PRACTICAL - 3

**AIM** : To implement Lamport’s Logical Clock in Java for event ordering in distributed systems.

**Course Outcome** : CO3

**Software Used** : Java SDK

# Theory :

In distributed systems, it is often necessary to order events in a consistent manner across multiple processes, even in the absence of a global clock. Lamport’s Logical Clock is a simple algorithm that provides a mechanism for this by assigning a numerical timestamp to each event.

The logical clock algorithm works as follows:

1. Initialization: Each process maintains a logical clock (counter) initialized to zero.
2. Event Occurrence: Before executing an event (including sending messages), the process increments its clock.
3. Message Sending: When a process sends a message, it includes its current logical clock value.
4. Message Receiving: Upon receiving a message, the process sets its clock to the maximum of its own clock and the received clock, then increments it.

This algorithm ensures that if an event a causally affects event b, then the logical clock value of a is less than that of b.

**Flowchart/Algorithm** :

1. Setup Java Environment:

* Install JDK and set up an IDE.
* Create a new Java project.

1. Define Classes for Process and Event:

* Create a class Process with an integer field for the logical clock.
* Define a class Message with fields for the sender, receiver, and logical clock value.

1. Simulate Events:

* Implement a method sendEvent() to simulate sending a message from one process to another. The method increments the sender's clock, packages the message with the clock value, and sends it.
* Implement a method receiveEvent() to simulate receiving a message. The method updates the receiver's clock according to the logical clock algorithm.

1. Event Handling:

* Implement logic to handle local events and message events, updating the logical clocks accordingly.
* Maintain a log of events and their logical timestamps.

1. Testing:

* Simulate a series of events and message exchanges between processes.
* Output the logical clock values and verify the correct ordering of events.

# Code:

import java.util.Scanner;

class LamportsClock { int logicalClock;

// Constructor to initialize the clock public LamportsClock() {

logicalClock = 0;

}

// Function to send an event (increments the clock)

public void sendEvent() { logicalClock++;

System.out.println("Send event occurred, updated logical clock: " + logicalClock);

}

// Function to receive an event (updates the clock based on received timestamp) public void receiveEvent(int receivedTimestamp) {

logicalClock = Math.max(logicalClock, receivedTimestamp) + 1; System.out.println("Receive event occurred, updated logical clock: " + logicalClock);

}

// Function to display the logical clock public void displayClock() {

System.out.println("Current logical clock value: " + logicalClock);

}

}

public class LamportsLogicalClock { public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Creating two processes with Lamport clocks LamportsClock process1 = new LamportsClock(); LamportsClock process2 = new LamportsClock(); System.out.println("Name: Rounak Ranjan"); System.out.println("Roll no.: 00520802721"); boolean running = true;

while (running) { System.out.println("\nChoose an option:");

System.out.println("1. Process 1 sends event"); System.out.println("2. Process 2 sends event"); System.out.println("3. Process 1 receives event"); System.out.println("4. Process 2 receives event"); System.out.println("5. Display clocks"); System.out.println("6. Exit");

int choice = sc.nextInt();

switch (choice) { case 1:

process1.sendEvent(); break;

case 2:

process2.sendEvent(); break;

case 3:

System.out.print("Enter the timestamp received by Process 1: "); int receivedTimestamp1 = sc.nextInt(); process1.receiveEvent(receivedTimestamp1);

break; case 4:

System.out.print("Enter the timestamp received by Process 2: "); int receivedTimestamp2 = sc.nextInt(); process2.receiveEvent(receivedTimestamp2);

break; case 5:

System.out.println("Process 1 Clock: "); process1.displayClock(); System.out.println("Process 2 Clock: "); process2.displayClock();

break;

case 6:

running = false; break;

default:

System.out.println("Invalid choice, please try again.");

}

}

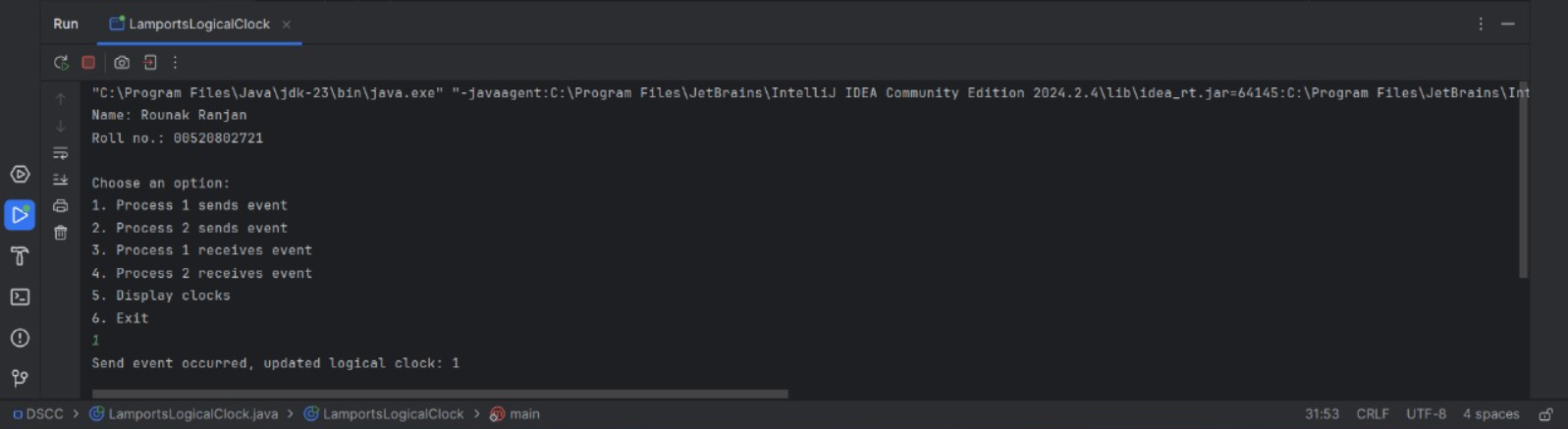
sc.close();

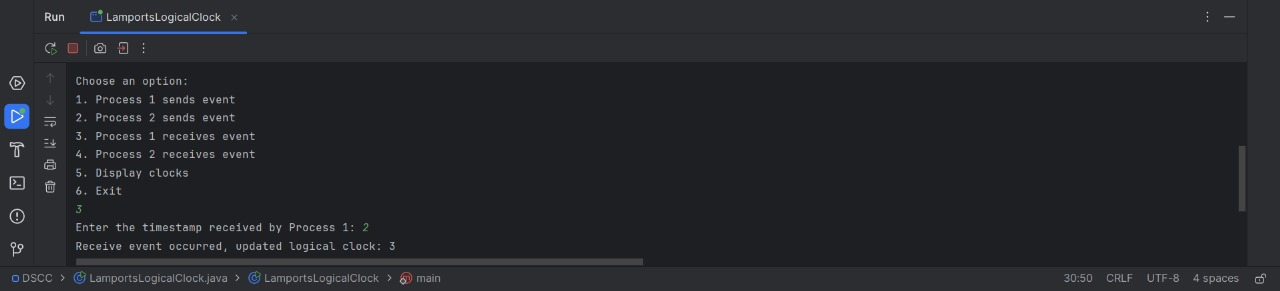
}

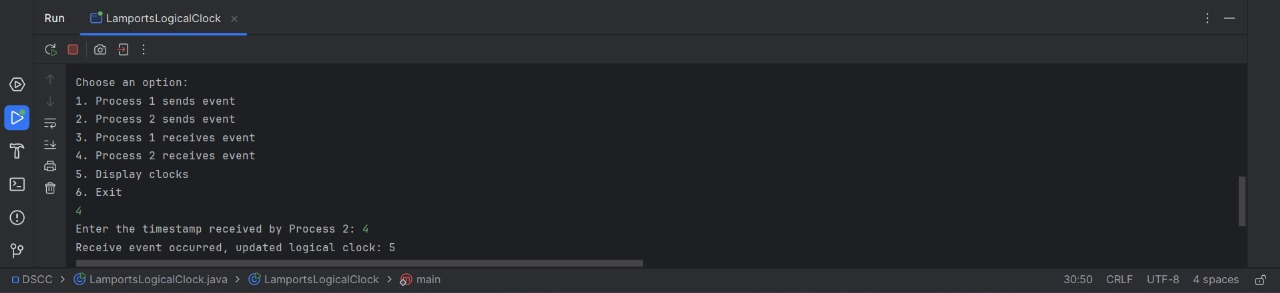
}

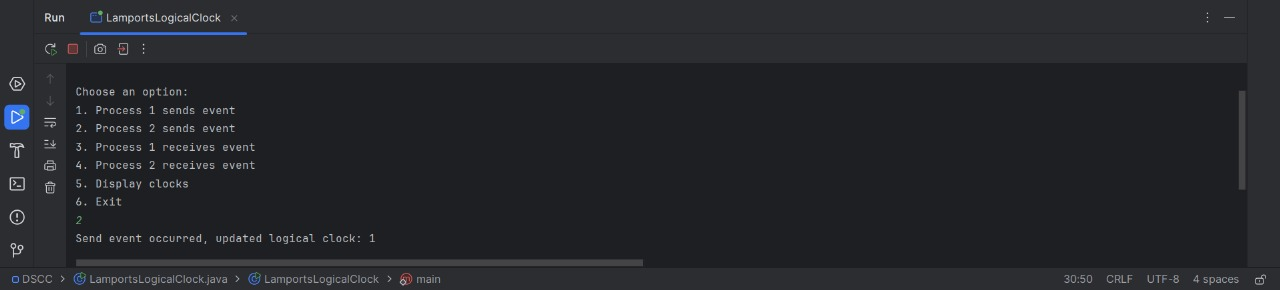
# Results:

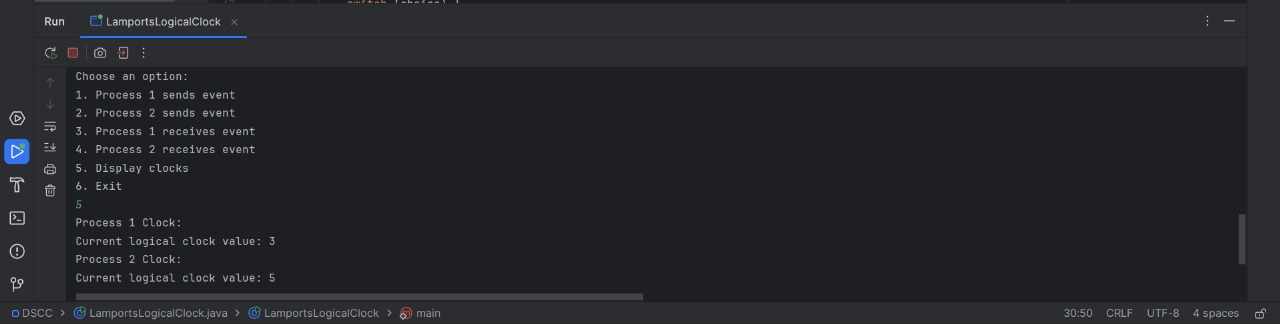
Correctly ordered events according to Lamport’s Logical Clock, ensuring causal relationships are maintained.

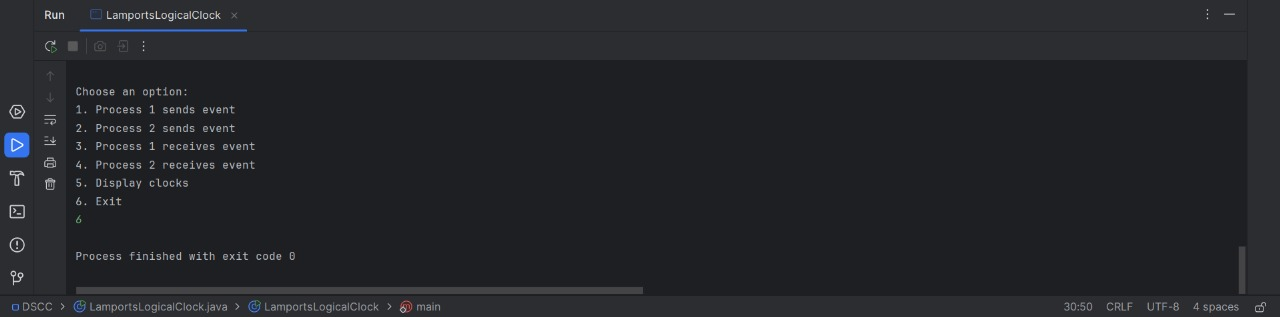












# Expected Outcome attained: Yes