Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering
Semester: (Fall, Year:2024), B.Sc. in CSE (Day)

Lab Report NO: 01
Course Title: Computer Networking Lab

Course Code: CSE-304 **Section:221-D21**

Lab Experiment Name: Implementation of HTTP POST and GET methods.

Student Details

	Name	**ID**
1.	Masum Hossain	221902164

Lab Date	**: 15/09/2024**
Submission Date	**: 15/11/2024**
Course Teacher's Name	**: Md. Saiful Islam Bhuiyan**

Lab Report Status
Marks: ………………………………… **Signature:.....................**
Comments:... **Date:..............................**

1. TITLE OF THE LAB REPORT EXPERIMENT

Implementation of HTTP POST and GET methods.

2. OBJECTIVES

This lab experiment involves creating a Java program to perform HTTP GET and POST requests using the HttpURLConnection class.As we are concerned that, the GET request fetches data from a specified endpoint, while the POST request sends data to a server. The aim is to understand and implement fundamental REST API (web service interface) request methods using Java as well as implementing the following objectives:

1. To understand the fundamentals of HTTP communication.
2. To implement a GET request to retrieve data from an external API.
3. To implement a POST request to send data to an external API.
4. To process and display the response from the server.

3. ANALYSIS

In this experiment both GET and POST methods have been implemented for fetching data from the server and sending data to the server respectively. Here we connect to an online REST API (https://jsonplaceholder.typicode.com) to perform GET and POST requests using Java.

Algorithm of the Program:

1.Define two URLs for GET and POST requests.
2.Create a method, sendGET for the GET request:
- Open an HTTP connection to the specified URL.
- Set the request method to GET.
- Check the response code; if successful (200) read and display the response content.

3.Create a method, sendPOST for the POST request:
- Open an HTTP connection to the specified URL.
- Set the request method to POST and define the content type as JSON.
- Enable output on the connection and write the JSON string to the request body.
- Check the response code; if successful (201), read and display the response content.

4.Execute the sendGET and sendPOST methods in the main function and handle any exceptions.

4. IMPLEMENTATION

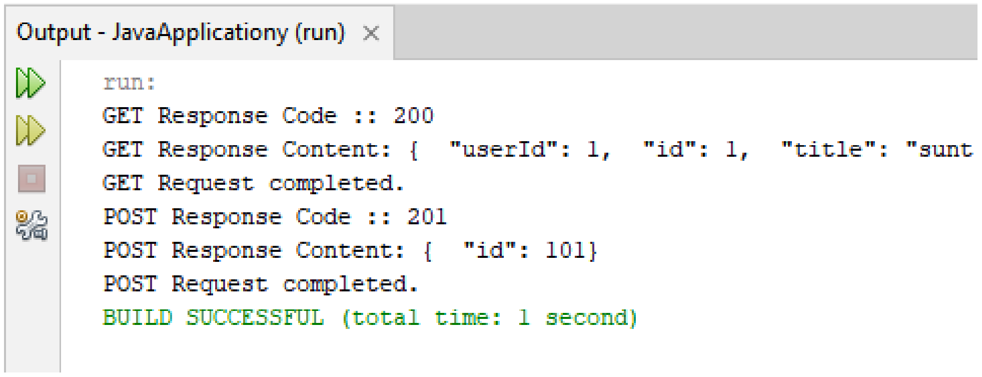
The program is implemented in Java using the HttpURLConnection class for network communication. We define two methods, sendGET and sendPOST. In sendGET, the program connects to a specified URL with a GET request, retrieves the data, and reads the response. In sendPOST, the program establishes a POST request connection to the URL and sends a JSON payload in the request body. The server's response code and response body are read and displayed for each request, validating the success of both operations.

Implemented Code

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;

public class GetPostMainClass {

    private static final String GET_URL =
"https://jsonplaceholder.typicode.com/posts/1";
    private static final String POST_URL =
"https://jsonplaceholder.typicode.com/posts";

    public static void main(String[] args) {
        try {
            sendGET();
            System.out.println("GET Request completed.");

            sendPOST();
            System.out.println("POST Request completed.");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private static void sendGET() throws Exception {
        URL url = new URL(GET_URL);
        HttpURLConnection httpURLConnection = (HttpURLConnection)
url.openConnection();
        httpURLConnection.setRequestMethod("GET");

        int responseCode = httpURLConnection.getResponseCode();
        System.out.println("GET Response Code :: " + responseCode);
        if (responseCode == HttpURLConnection.HTTP_OK) { // successGET
            BufferedReader in = new BufferedReader(new
InputStreamReader(httpURLConnection.getInputStream()));
            String inputLine;
            StringBuilder content = new StringBuilder();

            while ((inputLine = in.readLine()) != null) {
                content.append(inputLine);
            }

            System.out.println("GET Response Content: " + content.toString());
            in.close();
        } else {
            System.out.println("GET request failed.");
        }
    }

    private static void sendPOST() throws Exception {
        URL url = new URL(POST_URL);
        HttpURLConnection httpURLConnection = (HttpURLConnection)
url.openConnection();
        httpURLConnection.setRequestMethod("POST");
        httpURLConnection.setRequestProperty("Content-Type", "application/json; utf-
8");
        httpURLConnection.setRequestProperty("Accept", "application/json");
        httpURLConnection.setDoOutput(true);

        String jsonInputString = "{ \"title\": \"foo\", \"body\": \"bar\", \"userId\": 1 }";

        try (OutputStream os = httpURLConnection.getOutputStream()) {
            byte[] input = jsonInputString.getBytes("utf-8");
            os.write(input, 0, input.length);
        }

        int responseCode = httpURLConnection.getResponseCode();
        System.out.println("POST Response Code :: " + responseCode);

        if (responseCode == HttpURLConnection.HTTP_CREATED) {
            BufferedReader in = new BufferedReader(new
InputStreamReader(httpURLConnection.getInputStream()));
            String inputLine;
            StringBuilder content = new StringBuilder();

            while ((inputLine = in.readLine()) != null) {
                content.append(inputLine);
            }

            System.out.println("POST Response Content: " + content.toString());
            in.close();
        } else {
            System.out.println("POST request failed.");
        }}}
```

5. OUTPUT

For these three programs, these have been tested several times and we have got the desired output each time according to data sent by the server.



```
run:
GET Response Code :: 200
GET Response Content: {  "userId": 1,  "id": 1,  "title": "sunt
GET Request completed.
POST Response Code :: 201
POST Response Content: {  "id": 101}
POST Request completed.
BUILD SUCCESSFUL (total time: 1 second)
```

Fig 01: Showing the GET and POST response.

6. ANALYSIS AND DISCUSSION

The experiment was successful in demonstrating how to implement GET and POST requests using Java. The program made use of HttpURLConnection to establish the connection and retrieve responses, handling different response codes for validation. Through this exercise, we explored the importance of HTTP status codes in evaluating the success or failure of network requests and gained insights into handling JSON payloads in POST requests.

7. SUMMARY:

A REST API is a web service interface that allows communication between systems using standard HTTP methods to perform actions on resources identified by URLs.

Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Fall, Year:2024), B.Sc. in CSE (Day)

Lab Report NO: 02

Course Title: Computer Networking Lab

Course Code: CSE-304 **Section:221-D21**

Lab Experiment Name: Implementation of SMTP for two different networks.

Student Details

	Name	ID
1.	Masum Hossain	221902164

Lab Date	**: 22/09/2024**
Submission Date	**: 15/11/2024**
Course Teacher's Name	**: Md. Saiful Islam Bhuiyan**

Lab Report Status

Marks: ………………………………… **Signature:.....................**

Comments:... **Date:..............................**

1. TITLE OF THE LAB REPORT EXPERIMENT
Implementation of SMTP for Two Different Networks

2. OBJECTIVES
The purpose of this experiment was to configure and implement the Simple Mail Transfer Protocol (SMTP) across two distinct networks. The focus was to establish communication between devices on separate subnets to demonstrate email transmission using the SMTP protocol. The objectives of this lab were as follows:

1. To understand the fundamental working principles of SMTP.
2. To configure SMTP servers and clients within a Cisco Packet Tracer simulation.
3. To establish connectivity between two different networks and enable the exchange of email messages.
4. To analyze the flow of email messages across subnets using Packet Tracer simulation tools.

3. ANALYSIS
SMTP is a protocol used for sending email messages between servers. In this lab, we created a simulation environment using Cisco Packet Tracer, where two networks were configured to communicate via SMTP. Each network had its own email server and client PCs. Routing and connectivity were ensured between the networks.
The key steps of this experiment:
Network Design: Two distinct networks were created, each with its own IP addressing scheme.
SMTP Server Configuration: Email servers in both networks were configured with appropriate settings, including domain names and user accounts.
Client Setup: PCs in both networks were configured with email client software and user credentials for the SMTP servers.
Routing Configuration: Routers were set up to enable communication between the two networks.
Testing Email Transmission: Email messages were sent from a client in one network to a client in the other, ensuring successful message delivery.

4. IMPLEMENTATION

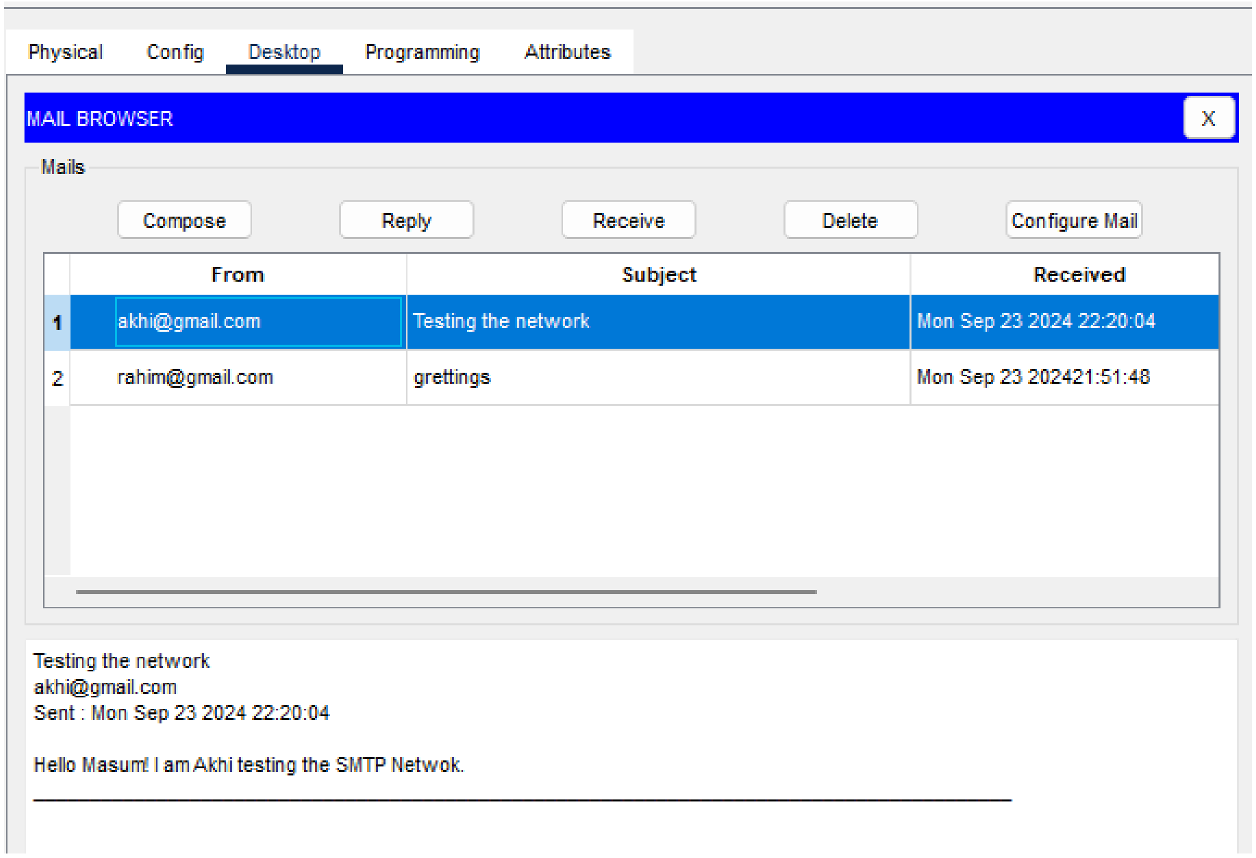
Steps in the Implementation:

- **Network Setup:**
  - Created two networks with separate subnets (2.168.1.0 and 192.168.2.0).
  - Configured routers with static routes or dynamic routing protocols to establish connectivity between the subnets.
- **SMTP Server Configuration:**

    - Deployed SMTP servers in both networks using Cisco Packet Tracer's server functionality.
    - Added user accounts on each server (rahim@gmail.com and akhi@gmail.com).
- **Client Configuration:**
    - Configured PCs with email client software to connect to the SMTP servers using the assigned user credentials.
- **Connectivity Testing:**
    - Verified inter-network communication by pinging devices across the networks.
- **Email Testing:**
    - Send emails from one network's client to the other. Monitored the flow of packets to ensure successful delivery.

OUTPUT

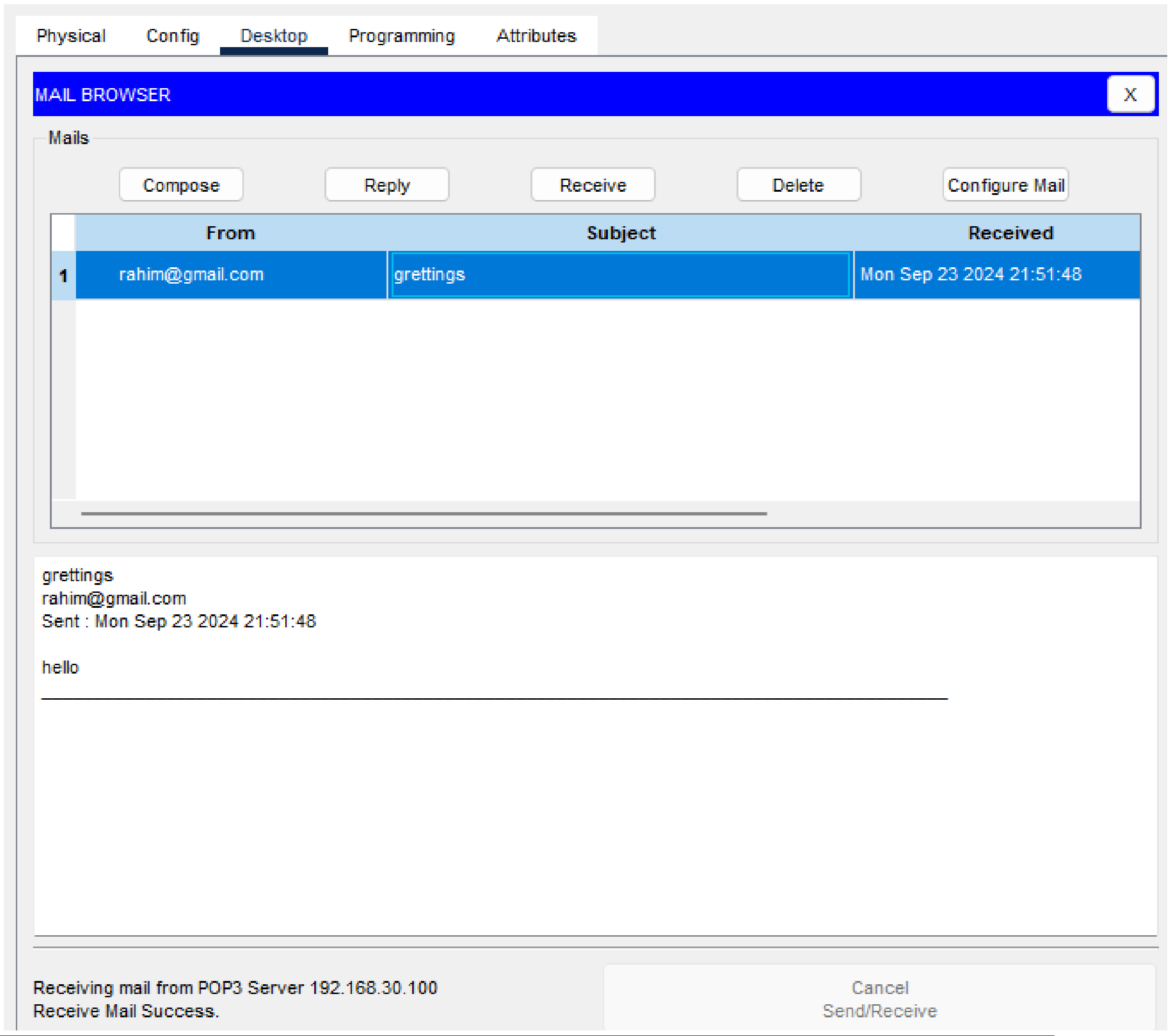
The simulation showed successful email transmission between the two networks. SMTP messages were observed traversing through routers, demonstrating proper protocol implementation and routing. Here a screenshot is included to demonstrate the process.



	From	Subject	Received
1	akhi@gmail.com	Testing the network	Mon Sep 23 2024 22:20:04
2	rahim@gmail.com	grettings	Mon Sep 23 202421:51:48

Testing the network
akhi@gmail.com
Sent : Mon Sep 23 2024 22:20:04

Hello Masum! I am Akhi testing the SMTP Netwok.



	From	Subject	Received
1	rahim@gmail.com	grettings	Mon Sep 23 2024 21:51:48

grettings
rahim@gmail.com
Sent : Mon Sep 23 2024 21:51:48

hello

```
Receiving mail from POP3 Server 192.168.30.100
Receive Mail Success.
```

Fig 01: Showing email flow between the networks.

6. ANALYSIS AND DISCUSSION

The experiment successfully demonstrated the implementation of SMTP for communication between two distinct networks. Proper configuration of routers and routing tables was critical to establishing seamless connectivity between the networks. Packet Tracer's simulation tools provided valuable insights by enabling the analysis of packet flow, allowing us to observe the functionality of the SMTP protocol in detail. Despite initial challenges with routing configurations and SMTP server settings, systematic troubleshooting ensured successful email transmission. This exercise emphasized the importance of protocol knowledge and accurate network configuration in achieving efficient communication between devices in separate networks which was successfully implemented in this lab experiment.

Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Fall, Year:2024), B.Sc. in CSE (Day)

Lab Report NO: 03

Course Title: Computer Networking Lab

Course Code: CSE-304 Section:221-D21

Lab Experiment Name: Implementation of Iterative DNS Query for DNS Records.

Student Details

	Name	ID
1.	Masum Hossain	221902164

Lab Date	**: 29/09/2024**
Submission Date	**: 15/11/2024**
Course Teacher's Name	**: Md. Saiful Islam Bhuiyan**

Lab Report Status

Marks: ………………………………… Signature:.....................

Comments:... Date:..............................

1. TITLE OF THE LAB REPORT EXPERIMENT

Implementation of Iterative DNS Query for DNS Records

2. OBJECTIVES

This lab experiment involves creating a Java program to perform iterative DNS queries to retrieve IP addresses associated with given domain names. The aim is to understand and implement fundamental concepts of DNS resolution using iterative queries. The specific objectives of this lab experiment are:

- To understand the structure and functionality of DNS records.
- To implement an iterative approach to query DNS records for IP addresses.
- To analyze the efficiency of iterative queries compared to recursive queries.
- To demonstrate the retrieval process of DNS records through a simple Java application.

3. ANALYSIS

In this experiment, both iterative and recursive approaches were implemented to fetch the IP address corresponding to a domain name. The program maintains a list of DNS records and searches through these records according to the chosen method. The algorithm follows these steps:

For Iterative Approach

1. Create a list of DNS records containing domain names and their corresponding IP addresses.
2. Implement a method to search for an IP address based on a given domain name using iteration.
3. Return the IP address if found, or indicate that the domain was not found.

For Recursive Approach

1. Create an array of DNS records containing domain names and their corresponding IP addresses.
2. Implement a method that searches for an IP address based on a given domain name using recursion.
3. Return the IP address if found, or proceed recursively until all records have been checked.

The main function allows the user to choose between the two methods and input a domain name to query.

4. IMPLEMENTATION

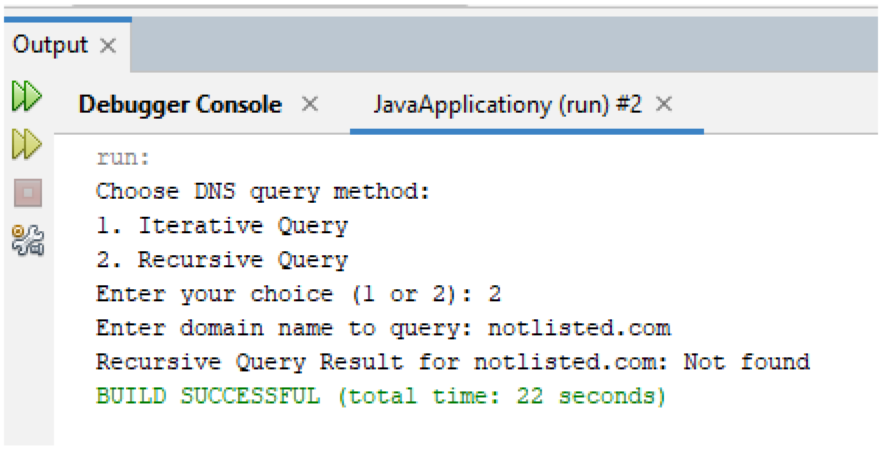
The program is implemented in Java, using a simple class structure for DNS records along with two separate classes for iterative and recursive queries.

Implemented Code

```java
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
class DNSRecord {
    private String domain;
    private String ipAddress;

    public DNSRecord(String domain, String ipAddress) {
        this.domain = domain;
        this.ipAddress = ipAddress;
    }
    public String getDomain() {
        return domain;
    }

    public String getIpAddress() {
        return ipAddress;
    }
}
class IterativeDNSQuery {
    private List<DNSRecord> dnsRecords;

    public IterativeDNSQuery() {
        dnsRecords = new ArrayList<>();
        dnsRecords.add(new DNSRecord("example.com", "93.184.216.34"));
        dnsRecords.add(new DNSRecord("example.org",
"2606:2800:220:1:248:1893:25c8:1946"));
        dnsRecords.add(new DNSRecord("google.com", "172.217.14.206"));
        dnsRecords.add(new DNSRecord("facebook.com", "157.240.22.35"));
    }
    public String findIpAddress(String domain) {
        for (DNSRecord record : dnsRecords) {
            if (record.getDomain().equals(domain)) {
                return record.getIpAddress();
            }
        }
        return "Not found";
    }
}
class RecursiveDNSQuery {
   private DNSRecord[] dnsRecords;
   public RecursiveDNSQuery() {
      dnsRecords = new DNSRecord[5];
      dnsRecords[0] = new DNSRecord("example.com", "93.184.216.34");
      dnsRecords[1] = new DNSRecord("example.org",
"2606:2800:220:1:248:1893:25c8:1946");
      dnsRecords[2] = new DNSRecord("google.com", "172.217.14.206");
      dnsRecords[3] = new DNSRecord("linkedin.com", "192.0.2.1");
      dnsRecords[4] = new DNSRecord("facebook.com", "157.240.22.35");
   }
   public String findIpAddress(String domain, int index) {
      if (index >= dnsRecords.length) {
         return "Not found";
      }
      if (dnsRecords[index].getDomain().equals(domain)) {
         return dnsRecords[index].getIpAddress();
      }
      return findIpAddress(domain, index + 1); }}
public class DNSQueryProgram {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      IterativeDNSQuery iterativeQuery = new IterativeDNSQuery();
      RecursiveDNSQuery recursiveQuery = new RecursiveDNSQuery();
      System.out.println("Choose DNS query method:");
      System.out.println("1. Iterative Query");
      System.out.println("2. Recursive Query");
      System.out.print("Enter your choice (1 or 2): ");
      int choice = scanner.nextInt();

      System.out.print("Enter domain name to query: ");
      String domain = scanner.next();
      if (choice == 1) {
         System.out.println("Iterative Query Result for " + domain + ": " +
iterativeQuery.findIpAddress(domain));
      } else if (choice == 2) {
         System.out.println("Recursive Query Result for " + domain + ": " +
recursiveQuery.findIpAddress(domain, 0));
      } else {
         System.out.println("Invalid choice! Please enter 1 or 2.");
      }
      scanner.close(); }}
```

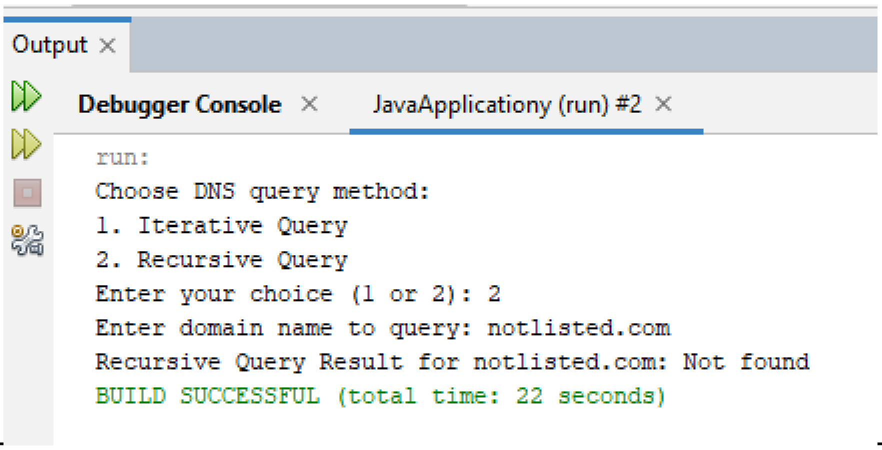
OUTPUT

The console output will display the result of the DNS query based on the user's input. For example, if the user queries "google.com," the output will look like this:



```
run:
Choose DNS query method:
1. Iterative Query
2. Recursive Query
Enter your choice (1 or 2): 2
Enter domain name to query: notlisted.com
Recursive Query Result for notlisted.com: Not found
BUILD SUCCESSFUL (total time: 22 seconds)
```

Fig 01: Showing the output for iterative Query listed on Array.



```
run:
Choose DNS query method:
1. Iterative Query
2. Recursive Query
Enter your choice (1 or 2): 2
Enter domain name to query: notlisted.com
Recursive Query Result for notlisted.com: Not found
BUILD SUCCESSFUL (total time: 22 seconds)
```

Fig 02: Showing the output for Recursive Query not listed on Array.

6. ANALYSIS AND DISCUSSION

The experiment successfully demonstrated the implementation of both iterative and recursive DNS queries within a Java program. By structuring DNS records and responding to user queries, we gained insights into the differences in implementation between the two methods.

The iterative approach proved straightforward and efficient for small datasets, while the recursive approach showcased the elegance of recursive function calls but may add overhead due to multiple stack frames. In practical applications, iterative queries often outperform recursive queries, especially as the size of DNS records grows.The process of developing this application reinforced the understanding of how DNS resolution works and the programming concepts of lists.

Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Fall, Year:2024), B.Sc. in CSE (Day)

Lab Report NO: 04

Course Title: Computer Networking Lab

Course Code: CSE-304 Section:221-D21

Lab Experiment Name: Configuration of static and dynamic routing protocols.

<u>Student Details</u>

	Name	ID
1.	Masum Hossain	221902164

Lab Date	**: 29/11/2024**
Submission Date	**: 15/12/2024**
Course Teacher's Name	**: Md. Saiful Islam Bhuiyan**

<u>Lab Report Status</u>

Marks: ………………………………… **Signature:.....................**

Comments:... **Date:..............................**

1. TITLE OF THE LAB REPORT EXPERIMENT

Configuration of static and dynamic routing protocols.

2. OBJECTIVES

This lab experiment involves creating a mesh networking for both static and dynamic routing.In case of static routing, we will define the path which will be followed from source to destination carrying the message.Conversely, for dynamic routing RIP(Routing Information Protocol) will be used where we will just input the router connected with the network. The specific objectives of this lab experiment are:

- To configure static routing in a network topology.
- To implement dynamic routing protocols (e.g., RIP, OSPF) in a simulated network.
- To analyze the performance and functionality of both static and dynamic routing.
- To compare the advantages and disadvantages of static and dynamic routing in terms of scalability, simplicity, and fault tolerance

3. ANALYSIS

Routing is an essential process in computer networks, enabling data packets to find their paths from source to destination. This lab focuses on configuring and analyzing both static and dynamic routing protocols using a Packet Tracer simulation. Static routing requires manual configuration, while dynamic routing adapts automatically to changes in the network topology. By implementing both types of routing in this lab, we will explore their functionalities and evaluate their use cases.

The lab consists of a simulated network created in Cisco Packet Tracer. The topology includes multiple routers and subnets, interconnected to represent a realistic network environment. Static routes were configured manually, and dynamic routing protocols such as RIP were applied to demonstrate automatic route discovery and management.

Key Components:

1. Multiple routers with interconnected FastEthernet and Serial interfaces.
2. Various subnets configured with unique IP addressing schemes(six different networks are assigned here).
3. Static and dynamic routing protocols implemented to manage data traffic.

4. IMPLEMENTATION

The program is implemented in Java,Static Routing Configuration
Define Static Routes:
For each router, static routes were added using the Config tab in Packet Tracer.
Example:
Network: 192.168.1.0,192.168.2.0 ……………….. 192.168.6.0 (six different networks)
Mask: 255.255.255.0
Next Hop: 192.168.4.2

Verification:
Used ping commands from devices to ensure connectivity.
Checked routing tables to confirm that static routes were added correctly.
Dynamic Routing Configuration (RIP)
Enable RIP Protocol:
On each router, RIP was activated using the Config tab.
Networks directly connected to each router were added to the RIP protocol.

Example:
Router 1: Added 192.168.1.0/24 and 192.168.2.0/24 to RIP.
Verification:
Observed automatic propagation of routes in the routing tables.
Used ping commands to verify end-to-end connectivity.

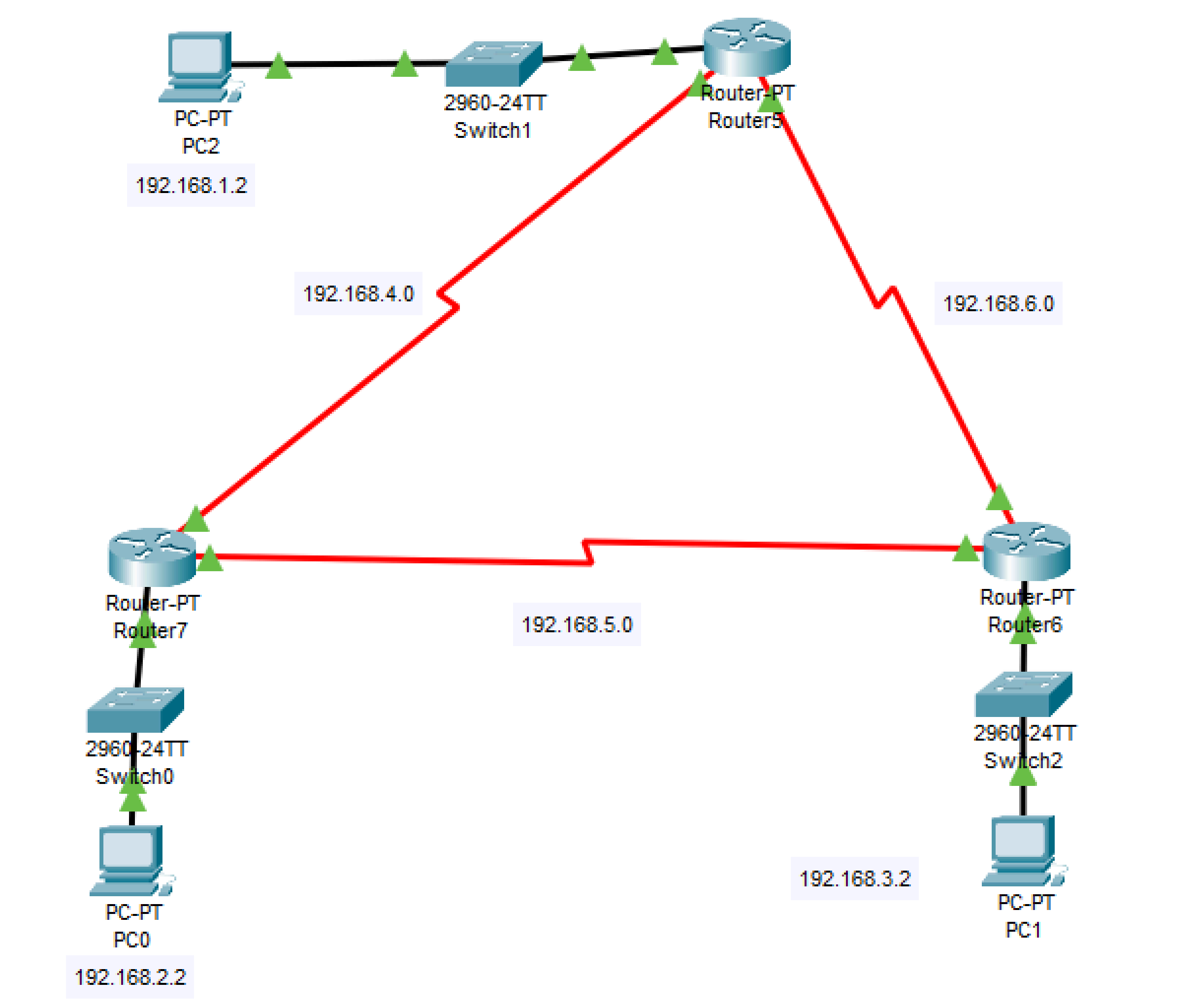
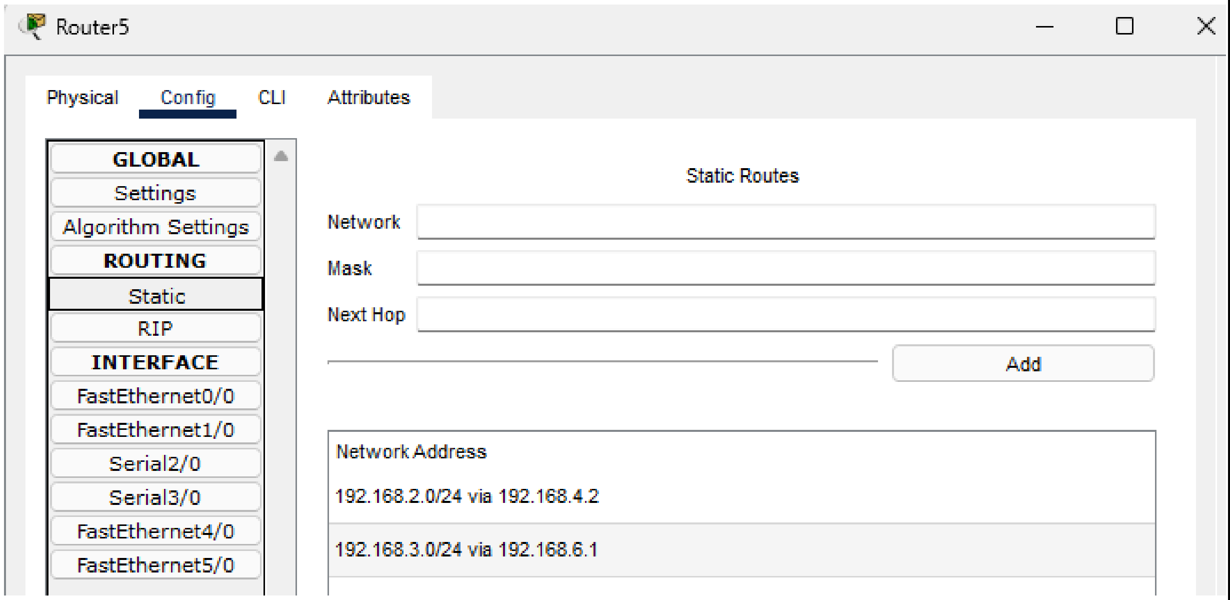


Fig 01: Configuring the routers as well as assigning the IP address to each device.



Static Routes

Network

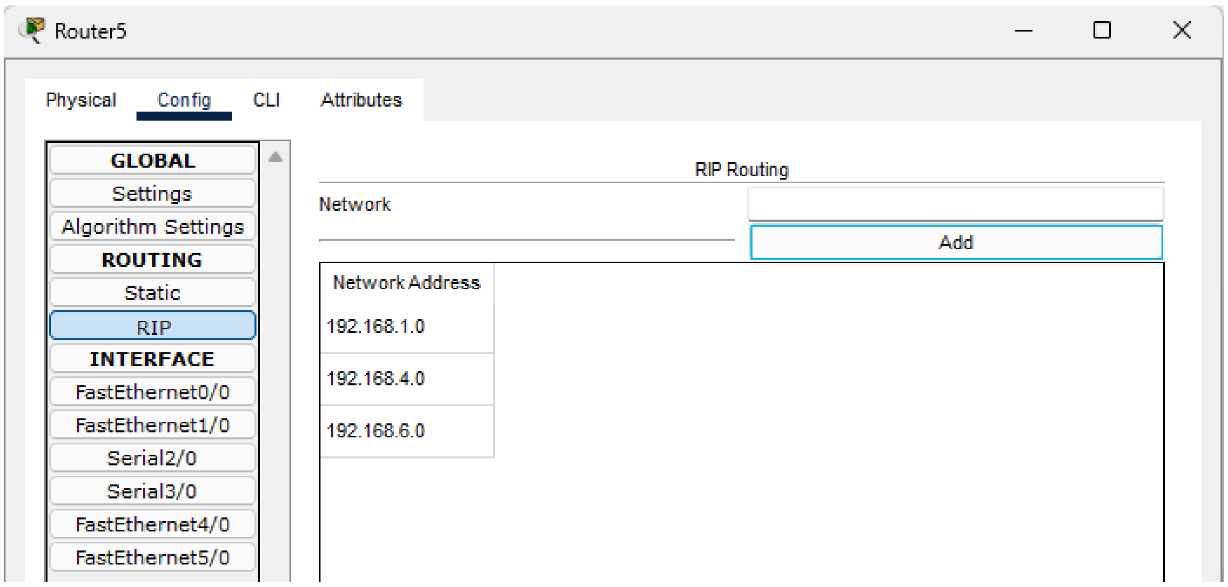
Mask

Next Hop

Network Address

192.168.2.0/24 via 192.168.4.2

192.168.3.0/24 via 192.168.6.1

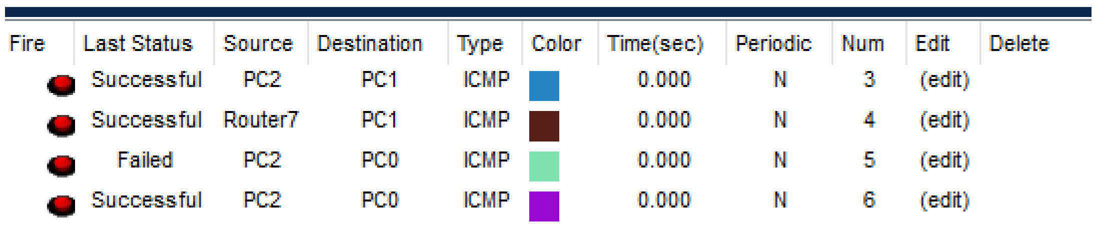


Network Address
192.168.1.0
192.168.4.0
192.168.6.0

Fig 03: Configuring the static and dynamic routing.

OUTPUT

The console output will display the result of the DNS query based on the user's input. For example, if the user queries "google.com," the output will look like this:



Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
●	Successful	PC2	PC1	ICMP	🟦	0.000	N	3	(edit)	
●	Successful	Router7	PC1	ICMP	🟫	0.000	N	4	(edit)	
●	Failed	PC2	PC0	ICMP	🟩	0.000	N	5	(edit)	
●	Successful	PC2	PC0	ICMP	🟪	0.000	N	6	(edit)	

Fig 04: Showing the message after a successful sending operation.

6. ANALYSIS AND DISCUSSION

In this lab, the configuration and comparison of static and dynamic routing protocols were successfully demonstrated. Static routing, though straightforward and predictable, is limited in scalability and adaptability. Conversely, dynamic routing protocols like RIP excel in dynamic and larger network environments by automatically adjusting to changes. Understanding these protocols and their respective strengths and weaknesses is vital for designing efficient and resilient network infrastructures. The knowledge gained from this lab will be instrumental in selecting the appropriate routing strategy based on specific network requirements.

Static routing offers precise control and simplicity but becomes impractical in larger, dynamic networks due to manual configurations. Dynamic routing, exemplified by RIP, adapts automatically to topology changes, saving time and minimizing errors. However, it demands higher resources and careful configuration. Ultimately, static routing suits small, stable networks, while dynamic routing is essential for scalability and fault tolerance in complex environments.

Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Fall, Year:2024), B.Sc. in CSE (Day)

Lab Report NO: 03

Course Title: Computer Networking Lab

Course Code: CSE-304 **Section:221-D21**

Lab Experiment Name: Implementation of socket programming using threading.

<u>Student Details</u>

	Name	**ID**
1.	Masum Hossain	221902164

Lab Date	**: 29/12/2024**
Submission Date	**: 15/12/2024**
Course Teacher's Name	**: Md. Saiful Islam Bhuiyan**

<u>Lab Report Status</u>

Marks: ………………………………… **Signature:.....................**

Comments:... **Date:..............................**

1. TITLE OF THE LAB REPORT EXPERIMENT

Implementation of Socket Programming Using Threading.

2. OBJECTIVES

This lab experiment involves creating a Java program to perform server and client communication for different scenarios.Here we will implement the client limit for 4 users only instead of unlimited users. The specific objectives of this lab experiment are:

- To understand the principles of socket programming and multithreading.
- To implement a Java-based server-client application that supports multiple client connections.
- To learn how to handle client requests and process data on a multithreaded server.
- To gain practical experience in data communication over TCP sockets.

3. ANALYSIS

This experiment involves implementing a multithreaded server-client application using Java. The server accepts multiple client connections, processes data, and communicates results back to the clients. The key features and processes include:

Server Side:

1. Listens for client connections on a specified port.
2. Accepts up to four clients and assigns each client a separate thread for interaction.
3. Processes sentences sent by clients by capitalizing the middle letters of words.

Client Side:

1. Connects to the server on the specified port.
2. Sends user-provided sentences to the server and receives the processed results.
3. Allows users to terminate the session by typing "Exit."

4. IMPLEMENTATION

The program is implemented in Java, using a simple class structure for both client side and server side.ClientHandler class is used to limit the number of clients.

Server Side Code

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

public class ServerThread {
    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket(5000);
        System.out.println("Server is running on port 5000...");

        int client = 0;

        while (client < 4) {
            Socket clientSocket = serverSocket.accept();
            client++;
            System.out.println("Client " + client + " connected: " + clientSocket);

            DataInputStream dis = new DataInputStream(clientSocket.getInputStream());
            DataOutputStream dos = new DataOutputStream(clientSocket.getOutputStream());

            Thread clientThread = new ClientHandler(clientSocket, dis, dos, client);
            clientThread.start();
        }

        System.out.println("Server lifetime ended. No more clients accepted.");
        serverSocket.close();
    }
}

class ClientHandler extends Thread {
    private final Socket socket;
    private final DataInputStream dis;
    private final DataOutputStream dos;
    private final int client;

    public ClientHandler(Socket socket, DataInputStream dis, DataOutputStream dos, int client)
{
        this.socket = socket;
        this.dis = dis;
        this.dos = dos;
        this.client = client;
    }
    @Override
    public void run() {
        try {
            int sentenceCount = 0;

            while (sentenceCount < 3) {
                dos.writeUTF("Client " + client + ": Send a sentence (or type 'Exit' to quit): ");
                String received = dis.readUTF();

                if (received.equalsIgnoreCase("Exit")) {
                    System.out.println("Client " + client + " disconnected.");
                    break;
                }

                String convertedSen = capitalizeMiddleLetters(received);
                dos.writeUTF("Processed: " + convertedSen);
                System.out.println("Processed for Client " + client + ": " + convertedSen);
                sentenceCount++;
            }

            System.out.println("Client " + client + " session ended.");
            socket.close();
            dis.close();
            dos.close();
        } catch (IOException e) {
            System.err.println("Error handling client " + client + ": " + e.getMessage());
        }
    }

    private String capitalizeMiddleLetters(String sentence) {
        String[] words = sentence.split(" ");
        StringBuilder result = new StringBuilder();

        for (String word : words) {
            if (word.length() == 0) continue;

            int mid = word.length() / 2;
            char middleChar = Character.toUpperCase(word.charAt(mid));

            String transformedWord = word.substring(0, mid) + middleChar + word.substring(mid
+ 1);
            result.append(transformedWord).append(" ");
        }

        return result.toString().trim();
    }
}
```

Client Side Code

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

public class ClientThread {
    public static void main(String[] args) {
        try {
            Socket clientSocket = new Socket("localhost", 5000);
            System.out.println("Connected to server.");

            DataOutputStream dos = new DataOutputStream(clientSocket.getOutputStream());
            DataInputStream dis = new DataInputStream(clientSocket.getInputStream());
            Scanner scanner = new Scanner(System.in);

            while (true) {
                String serverMessage = dis.readUTF();
                System.out.println(serverMessage);

                System.out.print("Please Enter any sentence: ");
                String input = scanner.nextLine();
                dos.writeUTF(input);

                if (input.equalsIgnoreCase("Exit")) {
                    System.out.println("Exiting session.");
                    break;
                }
                String response = dis.readUTF();
                System.out.println("Server Response: " + response);
            }

            clientSocket.close();
            dis.close();
            dos.close();
        } catch (IOException e) {
            System.err.println("Error: " + e.getMessage());
        }
    }
}
```

OUTPUT

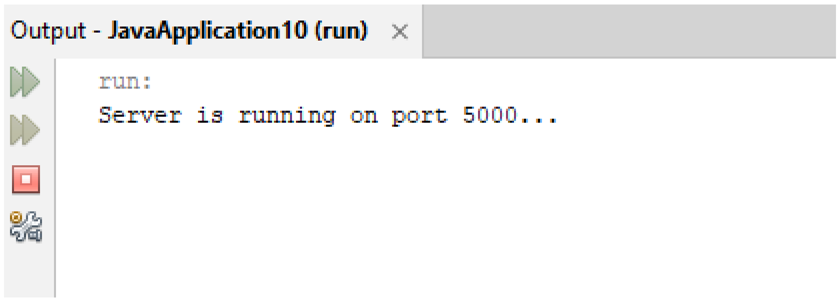
The console output will display the result based on the user's input. For example, if the user queries "," the output will look like this:

Output for Server Side

```
Server is running on port 5000...
Client 1 connected: Socket[addr=/127.0.0.1,port=50689,localport=5000]
Processed for Client 1: tEst Middle woRds
Client 1 session ended.
```

Output for Client Side

```
Connected to server.
Client 1: Send a sentence (or type 'Exit' to quit):
Please Enter any sentence: test middle words
Server Response: Processed: tEst Middle woRds
```



```
run:
Server is running on port 5000...
```

Fig 01: Showing the server is running with the desired port.

6. ANALYSIS AND DISCUSSION

This lab experiment successfully met its objectives by implementing a multithreaded server-client application in Java. Through the practical demonstration of socket programming and threading, it emphasized the importance of designing scalable and interactive networked applications. The experiment also reinforced the significance of concurrent processing in modern networking solutions, paving the way for further exploration of advanced topics in this domain.

The implementation of socket programming using threading provided valuable insights into building concurrent applications. The server's multithreaded design allowed simultaneous handling of multiple clients, demonstrating how threading enhances scalability and responsiveness. Each client was assigned a separate thread, ensuring individualized processing of sentences and seamless interaction. The communication between the client and server, facilitated by Java’s I/O streams, reinforced understanding of data exchange mechanisms in networked systems. Additionally, the program’s functionality to process sentences by capitalizing the middle letters of words showcased efficient string manipulation. Overall, this experiment not only strengthened foundational knowledge of networking and multithreading but also highlighted the practical applications of these concepts in real-world scenarios.