Practicla No : 05
Name : Kiran Dadarao Janjal
Roll No : 13216
Batch: B1
Program :

```
import java.util.concurrent.Semaphore;
import java.util.LinkedList;
import java.util.Queue;

// ====================== Producer-Consumer Problem
======================
class ProducerConsumer {
    private final int BUFFER_SIZE = 5;
    private Queue<Integer> buffer = new LinkedList<>();
    private Semaphore mutex = new Semaphore(1);
    private Semaphore full = new Semaphore(0);
    private Semaphore empty = new Semaphore(BUFFER_SIZE);
    private final int TOTAL_OPERATIONS = 10; // Number of items to
produce/consume

    class Producer extends Thread {
        public void run() {
            int item = 0;
            try {
                for (int i = 0; i < TOTAL_OPERATIONS; i++) {
                    empty.acquire();
                    mutex.acquire();
                    buffer.add(item);
                    System.out.println("Produced: " + item);
                    item++;
                    mutex.release();
                    full.release();
                    Thread.sleep(200);
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    class Consumer extends Thread {
        public void run() {
            try {
                for (int i = 0; i < TOTAL_OPERATIONS; i++) {
```

```
                    full.acquire();
                    mutex.acquire();
                    int item = buffer.poll();
                    System.out.println("Consumed: " + item);
                    mutex.release();
                    empty.release();
                    Thread.sleep(300);
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    public void startSimulation() throws InterruptedException {
        Thread producer = new Producer();
        Thread consumer = new Consumer();
        producer.start();
        consumer.start();
        producer.join();
        consumer.join();
    }
}

// ====================== Readers-Writers Problem (Reader Preference)
======================
class ReadersWriters {
    private int readCount = 0;
    private Semaphore mutex = new Semaphore(1);
    private Semaphore wrt = new Semaphore(1);
    private final int TOTAL_OPERATIONS = 5; // Each reader/writer runs
5 times

    class Reader extends Thread {
        private int id;
        Reader(int id) { this.id = id; }

        public void run() {
            try {
                for (int i = 0; i < TOTAL_OPERATIONS; i++) {
                    mutex.acquire();
                    readCount++;
                    if (readCount == 1) wrt.acquire();
                    mutex.release();
```

```
                    System.out.println("Reader " + id + " is
reading");
                    Thread.sleep(200);

                    mutex.acquire();
                    readCount--;
                    if (readCount == 0) wrt.release();
                    mutex.release();

                    Thread.sleep(200);
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    class Writer extends Thread {
        private int id;
        Writer(int id) { this.id = id; }

        public void run() {
            try {
                for (int i = 0; i < TOTAL_OPERATIONS; i++) {
                    wrt.acquire();
                    System.out.println("Writer " + id + " is
writing");
                    Thread.sleep(300);
                    wrt.release();
                    Thread.sleep(200);
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    public void startSimulation() throws InterruptedException {
        Thread r1 = new Reader(1);
        Thread r2 = new Reader(2);
        Thread w1 = new Writer(1);
        r1.start();
        r2.start();
```

```
        w1.start();
        r1.join();
        r2.join();
        w1.join();
    }
}

// ====================== Dining Philosophers Problem
======================
class DiningPhilosophers {
    private final int NUM_PHILOSOPHERS = 5;
    private Semaphore[] chopsticks = new Semaphore[NUM_PHILOSOPHERS];
    private final int TOTAL_MEALS = 3; // Each philosopher eats 3
times

    class Philosopher extends Thread {
        private int id;
        Philosopher(int id) { this.id = id; }

        public void run() {
            try {
                for (int i = 0; i < TOTAL_MEALS; i++) {
                    think();
                    chopsticks[id].acquire();
                    chopsticks[(id + 1) % NUM_PHILOSOPHERS].acquire();
                    eat();
                    chopsticks[id].release();
                    chopsticks[(id + 1) % NUM_PHILOSOPHERS].release();
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        private void think() throws InterruptedException {
            System.out.println("Philosopher " + id + " is thinking");
            Thread.sleep(100);
        }

        private void eat() throws InterruptedException {
            System.out.println("Philosopher " + id + " is eating");
            Thread.sleep(200);
        }
    }
```

```
        public void startSimulation() throws InterruptedException {
            for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
                chopsticks[i] = new Semaphore(1);
            }
            Thread[] philosophers = new Thread[NUM_PHILOSOPHERS];
            for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
                philosophers[i] = new Philosopher(i);
                philosophers[i].start();
            }
            for (Thread p : philosophers) p.join();
        }
    }

    // ====================== Main Class ======================
    public class SynchronizationProblemsFinite {
        public static void main(String[] args) throws InterruptedException
    {
            System.out.println("=== Producer-Consumer Simulation ===");
            new ProducerConsumer().startSimulation();

            System.out.println("\n=== Readers-Writers Simulation ===");
            new ReadersWriters().startSimulation();

            System.out.println("\n=== Dining Philosophers Simulation
    ===");
            new DiningPhilosophers().startSimulation();

            System.out.println("\n=== All Simulations Completed ===");
        }
    }
```

OUTPUT :

```
=== Producer-Consumer Simulation ===
Produced: 0
Consumed: 0
Produced: 1
Consumed: 1
Produced: 2
Produced: 3
Consumed: 2
Produced: 4
Consumed: 3
Produced: 5
Consumed: 4
```

```
Produced: 6
Produced: 7
Consumed: 5
Produced: 8
Consumed: 6
Produced: 9
Consumed: 7
Consumed: 8
Consumed: 9

=== Readers-Writers Simulation ===
Reader 2 is reading
Reader 1 is reading
Writer 1 is writing
Reader 1 is reading
Reader 2 is reading
Writer 1 is writing
Reader 1 is reading
Reader 2 is reading
Writer 1 is writing
Reader 1 is reading
Reader 2 is reading
Writer 1 is writing
Reader 1 is reading
Reader 2 is reading
Writer 1 is writing

=== Dining Philosophers Simulation ===
Philosopher 1 is thinking
Philosopher 0 is thinking
Philosopher 4 is thinking
Philosopher 3 is thinking
Philosopher 2 is thinking
```

```
PID     AT      BT      PR      CT      TAT     WT
1       0       5       2       14      14      9
2       1       3       1       11      10      7
3       2       8       4       22      20      12
4       3       6       3       20      17      11
Average Waiting Time: 9.75
Average Turnaround Time: 15.25
Gantt Chart:
|0 P1 |2 P2 |4 P3 |6 P1 |8 P4 |10 P2 |11 P3 |13 P1 |14 P4 |16 P3 |18 P4 |20 P3
|22|


[Program finished]
```