1. (25% of exam) Rewrite class ConsistentCounter to make it thread safe without using any intrinsic locks, extrinsic locks, or any form of waiting. This problem requires effective use of library classes. The method signatures must stay the same, but the implementation of these methods and data fields may change.

```
public interface IntegerPredicate {          // Do not change this interface.
    /** isSatisfied returns true if integer predicate test is satisfied, else returns false. **/
    public boolean isSatisfied(int valueToTest) ;
}
import java.util.concurrent.atomic.* ;       /* import state is optional in the exam. */
public class ConsistentCounter {
    private final /* int */ AtomicInteger counter ;
    public ConsistentCounter(int initialValue) {
        counter = new AtomicInteger(initialValue) ;
    }
    public int getValue() {
        return counter.get() ;
    }
    public int addThenGetValue(int increment) {
        /* counter += increment ; */
        return counter.addAndGet(increment) ;
    }
    public boolean guardedAdd(IntegerPredicate guard, int increment) {
        boolean result = false ;
        int original = counter.get();
        if (guard.isSatisfied(/* counter */ original)) {
            result = counter.compareAndSet(original, original+increment);
            /* counter += increment ;
               result = true ;
            */
        }
        return result ;
    }
}
```

**/* A more interesting solution would be to add two additional atomics as guards and use them like a ticket number at a deli. The following code would go into both of the latter two mutator methods. The field declarations for getTicket and callTicket would go outside the method code as usual.
*/**

```
private final AtomicLong getTicket = new AtomicLong(0L);
private final AtomicLong callTicket = new AtomicLong(0L);

long myticket = getTicket .getAndIncrement();
while (myticket != callTicket.get()) { }     /* Idle spin is very brief and guaranteed fair, no starvation. */
```

DO THE MUTATION WITHIN addThenGetValue or guardedAdd. The counter field can now be "private volatile int counter" and guardedAdd no longer needs to compareAndSet after checking the guard.

```
callTicket.incrementAndGet();                /* Pass ticket to next thread in FIFO order. */
```