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.

```java
import java.util.concurrent.atomic.*;

public interface IntegerPredicate {
    /** isSatisfied returns true if integer predicate test is satisfied, else returns false. **/
    public boolean isSatisfied(int valueToTest);
}

public class ConsistentCounter {
    private final AtomicInteger counter;
    public ConsistentCounter(int initialValue) {
        counter = new AtomicInteger(initialValue);
    }
    public int getValue() {
        return counter.get();
    }
    public int addThenGetValue(int increment) {
        return counter.addAndGet(increment);
    }

    public boolean guardedAdd(IntegerPredicate guard, int increment) {
        boolean result = false;
        int original = counter.get();
        if (guard.isSatisfied(original)) {
            result = counter.compareAndSet(original, original + increment);
        }
        return result;
    }

    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. */
    }

    public boolean guardedAdd(IntegerPredicate guard, int increment) {
        boolean result = false;
        int original = counter.get();
        if (guard.isSatisfied(original)) {
            result = counter.compareAndSet(original, original + increment);
        }
        return result;
    }

    // More interesting solution:
    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. */
    }

    public boolean guardedAdd(IntegerPredicate guard, int increment) {
        boolean result = false;
        int original = counter.get();
        if (guard.isSatisfied(original)) {
            result = counter.compareAndSet(original, original + increment);
        }
        return result;
    }

    // Break guardedAdd and getTicket for order.
    callTicket.incrementAndGet();
    /* Pass ticket to next thread in FIFO order. */
}
```

CSC 480 Multiprocessor Programming, a problem from last year’s midterm that uses atomic variables.