# EXPERIMENT NO: 03

**AIM:**

Write a Java program to implement **TCP Communication in one way.**

**ALGORITHM:**

Server:

1. Import all necessary java libraries.

2. Declare a class name ‘server’ and inside the try block create a ServerSocket on port 1346.

3. Display a message which shows that waiting for Client Connection.

4. Accept Client Connection.

5. Create a PrintStream to send data to the connected client.

6. Create a BufferedReader to read input from the console.

7. Prompt the user to input data at the server. Read a line of input from the console.

8. Send the input data to the connected client.

9. Close the PrintStream, BufferedReader, and Socket.

10. Catch and handle IOExceptions.

11 Conclude the program, marking the end of the execution flow.

Client:

1. Import all necessary java libraries.

2. Create a class called ‘client’ and within a try block create a Socket to connect to the server.

3. Print Connection Status.

4. Create a BufferedReader to read data from the server.

5. Print the response received from the server.

6. Close the socket.

7. Catch and handle IOExceptions.

8. Conclude the program, marking the end of the execution flow.

**PROGRAM:**

Server:

import java.io.\*;

import java.net.\*;

class server {

public static void main(String[] args)

{

try {

ServerSocket serversocket = new ServerSocket(1346);

System.out.println("waiting for request....");

Socket socket = serversocket.accept();

System.out.println("Request Accepted...");

PrintStream ps = new PrintStream(socket.getOutputStream());

BufferedReader br = new BufferedReader(

new InputStreamReader(System.in));

System.out.println(

"Input the data at the server...");

ps.print(br.readLine());

socket.close();

serversocket.close();

}

catch (IOException e) {

System.out.println("Not found data for socket"+ e);

} } }

Client:

import java.io.\*;

import java.net.\*;

class client {

public static void main(String[] args)

{

try {

Socket socket = new Socket("localhost", 1346);

System.out.println(

"Connected Successfully.....");

BufferedReader bs = new BufferedReader(

new InputStreamReader(socket.getInputStream()));

System.out.println("Response from Server.....");

System.out.println("Client Side : "

+ bs.readLine());

socket.close();

}

catch (UnknownHostException e) {

System.out.println("IP not found for" + e);

}

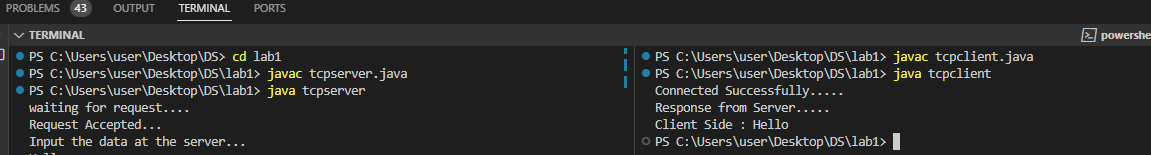
catch (IOException e) {

System.out.println("Not found data for socket"

+ e);

}} }

**OUTPUT:**

****

# EXPERIMENT NO: 04

**AIM:**

Write a Java program to implement **UDP Communication in one way.**

**ALGORITHM:**

Server:

1. Import all the necessary Java libraries for networking.

2. Define the UDP server class, encapsulating the functionality of the server.

3. Instantiate a DatagramSocket named aSocket to enable UDP communication.

4. Allocate a byte array within the try block to serve as a buffer for holding incoming data.

5. Construct a DatagramPacket to facilitate the reception and processing of data.

6. Implement exception handling within catch blocks to address potential SocketException and IOException occurrences, ensuring robust error management.

7. Safely close the DatagramSocket to release system resources in a finally block, enhancing the program's reliability.

8. Conclude the program, marking the end of the execution flow.

Client:

1. Import all necessary libraries.

2. Define the UDP client class, encapsulating the functionality of the client.

3. Create DatagramSocket named clientSocket to facilitate UDP communication.

4. Create a datagram Packet inside the try block with the message, destination InetAddress, Server port.

5. Print the response received from the server.

6. Catch any exceptions.

7. Safely close the DatagramSocket to release system resources in a finally block.

8. Conclude the program, marking the end of the execution flow.

**PROGRAM:**

Server:

import java.net.\*;

import java.io.\*;

public class UDPServer {

    public static void main(String args[]) {

        DatagramSocket aSocket = null;

        if (args.length < 1) {

            System.out.println("Usage: java UDPServer ");

            System.exit(1);  }

        try {

            int socket\_no = Integer.valueOf(args[0]).intValue();

            aSocket = new DatagramSocket(socket\_no);

            byte[] buffer = new byte[2000];

            while (true) {

                DatagramPacket request = new DatagramPacket(buffer, buffer.length);

                aSocket.receive(request);

                System.out.println("Received: " + new String(request.getData()));

                DatagramPacket reply = new DatagramPacket(request.getData(),

                        request.getLength(), request.getAddress(), request.getPort());

                aSocket.send(reply);

            }

        } catch (SocketException e) {

            System.out.println("Socket: " + e.getMessage());

        } catch (IOException e) {

            System.out.println("IO:" + e.getMessage());

        } finally {

            if (aSocket != null)

                aSocket.close();}}}

Client:

import java.net.\*;

import java.io.\*;

public class UDPClient {

    public static void main(String args[]) { // args give message contents and server hostname

        DatagramSocket aSocket = null;

        if (args.length < 3) {

            System.out.println("Usage: java UDPClient ");

            System.exit(1);

        }

        try {

            aSocket = new DatagramSocket();

            byte[] m = args[0].getBytes();

            InetAddress aHost = InetAddress.getByName(args[1]);

            int serverPort = Integer.valueOf(args[2]).intValue();

            DatagramPacket request = new DatagramPacket(m, args[0].length(), aHost, serverPort);

            aSocket.send(request);

            byte[] buffer = new byte[2000];

            DatagramPacket reply = new DatagramPacket(buffer, buffer.length);

            aSocket.receive(reply);

            System.out.println("Reply: " + new String(reply.getData()));

        } catch (SocketException e) {

            System.out.println("Socket: " + e.getMessage());

        } catch (IOException e) {

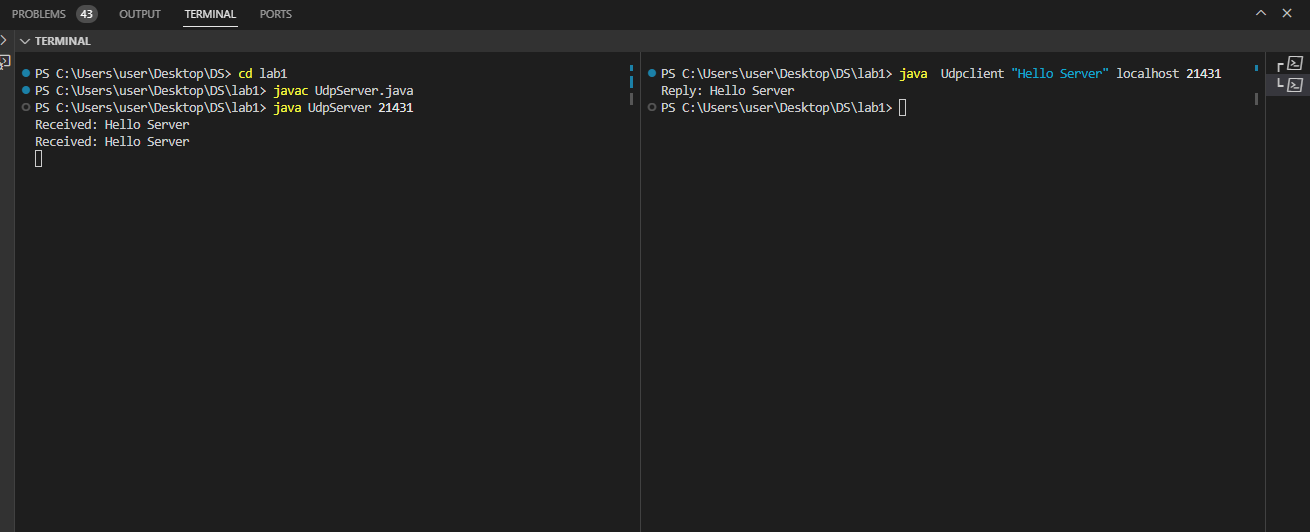
            System.out.println("IO: " + e.getMessage());

        } finally {

            if (aSocket != null)

                aSocket.close();}} }

**OUTPUT:**

****

# EXPERIMENT NO: 05

**AIM:**

Write a Java program to implement **TCP Communication in two way.**

**ALGORITHM:**

Server:

1. Import the required Java libraries for networking, including ServerSocket, Socket, DataInputStream, DataOutputStream, IOException, and BufferedReader.

2. Define the TCP Server class to handle communication with clients bidirectionally.

3. Create a ServerSocket to listen for incoming client connections.

4. Implement a loop to wait for client connections using the accept method.

5. Create a DataOutputStream to send data to the connected client.

6. Create a BufferedReader to read input from the console.

7. Send response data to the connected client using the DataOutputStream.

8. Safely close the DataOutputStream, BufferedReader, and Socket to release resources.

9. Catch and handle IOExceptions that may occur during socket communication.

10. Conclude the server program, marking the end of its execution.

Client:

1. Import the necessary Java libraries for networking, including Socket, DataInputStream, DataOutputStream, IOException, and BufferedReader.

2. Create a Socket to establish a connection with the server.

3. Set up a DataOutputStream to send data to the server.

4. Create a BufferedReader to read data from the server.

5. Print the response received from the server to the console.

6. Close all connections, including the DataOutputStream, BufferedReader, and Socket.

7. Conclude the client program, marking the end of its execution.

**PROGRAM:**

Server:

import java.io.\*;

import java.net.\*;

class TCPServer2 {

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

ServerSocket ss = new ServerSocket(888);

Socket s = ss.accept();

System.out.println("Connection established");

PrintStream ps = new PrintStream(s.getOutputStream());

BufferedReader br = new BufferedReader(new InputStreamReader(s.getInputStream()));

BufferedReader kb = new BufferedReader(new InputStreamReader(System.in));

while (true) {

String str, str1;

while ((str = br.readLine()) != null) {

System.out.println(str);

str1 = kb.readLine();

ps.println(str1); }

ps.close();

br.close();

kb.close();

ss.close();

s.close();

System.exit(0);

} } }

Client:

import java.io.\*;

import java.net.\*;

class TCPClient2 {

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

Socket s = new Socket("localhost", 888);

DataOutputStream dos = new DataOutputStream( s.getOutputStream());

BufferedReader br = new BufferedReader( new InputStreamReader(s.getInputStream()));

BufferedReader kb = new BufferedReader(new InputStreamReader(System.in));

String str, str1;

while (!(str = kb.readLine()).equals("exit")) {

dos.writeBytes(str + "\n");

str1 = br.readLine();

System.out.println(str1); }

dos.close();

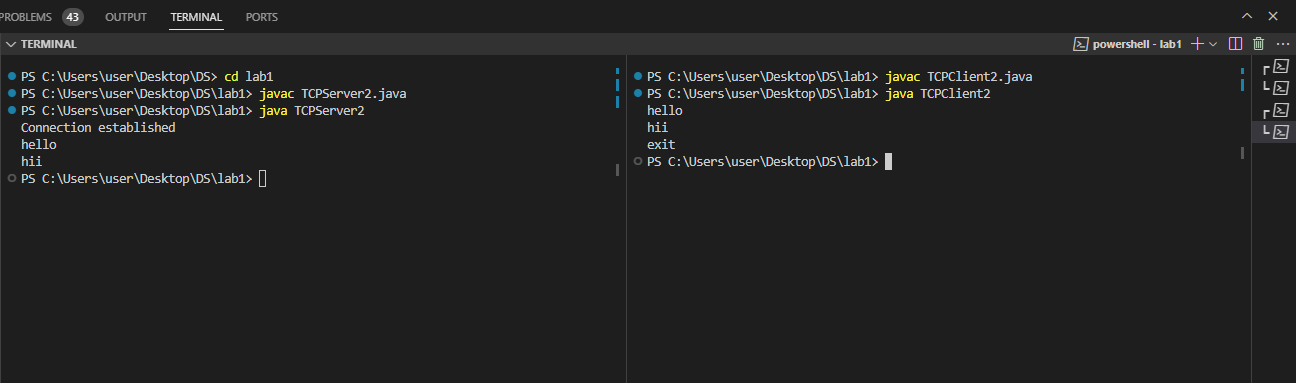
br.close();

kb.close();

s.close();

} }

**OUTPUT:**

****

# EXPERIMENT NO: 06

**AIM:**

Write a Java program to implement **UDP Communication in two way.**

**ALGORITHM:**

Server:

1. Initialize DatagramSocket and Receive Buffer.

2. Print Server Start Message.

3. Create a 'DatagramPacket' named 'receivePacket' to capture incoming data.

4. Convert the received byte array into a string, 'receivedMessage'.

5. Print Received Message.

6. Read Input from Console.

7. Retrieve Client's Address and Port.

8. Convert Input String to Byte Array. Convert the input string 'str1' into a byte array named 'sendData'.

9. Create DatagramPacket for Sending. Create a new 'DatagramPacket' named 'sendPacket' with the 'sendData', specifying the client's address and port.

10. Send Data to Client

11. Check for Exit Condition. Check if the input string `str1` is equal to "exit". If true, break out of the loop and terminate the server.

12. End of Loop.

Client:

1. Initialize a DatagramSocket, specifying the server address and port for communication.

2. Initialize an Input Reader and set up a continuous input loop to capture user input.

3. Read input from the console.

4. Convert the input string to a byte array to prepare it for transmission.

5. Create a DatagramPacket for sending the data to the server.

6. Send the data to the server using the DatagramSocket.

7. Check for an exit condition to determine whether to continue or terminate the communication loop.

8. Initialize a buffer for receiving data from the server.

9. Receive data from the server using a DatagramPacket.

10. Convert the received byte data to a string for processing.

11. Print the received message to the console.

12. End the loop and repeat the process if the exit condition is not met.

13. Close the DatagramSocket to release resources after the communication is complete.

**PROGRAM:**

Server:

import java.io.\*;

import java.net.\*;

class UDPServer2 {

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

DatagramSocket serverSocket = new DatagramSocket(888);

byte[] receiveData = new byte[1024];

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

while (true) {

DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

serverSocket.receive(receivePacket);

String receivedMessage = new String(receivePacket.getData(), 0, receivePacket.getLength());

System.out.println("Received from client: " + receivedMessage);

BufferedReader kb = new BufferedReader(new InputStreamReader(System.in));

String str1 = kb.readLine();

InetAddress clientAddress = receivePacket.getAddress();

int clientPort = receivePacket.getPort();

byte[] sendData = str1.getBytes();

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, clientAddress,

clientPort);

serverSocket.send(sendPacket);

if (str1.equals("exit")) {

break;

}}}}

Client:

import java.io.\*;

import java.net.\*;

class UDPClient2 {

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

DatagramSocket clientSocket = new DatagramSocket();

InetAddress serverAddress = InetAddress.getByName("localhost");

int serverPort = 888;

BufferedReader kb = new BufferedReader(new InputStreamReader(System.in));

while (true) {

String str = kb.readLine();

byte[] sendData = str.getBytes();

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, serverAddress,

serverPort);

clientSocket.send(sendPacket);

if (str.equals("exit")) {

break; }

byte[] receiveData = new byte[1024];

DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

clientSocket.receive(receivePacket);

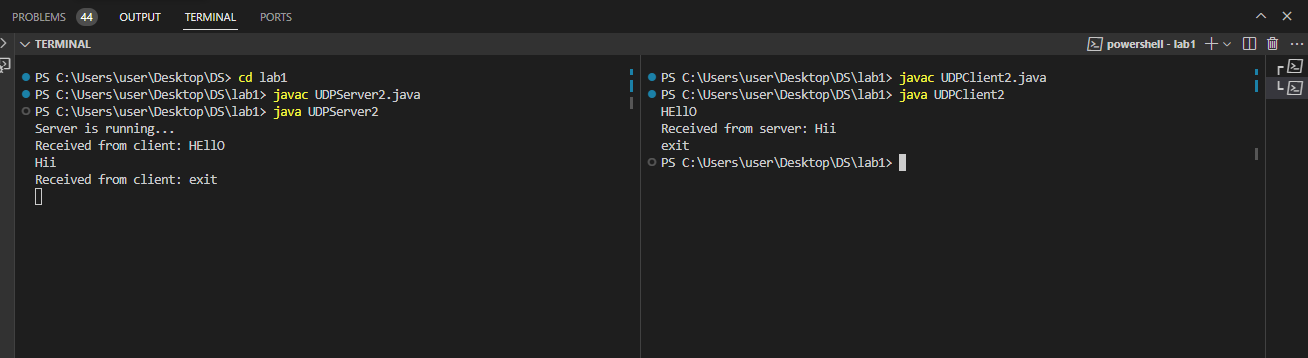
String receivedMessage = new String(receivePacket.getData(), 0, receivePacket.getLength());

System.out.println("Received from server: " + receivedMessage); }

clientSocket.close();

}}

**OUTPUT:**

****

# EXPERIMENT NO: 07

**AIM:**

Write a Java program to implement **Single Chatting application using GUI.**

**ALGORITHM:**

Server:

1. Import all necessary java libraries for networking and GUI.

2. Define Single chat server class that extends Jframe for GUI.

3. Declare instance variables .

4. Create server socket for listening incoming client connection.

5. Upon client connection establish input-output streams for communicaton.

6. Get text from message field.

7. Send response data to the connected client.

8. Instantiate the class to activate GUI and server functionality.

9. close all resources and end the program.

Client:

1. Import all necessary libraries for networking and GUI.

2. Define Single chat client class that extends Jframe for GUI.

3. Declare instance variables .

4. Establish socket connection to the server.

5. Establish input-output streams for communication with the server.

6. Get message from text field in the GUI and send it to server.

7. In main method create instance of the class o activate GUI and client functionality.

8. Close all resources end the program.

**PROGRAM:**

Server:

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

public class SingleChatServerGUI extends JFrame {

private JTextArea chatArea;

private JTextField messageField;

private PrintWriter out;

public SingleChatServerGUI() {

setTitle("Server");

setSize(400, 300);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLayout(new BorderLayout());

chatArea = new JTextArea();

chatArea.setEditable(false);

JScrollPane scrollPane = new JScrollPane(chatArea);

getContentPane().add(scrollPane, BorderLayout.CENTER);

JPanel bottomPanel = new JPanel(new BorderLayout());

messageField = new JTextField();

bottomPanel.add(messageField, BorderLayout.CENTER);

JButton sendButton = new JButton("Send");

sendButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

sendMessage();}});

bottomPanel.add(sendButton, BorderLayout.EAST);

getContentPane().add(bottomPanel, BorderLayout.SOUTH);

setVisible(true);

try {

ServerSocket serverSocket = new ServerSocket(12345);

Socket clientSocket = serverSocket.accept();

chatArea.append("Client connected: " + clientSocket + "\n");

out = new PrintWriter(clientSocket.getOutputStream(), true);

BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

String inputLine;

while ((inputLine = in.readLine()) != null) {

chatArea.append("Client: " + inputLine + "\n")}

in.close();

clientSocket.close();

serverSocket.close();

} catch (Exception e) {

e.printStackTrace();}}

private void sendMessage() {

String message = messageField.getText();

if (!message.isEmpty()) {

out.println(message);

chatArea.append("Server: " + message + "\n");

messageField.setText("");}}

public static void main(String[] args) {

new SingleChatServerGUI();}}

Client:

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.Socket;

public class SingleChatClientGUI extends JFrame {

private JTextArea chatArea;

private JTextField messageField;

private PrintWriter out;

public SingleChatClientGUI() {

setTitle("Client");

setSize(400, 300);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLayout(new BorderLayout());

chatArea = new JTextArea();

chatArea.setEditable(false);

JScrollPane scrollPane = new JScrollPane(chatArea);

getContentPane().add(scrollPane, BorderLayout.CENTER);

JPanel bottomPanel = new JPanel(new BorderLayout());

messageField = new JTextField();

bottomPanel.add(messageField, BorderLayout.CENTER);

JButton sendButton = new JButton("Send");

sendButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

sendMessage();} });

bottomPanel.add(sendButton, BorderLayout.EAST);

getContentPane().add(bottomPanel, BorderLayout.SOUTH);

setVisible(true);

try {

Socket socket = new Socket("localhost", 12345);

out = new PrintWriter(socket.getOutputStream(), true);

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

String inputLine;

while ((inputLine = in.readLine()) != null) {

chatArea.append("Server: " + inputLine + "\n");}

in.close();

socket.close();

} catch (Exception e) {

e.printStackTrace();}}

private void sendMessage() {

String message = messageField.getText();

if (!message.isEmpty()) {

out.println(message);

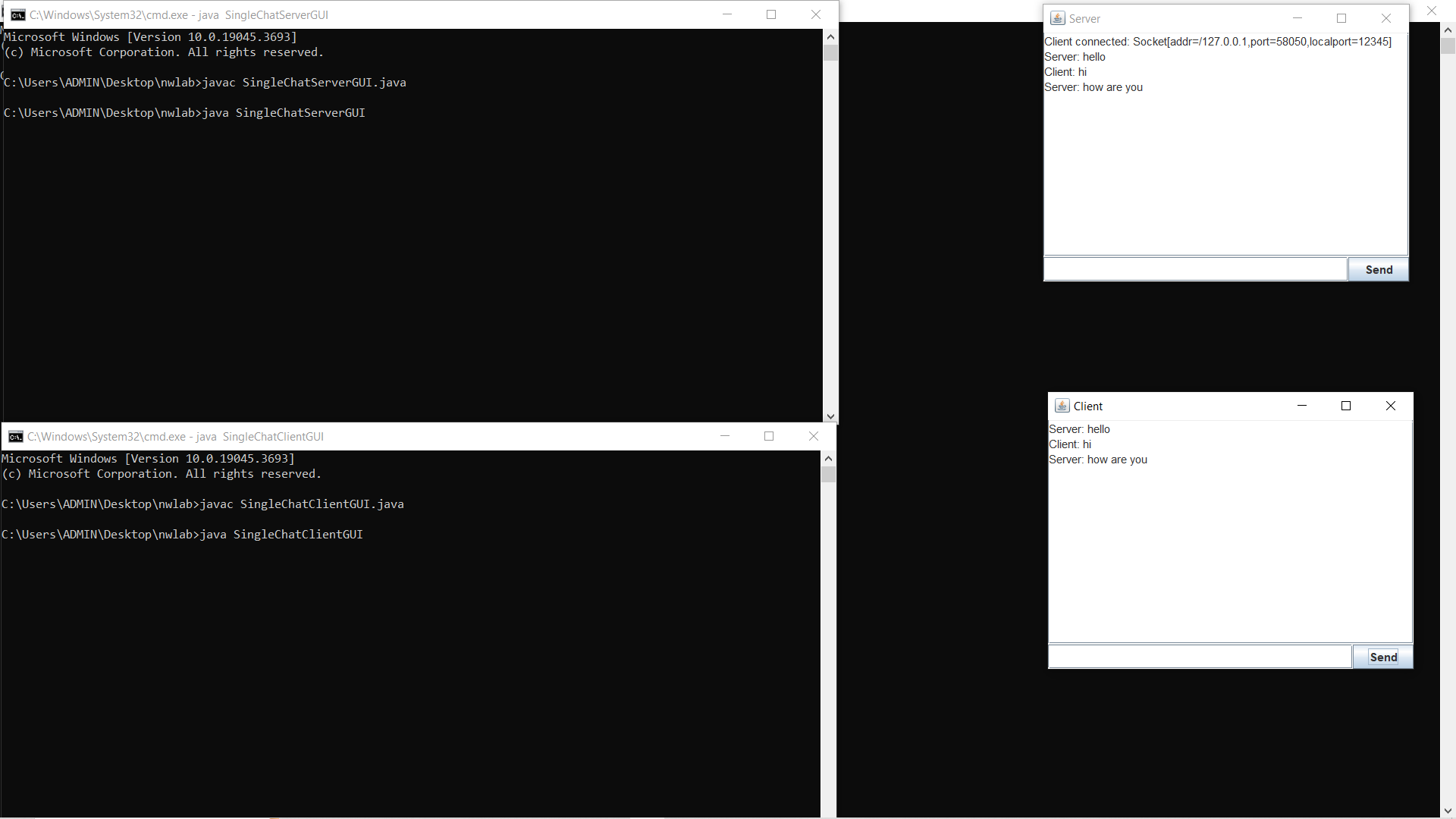
chatArea.append("Client: " + message + "\n");

messageField.setText("");}}

public static void main(String[] args) {

new SingleChatClientGUI();}}

**OUTPUT:**

****

# EXPERIMENT NO: 08

**AIM:**

Write a Java program to implement **MultiChatting application using GUI.**

**ALGORITHM:**

Server:

1. Import all necessary java libraries for networking and GUI.

2. Initialize server by creating a GUI window.

3. Handle clients using a separate class called clientHandle class.

4. Continuously handle and accept client connections using serverSocket.

5. Display the client messages on GUI.

6. Close resources when client disconnects.

7. Ensure GUI updates with Swing utilities for thread safety.

8. Instantiate the class to activate GUI and server functionalities.

9. Close all resources and end the program.

Client:

1. Import all necessary java libraries for networking and GUI.

2. Establish socket connection to the server.

3. create client window and setup components,.

4. Send messages to server when send button is clicked.

5. Continuously listen to server messages using input streams.

6. In main method create instance of the class and start itsfuntionality.

7. Close all resources and end the program.

**PROGRAM:**

Server:

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.IOException;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.Map;

import java.util.Scanner;

import java.util.concurrent.ConcurrentHashMap;

public class MultiChatServer extends JFrame {

private JTextArea chatArea;

private JTextField messageField;

private JButton sendButton;

private Map<Integer, PrintWriter> clientWriters = new ConcurrentHashMap<>();

private int clientIdCounter = 1;

public MultiChatServer() {

setTitle("Multi-Chat Server");

setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

setSize(400, 300);

setLayout(new BorderLayout());

chatArea = new JTextArea();

chatArea.setEditable(false);

JScrollPane scrollPane = new JScrollPane(chatArea);

getContentPane().add(scrollPane, BorderLayout.CENTER);

JPanel bottomPanel = new JPanel(new BorderLayout());

messageField = new JTextField();

bottomPanel.add(messageField, BorderLayout.CENTER);

sendButton = new JButton("Send");

sendButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

sendMessageToClient();} });

bottomPanel.add(sendButton, BorderLayout.EAST);

getContentPane().add(bottomPanel, BorderLayout.SOUTH);}

public void startServer() {

try (ServerSocket serverSocket = new ServerSocket(12345)) {

appendToChatArea("Server started. Waiting for clients...");

while (true) {

Socket clientSocket = serverSocket.accept();

appendToChatArea("New client connected: " + clientSocket);

PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

clientWriters.put(clientIdCounter, out);

Thread clientThread = new Thread(new ClientHandler(clientSocket, out, clientIdCounter));

clientThread.start();

clientIdCounter++; }

} catch (IOException e) {

e.printStackTrace();} }

private void appendToChatArea(String message) {

SwingUtilities.invokeLater(() -> {

chatArea.append(message + "\n");

chatArea.setCaretPosition(chatArea.getDocument().getLength());});}

private class ClientHandler implements Runnable {

private Socket socket;

private PrintWriter writer;

private Scanner in;

private int clientId;

public ClientHandler(Socket socket, PrintWriter writer, int clientId) {

this.socket = socket;

this.writer = writer;

this.clientId = clientId;}

@Override

public void run() {

try {

in = new Scanner(socket.getInputStream());

while (in.hasNextLine()) {

String message = in.nextLine();

appendToChatArea("Client " + clientId + ": " + message);

handleClientMessage(message); }} catch (IOException e) {

e.printStackTrace();

} finally {

try {

in.close();

socket.close();

clientWriters.remove(clientId);

} catch (IOException e) {

e.printStackTrace();}}}

private void handleClientMessage(String message) {

}}

private void sendMessageToClient() {

String message = messageField.getText();

if (!message.isEmpty()) {

int selectedClientId = Integer.parseInt(JOptionPane.showInputDialog("Enter Client ID to send the message:"));

PrintWriter clientWriter = clientWriters.get(selectedClientId);

if (clientWriter != null) {

clientWriter.println("Server: " + message);

appendToChatArea("Server: " + message);

} else {

JOptionPane.showMessageDialog(this, "Invalid Client ID!");

}

messageField.setText("");}}

public static void main(String[] args) {

MultiChatServer server = new MultiChatServer();

server.setVisible(true);

server.startServer();}}

Client:

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.IOException;

import java.io.PrintWriter;

import java.net.Socket;

import java.util.Scanner;

public class MultiChatClient extends JFrame {

private JTextArea chatArea;

private JTextField messageField;

private PrintWriter out;

private JButton sendbutton;

public MultiChatClient() {

setTitle("Client");

setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

setSize(400, 300);

setLayout(new BorderLayout());

chatArea = new JTextArea();

chatArea.setEditable(false);

JScrollPane scrollPane = new JScrollPane(chatArea);

getContentPane().add(scrollPane, BorderLayout.CENTER);

messageField = new JTextField();

JPanel bottomPanel = new JPanel(new BorderLayout());

bottomPanel.add(messageField, BorderLayout.CENTER);

sendbutton=new JButton("send");

sendbutton.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e){

sendMessage();} });

bottomPanel.add(sendbutton, BorderLayout.EAST);

getContentPane().add(bottomPanel, BorderLayout.SOUTH);}

public void connectToServer() {

try {

Socket socket = new Socket("localhost", 12345);

out = new PrintWriter(socket.getOutputStream(), true);

Scanner in = new Scanner(socket.getInputStream());

while (in.hasNextLine()) {

String message = in.nextLine();

appendToChatArea(message);}

} catch (IOException e) {

e.printStackTrace();}}

private void sendMessage() {

String message = messageField.getText();

appendToChatArea("Me: " + message);

out.println(message);

messageField.setText(""); }

private void appendToChatArea(String message) {

SwingUtilities.invokeLater(() -> {

chatArea.append(message + "\n");

chatArea.setCaretPosition(chatArea.getDocument().getLength());});}

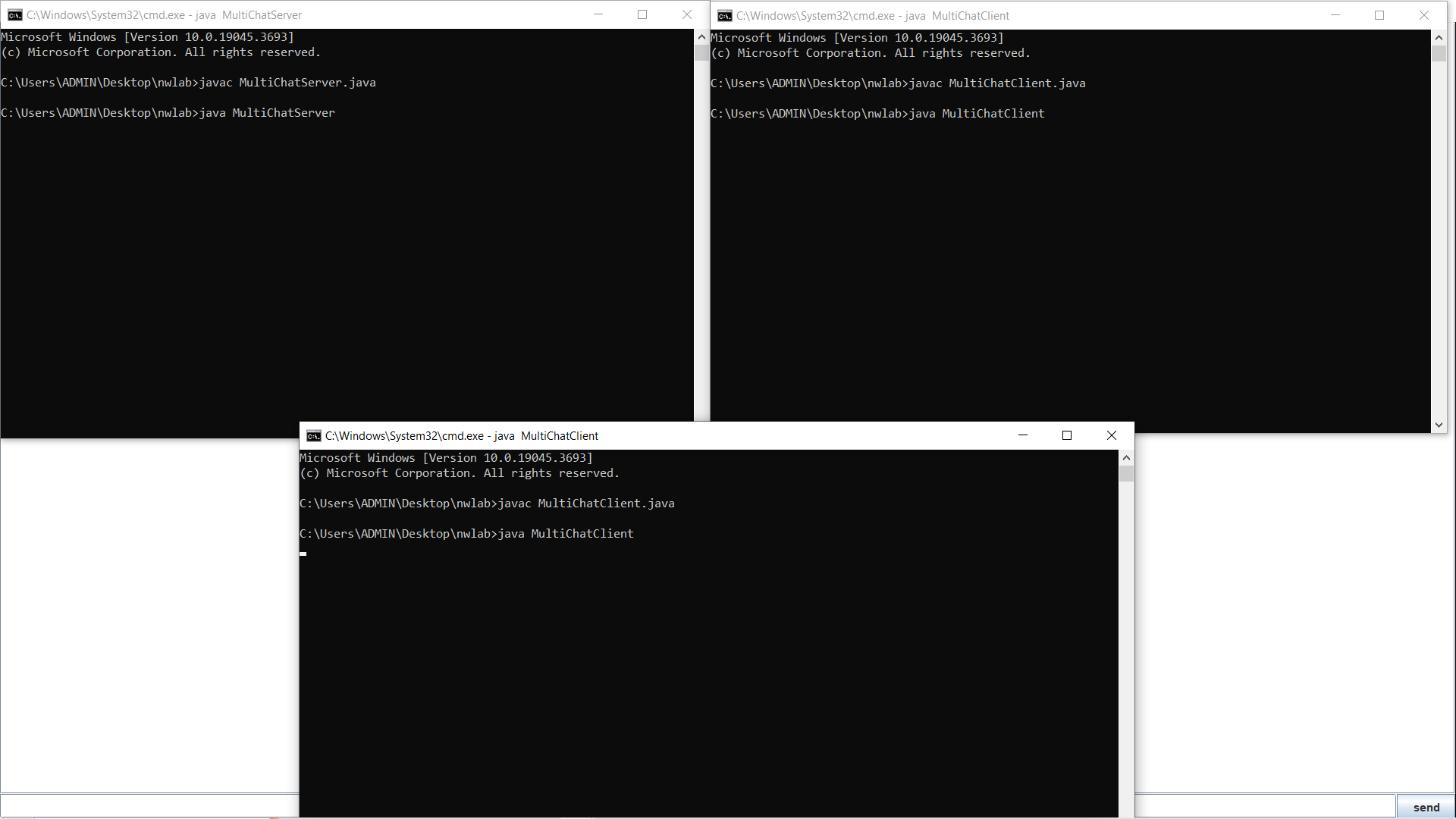
public static void main(String[] args) {

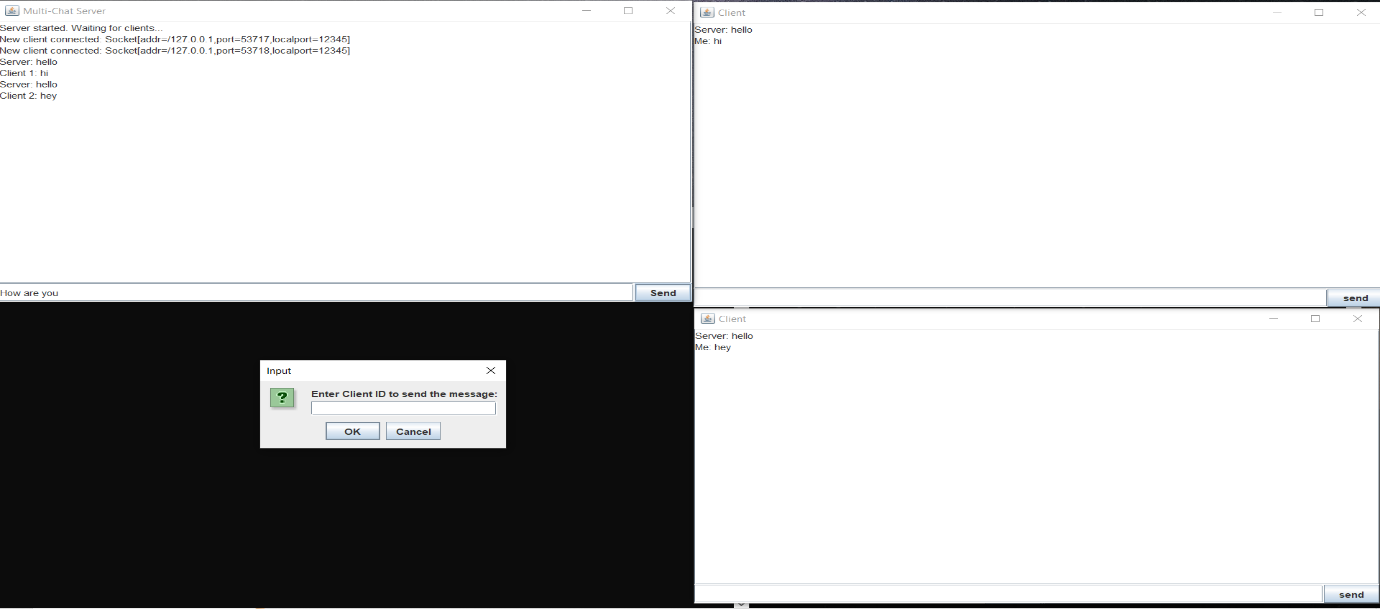
MultiChatClient client = new MultiChatClient();

client.setVisible(true);

client.connectToServer();}}

**OUTPUT:**

****

****

# EXPERIMENT NO: 09

**AIM:**

Write a Java program to implement **Error detection and correction using hamming code.**

**ALGORITHM:**

1. Read the number of bits for Hamming data (n) from the user.

2. Initialize an array 'a' of size 'n' to store the input data bits.

3. Read each bit of input data from the user and store it in the array 'a'.

4. Display the entered data bits.

5. Generate the Hamming code by calling the `generateCode` function, passing the input data array 'a'.

6. Display the generated Hamming code.

7. Read the position of a bit to alter for error simulation from the user.

8. If the user specifies a position (other than 0), alter the bit at that position in the Hamming code.

9. Display the sent code (Hamming code with or without an altered bit).

10. Call the `receive` function, passing the Hamming code and the number of parity bits.

11. In the `generateCode` function:

a. Determine the number of parity bits needed based on the size of the input data array.

b. Initialize an array 'b' to store the generated Hamming code.

c. Set parity bits in 'b' at positions corresponding to powers of 2.

d. Calculate and set data bits in 'b'.

e. Calculate and set parity bits using the `getParity` function.

f. Return the generated Hamming code 'b'.

12. In the `getParity` function:

a. Initialize 'parity' to 0.

b. Iterate through the Hamming code bits.

c. For each data bit contributing to the parity, calculate its position and update 'parity'.

13. In the `receive` function:

a. Initialize arrays to store parity values and syndrome.

b. Calculate and store parity values for each parity bit.

c. Determine the error location based on the syndrome.

d. If an error is detected, correct the bit and display the corrected code.

e. If no error is detected, indicate that there is no error in the received data.

f. Display the original data sent (excluding parity bits).

**PROGRAM:**

import java.util.\*;

class Hamming{

public static void main(String args[]) {

Scanner scan = new Scanner(System.in);

System.out.println("Enter the number of bits for the Hamming data:");

int n = scan.nextInt();

int a[] = new int[n];

for(int i=0 ; i < n ; i++){

System.out.println("Enter bit no. " + (n-i) + ":");

a[n-i-1] = scan.nextInt(); }

System.out.println("You entered:");

for(int i=0 ; i < n ; i++){

System.out.print(a[n-i-1]);}

System.out.println();

int b[] = generateCode(a);

System.out.println("Generated code is:");

for(int i=0 ; i < b.length ; i++) {

System.out.print(b[b.length-i-1]);}

System.out.println();

System.out.println("Enter position of a bit to alter to check for error detection at the receiver end (0 for no error):");

int error = scan.nextInt();

if(error != 0){

b[error-1] = (b[error-1]+1)%2;}

System.out.println("Sent code is:");

for(int i=0 ; i < b.length ; i++) {

System.out.print(b[b.length-i-1]); }

System.out.println();

receive(b, b.length - a.length);}

static int[] generateCode(int a[]){

int b[];

int i=0, parity\_count=0 ,j=0, k=0;

while(i < a.length){

if(Math.pow(2,parity\_count) == i+parity\_count + 1){

parity\_count++;}

else{

i++;}}

b = new int[a.length + parity\_count];

for(i=1 ; i <= b.length ; i++){

if(Math.pow(2, j) == i){

b[i-1] = 2;

j++;}

else{

b[k+j] = a[k++];}}

for(i=0 ; i < parity\_count ; i++){

b[((int) Math.pow(2, i))-1] = getParity(b, i);}

return b;}

static int getParity(int b[], int power) {

int parity = 0;

for(int i=0 ; i < b.length ; i++){

if(b[i] != 2){

int k = i+1;

String s = Integer.toBinaryString(k);

int x = ((Integer.parseInt(s))/((int) Math.pow(10, power)))%10;

if(x == 1){

if(b[i] == 1) {

parity = (parity+1)%2;}}}}

return parity;}

static void receive(int a[], int parity\_count) {

int power;

int parity[] = new int[parity\_count];

String syndrome = new String();

for(power=0 ; power < parity\_count ; power++){

for(int i=0 ; i < a.length ; i++){

int k = i+1;

String s = Integer.toBinaryString(k);

int bit = ((Integer.parseInt(s))/((int) Math.pow(10, power)))%10;

if(bit == 1){

if(a[i] == 1){

parity[power] = (parity[power]+1)%2;}}}

syndrome = parity[power] + syndrome;}

int error\_location = Integer.parseInt(syndrome, 2);

if(error\_location != 0){

System.out.println("Error is at location " + error\_location + ".");

a[error\_location-1] = (a[error\_location-1]+1)%2;

System.out.println("Corrected code is:");

for(int i=0 ; i < a.length ; i++)

{ System.out.print(a[a.length-i-1]);}

System.out.println();}else{

System.out.println("There is no error in the received data.");}

System.out.println("Original data sent was:");

power = parity\_count-1;

for(int i=a.length ; i > 0 ; i--){

if(Math.pow(2, power) != i){

System.out.print(a[i-1]);}

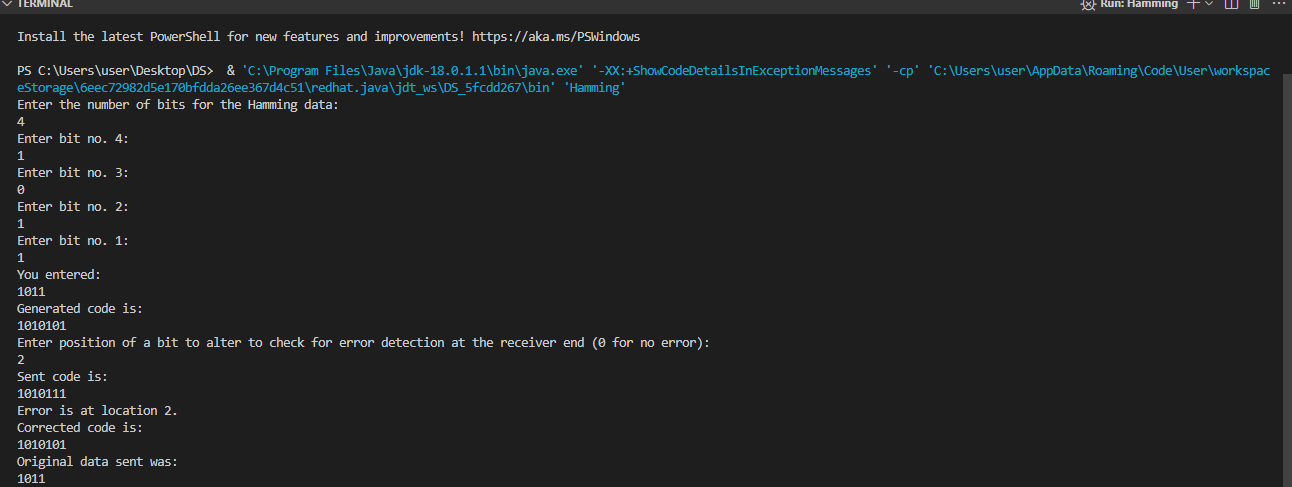
else{

power--;}}

System.out.println();

}}

**OUTPUT:**

****

# EXPERIMENT NO: 10

**AIM:**

Write a Java program to implement **Huffman code generation.**

**ALGORITHM:**

1. Create Node Class:

- Create a class `Node` to represent a node in the Huffman tree, storing character, frequency, left, and right child.

2. Huffman Tree Construction:

a. Create a method `createHuffmanTree` that takes a string as input.

b. Create a frequency map for each character in the input string.

c. Use a priority queue to build a min-heap of nodes based on character frequencies.

d. Construct the Huffman tree by repeatedly merging the two nodes with the lowest frequencies until there's only one node left.

3. Huffman Code Generation

a. After constructing the Huffman tree, create a method `encodeData` to generate Huffman codes for each character.

b. Traverse the Huffman tree and assign codes based on the path from the root to each leaf.

4. Print Huffman Codes:

- Print the generated Huffman codes for each character.

5. Encode Original String:

a. Print the initial input string.

b. Encode the input string using the generated Huffman codes and print the encoded string.

6. Decode String:

a. Create a method `decodeData` to decode the encoded string based on the Huffman tree.

b. Print the decoded string.

7. Helper Method - isLeaf:

- Create a helper method `isLeaf` to check if a given node is a leaf node in the Huffman tree.

8. Main Method:

a. Prompt the user to enter a string.

b. Read the input string.

c. Close the scanner.

d. Call the `createHuffmanTree` method with the input string to perform the Huffman coding.

**PROGRAM:**

import java.util.Comparator;

import java.util.HashMap;

import java.util.Map;

import java.util.PriorityQueue;

import java.util.Scanner;

class Node {

Character ch;

Integer freq;

Node left = null;

Node right = null;

Node(Character ch, Integer freq) {

this.ch = ch;

this.freq = freq;}

public Node(Character ch, Integer freq, Node left, Node right) {

this.ch = ch;

this.freq = freq;

this.left = left;

this.right = right; }}

public class huffman {

public static void createHuffmanTree(String text) {

if (text == null || text.length() == 0)

return;

Map<Character, Integer> freq = new HashMap<>();

for (char c : text.toCharArray()) {

freq.put(c, freq.getOrDefault(c, 0) + 1); }

PriorityQueue<Node> pq = new PriorityQueue<>(Comparator.comparingInt(l -> l.freq));

for (var entry : freq.entrySet()) {

pq.add(new Node(entry.getKey(), entry.getValue()));}

while (pq.size() != 1) {

Node left = pq.poll();

Node right = pq.poll();

int sum = left.freq + right.freq;

pq.add(new Node(null, sum, left, right));}

Node root = pq.peek();

Map<Character, String> huffmanCode = new HashMap<>();

encodeData(root, "", huffmanCode);

System.out.println("Huffman Codes of the characters are: " + huffmanCode);

System.out.println("The initial string is: " + text);

StringBuilder sb = new StringBuilder();

for (char c : text.toCharArray()) {

sb.append(huffmanCode.get(c));}

System.out.println("The encoded string is: " + sb);

System.out.print("The decoded string is: ");

if (isLeaf(root)) {

while (root.freq-- > 0) {

System.out.print(root.ch); }

} else {

int index = -1;

while (index < sb.length() - 1) {

index = decodeData(root, index, sb); } }}

public static void encodeData(Node root, String str, Map<Character, String> huffmanCode) {

if (root == null)

return;

if (isLeaf(root))

huffmanCode.put(root.ch, str.length() > 0 ? str : "1");

encodeData(root.left, str + '0', huffmanCode);

encodeData(root.right, str + '1', huffmanCode); }

public static int decodeData(Node root, int index, StringBuilder sb) {

if (root == null)

return index;

if (isLeaf(root)) {

System.out.print(root.ch);

return index;}

index++;

root = (sb.charAt(index) == '0') ? root.left : root.right;

index = decodeData(root, index, sb);

return index; }

public static boolean isLeaf(Node root) {

return root.left == null && root.right == null;}

public static void main(String args[]) {

Scanner scanner = new Scanner(System.in);

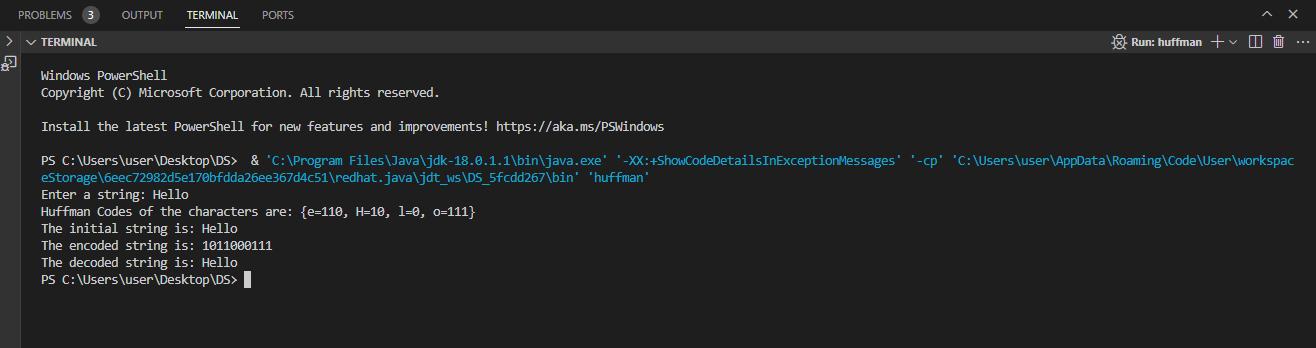
System.out.print("Enter a string: ");

String text = scanner.nextLine();

scanner.close();

createHuffmanTree(text); }}

**OUTPUT:**

****

# EXPERIMENT NO: 11

**AIM:**

Write a Java program for **simulation to test ARP and RARP**

**ALGORITHM:**

ARP:

1. Create ARPCache class with a private map to store IP-MAC address mappings.

- Initialize the cache in the constructor.

2. Create ARPSimulator class responsible for simulating ARP requests.

- Accept an ARPCache instance in the constructor.

3. In ARPSimulator, implement a method simulateARP(ipAddress) to simulate an ARP request:

a. Call getMacAddressFromARP(ipAddress).

b. Print the resulting MAC address for the given IP address.

4. In ARPSimulator, implement a private method getMacAddressFromARP(ipAddress):

a. Check if the MAC address for the given IP is already in the cache.

b. If found, return the cached MAC address.

c. If not found, generate a random MAC address, add it to the cache, and return the generated MAC address.

5. Create the main ARP class:

a. Instantiate ARPCache and ARPSimulator objects.

6. Simulate ARP requests in the main method using the ARPSimulator object.

7. End of Algorithm.

RARP:

1. Create RARPCache class with a private map to store MAC-IP address mappings.

- Initialize the cache in the constructor.

2. Create RARPSimulator class responsible for simulating RARP requests.

- Accept an RARPCache instance in the constructor.

3. In RARPSimulator, implement a method simulateRARP(macAddress) to simulate a RARP request:

a. Call getIPAddressFromRARP(macAddress).

b. Print the resulting IP address for the given MAC address.

4. In RARPSimulator, implement a private method getIPAddressFromRARP(macAddress):

a. Check if the IP address for the given MAC is already in the cache.

b. If found, return the cached IP address.

c. If not found, generate a random IP address, add it to the cache, and return the generated IP address.

5. Create the main Rarp class:

a. Instantiate RARPCache and RARPSimulator objects.

6. Simulate RARP requests in the main method using the RARPSimulator object.

7. End of Algorithm.

**PROGRAM:**

ARP:

import java.util.HashMap;

import java.util.Map;

class ARPCache{

private Map<String, String> cache;

public ARPCache() {

this.cache = new HashMap<>();

}public void addToCache(String ipAddress, String macAddress) {

cache.put(ipAddress, macAddress);

}public String getMacAddress(String ipAddress) {

return cache.get(ipAddress);}

}class ARPSimulator {

private ARPCache arpCache;

public ARPSimulator(ARPCache arpCache) {

this.arpCache = arpCache;}

public void simulateARP(String ipAddress) {

String macAddress = getMacAddressFromARP(ipAddress);

System.out.println("MAC Address for IP " + ipAddress + ": " + macAddress);}

private String getMacAddressFromARP(String ipAddress) {

String macAddress = arpCache.getMacAddress(ipAddress);

if (macAddress == null) {

macAddress = generateRandomMAC();

arpCache.addToCache(ipAddress, macAddress);}

}return macAddress;}

private String generateRandomMAC() {

return "00:1A:2B:3C:4D:5E";}}

public class ARP{

public static void main(String[] args) {

ARPCache arpCache = new ARPCache();

ARPSimulator arpSimulator = new ARPSimulator(arpCache);

arpSimulator.simulateARP("192.168.1.1");

arpSimulator.simulateARP("192.168.1.2");

arpSimulator.simulateARP("192.168.1.1");}}

RARP:

import java.util.HashMap;

import java.util.Map;

class RARPCache {

private Map<String, String> cache;

public RARPCache() {

this.cache = new HashMap<>();}

public void addToCache(String macAddress, String ipAddress) {

cache.put(macAddress, ipAddress);}

public String getIPAddress(String macAddress) {

return cache.get(macAddress);}}

class RARPSimulator {

private RARPCache rarpCache;

public RARPSimulator(RARPCache rarpCache) {

this.rarpCache = rarpCache;}

public void simulateRARP(String macAddress) {

String ipAddress = getIPAddressFromRARP(macAddress);

System.out.println("IP Address for MAC " + macAddress + ": " + ipAddress);}

private String getIPAddressFromRARP(String macAddress) {

String ipAddress = rarpCache.getIPAddress(macAddress);

if (ipAddress == null) {

ipAddress = generateRandomIP();

rarpCache.addToCache(macAddress, ipAddress);}

return ipAddress;}

private String generateRandomIP() {

return "192.168.1.1";}}

public class Rarp {

public static void main(String[] args) {

RARPCache rarpCache = new RARPCache();

RARPSimulator rarpSimulator = new RARPSimulator(rarpCache);

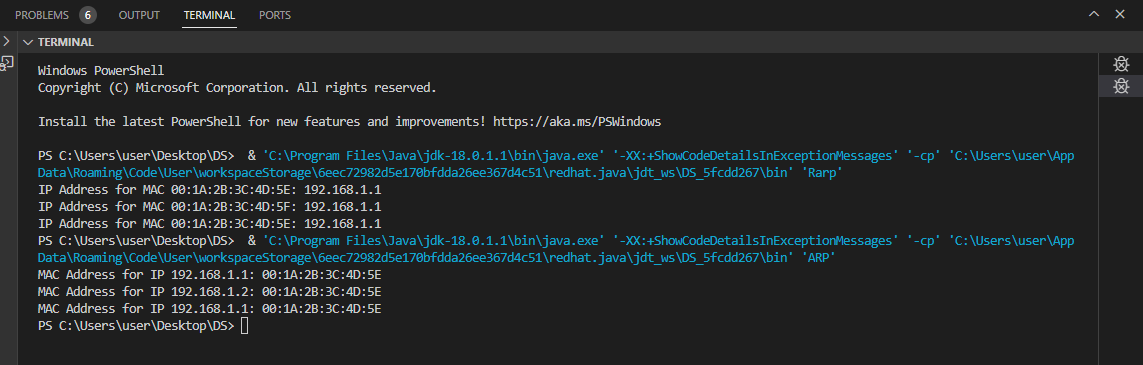
rarpSimulator.simulateRARP("00:1A:2B:3C:4D:5E");

rarpSimulator.simulateRARP("00:1A:2B:3C:4D:5F");

rarpSimulator.simulateRARP("00:1A:2B:3C:4D:5E");}}

**OUTPUT:**

RARP and ARP

****