CS1760: Multiprocessor Synchronization **Due: 20 September 2018 12:59 PM**
Professor: Maurice Herlihy *HTA: Jonathan Lister*

Programming Assignment 1 - Counter

This programming assignment requires you to implement three `Counter` objects that are each subject to different constraints. The stencil code for this assignment is located at the course's stencil directory at `/course/cs1760/stencil/counter/`, and can be installed with course scripts by typing

```
cs1760_install counter
```

into the terminal. We have provided a `Counter` interface that is based on the `Counter` object discussed in lecture. It offers two methods: `inc()`, which might be used to increment the value of the counter, and `getCount()`, which might be used to obtain the value of the counter. You are required to provide definitions for these methods in three different classes that implement this interface:

- `MagicCounter:` This counter object must consistently behave the same way regardless of whether it is accessed by a single thread or multiple threads (that is, it must return the same value after threads make any number of `inc()` calls sequentially or concurrently.) However, you are not allowed to use *any* multithreaded synchronization constructs in your implementation.
- `SingleThreadedCounter:` This type of counter is required to consistently return a counter value that matches the number of `inc()` calls at the end only when it is not accessed concurrently. Non-concurrent access should return the correct value, but we expect it to fail on concurrent access. (Is the value low or high? Why is this?)
- `MultiThreadedCounter:` This type of counter is required to consistently return a counter value that matches the number of `inc()` calls regardless of whether it is being accessed by a single thread or multiple threads concurrently; however, you are free to use multithreaded constructs to ensure its methods are thread safe.

Once you copy the stencil code into your directory, you'll be ready to go! The files you need to modify are:

```
MagicCounter.java
SingleThreadedCounter.java
MultiThreadedCounter.java
TestCounter.java
CounterTester.java
```

Note that a portion of this assignment's total score is reserved for testing. Passing basic functionality tests, located in `BasicTestCounter.java` and `BasicCounterTester.java`,

will reward you with some points; however, we also expect you to write any additional tests you deem necessary.
Failing basic functionality tests does not necessarily imply that you will receive no credit for the assignment. At the same time, the course staff will be unable to conduct a rigorous inspection of non-functional code to award partial credit.

Here are some tips to get you started on the assignment:

1. Be sure to `read` the comments in each file to get a sense of the interactions between different components of the program.

2. As stated in the comments in `MagicCounter.java`, please consult the TAs if you are unsure about whether you can use a certain construct in your implementation.

3. Follow this link to learn about the details and conventions around Java's thread objects. Most importantly, make sure you understand the distinction between `thread.start()` and `thread.run()`!
   https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html

4. When thinking about how something behaves, we can consider certain actions to happen at a specific point in time - actions that completely occur at a specific instant are called **linearizable**. You do not need to discuss this in your answer for this question, but it may help you to think about how events occur. In Java for this assignment, we can treat the following as linearizable operations (this is not an exhaustive list, but should grant some insight into a counter):
   - Assigning a specific value to a variable.
   - Reading a variable's value.

   If you don't want anything unexpected to happen between two operations which don't happen at the exact same time, use synchronization constructs to make sure no other thread can mess with what's happening.

5. Please make sure the final counter value is divisible by the number of threads when you run it. The support code expects this and if you don't do this, it will round down to the nearest multiple of `numThr` and you will get a weird answer.

Once you're finished with this assignment, run

```
cs1760_handin counter
```

to hand it in!