Exercise 1

exercise 1.1.

exercise 1.1.1

> Task :app:TestLongCounterExperiments.main()
> Count is 19852859 and should be 20000000

The output is not the expected output.

exercise 1.1.2

> Task :app:TestLongCounterExperiments.main()
> Count is 200 and should be 200

The output is the expected 200.

exercise 1.1.3

Using `count += 1`, I get

> Task :app:TestLongCounterExperiments.main()
> Count is 19856928 and should be 20000000

Using `count ++`, I get

> Task :app:TestLongCounterExperiments.main()
> Count is 19738226 and should be 20000000

Make no difference, since both of the two methods are not atomic.

exercise 1.1.4

change the code to:

```
public void increment() {
      lock.lock();
      count = count + 1;
      lock.unlock();
    }
```

In this program, there is a data race between two thread in `increment` method since:

- Access a shared memory location
- Both thread writes to the memory location

And the result of the computation depends on the interleavings of the operations, thus there is a race condition. By using `ReentrantLock`, we can guarantee that only one thread can access it at a time.

exercise 1.1.5

A critical section is a part of the program that only one thread can execute at the same time. In this program, the critical section is

```
public void increment() {
      lock.lock();
      count = count + 1;
      lock.unlock();
    }
```

The current solution contains the least number of lines of code.