2.4 Lecture Summary

2.4 Atomic Variables

Lecture Summary: In this lecture, we studied *Atomic Variables*, an important special case of object-based isolation which can be very efficiently implemented on modern computer systems. In the example given in the lecture, we have multiple threads processing an array, each using object-based isolation to safely increment a shared object, *cur*, to compute an index *j* which can then be used by the thread to access a thread-specific element of the array.

However, instead of using object-based isolation, we can declare the index *cur* to be an *Atomic Integer* variable and use an atomic operation called *getAndAdd()* to atomically read the current value of *cur* and increment its value by 1. Thus, `j = cur.getAndAdd(1)` has the same semantics as `isolated (cur) { j = cur; cur = cur + 1; }` but is implemented much more efficiently using hardware support on today's machines.

Another example that we studied in the lecture concerns *Atomic Reference* variables, which are reference variables that can be atomically read and modified using methods such as *compareAndSet()*. If we have an atomic reference *ref*, then the call to *ref.compareAndSet(expected, new)* will compare the value of *ref* to *expected*, and if they are the same, set the value of *ref* to *new* and return *true*. This all occurs in one atomic operation that cannot be interrupted by any other methods invoked on the *ref* object. If *ref* and *expected* have different values, *compareAndSet()* will not modify anything and will simply return *false*.

Optional Reading:

1. Tutorial on Atomic Integers in Java

2. Article in Java theory and practice series on Going atomic

3. Wikipedia article on Atomic Wrapper Classes in Java