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

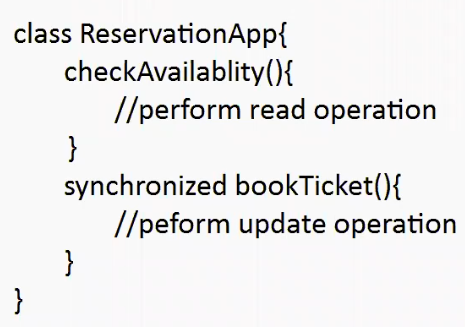
Lock concept is applied at the object level not at the method level

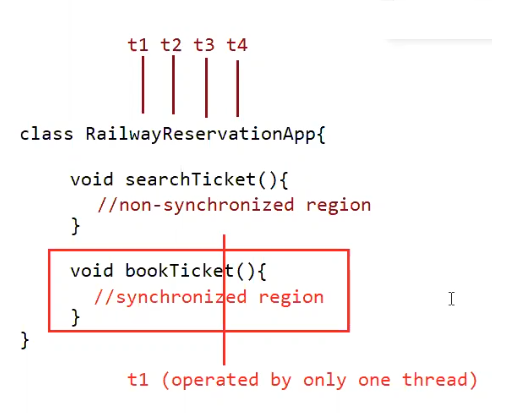
* Every object in java has a unique lock. Whenever we are using synchronized keyword then only lock concept will come into picture
* If a thread wants to execute any synchronized method on the object first it has to get the lock of the object. once the thread got the lock of that object then it allows 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.

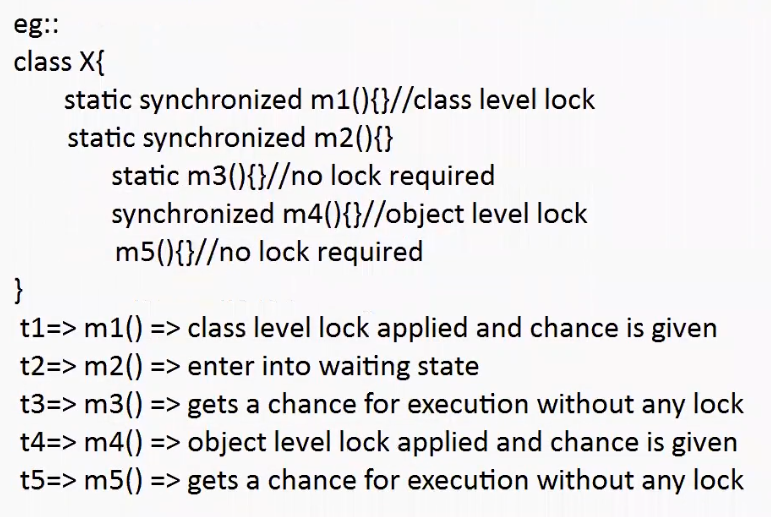
Conclusion:

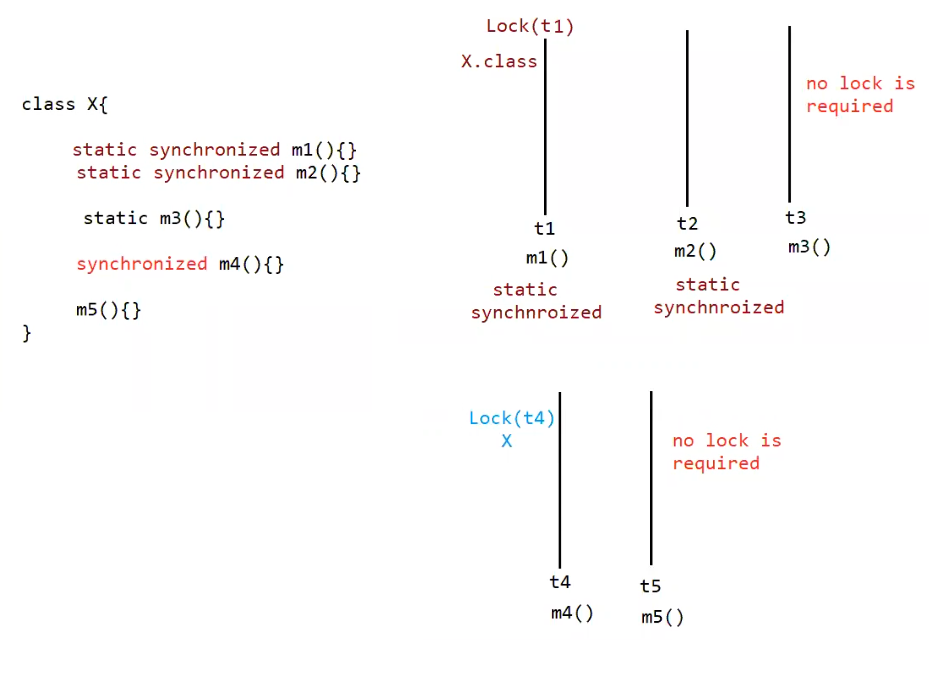
If multiple threads are operating on multiple objects then there is no impact of synchronization

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

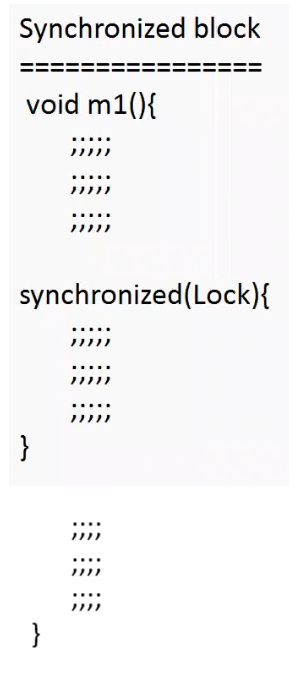
// go through the code

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){

;;;;;

}

Eg: Synchronized\_Block\_Current\_Object\_This

// go through the code

Eg: Synchronized\_Block\_Using\_This\_On\_Different\_Objects

// go through the code

Irregular output because two object and two threads acting on two different objects.

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(Display.class){

;;;;;

;;;;;

}

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

Eg: Synchronized\_At\_Class\_Level

Note:

* 2 objects , 2 threads , but the thread which gets a chance will make a class-level lock . so output is regular
* Lock concept is applicable only for objects and class type ,but not for primitives . it we try to do it , it would result in compile time error.
* Eg : int x = 10;

synchronized( x ) { // C.E unexcepcted :int required : reference

}

Inter-Thread communication:

Two threads can communicate each other with the help of

1. notify()
2. notifyAll()
3. wait()

notify() : thread which is performing updation should call notify() , so the waiting thread will get notification ,so it will continue its execution with the updated items.

wait() : thread which is expecting notification/updation should call wait() , immediately the thread will enter into waiting state.

* wait() , notify() , notifyAll() is present in Object class, but not Thread class why?
* Thread will call wait() , notify() , notifyAll() on any type of objects like Student , Customer, Engineer
* If a thread wants to call wait() , notify() , notifyAll() then compulsorily the thread should be the owner of the object otherwise it would result in “ IllegalMonitorStateException”
* We say thread to be the owner of the object if thread has the lock of the object
* It means these methods are a part of the synchronized block (or) synchronized method , if we try to use outside synchronized area it would result in runtime exception called IllegalMonitorStateException.

If a thread calls wait() on any object, then first it immediately releases the lock on that object and enters into waiting state

If a thread calls notify() on any object then he may or may not release the lock in that object immediatey.

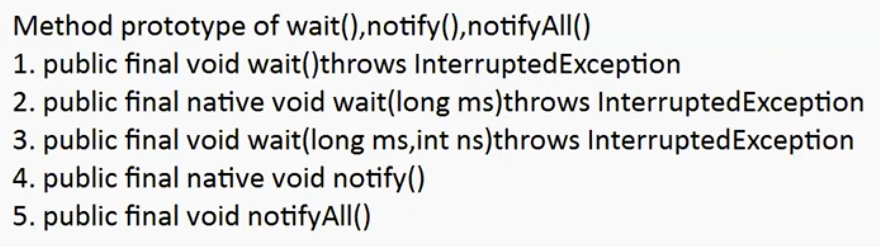
Except wait() , notify() , notifyAll() lock cant be released by other methods.

Note:

yield() , sleep() , join() cant release the lock

wait() , notify() , notifyAll() will release the lock otherwise inter-thread communication will not happen.

Once a thread calls wait() , notify() , notifyAll() methods on any object then it releases the lock of the particular object but not all locks it has

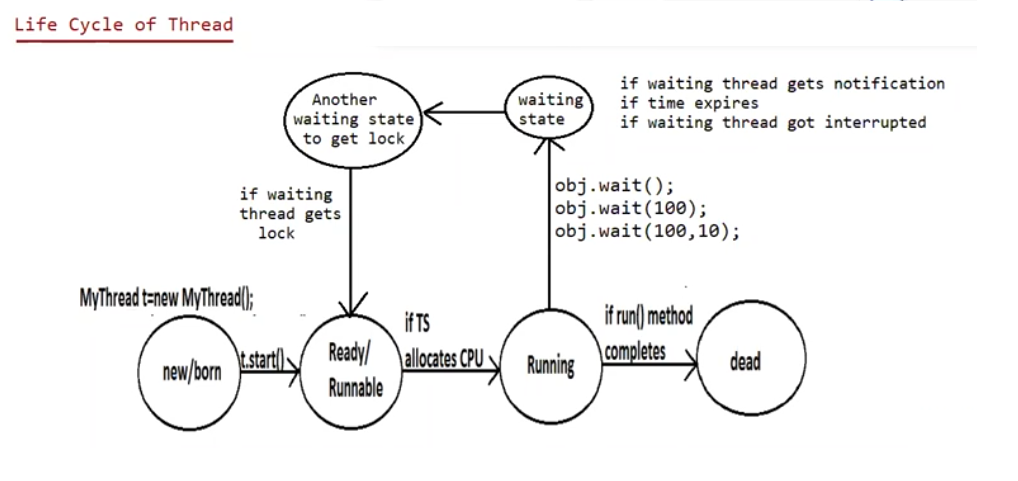


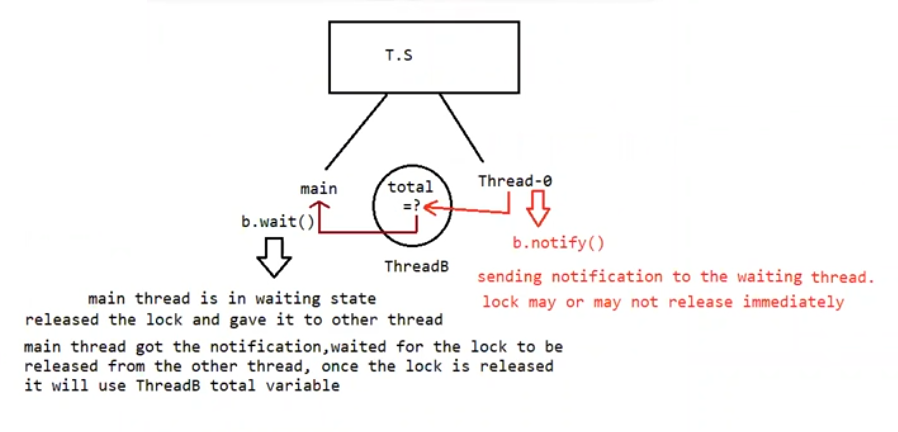
Methods like wait() , notify() , notifyAll() are present inside Object class , why not in Thread class?

Thread will call wait() , notify() , notifyAll() on Objects like PostBox , Stack, Customer, Student.

obj.wait() , obj.notify() , obj.notifyAll()

Even though the methods yield() , sleep() , join() are used in mutli-threading environment environment . that thread will make a method call to yield() , sleep() , join() not on a thread . it will call on any object . if a method has to be present for every object in java , then that methods should be present in the parent of the that class. Parent of every java class is object so it present in Object class.





Note: if the waiting time expires the thread goes to another waiting state to get the lock, if there is no another waiting state the thread would move to ready state immediately and it would result in the infinite wait.

Eg: Inter\_Thread\_Communication

// go through the code

Eg: Thread\_Without\_Communication

// go through the code

Eg: Thread\_Without\_Communication\_Using\_Join\_Method

//go through the code

Eg: Inter\_Thread\_Communication\_With\_Sleep\_Method

// go through the code

Eg: Inter\_Thread\_Communication\_Wait\_With\_MilliSeconds

// go through the code

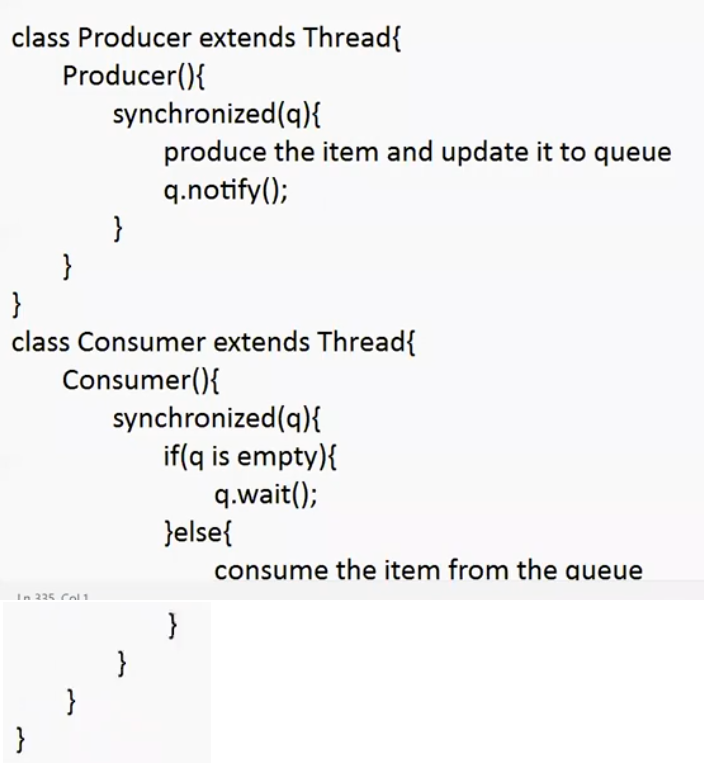
Eg: Inter\_Thread\_Communication\_With\_Sleep\_Method\_In\_Child\_Thread

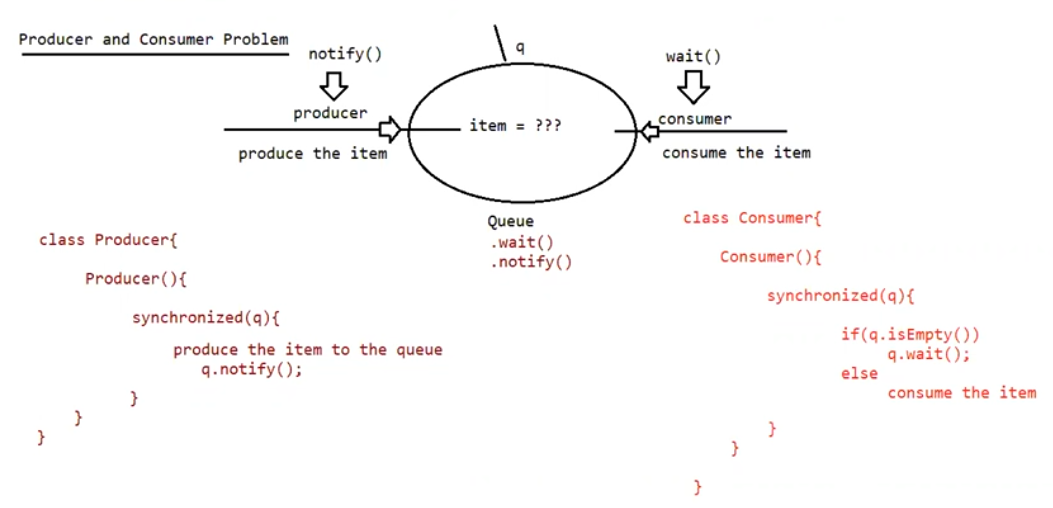
// go through the code

Producer Consumer problem:

Producer : produce the item and update in the queue

Consumer : consume the item from the queue





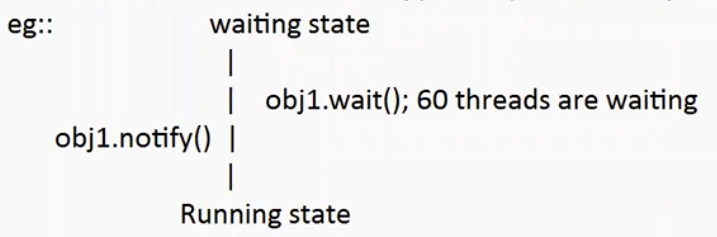
// go through the problem on the internet

Difference between notify() and notifyAll()

notify() : To give notification only for one waiting thread

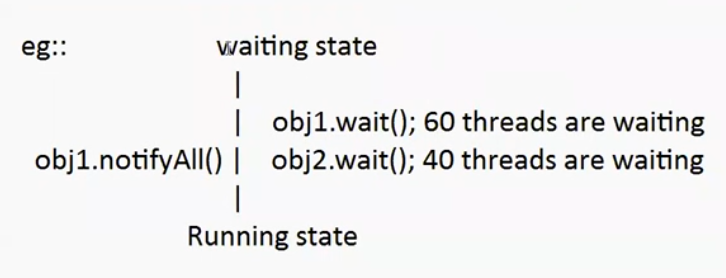
notifyAll() : To give notification for many waiting thread

we can use notify() to give notification only for one thread. If multiple threads are waiting then only one thread will get a chance and remaining thread has to wait for further notification. But which thread will notify ( inform) we cant expect ,it depends on jvm



Among 60 threads we can’t say , which thread will get a chance it is decided by jvm thread scheduler

We use notifyAll() method to give the notification for all waiting threads of a particular object all waiting threads will be notified and will be executed one by one , because they required lock



Note: on which object we are calling wait() , notify() and notifyAll() methods that corresponding object lock we have to get but not other object locks

