****

**PROJECT 2 -REPORT DISTRIBUTED SYSTEMS CSE-5306**

**Table of Contents**

**Problem 1**

|  |  |
| --- | --- |
| **Topic** | **Page No** |
| Problem Statement | 3 |
| Inference | 3 |
| Solution | 4-10 |

**Problem 2**

|  |  |
| --- | --- |
| **Topic** | **Page No** |
| Problem Statement | 11 |
| Inference | 11 |
| Solution | 11-16 |

**PROBLEM 1**

**PROBLEM STATEMENT:**

Suppose the logical clock on each machine represents the number of messages have been sent and received by this machine. It is actually a counter used by the process (or the machine emulator) to count events. Randomly initialize the logical clock of individual processes and use Berkeley’s algorithm to synchronize these clocks to the average clock. You can select any process as the time daemon to initiate the clock synchronization. After the synchronization, each process prints out its logical clock to check the result of synchronization.

**INFERENCE FROM QUESTION:**

From the above stated problem, we can infer that we are asked to design a distributed system that implements a totally ordered multicasting, using Lamport’s Algorithm.

We need to use Berkeley Algorithm to get an initial agreement on logical clocks.

**LAMPORT’s ALGORITHM:**

Lamport was the first to give a distributed mutual exclusion algorithm as an illustration of his clock synchronization scheme. Let Ri be the request set of site Si , i.e. the set of sites from which Si needs permission when it wants to enter CS. In Lamport's algorithm, ∀i : 1 ≤ i ≤ N :: Ri= {S1, S2,...,SN}. Every site Si keeps a queue, request\_queuei, which contains mutual exclusion requests ordered by their timestamps. This algorithm requires messages to be delivered in the FIFO order between every pair of sites.

**The Algorithm**

**Requesting the critical section**.

1. When a site Si wants to enter the CS, it sends REQUEST(tsi, i) message to all the sites in its request set Ri and places the request on request\_queuei (tsi is the timestamp of the request).

2. When a site Sj receives the REQUEST(tsi, i) message from site Si, it returns a timestamped REPLY message to Si and places site S'is request on request\_queuej.

**Executing the critical section.**

1. Site Si enters the CS when the two following conditions hold:

a) [L1:] Si has received a message with timestamp larger than (tsi, i) from all other sites. b) [L2:] S'is request is at the top request\_queuei.

**Releasing the critical section.**

1. Site Si, upon exiting the CS, removes its request from the top of its request queue and sends a timestamped RELEASE message to all the sites in its request set.

2. When a site Sj receives a RELEASE message from site Si, it removes S'is request from its request queue.

When a site removes a request from its request queue, its own request may come at the top of the queue, enabling it to enter CS. The algorithm executes CS requests in the increasing order of timestamps.

**SOLUTION:**

**1.SOFTWARE USED:**

* Java Eclipse IDE Oxygen (June 2017)

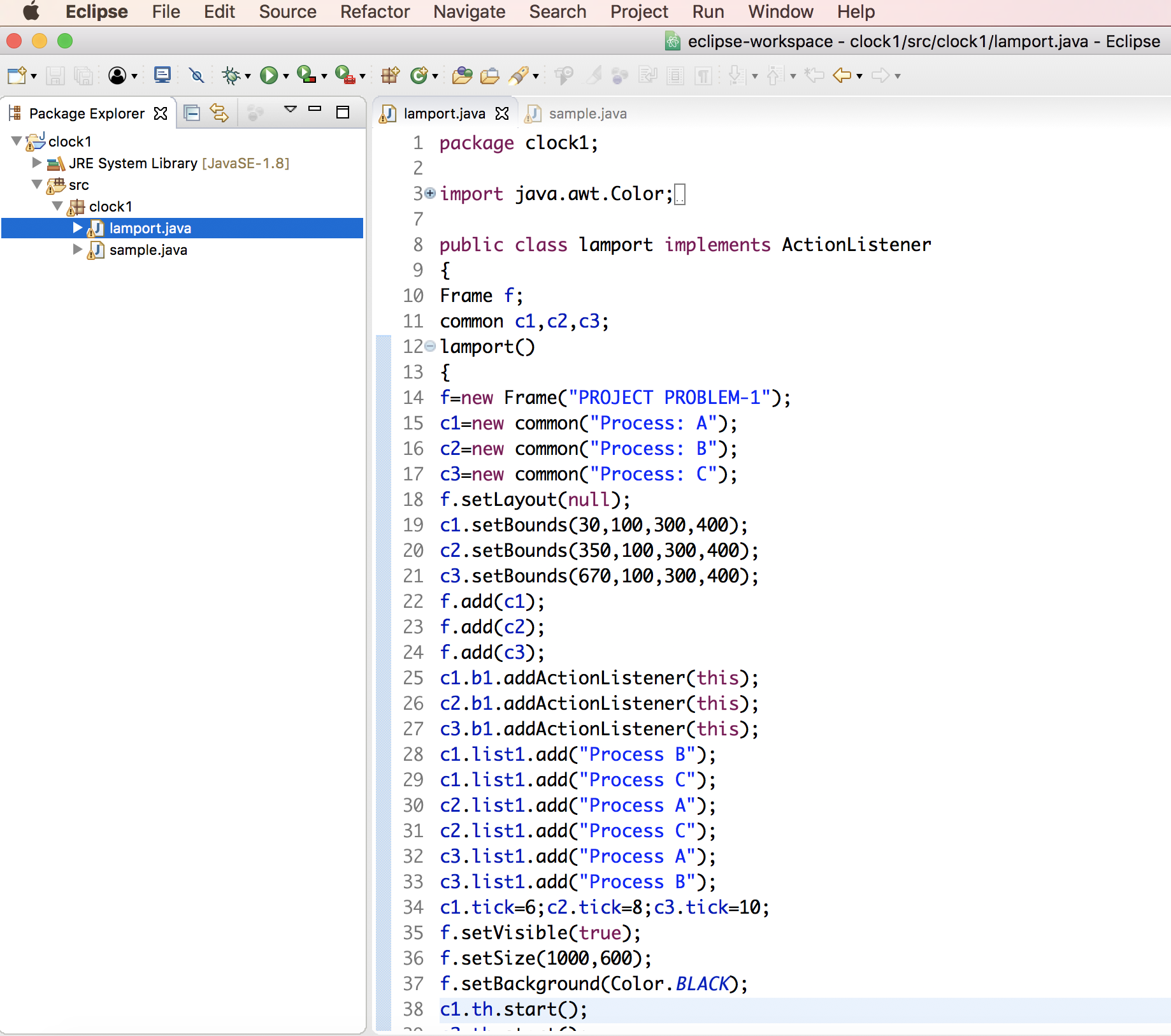
**2. RUNNING THE APPLICATION:**

The totally ordered multicast program using Lamport’s Algorithm comprises of the following segment of codes:

* Lamport
* Sample

**Now let us run the code and see the output.**

Click on the run as Java Application from the Package Explorer option of the Eclipse IDE.

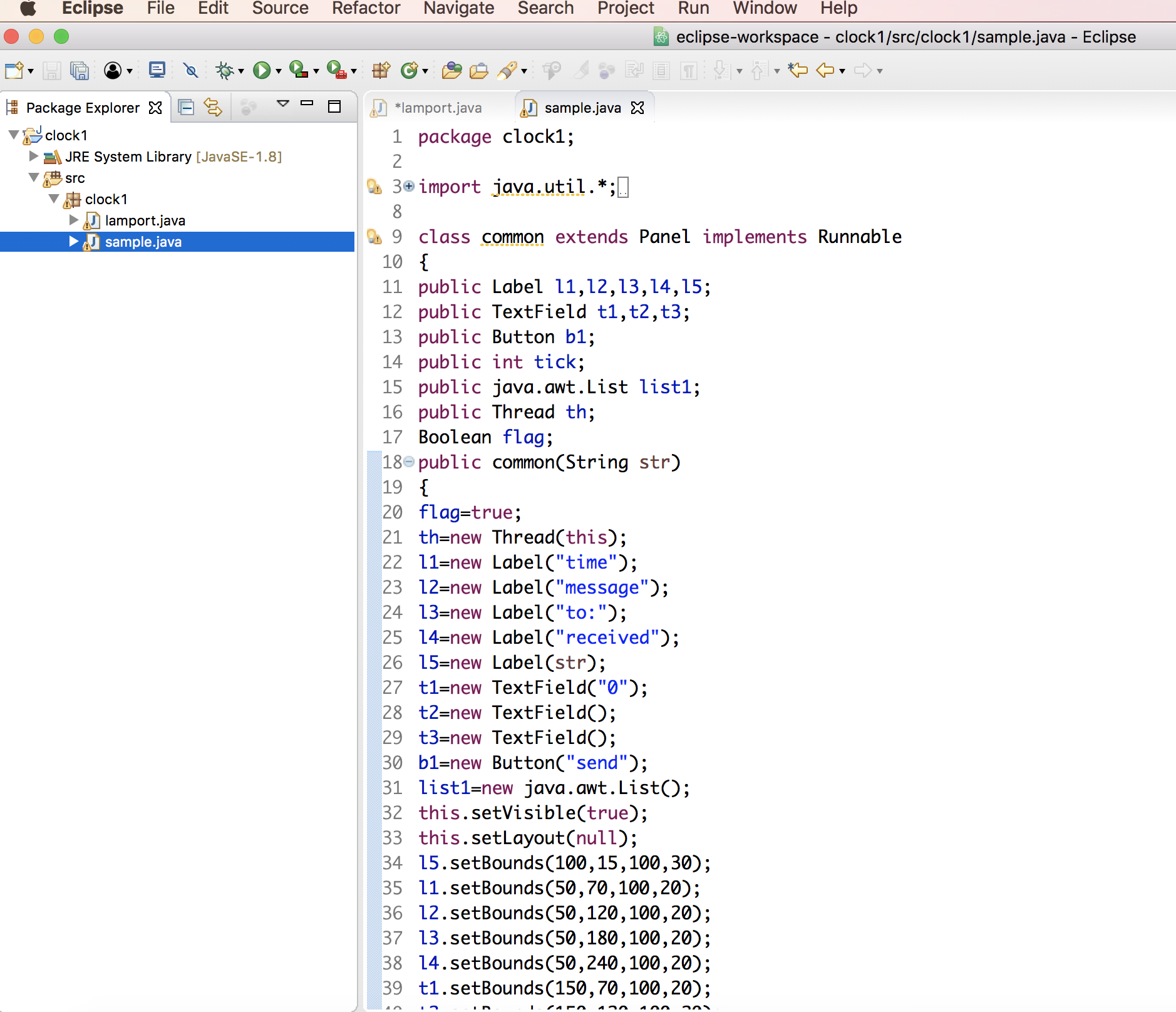


**Fig 1.1: Setting the Layout of the GUI for 3 processes**

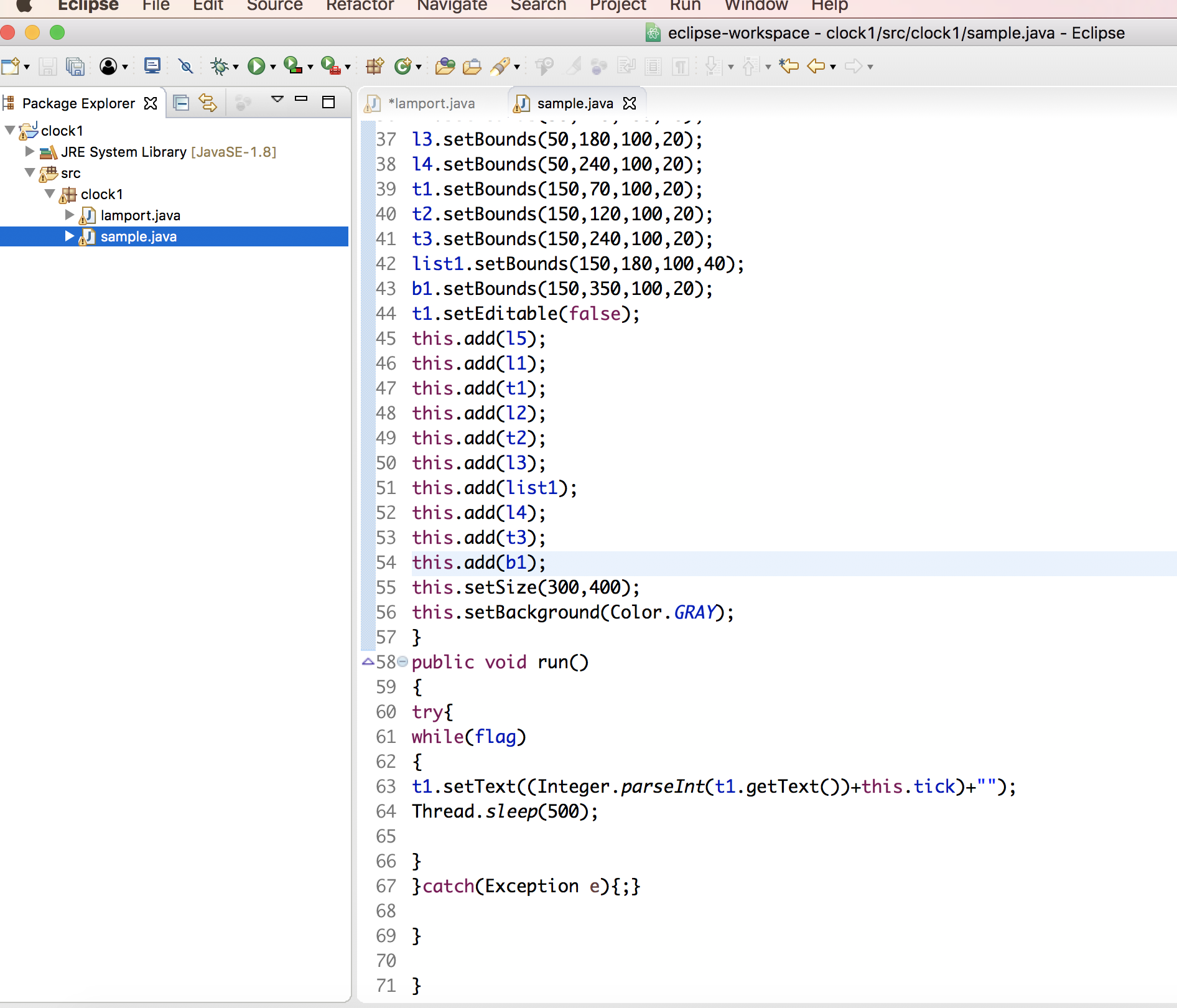
We have used Berkeley’s Algorithm to average out the clock timings for all the 3 processes.



**Fig:1.2 Sending messages with respective timestamps to other processes**

****

**Fig 1.3: Creating Thread class**

****

**Fig 1.4: Creating Processes**

**Output:**

****

**Fig 1.5a: Sending message from Process A to Process B**

****

**Fig 1.5b: Sending message from Process A to Process B**

****

**Fig 1.6a: Sending message from Process B to Process C**

**PROBLEM-2**

**PROBLEM STATEMENT:**

Implement the totally ordered multicasting for the distributed system. Create two threads for each process, one for sending the multicast message to other nodes and one for listening to it communication port. Use either Lamport’s algorithm or the vector clocks to enforce the order of messages.

**INFERENCE FROM QUESTION:**

From the above stated problem, we can infer that we are asked to design a distributed system which that implements a totally ordered multicasting

**TOTALLY ORDERED MULTICAST:**

* To get the same answers, we need totally-ordered multicast, i.e. all messages are delivered in the same order to each receiver.
* Can be done using vector timestamp.
* Consider a group of processes multi-casting to each other.
* Each message is time-stamped with the logical time of the sender
* Assume multicast also goes to the sender
* Assume messages from the same sender are received by any one receiver in the order sent
* Assume that no messages are lost
* When message is received, put it in a local queue ordered by its time-stamp
* The receiver multi-casts an ACK to all processes (Note ACK is not queued)
* A process can "act on" a queued message when it is at the head of the queue, and has been ACK by every other process
* Each process has the same contents in the queue

**SOLUTION:**

**1.SOFTWARE USED:**

* Java Eclipse IDE Oxygen (June 2017)

**2. RUNNING THE APPLICATION:**

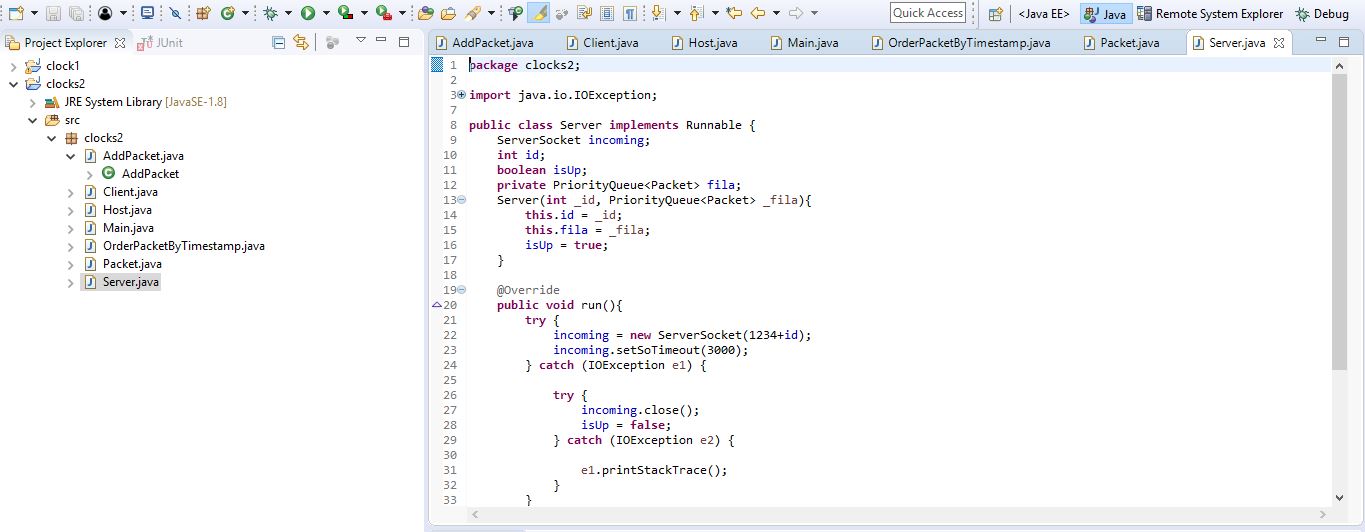
The totally ordered multicast program using Vector Clocks comprises of the following segment of codes:

* AddPacket
* Client
* Host
* Main
* OrderPacketBytimestamp
* Packet
* Server

All the segments are predominantly written using Java Socket programming and threading techniques.

Now let us run the code and see the output.

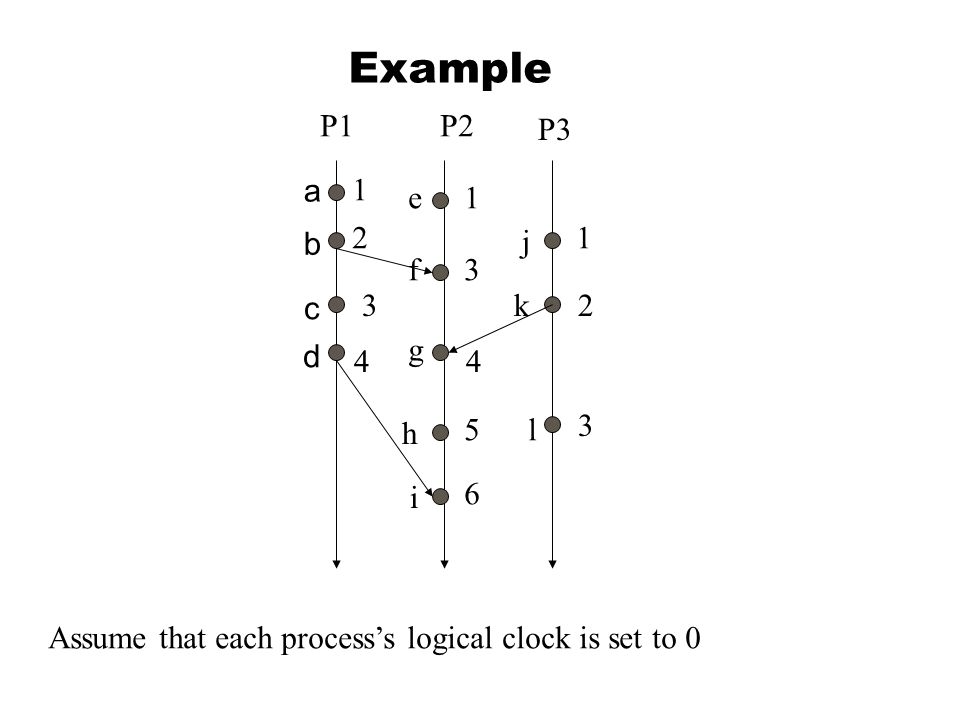
Click on the run as Java Application from the Package Explorer option of the Eclipse IDE.



**FIG 2.1- CODE SEGMENT**

The Eclipse IDE cumulatively runs the code segments and establishes the Totally Ordered multicast.

The process of totally ordered multicast is represented as shown below.



**FIG 2.2- TOTALLY ORDERED MULTICAST REPRESENTATION**

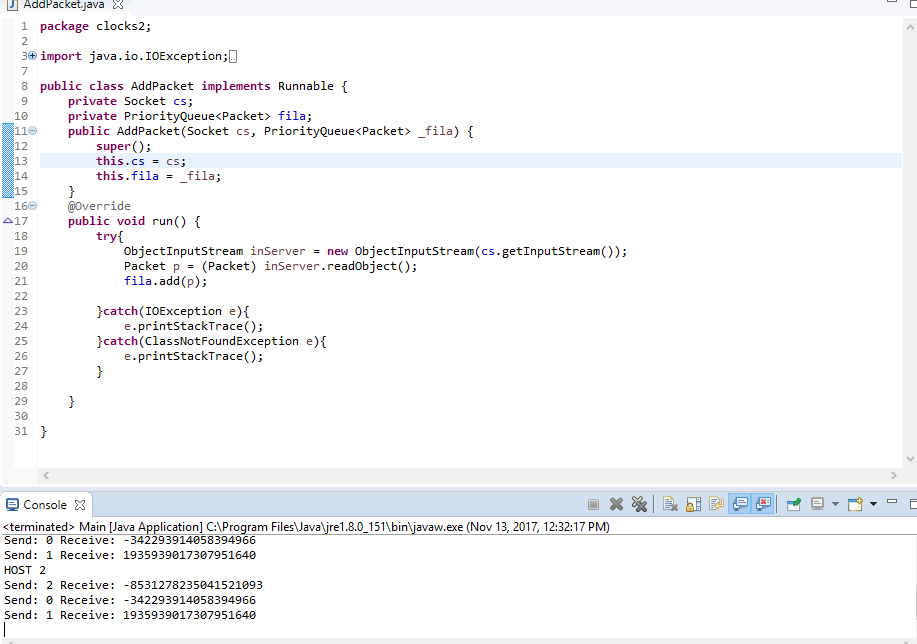
**Explanation of Code:**

Implemented Vector Clocks Algorithm to achieve multicast communication while ensuring delivery of messages in order across all nodes using TCP/IP & multithreading in Java.

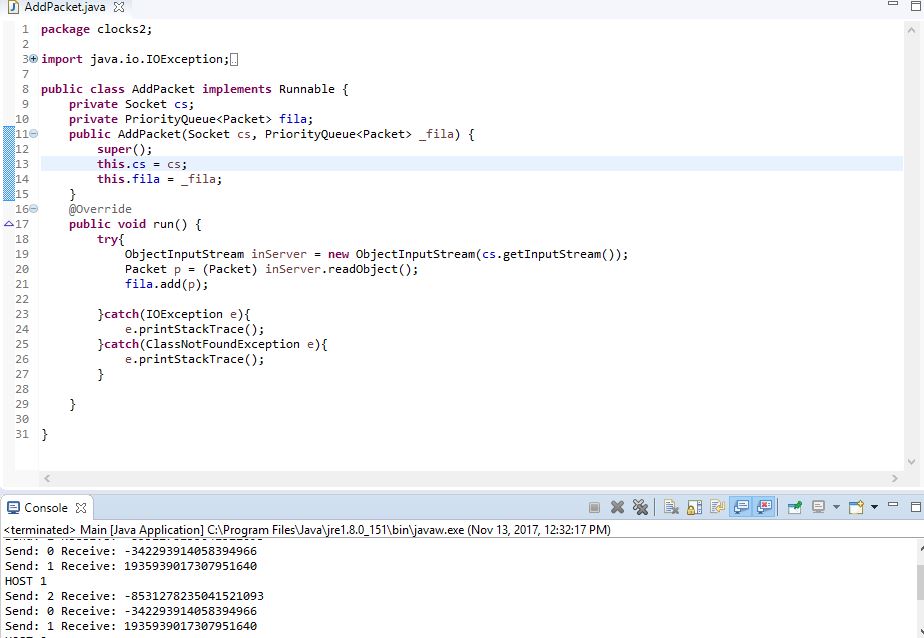
**OUTPUT GENERATED:**

On execution of the code, the threads are executed concurrently, and the messages are passed along with their timestamps.

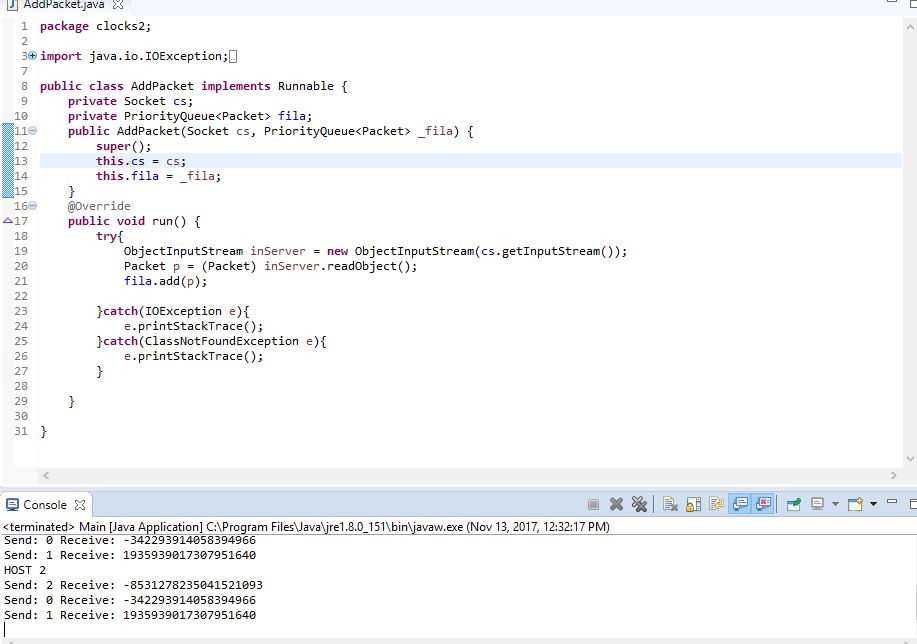
Once a process delivers a received message to a user, it prints out the message onscreen. A totally ordered multicasting requires that all process print out received messages in the same order



**FIG 2.3a- TOTALLY ORDERED MULTICAST MESSAGE PASSING**



**FIG 2.3b- TOTALLY ORDERED MULTICAST MESSAGE PASSING**



**FIG 2.3c- TOTALLY ORDERED MULTICAST MESSAGE PASSING**

**BIBLIOGRAPHY:**

[1]- <http://web.cs.iastate.edu/~cs554/NOTES/Ch6-LogicalClocks.pdf>

[2]- <http://www.cs.put.poznan.pl/akobusinska/downloads/rso/mutual_exclusion_algs.pdf>

[3]- <https://www.cs.unc.edu/~anderson/papers/lamport.pdf>

[4]- <https://github.com/>

[5]- <https://dl.acm.org/citation.cfm?id=214456>

[6]- <https://docs.oracle.com/javase/tutorial/essential/>

[7]- <https://dl.acm.org/citation.cfm?id=359563>

[8]- <https://beginnersbook.com/2013/03/multithreading-in-java/>