Task 7: Writing Thread-Safe Code, Immutable Objects
Design a thread-safe Counter class with increment and decrement methods. Then demonstrate its usage from multiple threads. Also, implement and use an immutable class to share data between threads.
ANS:

```
package com.Day23;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
public class Task7 {
  public static void main(String[] args) {
    Counter counter = new Counter();
    Runnable incrementTask = () -> {
      for (int i = 0; i < 1000; i++) {
        counter.increment();
      }
    };
    Runnable decrementTask = () -> {
      for (int i = 0; i < 1000; i++) {
        counter.decrement();
      }
    };
    Thread incrementThread1 = new Thread(incrementTask);
    Thread incrementThread2 = new Thread(incrementTask);
    Thread decrementThread1 = new Thread(decrementTask);
    Thread decrementThread2 = new Thread(decrementTask);
    incrementThread1.start();
    incrementThread2.start();
    decrementThread1.start();
    decrementThread2.start();
    try {
      incrementThread1.join();
      incrementThread2.join();
      decrementThread1.join();
      decrementThread2.join();
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
    System.out.println("Final count: " + counter.getCount());
    // Using ImmutableData to share data between threads
    ImmutableData immutableData = new ImmutableData(42);
    Runnable readDataTask = () -> {
      System.out.println("Immutable data value: " + immutableData.getValue());
    };
    Thread readDataThread1 = new Thread(readDataTask);
    Thread readDataThread2 = new Thread(readDataTask);
    readDataThread1.start();
    readDataThread2.start();
  }
  static class Counter {
    private int count;
    private final Lock lock = new ReentrantLock();
    public Counter() {
```

```
            this.count = 0;
        }
        public void increment() {
            lock.lock();
            try {
                count++;
            } finally {
                lock.unlock();
            }
        }
        public void decrement() {
            lock.lock();
            try {
                count--;
            } finally {
                lock.unlock();
            }
        }
        public int getCount() {
            return count;
        }
    }
    static final class ImmutableData {
        private final int value;
        public ImmutableData(int value) {
            this.value = value;
        }
        public int getValue() {
            return value;
        }
    }
}
```