**Write a program in Java for simplex communication using Connection Oriented service**

**TCP Server**

import java.io.\*;

import java.net.\*;

public class TCPServer {

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

int port = 12345;

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

Socket socket = serverSocket.accept();

InputStream input = socket.getInputStream();

BufferedReader reader = new BufferedReader(new InputStreamReader(input));

String line = reader.readLine();

System.out.println("Received: " + line);

}

}

}

**TCP Client**

import java.io.\*;

import java.net.\*;

public class TCPClient {

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

String hostname = "localhost";

int port = 12345;

String message = "Hello, Server!";

try (Socket socket = new Socket(hostname, port)) {

OutputStream output = socket.getOutputStream();

PrintWriter writer = new PrintWriter(output, true);

System.out.println("Sending: " + message);

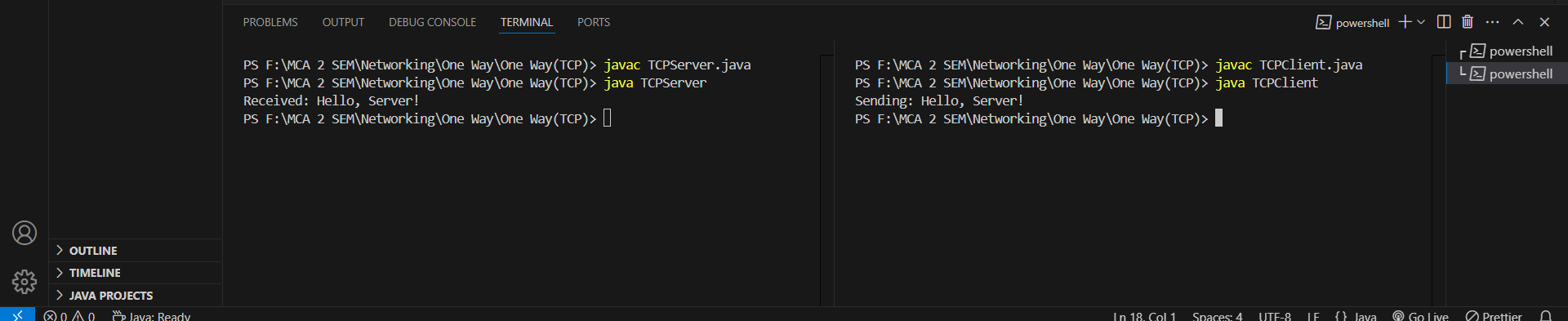
writer.println(message);

}

}

}

**Output:**



**Write a program in Java for simplex communication using Connection Less service.**

**UDP Server**

import java.io.\*;

import java.net.\*;

public class UDPServer {

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

int port = 12345;

DatagramSocket socket = new DatagramSocket(port);

byte[] buffer = new byte[1024];

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

System.out.println("Server is listening on port " + port);

socket.receive(packet); // Receive a packet from the client

String message = new String(packet.getData(), 0, packet.getLength());

System.out.println("Received: " + message);

socket.close();

}

}

**UDP Client**

import java.io.\*;

import java.net.\*;

public class UDPClient {

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

String hostname = "localhost";

int port = 12345;

String message = "Hello, Server!";

DatagramSocket socket = new DatagramSocket();

byte[] buffer = message.getBytes();

DatagramPacket packet = new DatagramPacket(buffer, buffer.length, InetAddress.getByName(hostname), port);

System.out.println("Sending: " + message);

socket.send(packet);

socket.close();

}

}

**Output:**



**Write a program in Java for full duplex communication using Connection Oriented service.**

**TCP Server**

import java.io.\*;

import java.net.\*;

public class TCPServer {

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

int port = 5000;

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

System.out.println("Server started. Waiting for a client...");

Socket clientSocket = serverSocket.accept();

System.out.println("Client connected.");

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

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

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

String received, serverMessage;

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

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

System.out.print("Enter your message: ");

serverMessage = consoleReader.readLine();

out.println("Server: " + serverMessage);

}

clientSocket.close();

}

}

}

**TCP Client**

import java.io.\*;

import java.net.\*;

public class TCPClient {

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

String serverIP = "localhost";

int port = 5000;

try (Socket socket = new Socket(serverIP, port)) {

System.out.println("Connected to the server.");

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

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

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

String message;

while(true){

System.out.print("Enter your message: ");

message = consoleReader.readLine();

if (message==null){

break;

}

out.println(message);

System.out.println(in.readLine());

}

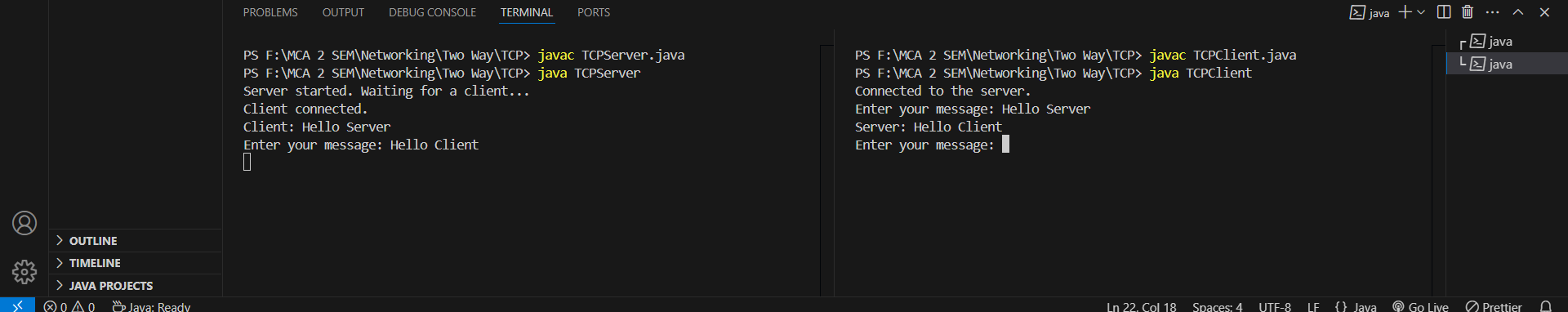
socket.close();

}

}

}

**Output:**



**Write a program in Java for full duplex communication using Connection Less service.**

**UDP Server**

import java.net.\*;

public class UDPServer {

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

int port = 5000;

DatagramSocket socket = new DatagramSocket(port); // Create a socket on port 5000

byte[] receiveData = new byte[1024]; // Buffer to hold received data

System.out.println("Server started. Waiting for client...");

while (true) {

// Receive packet from client

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

socket.receive(receivePacket); // Receive message from client

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

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

// Prepare server response

String responseMessage = "Server: " + clientMessage; // Echo back the received message

// Send response back to client

InetAddress clientAddress = receivePacket.getAddress();

int clientPort = receivePacket.getPort();

DatagramPacket sendPacket = new DatagramPacket(responseMessage.getBytes(), responseMessage.length(), clientAddress, clientPort);

socket.send(sendPacket); // Send response

}

}

}

**UDP Client**

import java.net.\*;

import java.util.Scanner;

public class UDPClient {

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

String serverIP = "localhost"; // Server's IP address

int serverPort = 5000; // Server's port

DatagramSocket socket = new DatagramSocket(); // Create a socket to send and receive packets

InetAddress serverAddress = InetAddress.getByName(serverIP); // Get the server's IP address

Scanner scanner = new Scanner(System.in); // To read user input

while (true) {

System.out.print("Enter your message: ");

String message = scanner.nextLine(); // Read the message from user input

if ("exit".equalsIgnoreCase(message)) {

break; // Exit loop if user types "exit"

}

// Send message to server

DatagramPacket sendPacket = new DatagramPacket(message.getBytes(), message.length(), serverAddress, serverPort);

socket.send(sendPacket);

// Receive response from server

byte[] receiveData = new byte[1024];

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

socket.receive(receivePacket); // Wait for server's response

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

System.out.println("Server: " + serverResponse); // Print server's response

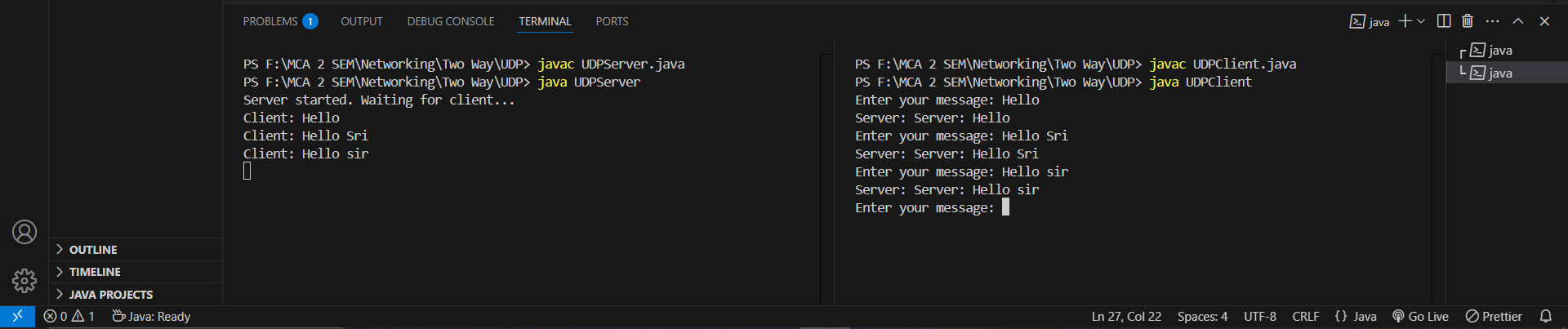
}

socket.close(); // Close the socket when done

}

}

**Output:**



**Implement an Unrestricted Simplex Protocol of Data Link Layer.**

import java.util.concurrent.ArrayBlockingQueue;

import java.util.concurrent.BlockingQueue;

// Frame class represents a data frame.

class Frame {

private String data;

public Frame(String data) {

this.data = data;

}

public String getData() {

return data;

}

@Override

public String toString() {

return "Frame{data='" + data + "'}";

}

}

// Sender class sends data frames to the receiver.

class Sender implements Runnable {

private BlockingQueue<Frame> queue;

public Sender(BlockingQueue<Frame> queue) {

this.queue = queue;

}

public void sendFrame(Frame frame) {

try {

// Send the frame to the queue

queue.put(frame);

System.out.println("Sender: Sent frame -> " + frame);

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

System.err.println("Sender interrupted");

}

}

@Override

public void run() {

// Simulating sending frames in an infinite loop

for (int i = 1; i <= 5; i++) {

sendFrame(new Frame("Data Frame " + i));

try {

Thread.sleep(1000); // Delay to simulate time between sending frames

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

}

// Receiver class receives data frames from the sender.

class Receiver implements Runnable {

private BlockingQueue<Frame> queue;

public Receiver(BlockingQueue<Frame> queue) {

this.queue = queue;

}

public void receiveFrame() {

try {

// Take the frame from the queue

Frame frame = queue.take();

System.out.println("Receiver: Received frame -> " + frame);

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

System.err.println("Receiver interrupted");

}

}

@Override

public void run() {

// Continuously listen for incoming frames

while (true) {

receiveFrame();

}

}

}

// Main class that sets up the USP protocol communication.

public class UnrestrictedSimplexProtocol {

public static void main(String[] args) {

// Create a blocking queue for communication between sender and receiver

BlockingQueue<Frame> queue = new ArrayBlockingQueue<>(10);

// Create sender and receiver threads

Sender sender = new Sender(queue);

Receiver receiver = new Receiver(queue);

Thread senderThread = new Thread(sender);

Thread receiverThread = new Thread(receiver);

// Start both threads

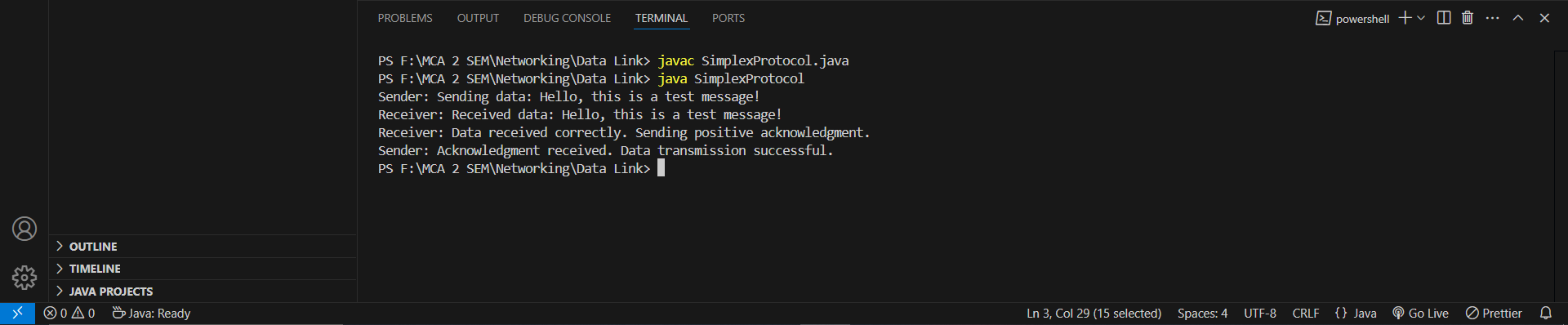
senderThread.start();

receiverThread.start();

}

}

**Output:**



**Implement A Simplex Stop-and-Wait Protocol of Data Link Layer.**

import java.util.concurrent.ArrayBlockingQueue;

import java.util.concurrent.BlockingQueue;

public class SimplexStopAndWaitProtocol {

// Sender class

static class Sender {

private BlockingQueue<String> sendQueue;

private BlockingQueue<String> ackQueue;

public Sender(BlockingQueue<String> sendQueue, BlockingQueue<String> ackQueue) {

this.sendQueue = sendQueue;

this.ackQueue = ackQueue;

}

public void send(String message) throws InterruptedException {

System.out.println("Sender: Sending message: " + message);

sendQueue.put(message); // Send message

String ack = ackQueue.take(); // Wait for ACK

System.out.println("Sender: Received ACK: " + ack);

}

}

// Receiver class

static class Receiver {

private BlockingQueue<String> sendQueue;

private BlockingQueue<String> ackQueue;

public Receiver(BlockingQueue<String> sendQueue, BlockingQueue<String> ackQueue) {

this.sendQueue = sendQueue;

this.ackQueue = ackQueue;

}

public void receive() throws InterruptedException {

while (true) {

String message = sendQueue.take(); // Wait for message

System.out.println("Receiver: Received message: " + message);

// Simulate processing the message

System.out.println("Receiver: Acknowledging message: " + message);

ackQueue.put("ACK for: " + message); // Send acknowledgment

}

}

}

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

// Create blocking queues to simulate message transmission

BlockingQueue<String> sendQueue = new ArrayBlockingQueue<>(1);

BlockingQueue<String> ackQueue = new ArrayBlockingQueue<>(1);

// Create sender and receiver

Sender sender = new Sender(sendQueue, ackQueue);

Receiver receiver = new Receiver(sendQueue, ackQueue);

// Start receiver in a separate thread to listen for incoming messages

Thread receiverThread = new Thread(() -> {

try {

receiver.receive();

} catch (InterruptedException e) {

e.printStackTrace();

}

});

receiverThread.start();

// Simulate the sender sending messages

String[] messages = {"Message 1", "Message 2", "Message 3"};

for (String message : messages) {

sender.send(message);

Thread.sleep(1000); // Simulate time delay between messages

}

// Let receiver finish before terminating

receiverThread.interrupt();

}

}

**Output:**

