```
Javac filename.java
Java filename
```

BULLY ALGORITHM : (ELECTION ALGORITHM)

```
class Process {
    int id;
    boolean active;

    public Process(int id) {
        this.id = id;
        this.active = true;
    }
}

public class BullyAlgorithm {
    Process[] processes;
    int coordinator;

    public BullyAlgorithm(int n) {
        processes = new Process[n];
        for (int i = 0; i < n; i++) {
            processes[i] = new Process(i);
        }
        coordinator = n - 1; // Assume highest ID process is initially coordinator
    }

    public void failProcess(int id) {
        if (id < processes.length) {
            processes[id].active = false;
            System.out.println("Process " + id + " has failed.");
            if (id == coordinator) {
                startElection();
            }
        }
    }

    public void startElection() {
        System.out.println("Starting Election...");
        int newCoordinator = -1;
        for (int i = processes.length - 1; i >= 0; i--) {
            if (processes[i].active) {
                newCoordinator = i;
                break;
            }
        }

        if (newCoordinator != -1) {
            coordinator = newCoordinator;
            System.out.println("Process " + coordinator + " is elected as the new coordinator.");
        } else {
            System.out.println("No active processes available to elect a coordinator.");
        }
    }

    public void recoverProcess(int id) {
        if (id < processes.length) {
            processes[id].active = true;
            System.out.println("Process " + id + " has recovered.");
            startElection();
        }
    }

    public static void main(String[] args) {
        BullyAlgorithm ba = new BullyAlgorithm(5);
        System.out.println("Initial Coordinator: Process " + ba.coordinator);

        ba.failProcess(4);
        ba.failProcess(3);
        ba.failProcess(2);
        ba.recoverProcess(3);
        ba.recoverProcess(0);
    }
}
```

LAMPORT LOGICAL CLOCK

```
import java.util.Random;

class Process extends Thread {

    private int processId;
    private int logicalClock;
    private Process[] processes;

    public Process(int processId, Process[] processes) {
        this.processId = processId;
        this.processes = processes;
        this.logicalClock = 0;
    }

    private void internalEvent() {
        logicalClock++;
        System.out.println("Process " + processId + " executed an internal event. Timestamp: " +
logicalClock);
    }

    private void sendMessage(int receiverId) {
        logicalClock++;
        System.out.println("Process " + processId + " sent a message to  " + receiverId + " with
timestamp: " + logicalClock);
        processes[receiverId].receiveMessage(logicalClock, processId);
    }

    public synchronized void receiveMessage(int senderClock, int senderId) {
        logicalClock = Math.max(logicalClock, senderClock) + 1;
        System.out.println("Process " + processId + " received a message from Process " +
senderId + ". Updated Timestamp: " + logicalClock);
    }

    public void run() {
        Random rand = new Random();

        for (int i = 0; i < 5; i++) {
            try {
                Thread.sleep(rand.nextInt(1000));
                int action = rand.nextInt(3);

                if (action == 0) {
                    internalEvent();
                } else if (action == 1) {
                    int receiverId = rand.nextInt(processes.length);
                    if (receiverId != processId) {
                        sendMessage(receiverId);
                    }
                }

            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

public class LamportLogicalClock {

    public static void main(String[] args) {
        int numProcesses = 3;
        Process[] processes = new Process[numProcesses];

        for (int i = 0; i < numProcesses; i++) {
            processes[i] = new Process(i, processes);
        }

        for (Process p : processes) {
            p.start();
        }

        for (Process p : processes) {
            try {
                p.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

INTERPROCESS COMMUNICATION USING MULTITHREADED APPLICATION

```
javac ThreadDemo.java
javac TestThread.java
java TestThread
```

notes
do compile this code before ,no need to run
filename should be ThreadDemo.java

Code 1:

```
class ThreadDemo extends Thread {

    private Thread t;
    private String threadName;

    ThreadDemo(String name) {
        threadName = name;
        System.out.println("Creating " + threadName);
    }

    public void run() {
        System.out.println("Running" + threadName);
        try {
            for (int i = 7; i > 0; i--) {
                System.out.println("Thread: " + threadName + ", " + i);
                // Let the thread sleep for a while.
                Thread.sleep(50);
            }
        } catch (InterruptedException e) {
            System.out.println("Thread " + threadName + " interrupted.");
        }
        System.out.println("Thread " + threadName + " exiting.");
    }

    public void start() {
        System.out.println("Starting " + threadName);
        if (t == null) {
            t = new Thread(this, threadName);
            t.start();
        }
    }
}
```

Code 2:

notes
exp :- implement multithreaded application
do compile this code after and run this file
filename should be TestThread.java

```
public class TestThread {

   public static void main(String args[]) {
      ThreadDemo T1 = new ThreadDemo("Thread 1");
      T1.start();

      ThreadDemo T2 = new ThreadDemo("Thread 2");
      T2.start();

      ThreadDemo T3 = new ThreadDemo("Thread 3");
      T3.start();
   }
}
```

IMPLEMENT PROGRAM FOR GROUP COMMUNICATION

Terminal 1:-
```
javac ChatServer.java
javac ChatClient.java
javac ChatClient2.java

java ChatServer
```

Terminal 2:-
```
java ChatClient
```

Terminal 3:-
```
java ChatClient2
```

NOTES:-
run ChatServer.java only once
Make multiple copies of ChatClient.java with different different file names and class name
Don’t forget to change class name in each copy of client file as per new file name
Run each of client and server this individually
Finally start communication

ChatServer.java :

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

public class ChatServer {
    private static final int PORT = 12345;
    private static final List<ClientHandler> clients = new ArrayList<>();

    public static void main(String[] args) {
        System.out.println("Server is running...");
        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("New client connected: " + clientSocket.getInetAddress());

                ClientHandler clientThread = new ClientHandler(clientSocket);
                clients.add(clientThread);
                clientThread.start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    static void broadcast(String message, ClientHandler sender) {
        for (ClientHandler client : clients) {
            if (client != sender) {
                client.sendMessage(message);
            }
        }
    }

    static void removeClient(ClientHandler client) {
        clients.remove(client);
    }
}

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

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

    public void run() {
        try {
            in  = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            out = new PrintWriter(socket.getOutputStream(), true);

            // Get nickname and first message
            out.println("Enter your nickname:");
            nickname = in.readLine();

            out.println("Enter your first message:");
            String firstMessage = in.readLine();
            ChatServer.broadcast(nickname + ": " + firstMessage, this);

            // Read further messages
            String message;
            while ((message = in.readLine()) != null) {
                ChatServer.broadcast(nickname + ": " + message, this);
            }

        } catch (IOException e) {
            System.out.println(nickname + " disconnected.");
        } finally {
            ChatServer.broadcast(nickname + " left the chat.", this);
            ChatServer.removeClient(this);
            try {
                socket.close();
            } catch (IOException e) { e.printStackTrace(); }
        }
    }

    public void sendMessage(String message) {
        out.println(message);
    }
}
```

ChatClient.java :

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

public class ChatClient {
    private static final String SERVER_IP = "127.0.0.1";
    private static final int SERVER_PORT = 12345;

    public static void main(String[] args) {
        try (
            Socket socket = new Socket(SERVER_IP, SERVER_PORT);
            BufferedReader serverInput = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter serverOutput = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in));
        ) {
            // Handle server input asynchronously
            Thread receiveThread = new Thread(() -> {
                String msg;
                try {
                    while ((msg = serverInput.readLine()) != null) {
                        System.out.println(msg);
                    }
                } catch (IOException e) {
                    System.out.println("Connection closed.");
                }
            });
            receiveThread.start();

            // User input
            String inputLine;
            while ((inputLine = userInput.readLine()) != null) {
                serverOutput.println(inputLine);
            }

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

Load balancer

Notes:- how to run

Terminal 1:- **javac Server.java** **javac Client.java** **javac LoadBalancer.java**	**Terminal 2:-** **java Server 5001 Server1** **Terminal 3:-** **java Server 5002 Server2** **Terminal 4:-** **java Server 5003 Server3**	**Terminal 5:-** **java LoadBalancer** **Terminal 6** **java Client**

Client.java

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

public class Client {
    public static void main(String[] args) {
        String loadBalancerHost = "localhost";  // Change this if Load Balancer is on another
machine
        int loadBalancerPort = 8080;  // The port on which Load Balancer is running

        for (int i = 1; i <= 5; i++) {  // Send 5 requests to Load Balancer
            try (Socket socket = new Socket(loadBalancerHost, loadBalancerPort);
                 PrintWriter output = new PrintWriter(socket.getOutputStream(), true);
                 BufferedReader input = new BufferedReader(new
InputStreamReader(socket.getInputStream()))) {

                String request = "Request " + i;
                output.println(request);  // Send request to Load Balancer

                // Receive response from the selected server
                String response = input.readLine();
                System.out.println("Client received: " + response);

            } catch (IOException e) {
                e.printStackTrace();
            }
            try {
                Thread.sleep(1000);  // Wait 1 second between requests for better visualization
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

LoadBalancer.java

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

public class LoadBalancer {
    private List<String> servers;
    private int currentIndex;

    public LoadBalancer(List<String> servers) {
        this.servers = servers;
        this.currentIndex = 0;
    }

    public String getNextServer() {
        String selectedServer = servers.get(currentIndex);
        currentIndex = (currentIndex + 1) % servers.size();
        return selectedServer;
    }

    public void start(int port) {
        try (ServerSocket serverSocket = new ServerSocket(port)) {
            System.out.println("Load Balancer started on port " + port);

            while (true) {
                Socket clientSocket = serverSocket.accept();
                BufferedReader input = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
                PrintWriter output = new PrintWriter(clientSocket.getOutputStream(), true);
                String request = input.readLine();
                String selectedServer = getNextServer();
                System.out.println("Forwarding request: " + request + " to " + selectedServer);

                // Extract server details (IP and port)
                String[] parts = selectedServer.split(":");
                String serverIp = parts[0];
                int serverPort = Integer.parseInt(parts[1]);

                // Forward request to the selected server
                try (Socket serverSocketConn = new Socket(serverIp, serverPort);
                     BufferedReader serverInput = new BufferedReader(new
InputStreamReader(serverSocketConn.getInputStream()));
                     PrintWriter serverOutput = new PrintWriter(serverSocketConn.getOutputStream(),
true)) {

                serverOutput.println(request);
                String serverResponse = serverInput.readLine();
                output.println(serverResponse);
                }

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

    public static void main(String[] args) {
        List<String> servers = Arrays.asList(
            "localhost:5001",
            "localhost:5002",
            "localhost:5003"
        );

        int balancerPort = 8080; // Load balancer port
        new LoadBalancer(servers).start(balancerPort);
    }
}
```

Server.java :

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

public class Server {
    private int port;
    private String serverName;

    public Server(int port, String serverName) {
        this.port = port;
        this.serverName = serverName;
    }

    public void start() {
        try (ServerSocket serverSocket = new ServerSocket(port)) {
            System.out.println(serverName + " started on port " + port);

            while (true) {
                Socket clientSocket = serverSocket.accept();
                BufferedReader input = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
                PrintWriter output = new PrintWriter(clientSocket.getOutputStream(), true);

                String request = input.readLine();
                System.out.println(serverName + " received request: " + request);
                output.println(serverName + " processed request: " + request);

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

    public static void main(String[] args) {
        int port = Integer.parseInt(args[0]);
        String serverName = args[1];
        new Server(port, serverName).start();
    }
}
```

RPC/RMI

To run:-
rmiregistry &
Compile all files in any order
Run only two files infollowing order
java DateTimeServer
 java DateTimeClient

DateTimeClient.java :

```
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;
public class DateTimeClient
{
	public static void main(String args[]) throws Exception
	{
		DateTimeInter obj = (DateTimeInter)Naming.lookup("abc");
		System.out.println("Date Time is :"+obj.getDateTime());

	}

}
```

DateTimeImpl.java

```
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;
import java.util.Date;
public class DateTimeImpl extends UnicastRemoteObject implements DateTimeInter
{
	public DateTimeImpl() throws Exception
	{
		super();
	}

	public String getDateTime() throws Exception
	{
		Date d=new Date();
		return d.toString();
```

```
          }
}
```

DateTimeInter.java

```
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;
public interface DateTimeInter extends Remote
{
          public String getDateTime() throws Exception;

}
```

DateTimeServer.java

```
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;
public class DateTimeServer
{
          public static void main(String args[]) throws Exception
          {
                    DateTimeImpl obj = new DateTimeImpl();
                    Naming.bind("abc",obj);
                    System.out.println("Waiting for Client");
          }

}
```

DateTimeServerr.java

```
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;
public class DateTimeServer
{
	public static void main(String args[]) throws Exception
	{
		DateTimeImpl obj = new DateTimeImpl();
		Naming.bind("abc",obj);
		System.out.println("Waiting for Client");
	}

}
```