Chapter 4 - solutions to selected exercises

4.1 Recursive Locking in Java

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

RECURSIVE_LOCK   = (acquire[p:P] -> LOCKED[p][0]),
LOCKED[p:P][c:C] = (when c<N  acquire[p] -> LOCKED[p][c+1]
                    |when c>0  release[p] -> LOCKED[p][c-1]
                    |when c==0 release[p] -> RECURSIVE_LOCK
                    ).
```

Chapter 5 - solutions to selected exercises

5.1

```
class OneBuf {
  Object slot = null;

  public synchronized void put (Object o) {
      while (slot!=null)
           try{wait();} catch(InterruptedException e){};
      slot= o;
      notify();
  }

  public synchronized Object get () {
      while (slot==null)
        try{wait();} catch(InterruptedException e){};
      notify();
      Object o = slot;
      slot=null;
      return o;
  }

}
```

5.2

- same as bounded buffer notes.

- Still require mutual exclusion as need atomic test and assignment. Scalar data types are atomic with respect to single operations such as addition & assignment.

5.3

- CONTROL should be monitor.

5.4

```
class Barrier {
  int n;
  int blocked = 0;
  Barrier(int n) {this.n = n;}

  public synchronized void sync()
               throws InterruptedException {
     ++blocked;
     if (blocked < n)
          wait();
      else {
          notifyAll();
          blocked=0;
         }
    }
}
```

5.5

```
BANKACCOUNT    = BALANCE[0],
BALANCE[bal:M] = (when (bal>0)
                    withdraw[a:1..bal]->BALANCE[bal-a]
                 |deposit[a:M] -> BALANCE[bal+a]
                 ),
BALANCE[Max+1..2*Max] = ERROR.  //Overflow
```