LAB # 6

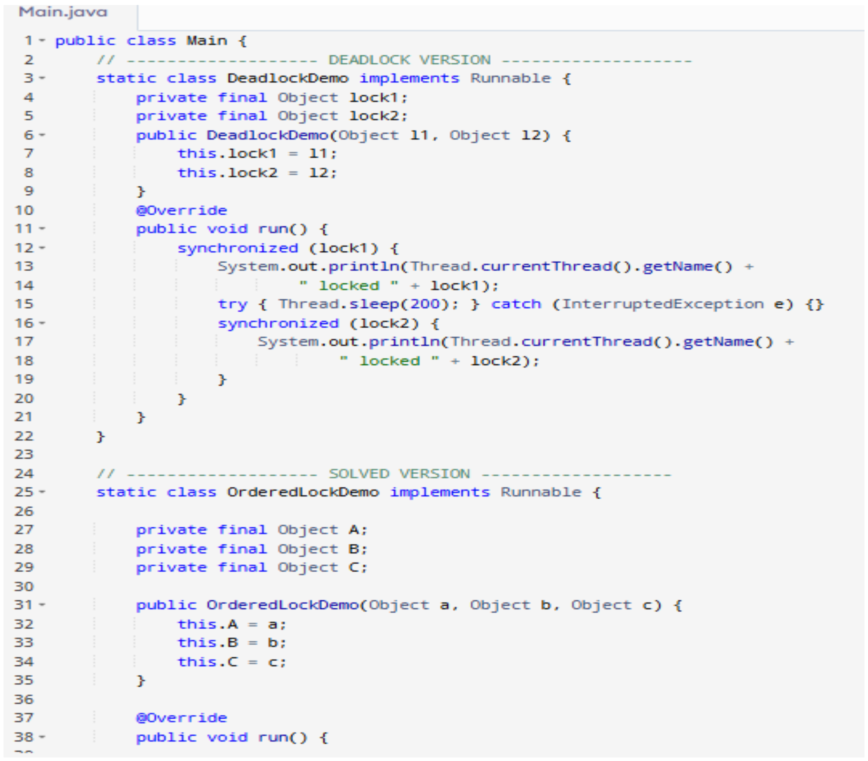
Deadlock in concurrency:

OBJECTIVE:

Implementing multiple thread blocked resources with help of lock and deadlock conditions.

Lab Task:

Create three threads by implementing thread synchronization block through 3 locks. (Hint: Apply un-sequenced lock to analyze deadlock and solve it through provided solution:



Main.java

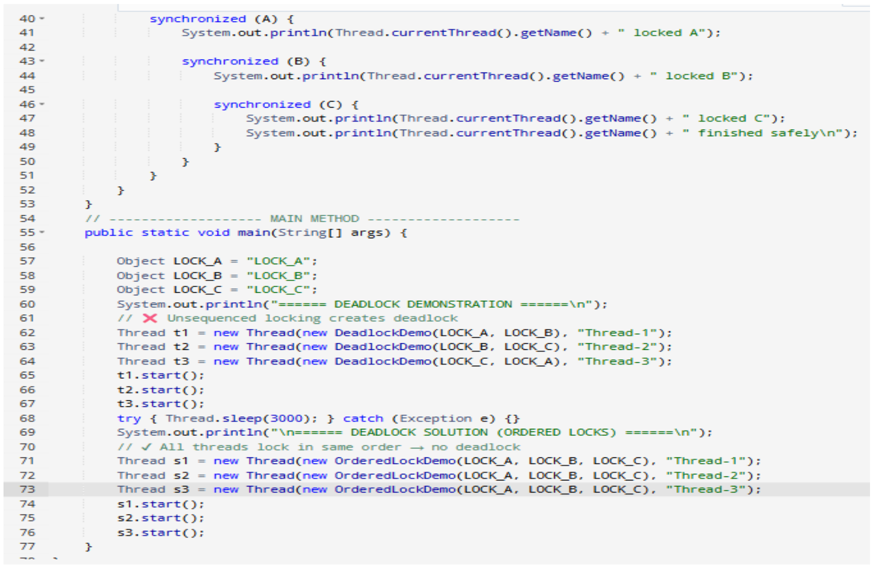
```java
public class Main {
    // -------------------- DEADLOCK VERSION --------------------
    static class DeadlockDemo implements Runnable {
        private final Object lock1;
        private final Object lock2;
        public DeadlockDemo(Object l1, Object l2) {
            this.lock1 = l1;
            this.lock2 = l2;
        }
        @Override
        public void run() {
            synchronized (lock1) {
                System.out.println(Thread.currentThread().getName() +
                        " locked " + lock1);
                try { Thread.sleep(200); } catch (InterruptedException e) {}
                synchronized (lock2) {
                    System.out.println(Thread.currentThread().getName() +
                            " locked " + lock2);
                }
            }
        }
    }

    // -------------------- SOLVED VERSION --------------------
    static class OrderedLockDemo implements Runnable {

        private final Object A;
        private final Object B;
        private final Object C;

        public OrderedLockDemo(Object a, Object b, Object c) {
            this.A = a;
            this.B = b;
            this.C = c;
        }

        @Override
        public void run() {
```



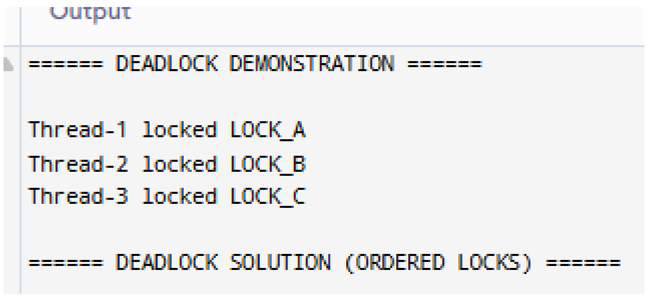
```java
            synchronized (A) {
                System.out.println(Thread.currentThread().getName() + " locked A");

                synchronized (B) {
                    System.out.println(Thread.currentThread().getName() + " locked B");

                    synchronized (C) {
                        System.out.println(Thread.currentThread().getName() + " locked C");
                        System.out.println(Thread.currentThread().getName() + " finished safely\n");
                    }
                }
            }
        }
    }
    // -------------------- MAIN METHOD --------------------
    public static void main(String[] args) {

        Object LOCK_A = "LOCK_A";
        Object LOCK_B = "LOCK_B";
        Object LOCK_C = "LOCK_C";
        System.out.println("====== DEADLOCK DEMONSTRATION ======\n");
        // ❌ Unsequenced locking creates deadlock
        Thread t1 = new Thread(new DeadlockDemo(LOCK_A, LOCK_B), "Thread-1");
        Thread t2 = new Thread(new DeadlockDemo(LOCK_B, LOCK_C), "Thread-2");
        Thread t3 = new Thread(new DeadlockDemo(LOCK_C, LOCK_A), "Thread-3");
        t1.start();
        t2.start();
        t3.start();
        try { Thread.sleep(3000); } catch (Exception e) {}
        System.out.println("\n====== DEADLOCK SOLUTION (ORDERED LOCKS) ======\n");
        // ✓ All threads lock in same order → no deadlock
        Thread s1 = new Thread(new OrderedLockDemo(LOCK_A, LOCK_B, LOCK_C), "Thread-1");
        Thread s2 = new Thread(new OrderedLockDemo(LOCK_A, LOCK_B, LOCK_C), "Thread-2");
        Thread s3 = new Thread(new OrderedLockDemo(LOCK_A, LOCK_B, LOCK_C), "Thread-3");
        s1.start();
        s2.start();
        s3.start();
    }
```

Output:



```
====== DEADLOCK DEMONSTRATION ======

Thread-1 locked LOCK_A
Thread-2 locked LOCK_B
Thread-3 locked LOCK_C

====== DEADLOCK SOLUTION (ORDERED LOCKS) ======
```