Thread synchronization, object level locking and class level locking

MARCH 8, 2013 LOKESH 37 COMMENTS

Synchronization refers to multi-threading. A synchronized block of code can only be executed by one thread at a time.

Java supports multiple threads to be executed. This may cause two or more threads to access the same fields or objects. Synchronization is a process which keeps all concurrent threads in execution to be in synch. Synchronization avoids memory consistence errors caused due to inconsistent view of shared memory. When a method is declared as synchronized; the thread holds the monitor for that method's object If another thread is executing the synchronized

method, your thread is blocked until that thread releases the monitor.

Synchronization in java is achieved using ***synchronized*** keyword. ***You can use synchronized keyword in your class on defined methods or blocks. Keyword can not be used with variables or attributes in class definition.***

Object level locking

Object level locking is mechanism when you want to synchronize a non-static method or non-static code block such that only one thread will be able to execute the code block on given instance of the class. This should always be done to make instance level data thread safe. This can be done as below :

```
1  public class DemoClass
2  {
3      public synchronized void demoMethod(){}
4  }
5
6  or
7
8  public class DemoClass
9  {
10     public void demoMethod(){
11         synchronized (this)
12         {
13             //other thread safe code
14         }
15     }
16 }
17
18 or
19
20 public class DemoClass
21 {
22     private final Object lock = new Object();
23     public void demoMethod(){
24         synchronized (lock)
25         {
26             //other thread safe code
27         }
28     }
29 }
```


YOU MAY ALSO LIKE

Difference between sleep() and wait()?

What is Thread Safety?

Writing a deadlock and resolving in java

Class level locking

Class level locking prevents multiple threads to enter in synchronized block in any of all available instances on runtime. This means if in runtime there are 100 instances of DemoClass, then only one thread will be able to execute demoMethod() in any one of instance at a time, and all other instances will be locked for other threads. This should always be done to make static data thread safe.

```
1  public class DemoClass
2  {
3      public synchronized static void demoMethod(){}
4  }
5
6  or
7
8  public class DemoClass
9  {
10     public void demoMethod(){
11         synchronized (DemoClass.class)
12         {
13             //other thread safe code
14         }
15     }
16 }
17
18 or
19
20 public class DemoClass
21 {
22     private final static Object lock = new Object();
23     public void demoMethod(){
24         synchronized (lock)
25         {
26             //other thread safe code
27         }
28     }
29 }
```

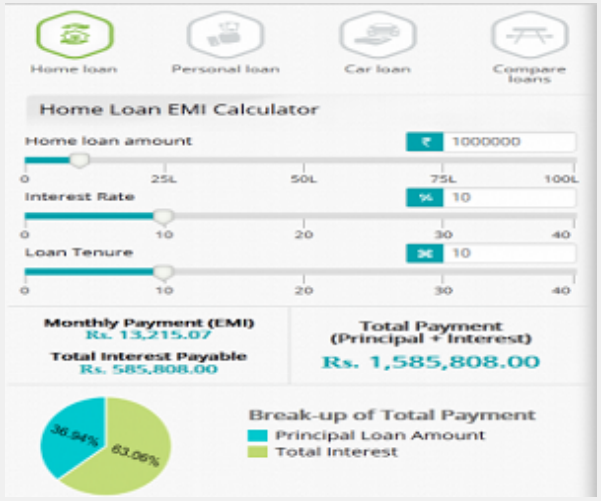
Some Important notes

1. Synchronization in java guarantees that no two threads can execute a synchronized method which requires same lock simultaneously or concurrently.
2. synchronized keyword can be used only with methods and code blocks. These methods or

Difference between “implements Runnable” and “extends Thread” in java

Core java interview questions series : Part 3

Inter-thread communication using piped streams in java



blocks can be static or non-static both.

3. When ever a thread enters into java synchronized method or block it acquires a lock and whenever it leaves java synchronized method or block it releases the lock. Lock is released even if thread leaves synchronized method after completion or due to any Error or Exception.
4. java synchronized keyword is re-entrant in nature it means if a java synchronized method calls another synchronized method which requires same lock then current thread which is holding lock can enter into that method without acquiring lock.
5. Java Synchronization will throw NullPointerException if object used in java synchronized block is null. For example, in above code sample if lock is initialized as null, the synchronized (lock) will throw NullPointerException.
6. Synchronized methods in Java put a performance cost on your application. So use synchronization when it is absolutely required. Also, consider using synchronized code blocks for synchronizing only critical section of your code.
7. It's possible that both static synchronized and non static synchronized method can run simultaneously or concurrently because they lock on different object.
8. According to the Java language specification you can not use java synchronized keyword with constructor it's illegal and result in compilation error.
9. Do not synchronize on non final field on synchronized block in Java. because reference of non final field may change any time and then different thread might synchronizing on different objects i.e. no synchronization at all. Best is to use String class, which is already immutable and declared final.

Happy Learning !!

Related Posts:

1. Difference between sleep() and wait()?
2. What is Thread Safety?
3. Writing a deadlock and resolving in java
4. Difference between “implements Runnable” and “extends Thread” in java
5. Core java interview questions series : Part 3
6. Inter-thread communication using piped streams in java

LOCKING MULTI THREADING SYNCHRONIZED

PREVIOUS POST

Object initialization best practices : Internal caching in wrapper classes

NEXT POST

Difference between sleep() and wait()?

37 THOUGHTS ON “THREAD SYNCHRONIZATION, OBJECT LEVEL LOCKING AND CLASS LEVEL LOCKING”

Amreesh

OCTOBER 23, 2014 AT 3:38 PM

```
==================================How to lock Object==================================
class A
{
synchronized void test1()
{
for(int i=0;i<100;i++)
{
System.out.println(i);
}
}
synchronized void test2()
{
for(int i=200;i<300;i++)
{
System.out.println(i);
}
}
}
class B extends Thread
{
A a1;
B(A a1)
{
this.a1=a1;
}
public void run()
{
a1.test1();
}
}
class C extends Thread
{
A a1;
C(A a1)
```

```
{
this.a1=a1;
}
public void run()
{
a1.test2();
}
}
public class Manager1
{
public static void main(String[] args)
{
A a1=new A();
//A a2=new A();
B b1=new B(a1);
C c1=new C(a1); //Here is chance to currpt the data because same references
b1.start(); //If not same ref then no chance to currpt data
System.out.println("============");
c1.start();
}
}
```

REPLY

Amreesh

OCTOBER 23, 2014 AT 7:48 AM

I have confusion in Thread,What is different b/w Class level lock and Object level lock???
Thanks..

REPLY

Prahald sahu

AUGUST 5, 2014 AT 4:30 PM

Mind blowing Lokesh..

REPLY

krishna Kankal

JULY 30, 2014 AT 7:18 AM

Nice article Lokesh

REPLY

Prateek

JULY 17, 2014 AT 7:22 AM

Hi Lokesh,
I have always seen class level locking in my coding but never have i come across a requirement where i require an object level lock.Can you please tell me any real life scenario where we can use Object Level locking?

REPLY

Rohit

JUNE 3, 2014 AT 4:11 PM

Hi Sir,
Really nice article, but i have doubt over one point can u pls elaborate it with example…
7. It's possible that both static synchronized and non static synchronized method can run simultaneously or concurrently because they lock on different object.

REPLY

Ankit

JULY 11, 2014 AT 3:32 PM

Hi,
I would like to elaborate on you 7th point. As far as static synchronized methods are concerned, we are not taking any object into consideration as all static methods are class methods and can only be invoked by class variables. So, we can only have class lock on static methods.

Now, for non static synchronized methods, we are talking about object only as these methods invoke on object. So, here we can have class locks as well as objects locks.

So, two methods one is static and other is non static can take lock on object and class. In this way both the methods can run simultaneously.
Hope this thing is clear to you.

Thanks

REPLY

Nishant Sharma

MAY 21, 2014 AT 4:09 AM

Hi Lokesh,

Thanks for the article on synchronization. I liked it ..But I have query in my mind that . you said Synchronization method will allow lock on OBJECT level and by adding static synchronization it add the functionality to allow lock on CLASS level..

But in the below code i have created 2 OBJECTS of same class but still i am not getting the desired effect of synchronization block .. but when i applied static to it hen it was working.. Please explain..

Thanks in advance...

```
1  public class Thread3 extends Thread
2  {
3      //synchronized public void testsync()
4      synchronized public static void testsync()
5      {
6          for (int i = 0; i &lt;= 5; i++) {
7              System.out.println(i);
8              try {
9                  Thread.sleep(500);
10             } catch (Exception e) {
11                 System.out.println(e);
12             }
13         }
```

```
14      }
15
16  public void run() {
17      testsync();
18  }
19
20  public static void main(String[] args) {
21      Thread3 obj = new Thread3();
22      Thread3 obj2 = new Thread3();
23      obj.start();
24      obj2.start();
25  }
26  }
```

REPLY

★ Lokesh

MAY 21, 2014 AT 4:58 AM

I executed the above program. Below are the outputs:
When method is static: 0 1 2 3 4 5 0 1 2 3 4 5
When method is non-static: 0 0 1 1 2 2 3 3 4 4 5 5

When method is static: class level lock i.e. both instances of Thread3 are locked by first obj and then obj2. They execute the method in sequence.
When method is non-static: Both instances execute the method independently in their instances.

In fact, in non-static mode they do not share any shared resource/method. So if you drop the synchronized keyword itself, it doesn't make any difference. synchronized keyword should be used to protect some shared resource which will be accesses simultaneously by multiple threads. Here testSync() method does not do anything like that.

REPLY

muppanashiva

JULY 14, 2014 AT 6:45 AM

hii lokesh sir.. i have a general doubt as. can we lock object except

synchronization concept..
i mean am creating object like ,
Test t=new Test();
now i want to lock t object with out using sync concept..
is it possible

REPLY

★ Lokesh

JULY 14, 2014 AT 9:15 AM

You can use Lock implementations of http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/Lock.html

REPLY

muppanashiva

JULY 14, 2014 AT 9:21 AM

thank u sir.. i have already referred these docs.. but i didnt get anything..
that is y am asking u

REPLY

★ Lokesh

JULY 14, 2014 AT 12:34 PM

So basically you want a simple tutorial on how to use these lock objects, right? I will post one for you soon.

REPLY

muppanashiva

JULY 14, 2014 AT 1:32 PM

thanks in advance

REPLY

Mit

JULY 17, 2014 AT 5:07 PM

What changes should i make so that the output when the method is non static is equal to the output using static method. I don't want to use static keyword. I just want to know how can we achieve this for non static methods.

And one more thing. Can you please explain what object should we keep in synchronized public void testsync('object') between the brackets. How do we decide what to keep here. I saw on most websites that thy keep a variable of the class here, even if that variable is not used anywhere in the method. Thanks in advance

REPLY

Kamal

APRIL 23, 2014 AT 6:03 PM

Hi Lokesh,

Thanks for you articles, they are great!!

I would like to mention one thing though that your explanation on Class Level Lock is bit misleading:

"Class level locking prevents multiple threads to enter in synchronized block in any of all available instances on runtime. This means if in runtime there are 100 instances of DemoClass, then only one thread will be able to execute demoMethod() in any one of instance at a time, and all other instances will be locked for other threads. This should always be done to make static data thread safe."

Static code doesn't belong to any instance, so it has no relation with instances at

all. Yes out of 100 threads contending only one thread having managed to acquire the class level lock will be able to execute the static block of code.
This should be enough, as code block belongs to instances (non-static methods) can still be executed in different thread at the same time even if they are guarded with Object level lock.

Thanks,
Kamal

REPLY

Dave

APRIL 13, 2014 AT 2:15 PM

Excellent tutorial.. Thank you..

REPLY

suma

MARCH 2, 2014 AT 3:22 PM

In one of the interview, I have asked a question. Synchronized means, thread will get lock on object/instance. what happened in case of static synchorized?I said that thread will get lockon class.then interviewer asked where/how does class exists? I stuck over here.Can you plz explain, how a thread get lock on class?where does the class(template) exists

REPLY

★ Lokesh

MARCH 2, 2014 AT 6:46 PM

plz search more on semaphores. or wait one week. I am out of town on vacations.

REPLY

Snehal Masne

FEBRUARY 26, 2014 AT 4:26 AM

Wonderful notes Lokesh, clarified most of my doubts. Thanks and Keep it up

REPLY

Josh Molina

FEBRUARY 13, 2014 AT 7:11 PM

Hi Lokesh, very nicely done. Good job!

REPLY

Shreedhar

JANUARY 10, 2014 AT 8:50 PM

Hi, Lokesh , I want to ask you that does it mean that if we synchronize a block of code or method , we are are assured that it will always and successfully execute ?

REPLY

★ Lokesh Gupta

JANUARY 10, 2014 AT 9:12 PM

Method will execute anyway. Synchronization means method will be executed in a multi-threaded environment like it is running in single thread. It means multiple threads will not have effect of one another in their execution path.

REPLY

dowla

DECEMBER 1, 2013 AT 11:42 AM

Hi lokesh i need to know when the synchronized block will get executed.

REPLY

★ Lokesh Gupta

DECEMBER 1, 2013 AT 1:46 PM

There is nothing special in when synchronization block gets executed. It's all about only one thread at a time.

REPLY

Bhumik

NOVEMBER 19, 2013 AT 11:24 PM

Hi Lokesh, could you please explain the reason behind: "According to the Java language specification you can not use java synchronized keyword with constructor it's illegal and result in compilation error"

REPLY

★ Lokesh Gupta

NOVEMBER 20, 2013 AT 12:26 AM

Usually it is considered bad style to "give out" your not-yet-constructed object, so a synchronized constructor is not necessary.
JLS actually says: "There is no practical need for a constructor to be synchronized, because it would lock the object under construction, which is ***normally*** not made available to other threads until all constructors for the object have completed their work."

But sometimes construction process is long enough to violate this rule so you can use synchronized block inside constructor. This is allowed.

```
public class Test {
public Test() {
final Test me = this;
synchronized(this) {
new Thread() {
@Override
```

```
public void run() {
// ... Reference 'me,' the object being constructed
synchronized(me) {
// do something dangerous with 'me'.
}
}
}.start();
// do something dangerous with this
}
}
}
```

REPLY

Ramana

NOVEMBER 17, 2013 AT 5:36 PM

HI Lokesh, You asked
me same questions in interview

REPLY

★ Lokesh Gupta

NOVEMBER 17, 2013 AT 7:32 PM

Sorry Ramana, I really don't remember but I hope you answered well… 🙂

REPLY

Radhe

MARCH 4, 2014 AT 6:01 PM

Ramana, Whenever a class gets loaded by JVM, JVM creates an object which is called class.

Say for example MyTest.java, where MyTest is class. So when MyTest gets loaded 'MyTest.class' object gets created.
if MyTest class having any static syncronized method then lock would be on MyTest.class object.

For more detail please visit
http://radhedubey.wordpress.com/2013/12/10/thread-synchronization-object-level-locking-and-class-level-locking/

REPLY

H Singh

OCTOBER 11, 2013 AT 9:16 PM

Hi Lokesh,

At RUNTIME, is it possible that two threads can access same synchronized block, object ?

It seems that it can not happen, then how "Static Synchronized" and "Synchronized" declaration for any block or method behaves differently at run time.

Say I have a data bean class that gets current USD to INR rate. For this it has a method that calls RBI webService to get current rate. Now please provide your view on difference in following conditions:

1) If I declare – static synchronized getCurrentRate()
2) If I declare – synchronized getCurrentRate()

Also putting "Static Synchronized" in declaration of any method or code block, will impact performance more as compare to "Synchronized"

I just want to make clear that exactly what is the difference between Class level and Object level Synchronization, in terms of performance and data inconsistency.

Thanks.

REPLY

★ **Lokesh Gupta**

OCTOBER 12, 2013 AT 12:03 AM

Please again read class level locking description in post:

“if in runtime there are 100 instances of DemoClass, then only one thread will be able to execute demoMethod() in any one of instance at a time, and all other instances will be locked for other threads. This should always be done to make static data thread safe.”

Taking your example: (Let’s say getCurrentRate() method is defined in CurrencyConverter.java and there are 10 instances of this class)

1) If I declare – synchronized getCurrentRate() : In this case, 10 seperate threads will be able to access this method in all 10 different instances. In simple words, 1 thread per instance. By the time, any thread is executing this method in any of instances, 11th thread must wait.

2) If I declare – static synchronized getCurrentRate() : In this case, if one thread got access to instance one, then all other nine instances will also be locked and rest nine threads will be in waiting state. Once thread one finishes it’s job and release the lock, one of nine threads again will get access and rest eight will be in wait state.

I will not try to compare the performance of both blocks because they are entirely different functionalities.

As far as data consistency is concerned, non-static synchronized blocks are used to thread-safe instance level data, while static synchronized blocks are used to safe static data.

REPLY

H Singh

OCTOBER 12, 2013 AT 4:20 AM

Hi Lokesh,

Thanks for responding.

Your explanation For synchronized getCurrentRate() : Seems that in this case , local variables are safe but not global and Static variables. In this case Local variables will not be accessed and updated by more than one thread. But in this case more than one thread can change values for Static and global variables.

Can you provide a real World example for making it more clear.

REPLY

★ Lokesh Gupta

OCTOBER 12, 2013 AT 7:25 AM

What you re-iterated is completely true. It is essentially the way synchronization block works.

I really never used static synchronized block as far i can remember (apart from tutorials). I will really appreciate if you update this conversation with your past experience if you have any.

REPLY

Satheesh

DECEMBER 6, 2013 AT 6:15 AM

Hi Lokesh,

I have been reading some of your post and it is really informative and easy to follow.

I just wanted to clarify one thing in this post, you

mentioned that “static synchronized blocks are used to safe guard static data”.

But my understanding is that the static data are inherently thread-safe as they are initialized during class loading which is guaranteed to be thread-safe by the JVM.

Also it is my believe that no need to make the local variable thread-safe as each thread will have its own call stack so no issue of race condition.

Regards,
Satheesh

REPLY

★ Lokesh Gupta

DECEMBER 7, 2013 AT 10:40 PM

Thanks for sharing your views. I appreciate this. Synchronization is there to prevent the data from being corrupt when it is simultaneously accessed by two (or more) threads. You are right that at initialization time, data will be safe most of the time (not all the time if object creation process is long enough, in this case JVM may return the reference of not fully-prepared object). Still static data needs synchronization in other phases of application. Reason is that most JVM operations are multi-step (non-atomic) and there is high chances of data manipulation by two threads such that resulting data override each other’s changes.

REPLY

radhedubeyradhe

DECEMBER 10, 2013 AT 10:13 AM

Just want to clear few things, synchronization means synchronizing state changes of Object and that we can achieve it via synchronizing block in any java class. Now suppose there are 100 threads which trying to work with same object say obj1, which has some synchronized block, and all 100 threads trying to enter in that block.That point of time only one thread gets lock & will execute synchronized block.

Now suppose all 100 threads trying to enter in synchronized block of java class with different- different Object instances say obj1, obj2……. obj100, Then all 100 threads can get entered in block because lock is on current obj.

Suppose you have static synchronized block, locking get applied for javaclass.class object which is created during class loading.

REPLY

Note:- In comment box, please put your code inside **[java] ... [/java]** OR **[xml] ... [/xml]** tags otherwise it may not appear as intended.

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

Are you Human and NOT robot? Solve this to prove !! *

6 × six =

Comment

You may use these HTML tags and attributes:

```
<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <strike> <strong>
```

POST COMMENT

☐ Notify me of follow-up comments by email.

▣ Notify me of new posts by email.

SUBSCRIBE TO BLOG VIA EMAIL

Join 3,665 other subscribers

Email Address

Email Address

SUBSCRIBE

REFERENCES

Oracle Java Docs

Spring 3 Reference

Spring 4 References

Jboss RESTEasy JAX-RS

Hibernate Developer Guide

Junit Wiki

Maven FAQs

Dzone Dev Links

JSP Homepage

ANT Homepage

META LINKS

POPULAR POSTS