**🌀 Token Ring Mutual Exclusion Algorithm – Java Implementation (Detailed Assignment Guide)**

**🧾 Assignment Statement**

Design and implement a simulation of the Token Ring-based Mutual Exclusion Algorithm using Java. Demonstrate how data (or token) is passed among nodes in a logical ring to ensure mutual exclusion.

**📌 What Is the Token Ring Algorithm?**

It is a distributed mutual exclusion algorithm that uses a token (a unique message) which circulates among processes arranged in a logical ring. Only the process holding the token can enter the critical section or initiate communication.

* It avoids deadlock and starvation.
* Uses a circular queue (logical ring) of processes.
* Token acts like a permission slip for critical section access.

**💡 Real-World Analogy**

Imagine students sitting in a circle sharing one microphone. Only the student holding the microphone is allowed to speak (i.e., access the critical section). Once done, the student passes the microphone clockwise.

**⚙️ Java Code – Full Line-by-Line Explanation**

Here's your code with complete annotations for each logic block:

import java.util.Scanner;

Import Scanner to take user input from the console.

public class TokenRing {

Main class definition.

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Entry point of the program. A Scanner is initialized for input.

System.out.print("Enter Number of Nodes you want in the Ring: ");

int n = sc.nextInt();

User inputs how many nodes (processes) are in the ring.

System.out.println("\nRing Formed is as below: ");

for (int i = 0; i < n; i++) {

System.out.print(i + " ");

}

System.out.println("0");

Displays the circular ring of nodes visually. For example, if n=4: 0 1 2 3 0

int choice = 0;

do {

A loop starts to allow multiple data send attempts.

System.out.println("Enter Sender: ");

int sender = sc.nextInt();

System.out.println("Enter Receiver: ");

int receiver = sc.nextInt();

sc.nextLine(); // Clear newline

System.out.print("Enter data to be sent: ");

String data = sc.nextLine();

Sender and receiver node numbers and the data message are taken as input.

int token = 0; // Token starts at process 0

System.out.print("Token Passing: ");

for (int i = token; i < receiver; i++) {

System.out.print(" " + i + " ->");

}

System.out.println(" " + receiver);

Simulates token movement from node 0 to the receiver.

⚠️ Limitation: It assumes sender/receiver are ahead of 0 in linear order. It doesn’t wrap from n-1 to 0.

System.out.println("Sender: " + sender + " , Sending Data: " + data);

Logs the sender and data being sent.

for (int i = sender; i != receiver; i = (i + 1) % n) {

System.out.println("Data: " + data + " , Forwarded By: " + i);

}

Simulates the data being forwarded from the sender node to the receiver node through the ring. It uses modulo to wrap around the ring.

System.out.println("Receiver: " + receiver + " , Received the Data: " + data);

Receiver logs the data reception.

token = sender;

System.out.println();

System.out.print("Do you want to send data again? If YES enter 1, If NO enter 0: ");

choice = sc.nextInt();

Asks the user if they want to continue or exit.

} while (choice == 1);

sc.close();

}

}

Ends the loop and closes the scanner.

**🧪 How the Flow Works (Step-by-Step)**

1. User enters number of nodes, e.g., 4 → Ring: 0 1 2 3 0
2. User enters sender, receiver, and data → e.g., sender: 2, receiver: 0, data: "Hi"
3. Token is passed starting from 0 to receiver (not always accurate if receiver < 0)
4. Sender begins forwarding the message
5. Each node forwards until it reaches receiver
6. Receiver logs the data

**✅ Features Implemented**

✔️ Ring formation  
✔️ Token passing simulation  
✔️ Message/data transfer  
✔️ Looping for multiple transmissions

**⚠️ Current Limitations**

❌ Does not correctly simulate wrap-around token passing (e.g., 3 → 0)  
❌ Only token path from 0 is simulated, not from sender → receiver  
❌ No critical section logic or mutual exclusion flag  
❌ No fault tolerance or lost token handling

**✨ Possible Enhancements**

* Implement dynamic ring: remove/add nodes
* Simulate true mutual exclusion (process must acquire and release token)
* Add failure detection and token regeneration
* Wrap-around token passing fix using:  
  From sender to receiver: while (i != receiver) i = (i + 1) % n
* GUI using JavaFX/Swing to visualize the ring and data flow

**🎓 Viva Questions & Answers**

| **Question** | **Answer** |
| --- | --- |
| What is a token? | A permission message that gives access to the critical section. |
| What is the purpose of the ring? | It ensures every process gets a turn in a fair order. |
| Can two processes hold the token at once? | No. Token is unique and singular. |
| What happens if the token is lost? | A recovery mechanism must detect and regenerate the token. |
| Is this implementation fault-tolerant? | No. Fault tolerance is not yet handled. |
| What is the time complexity? | O(n) in worst case (entire ring traversal). |
| Why do we use modulo in the loop? | To wrap the process index circularly (e.g., 3 → 0). |
| What are real-life uses of token ring? | Networks (like old IEEE 802.5), process synchronization in distributed OS. |