**Package java.util.concurrent.atomic Description**

A small toolkit of classes that **support lock-free thread-safe programming on single variables**. In essence, the classes in this package extend the notion of **volatile values, fields, and array** elements to those that also provide an **atomic conditional update operation** of the form:

**boolean compareAndSet(expectedValue, updateValue);**

This method (which varies in argument types across different classes) atomically sets a variable to the **updateValue** if it currently holds the **expectedValue**, reporting **true** **on success**.

**AtomicInteger**

While **a single store to** or **a single load from** **an** ordinary **int is atomic** in Java, we cannot atomically, say, increment it. Doing so would require us to **first load the value**, then **compute the new value** depending on it and then **store the new value back**. But between the two accesses, another thread might have modified the value. **AtomicInteger** provides operations like [**getAndIncrement**](https://docs.oracle.com/javase/8/docs/api/index.html?java/util/concurrent/atomic/AtomicInteger.html) that can be used for this purpose **without having to use a lock**.

There are two **main uses of AtomicInteger**:

1. **As an atomic counter** (**incrementAndGet()**, etc) that **can be used by many threads concurrently**
2. **As a primitive that supports** [**compare-and-swap**](http://en.wikipedia.org/wiki/Compare-and-swap) instruction (**compareAndSet()**) to **implement non-blocking algorithms**.

