CN-PRACTICAL

🧠 1️⃣ Simple HTTP Client using TCP

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

class HttpClient {
    public static void main(String[] args) throws Exception {
        Socket s = new Socket("example.com", 80);
        PrintWriter out = new PrintWriter(s.getOutputStream());
        out.println("GET / HTTP/1.0\r\n\r\n");
        out.flush();
        BufferedReader in = new BufferedReader(new InputStreamReader(s.g
etInputStream()));
        String line;
        while ((line = in.readLine()) != null)
            System.out.println(line);
        s.close();
    }
}
```

🧾 Output

```
HTTP/1.0 200 OK
Content-Type: text/html
...
<html>
<head><title>Example Domain</title></head>
<body>Example Domain...</body>
</html>
```

💬 2️⃣ TCP Echo Client & Server

Server.java

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

class EchoServer {
   public static void main(String[] args) throws Exception {
      ServerSocket ss = new ServerSocket(5000);
      Socket s = ss.accept();
      BufferedReader in = new BufferedReader(new InputStreamReader(s.g
etInputStream()));
      PrintWriter out = new PrintWriter(s.getOutputStream(), true);
      String msg;
      while ((msg = in.readLine()) != null)
         out.println("Echo: " + msg);
      ss.close();
   }
}
```

Client.java

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

class EchoClient {
   public static void main(String[] args) throws Exception {
      Socket s = new Socket("localhost", 5000);
      BufferedReader in = new BufferedReader(new InputStreamReader(Sys
tem.in));
      PrintWriter out = new PrintWriter(s.getOutputStream(), true);
      BufferedReader serverIn = new BufferedReader(new InputStreamRead
er(s.getInputStream()));
      String msg;
      while (!(msg = in.readLine()).equals("bye")) {
         out.println(msg);
         System.out.println(serverIn.readLine());
      }
      s.close();
```

```
    }
}
```

🧾 Output

Client side:

```
Hi
Echo: Hi
Hello
Echo: Hello
bye
```

Server side:

```
Client connected...
(Repeats whatever client sends)
```

🗨️ 3️⃣ Interprocess Chat (TCP)

Server.java

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

class ChatServer {
    public static void main(String[] args) throws Exception {
        ServerSocket ss = new ServerSocket(5001);
        Socket s = ss.accept();
        BufferedReader in = new BufferedReader(new InputStreamReader(s.g
etInputStream()));
        PrintWriter out = new PrintWriter(s.getOutputStream(), true);
        BufferedReader console = new BufferedReader(new InputStreamRead
er(System.in));
        String msg;
        while (true) {
            msg = in.readLine();
            if (msg.equalsIgnoreCase("bye")) break;
```

```
            System.out.println("Client: " + msg);
            System.out.print("You: ");
            out.println(console.readLine());
        }
        s.close(); ss.close();
    }
}
```

Client.java

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

class ChatClient {
    public static void main(String[] args) throws Exception {
        Socket s = new Socket("localhost", 5001);
        BufferedReader in = new BufferedReader(new InputStreamReader(s.g
etInputStream()));
        PrintWriter out = new PrintWriter(s.getOutputStream(), true);
        BufferedReader console = new BufferedReader(new InputStreamRead
er(System.in));
        String msg;
        while (true) {
            System.out.print("You: ");
            out.println(console.readLine());
            msg = in.readLine();
            if (msg.equalsIgnoreCase("bye")) break;
            System.out.println("Server: " + msg);
        }
        s.close();
    }
}
```

 Output

```
Client: hi
Server: hey!
Client: how are you?
```

```
Server: good bro!
Client: bye
```

📁 4️⃣ File Server using TCP

Server.java

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

class FileServer {
    public static void main(String[] args) throws Exception {
        ServerSocket ss = new ServerSocket(5002);
        Socket s = ss.accept();
        BufferedReader in = new BufferedReader(new InputStreamReader(s.g
etInputStream()));
        PrintWriter out = new PrintWriter(s.getOutputStream(), true);
        File f = new File(in.readLine());
        if (f.exists()) {
            BufferedReader fr = new BufferedReader(new FileReader(f));
            String line; while ((line = fr.readLine()) != null) out.println(line);
            fr.close();
        } else out.println("File not found");
        s.close(); ss.close();
    }
}
```

Client.java

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

class FileClient {
    public static void main(String[] args) throws Exception {
        Socket s = new Socket("localhost", 5002);
        PrintWriter out = new PrintWriter(s.getOutputStream(), true);
        BufferedReader in = new BufferedReader(new InputStreamReader(s.g
etInputStream()));
```

```
        out.println("data.txt");
        String line;
        while ((line = in.readLine()) != null)
            System.out.println(line);
        s.close();
    }
}
```

🧾 Output

```
Contents of data.txt:
Hello World
This is a sample file.
```

🌐 5️⃣ DNS Lookup using UDP

```
import java.net.*;

class DNS {
    public static void main(String[] args) throws Exception {
        InetAddress ip = InetAddress.getByName("www.google.com");
        System.out.println("IP Address: " + ip.getHostAddress());
    }
}
```

🧾 Output

```
IP Address: 142.250.182.68
```

⚙️ 6️⃣ ARP/RARP Simulation

```
import java.util.*;

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

```
        Map<String, String> arp = Map.of(
            "192.168.1.1", "00:0a:95:9d:68:16",
            "192.168.1.2", "00:0a:95:9d:68:17"
        );
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter IP: ");
        String ip = sc.next();
        System.out.println("MAC Address: " + arp.getOrDefault(ip, "Not foun
d"));
    }
}
```

🧾 Output

```
Enter IP: 192.168.1.2
MAC Address: 00:0a:95:9d:68:17
```

🛰️ 7️⃣ Distance Vector Routing

```
import java.util.*;

class DVR {
    public static void main(String[] args) {
        int[][] cost = { {0, 2, 7}, {2, 0, 1}, {7, 1, 0} };
        int n = cost.length;
        for (int i = 0; i < n; i++) {
            System.out.println("Router " + i + ":");
            for (int j = 0; j < n; j++)
                System.out.println(" → " + j + " = " + cost[i][j]);
        }
    }
}
```

🧾 Output

```
Router 0:
 → 0 = 0
 → 1 = 2
 → 2 = 7
Router 1:
 → 0 = 2
 → 1 = 0
 → 2 = 1
Router 2:
 → 0 = 7
 → 1 = 1
 → 2 = 0
```

🌍 8️⃣ Link State Routing

```
import java.util.*;

class LSR {
    public static void main(String[] args) {
        int[][] g = {{0,4,0,0,8},{4,0,8,0,11},{0,8,0,7,0},{0,0,7,0,9},{8,11,0,9,0}};
        int n = g.length, src = 0;
        int[] dist = new int[n]; boolean[] vis = new boolean[n];
        Arrays.fill(dist, 9999); dist[src] = 0;
        for (int i=0; i<n-1; i++) {
            int u=-1, min=9999;
            for (int j=0; j<n; j++) if(!vis[j] && dist[j]<min){min=dist[j];u=j;}
            vis[u] = true;
            for (int v=0; v<n; v++)
                if(g[u][v]!=0 && !vis[v] && dist[u]+g[u][v]<dist[v])
                    dist[v] = dist[u]+g[u][v];
        }
        for (int i=0; i<n; i++) System.out.println("Node " + i + " Distance: " + di
st[i]);
    }
}
```

🧾 Output

```
Node 0 Distance: 0
Node 1 Distance: 4
Node 2 Distance: 12
Node 3 Distance: 19
Node 4 Distance: 8
```

🔍 🔟 Network Command Simulation

1. `tcpdump`

- **Use:** `tcpdump` is a powerful **command-line packet analyzer**. It intercepts and displays TCP/IP and other packets being transmitted or received over a network. It is primarily used for network troubleshooting, security analysis, and developing applications.
- **PDU Relation & Simulation:** `tcpdump` **directly captures and displays PDUs (packets/frames)** as they traverse the network interface.
    - **Simulation Output:** Would show the raw header and data of **L2 Frames (Ethernet/Wi-Fi)**, **L3 Packets (IP)**, and **L4 Segments (TCP/UDP)**. For example, it would show the **source/destination MAC addresses (Frame PDU)**, **source/destination IP addresses (Packet PDU)**, and **source/destination ports (Segment PDU)**.

2. `netstat`

- **Use:** `netstat` (**network statistics**) is a utility that displays **active network connections** (both incoming and outgoing), routing tables, interface statistics, and multicast group memberships. It's useful for seeing which ports are open and which applications are using them.
- **PDU Relation & Simulation:** `netstat` reports on the **state** of **Layer 4 (Transport layer) connections (TCP Segments and UDP Datagrams)**.
    - **Simulation Output:** Would show the **state** of a **TCP connection** (e.g., `ESTABLISHED`, `LISTEN`, `TIME_WAIT`) between two **L4 Endpoints** defined by **IP address and port number** (the identifying fields in the **IP Packet PDU** and **TCP Segment PDU**).

3. `ifconfig` (or `ip addr` in newer Linux)

- **Use:** `ifconfig` (**interface configuration**) is used to configure, view, or manage network interfaces. It displays information like the **IP address, subnet mask, MAC address, and status** of each network interface.
- **PDU Relation & Simulation:** `ifconfig` displays the **configuration data** necessary for forming **Layer 2 (Data Link) Frames** and **Layer 3 (Network) Packets**.
    - **Simulation Output:** Would display the **MAC address (used in L2 Frame PDU header)** and the **IP address/Subnet Mask (used in L3 Packet PDU header)**.

4. `nslookup`

- **Use:** `nslookup` (**name server lookup**) is a utility used to query **Domain Name System (DNS)** servers to obtain domain name or IP address mapping or other DNS records. It helps ensure name resolution is working correctly.
- **PDU Relation & Simulation:** `nslookup` initiates a **DNS Query**, which is typically encapsulated in a **UDP Datagram PDU (L4)**, which is then inside an **IP Packet PDU (L3)**.
    - **Simulation Output:** Would show the **DNS Query** traveling to the DNS server and the **DNS Reply** returning. The associated PDUs would be **UDP Datagrams** containing the DNS data.

5. `traceroute` (or `tracert` on Windows)

- **Use:** `traceroute` determines the **path (route) and measures transit times of packets** across an IP network to a specified destination. It reveals the sequence of routers (hops) a packet takes.
- **PDU Relation & Simulation:** `traceroute` relies on sending **ICMP (Internet Control Message Protocol) Packets (L3 PDU)** or sometimes **UDP Datagrams (L4 PDU)** and using the **"Time-to-Live (TTL) expired"** message from routers to map the path.
    - **Simulation Output:** Would show a series of **ICMP Packets (L3 PDU)** being sent with sequentially increasing TTL values, and the **ICMP "Time Exceeded" reply (L3 PDU)** coming back from each router (hop).
