Homework (Week 4)

Submission: Due on Friday, 1st of July, 11am Sydney Time. Please submit using the CSE Give System either online or via this command on a CSE terminal:

```
give cs3151 hw4 *Semaphore.java CigaretteSmokers.java
```

Late submissions are accepted up to five days after the deadline, but at a penalty: 5% off your total mark per day.

You may also submit any other files with a `.java` extension, but these are the only ones we need.

Download the code to get started with this homework.

To avoid name conflicts, `signal` is here called `V()` and `wait` is called `P()`.

Cigarette Smokers Problem (6 marks)

Three smokers are rolling their own cigarettes and smoking them. In order to do so, they need three resources: paper, tobacco, and matches. Each smoker has their own limitless supply of one of the three ingredients. For the other two ingredients, they must acquire them from an *Agent*. They can ask for an Agent to distribute resources to them by signalling on the `agent` semaphore. One of the `Agent` threads will then awaken, and produce two of the three different resources by signalling on two of the three semaphores `tobacco`, `paper` and `match`. Note that the smoker has no control over which `Agent` thread awakens and which resources are produced.

The existing code where smokers get resources directly from the agents is highly vulnerable to a deadlock scenario. In this exercise, we will implement a solution to this problem that ensures progress by introducing a "middle-man" between the agents and smokers.

Step One

Introduce a dedicated semaphore for each smoker, and the smoker will just wait on that semaphore, smoke a cigarette, and signal an agent forever.

Step Two

Add a thread called a *pusher* for each of the three resources. The pusher will perpetually:

1. Wait for its resource to become available.
2. Update some shared state (shared with the other pushers) regarding what resources are available.
3. If two of the three resources are available, update the shared state to remove those resources, and signal the smoker that requires those two resources, to indicate that they may smoke the cigarette.

Caution

You must make the loop body of the pusher a critical section - mutual exclusion is required when accessing the shared state. You can accomplish this by adding another semaphore for the pushers to use.

Semaphores with Monitors (6 marks)

In the `ProducerConsumer` module you will find an implementation of the Producer-Consumer problem we analysed in class. This (and the Cigarette Smokers example from earlier) use a semaphore class called `JavaSemaphore` that wraps around Java's built-in semaphores. We have two other classes, `BusyWaitSemaphore` and `WeakSemaphore`, without implementations added.

Step One

Using **only** Java's built-in concurrency primitives, specifically `synchronized` blocks/methods and `Thread.yield()`, implement a busy-wait semaphore in `BusyWaitSemaphore`. You can test it using `ProducerConsumer` or `CigaretteSmokers`. (3 marks)

Step Two

Using **only** Java's built-in concurrency primitives and monitor-like constructs, specifically `synchronized` blocks, `Object.wait()` and `Object.notify()`, implement a weak semaphore in `WeakSemaphore`. You can test it using the same examples. (3 marks)

For a Meaningless Brownie Point

Using **only** Java's built-in concurrency primitives and monitor-like constructs, as well as an `ArrayDeque` of `Object`, implement a strong semaphore in `StrongSemaphore`. You may use `Object.notifyAll()` so long as the first process to *leave* its await is the first process in the queue.

Hint: your solution probably shouldn't depend on a `Thread.yield()` for correctness. `Thread.yield()` is a hint to the scheduler that a thread is willing to yield its

scheduling slice. There's no guarantee that the scheduler will take the hint.