Experiment 1: Working with InetAddress, Inet4Address, Inet6Address

Aim:

Write a program to display IP address and host name using `InetAddress`.

Algorithm:

1. Import `java.net.*`
2. Use `InetAddress.getLocalHost()` to get local IP and hostname.
3. Use `InetAddress.getByName()` to get IP of a website.
4. Display results.

Code:

```
import java.net.*;

public class InetAddressDemo {
    public static void main(String[] args) throws Exception {
        InetAddress local = InetAddress.getLocalHost();
        System.out.println("Local Host: " + local.getHostName());
        System.out.println("Local IP: " + local.getHostAddress());

        InetAddress google = InetAddress.getByName("www.google.com");
        System.out.println("Google Hostname: " + google.getHostName());
        System.out.println("Google IP: " + google.getHostAddress());
    }
}
```

Experiment 2: TCP/IP Client and Server (Sockets)

Aim:

Implement a TCP Client and Server to exchange messages.

Server Code (`TCPServer.java`):

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

public class TCPServer {
    public static void main(String[] args) throws Exception {
        ServerSocket ss = new ServerSocket(5000);
        System.out.println("Server waiting...");
        Socket s = ss.accept();
        BufferedReader br = new BufferedReader(new
InputStreamReader(s.getInputStream()));
        PrintWriter pw = new PrintWriter(s.getOutputStream(), true);

        String msg = br.readLine();
        System.out.println("Client says: " + msg);
        pw.println("Hello from Server!");
        s.close();
        ss.close();
    }
}
```

Client Code (`TCPClient.java`):

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

public class TCPClient {
    public static void main(String[] args) throws Exception {
        Socket s = new Socket("localhost", 5000);
        BufferedReader br = new BufferedReader(new 
InputStreamReader(System.in));
        PrintWriter pw = new PrintWriter(s.getOutputStream(), true);

        System.out.print("Enter message for server: ");
        String msg = br.readLine();
        pw.println(msg);

        BufferedReader serverMsg = new BufferedReader(new 
InputStreamReader(s.getInputStream()));
        System.out.println("Server says: " + serverMsg.readLine());
        s.close();
    }
}
```

Experiment 3: Working with URL and URLConnection

Aim:

Fetch data from a website using `URL` and `URLConnection`.

Code:

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

public class URLDemo {
    public static void main(String[] args) throws Exception {
        URL url = new URL("https://www.example.com");
        URLConnection conn = url.openConnection();

        BufferedReader br = new BufferedReader(new 
InputStreamReader(conn.getInputStream()));
        String line;
        while ((line = br.readLine()) != null) {
            System.out.println(line);
        }
        br.close();
    }
}
```

Experiment 4: HttpURLConnection Example

Aim:

Send an HTTP GET request and display the response code and message.

Code:

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

public class HttpURLConnectionDemo {
    public static void main(String[] args) throws Exception {
        URL url = new URL("https://www.google.com");
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");

        System.out.println("Response Code: " + conn.getResponseCode());
        System.out.println("Response Message: " +
conn.getResponseMessage());
        conn.disconnect();
    }
}
```

Experiment 5: Using the URI Class

Aim:

Parse and display parts of a URI.

Code:

```
import java.net.*;

public class URIDemo {
    public static void main(String[] args) throws Exception {
        URI uri = new
URI("https://www.example.com:8080/test/path?query=123#section");

        System.out.println("Scheme: " + uri.getScheme());
        System.out.println("Host: " + uri.getHost());
        System.out.println("Port: " + uri.getPort());
        System.out.println("Path: " + uri.getPath());
        System.out.println("Query: " + uri.getQuery());
        System.out.println("Fragment: " + uri.getFragment());
    }
}
```

Experiment 6: UDP Communication (DatagramSocket & DatagramPacket)

Aim:

Implement a UDP Server and Client to send and receive messages.

UDP Server (`UDPServer.java`):

```
import java.net.*;

public class UDPServer {
    public static void main(String[] args) throws Exception {
        DatagramSocket ds = new DatagramSocket(6000);
        byte[] buffer = new byte[1024];
        DatagramPacket dp = new DatagramPacket(buffer, buffer.length);
        System.out.println("UDP Server waiting...");

        ds.receive(dp);
        String msg = new String(dp.getData(), 0, dp.getLength());
        System.out.println("Client says: " + msg);
        ds.close();
    }
}
```

UDP Client (`UDPClient.java`):

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

public class UDPClient {
    public static void main(String[] args) throws Exception {
        DatagramSocket ds = new DatagramSocket();
        BufferedReader br = new BufferedReader(new 
InputStreamReader(System.in));

        System.out.print("Enter message for server: ");
        String msg = br.readLine();
        byte[] data = msg.getBytes();

        InetAddress ip = InetAddress.getByName("localhost");
        DatagramPacket dp = new DatagramPacket(data, data.length, ip, 
6000);
        ds.send(dp);
        ds.close();
    }
}
```

Experiment 7: Working with Cookies (HttpCookie)

Aim:

Demonstrate creating and reading HTTP Cookies.. cv `import java.net.HttpCookie;`

```
public class CookieDemo {
    public static void main(String[] args) {
        HttpCookie cookie = new HttpCookie("user", "John123");
        cookie.setDomain("example.com");
        cookie.setPath("/");

        System.out.println("Cookie Name: " + cookie.getName());
        System.out.println("Cookie Value: " + cookie.getValue());
        System.out.println("Domain: " + cookie.getDomain());
        System.out.println("Path: " + cookie.getPath());
    }
}
```

Code:

```
import java.net.HttpCookie;

public class CookieDemo {
    public static void main(String[] args) {
        HttpCookie cookie = new HttpCookie("user", "John123");
        cookie.setDomain("example.com");
        cookie.setPath("/");

        System.out.println("Cookie Name: " + cookie.getName());
        System.out.println("Cookie Value: " + cookie.getValue());
        System.out.println("Domain: " + cookie.getDomain());
        System.out.println("Path: " + cookie.getPath());
    }
}
```

Experiment 8: TCP File Transfer in Java (Server & Client)

Server Code (`FileServer.java`)

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

public class FileServer {
    public static void main(String[] args) {
        int port = 5001;
        try {
            ServerSocket serverSocket = new ServerSocket(port);
            System.out.println("Server started. Waiting for client...");

            Socket socket = serverSocket.accept();
            System.out.println("Client connected: " +
socket.getInetAddress());

            // File to send
            File file = new File("sample.txt");
            FileInputStream fis = new FileInputStream(file);
            BufferedInputStream bis = new BufferedInputStream(fis);

            OutputStream os = socket.getOutputStream();
            byte[] buffer = new byte[4096];
            int bytesRead;
            while ((bytesRead = bis.read(buffer)) != -1) {
                os.write(buffer, 0, bytesRead);
            }

            System.out.println("File sent successfully.");

            bis.close();
            socket.close();
            serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Client Code (`FileClient.java`)

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

public class FileClient {
    public static void main(String[] args) {
        String host = "127.0.0.1";
        int port = 5001;

        try {
            Socket socket = new Socket(host, port);
            System.out.println("Connected to server.");

            InputStream is = socket.getInputStream();
            FileOutputStream fos = new FileOutputStream("received.txt");
            BufferedOutputStream bos = new BufferedOutputStream(fos);

            byte[] buffer = new byte[4096];
            int bytesRead;

            while ((bytesRead = is.read(buffer)) != -1) {
                bos.write(buffer, 0, bytesRead);
            }

            System.out.println("File received and saved as received.txt");

            bos.close();
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Steps to Run in NetBeans:

1. **Create a new Java project** in NetBeans.
2. Inside `Source Packages`, create two separate classes:
    - `FileServer.java`
    - `FileClient.java`

3. Create a file named `sample.txt` in the project folder with some content.
4. **Run `FileServer` first.**
5. **Run `FileClient` next.**
6. After execution, check for `received.txt` in the project folder – it should match `sample.txt`.

TCP and UDP communication differ in how they transmit data over the internet. Here's a clear comparison:

TCP vs UDP Communication

Feature	TCP (Transmission Control Protocol)	UDP (User Datagram Protocol)
Connection Type	**Connection-oriented**	**Connectionless**
Reliability	Reliable – ensures data delivery, ordering	Unreliable – no guarantee of delivery
Speed	Slower (due to overhead of checking)	Faster (less overhead, no checks)
Error Checking	Yes – with acknowledgment and retransmission	Yes – but no retransmission
Ordering of Packets	Guarantees packet order	No guarantee of order
Data Flow Control	Yes	No
Use Cases	Web (HTTP/HTTPS), Email (SMTP), File Transfer (FTP)	Video streaming, VoIP, Online Gaming
Header Size	Larger (20–60 bytes)	Smaller (8 bytes)
Congestion Control	Yes	No
Example Ports	HTTP: 80, HTTPS: 443	DNS: 53, TFTP: 69, VoIP: 5060

In Simple Terms:

Scenario	Use
Need all data safely? (e.g., downloading a file)	Use **TCP**
Speed over reliability? (e.g., live video/gaming)	Use **UDP**

Summary

- **TCP** = Safe, Ordered, Reliable, but Slower
- **UDP** = Fast, Lightweight, but No Guarantees

Creating a **chat application** can be done in several ways using different protocols (TCP/UDP), languages (Python, Java), and technologies (WebSocket, Flutter, etc.).

create a **simple chat application in Java** using **TCP sockets**. It includes:

- One **Server** that handles multiple clients using **threads**
- Multiple **Clients** that can send and receive messages

Experiment 9: Java Chat Application using TCP

1. Server Code (`ChatServer.java`)

Handles multiple clients using threads.

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

public class ChatServer {
    private static final int PORT = 1234;
    private static Set<Socket> clientSockets = 
Collections.synchronizedSet(new HashSet<>());

    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket(PORT);
        System.out.println("Server started. Listening on port " + PORT);

        while (true) {
            Socket clientSocket = serverSocket.accept();
            clientSockets.add(clientSocket);
            System.out.println("New client connected: " + 
clientSocket.getInetAddress());
```

```
            new ClientHandler(clientSocket).start();
        }
    }

    static class ClientHandler extends Thread {
        private Socket socket;
        private BufferedReader in;
        private PrintWriter out;

        public ClientHandler(Socket socket) throws IOException {
            this.socket = socket;
            in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            out = new PrintWriter(socket.getOutputStream(), true);
        }

        public void run() {
            try {
                String message;
                while ((message = in.readLine()) != null) {
                    System.out.println("Received: " + message);
                    broadcast(message, socket);
                }
            } catch (IOException e) {
                System.out.println("Client disconnected.");
            } finally {
                try {
                    clientSockets.remove(socket);
                    socket.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

        private void broadcast(String message, Socket senderSocket) {
            synchronized (clientSockets) {
                for (Socket client : clientSockets) {
                    if (client != senderSocket) {
                        try {
                            PrintWriter clientOut = new
PrintWriter(client.getOutputStream(), true);
                            clientOut.println("Client says: " + message);
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                    }
                }
            }
        }
    }
}
```

2. Client Code (`ChatClient.java`)

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

public class ChatClient {
    private static final String SERVER_IP = "127.0.0.1";  // or your server
IP
    private static final int SERVER_PORT = 1234;

    public static void main(String[] args) throws IOException {
        Socket socket = new Socket(SERVER_IP, SERVER_PORT);
        System.out.println("Connected to the chat server.");

        BufferedReader input = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        PrintWriter output = new PrintWriter(socket.getOutputStream(),
true);
        BufferedReader console = new BufferedReader(new
InputStreamReader(System.in));

        // Thread to receive messages
        new Thread(() -> {
            try {
                String serverMsg;
                while ((serverMsg = input.readLine()) != null) {
                    System.out.println(serverMsg);
                }
            } catch (IOException e) {
                System.out.println("Disconnected from server.");
            }
        }).start();

        // Main thread to send messages
        String userInput;
        while ((userInput = console.readLine()) != null) {
            output.println(userInput);
        }

        socket.close();
    }
}
```

How to Run

1. Compile:

   ```
   javac ChatServer.java ChatClient.java
   ```

2. Run Server:

   ```
   java ChatServer
   ```

3. Run multiple Clients (in different terminals):

   ```
   java ChatClient
   ```

Features You Can Add Later

Feature	How
Nicknames	Ask username before chat
GUI	Use Swing or JavaFX
Private Chat	Commands like `/pm user message`
Disconnect Msg	Notify others when a user disconnects
Encryption	Use SSL sockets or RSA-based encryption

Experiment 10:

Implement a simple **TCP client-server program in Java** (using NetBeans) where:

- The **client sends**: `"Welcome to Gujarat Technological UNIVERSITY "`
- The **server responds** with the **reverse** of the string (case reversed too).

---

1. TCP Server (Java using NetBeans)

```
// File: TCPServer.java
import java.io.*;
import java.net.*;

public class TCPServer {
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(6789);
            System.out.println("Server started and waiting for client...");

            Socket connectionSocket = serverSocket.accept();
            System.out.println("Client connected.");

            BufferedReader inFromClient = new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream()));
            DataOutputStream outToClient = new
DataOutputStream(connectionSocket.getOutputStream());

            String clientSentence = inFromClient.readLine();
            System.out.println("Received from client: " + clientSentence);

            // Reverse string and reverse case
            String response = new StringBuilder(clientSentence)
                                .reverse()
                                .toString()
                                .chars()
                                .mapToObj(c -> Character.isUpperCase(c) ?
Character.toLowerCase((char) c) : Character.toUpperCase((char) c))
                                .collect(StringBuilder::new,
StringBuilder::append, StringBuilder::append)
                                .toString();

            outToClient.writeBytes(response + "\n");
            connectionSocket.close();
            serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

---

2. TCP Client (Java using NetBeans)

```
// File: TCPClient.java
import java.io.*;
import java.net.*;

public class TCPClient {
    public static void main(String[] args) {
        try {
            Socket clientSocket = new Socket("localhost", 6789);

            DataOutputStream outToServer = new
DataOutputStream(clientSocket.getOutputStream());
            BufferedReader inFromServer = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

            String sentence = "Welcome to Gujarat Technological
UNIVERSITY";
            System.out.println("Sending to server: " + sentence);
            outToServer.writeBytes(sentence + "\n");

            String response = inFromServer.readLine();
            System.out.println("Response from server: " + response);

            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

How to Run in NetBeans

1. Create a new **Java Project**.
2. Add two classes: `TCPServer` and `TCPClient`.
3. Run `TCPServer.java` first.
4. Then run `TCPClient.java`.

Expected Output

Client Output:

```
Sending to server: Welcome to Gujarat Technological UNIVERSITY
Response from server: ytisrevinu LACIGOLONHCEt TARAJUg TO EMOCLEw
```

Server Output:

```
Server started and waiting for client...
Client connected.
Received from client: Welcome to Gujarat Technological UNIVERSITY
```

Experiment 11: **TCP-based Java program** (compatible with NetBeans) where:

- The **client sends 10 numbers** (comma-separated).
- The **server receives, sorts** them, and **sends them back**.

---

1. TCP Server (Java)

```
// File: SortTCPServer.java
import java.io.*;
import java.net.*;
import java.util.*;

public class SortTCPServer {
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(6789);
            System.out.println("Server started. Waiting for client...");

            Socket connectionSocket = serverSocket.accept();
            System.out.println("Client connected.");

            BufferedReader inFromClient = new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream()));
            DataOutputStream outToClient = new
DataOutputStream(connectionSocket.getOutputStream());

            String numbers = inFromClient.readLine();
            System.out.println("Received numbers: " + numbers);

            // Split, parse and sort
            String[] parts = numbers.split(",");
            int[] nums = new int[parts.length];
            for (int i = 0; i < parts.length; i++) {
                nums[i] = Integer.parseInt(parts[i].trim());
            }
            Arrays.sort(nums);

            // Prepare sorted string
            StringBuilder sortedStr = new StringBuilder();
            for (int n : nums) {
                sortedStr.append(n).append(" ");
            }

            outToClient.writeBytes("Sorted: " + sortedStr.toString().trim()
+ "\n");

            connectionSocket.close();
            serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

---

2. TCP Client (Java)

```
// File: SortTCPClient.java
import java.io.*;
import java.net.*;
import java.util.Scanner;

public class SortTCPClient {
    public static void main(String[] args) {
        try {
            Socket clientSocket = new Socket("localhost", 6789);
            DataOutputStream outToServer = new
DataOutputStream(clientSocket.getOutputStream());
            BufferedReader inFromServer = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

            Scanner scanner = new Scanner(System.in);
            System.out.println("Enter 10 numbers separated by commas:");
            String numbers = scanner.nextLine();

            outToServer.writeBytes(numbers + "\n");

            String response = inFromServer.readLine();
            System.out.println("Server response: " + response);

            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

How to Run in NetBeans

1. Create a new Java Project.

2. Add both classes: `SortTCPServer` and `SortTCPClient`.
3. Run `SortTCPServer.java` first.
4. Then run `SortTCPClient.java` and input 10 numbers like:

   ```
   45, 12, 5, 99, 1, 7, 34, 56, 3, 18
   ```

Expected Output

Client:

```
Enter 10 numbers separated by commas:
45, 12, 5, 99, 1, 7, 34, 56, 3, 18
Server response: Sorted: 1 3 5 7 12 18 34 45 56 99
```

Server:

```
Server started. Waiting for client...
Client connected.
Received numbers: 45, 12, 5, 99, 1, 7, 34, 56, 3, 18
```

Experiment 12: **TCP Client-Server Java program** using NetBeans, where:

- The **client sends a request** to get the current **Date & Time**.
- The **server sends back** the current **system date and time**.

---

1. TCP Server: DateTimeTCPServer.java

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

public class DateTimeTCPServer {
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(6789);
            System.out.println("Server started. Waiting for client...");

            Socket connectionSocket = serverSocket.accept();
            System.out.println("Client connected.");

            BufferedReader inFromClient = new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream()));
            DataOutputStream outToClient = new
DataOutputStream(connectionSocket.getOutputStream());

            String request = inFromClient.readLine();
            System.out.println("Received from client: " + request);
```

```
            if (request.equalsIgnoreCase("GET_DATETIME")) {
                Date date = new Date();
                String dateTime = date.toString();
                outToClient.writeBytes("Server Date & Time: " + dateTime +
"\n");
                System.out.println("Date & Time sent to client.");
            } else {
                outToClient.writeBytes("Invalid Request\n");
            }

            connectionSocket.close();
            serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

---

2. TCP Client: DateTimeTCPClient.java

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

public class DateTimeTCPClient {
    public static void main(String[] args) {
        try {
            Socket clientSocket = new Socket("localhost", 6789);
            DataOutputStream outToServer = new
DataOutputStream(clientSocket.getOutputStream());
            BufferedReader inFromServer = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

            // Send request
            outToServer.writeBytes("GET_DATETIME\n");

            // Receive response
            String response = inFromServer.readLine();
            System.out.println("Response from server: " + response);

            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

How to Run in NetBeans

1. Open NetBeans and create a new **Java project**.
2. Add both classes (`DateTimeTCPServer` and `DateTimeTCPClient`) to the project.
3. Run `DateTimeTCPServer` first.
4. Then run `DateTimeTCPClient`.

Sample Output

Client Output:

```
Response from server: Server Date & Time: Thu Aug 08 17:45:12 IST 2025
```

Server Output:

```
Server started. Waiting for client...
Client connected.
Received from client: GET_DATETIME
Date & Time sent to client.
```

Experiment 13: Simple **TCP client-server program in Java** where:

- The **client sends two numbers**.
- The **server calculates the sum** and sends the result back.

---

1. TCP Server (SumTCPServer.java)

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

public class SumTCPServer {
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(6789);
            System.out.println("Server started. Waiting for client...");

            Socket connectionSocket = serverSocket.accept();
            System.out.println("Client connected.");

            BufferedReader inFromClient = new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream()));
            DataOutputStream outToClient = new
DataOutputStream(connectionSocket.getOutputStream());

            // Receive numbers
            String num1Str = inFromClient.readLine();
            String num2Str = inFromClient.readLine();
            int num1 = Integer.parseInt(num1Str.trim());
            int num2 = Integer.parseInt(num2Str.trim());

            int sum = num1 + num2;
            outToClient.writeBytes("Sum = " + sum + "\n");

            connectionSocket.close();
            serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

---

2. TCP Client (SumTCPClient.java)

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

public class SumTCPClient {
    public static void main(String[] args) {
        try {
            Socket clientSocket = new Socket("localhost", 6789);

            DataOutputStream outToServer = new
DataOutputStream(clientSocket.getOutputStream());
            BufferedReader inFromServer = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

            Scanner scanner = new Scanner(System.in);

            // Input two numbers
            System.out.print("Enter first number: ");
            String num1 = scanner.nextLine();
            System.out.print("Enter second number: ");
            String num2 = scanner.nextLine();

            // Send to server
            outToServer.writeBytes(num1 + "\n");
            outToServer.writeBytes(num2 + "\n");

            // Get response
            String response = inFromServer.readLine();
            System.out.println("Server says: " + response);

            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

How to Run in NetBeans

1. Create a new **Java project**.

2. Add two classes: `SumTCPServer` and `SumTCPClient`.
3. Run `SumTCPServer` first.
4. Then run `SumTCPClient`.

Sample Output

Client:

```
Enter first number: 25
Enter second number: 17
Server says: Sum = 42
```

Server:

```
Server started. Waiting for client...
Client connected.
```

Java TCP client-server program that:

- The **client sends a string**.
- The **server checks** if the string is a **palindrome**.
- The **server replies** with a message: `"Palindrome"` or `"Not a Palindrome"`.

---

1. TCP Server: PalindromeTCPServer.java

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

public class PalindromeTCPServer {
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(6789);
            System.out.println("Server started. Waiting for client...");

            Socket connectionSocket = serverSocket.accept();
            System.out.println("Client connected.");

            BufferedReader inFromClient = new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream()));
            DataOutputStream outToClient = new
DataOutputStream(connectionSocket.getOutputStream());

            String receivedString = inFromClient.readLine();
            System.out.println("Received from client: " + receivedString);
            // Normalize string (remove spaces and lowercase)
            String cleaned = receivedString.replaceAll("\\s+",
"").toLowerCase();
            String reversed = new
StringBuilder(cleaned).reverse().toString();

            String result = cleaned.equals(reversed) ? "Palindrome" : "Not
a Palindrome";

            outToClient.writeBytes(result + "\n");

            connectionSocket.close();
            serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

---

2. TCP Client: PalindromeTCPClient.java

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

public class PalindromeTCPClient {
    public static void main(String[] args) {
        try {
            Socket clientSocket = new Socket("localhost", 6789);
            DataOutputStream outToServer = new
DataOutputStream(clientSocket.getOutputStream());
            BufferedReader inFromServer = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

            Scanner scanner = new Scanner(System.in);
            System.out.print("Enter a string to check: ");
            String input = scanner.nextLine();

            outToServer.writeBytes(input + "\n");

            String response = inFromServer.readLine();
            System.out.println("Server response: " + response);

            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

How to Run (NetBeans or any IDE)

1. Create a new **Java project**.
2. Add both classes: `PalindromeTCPServer` and `PalindromeTCPClient`.
3. Run the server first, then run the client.

Sample Output

Client:

```
Enter a string to check: Madam
Server response: Palindrome
```

Server:

```
Server started. Waiting for client...
Client connected.
Received from client: Madam
```

Steps to Create Cookies in NetBeans

1. Create a New Project

1. Open **NetBeans**.
2. Go to **File > New Project**.
3. Select **Java Web > Web Application** → Click **Next**.
4. Give your project a name (e.g., `CookieDemo`) → Click **Finish**.

2. Create a Servlet for Setting Cookies

1. Right-click **Source Packages > New > Servlet**.
2. Name it `SetCookieServlet`.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SetCookieServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        // Create a cookie
        Cookie userCookie = new Cookie("username", "Virendra");
        userCookie.setMaxAge(60*60*24); // 1 day in seconds

        // Add cookie to response
        response.addCookie(userCookie);

        out.println("<h2>Cookie has been set successfully!</h2>");
    }
}
```

3. Create Another Servlet to Read Cookies

1. Right-click **Source Packages > New > Servlet**.
2. Name it `ReadCookieServlet`.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ReadCookieServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
```

```
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        Cookie[] cookies = request.getCookies();
        if (cookies != null) {
            for (Cookie c : cookies) {
                out.println("<p>" + c.getName() + " : " + c.getValue() +
"</p>");
            }
        } else {
            out.println("<h2>No cookies found!</h2>");
        }
    }
}
```

4. Deploy and Run

- Right-click the project → **Run**.
- Try opening:
  - `http://localhost:8080/CookieDemo/SetCookieServlet` → Sets the cookie.
  - `http://localhost:8080/CookieDemo/ReadCookieServlet` → Reads and displays cookie values.