Chapter 4 - exercises

4.1 Recursive Locking in Java

Once a thread has acquired the lock on an object by executing a synchronized method, that method may itself call another synchronized method from the same object (directly or indirectly) without having to wait to acquire the lock again. The lock counts how many times it has been acquired by the same thread and does not allow another thread to access the object until there has been an equivalent number of releases. This locking strategy is sometimes termed *recursive* locking since it permits recursive synchronized methods. For example:

```
public synchronized void increment(int n) {
   if (n>0) {
        ++value;
        increment(n-1);
   } else return;
}
```

This is a rather unlikely recursive version of a method which increments `value` by `n`. If locking in Java was not recursive, it would cause a calling thread to block resulting in a deadlock.

Given the following declarations:

```
const N = 3
range P = 1..2   //thread identities
range C = 0..N   //counter range for lock
```

Model a Java recursive lock as the *FSP* process `RECURSIVE_LOCK` with the alphabet `{acquire[p:P],release[p:P]}`. `acquire[p]` acquires the lock for thread `p`.