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:

Code:

Task1.java

```
package Lab6;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
public class Task1 {
    static final Lock lock1 = new ReentrantLock();
    static final Lock lock2 = new ReentrantLock();
    static final Lock lock3 = new ReentrantLock();
    static class DeadlockThread extends Thread {
        private final Lock firstLock, secondLock;
        DeadlockThread(Lock firstLock, Lock secondLock) {
            this.firstLock = firstLock;
            this.secondLock = secondLock;}
        public void run() {try {
                firstLock.lock();
                System.out.println(Thread.currentThread().getName() + ": Acquired " + firstLock);
                Thread.sleep(100);
                secondLock.lock();
                System.out.println(Thread.currentThread().getName() + ": Acquired " + secondLock);
                Thread.sleep(100);} catch (InterruptedException e) {
                e.printStackTrace();} finally {
                unlock(firstLock);
                unlock(secondLock);}}
        private void unlock(Lock lock) {
            if (lock instanceof ReentrantLock && ((ReentrantLock) lock).isHeldByCurrentThread()) {
                lock.unlock();}}}
    static class ResolvedThread extends Thread {
        private final Lock[] locks;
        ResolvedThread(Lock... locks) {
            this.locks = locks;}
        public void run() {try {
                for (Lock lock : locks) {
                    lock.lock();
                    System.out.println(Thread.currentThread().getName() + ": Acquired " + lock);
                    Thread.sleep(50);}
```

Task1.java

```
                Thread.sleep(100);} catch (InterruptedException e) {
                e.printStackTrace();} finally {
                for (int i = locks.length - 1; i >= 0; i--) {
                    unlock(locks[i]);}}}
        private void unlock(Lock lock) {
            if (lock instanceof ReentrantLock && ((ReentrantLock) lock).isHeldByCurrentThread()) {
                lock.unlock();}}}
    static class TryLockThread extends Thread {
        private final Lock[] locks;
        TryLockThread(Lock... locks) {
            this.locks = locks;}
        public void run() {try {
                if (tryLockAll()) {
                    System.out.println(Thread.currentThread().getName() + ": Acquired all locks");
                    Thread.sleep(100);} else {
                    System.out.println(Thread.currentThread().getName() + ": Could not acquire all locks,
            } catch (InterruptedException e) {
                e.printStackTrace();} finally {
                unlockAll();}}
        private boolean tryLockAll() {
            for (Lock lock : locks) {
                if (!lock.tryLock()) return false;}
            return true;}
        private void unlockAll() {
            for (Lock lock : locks) {
                unlock(lock);}}
        private void unlock(Lock lock) {
            if (lock instanceof ReentrantLock && ((ReentrantLock) lock).isHeldByCurrentThread()) {
                lock.unlock();}}}
    public static void main(String[] args) throws InterruptedException {
        System.out.println("Simulating Deadlock:");
        Thread t1 = new DeadlockThread(lock1, lock2);
        Thread t2 = new DeadlockThread(lock2, lock3);
        Thread t3 = new DeadlockThread(lock3, lock1);
```

```
            for (Lock lock : locks) {
                if (!lock.tryLock()) return false;}
            return true;}
        private void unlockAll() {
            for (Lock lock : locks) {
                unlock(lock);}}
        private void unlock(Lock lock) {
            if (lock instanceof ReentrantLock && ((ReentrantLock) lock).isHeldByCurrentThread()) {
                lock.unlock();}}}
    public static void main(String[] args) throws InterruptedException {
        System.out.println("Simulating Deadlock:");
        Thread t1 = new DeadlockThread(lock1, lock2);
        Thread t2 = new DeadlockThread(lock2, lock3);
        Thread t3 = new DeadlockThread(lock3, lock1);
        t1.start(); t2.start(); t3.start();
        t1.join(); t2.join(); t3.join();
        System.out.println("\nSolving Deadlock with Consistent Lock Order:");
        Thread t1Resolved = new ResolvedThread(lock1, lock2, lock3);
        Thread t2Resolved = new ResolvedThread(lock1, lock2, lock3);
        Thread t3Resolved = new ResolvedThread(lock1, lock2, lock3);
        t1Resolved.start(); t2Resolved.start(); t3Resolved.start();
        t1Resolved.join(); t2Resolved.join(); t3Resolved.join();
        System.out.println("\nSolving Deadlock with tryLock():");
        Thread t1TryLock = new TryLockThread(lock1, lock2, lock3);
        Thread t2TryLock = new TryLockThread(lock1, lock2, lock3);
        Thread t3TryLock = new TryLockThread(lock1, lock2, lock3);
        t1TryLock.start(); t2TryLock.start(); t3TryLock.start();
        t1TryLock.join(); t2TryLock.join(); t3TryLock.join();}}
```

Output: