Exercicio 01

```
public class THREADS {

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

// TODO Auto-generated method stub


/*

 * O programa deve criar uma variável inteira compartilhada e iniciar duas

 * threads que tentam incrementar o valor dessa variável em um loop. Cada thread

 * deve incrementar o valor da variável várias vezes em sequência, e o resultado

 * final esperado é a soma do número de incrementos de ambas as threads. No

 * entanto, como as threads compartilham a mesma variável e não há sincronização

 * adequada, é possível que ocorra uma condição de corrida, em que as threads

 * executam em uma ordem imprevisível e o resultado final não corresponde à soma

 * dos incrementos.

 */

ValHolder ValHolder = new ValHolder();


Thread thread1 = new Thread() {

public void run() {

for (int i = 0; i < 2; i++) {

ValHolder.increment();

}

}

};
```

```
Thread thread2 = new Thread() {
public void run() {
for (int i = 0; i < 5; i++) {
ValHolder.increment();
}
}
};

Thread thread3 = new Thread() {
public void run() {
for (int i = 0; i < 6; i++) {
ValHolder.increment();
}
}
};

thread1.start();
thread2.start();
thread3.start();


try {
thread1.join();
thread2.join();
thread3.join();
```

```
} catch (Exception e) {
// TODO: handle exception
System.out.println("Erro:"+e.getMessage());
}

System.out.println(ValHolder.getVal());

}
}

class ValHolder {
private int val = 0;

public synchronized void increment() {
val++;
}

public synchronized int getVal() {
return this.val;
}

}
```

Exercicio 02

```
public class DeadlockExample {
    public static void main(String[] args) {
        Object lockA = new Object();
        Object lockB = new Object();
        Object lockC = new Object();

        // Criação das threads
        Thread threadA = new Thread(new DeadlockTask(lockA, lockB), "Thread A");
        Thread threadB = new Thread(new DeadlockTask(lockB, lockC), "Thread B");
        Thread threadC = new Thread(new DeadlockTask(lockC, lockA), "Thread C");

        // Inicia as threads
        threadA.start();
        threadB.start();
        threadC.start();
    }
}

class DeadlockTask implements Runnable {
    private final Object firstLock;
    private final Object secondLock;

    public DeadlockTask(Object firstLock, Object secondLock) {
        this.firstLock = firstLock;
        this.secondLock = secondLock;
    }
```

```
    @Override
    public void run() {
        String threadName = Thread.currentThread().getName();

        synchronized (firstLock) {
            System.out.println(threadName + " acquired lock on " + firstLock);

            try {
                Thread.sleep(100); // Simulando algum processamento
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            System.out.println(threadName + " trying to acquire lock on " + secondLock);

            synchronized (secondLock) {
                System.out.println(threadName + " acquired lock on " + secondLock);
            }
        }
    }
}
```