Exercises Week 3

Exercise 1: `FindMax.java` / `FindSum.java`

a) `Semaphore` vs `synchronized`

Change your solution from lasts week's exercise `FindMax.java` / `FindSum.java` (your choice :D), and use one `Semaphore` object to replace the `synchronized` method. Compare the runtimes of the algorithms using the `synchronized` method versus the algorithms using `Semaphore`. Look especially at the runtimes of the methods using the `synchronized` method for each element, versus the methods using the `Semaphore` for each element. What do you notice? Is `Semaphore` or `synchronized` the most efficient?

b) `Semaphore` as a barrier

Add another `Semaphore` object to replace the `CyclicBarrier`. This means that here, instead of making it the access point of a shared resource, make it act as a barrier. Hint: `Semaphore`'s are allowed to take negative integers in their constructor.

CHALLENGE! `DivisibleBy.java`

Highlight the numbers divisible by `n`, where `n` is an integer greater than 1. Create two different `Runnable` classes, each with their own `run()` method. One of the `run()` methods should be printing out the numbers divisible by `n` and the other one should be printing out the numbers *not* divisble by `n`.

By using two `Semaphore` objects, force the threads to print out the numbers in sequential order (0, 1, 2, …). This way you will get a sequence of numbers where only the ones divisible by `n` are highlighted.

If you want another challenge, you can allow the user to choose both the start and end point (as opposed to only the end point) of your number sequence. You can assume that you only get positive integers.

Fun tip: You can highlight the numbers by using colors in the print statements. Just remember to reset! Colors can be found here.

Challenge accepted?

```java
if (challengeAccepted) System.out.println("You are awesome!");
```