import java.util.\*;

import java.util.concurrent.\*;

class TokenRing {

static final int NUM\_PROCESSES = 5; // Number of processes

static List<Process> processes = new ArrayList<>();

static Semaphore mutex = new Semaphore(1);

static boolean[] flags = new boolean[NUM\_PROCESSES];

static int tokenHolder = 0; // Initially, process 0 holds the token

static Random random = new Random();

public static void main(String[] args) throws InterruptedException {

for (int i = 0; i < NUM\_PROCESSES; i++) {

Process p = new Process(i);

processes.add(p);

new Thread(p).start();

}

}

static class Process implements Runnable {

int id;

Process(int id) {

this.id = id;

}

public void run() {

while (true) {

try {

Thread.sleep(random.nextInt(1000)); // Simulate random wait

requestToken();

enterCriticalSection();

releaseToken();

Thread.sleep(random.nextInt(1000)); // Simulate random wait before next request

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

void requestToken() throws InterruptedException {

mutex.acquire();

while (tokenHolder != id) { // Wait for the token to be passed

System.out.println("Process " + id + " waiting for the token.");

mutex.release();

Thread.sleep(100); // Simulate waiting time

mutex.acquire();

}

System.out.println("Process " + id + " acquired the token.");

mutex.release();

}

void enterCriticalSection() throws InterruptedException {

System.out.println("Process " + id + " entering critical section.");

Thread.sleep(random.nextInt(500)); // Simulate critical section work

}

void releaseToken() throws InterruptedException {

mutex.acquire();

System.out.println("Process " + id + " exiting critical section and passing token.");

tokenHolder = (tokenHolder + 1) % NUM\_PROCESSES; // Pass token to the next process

mutex.release();

}

}

}