Green University of Bangladesh
Department of Computer Science and Engineering (CSE)
Faculty of Sciences and Engineering
Semester: (Summer, Year: 2025), B.Sc. in CSE (Day)

LAB REPORT NO: 03
Course Title: Computer Networking Lab
Course Code: CSE-318   Section: 231-D2

Lab Experiment Name: Implementation of socket programming   using threading

<u>Student Details</u>

	Name	ID
1.	**Promod Chandra Das**	**231002005**

Lab Date : 03.08.2025

Submission Date: 10.08.2025

Course Teacher's Name : Fatema Akter

[For Teachers use only: Don't Write Anything inside this box]

<u>Lab Report Status</u>
Marks: ………………………………
Comments: ....

Signature: ....
Date: ....

➢ **TITLE OF THE LAB EXPERIMENT :**

   Implementation of socket programming using threading

➢ **OBJECTIVES :**

1. To understand the concept of socket programming for network communication.
2. To implement a client-server model using Python/Java/C where multiple clients can connect simultaneously
3. To apply **threading** to handle multiple client requests concurrently without blocking.
4. To gain practical experience in network programming and multi-threaded application design.
5. To demonstrate real-time data exchange between multiple clients and a server.

➢ **PROCEDURE :**

1. Start the server and create a `ServerSocket` on a fixed port.
2. Initialize a counter for the number of clients served.
3. Use `accept()` to wait for client connections.
4. On each connection, increment the counter.
5. Pass the `Socket` to a `ClientHandler` class (Runnable).
6. Create and start a new thread for the handler.
7. In `ClientHandler`, use `DataInputStream` and `DataOutputStream` for communication.
8. Read client input in the format `num1,num2,operation.`
9. If `"ENDS"` is received, send a closing message and exit.
10. Parse the numbers and operation.
11. Perform the operation: Add, Subtract, Multiply, Divide, Modulus.
12. Send the result to the client.
13. Repeat until `"ENDS"` is received.
14. Close streams and socket after disconnect.
15. Stop new connections if client count reaches
16. End both server and client programs.

❖ **Server Side:**

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

public class MathServer {
    private static int clientCount = 0; // Number of clients served

    public static void main(String[] args) {
        final int PORT = 5000;
        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            System.out.println("Server started on port " + PORT);

            while (clientCount < 5) {
                Socket socket = serverSocket.accept();
                clientCount++;
                System.out.println("Client connected. Total clients served: " + clientCount);

                Thread t = new Thread(new ClientHandler(socket));
                t.start();
            }

            System.out.println("Server reached max client limit (5). Stopping new connections.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

class ClientHandler implements Runnable {
    private Socket socket;

    public ClientHandler(Socket socket) {
        this.socket = socket;
    }

    @Override
    public void run() {
        try {
            DataInputStream dis = new DataInputStream(socket.getInputStream());
            DataOutputStream dos = new DataOutputStream(socket.getOutputStream());

            dos.writeUTF("Connected to Math Server. Send in format: num1,num2,operation OR ENDS to quit");

            while (true) {
                String clientMsg = dis.readUTF();
```

```java
        if (clientMsg.equalsIgnoreCase("ENDS")) {
            dos.writeUTF("Connection closed by client.");
            break;
        }

        String[] parts = clientMsg.split(",");
        if (parts.length != 3) {
            dos.writeUTF("Invalid format. Use: num1,num2,operation");
            continue;
        }

        try {
            int num1 = Integer.parseInt(parts[0].trim());
            int num2 = Integer.parseInt(parts[1].trim());
            String op = parts[2].trim().toLowerCase();
            int result = 0;
            boolean validOp = true;

            switch (op) {
                case "sum":
                    result = num1 + num2;
                    break;
                case "subtract":
                    result = num1 - num2;
                    break;
                case "multiplication":
                    result = num1 * num2;
                    break;
                case "division":
                    if (num2 != 0) {
                        result = num1 / num2;
                    } else {
                        dos.writeUTF("Error: Division by zero");
                        continue;
                    }
                    break;
                case "modules": // modulus
                    result = num1 % num2;
                    break;
                default:
                    validOp = false;
                    dos.writeUTF("Invalid operation. Use: Sum, Subtract, Multiplication, Division, Modules");
            }

            if (validOp) {
                dos.writeUTF("Result: " + result);
            }
```

```java
                } catch (NumberFormatException e) {
                    dos.writeUTF("Error: Please enter valid integers.");
                }
            }

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

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

public class MathServer {
    private static int clientCount = 0; // Number of clients served

    public static void main(String[] args) {
        final int PORT = 5000;
        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            System.out.println("Server started on port " + PORT);

            while (clientCount < 5) {
                Socket socket = serverSocket.accept();
                clientCount++;
                System.out.println("Client connected. Total clients served: " + clientCount);

                Thread t = new Thread(new ClientHandler(socket));
                t.start();
            }

            System.out.println("Server reached max client limit (5). Stopping new connections.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```java
// server.java
public class MathServer {
}

class ClientHandler implements Runnable {
    private Socket socket;

    public ClientHandler(Socket socket) {
        this.socket = socket;
    }

    @Override
    public void run() {
        try {
            DataInputStream dis = new DataInputStream(socket.getInputStream());
            DataOutputStream dos = new DataOutputStream(socket.getOutputStream());

            dos.writeUTF("Connected to Math Server. Send in format: num1,num2,operation OR ENDS to quit");

            while (true) {
                String clientMsg = dis.readUTF();

                if (clientMsg.equalsIgnoreCase("ENDS")) {
                    dos.writeUTF("Connection closed by client.");
                    break;
                }

                String[] parts = clientMsg.split(",");
                if (parts.length != 3) {
                    dos.writeUTF("Invalid format. Use: num1,num2,operation");
```

```
        try {
            int num1 = Integer.parseInt(parts[0].trim());
            int num2 = Integer.parseInt(parts[1].trim());
            String op = parts[2].trim().toLowerCase();
            int result = 0;
            boolean validOp = true;

            switch (op) {
                case "sum":
                    result = num1 + num2;
                    break;
                case "subtract":
                    result = num1 - num2;
                    break;
                case "multiplication":
                    result = num1 * num2;
                    break;
                case "division":
                    if (num2 != 0) {
                        result = num1 / num2;
                    } else {
                        dos.writeUTF("Error: Division by zero");
                        continue;
                    }
                    break;
                case "modules": // modulus
                    result = num1 % num2;
                    break;
                default:
                    validOp = false;
                    dos.writeUTF("Invalid operation. Use: Sum, Subtract, Multiplication, Division, Modules");
            }

            if (validOp) {
                dos.writeUTF("Result: " + result);
            }

        } catch (NumberFormatException e) {
            dos.writeUTF("Error: Please enter valid integers.");
        }
    }

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

Fiqure: Server Side

❖ **Server side Output:**

Server started on port 5000
Client connected. Total clients served: 1
Client connected. Total clients served: 2
Client connected. Total clients served: 3
Client connected. Total clients served: 4
Client connected. Total clients served: 5
Server reached max client limit (5)
 Stopping new connections.

```
client.java
1    Server started on port 5000
2    Client connected. Total clients served: 1
3    Client connected. Total clients served: 2
4    Client connected. Total clients served: 3
5    Client connected. Total clients served: 4
6    Client connected. Total clients served: 5
7    Server reached max client limit (5)
8    Stopping new connections.
9
```

Fiqure: Server side Output:

❖ **Client side:**

```java
import java.io.*;import java.net.*;import java.util.Scanner;
public class MathClient {
   public static void main(String[] args) {
      final String SERVER = "localhost";
      final int PORT = 5000;

      try (Socket socket = new Socket(SERVER, PORT)) {
         DataInputStream dis = new DataInputStream(socket.getInputStream());
         DataOutputStream dos = new DataOutputStream(socket.getOutputStream());
         Scanner sc = new Scanner(System.in);

         System.out.println("Server: " + dis.readUTF());

         while (true) {
            System.out.print("Enter request (num1,num2,operation) or ENDS to quit: ");
            String input = sc.nextLine();
            dos.writeUTF(input);

            if (input.equalsIgnoreCase("ENDS")) {
               System.out.println("Server: " + dis.readUTF());
               break;
            }

            String response = dis.readUTF();
            System.out.println("Server: " + response);
         }

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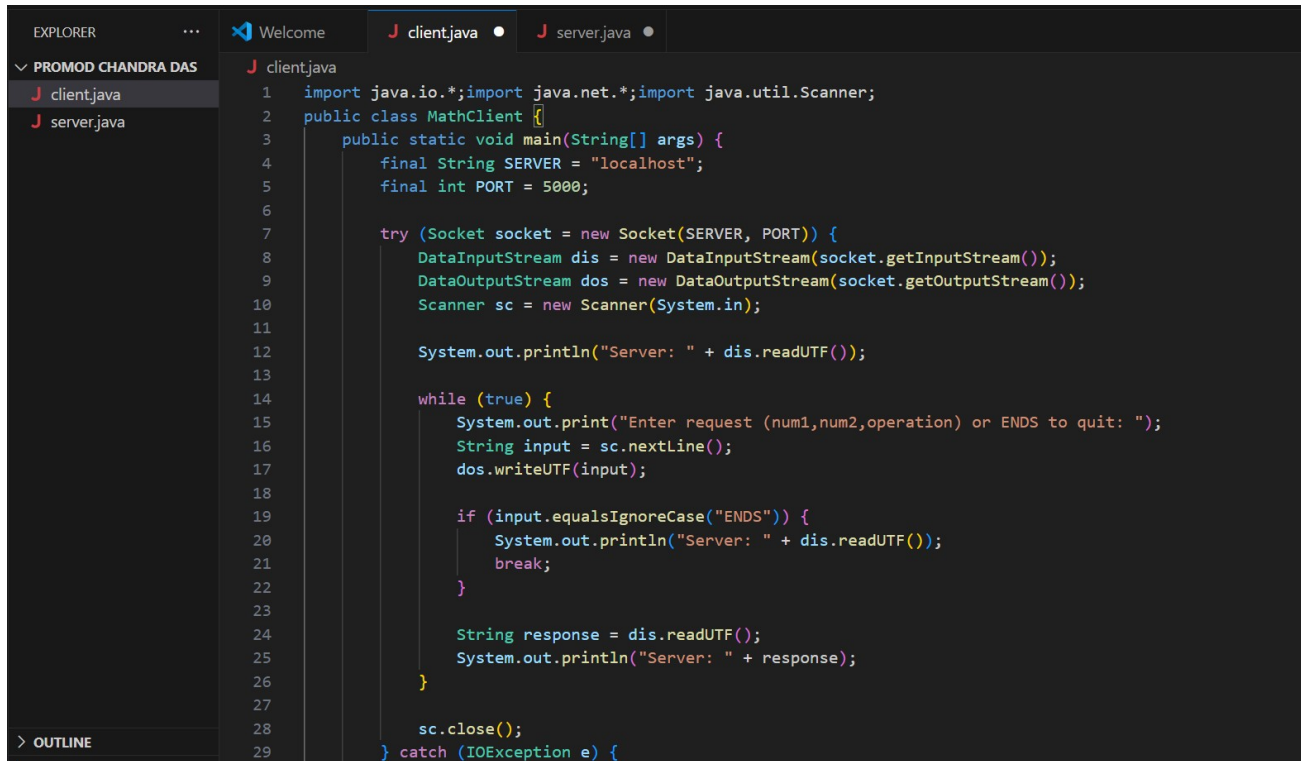
```
    }
}
```


Fiqure: Client side

❖ **Client Side Output :**

Server: Connected to Math Server. Send in format: num1,num2,operation OR ENDS to quit

Enter request (num1,num2,operation) or ENDS to quit: 10,20,Sum

Server: Result: 30

Enter request (num1,num2,operation) or ENDS to quit: 20,5,Subtract

Server: Result: 15

Enter request (num1,num2,operation) or ENDS to quit: 20,5,Multiplication

Server: Result: 100

Enter request (num1,num2,operation) or ENDS to quit: 20,5,Division

Server: Result: 4

Enter request (num1,num2,operation) or ENDS to quit: 16,3,Modules

Server: Result: 1

Enter request (num1,num2,operation) or ENDS to quit: ENDS

Server: Connection closed by client.


```
Server: Connected to Math Server. Send in format: num1,num2,operation OR ENDS to quit
Enter request (num1,num2,operation) or ENDS to quit: 10,20,Sum
Server: Result: 30
Enter request (num1,num2,operation) or ENDS to quit: 20,5,Subtract
Server: Result: 15
Enter request (num1,num2,operation) or ENDS to quit: 20,5,Multiplication
Server: Result: 100
Enter request (num1,num2,operation) or ENDS to quit: 20,5,Division
Server: Result: 4
Enter request (num1,num2,operation) or ENDS to quit: 16,3,Modules
Server: Result: 1
Enter request (num1,num2,operation) or ENDS to quit: ENDS
Server: Connection closed by client.
```

❖   **Fiqure: Client Side Output :**

➢ **DISCUSSION :**

❖ **Why Threading?**
Without threading, the server can only handle one client at a time (sequential processing). Threading allows simultaneous handling of multiple clients, improving responsiveness.

❖ **Advantages**: Efficient use of server resources.Real-time communication with multiple clients.Better user experience in chat, gaming, or file-transfer systems.

❖ **Limitations**:More complex debugging due to multiple threads running concurrently.Thread synchronization issues may arise if shared resources are used.Higher memory usage when many threads are active.

❖ **Real-world Applications**: Online chat applications (WhatsApp Web, Messenger).Multiplayer online games.Web servers handling multiple HTTP requests.