Ex. No : 2a	**Implementation of the Stop-and-Wait Protocol**
Date :	

Aim :-

To implement and execute stop and wait protocol using java.

Algorithm :-

Sender :

Step 1 : Start the program.

Step 2 : Set Sequence = 0

Step 3 : Accept new packet and assign a sequence to it.

Step 4 : Send packet sequence with sequence number.

Step 5 : Set timer for recently sent packets.

Step 6 : If error free acknowledgment from receiver and Next Frame Expected -> sequence then sequence -> Next Frame Expected.

Step 7 : If time out then go to step 3. Step 8 : Stop the program.

Receiver :

Step 1 : Start the program.

Step 2 : Next frame expected = 0, repeat step 3 forever.

Step 3 : If error free frame received and sequence -> Next Frame Expected, the pass packet to higher layer and Next Frame Expected -> Next Frame Expected +1 (modulo 2).

Step 4 : Stop the program.

Program :-

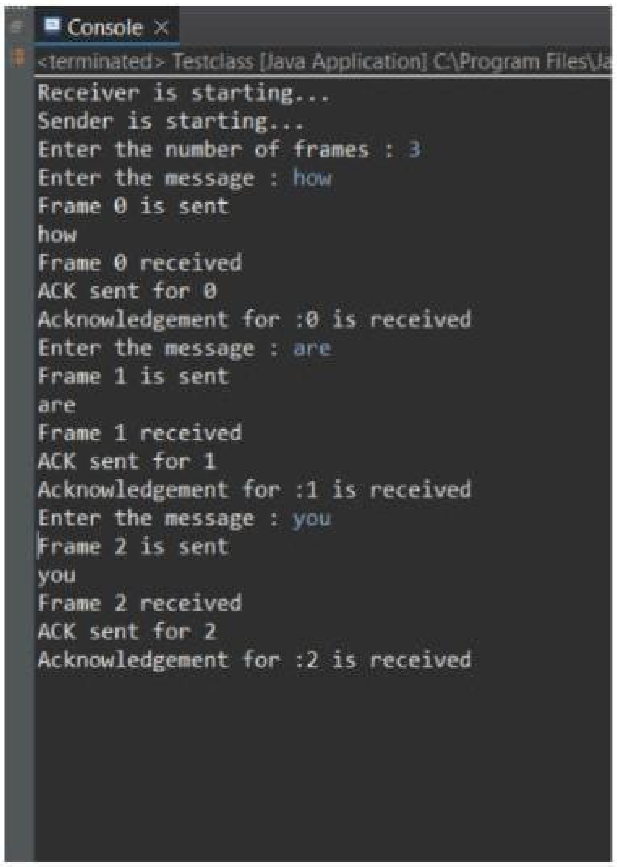
Sender.java import

```
java.io.*; import
java.net.*;
public class Sender {
public static void main(String[] args)
{ String serverAddress = "localhost";
int serverPort = 9876;
int timeout = 5000; // Increased timeout
try (DatagramSocket socket = new DatagramSocket()) {
InetAddress serverInetAddress = InetAddress.getByName(serverAddress);
String[] messages = {"Message 1", "Message 2", "Message 3"};
for (String message : messages) { byte[]
sendData = message.getBytes();
DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
serverInetAddress, serverPort);
socket.send(sendPacket);
System.out.println("Sent: " + message);
socket.setSoTimeout(timeout);
boolean ackReceived = false;
long startTime = System.currentTimeMillis();
while (System.currentTimeMillis() - startTime < timeout) { try
{
byte[] receiveData = new byte[1024];
DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
socket.receive(receivePacket);
String ackMessage = new String(receivePacket.getData(), 0,
receivePacket.getLength());
if (ackMessage.equals("ACK"))
{ ackReceived = true;
System.out.println("Received ACK for: " + message);
break;
}
} catch (SocketTimeoutException e) { System.out.println("Timeout
expired, resending: " + message); socket.send(sendPacket); // Resend
if no ACK is received
}
}
if (!ackReceived) {
System.out.println("No ACK received after timeout, retrying...");
socket.send(sendPacket);
}
}
} catch (Exception e)
{ e.printStackTrace();
}
}
}
```

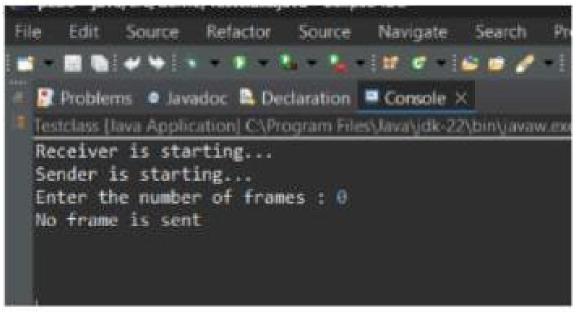
Receiver.java import

```
java.io.*; import
java.net.*;
public class Receiver {
public static void main(String[] args) { int
serverPort = 9876;
try (DatagramSocket socket = new DatagramSocket(serverPort))
{ byte[] receiveData = new byte[1024];
while (true) {
DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
socket.receive(receivePacket);
String receivedMessage = new String(receivePacket.getData(), 0,
receivePacket.getLength());
System.out.println("Received: " + receivedMessage);
Thread.sleep(500);
InetAddress senderAddress = receivePacket.getAddress(); int
senderPort = receivePacket.getPort();
String ackMessage = "ACK";
byte[] sendData = ackMessage.getBytes();
DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
senderAddress, senderPort);
socket.send(sendPacket);
System.out.println("Sent ACK for: " + receivedMessage);
}
} catch (Exception e)
{ e.printStackTrace();
}
}
}
```

Output :-



```
<terminated> Testclass [Java Application] C:\Program Files\Ja
Receiver is starting...
Sender is starting...
Enter the number of frames : 3
Enter the message : how
Frame 0 is sent
how
Frame 0 received
ACK sent for 0
Acknowledgement for :0 is received
Enter the message : are
Frame 1 is sent
are
Frame 1 received
ACK sent for 1
Acknowledgement for :1 is received
Enter the message : you
Frame 2 is sent
you
Frame 2 received
ACK sent for 2
Acknowledgement for :2 is received
```



```
Testclass [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe
Receiver is starting...
Sender is starting...
Enter the number of frames : 0
No frame is sent
```

Result :-

Thus, the program for stop and wait protocol was implemented and executed successfully.

Ex. No : 2b	**Implementation of the Sliding Window Protocol**
Date :	

Aim :-

To implement and execute sliding window protocol using java.

Algorithm :-

Sender :

Step 1 : Start the program.

Step 2 : Implement .net and other necessary packages.

Step 3 : Declare objects for input/output and socket to receive the frame and send acknowledgment.

Step 4 : Wait for the client to establish connection.

Step 5 : Receive the frame one by one from the socket and return the frame number as acknowledgment within the thread sleep time.

Step 6 : Send acknowledgment for each receiving frame and continue the process until acknowledgment for all frames are sent.

Step 7 : Receive the acknowledgment of last frame. Step
8 : Stop the program.

Receiver :

Step 1 : Start the program.

Step 2 : Import .net and other necessary packages.

Step 3 : Create objects for server socket, socket to send the frames and display the server address when the server is connected.

Step 4 : Get the number of frames to be sent, from the user.

Step 5 : Send the first frame to server using the socket.

Step 6 : When one frame is sent, wait for the acknowledgment from the receiver for the previous frame.

Programs :-

Sender.java

```
import java.util.LinkedList;
import java.util.Queue; public
class Sender { private int
windowSize;
private Queue<Integer> window; private
int totalPackets;
private int nextSeqNum;
public Sender(int windowSize, int totalPackets)
{ this.windowSize = windowSize;
this.totalPackets = totalPackets;
this.window = new LinkedList<>();
this.nextSeqNum = 0;
}
public void startTransmission(Receiver receiver) {
while (nextSeqNum < totalPackets || !window.isEmpty()) {
while (window.size() < windowSize && nextSeqNum < totalPackets)
{ System.out.println("Sender: Sending packet with sequence number " + nextSeqNum);
window.add(nextSeqNum);
receiver.receivePacket(nextSeqNum);
nextSeqNum++;
}
receiver.acknowledge(window);
window.clear();
}
}
}
```

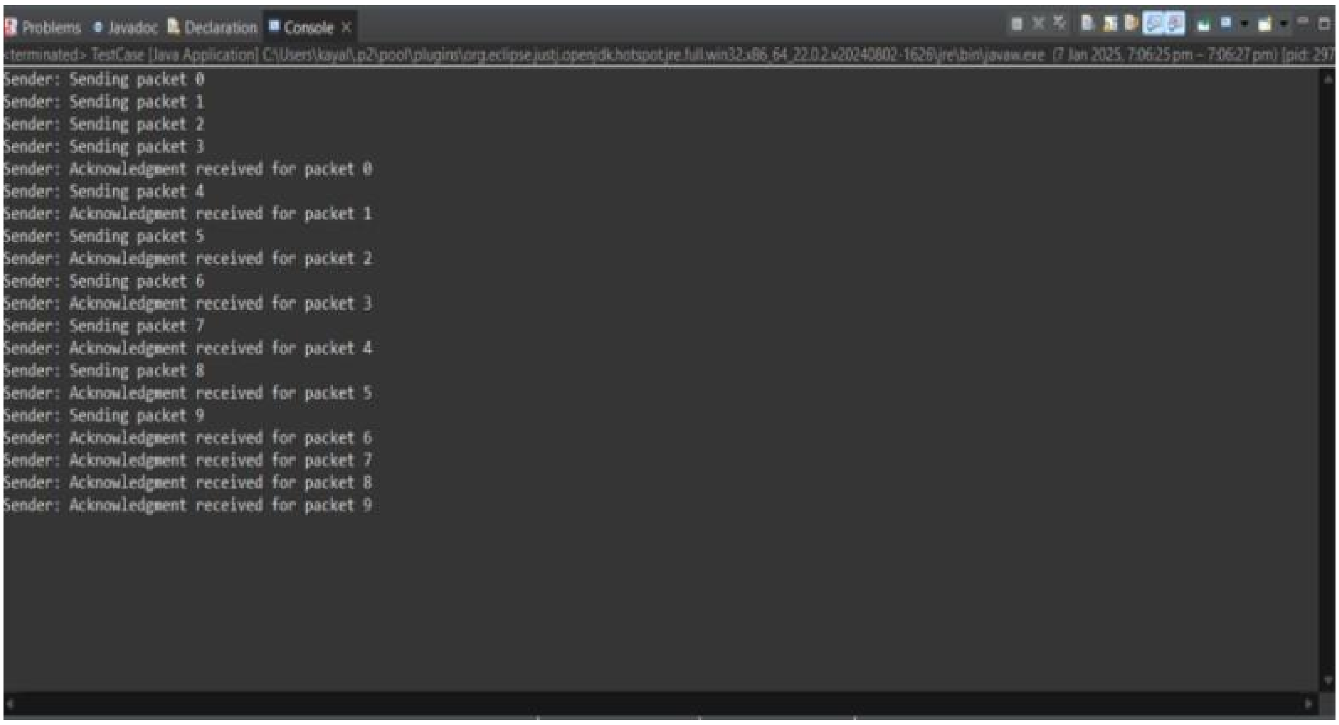
Receiver1.java

```
import java.util.Queue;
public class Receiver {
public void receivePacket(int packet) {
System.out.println("Receiver: Received packet with sequence number " + packet);
}
public void acknowledge(Queue<Integer> window) { for
(Integer seqNum : window) {
System.out.println("Receiver: Acknowledging packet with sequence number " + seqNum);
}
}
}
```

Main.java

```
import java.util.Scanner;
public class Main {
public static void main(String[] args)
{ Scanner scanner = new Scanner(System.in);
System.out.print("Enter the window size: ");
int windowSize = scanner.nextInt();
  System.out.print("Enter the total number of packets: ");
  int totalPackets = scanner.nextInt();
Sender sender = new Sender1(windowSize, totalPackets);
Receiver receiver = new Receiver1();
sender.startTransmission(receiver);
scanner.close();
}
}
```

Output :



```
<terminated> TestCase [Java Application] C:\Users\kayal\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_22.0.2.v20240802-1626\jre\bin\javaw.exe (7 Jan 2025, 7:06:25 pm – 7:06:27 pm) [pid: 297
Sender: Sending packet 0
Sender: Sending packet 1
Sender: Sending packet 2
Sender: Sending packet 3
Sender: Acknowledgment received for packet 0
Sender: Sending packet 4
Sender: Acknowledgment received for packet 1
Sender: Sending packet 5
Sender: Acknowledgment received for packet 2
Sender: Sending packet 6
Sender: Acknowledgment received for packet 3
Sender: Sending packet 7
Sender: Acknowledgment received for packet 4
Sender: Sending packet 8
Sender: Acknowledgment received for packet 5
Sender: Sending packet 9
Sender: Acknowledgment received for packet 6
Sender: Acknowledgment received for packet 7
Sender: Acknowledgment received for packet 8
Sender: Acknowledgment received for packet 9
```

Result :-

Thus, the program for sliding window protocol was implemented and executed successfully.

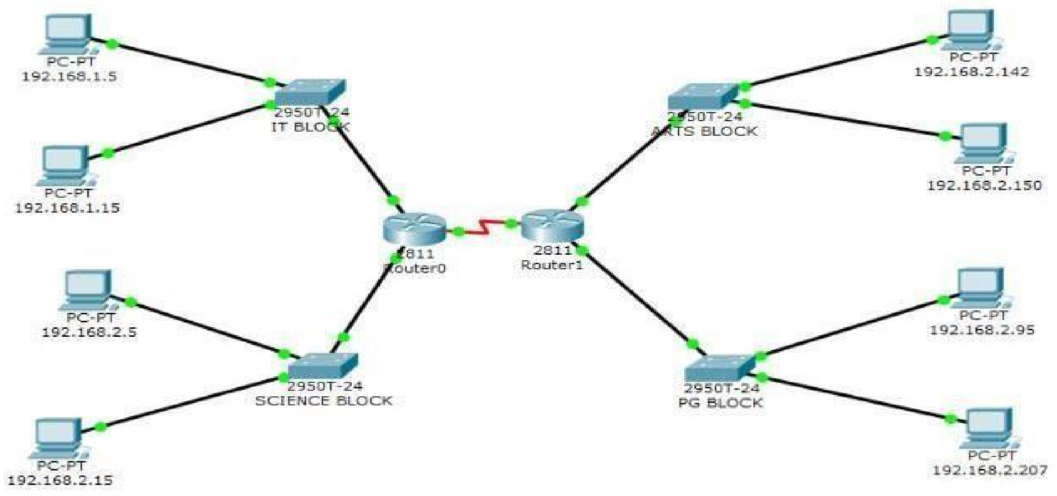
Ex. No : 3	IMPLEMENT SUBNETTING TECHNIQUES TO OPTIMIZE NETWORK ADDRESSING USING CISCO PACKET TRACER
Date :	

Aim :-

To implement subnetting and find the subnet masks using Cisco Packet Tracer.

Implementing an IP Addressing and Subnetting Topology

Diagram :-



OBJECTIVES :-

- Subnet an address space based on the host requirements.
- Assign host addresses to devices.
- Configure devices with IP addressing.
- Verify the addressing configuration.

Background / Preparation :-

In this activity, you will subnet the private address space 192.168.1.0/24 to provide enough host addresses for the two LANs attached to the router. You will then assign valid host addresses to the appropriate devices and interfaces. Finally, you will test connectivity to verify your IP address implementation.

Step 1: Subnet an address space based on the host requirements.

You are given the private address space 192.168.1.0/24. Subnet this address space based on the following requirements

IT BLOCK – 136 host

Science Block – 95 host Arts Block:

51 host

PG Block: 24 host

Step 2: Assign host addresses to devices.
What is the subnet address for subnet 0? 255.255.255.0 What is the subnet address of it ? **55.255.255.126** .

What is the subnet address for subnet 2? **255.255.255.192**

What is the subnet address for subnet 2? **255.255.255.224**

Assign subnet 0 to IT Block, assign subnet 1 to Science Block assign subnet 2 to arts Block and assign Subnet 3 to PG block.
This address is assigned the FastEthernet0/0 interface on Customer Router. What is the first address in subnet1? **192.168.2.1**
This address is assigned the FastEthernet0/0 interface on Customer Router. What is the first address in subnet2? **192.168.2.129**
This address is assigned the FastEthernet0/1 interface on Customer What is the first address in subnet3? **192.168.2.193**

What is the first address in subnet0? **192.168.1.1**
This address is assigned the FastEthernet0/2 interface on Customer

Assign address for serial connection for Router 0 to Router 1

Serial 0/0 for Router 0? **192.168.2.225**

Serial 0/0 for Router 1? **192.168.2.226**

Step 3: Configure devices with IP addressing.

Configure the LAN interfaces on Customer Router with IP addresses and a subnet mask.

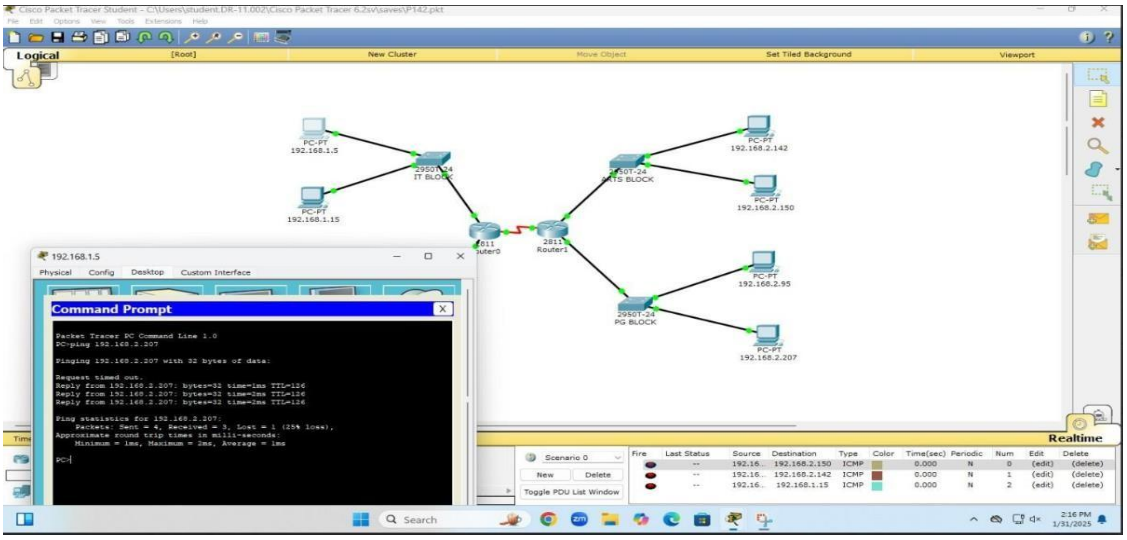
For a better view of the commands, you can increase the size of the window. To resize the window, place your mouse pointer over the bottom border of the window. When the cursor turns into a double headed arrow, click and drag.

Step 4: Verify the addressing configuration.

a. TestconnectivitybetweenHost1, Host3, ISP Workstation, and ISP Server. You can usetheAddSimplePDUtooltocreatepingsbetweenthedevices.Youcanalsoclick Host 1 or Host 3, then the Desktop tab, and then Command Prompt. Use the ping command to test connectivity to other devices. To obtain the IP address of another device, place your mouse pointer over the device.

b. Check results. On the Connectivity Tests tab, the status of each test should be successful.

OUTPUT :-



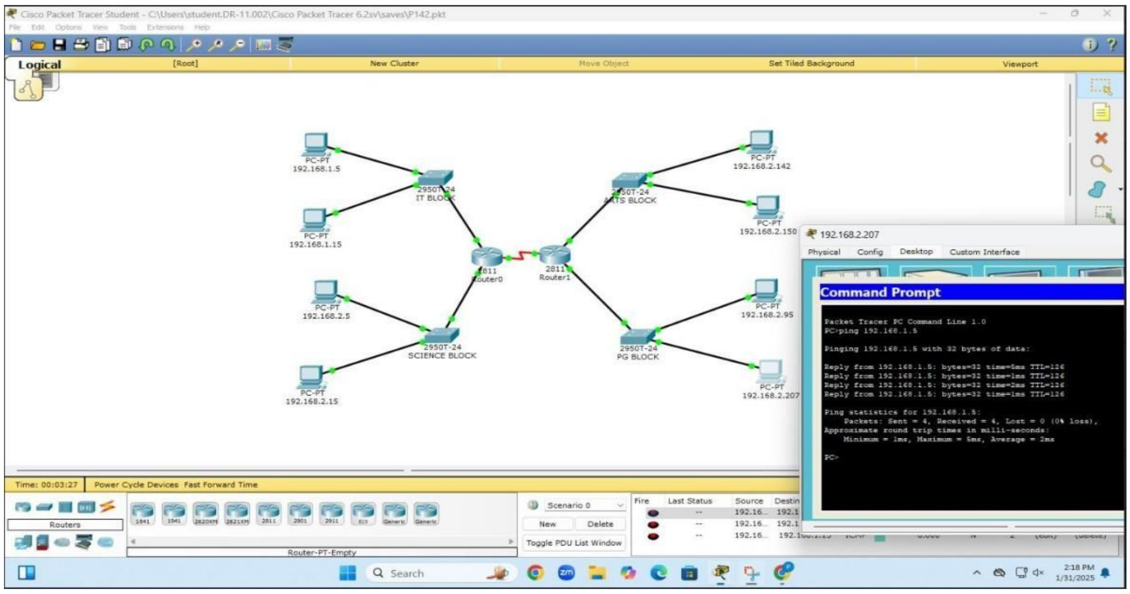
```
Packet Tracer PC Command Line 1.0
PC>ping 192.168.2.207

Pinging 192.168.2.207 with 32 bytes of data:

Request timed out.
Reply from 192.168.2.207: bytes=32 time=1ms TTL=126
Reply from 192.168.2.207: bytes=32 time=2ms TTL=126
Reply from 192.168.2.207: bytes=32 time=2ms TTL=126

Ping statistics for 192.168.2.207:
    Packets: Sent = 4, Received = 3, Lost = 1 (25% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 2ms, Average = 1ms

PC>
```



```
Packet Tracer PC Command Line 1.0
PC>ping 192.168.1.5

Pinging 192.168.1.5 with 32 bytes of data:

Reply from 192.168.1.5: bytes=32 time=5ms TTL=126
Reply from 192.168.1.5: bytes=32 time=1ms TTL=126
Reply from 192.168.1.5: bytes=32 time=2ms TTL=126
Reply from 192.168.1.5: bytes=32 time=1ms TTL=126

Ping statistics for 192.168.1.5:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 5ms, Average = 2ms

PC>
```

Result :-

Thus the concept of implementing of subnetting has been executed successfully and its output verified.