Note:

class X{

synchronized void m1(){}

synchronized void m2(){}

void m3(){}

}

Keypoints :

1. If t1 thread invokes m1() then on the Object X lock will be applied
2. If t2 thread invokes m2() then m2() cant be called because lock of X object is with m1
3. If thread t3 invokes m3() then execution will happen because m3() is not synchronized

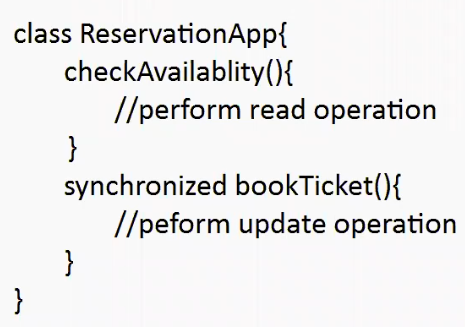
Lock concept is applied at the object level not at the method level

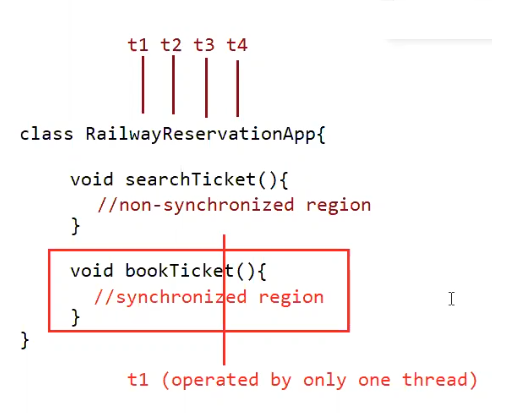
* Every object in java has a unique lock. Whenever we are using synchronized keyword then only lock concept will come into picture
* If a thread wants to execute any synchronized method on the object first it has to get the lock of the object. once the thread got the lock of that object then it allows to execute any synchronized method on that object.
* If the synchronized method execution completes then automatically thread lock releases.
* While a thread executing any synchronized method the remaining threads are not allowed to execute any synchronized method on that object simultaneously.
* But the remaining threads are allowed to execute any non-synchronized methods simultaneously ( lock concept is implemented based on the object but not based on the method)

Note: every object will have 2 area ( synchronized and non- synchronized)

Synchronized area –> write the code only to perform update, insert, delete.

Non-synchronized area -> write the code only to perform selection operation





Eg: Thread\_Synchronization\_With\_Multiple\_Objects

In the above case we get irregular output , because if multiple threads are operating on multiple objects there is no impact of synchronization.

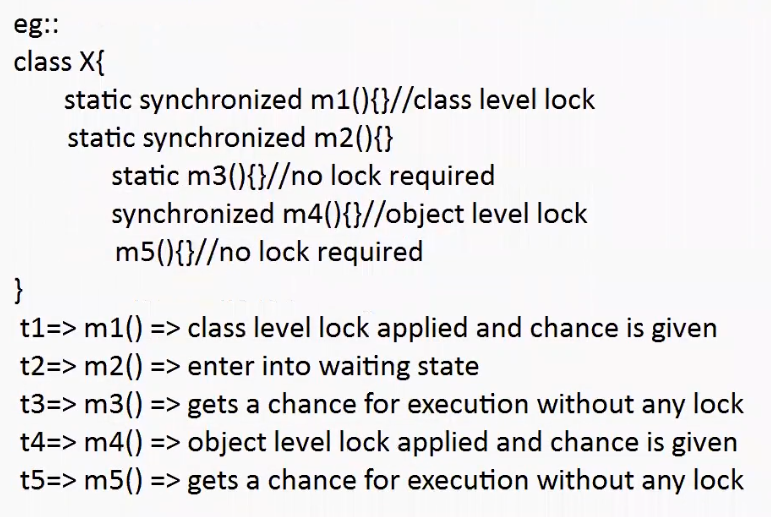
Conclusion:

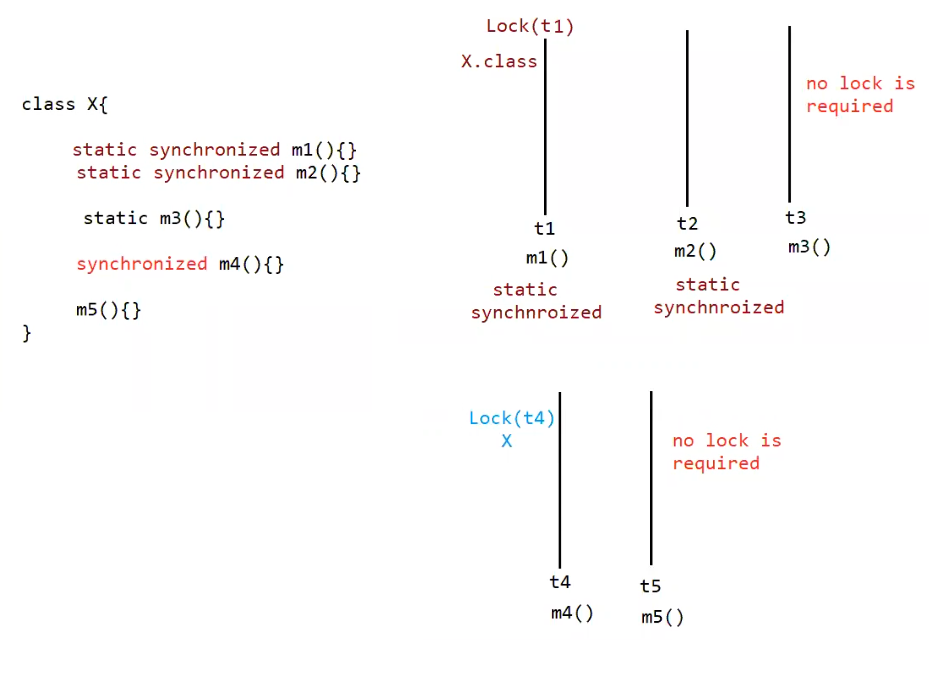
If multiple threads are operating on multiple objects then there is no impact of synchronization

If multiple threads are operating on single object then synchronized concept is applicable.

Class-level lock:

* Every class in java has a unique level lock
* If a thread wants to execute static synchronized method then the thread requires class-level lock.
* While a thread executing any static synchronized method the remaining methods are not allowed to execute any static synchronized method of that class simultaneously
* But the remaining threads are allowed to execute normal synchronized methods, normal static methods, normal instance, methods simultaneously.
* Class level lock and object level lock both are different and there is no relationship between these two





Eg: Static\_Synchronized\_Method

// go through the code

Eg: Synchronized\_Method\_Without\_Static

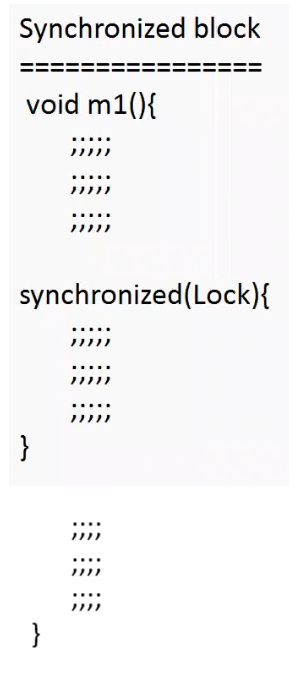
// go through the code

Eg: Without\_Static\_And\_Synchronized

// go through the code

Eg: One\_Static\_Synchronized\_Method\_Normal\_Static\_Method

// go through the code



If few lines of a code is required to get synchronized then it is not recommended to make method only as synchronized. if the whole method is made as synchronized then the thread performance will low.

To resolve this problem we use synchronized block, due to synchronized block performance will be improved.

Case study:

1. If a thread got a lock of current object, then it is allowed to execute that block

synchronized (this){

;;;;;

}

1. To get a lock of particular object

synchronized(B){

;;;;;

;;;;;

}

If a thread got a lock of particular object B, then it is allowed to execute that block

1. To get a class level lock we have to declare synchronized block as follows

synchronized(B){

;;;;;

;;;;;

}

If a thread gets class level lock, then it is allowed to execute that block.