**Advanced Java: Multi-threading Part 3 -- The Synchronized Keyword**

https://www.youtube.com/watch?v=lotAYC3hLVo

**class** apples {

**private** **int** count= 0;

**public** **synchronized** **void** Increment(){

//"synchronized" gives turns to each thread

// i.e. t2 can't use increment method if t1 is using it

count++;

}

**public** **static** **void** main(String args[]){

apples app = **new** apples();

app.doWork();

}

**public** **void** doWork(){

Thread t1 = **new** Thread(**new** Runnable(){

**public** **void** run(){

**for**(**int** i=0;i<10000;i++){

Increment();

}

}

});

Thread t2 = **new** Thread(**new** Runnable(){

**public** **void** run(){

**for**(**int** i=0;i<10000;i++){

Increment();

}

}

});

t1.start();

t2.start();

**try** {

t1.join(); //reason why we used join is simple

t2.join(); //To wait for the threads to finish before giving final answer(count)

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.*out*.println("Count is: "+count);

}

}

**Result:**

Count is: 20000

**Explaianation:** There were several problems initially.

1. Before we put the “*t1.join*” and “*t2.join()*”, the result was giving *Count is: 0*.

**Reason**: Just as the threads were starting, the count value was displayed.

**Solution**: We put in *t1/t2.join()* in order to wait till the threads finish.

1. The answer was often 20000. But in some cases, the answer was also random numbers

(e.g. 11520, 12600)

**Reason**: There were times when thread 2 or thread 1 was faster than the other. If one of then was incrementing once, the other might have already incremented twice, resulting in few skipped increments.

**Solution**: The programmer then planned to use “**synchronized**” to fix this problem. In order to utilize the “**synchronized**” keyword, we replaced “*count++*” with “*Increment()*”.