<https://stackoverflow.com/questions/44963842/synchronized-block-will-lock-the-whole-object-or-the-method-alone/44964544>

# [Synchronized block will lock the whole object or the method alone?](https://stackoverflow.com/questions/44963842/synchronized-block-will-lock-the-whole-object-or-the-method-alone)

Q:

I have multiple methods in a class and most of the methods are having critical sections(shared data). So I made those methods as synchronized. Say thread t1 is running one of the synchronized block . At the same time thread t2 can access the other methods critical section ?

class Sample{

synchronized public void method1(){

}

synchronized public void method2(){

}

synchronized public void method3(){

}

public void method4(){

}

}

A:

<https://stackoverflow.com/a/44964544/6706875>

synchronized is always locking on an object. In case of a synchronized method, the object is this. So basically these two methods do the same:

synchronized public void method1() {

// do something

}

and

public void method1() {

synchronized(this){

// do something

}

}

As long as one thread has the lock on the lock object, no other thread can lock this object. So in your example, the synchronized methods (one, two and three) can never be executed at the same time. method4 is not synchronized, so it can access the object anytime.

If you want a more fine-grained locking, because method1and method2 should be exclusive and method3and method4 you could use for example something like this:

class Sample{

private final Object lock1 = new Object();

private final Object lock2 = new Object();

public void method1(){

synchronized(lock1) {

// do something

}

}

public void method2(){

synchronized(lock1) {

// do something

}

}

public void method3(){

synchronized(lock2) {

// do something

}

}

public void method4(){

synchronized(lock2) {

// do something

}

}

}

You can then even use the synchonized(lock) method to just wrap the statements that need to be synchronized, not the whole method:

public void method() {

// some code

synchronized(lock) {

// code that must be synchronized

}

// some other code

}

With this approach you can keep the lock duration to a minimum.

This is correct, but it should be noted, that in general synchronized should be avoided. It is a brutish method to archive thread-safety and in most cases Concurrent classes are much more suited for the task. – [TwoThe](https://stackoverflow.com/users/2798291/twothe) [Jul 7 '17 at 7:15](https://stackoverflow.com/questions/44963842/synchronized-block-will-lock-the-whole-object-or-the-method-alone#comment76904297_44964544)

* @TwoThe: I totally agree, I prefer immutable objects, event based and / or stream based architectures etc., but that is a different story. – [P.J.Meisch](https://stackoverflow.com/users/4393565/p-j-meisch) [Jul 7 '17 at 7:18](https://stackoverflow.com/questions/44963842/synchronized-block-will-lock-the-whole-object-or-the-method-alone#comment76904392_44964544)
* It should probably be also noted that synchronized static methods lock on the Class object of the class.

Best example:

<https://github.com/lampardchelsea/hello-world/blob/master/leetcode/Concurrency/DesignBoundedBlockingQueue.java>