ISSN No. 0976-5697

Volume 5, No. 5, May-June 2014

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Advantages of Object Synchronization over Resource Synchronization in Java

Mr. Chetan Singh Khinchi
Department of Computer Applications
Acropolis Institute of Technology and Research
Indore, India

Mr. Vaibhav Sharma
Department of Computer Applications
Acropolis Institute of Technology and Research
Indore, India.

Mr. Ronak Jain
Department of Information Technology
Acropolis Institute of Technology and Research,
Indore, India.

Abstract: In java synchronization is basically of two types – First is , Method(Resource) synchronization and Second one is Block(Object) synchronization . Both of them adds to the security of code while multiple threads are in action and have very minute differences over one another. People often get confused with both of the ways and do not understand the usage and applicability of this ways by themselves[1].

Here I suggest that rather than using Method(Resource) synchronization for security of code in multithreading in java one must give preference to the Block(Object) synchronization .This paper is an Analytical as well as a Suggestive paper which actually describes both the ways with all minute differences covered with a proper analysis of usage and applicability, and then suggests that why one must use Block(Object) synchronization over Method(Resource) synchronization in java while multiple threads are in action.

Keywords: Synchronization , block , resource , object , method , Monitor , thread, sleep , join

I. INTRODUCTION

Synchronization is the way of preventing threads to share the same resources at the same time thus preventing unwanted results due to concurrency issues in a java program. With synchronization we synchronize the action of multiple threads so that only one thread uses a resource in one time which belongs to a specific Object. This complete logic depends upon one thing called as : Object Monitor - which is a lock contained by default by every object in java. It is said and happens to be that only one thread in one time can hold objects monitor and can use the resources pertaining to that object . Other threads can only enter objects monitor if and only if the first thread has come out of the objects monitor and have unlocked it after coming out. Beside this point, there are several issues which will be discussed that affects the resource sharing by multiple threads in java.

There are “two-ways” in java through which multiple threads may run at one time but can be managed so that they do not disturb each others execution and uses the resources in a synchronized way. The first way is to synchronize the resources of a specific object which is to be shared by multiple threads. The second way is to synchronize the point from where multiple threads may enter and reach the resource to be utilized[1][6].

II. SYNCHRONIZING METHODS(RESOURCES) IN JAVA

A variable , constructor or methods are called as the resources of an object. Out of these resources , particularly methods can be shared by threads , but it may happen that while a thread is utilizing a resource , another thread comes in and starts utilizing the same resource based on some functionality and results in a result which might be far more different then expected. So in such a case it is very much required that the resource must be synchronized , so that it can only be used by one thread at a time. It still can be shared but avoids the mix up of threads and thus makes a thread wait outside the resource until a first thread primarily saves the state and come out of the resource. In this fashion we can make multiple threads work together and share the same resource on “one-by-one” basis. First I will quote an example**[Program A]** where multithreading is used but no synchronization is used and thus threads interrupt each other and hence results in unexpected outcome[2].

```
class Result_Producer

{

public void   show_int(int i)

{

System.out.print(“[” + i);

try

{

Thread.sleep(1000);

}
Catch(InterruptedException e)
{
e.printStackTrace();
}
System.out.print(“]”);
} }


class Controller_of_Result_Producer implements Runnable
{
Thread t;
int j;
Result_Producer rp;
public Controller_of_Result_Producer(Result_Producer rpro , int a)
{
rp=rpro;
j=a;
t=new thread(this);
t.start()
}
public void run()
{
rp.show_int(j);
} }
class main_Class
{
public static void main(String args[])
{
Result_Producer rp1=new Result_Producer();
Controller_of_Result_Producer crp1=new Controller_of_Result_Producer(rp1,10);
Controller_of_Result_Producer crp2=new Controller_of_Result_Producer(rp1,20);
Controller_of_Result_Producer crp3=new Controller_of_Result_Producer(rp1,30);
try
{
crp1.t.join();
crp2.t.join();
crp3.t.join();
}
Catch(InterruptedException e)
{
e.printStackTrace();
} } }
```

Output is :

```
[10[20[30]]]
```

Expected Output :

```
[10][20][30]
```

In the above example we have three classes – First is , Result_Producer which is given a task to take an integer as input and produce result as [integer], and we want to perform this task for three times.The class Result_Producer is controlled by Controller_of_Result_Producer , which actually provides the object of Result_Producer inorder to give a call to the show_int method of Result_Producer. The object through which call will be given to Result_Producer is named as “rp”.The third class is main_Class which actually controls whole program.The integers passed are 10, 20 , 30.As you can see the output expected was [10][20][30] , but what has been received is [10[20[30]]]. This is because of the following reasons:

1.Thread corresponding to object cp1 actually finds the object “rp” vacant so it enters the object monitor and starts performing the given task that is to print [10] , but when it prints [10 we have given a call to sleep(1000) which forces this thread to save the state and leave the monitor for 1 second(1000 milliseconds).

2. Now since it’s a case of multithreading , thread corresponding to object cp2 find the object “rp” vacant and enters the objects monitor , and starts performing the given task while thread of cp1 is sleeping.We wanted it to print [20], but when it prints [20 it also encounters a call to sleep method , thus it saves the state and leaves the monitor of the object.

3. Now since it’s a case of multithreading , thread corresponding to object cp3 find the object “rp” vacant and enters the objects monitor , and starts performing the given task while thread of cp1 and cp2 are sleeping. We wanted it to print [30], but when it prints [30 it also encounters a call to sleep method , thus it saves the state and leaves the monitor of the object.

4. Since we have applied join on all of the above three methods , so it means thread of cp1 will complete first , cp2 second and cp3 third.

5. Now after executing sleep(1000) all the threads come back one by one and executes the remaining statements , i.e. "System.out.println("]")". Which actually makes the output as "[10[20[30]]]".

So the point is , that this all happened because the resource (actually the method – show_int()) is not synchronized .So , this time "in-order to synchronize the Resource(Method)"we will rewrite the Result_Producer class as :

```
class Result_Producer

{

synchronized public void   show_int(int i)

{

System.out.print("[" + i);

try

{

Thread.sleep(1000);

}

Catch(InterruptedException e)

{

e.printStackTrace();

}

System.out.print("]");

} }
```

This time everything will remain same , only a keyword synchronized has been prefixed before the method name . Which makes the outcome to be [10][20][30] (The desired outcome). **Why ?**

III. DISCUSSION ON PROS AND CONS OF THE METHOD

Looking at the above program we can very finely discuss the pros of this method: Pros can only be discussed by explaining it through an example (Pros):

1.This time when thread corresponding to object cp1 enters the monitor of object "rp", it starts exceuting the show_int() and prints [10 and finds a call to sleep() method , saves its state and goes out of the monitor of "rp"[3][4][5].

2.Now the thread corresponding to object cp2 enters the monitor of object "rp", it will try to enter the method show_int(), but this time it will not be allowed to enter the method , due to availability of "synchronized" keyword , which actually stops other threads to enter the same method until and unless the first thread already in the method performs its task completely. Do not forget that we have also applied join().So thread of cp2 has to wait outside of the object "rp"[3][4][5].

3.Similarly , thread corresponding to object cp3 also enters the monitor of object "rp", it will try to enter the method show_int(), but this time it will not be allowed to enter the method , due to availability of "synchronized" keyword , which actually stops other threads to enter the same method until and unless the first thread already in the method performs its task completely. Do not forget that we have also applied join().So thread of cp3 has to wait outside of the object "rp".

4.Finally , First thread returns to the object's monitor and completes the task by executing statement "System.out.println("]")" . Thus in this way First thread gives output as [10].

5.The same logic is repeated for all the threads available, and a result(Expected Result) [10][20][30] is produced.

Cons :

In the above example , when First thread executed the call to sleep() , it saved its state and went on to sleep for some time , due to availability of synchronized keyword meanwhile no thread was allowed to enter the method(Resource) of the object "rs". This event actually raises two questions:

Q1: If second and third thread is not allowed to execute a method which is marked synchronized if already one thread is in, then why they are allowed to enter the objects monitor , and this process actually wastes cpu cycle since , threads two and three have to march in and out of the objects monitor[7].

Q2: In this scenario , suppose if there is a non synchronized method which actually receives input from the synchronized method and if thread two starts executing the non synchronized method which yet hasn't received input from thread one , may produce unwanted results. So it actually also posess a problem of resource de-optimization[7][8].

IV. SYNCHRONIZED BLOCK(OBJECT) IN JAVA

Here we can define our **[Program A]** again with some minute changes , inorder to perform Block(Object) synchronization in java.

```
 class Result_Producer

{

public void   show_int(int i)

{

System.out.print("[" + i);

try

{
```

```
Thread.sleep(1000);

}

Catch(InterruptedException e)

{

e.printStackTrace();

}

System.out.print("]");

} }


class Controller_of_Result_Producer implements Runnable

{

Thread t;

int j;

Result_Producer rp;

public Controller_of_Result_Producer(Result_Producer
rpro , int a)

{

rp=rpro;

j=a;

t=new thread(this);

t.start()

}

public void run()

{

synchronized(rp)

    {

        rp.show_int(j);

    } } }

class main_Class

{

public static void main(String args[])

{

Result_Producer rp1=new Result_Producer();

Controller_of_Result_Producer crp1=new
Controller_of_Result_Producer(rp1,10);

Controller_of_Result_Producer crp2=new
Controller_of_Result_Producer(rp1,20);

Controller_of_Result_Producer crp3=new
Controller_of_Result_Producer(rp1,30);

try

{

crp1.t.join();

crp2.t.join();

crp3.t.join();

}

Catch(InterruptedException e)

{

e.printStackTrace(); } } }
```

Output is :

```
[10][20][30]
```

1. The first change in **[Program A]** is that ,this time no method has been marked as synchronized in the program.

2. In public void run() , we have written "synchronized(rp) { rp.show_int(j)} " which actually synchronizes the point of entry of multiple threads i.e. the "Object rp" .

3. In the above example When thread corresponding to object cp1 finds the monitor of object rp vacant , it enters the monitor and starts executing the method show_int(int) , after printing [10 it finds a call to sleep(1000) method , it first saves its state and along with it keeps the lock to the monitor and finally goes to the sleeping state(Blocked) . So thread corresponding to object cp2 is not at all allowed to enter the objects monitor since thread of cp1 hasn't released it yet . Until and unless thread of cp1 comes back and executes the remaining code i.e. "System.out.println("]")" and leaves the monitor producing output as : [10] no other will be allowed to enter the same object's monitor and use public void show_int(int).

V. DISCUSSION ON PROS AND CONS OF THE METHOD(BLOCK (OBJECT) SYNCHRONIZATION)

Pros:

1. If the programmer do not have access to the methods of a program , he can still synchronize the actions of threads by just making the entry point of threads "synchronized", i.e he can atleast synchronize the object through which threads enter and utilize the resources[1][9] .

2. Complexity reduces and wastage of cpu cycles is stopped since other threads are not allowed to enter object's monitor until and unless first thread wakes up from sleep() , enters the objects monitor , performs its task completely and then

comes out of the object's monitor and releases the lock[1][9].

3. Higher level security is provided, which increases code reliability.

Cons:

1. If an object has references to more then one methods , then those methods cannot be utilized by other threads even if the first thread is not executing them and sleeping , since they cannot even enter the objects monitor until and unless first thread releases the lock. This way the problem of resource de-optimization remains the same.

VI. COMPARISON AND DETAILED ANALYSIS OF BOTH THE METHODS IN JAVA[10]

Table 1 : Comparison of Both the Methods

S.No.	Method(Resource) Synchronization	Block(Object) Synchronization
1	Synchronizes the action of multiple threads	Synchronizes the action of multiple threads
2	Uses synchronized keyword	Uses synchronized keyword
3	Other threads cannot utilize the same resource which is utilized by one thread	Here also , Other threads cannot utilize the same resource which is utilized by one thread.
4	Allows sleep , wait etc. methods to be applied.	It also allows sleep , wait etc. methods to be applied

Table 2 : Differential cum Advantage Analysis of Bothe the Methods

S.No.	Method(Resource) Synchronization	Block(Object) Synchronization
1	Synchronized is written before a method	Synchronized is written before an object
2	An object which contains a synchronized method allows other threads to enter its monitor when the first thread is sleeping , but other threads cannot enter the synchronized method until and unless the first thread completes and releases the lock on the same object	An object which is synchronized does not allow any other threads to even enter the objects monitor until and unless the first thread wakes up from the sleep , comes back and executes and thus releases the lock on the object . After this only , other threads will be allowed to enter the objects monitor
3	Point 2 actually causes complexity to raise , since other threads can enter objects monitor but can't utilize a synchronized method	Point 2 does not causes any complexity to occur since threads are stopped outside the object
4	Because of point 2 threads can enter an objects monitor and thus can utilize synchronized methods on the same object while thread 1 is sleeping	No non synchronized methods can be invoked when the first thread is sleeping.
5	Provides security when multiple threads are in action , but does not stop other threads to enter objects monitor and utilize non synchronized methods while thread 1 is asleep , this causes a potential threat to the expected outcome.	It also provides security when multiple threads are in action , but provides a higher level of security by stopping threads outside the object , thus other threads cannot enter objects monitor when first thread is asleep , thus it removes the threat created by other threads which may execute the non synchronized methods as in the case of the Method(Resource) Synchronization
6	Cannot be applied when the developer doesn't have accessibility to the methods of the program	Can be applied even if the developer doesn't have accessibility to the methods of the program. He just have to synchronize the Block(Object) from where multiple threads will enter the objects monitor and uses its methods

VII. SUGGESTION ? WHY TO PREFER BLOCK(OBJECT) SYNCHRONIZATION TO AVOID ANY TYPE OF COMPLEXITY.

As we have come across both the methods we can clearly understand that synchronizing a block(object) is far more better for providing security and eradicate concurrency related issues as compared to synchronizing a method(resource). **I suggest to prefer synchronization of a block(object) to be used over synchronization of method(resource) over the following points:**

1. Easy to use.
2. No need of method accessibility.
3. Reduces complexity by stopping threads out of an object, thus saves cpu cycles.
4. Reduces potential threat of non synchronized methods on same object since accept first thread other threads are not given access to objects monitor until and unless the first thread releases the lock.
5. No doubt resource de optimization occurs but to gain complete security some thing has to be sacrificed. But you can be sure that code will perform as you have designed it to be.
6. Just have to write a synchronized keyword and make a block.

7. Last but not the least performs all those things, which are performed by a synchronized method.

VIII. SUMMARY

So here by I conclude stating that if an easy and non complex way of carrying out something specially while coding is present then why will one choose the complex way of doing it. I suggest usage of Block(Object) synchronization is far better way as compared to Method(Resource) synchronization for achieving multiple thread synchronization which is the key or one can say one of the strong pillars to the success of java worldwide. People can choose it another way also , but they must compare the points I have stated in this paper. People may find different solutions to this problem in different situations but what I have suggested will definitely help people to come to a decision or help understanding resolving complexity. I do not say that out of the above two stated methods, any one method is lesser as compared to the another, java has given the world a language which will continue to exist for long , since it is the base for many of the languages in world, and yes “may it will live long” .But what I suggest that choose one of the any right methods depending upon a condition.

IX. REFERENCES

[1] Herbert Schildt ,The Complete Reference-Java 2 , V Edition , Volume 1.4 , 2002, (Tata McGraw – Hill Publishing Company Limited) , New Delhi

[2] Bruce Eckel , Thinking in JAVA, III Edition Revision 2.0 ,Volume 2 ,2002, (Prentice Hall Publication) , New Jersey , USA

[3] Joshua Bloch , Neal Gafter , JAVA PUZZLERS: TRAPS , PITFALLS AND CORNER CASES , II Edition , Volume 2 , 2008 , (Addison-Wesley Professional)

[4] Kathy Sierra , Bert Bates, Head First Java , II Edition,2005 , (O'Reilly Media)

[5] Oracle , “Synchronized methods”

http://docs.oracle.com/javase/tutorial/essential/concurrency/syncmeth.html

[6] TutorialsPoint , Java – Thread Synchronization

http://www.tutorialspoint.com/java/java_thread_synchronization.htm

[7] Joshua Bloch , Effective Java , II Edition , Volume 1 , 2008 , (Prentice Hall)

[8] Goetz and Tim Peierls, JAVA CONCURRENCY IN PRACTICE , I Edition ,2006 , (Addison –Wesley)

[9] Dr. R. Nageswara Rao , Core Java An Integrated Approach , I Edition , Volume 1 ,2008 , (Dreamtech Press)

[10] Cay S. Horstmann and Gary Cornell , Core Java Advanced Features , IX Edition , Volume 2, 2008, (Prentice Hall)