Exercise 2.1

Question 1

Yes

Question 2

Yes

Question 3

No, and you shouldn't rely on it. The changes to the variable is not made visible to the other thread. By making both methods synchronized, we rely on the visibility guarantees given by locking the object.

Question 4

The thread still terminates as expected. By using a lock we get two guarantees: visibility and atomicity. Using the keyword `volatile` we are only gauranteed visibility. In this case we only need the visibility guarantee, so we can simply use `volatile` instead of locking.

Exercise 2.2

Question 1

```
Sequential result:     664579

real    0m6.968s
user    0m6.967s
sys     0m0.037s
```

Question 2

The 10 thread version executes faster:

```
Parallel10 result:     664579

real    0m1.906s
user    0m12.045s
sys     0m0.052s
```

If we look at the real execution time the code is now 3.7 times faster. But if we consider the time spent by all the threads in total, the execution time almost doubled.

Question 3

No, in this particular case we only got 663,733, thereby missing 846 primes:

```
Parallel2  result:     663733

real    0m4.467s
user    0m7.119s
sys     0m0.035s
```

When `increment` isn't synchronized we risk getting race conditions, as the incrementation isn't atomic anymore.

Question 4

In this particular case it doesn't matter. `get` is not called while more than one thread is running. All the incrementations are done before the call to `get`, so we don't risk getting race conditions here.

Exercise 2.3

Question 1

```
Total number of factors is  18703729

real    0m7.345s
user    0m7.340s
sys     0m0.041s
```

Question 3

Yes:

```
Total number of factors is  18703729

real    0m2.283s
user    0m14.700s
sys     0m0.104s
```

Question 4

No, we need `addAndGet` to be executed atomically.

Question 5

There is a slight increase in performance:

```
Total number of factors is  18703729

real    0m2.145s
user    0m13.886s
sys     0m0.062s
```

We do not need to declare the `AtomicInteger` `final`, since the class is already thread-safe, but it is good practice to do it, as it makes it easy to argue about.

Exercise 2.4

Question 1

It is important to make the `cache` variable volatile to ensure that all threads have the same object, i.e. the most current version of the cache. Leaving out the `volatile` keyword will not produce an incorrect answer in this case, but it will ruin the original intention of the cache.

Question 2

Both of the fields in `OneValueCache` needs to be final in order to make the object immutable. This is also ensured by not making a setter for the fields.