DAY 23:

ASSIGNMENTS 7:

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.

```
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;


public class Main {
  public static void main(String[] args) throws InterruptedException {
    Counter counter = new Counter();
    ImmutableData immutableData = new ImmutableData(10);


    Thread incrementThread = new Thread(() -> {
      for (int i = 0; i < 1000; i++) {
        counter.increment();
      }
    });


    Thread decrementThread = new Thread(() -> {
      for (int i = 0; i < 1000; i++) {
        counter.decrement();
```

```
          }
        });

        Thread readThread = new Thread(() -> {
            System.out.println("Immutable Data Value: " +
immutableData.getValue());
        });

        incrementThread.start();
        decrementThread.start();
        readThread.start();

        incrementThread.join();
        decrementThread.join();
        readThread.join();

        System.out.println("Counter Value: " + counter.getCount());
    }
}

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;
    }
}

final class ImmutableData {
    private final int value;
    public ImmutableData(int value) {
        this.value = value;
    }

    public int getValue() {
        return value;
    }
}
```

Explanation

1. *Counter Class*:

   - The Counter class uses a ReentrantLock to ensure thread safety for increment, decrement, and getCount methods.

   - The lock.lock() method acquires the lock, and the finally block with lock.unlock() ensures that the lock is released even if an exception occurs.

2. *ImmutableData Class*:

   - This class is immutable because its state (the value field) cannot be changed after it is created. This makes it inherently thread-safe.

3. *Main Class*:

   - Multiple threads are created to increment and decrement the counter.

   - The join method ensures that the main thread waits for these threads to finish before printing the final counter value.

   - Immutable data is shared across multiple threads which read and print its value.

This code demonstrates both thread-safe mutable state management and safe sharing of immutable data between threads in Java.