

**NAME: SHASHWAT SHAH**

**SAP ID: 60004220126**

**DIV/BATCH: C22**

**DISTRIBUTED COMPUTING (DC)  
EXPERIMENT 01**

**AIM: To implement Client-Server using RMI.**

**CODE:**

**AddInterface.Java**

```
import java.rmi.*;
public interface AddInterface extends Remote {
    // Declaring the sum method to be used remotely
    public int sum(int n1, int n2) throws RemoteException;
}
```

**Add.Java**

```
import java.rmi.*;
import java.rmi.server.*;
public class Add extends UnicastRemoteObject implements AddInterface {
    // Default constructor for Add
    public Add() throws RemoteException {
        super();
    }
    // Implementation of the sum method
    @Override
    public int sum(int n1, int n2) throws RemoteException {
        return n1 + n2;
    }
}
```

**AddServer.Java**

```
import java.rmi.Naming;
public class AddServer {
    public static void main(String[] args) {
        try {
            // Binding the remote object to the name "Add"
            Naming.rebind("Add", new Add());
            System.out.println("Server is connected and waiting for the client...");
        } catch (Exception e) {
            System.out.println("Server could not connect: " + e);
        }
    }
}
```

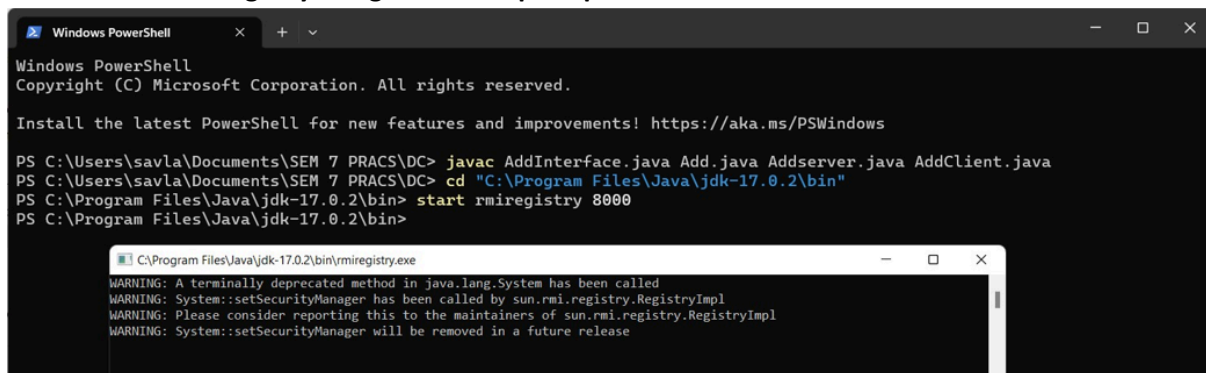
### AddClient.java

```
import java.rmi.Naming;

public class AddClient {
    public static void main(String[] args) {
        try {
            // Looking up the remote object using its name
            AddInterface ai = (AddInterface) Naming.lookup("//localhost/Add");
            System.out.println("The sum of two numbers is: " + ai.sum(10, 1));
        } catch (Exception e) {
            System.out.println("Client Exception: " + e);
        }
    }
}
```

### OUTPUT:

- Compile AddInterface.java, Add.java, AddServer.java and AddClient.java
- Start RMI registry using command prompt



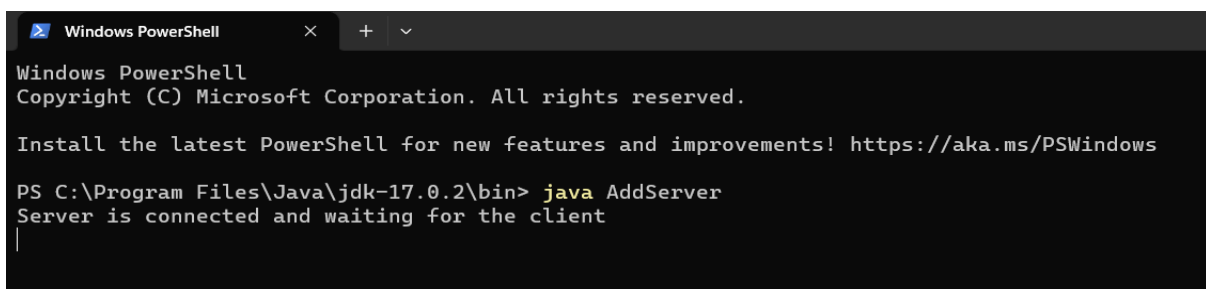
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\savla\Documents\SEM 7 PRACS\DC> javac AddInterface.java Add.java Addserver.java AddClient.java
PS C:\Users\savla\Documents\SEM 7 PRACS\DC> cd "C:\Program Files\Java\jdk-17.0.2\bin"
PS C:\Program Files\Java\jdk-17.0.2\bin> start rmiregistry 8000
PS C:\Program Files\Java\jdk-17.0.2\bin>
```

A secondary window titled 'C:\Program Files\Java\jdk-17.0.2\bin\rmiregistry.exe' is also visible, showing several warning messages about deprecated methods and security manager changes.

- Execute AddCServer.java

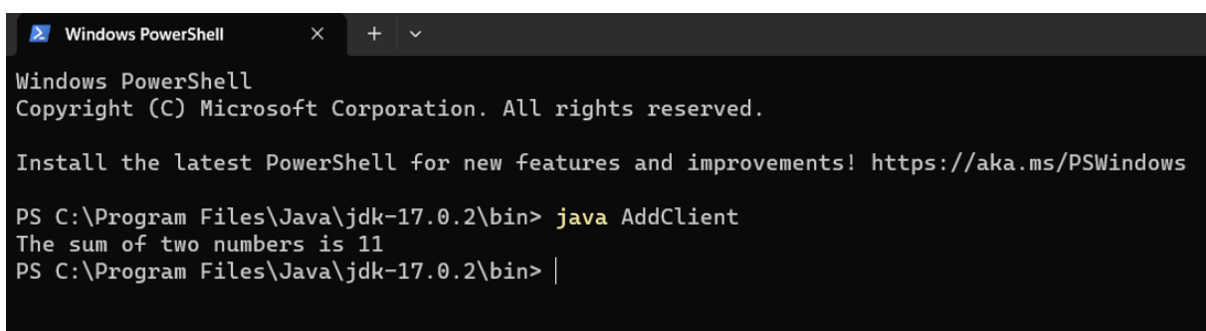


```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Program Files\Java\jdk-17.0.2\bin> java AddServer
Server is connected and waiting for the client
|
```

- Execute AddClient.java



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Program Files\Java\jdk-17.0.2\bin> java AddClient
The sum of two numbers is 11
PS C:\Program Files\Java\jdk-17.0.2\bin> |
```

**NAME: SHASHWAT SHAH**  
**SAP ID: 60004220126**  
**DIV/BATCH: C22**

**DISTRIBUTED COMPUTING (DC)**  
**EXPERIMENT 02**

**AIM: To implement Multithreading application using JAVA.**

**CODE:**

```
class MyThread extends Thread {
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println(Thread.currentThread().getName() + " - Value: " + i);
            try {
                Thread.sleep(500); // Sleep for 500 milliseconds
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }
}

public static void main(String[] args) {
    MyThread t1 = new MyThread();
    MyThread t2 = new MyThread();

    t1.start(); // Starts thread 1
    t2.start(); // Starts thread 2
}
```

```
class MyRunnable implements Runnable {
    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println(Thread.currentThread().getName() + " - Value: " + i);
            try {
                Thread.sleep(500); // Sleep for 500 milliseconds
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }
}

public static void main(String[] args) {
    Thread t1 = new Thread(new MyRunnable());
    Thread t2 = new Thread(new MyRunnable());

    t1.start(); // Starts thread 1
    t2.start(); // Starts thread 2
}
```

OUTPUT:

```
Thread-1 - Value: 0
Thread-0 - Value: 0
Thread-0 - Value: 1
Thread-1 - Value: 1
Thread-1 - Value: 2
Thread-0 - Value: 2
Thread-0 - Value: 3
Thread-1 - Value: 3
Thread-0 - Value: 4
Thread-1 - Value: 4
```

**NAME: SHASHWAT SHAH**  
**SAP ID: 60004220126**  
**DIV/BATCH: C22**

**DISTRIBUTED COMPUTING (DC)**  
**EXPERIMENT 03**

**AIM: To implement Inter-process communication using TCP based in Socket Programming.**

**CODE:**

**TCPServer.Java**

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

public class TCPServer {
    public static void main(String[] args) throws Exception {
        // Creating ServerSocket at port 25
        ServerSocket server = new ServerSocket(25);
        System.out.println("Connecting...");

        // Accepting client connection
        Socket ss = server.accept();
        System.out.println("Connected");

        // Creating input and output streams
        DataInputStream din = new DataInputStream(ss.getInputStream());
        DataOutputStream dout = new DataOutputStream(ss.getOutputStream());

        String str = "";
        int sum = 0;
        System.out.println("Receiving integers from client...");

        // Receiving integers from client and calculating sum
        while (true) {
            str = din.readUTF();
            if (str.equals("stop")) {
                System.out.println("Sum of integers received from client: " + sum);
                dout.writeUTF("" + sum);
                dout.flush();
                break;
            }
            sum += Integer.parseInt(str);
        }

        // Closing resources
        din.close();
        dout.close();
        ss.close();
        server.close();
    }
}
```

### **TCPClient.Java**

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

public class TCPClient {
    public static void main(String[] args) throws Exception {
        System.out.println("Connecting...");

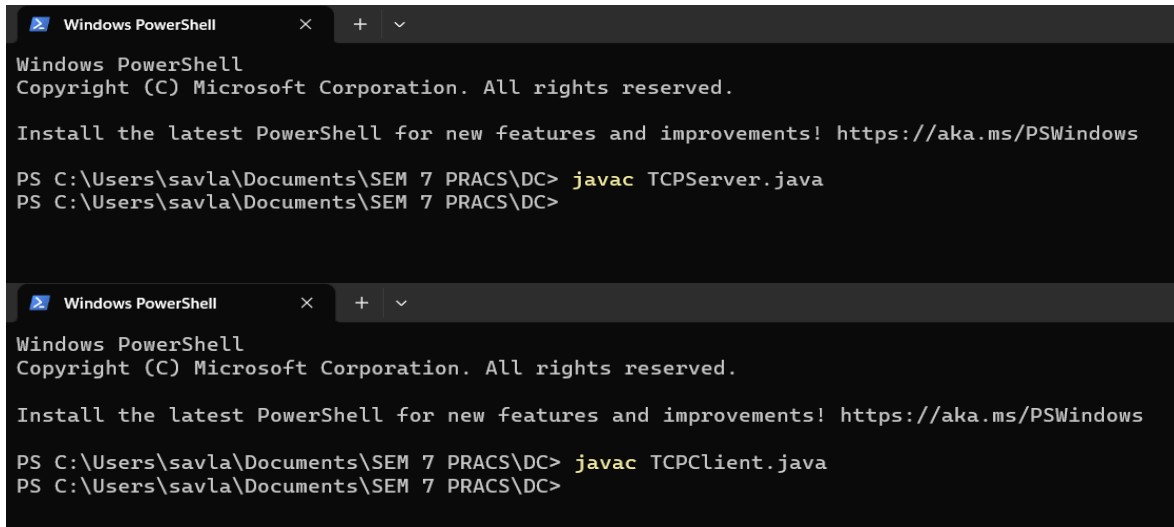
        // Connecting to the server at localhost on port 25
        Socket s = new Socket("localhost", 25);
        System.out.println("Connected");

        // Creating input and output streams
        DataInputStream din = new DataInputStream(s.getInputStream());
        DataOutputStream dout = new DataOutputStream(s.getOutputStream());
        Scanner sc = new Scanner(System.in);
        String str = "";
        // Sending integers to server until "stop" is entered
        while (true) {
            str = sc.nextLine();
            dout.writeUTF(str);
            dout.flush();

            if (str.equals("stop")) {
                System.out.println("Sum of integers received from server: " + din.readUTF());
                break;
            }
        }
        // Closing resources
        din.close();
        dout.close();
        s.close();
        sc.close();
    }
}
```

## OUTPUT:

- Compile TCPServer.java & TCPClient.java



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\savla\Documents\SEM 7 PRACS\DC> javac TCPServer.java
PS C:\Users\savla\Documents\SEM 7 PRACS\DC>

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\savla\Documents\SEM 7 PRACS\DC> javac TCPClient.java
PS C:\Users\savla\Documents\SEM 7 PRACS\DC>
```

- Execute TCPServer

```
Connecting...
Connected
Receiving integers from client...
Sum of integers received from client: 15
```

- Execute TCPClient

```
Connecting...
Connected
1
2
3
4
5
stop
Sum of integers received from server: 15
```

**NAME: SHASHWAT SHAH**  
**SAP ID: 60004220126**  
**DIV/BATCH: C22**

**DISTRIBUTED COMPUTING (DC)**  
**EXPERIMENT 04**

**AIM: Implement group communication using JAVA.**

**CODE:**

**GCServer.java**

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

public class GCServer {
    // List to store client handlers
    static ArrayList<ClientHandler> clients = new ArrayList<ClientHandler>();

    public static void main(String[] args) throws Exception {
        // Server listens on port 25
        ServerSocket server = new ServerSocket(25);
        Message msg = new Message();
        int count = 0;

        // Accepting clients continuously
        while (true) {
            Socket ss = server.accept();
            DataInputStream din = new DataInputStream(ss.getInputStream());
            DataOutputStream dout = new DataOutputStream(ss.getOutputStream());

            // Creating a new client handler thread
            ClientHandler chlr = new ClientHandler(ss, din, dout, msg);
            clients.add(chlr);
            chlr.start();
            count++;
        }
    }

    // Class to handle message broadcasting
    class Message {
        String msg;

        public void setMsg(String msg) {
            this.msg = msg;
        }

        public void getMsg() {
            System.out.println("\nNEW GROUP MESSAGE: " + this.msg);

            // Broadcasting message to all connected clients
        }
    }
}
```



```

for (int i = 0; i < GCServer.clients.size(); i++) {
    try {
        System.out.println("Client: " + GCServer.clients.get(i).ip + " ");
        GCServer.clients.get(i).out.writeUTF(this.msg);
        GCServer.clients.get(i).out.flush();
    } catch (Exception e) {
        System.out.println(e);
    }
}
}
}

```

// Class to handle individual client connections

class ClientHandler extends Thread {

```

    DataInputStream in;
    DataOutputStream out;
    Socket socket;
    boolean conn;
    Message msg;
    String ip;

```

```

    public ClientHandler(Socket s, DataInputStream din, DataOutputStream dout, Message msg) {
        this.socket = s;
        this.in = din;
        this.out = dout;
        this.msg = msg;
        this.conn = true;
        this.ip = (((InetSocketAddress) s.getRemoteSocketAddress()).getAddress()).toString().replace("/", "");
    }

```

```

    public void run() {
        while (conn) {
            try {
                // Reading message from client and broadcasting
                String input = this.in.readUTF();
                this.msg.setMsg(input);
                this.msg.getMsg();
            } catch (Exception e) {
                conn = false;
                System.out.println(e);
            }
        }
        closeConn();
    }

```

// Method to close client connection

```

    public void closeConn() {
        try {
            this.in.close();
            this.out.close();

```

```

        this.socket.close();
    } catch (Exception e) {
        System.out.println(e);
    }
}
}

```

### **GCMaster.Java**

```

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

public class GCMaster {
    public static void main(String[] args) throws Exception {
        // Connecting to the server on localhost at port 25
        Socket client = new Socket("127.0.0.1", 25);
        DataInputStream din = new DataInputStream(client.getInputStream());
        DataOutputStream dout = new DataOutputStream(client.getOutputStream());

        System.out.println("Connected as Master");
        Scanner sc = new Scanner(System.in);
        String send = "";

        // Sending messages to server until "stop" is entered
        do {
            System.out.print("Enter message: ");
            send = sc.nextLine();
            dout.writeUTF(send);
            dout.flush();
        } while (!send.equals("stop"));

        // Closing connections
        dout.close();
        din.close();
        client.close();
        sc.close();
    }
}

```

### **GCSlave.Java**

```

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

public class GCSlave {
    public static void main(String[] args) throws Exception {
        // Connecting to the server on localhost at port 25
        Socket client = new Socket("127.0.0.1", 25);
        DataInputStream din = new DataInputStream(client.getInputStream());
    }
}

```

```

System.out.println("Connected as Slave");
String recv = "";

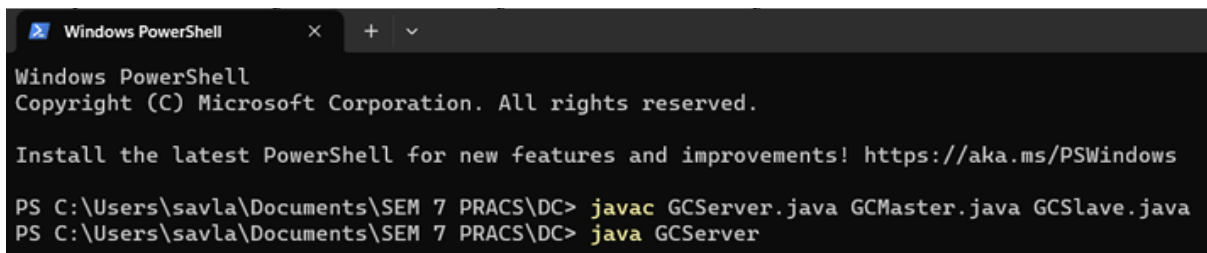
// Receiving messages from server until "stop" is received
do {
    recv = din.readUTF();
    System.out.println("Master Message: " + recv);
} while (!recv.equals("stop"));

// Closing connection
din.close();
client.close();
}
}

```

#### OUTPUT:

- Compile GCServer.java, GCMaster.java and GCSlave.java



```

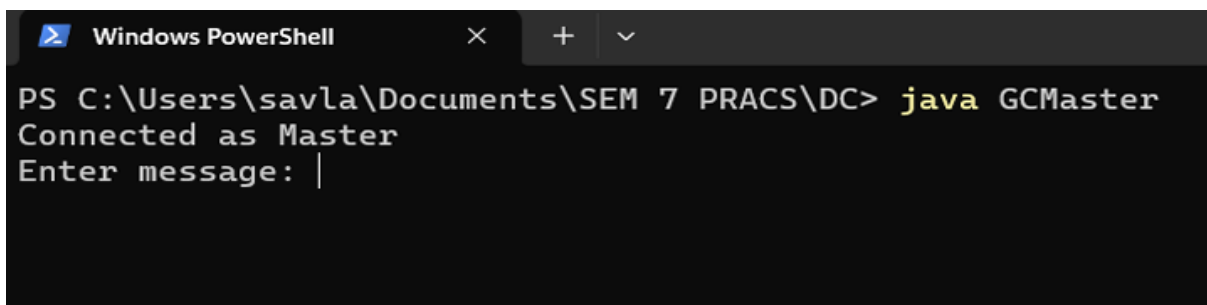
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\savla\Documents\SEM 7 PRACS\DC> javac GCServer.java GCMaster.java GCSlave.java
PS C:\Users\savla\Documents\SEM 7 PRACS\DC> java GCServer

```

- Execute GCMaster.java



```

Windows PowerShell
PS C:\Users\savla\Documents\SEM 7 PRACS\DC> java GCMaster
Connected as Master
Enter message: |

```

- Execute GCSlave.java in multiple command prompts

```
Windows PowerShell
PS C:\Users\savla\Documents\SEM 7 PRACS\DC> java GCSlave
Connected as Slave

Windows PowerShell
PS C:\Users\savla\Documents\SEM 7 PRACS\DC> java GCSlave
Connected as Slave
```

- Type Message in GCMaster.java

```
Windows PowerShell
PS C:\Users\savla\Documents\SEM 7 PRACS\DC> java GCMaster
Connected as Master
Enter message: I solemnly swear to god that I am upto no good
Enter message: Mischief Managed
Enter message: stop
PS C:\Users\savla\Documents\SEM 7 PRACS\DC>

Windows PowerShell
NEW GROUP MESSAGE: I solemnly swear to god that I am upto no good
Client: 127.0.0.1;
Client: 127.0.0.1;
Client: 127.0.0.1;

NEW GROUP MESSAGE: Mischief Managed
Client: 127.0.0.1;
Client: 127.0.0.1;
Client: 127.0.0.1;

NEW GROUP MESSAGE: stop
Client: 127.0.0.1;
Client: 127.0.0.1;
Client: 127.0.0.1;
java.io.EOFException
java.io.EOFException
java.io.EOFException

Windows PowerShell
PS C:\Users\savla\Documents\SEM 7 PRACS\DC> java GCSlave
Connected as Slave
Master Message: I solemnly swear to god that I am upto no good
Master Message: Mischief Managed
Master Message: stop
PS C:\Users\savla\Documents\SEM 7 PRACS\DC>
```

<pre>Windows PowerShell NEW GROUP MESSAGE: I solemnly swear to god that I an upto no good Client: 127.0.0.1; Client: 127.0.0.1; Client: 127.0.0.1;  NEW GROUP MESSAGE: Mischief Managed Client: 127.0.0.1; Client: 127.0.0.1; Client: 127.0.0.1;  NEW GROUP MESSAGE: stop Client: 127.0.0.1; Client: 127.0.0.1; Client: 127.0.0.1; java.io.EOFException java.io.EOFException java.io.EOFException</pre>	<pre>Windows PowerShell PS C:\Users\savla\Documents\SEM 7 PRACS\DC&gt; java GCSlave Connected as Slave Master Message: I solemnly swear to god that I an upto no good Master Message: Mischief Managed Master Message: stop PS C:\Users\savla\Documents\SEM 7 PRACS\DC&gt;</pre>
<pre>Windows PowerShell PS C:\Users\savla\Documents\SEM 7 PRACS\DC&gt; java GCMaster Connected as Master Enter message: I solemnly swear to god that I am upto no good Enter message: Mischief Managed Enter message: stop PS C:\Users\savla\Documents\SEM 7 PRACS\DC&gt;</pre>	<pre>Windows PowerShell PS C:\Users\savla\Documents\SEM 7 PRACS\DC&gt; java GCSlave Connected as Slave Master Message: I solemnly swear to god that I an upto no good Master Message: Mischief Managed Master Message: stop PS C:\Users\savla\Documents\SEM 7 PRACS\DC&gt;</pre>

**NAME: SHASHWAT SHAH**  
**SAP ID: 60004220126**  
**DIV/BATCH: C22**

**DISTRIBUTED COMPUTING (DC)**  
**EXPERIMENT 05**

**AIM: To implement Load-Balancing using JAVA.**

**CODE:**

**LoadBalance.java**

```
import java.util.Scanner;

public class LoadBalance {

    // Method to print the load distribution across servers
    static void printLoad(int server, int processes) {
        int each = processes / server; // Processes per server
        int extra = processes % server; // Extra processes to be distributed

        // Distribute extra processes first
        for (int i = 0; i < extra; i++) {
            System.out.println("Server " + (i + 1) + " has " + (each + 1) + " processes");
        }

        // Distribute remaining processes
        for (int i = extra; i < server; i++) {
            System.out.println("Server " + (i + 1) + " has " + each + " processes");
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // Input number of servers and processes
        System.out.print("Enter the number of servers: ");
        int servers = sc.nextInt();
        System.out.print("Enter the number of processes: ");
        int processes = sc.nextInt();

        // Infinite loop for the load balancing options
        while (true) {
            // Display current load distribution
            printLoad(servers, processes);

            // Menu options for modifying servers and processes
            System.out.println("\n1. Add Servers");
            System.out.println("2. Remove Servers");
            System.out.println("3. Add Processes");
            System.out.println("4. Remove Processes");
            System.out.println("5. Exit");
            System.out.print("> ");
        }
    }
}
```

```

// Handle user input based on choice
switch (sc.nextInt()) {
    case 1:
        System.out.print("Enter the number of servers to add: ");
        servers += sc.nextInt();
        break;

    case 2:
        System.out.print("Enter the number of servers to remove: ");
        servers -= sc.nextInt();
        if (servers < 1) {
            servers = 1; // Ensure at least one server is present
            System.out.println("At least one server is required.");
        }
        break;

    case 3:
        System.out.print("Enter the number of processes to add: ");
        processes += sc.nextInt();
        break;

    case 4:
        System.out.print("Enter the number of processes to remove: ");
        processes -= sc.nextInt();
        if (processes < 0) {
            processes = 0; // Ensure non-negative process count
            System.out.println("Number of processes cannot be negative.");
        }
        break;

    case 5:
        System.out.println("Exiting...");
        sc.close();
        System.exit(0);

    default:
        System.out.println("Invalid choice. Please select a valid option.");
}
}
}
}

```

## OUTPUT:

- Compile LoadBalance.java

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\savla\Documents\SEM 7 PRACS\DC> javac LoadBalance.java
PS C:\Users\savla\Documents\SEM 7 PRACS\DC> |
```

- Execute LoadBalance.java

```
Windows PowerShell
PS C:\Users\savla\Documents\SEM 7 PRACS\DC> javac LoadBalance.java
PS C:\Users\savla\Documents\SEM 7 PRACS\DC> java LoadBalance
Enter the number of servers:
4
Enter the number of processes:
17
Server 1 has 5 processes
Server 2 has 4 processes
Server 3 has 4 processes
Server 4 has 4 processes

1.Add Servers 2.Remove Servers 3.Add Processes 4.Remove Processes 5.Exit
> 3
Enter the number of processes to add:
3
Server 1 has 5 processes
Server 2 has 5 processes
Server 3 has 5 processes
Server 4 has 5 processes

1.Add Servers 2.Remove Servers 3.Add Processes 4.Remove Processes 5.Exit
> 4
Enter the number of processes to remove:
7
Server 1 has 4 processes
Server 2 has 3 processes
Server 3 has 3 processes
Server 4 has 3 processes

1.Add Servers 2.Remove Servers 3.Add Processes 4.Remove Processes 5.Exit
> 1
Enter the number of servers to add:
1
Server 1 has 3 processes
Server 2 has 3 processes
Server 3 has 3 processes
Server 4 has 2 processes
Server 5 has 2 processes

1.Add Servers 2.Remove Servers 3.Add Processes 4.Remove Processes 5.Exit
> 5
```



**NAME: SHASHWAT SHAH**  
**SAP ID: 60004220126**  
**DIV/BATCH: C22**

**DISTRIBUTED COMPUTING (DC)**  
**EXPERIMENT 06**

**AIM: To implement Name Resolution Protocol using JAVA.**

**CODE:**

**DNS\_Server.java**

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

public class DNS_Server {
    public static void main(String[] args) {
        try {
            // Create a socket to listen on port 9876
            DatagramSocket serverSocket = new DatagramSocket(9876);

            // DNS records: Domain name -> IP address mapping
            HashMap<String, String> dnsRecords = new HashMap<>();
            dnsRecords.put("google.com", "142.250.182.78");
            dnsRecords.put("yahoo.com", "98.138.219.231");
            dnsRecords.put("example.com", "93.184.216.34");
            dnsRecords.put("facebook.com", "157.240.229.35");

            byte[] receiveBuffer = new byte[1024];
            byte[] sendBuffer;

            System.out.println("DNS Server is running...");

            while (true) {
                // Receive request from client
                DatagramPacket receivePacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);
                serverSocket.receive(receivePacket);

                String domainName = new String(receivePacket.getData(), 0, receivePacket.getLength());
                System.out.println("Request received for domain: " + domainName);

                // Get IP address from DNS records
                String ipAddress = dnsRecords.getOrDefault(domainName, "Domain not found");

                // Send response to client
                sendBuffer = ipAddress.getBytes();
                InetAddress clientAddress = receivePacket.getAddress();
                int clientPort = receivePacket.getPort();

                DatagramPacket sendPacket = new DatagramPacket(sendBuffer, sendBuffer.length,
                    clientAddress, clientPort);
                serverSocket.send(sendPacket);
                serverSocket.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```

    }
} catch (Exception e) {
    System.out.println("Error: " + e.getMessage());
}
}
}
}

```

### **DNS\_Client.java**

```

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

public class DNS_Client {
    public static void main(String[] args) {
        try {
            // Create a socket to send and receive data
            DatagramSocket clientSocket = new DatagramSocket();

            InetAddress serverAddress = InetAddress.getByName("localhost");

            Scanner scanner = new Scanner(System.in);

            byte[] sendBuffer;

            byte[] receiveBuffer = new byte[1024];

            System.out.print("Enter domain name to resolve: ");
            String domainName = scanner.nextLine();

            // Send domain name to server
            sendBuffer = domainName.getBytes();

            DatagramPacket sendPacket = new DatagramPacket(sendBuffer, sendBuffer.length,
serverAddress, 9876);

            clientSocket.send(sendPacket);

            // Receive resolved IP address from server
            DatagramPacket receivePacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);
            clientSocket.receive(receivePacket);

```

```
String ipAddress = new String(receivePacket.getData(), 0, receivePacket.getLength());  
System.out.println("Resolved IP Address: " + ipAddress);  
  
// Close resources  
clientSocket.close();  
scanner.close();  
} catch (Exception e) {  
    System.out.println("Error: " + e.getMessage());  
}  
}  
}
```

**OUTPUT:**

```
Enter domain name to resolve: google.com  
Resolved IP Address: 142.250.182.78
```



```

        System.out.println("Invalid process number.");
    }
    break;

case 2: // Recover a process
    System.out.print("Enter the process number to recover: ");
    c = sc.nextInt();
    if (c <= n && c > 0) {
        sta[c - 1] = 1; // Mark process as active
        System.out.println("Process " + c + " has recovered.");
        cl = 1;
    } else {
        System.out.println("Invalid process number.");
    }
    break;

case 3: // Exit the program
    choice = false;
    cl = 0;
    break;

default:
    System.out.println("Invalid choice. Please try again.");
}

// If a process crashed or recovered, initiate election
if (cl == 1) {
    System.out.print("Which process will initiate election? ");
    int ele = sc.nextInt();
    if (ele <= n && ele > 0 && sta[ele - 1] == 1) {
        elect(ele);
        System.out.println("Final coordinator is Process " + co);
    } else {
        System.out.println("Invalid or inactive process cannot initiate an election.");
    }
}
} while (choice);

sc.close();
}

// Election method based on Bully Algorithm
static void elect(int ele) {
    ele = ele - 1; // Adjust for zero-based indexing
    co = ele + 1; // Assume initiator as coordinator

    // Send election messages to higher-numbered processes
    for (int i = 0; i < n; i++) {
        if (pro[ele] < pro[i]) {
            System.out.println("Election message is sent from Process " + (ele + 1) + " to Process " + (i +
1));

```

**OUTPUT:**

```

PROBLEMS 10 OUTPUT TERMINAL DEBUG CONSOLE PORTS
C:\Users\savla\Documents\SEM 7 PRACS\DC> cd "c:\Users\savla\Documents\SEM 7 PRACS\DC\" ; if ($?) { javac Bully.java } ; if ($?) { java Bully }
Enter the number of process:
5
Enter Your Choice
1. Crash Process
2. Recover Process
3. Exit
> 1
Enter the process number: 1
Which process will initiate election? = 2
Election message is sent from 2 to 3
Ok message is sent from 3 to 2
Election message is sent from 3 to 4
Ok message is sent from 4 to 3
Election message is sent from 4 to 5
Ok message is sent from 5 to 4
Election message is sent from 3 to 5
Ok message is sent from 5 to 3
Election message is sent from 2 to 4
Ok message is sent from 4 to 2
Election message is sent from 4 to 5
Ok message is sent from 5 to 4
Election message is sent from 2 to 5
Ok message is sent from 5 to 2
Final coordinator is 5
Enter Your Choice
1. Crash Process
2. Recover Process
3. Exit
> 1
Enter the process number: 5
Which process will initiate election? = 3
Election message is sent from 3 to 4
Ok message is sent from 4 to 3
Election message is sent from 4 to 5
Election message is sent from 3 to 5
Final coordinator is 4
Enter Your Choice
1. Crash Process
2. Recover Process
3. Exit
> 2
> 2
Enter the process number: 1
Which process will initiate election? = 3
Election message is sent from 3 to 4
Ok message is sent from 4 to 3
Election message is sent from 4 to 5
Election message is sent from 3 to 5
Final coordinator is 4
Enter Your Choice
1. Crash Process
2. Recover Process
3. Exit
> 1

```

**NAME: SHASHWAT SHAH**  
**SAP ID: 60004220126**  
**DIV/BATCH: C22**

**DISTRIBUTED COMPUTING (DC)**  
**EXPERIMENT 08**

**AIM: To demonstrate clock synchronization algorithm using JAVA.**

**CODE:**

```
import java.util.HashMap;
import java.util.Map;
import java.util.Random;

public class BerkeleyClockSync {

    // Function to adjust time based on average difference
    public static double adjustTime(double timeDiff, double nodeTime) {
        return nodeTime + timeDiff;
    }

    // Berkeley Clock Synchronization function
    public static Map<String, Double> berkeleyAlgorithm(Map<String, Double> nodeTimes) {
        // Randomly select a master node
        Random rand = new Random();
        Object[] nodes = nodeTimes.keySet().toArray();
        String masterNode = (String) nodes[rand.nextInt(nodes.length)];

        System.out.println("Master node is: " + masterNode);

        // Step 1: Master node calculates time differences
        Map<String, Double> timeDiffs = new HashMap<>();
        double masterTime = nodeTimes.get(masterNode);

        for (Map.Entry<String, Double> entry : nodeTimes.entrySet()) {
            String node = entry.getKey();
            double time = entry.getValue();
            if (!node.equals(masterNode)) {
                double timeDiff = time - masterTime;
                timeDiffs.put(node, timeDiff);
                System.out.println("Node " + node + " has a time difference of " + timeDiff + " with master.");
            }
        }

        // Step 2: Calculate average time difference
        double avgDiff = timeDiffs.values().stream().mapToDouble(Double::doubleValue).sum() /
            timeDiffs.size();
        System.out.println("Average time difference: " + avgDiff);

        // Step 3: Adjust time for all nodes
        for (String node : nodeTimes.keySet()) {
```

```

        if (!node.equals(masterNode)) {
            nodeTimes.put(node, adjustTime(-avgDiff, nodeTimes.get(node)));
        } else {
            // Master node adjusts its own time
            nodeTimes.put(node, masterTime + avgDiff);
        }
    }

    return nodeTimes;
}

public static void main(String[] args) {
    // Sample node times in seconds
    Map<String, Double> nodeTimes = new HashMap<>();
    nodeTimes.put("Node 1", 100.0);
    nodeTimes.put("Node 2", 150.0);
    nodeTimes.put("Node 3", 130.0);
    nodeTimes.put("Node 4", 120.0);

    System.out.println("Initial times: " + nodeTimes);

    // Synchronize using Berkeley Algorithm
    Map<String, Double> synchronizedTimes = berkeleyAlgorithm(nodeTimes);

    System.out.println("Synchronized times: " + synchronizedTimes);
}
}

```

## OUTPUT:

```

Initial times: {Node 3=130.0, Node 2=150.0, Node 4=120.0, Node 1=100.0}
Master node is: Node 1
Node Node 3 has a time difference of 30.0 with master.
Node Node 2 has a time difference of 50.0 with master.
Node Node 4 has a time difference of 20.0 with master.
Average time difference: 33.333333333333336
Synchronized times: {Node 3=96.66666666666666, Node 2=116.66666666666666, Node 4=86.66666666666666, Node 1=133.33333333333334}

```



**NAME: SHASHWAT SHAH**

**SAP ID: 60004220126**

**DIV/BATCH: C22**

**DISTRIBUTED COMPUTING (DC)  
EXPERIMENT 09**

**AIM: To demonstrate mutual exclusion algorithm using JAVA.**

**CODE:**

**MutualServer.java**

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

public class MutualServer implements Runnable {
    Socket socket = null;
    static ServerSocket ss;

    // Constructor to initialize socket
    MutualServer(Socket newSocket) {
        this.socket = newSocket;
    }

    public static void main(String[] args) throws IOException {
        ss = new ServerSocket(7000);
        System.out.println("Server Started...");

        while (true) {
            Socket s = ss.accept();
            System.out.println("New client connected...");
            MutualServer es = new MutualServer(s);
            Thread t = new Thread(es);
            t.start();
        }
    }

    // Runnable method to read data from client
    public void run() {
        try {
            BufferedReader br = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            while (true) {
                String str = br.readLine();
                if (str != null) {
                    System.out.println("Client: " + str);
                }
            }
        } catch (Exception e) {
            System.out.println("Exception: " + e);
        }
    }
}
```

### **ClientOne.Java**

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

public class ClientOne {
    public static void main(String[] args) throws IOException {
        // Connect to MutualServer on port 7000
        Socket s = new Socket("localhost", 7000);
        PrintStream out = new PrintStream(s.getOutputStream());

        // Create a server socket to receive the token from ClientTwo
        ServerSocket ss = new ServerSocket(7001);
        Socket s1 = ss.accept();
        BufferedReader in1 = new BufferedReader(new InputStreamReader(s1.getInputStream()));
        PrintStream out1 = new PrintStream(s1.getOutputStream());
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        String str = "Token";
        while (true) {
            if (str.equalsIgnoreCase("Token")) {
                System.out.println("Do you want to send some data?");
                System.out.println("Enter Yes or No");
                str = br.readLine();

                if (str.equalsIgnoreCase("Yes")) {
                    System.out.println("Enter the data");
                    str = br.readLine();
                    out.println(str); // Send data to MutualServer
                }

                // Pass the token to ClientTwo
                out1.println("Token");
            }

            System.out.println("Waiting for Token");
            str = in1.readLine();
        }
    }
}
```

### **ClientTwo.Java**

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

public class ClientTwo {
    public static void main(String[] args) throws IOException {
        // Connect to MutualServer on port 7000
        Socket s = new Socket("localhost", 7000);
        PrintStream out = new PrintStream(s.getOutputStream());
```

```
// Connect to ClientOne's server socket on port 7001 to receive and send the token
Socket s2 = new Socket("localhost", 7001);
BufferedReader in2 = new BufferedReader(new InputStreamReader(s2.getInputStream()));
PrintStream out2 = new PrintStream(s2.getOutputStream());
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String str = "Token";
while (true) {
    System.out.println("Waiting for Token");
    str = in2.readLine(); // Wait for the token from ClientOne

    if (str.equalsIgnoreCase("Token")) {
        System.out.println("Do you want to send some data?");
        System.out.println("Enter Yes or No");
        str = br.readLine();

        if (str.equalsIgnoreCase("Yes")) {
            System.out.println("Enter the data");
            str = br.readLine();
            out.println(str); // Send data to MutualServer
        }

        // Pass the token back to ClientOne
        out2.println("Token");
    }
}
}
```

#### OUTPUT:

- **Compile MutualServer.java, ClientOne.java and ClientTwo.java**

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\savla\Documents\SEM 7 PRACS\DC> javac .\MutualServer.java
PS C:\Users\savla\Documents\SEM 7 PRACS\DC> javac ClientOne.java
PS C:\Users\savla\Documents\SEM 7 PRACS\DC> javac ClientTwo.java
PS C:\Users\savla\Documents\SEM 7 PRACS\DC> |
```

- **Execute MutualServer.java**

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\savla\Documents\SEM 7 PRACS\DC> java MutualServer.java
Server Started...
New client connected...
```

- Open a new command prompt and execute ClientOne.java. Keep it running till ClientTwo starts

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\savla\Documents\SEM 7 PRACS\DC> java ClientOne
```

- Open another command prompt and execute ClientTwo.java. The output allows both the clients to use tokens and share their messages with each other using Token Ring concept. To send the message, the client has to accept the token by typing Yes followed by the message alternately and has to type No to release the token

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\savla\Documents\SEM 7 PRACS\DC> javac ClientTwo.java
PS C:\Users\savla\Documents\SEM 7 PRACS\DC> java ClientTwo
Waiting for Token
```

<pre>Windows PowerShell PS C:\Users\savla\Documents\SEM 7 PRACS\DC&gt; java MutualServer.java Server Started... New client connected... New client connected... Client: Distributed Computing Client: Parallel Computing  </pre>	<pre>Windows PowerShell PS C:\Users\savla\Documents\SEM 7 PRACS\DC&gt; java ClientOne Do you want to send some data? Enter Yes or No Yes Enter the data Distributed Computing Waiting for Token Do you want to send some data? Enter Yes or No Y</pre>
	<pre>Windows PowerShell PS C:\Users\savla\Documents\SEM 7 PRACS\DC&gt; java ClientTwo Waiting for Token Do you want to send some data? Enter Yes or No Yes Enter the data Parallel Computing Waiting for Token</pre>

NAME: SHASHWAT SHAH

SAP ID: 60004220126

DIV/BATCH: C22

Subject :Distributed Computing

**Aim:** To perform a case study on based on Edge, Cloud and Fog Computing.

## **Introduction and Background**

The paper by Ahmad et al. discusses the evolution and integration of edge, cloud, and fog computing within the context of growing IoT applications. As these applications require real-time data processing and analysis, conventional cloud infrastructures alone fall short in meeting such demands, primarily due to latency issues and data transfer bottlenecks. The emergence of edge and fog computing addresses these challenges by distributing computational tasks closer to data sources.

## **Core Objectives of the Study**

The paper's primary aim was to explore how distributed frameworks involving edge and fog computing can bolster the effectiveness of cloud systems. Key objectives included:

**Assessing Latency Improvements:** Measuring how edge and fog computing impact response times compared to cloud-only systems.

**Optimizing Resource Utilization:** Examining methods for efficient task distribution across edge, fog, and cloud nodes.

**Security Analysis:** Investigating potential vulnerabilities in a decentralized framework and proposing mitigation strategies.

## **Research Methodology**

The research was conducted through a combination of theoretical modeling and simulation-based evaluations:

**Simulation Environment:** Simulated IoT data flows and analyzed network performance metrics under different configurations (edge-only, fog-only, combined edge-fog-cloud).

Performance Metrics: Key performance indicators included processing latency, bandwidth usage, data transfer rates, and computational load distribution.

## **Key Findings**

1. **Significant Latency Reduction:** The deployment of edge nodes drastically minimized latency by processing data close to its source. For instance, in IoT-based real-time applications, such as emergency response systems, latency was reduced by 40-60% compared to cloud-only architectures.
2. **Improved Resource Efficiency:** Integrating fog computing facilitated local aggregation and preliminary data processing, allowing only relevant, processed information to be sent to the cloud. This led to a reduction in overall bandwidth usage and cloud server load.
3. **Scalability and Fault Tolerance:** Edge and fog computing provided scalable frameworks adaptable to high-traffic scenarios. By distributing tasks among various nodes, the system also exhibited improved fault tolerance. In cases where an edge node failed, fog nodes could temporarily assume its role, maintaining service continuity.

## **Detailed Application Example: Smart City Infrastructure**

The paper illustrated a real-world scenario involving smart city deployments. Sensors embedded across a city collected data on traffic, air quality, and energy consumption. The research demonstrated:

**Edge Nodes for Immediate Action:** Traffic sensors communicated directly with edge computing units for real-time congestion control. Decisions like changing traffic signals or redirecting vehicles were made locally, ensuring immediate response.

**Fog Computing for Intermediate Processing:** Energy consumption data was aggregated by fog nodes for short-term analytics, such as peak usage detection and dynamic load balancing.

**Cloud for Comprehensive Analysis:** The cloud stored long-term data and ran predictive models to forecast future energy needs and analyze historical trends in air quality.

Results: This multi-layered approach improved real-time traffic management efficiency by 30% and reduced peak energy demands by optimizing load distribution across the city.

## **Challenges and Solutions**

### **1. Complex Coordination:**

Challenge: Managing synchronized operations between the edge, fog, and cloud layers is complex, especially during data surges.

Solution Proposed: The study suggested developing intelligent orchestration mechanisms powered by machine learning to dynamically allocate resources based on current network demand.

### **2. Security Concerns:**

Challenge: Decentralized data processing introduces additional points of vulnerability.

Solution Proposed: Enhancing encryption methods and incorporating zero-trust architecture within edge and fog nodes to safeguard data at every stage.

### **3. Middleware Development:**

Challenge: Communication between computing layers required robust middleware solutions.

Solution Proposed: The paper highlighted the potential of developing modular middleware platforms capable of facilitating seamless data exchange and maintaining consistent performance across the entire system.

## **Recommendations for Future Research**

The authors stressed the need for:



AI-Enhanced Orchestration: Leveraging machine learning for proactive resource allocation and task scheduling.

Cross-Layer Security Frameworks: Establishing a unified security model covering all layers (edge, fog, cloud).

Standardized Middleware Solutions: Developing industry standards for middleware that supports diverse IoT applications without customization.

## **Conclusion**

This paper underscored the transformative potential of integrating edge and fog computing with cloud infrastructure to meet the real-time and low-latency requirements of modern IoT applications. The tiered approach presented benefits such as enhanced responsiveness, better resource management, and scalability. However, the challenges related to coordination, security, and infrastructure complexity point to opportunities for future exploration and technological advancements.

This detailed study provides a comprehensive understanding of how combined edge, fog, and cloud computing can revolutionize IoT and other data-driven fields, emphasizing a pathway for future research and development in distributed computing.