**AtomicInteger in Java**

AUGUST 10, 2016 BY [PANKAJ](http://www.journaldev.com/author/pankaj) [19 COMMENTS](http://www.journaldev.com/1095/atomicinteger-java#comments)

Today we will look into AtomicInteger in Java. **Atomic operations** are performed in a single unit of task without interference from other operations. Atomic operations are necessity in multi-threaded environment to avoid data inconsistency.

**AtomicInteger**

Let’s create a simple multi-threaded program where every thread increments the shared count variable 4 times. So if there are two threads, after they finish count value should be 8.

JavaAtomic.java

package com.journaldev.concurrency;

public class JavaAtomic {

public static void main(String[] args) throws InterruptedException {

ProcessingThread pt = new ProcessingThread();

Thread t1 = new Thread(pt, "t1");

t1.start();

Thread t2 = new Thread(pt, "t2");

t2.start();

t1.join();

t2.join();

System.out.println("Processing count=" + pt.getCount());

}

}

class ProcessingThread implements Runnable {

private int count;

@Override

public void run() {

for (int i = 1; i < 5; i++) {

processSomething(i);

count++;

}

}

public int getCount() {

return this.count;

}

private void processSomething(int i) {

// processing some job

try {

Thread.sleep(i \* 1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

If you will run above program, you will notice that count value varies between 5,6,7,8. The reason is because **count++** is not an atomic operation. So by the time one threads read it's value and increment it by one, other thread has read the older value leading to wrong result.

To solve this issue, we will have to make sure that increment operation on count is atomic, we can do that using [Synchronization](http://www.journaldev.com/1061/thread-safety-in-java) but Java 5 java.util.concurrent.atomic provides wrapper classes for int and long that can be used to achieve this atomic operation without usage of Synchronization.

**Java AtomicInteger Example**

Here is the updated program that will always output count value as 8 because AtomicInteger method incrementAndGet() atomically increments the current value by one.

package com.journaldev.concurrency;

import java.util.concurrent.atomic.AtomicInteger;

public class JavaAtomic {

public static void main(String[] args) throws InterruptedException {

ProcessingThread pt = new ProcessingThread();

Thread t1 = new Thread(pt, "t1");

t1.start();

Thread t2 = new Thread(pt, "t2");

t2.start();

t1.join();

t2.join();

System.out.println("Processing count=" + pt.getCount());

}

}

class ProcessingThread implements Runnable {

private AtomicInteger count = new AtomicInteger();

@Override

public void run() {

for (int i = 1; i < 5; i++) {

processSomething(i);

count.incrementAndGet();

}

}

public int getCount() {

return this.count.get();

}

private void processSomething(int i) {

// processing some job

try {

Thread.sleep(i \* 1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

Benefits of using Concurrency classes for atomic operation is that we don't need to worry about synchronization. This improves code readability and chance of errors are reduced. Also atomic operation concurrency classes are assumed to be more efficient that synchronization which involves locking resources.