**Implement A Simplex Protocol for a Noisy Channel of Data Link Layer**

import java.util.Random;

import java.util.Scanner;

public class SimplexProtocolNoisyChannel {

static class Sender {

private Receiver receiver;

private int messageCount;

public Sender(Receiver receiver, int messageCount) {

this.receiver = receiver;

this.messageCount = messageCount;

}

public void sendMessages(Scanner scanner) {

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

System.out.print("Enter message " + i + ": ");

String message = scanner.nextLine();

boolean messageSent = false;

while (!messageSent) {

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

boolean errorOccurred = receiver.receiveMessage(message);

// Simulate waiting for acknowledgment

if (!errorOccurred) {

System.out.println("Sender: Received ACK for Message " + i);

messageSent = true; // ACK received, message sent successfully

} else {

System.out.println("Sender: Error occurred, resending Message " + i);

}

}

}

}

static class Receiver {

private Random random = new Random();

public boolean receiveMessage(String message) {

// Simulate a noisy channel with a probability of error

boolean errorOccurred = random.nextInt(10) < 3; // 30% chance of error

if (errorOccurred) {

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

return true; // Indicate that an error occurred

} else {

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

sendAcknowledgment();

return false; // No error occurred

}

}

private void sendAcknowledgment() {

System.out.println("Receiver: Sending ACK");

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of messages to send: ");

int numberOfMessages = Integer.parseInt(scanner.nextLine());

Receiver receiver = new Receiver();

Sender sender = new Sender(receiver, numberOfMessages);

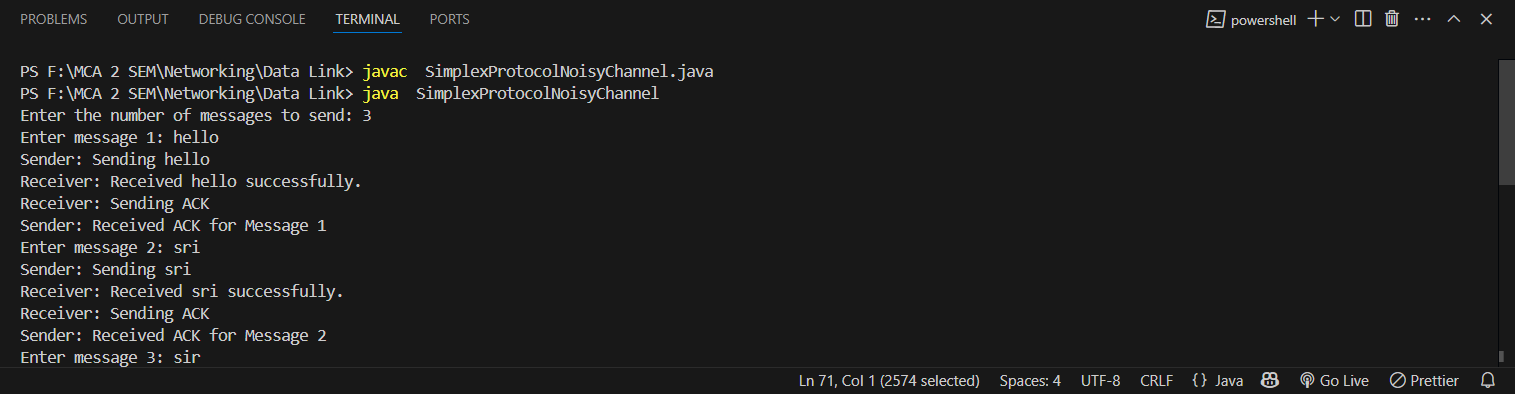
sender.sendMessages(scanner);

scanner.close();

}

}

**Output :**

****

**Implement the following Sliding Window Protocol:**

1. **A One-Bit Sliding Window Protocol**

import java.util.Scanner;

import java.util.Random;

class SenderOneBit {

private boolean isFrameAcknowledged = false;

// Method to send a frame and wait for acknowledgment

public void sendFrame(int frame) {

isFrameAcknowledged = false;

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

if (isFrameCorrupted(frame)) {

System.out.println("Sender: Frame corrupted during transmission.");

} else {

System.out.println("Sender: Frame sent successfully.");

}

}

// Method to acknowledge the frame

public void acknowledgeFrame() {

isFrameAcknowledged = true;

System.out.println("Sender: Frame acknowledged by receiver.");

}

// Simulate frame corruption

private boolean isFrameCorrupted(int frame) {

Random rand = new Random();

return rand.nextInt(100) < 20; // 20% chance of corruption

}

}

class ReceiverOneBit {

private Random rand = new Random();

// Method to receive a frame and check for errors

public void receiveFrame(int frame, SenderOneBit sender) {

System.out.println("Receiver: Frame received: " + frame);

boolean isCorrupted = isFrameCorrupted(frame);

if (isCorrupted) {

System.out.println("Receiver: Frame corrupted. Requesting retransmission.");

return;

}

System.out.println("Receiver: Frame received successfully.");

sender.acknowledgeFrame();

}

// Simulate frame corruption

private boolean isFrameCorrupted(int frame) {

return rand.nextInt(100) < 20; // 20% chance of corruption

}

}

public class OneBitSlidingWindowProtocol {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

SenderOneBit sender = new SenderOneBit();

ReceiverOneBit receiver = new ReceiverOneBit();

System.out.println("One-Bit Sliding Window Protocol Simulation");

while (true) {

System.out.print("\nEnter frame to send (0 or 1) or type 'exit' to quit: ");

String input = scanner.nextLine();

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

System.out.println("Exiting the simulation...");

break;

try {

int frame = Integer.parseInt(input);

// Sender sends the frame

sender.sendFrame(frame);

// Receiver receives the frame

receiver.receiveFrame(frame, sender);

} catch (NumberFormatException e) {

System.out.println("Invalid input. Please enter 0 or 1 for the frame.");

}

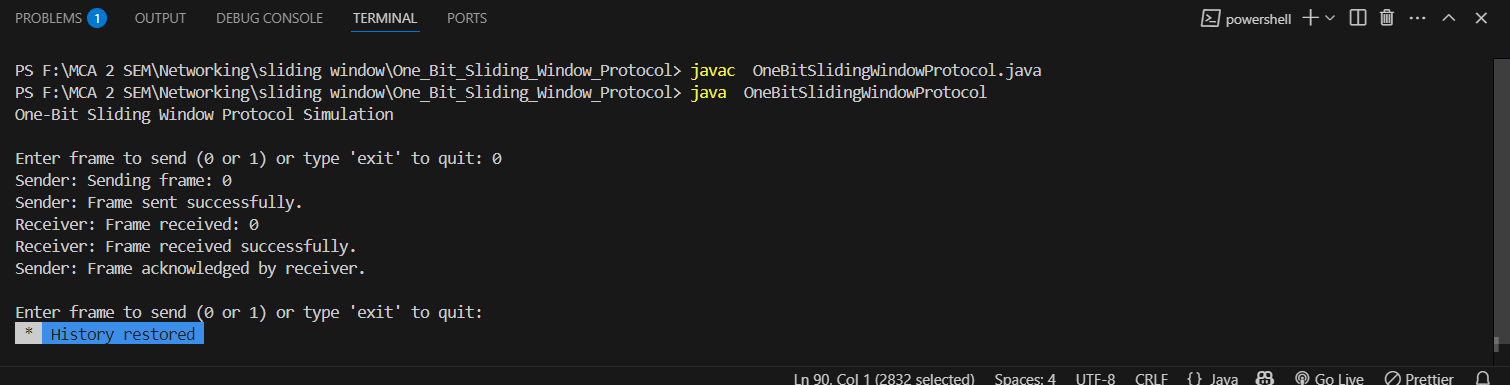
}

scanner.close();

}

}

**Output :**

****

**Implement the following Sliding Window Protocol:**

1. **A Protocol Using Go Back N**

import java.util.Scanner;

import java.util.Random;

class SenderGoBackN {

private int windowSize = 4;

private boolean[] ackReceived = new boolean[windowSize];

public void sendFrames(int[] frames) {

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

System.out.println("Sender: Sending frame: " + frames[i]);

if (isFrameCorrupted(frames[i])) {

System.out.println("Sender: Frame " + frames[i] + " corrupted during transmission.");

}

}

}

public void acknowledgeFrame(int frame) {

ackReceived[frame % windowSize] = true;

System.out.println("Sender: Acknowledgment received for frame: " + frame);

}

private boolean isFrameCorrupted(int frame) {

Random rand = new Random();

return rand.nextInt(100) < 20;

}

}

class ReceiverGoBackN {

private Random rand = new Random();

public void receiveFrames(int[] frames, SenderGoBackN sender) {

for (int frame : frames) {

System.out.println("Receiver: Frame received: " + frame);

if (isFrameCorrupted(frame)) {

System.out.println("Receiver: Frame corrupted. Retransmission needed.");

continue;

}

System.out.println("Receiver: Frame received successfully.");

sender.acknowledgeFrame(frame);

}

}

private boolean isFrameCorrupted(int frame) {

return rand.nextInt(100) < 20;

}

}

public class GoBackNSlidingWindowProtocol {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

SenderGoBackN sender = new SenderGoBackN();

ReceiverGoBackN receiver = new ReceiverGoBackN();

System.out.println("Go-Back-N Sliding Window Protocol Simulation");

while (true) {

System.out.print("\nEnter frames to send (space-separated integers) or 'exit' to quit: ");

String input = scanner.nextLine();

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

System.out.println("Exiting the simulation...");

break;

}

try {

String[] inputStrings = input.split(" ");

int[] frames = new int[inputStrings.length];

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

frames[i] = Integer.parseInt(inputStrings[i]);

}

sender.sendFrames(frames);

receiver.receiveFrames(frames, sender);

} catch (NumberFormatException e) {

System.out.println("Invalid input. Please enter integers only.");

}

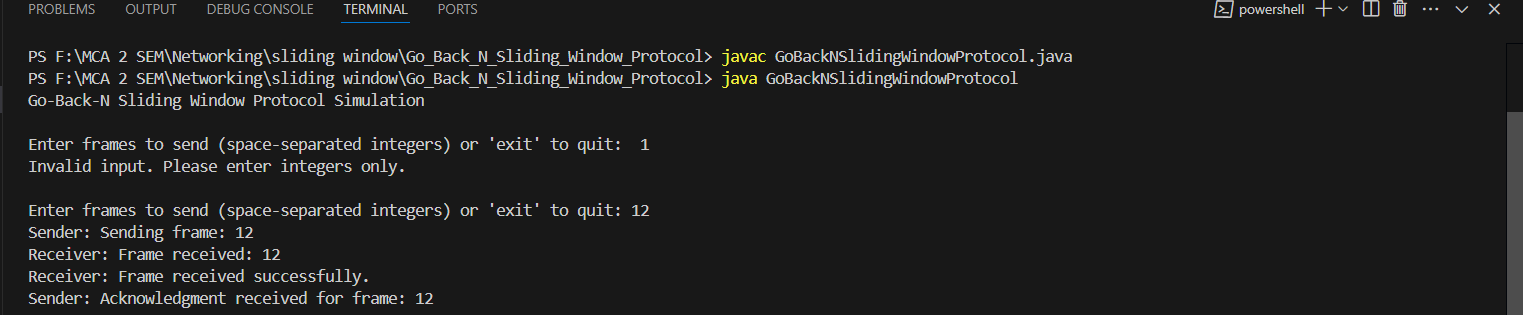
}

scanner.close();

}

}

**Output :**

****

**Implement the following Sliding Window Protocol:**

1. **A Protocol Using Selective Repeat**

import java.util.Scanner;

import java.util.Random;

class SenderSelectiveRepeat {

private boolean[] ackReceived;

public SenderSelectiveRepeat(int windowSize) {

ackReceived = new boolean[windowSize];

}

// Send frames to the receiver

public void sendFrames(int[] frames) {

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

System.out.println("Sender: Sending frame: " + frames[i]);

if (isFrameCorrupted(frames[i])) {

System.out.println("Sender: Frame " + frames[i] + " corrupted during transmission.");

}

}

}

// Acknowledge the received frame

public void acknowledgeFrame(int frame) {

ackReceived[frame] = true;

System.out.println("Sender: Acknowledgment received for frame: " + frame);

}

// Simulate frame corruption

private boolean isFrameCorrupted(int frame) {

Random rand = new Random();

return rand.nextInt(100) < 20; // 20% chance of corruption

}}

class ReceiverSelectiveRepeat {

private Random rand = new Random();

// Method to receive frames and send acknowledgment

public void receiveFrames(int[] frames, SenderSelectiveRepeat sender) {

for (int frame : frames) {

System.out.println("Receiver: Frame received: " + frame);

if (isFrameCorrupted(frame)) {

System.out.println("Receiver: Frame corrupted. Retransmission needed.");

continue;

}

System.out.println("Receiver: Frame received successfully.");

sender.acknowledgeFrame(frame);

}

}

// Simulate frame corruption

private boolean isFrameCorrupted(int frame) {

return rand.nextInt(100) < 20; // 20% chance of corruption

}

}

public class SelectiveRepeatSlidingWindowProtocol {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

SenderSelectiveRepeat sender = new SenderSelectiveRepeat(4);

ReceiverSelectiveRepeat receiver = new ReceiverSelectiveRepeat();

System.out.println("Selective Repeat Sliding Window Protocol Simulation");

while (true) {

System.out.print("\nEnter frames to send (space-separated integers) or 'exit' to quit: ");

String input = scanner.nextLine();

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

System.out.println("Exiting the simulation...");

break;

}

try {

String[] inputStrings = input.split(" ");

int[] frames = new int[inputStrings.length];

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

frames[i] = Integer.parseInt(inputStrings[i]);

}

// Sender sends frames

sender.sendFrames(frames);

// Receiver receives frames

receiver.receiveFrames(frames, sender);

} catch (NumberFormatException e) {

System.out.println("Invalid input. Please enter integers only.");

}

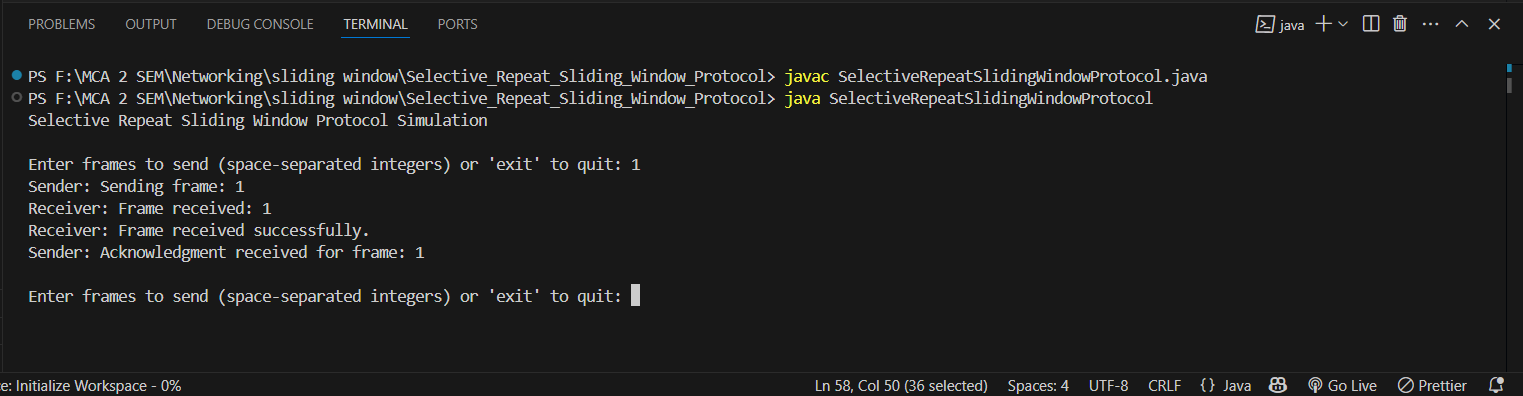
}

scanner.close();

}

}

**Output :**

****

**Implement Multicast Routing of Network Layer.**

**Sender :**

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.util.Scanner;

public class MulticastSender {

public static void main(String[] args) {

try {

DatagramSocket socket = new DatagramSocket();

InetAddress group = InetAddress.getByName("230.0.0.0"); // Multicast IP address

int port = 5000;

Scanner scanner = new Scanner(System.in);

System.out.println("Enter messages to send (type 'exit' to quit):");

while (true) {

String message = scanner.nextLine();

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

break;

}

byte[] buffer = message.getBytes();

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

socket.send(packet);

System.out.println("Message sent: " + message);

}

socket.close();

scanner.close();

} catch (Exception e) {

e.printStackTrace(); // <-- This line was incomplete before, now fixed

}

}

}

**Sender :**

import java.net.\*;

public class MulticastReceiver {

public static void main(String[] args) {

try {

int port = 5000;

InetAddress group = InetAddress.getByName("230.0.0.0"); // Multicast IP address

NetworkInterface networkInterface = NetworkInterface.getByName("wlan0"); // Change for your interface (e.g., eth0)

MulticastSocket socket = new MulticastSocket(port);

socket.setReuseAddress(true);

socket.joinGroup(new InetSocketAddress(group, port), networkInterface);

byte[] buffer = new byte[256];

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

System.out.println("Waiting for multicast messages...");

while (true) {

socket.receive(packet);

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

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

}

} catch (Exception e) {

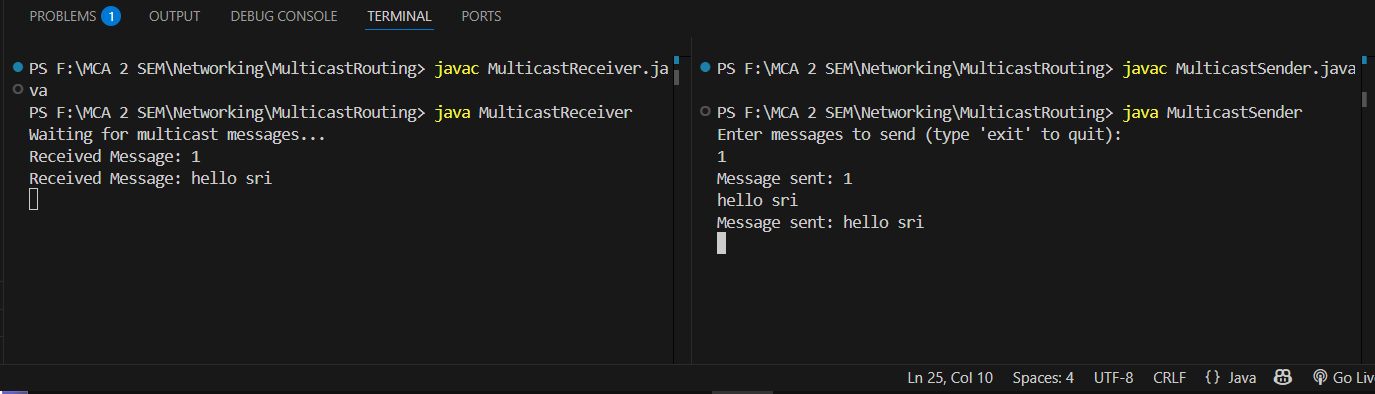
e.printStackTrace();

}

}

}

**Output :**

****

**Implement Transfer a File from one system to another system using FTP.**

**FTPClient :**

import java.io.\*;

import java.net.\*;

public class FTPClient {

public static void main(String[] args) {

String serverAddress = "localhost"; // Server address

int port = 12345; // Server port

try (Socket socket = new Socket(serverAddress, port);

DataInputStream dis = new DataInputStream(socket.getInputStream());

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

BufferedReader br = new BufferedReader(new InputStreamReader(System.in))) {

System.out.print("Enter the file name to download: ");

String fileName = br.readLine(); // Read file name from user

dos.writeUTF(fileName); // Send file name to the server

boolean fileExists = dis.readBoolean(); // Check if file exists

if (fileExists) {

long fileLength = dis.readLong(); // Get file length

try (FileOutputStream fos = new FileOutputStream("downloaded\_" + fileName)) {

byte[] buffer = new byte[4096];

int bytesRead;

long totalBytesRead = 0;

while (totalBytesRead < fileLength && (bytesRead = dis.read(buffer)) != -1) {

fos.write(buffer, 0, bytesRead); // Write to file

totalBytesRead += bytesRead;

}

System.out.println("File downloaded: " + fileName);

}

} else {

System.out.println("File not found on the server.");

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**FTPServer :**

import java.io.\*;

import java.net.\*;

public class FTPServer {

public static void main(String[] args) {

int port = 12345; // Server port

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

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

while (true) {

try (Socket clientSocket = serverSocket.accept()) {

System.out.println("Client connected: " + clientSocket.getInetAddress());

handleClient(clientSocket);

}

}

} catch (IOException e) {

e.printStackTrace();

}

}

private static void handleClient(Socket clientSocket) {

try (DataInputStream dis = new DataInputStream(clientSocket.getInputStream());

DataOutputStream dos = new DataOutputStream(clientSocket.getOutputStream())) {

String fileName = dis.readUTF(); // Get the requested file name

File file = new File(fileName);

if (file.exists() && !file.isDirectory()) {

dos.writeBoolean(true); // File exists

dos.writeLong(file.length()); // Send file length

try (FileInputStream fis = new FileInputStream(file)) {

byte[] buffer = new byte[4096];

int bytesRead;

while ((bytesRead = fis.read(buffer)) != -1) {

dos.write(buffer, 0, bytesRead); // Send file data

}

System.out.println("File sent: " + fileName);

}

} else {

dos.writeBoolean(false); // File does not exist

System.out.println("File not found: " + fileName);

}

} catch (IOException e) {

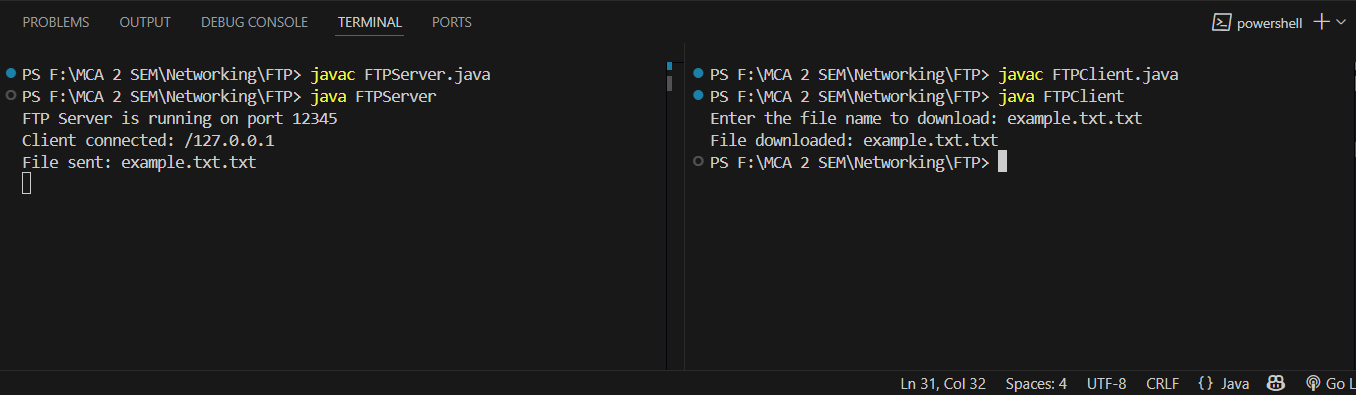
e.printStackTrace();

}

}

}

**Output :**

****