000);
            System.out.println("Connected to the server!");

            // Get input and output streams for communication with the server
            OutputStream output = socket.getOutputStream();
            PrintWriter writer = new PrintWriter(output, true);

            InputStream input = socket.getInputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader(input));

            // Send a message to the server
            writer.println("Hello from client!");

            // Read a message sent by the server
            String serverMessage = reader.readLine();
            System.out.println("Received from server: " + serverMessage);

            // Close the socket connection
```

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

OUTPUT:

```
C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>java BasicServer
Server is waiting for client on port 5000...
Client connected!
Received from client: Hello from client!
```

```
C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>java BasicClient
Connected to the server!
Received from server: Hello from server!

C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>
```

RESULT:

Thus the Socket programming in Java is a powerful tool for creating networked applications, enabling real-time communication between a client and a server.

IT22512 – DATA COMMUNICATION AND NETWORKING Laboratory

EX NO :

DATE :

APPLICATIONS USING TCP SOCKETS

QUESTION: Implement Applications using TCP Sockets.

AIM:

To Implement the Applications using TCP Sockets.

1.DATE AND TIME SERVER & CLIENT:

ALGORITHM:

TCP Date Server Program Algorithm

STEP 1: Initialize ServerSocket with port 6000.
STEP 2: Print "waiting for client" to indicate that the server is ready to accept a connection.
STEP 3: Wait for and accept a client connection using accept() on the ServerSocket. Create a Socket object to handle the connection.
STEP 4: Get the current date and time using new Date().toString(), then convert it to a UTF-encoded string and send it to the client using writeUTF() on DataOutputStream.
STEP 5: Print "Time and date sent is " followed by the current date and time, then close the DataOutputStream, Socket, and ServerSocket objects.

TCP Date Client Program Algorithm

STEP 1: Initialize a Socket to connect to localhost on port 6000.
STEP 2: Create a DataInputStream using the InputStream from the Socket.
STEP 3: Read the UTF-encoded date and time sent by the server using readUTF() on the DataInputStream.
STEP 4: Print "current date and time received from server is " followed by the received date and time.
STEP 5: Close the DataInputStream and Socket objects.

PROGRAM:

SERVER:

```
import java.io.*;
import java.net.*;
import java.util.Date;
public class Tcpdateserver{
    public static void main(String a[]){
```

```
            try{
            ServerSocket ss=new ServerSocket(6000);
             System.out.println("waituing for client " );
            Socket s=ss.accept();
            System.out.println("client connected ");
            DataOutputStream d  = new DataOutputStream(s.getOutputStream());
            String currentDateTime = new Date().toString();
            d.writeUTF(currentDateTime);
            System.out.println("Time and date sent is " + currentDateTime);
            ss.close();
      }
      catch(Exception e){
            System.out.println(e);
      }
}
}
```

CLIENT:

```
import java.io.*;
import java.net.*;
public class Tcpdateclient{
      public static void main(String a[]){
            try{
                  Socket s=new Socket("localhost",6000);
                  DataInputStream d=new DataInputStream(s.getInputStream());
                  String date=d.readUTF();
                  System.out.println("current date and time received from server is "+
date);
                  d.close();
                  s.close();
            }
            catch(Exception e){
                  System.out.println(e);
            }
      }
}
```

OUTPUT:

```
C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>java Tcpdateserver
waituing for client
client connected
Time and date sent is Mon Aug 19 00:30:05 IST 2024
```

```
C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>java Tcpdateclient
current date and time received from server is Mon Aug 19 00:30:05 IST 2024

C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>
```

2. ECHO CLIENT SERVER

ALGORITHM:

TCP Server Program Algorithm
STEP 1: Initialize ServerSocket with port 7000.
STEP 2: Print "waiting for client" to indicate that the server is ready to accept a connection.
STEP 3: Wait for and accept a client connection using accept() on the ServerSocket. Create a Socket object to handle the connection.
STEP 4: Create a DataInputStream using the InputStream from the Socket, then read a UTF-encoded message sent by the client using readUTF().
STEP 5: Print "message: " followed by the received message, then close the DataInputStream, Socket, and ServerSocket objects.

TCP Client Program Algorithm
STEP 1: Initialize a Socket to connect to localhost on port 7000.
STEP 2: Create a DataOutputStream using the OutputStream from the Socket.
STEP 3: Send the UTF-encoded message "hello" to the server using writeUTF() on the DataOutputStream.
STEP 4: Flush the DataOutputStream to ensure that the message is sent immediately.
STEP 5: Close the DataOutputStream and Socket objects.

PROGRAM:

server:
```
import java.io.*;
import java.net.*;
public class Server{
	public static void main(String a[]){
		try{
			ServerSocket ss=new ServerSocket(7000);
			System.out.println("waiting for client");
			Socket se=ss.accept();
			System.out.println("client connected");
			DataInputStream d=new DataInputStream(se.getInputStream());
			String str=(String)d.readUTF();
			System.out.println("message : " + str);
			ss.close();
		}
		catch(Exception e){
			System.out.println(e);
		}
	}
}
```

Client:

```
import java.io.*;
import java.net.*;
public class Client{
	public static void main(String a[]){
		try{
			Socket s=new Socket("localhost",7000);
			DataOutputStream d=new DataOutputStream(s.getOutputStream());
			d.writeUTF("hello");
			d.flush();
			d.close();
			s.close();
		}
		catch(Exception e){
			System.out.println(e);
		}
	}
}
```

OUTPUT:

```
C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>javac Client.java

C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>java Client
```

```
C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>javac Server.java

C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>java Server
waiting for client
client connected
message : hello
```

3.TCP CHATTING

ALGORITHM:

TCP Chatting Server Program Algorithm

STEP 1: Initialize ServerSocket with port 7000.
STEP 2: Print "waiting for client" to indicate that the server is ready to accept a connection.
STEP 3: Wait for and accept a client connection using accept() on the ServerSocket. Create a Socket object to handle the connection.
STEP 4: Create DataInputStream and DataOutputStream objects using the InputStream and OutputStream from the Socket.
STEP 5: Implement a loop to continuously read messages from the client and send responses back until the client disconnects. Use readUTF() to receive messages and writeUTF() to send responses. Print received messages and handle exceptions.
STEP 6: Close the DataInputStream, DataOutputStream, Socket, and ServerSocket objects.

TCP Chatting Client Program Algorithm

STEP 1: Initialize a Socket to connect to localhost on port 7000.
STEP 2: Create DataInputStream and DataOutputStream objects using the InputStream and OutputStream from the Socket.
STEP 3: Implement a loop to continuously read user input, send messages to the server using writeUTF(), and read responses from the server using readUTF(). Print the server responses and handle exceptions.
STEP 4: Close the DataInputStream, DataOutputStream, and Socket objects.

PROGRAM:

SERVER:

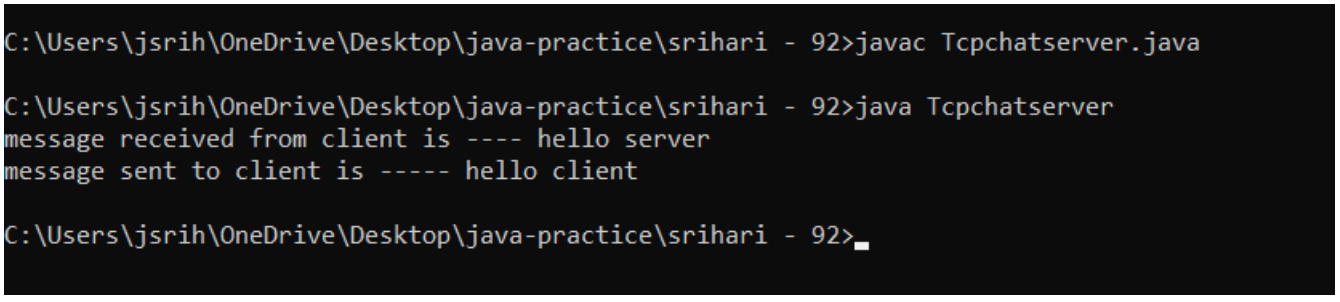
```
 import java.io.*;
import java.net.*;
public class Tcpchatserver{
	public static void main(String a[]){
		try{
			ServerSocket ss=new ServerSocket(9000);
			Socket se=ss.accept();
			DataInputStream di=new DataInputStream(se.getInputStream());
			String msg=(String)di.readUTF();
			System.out.println("message received from client is ---- "+ msg);
			DataOutputStream dop=new DataOutputStream(se.getOutputStream());
			dop.writeUTF("hello client");
			 System.out.println("message sent to client is ----- hello client ");
			dop.flush();
			dop.close();
			ss.close();
		}
```

```
            catch(Exception e){
                System.out.println(e);
            }
        }
}
```

CLIENT:

```
import java.io.*;
import java.net.*;
public class Tcpchatclient{
    public static void main(String a[]){
        try{
            Socket s=new Socket("localhost",9000);
            DataOutputStream dop=new DataOutputStream(s.getOutputStream());
            dop.writeUTF("hello server");
             System.out.println("message sent to server is --- hello server");
            DataInputStream di=new DataInputStream(s.getInputStream());
            String str=(String)di.readUTF();
            System.out.println("msg from server is ---- "+ str);
            dop.flush();
            dop.close();
            s.close();
        }
        catch(Exception e){
            System.out.println(e);
        }
    }
}
```

OUTPUT:



```
C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>javac Tcpchatclient.java

C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>java Tcpchatclient
message sent to server is --- hello server
msg from server is ---- hello client
```

RESULT:

Thus the Implementation of Applications for TCP Sockets is done successfully.

IT22512 – DATA COMMUNICATION AND NETWORKING Laboratory

EX NO :

DATE :

APPLICATIONS USING UDP SOCKETS

QUESTION: Implement Applications using UDP Sockets.

AIM:

To Implement the Applications using UDP Sockets.

1.DATE AND TIME SERVER & CLIENT:

ALGORITHM:

UDP Date and Time Server Program Algorithm
STEP 1: Initialize DatagramSocket with port 5000.
STEP 2: Get the current date and time using new Date().toString().
STEP 3: Convert the date and time to bytes.
STEP 4: Create a DatagramPacket with the date and time bytes, specifying the client's IP address and port (which should be obtained from a request if the server is to handle multiple clients).
STEP 5: Send the DatagramPacket using send() on the DatagramSocket, then close the socket.

UDP Date and Time Client Program Algorithm
STEP 1: Initialize DatagramSocket with a random port (e.g., 0).
STEP 2: Create a DatagramPacket to receive the data, with a buffer size of 1024 bytes.
STEP 3: Use the receive() method on the DatagramSocket to wait for and receive the date and time packet from the server.
STEP 4: Extract and print the received date and time with the prefix "current date and time received from server is ".
STEP 5: Close the DatagramSocket.

PROGRAM:

SERVER:

```
import java.net.*;
import java.util.Date;
//reciever...
class Udpdateserver {
   public static void main(String[] args) {
      try {
```

```
            // Create a DatagramSocket to listen on port 5000
            DatagramSocket serverSocket = new DatagramSocket(5000);
            byte[] receiveBuffer = new byte[1024];
            byte[] sendBuffer;

            System.out.println("UDP Date Server is running...");

                // Prepare to receive a request
                DatagramPacket receivePacket = new DatagramPacket(receiveBuffer,
receiveBuffer.length);
                serverSocket.receive(receivePacket); // Receive the request

                // Get the current date and time
                String currentTime = new Date().toString();
                sendBuffer = currentTime.getBytes(); // Convert date and time to bytes

                // Get client's IP address and port number
                InetAddress clientIP = receivePacket.getAddress();
                int clientPort = receivePacket.getPort();

                // Prepare the packet to send the date and time back to the client
                DatagramPacket sendPacket = new DatagramPacket(sendBuffer,
sendBuffer.length, clientIP, clientPort);
                serverSocket.send(sendPacket); // Send the packet

                System.out.println("Sent date and time to client: " + currentTime);

        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

CLIENT:

```
import java.net.*;
//sender
class Udpdateclient {
    public static void main(String[] args) {
        try {
            // Create a DatagramSocket
            DatagramSocket clientSocket = new DatagramSocket();
            InetAddress serverIP = InetAddress.getByName("127.0.0.1");
            byte[] sendBuffer = "REQUEST".getBytes(); // You can send any request
message
            byte[] receiveBuffer = new byte[1024];
```

```
            // Send a request to the server
            DatagramPacket sendPacket = new DatagramPacket(sendBuffer,
    sendBuffer.length, serverIP, 5000);
            clientSocket.send(sendPacket);

            // Receive the date and time from the server
            DatagramPacket receivePacket = new DatagramPacket(receiveBuffer,
    receiveBuffer.length);
            clientSocket.receive(receivePacket);

            // Convert the received bytes into a string and print it
            String receivedTime = new String(receivePacket.getData(), 0,
    receivePacket.getLength());
            System.out.println("Current Date and Time from Server: " + receivedTime);

            clientSocket.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

OUTPUT:

```
C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>javac Udpdateserver.java

C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>java Udpdateserver
UDP Date Server is running...
Sent date and time to client: Mon Aug 19 00:45:07 IST 2024
```

```
C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>javac Udpdateclient.java

C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>java Udpdateclient
Current Date and Time from Server: Mon Aug 19 00:45:07 IST 2024

C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>
```

2. ECHO CLIENT SERVER

ALGORITHM:

UDP Echo Server Program Algorithm
STEP 1: Initialize DatagramSocket with port 5000.
STEP 2: Create a buffer to receive incoming data (e.g., byte array of size 1024).
STEP 3: Continuously receive DatagramPacket from clients using receive() on the DatagramSocket.
STEP 4: Send the received data back to the client using send() on the DatagramSocket.
STEP 5: Close the DatagramSocket after handling all requests (or handle termination separately).

UDP Echo Client Program Algorithm
STEP 1: Initialize DatagramSocket with a random port (e.g., 0).
STEP 2: Prepare a DatagramPacket with the message to be sent and the server's IP address and port.
STEP 3: Send the DatagramPacket to the server using send() on the DatagramSocket.
STEP 4: Create a buffer to receive the echoed data (e.g., byte array of size 1024).
STEP 5: Receive the echoed message from the server using receive() on the DatagramSocket and print it with the prefix "Echoed message from server is ".
STEP 6: Close the DatagramSocket.

PROGRAM:

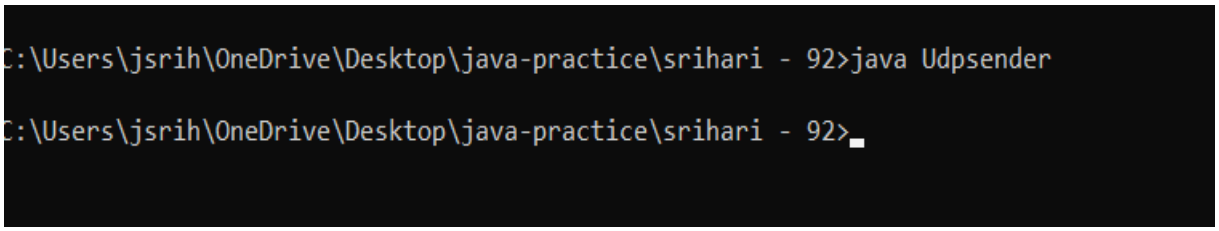
sender:

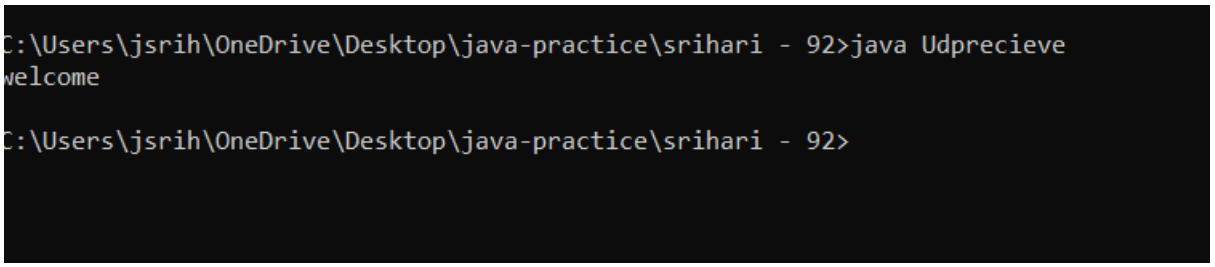
```
import java.io.*;
import java.net.*;

class Udpsender {
   public static void main(String a[]) {
try{

      DatagramSocket ss = new DatagramSocket();
      String str = "welcome";
      InetAddress ip = InetAddress.getByName("127.0.0.1");
      DatagramPacket dp = new DatagramPacket(str.getBytes(), str.getBytes().length, ip,
5000);
      ss.send(dp);
      ss.close();
}catch(Exception e){
System.out.println(e);
}
   }
}
```

Receiver:

```
import java.net.*;
class Udprecieve{
public static void main(String a[]) {
try{
DatagramSocket ds=new DatagramSocket(5000);
byte[] buf=new byte[1024];
DatagramPacket dp=new DatagramPacket(buf,1024);
ds.receive(dp);
String str=new String(dp.getData(),0,dp.getLength());
System.out.println(str);
ds.close();
}catch(Exception e){
System.out.println(e);
}
}
}
```

OUTPUT:





3.UDP CHATTING

ALGORITHM:

UDP Chatting Server Program Algorithm
STEP 1: Initialize DatagramSocket with port 5000.
STEP 2: Create a buffer to receive incoming messages (e.g., byte array of size 1024).
STEP 3: Implement a loop to continuously receive messages from clients using receive() on the DatagramSocket.
STEP 4: Print the received message and prepare a response. Send the response back to the client using send() on the DatagramSocket.
STEP 5: Close the DatagramSocket after handling all requests (or handle termination separately).

UDP Chatting Client Program Algorithm
STEP 1: Initialize DatagramSocket with a random port (e.g., 0).
STEP 2: Create a DatagramPacket with the message to be sent and the server's IP address and port.
STEP 3: Send the DatagramPacket to the server using send() on the DatagramSocket.
STEP 4: Create a buffer to receive the server's response (e.g., byte array of size 1024).
STEP 5: Receive and print the server's response with the prefix "Received from server: " using receive() on the DatagramSocket.
STEP 6: Close the DatagramSocket.

PROGRAM:

SERVER:

```
 import java.net.*;
import java.util.Scanner;

class Udpchatserver {
   public static void main(String[] args) {
      try {
         DatagramSocket serverSocket = new DatagramSocket(5000);
         byte[] receiveBuffer = new byte[1024];
         byte[] sendBuffer;

         InetAddress clientIP = null;
         int clientPort = 0;

         System.out.println("UDP Chat Server is running...");

         Scanner scanner = new Scanner(System.in);

         while (true) {
            // Prepare to receive a message from the client
            DatagramPacket receivePacket = new DatagramPacket(receiveBuffer,
```

```
receiveBuffer.length);
            serverSocket.receive(receivePacket); // Receive the message

            // Extract the message and client's details
            String clientMessage = new String(receivePacket.getData(), 0,
receivePacket.getLength());
            clientIP = receivePacket.getAddress();
            clientPort = receivePacket.getPort();

            System.out.println("Client: " + clientMessage);

            // Break the chat loop if the client sends "bye"
            if (clientMessage.equalsIgnoreCase("bye")) {
               System.out.println("Client disconnected.");
               break;
            }

            // Prepare to send a message back to the client
            System.out.print("You: ");
            String serverMessage = scanner.nextLine();
            sendBuffer = serverMessage.getBytes();
            DatagramPacket sendPacket = new DatagramPacket(sendBuffer,
sendBuffer.length, clientIP, clientPort);
            serverSocket.send(sendPacket); // Send the message

            // Break the chat loop if the server sends "bye"
            if (serverMessage.equalsIgnoreCase("bye")) {
               System.out.println("Server disconnected.");
               break;
            }
         }

         serverSocket.close();
         scanner.close();
      } catch (Exception e) {
         System.out.println(e);
      }
   }
}
```

CLIENT:

```
import java.net.*;
import java.util.Scanner;

class Udpchatclient {
   public static void main(String[] args) {
```

```
        try {
            DatagramSocket clientSocket = new DatagramSocket();
            InetAddress serverIP = InetAddress.getByName("127.0.0.1");
            byte[] sendBuffer;
            byte[] receiveBuffer = new byte[1024];

            System.out.println("UDP Chat Client is running...");
            Scanner scanner = new Scanner(System.in);

            while (true) {
                // Prepare to send a message to the server
                System.out.print("You: ");
                String clientMessage = scanner.nextLine();
                sendBuffer = clientMessage.getBytes();
                DatagramPacket sendPacket = new DatagramPacket(sendBuffer,
sendBuffer.length, serverIP, 5000);
                clientSocket.send(sendPacket); // Send the message

                // Break the chat loop if the client sends "bye"
                if (clientMessage.equalsIgnoreCase("bye")) {
                    System.out.println("You disconnected.");
                    break;
                }

                // Prepare to receive a message from the server
                DatagramPacket receivePacket = new DatagramPacket(receiveBuffer,
receiveBuffer.length);
                clientSocket.receive(receivePacket); // Receive the message

                String serverMessage = new String(receivePacket.getData(), 0,
receivePacket.getLength());
                System.out.println("Server: " + serverMessage);

                // Break the chat loop if the server sends "bye"
                if (serverMessage.equalsIgnoreCase("bye")) {
                    System.out.println("Server disconnected.");
                    break;
                }
            }

            clientSocket.close();
            scanner.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
```

}

OUTPUT:

```
C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>javac Udpchatserver.java

C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>java Udpchatserver
UDP Chat Server is running...
Client: 1
You: 2
Client: 3
You: 4
Client: 5
You: 6
```

```
C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>javac Udpchatclient.java

C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>java Udpchatclient
UDP Chat Client is running...
You: 1
Server: 2
You: 3
Server: 4
You: 5
Server: 6
```

RESULT:

Thus the Implementation of Applications for UDP Sockets is done successfully.

IT22512 – DATA COMMUNICATION AND NETWORKING Laboratory

EX NO :

DATE :

BIT STUFFING

AIM:

ALGORITHM:

STEP 1: Initialize an empty string `stuffedBits` and set a counter “consecutive Ones” to 0.

STEP 2: Read the input bit sequence from the user using “Scanner.nextLine()”.

STEP 3: Iterate over each bit in the input sequence using a loop.

STEP 4: Append the current bit to the `stuffedBits` string.

STEP 5: If the current bit is '1', increment the “consecutive Ones” counter.

STEP 6: If “consecutive Ones” reaches 5, append a '0' to “stuffedBits” and reset “consecutive Ones” to 0.

STEP 7: If the current bit is '0', reset the “consecutive Ones” counter to 0 and continue the loop.

PROGRAM:

```
import java.util.ArrayList;
import java.util.List;

public class BitStuffing {

    // Method to perform bit stuffing
    public static List<Integer> bitStuff(int[] data) {
        List<Integer> stuffedData = new ArrayList<>();
        int count = 0;

        for (int bit : data) {
```

```
            if (bit == 1) {
                count++;
            } else {
                count = 0;
            }

            stuffedData.add(bit);

            // If five consecutive 1's are found, insert a 0
            if (count == 5) {
                stuffedData.add(0);
                count = 0; // Reset count after stuffing
            }
        }

        return stuffedData;
    }

    public static void main(String[] args) {
        // Sample data: 11011111101111
        int[] data = {1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1};

        System.out.println("Original Data:");
        for (int bit : data) {
            System.out.print(bit);
        }
        System.out.println();

        List<Integer> stuffedData = bitStuff(data);

        System.out.println("Stuffed Data:");
        for (int bit : stuffedData) {
            System.out.print(bit);
        }
    }
}
```

OUTPUT

```
C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>javac BitStuffing.java

C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>java BitStuffing
Original Data:
1101111101111
Stuffed Data:
11011111001111
C:\Users\jsrih\OneDrive\Desktop\java-practice\srihari - 92>
```

RESULT

IT22512 – DATA COMMUNICATION AND NETWORKING LABORATORY

EX NO : 5

DATE :

QUESTION:

AIM:

ALGORITHM:

STEP 1: Prompt the user to enter two 8-bit subunits using Scanner.nextLine().

STEP 2: Generate the checksum by adding the two subunits using binary addition and then taking the complement of the sum.

STEP 3: Combine the two subunits and the checksum to form the transmitted data.

STEP 4: Prompt the user to enter the received 8-bit subunits and checksum.

STEP 5: Add the received subunits and the received checksum using binary addition.

STEP 6: Take the complement of the sum from Step 5 and check if it equals "00000000".

STEP 7: If the result is "00000000", print "No error detected"; otherwise, print "Error detected in received data."

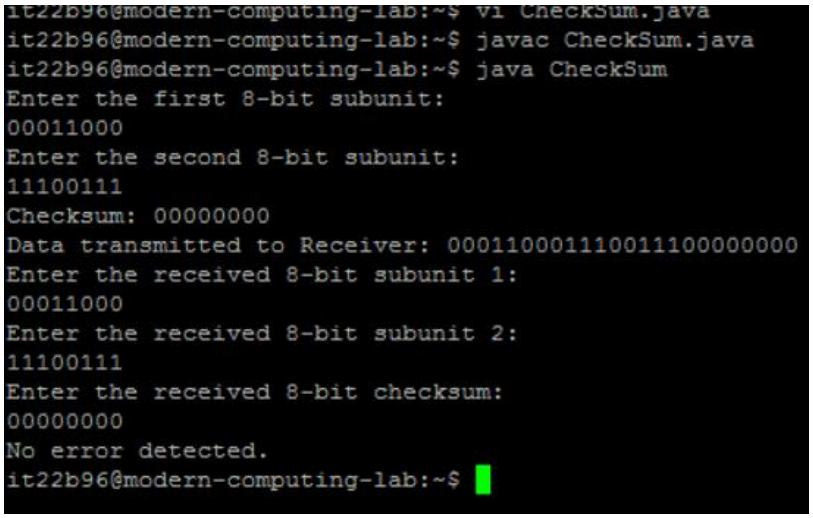
PROGRAM:

```
import java.util.Scanner;

public class CheckSum {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the first 8-bit subunit:");
        String subunit1 = scanner.nextLine();
```

```
        System.out.println("Enter the second 8-bit subunit:");
        String subunit2 = scanner.nextLine();
        String checksum = generateChecksum(subunit1, subunit2);
        System.out.println("Checksum: " + checksum);
        String transmittedData = subunit1 + subunit2 + checksum;
        System.out.println("Data transmitted to Receiver: " +
 transmittedData);
        System.out.println("Enter the received 8-bit subunit 1:");
        String receivedSubunit1 = scanner.nextLine();
        System.out.println("Enter the received 8-bit subunit 2:");
        String receivedSubunit2 = scanner.nextLine();
        System.out.println("Enter the received 8-bit checksum:");
        String receivedChecksum = scanner.nextLine();
        if (verifyChecksum(receivedSubunit1, receivedSubunit2,
receivedChecksum)) {
            System.out.println("No error detected.");
        } else {
            System.out.println("Error detected in received data.");
        }
    }

    public static String generateChecksum(String subunit1, String
 subunit2) {
        int sum = addBinary(subunit1, subunit2);
        String checksum = complement(sum);
        return checksum;}
     public static boolean verifyChecksum(String subunit1, String
 subunit2, String checksum) {
        int sum = addBinary(subunit1, subunit2);
        sum = addBinary(Integer.toBinaryString(sum), checksum);
        return complement(sum).equals("00000000");
    }
   public static int addBinary(String a, String b) {
        int sum = Integer.parseInt(a, 2) + Integer.parseInt(b, 2);
        if (Integer.toBinaryString(sum).length() > 8) {
            sum = sum + 1;
            sum = sum & 0xFF;
        }
        return sum;
    }

    public static String complement(int sum) {
        String binarySum = Integer.toBinaryString(sum);
        while (binarySum.length() < 8) {
```

```
            binarySum = "0" + binarySum;
        }
        StringBuilder complement = new StringBuilder();
        for (char bit : binarySum.toCharArray()) {
            complement.append(bit == '0' ? '1' : '0');
        }
        return complement.toString();
    }
}
```

OUTPUT:



RESULT: