Java Synchronization

At times when more than one thread try to access a shared resource, we need to ensure that resource will be used by only one thread at a time. The process by which this is achieved is called synchronization. The synchronization keyword in java creates a block of code referred to as critical section.

Synchronization in Java is an important concept since Java is a multi-threaded language where multiple threads run in parallel to complete program execution. In multi-threaded environment synchronization of Java object or synchronization of Java class becomes extremely important. Synchronization in Java is possible by using Java keywords **"synchronized"** and **"volatile"**.

Concurrent access of shared objects in Java introduces to kind of errors: thread interference and memory consistency errors and to avoid these errors you need to properly synchronize your Java object to allow mutual exclusive access of critical section to two threads.

- Synchronized is the keyword applicable for methods and blocks but not for classes and variables.
- If a method or block declared as the synchronized then at a time only one thread is allow to execute that method or block on the given object.
- The main advantage of synchronized keyword is we can resolved inconsistency problems.
- But the main disadvantage of synchronized keyword is it increases waiting time of the thread and effects performance of the system.
- Hence if there is no specific requirement then never recommended to use synchronized keyword.
- Internally synchronization concept is implemented by using lock concept.
- Every object in java has a unique lock.Whenever we are using synchronized keyword then only lock concept will come into thepicture.
- If a thread wants to execute any synchronized method on the given object first it has to get the lock of that object. Once a thread got the lock of that object then it's allow to execute any synchronized method on that object. If synchronized method execution completes then automatically thread releases the lock.

Example:

```
class Display
 {
  public synchronized void wish(Stringname)
    {
       for(inti=0;i<5;i++)
         { System.out.print("goodmorning:");
           try { Thread.sleep(1000); } catch(InterruptedExceptione) {}
          System.out.println(name);
       }
}}

 class MyThread extends Thread
  {   Display  d;
     String name;
     MyThread(Display d,String name)
       { this.d=d;
         this.name=name; }
      public void run()
       {  d.wish(name); }
}
class  SynchronizedDemo
  {
   Public static void main(String[] args)
  {
   Display d1=new Display();
   MyThread t1=new MyThread(d1,"dhoni");
   MyThread t2=new MyThread(d1,"yuvaraj");
   t1.start();
   t2.start();
  }
}
```

•If we are not declaring wish() method as synchronized then both threads will be executed simultaneously and we will get irregular output.

Note:

```
Display d1=new Display();
Display d2=new Display();
MyThread t1=new MyThread(d1,"dhoni");
 MyThread t2=new MyThread(d2,"yuvaraj");
 t1.start();
 t2.start();
```

•Even though we declared wish() method as synchronized but we will get irregular output in this case,because both threads are operating on different objects.

While a Thread executing any synchronized method the remaining threads are not allowed execute any synchronized method on that object simultaneously. But remaining threads are allowed to execute any non-synchronized method simultaneously. [lock concept is implemented based on object but not based on method].

```
class Hello
{
   public synchronized  void hi(String name)
    {
   for(int i=0;i<10;i++)
    {
     System.out.print("Hello All  :");
     try { Thread.sleep(2000); }
     catch (InterruptedException e) { }
    System.out.println(name);
    }
}
public  synchronized   void hi1(String name)
    {
   for(int i=0;i<10;i++)
    {
     System.out.print("Hello All  :");
     try { Thread.sleep(2000); }
     catch (InterruptedException e) { }
    System.out.println(name);
    }
}
}

class MyThread1 extends Thread
{
  Hello h;
  String name;
  MyThread1(Hello h1, String name)
   { this.h=h1; this.name= name; }

public void run()
    { h.hi1(name); }
}
```

```
class MyThread extends Thread
{
  Hello h;
  String name;
  MyThread(Hello h1, String name)
   { this.h=h1; this.name= name; }
  public void run()
    { h.hi(name); }
}

class SynchronizedDemo
{
public static void main(String[] args)
{
Hello h =new Hello();
MyThread t1 = new MyThread(h,"Neeraj");
t1.start();

MyThread1 t3 = new MyThread1(h,"Neeraj1");
t3.start();

}
}
```

Class level lock:

•Every class in java has a unique lock. If a thread wants to execute a **static synchronized method** then it required class level lock.

• Once a thread got class level lock then it is allow to execute any static synchronized method of that class.

• While a thread executing any static synchronized method the remaining threads are not allow to execute any static synchronized method of that class simultaneously. But remaining threads are allowed to execute normal synchronized methods, normal static methods,and normal instance methods simultaneously.

• Class level lock and object lock both are different and there is no relationship between these two.

```
class Hello
{
   public  static synchronized void hi(String name)
    {
   for(int i=0;i<10;i++)
```

```
    {
     System.out.print("Hello All  :");
      try { Thread.sleep(2000); }
     catch (InterruptedException e) { }
    System.out.println(name);
    }
}
}

class MyThread extends Thread
{
  Hello h;
  String name;
  MyThread(Hello h1, String name)
   { this.h=h1; this.name= name; }
  public void run()
    { h.hi(name); }
}

class StaticSynchronizedDemo
{
public static void main(String[] args)
{
Hello h =new Hello();
                            //Hello h1 =new Hello();
MyThread t1 = new MyThread(h,"Neeraj");
                            //MyThread t2 = new MyThread(h1,"Paliwal");
MyThread t3 = new MyThread(h,"Neeraj1");
                            //MyThread t4 = new MyThread(h1,"Paliwal1");
t1.start();
                            //t2.start();
t3.start();
                            //t4.start();

}
}
```

Synchronizedblock:

•If very few lines of the code required synchronization then it's never recommended to declare entire method as synchronized we have to enclose those few lines of the code with in synchronized block.
• The main advantage of synchronized block over synchronized method is, it reduces waiting time of thread and improves perforamnce of the system.

1. To get lock of current object we can declare synchronized block as follows.
   synchronized(this){}
2. To get the lock of a particular object 'b' we have to declare a synchronized block as follows.
   synchronized(b){}
3. To get class level lock we have to declare synchronized block as follows.
   Synchronized(Display.class){}

As the argument to the synchronized block we can pass eithere object reference or ".classfile" and we can't pass primitive values as argument [because lock concept is dependent only for objects and classes but not for primitives]

Example : To get lock of current object we can declare synchronized block

```
class Hello
{
   public void hi()
    {
     System.out.println("Hello 1   ");
    System.out.println("Hello 2  ");
   System.out.println("Hello 3   ");
   System.out.println();
   synchronized (this)
    {
    for(int i=0;i<10;i++)
      {
     System.out.print(i);
     try { Thread.sleep(2000); }
     catch (InterruptedException e) { }
       }
     }
System.out.println();
System.out.println("Hi 1");
System.out.println("Hi 2");
System.out.println("Hi 3");
System.out.println();
  }
}

class MyThread extends Thread
{ Hello h;
  MyThread(Hello h)
   { this.h=h; }
  public void run()
    { h.hi(); }
}
```

```
class MyThread1 extends Thread
{ Hello h;
  MyThread1(Hello h)
   { this.h=h; }
  public void run()
    { h.hi(); }
}

class SynchronizedDemo4
{
public static void main(String[] args)
{
Hello h =new Hello();
Hello h2 = new Hello();
MyThread t1 = new MyThread(h);
                          //MyThread t2 = new MyThread(h2);
                          //MyThread t2 = new MyThread(h2);
MyThread1 t2 = new MyThread1(h);
t1.start();
t2.start();
}
}
```

Example : To get the lock of a particular object 'b' we have to declare a synchronized block

```
class X
{
 void m()
{
for(int i=0;i<10;i++)
      {
    System.out.print(i);
    try { Thread.sleep(2000); }
    catch (InterruptedException e) { }
       }
 }
}
class Hello
{

   public void hi(X x)
    {
     System.out.print("Hello ");
    System.out.print("Hello ");
   System.out.print("Hello ");
   synchronized (x)
    {
     x.m();
    }
System.out.print("Hi ");
System.out.print("Hi ");
System.out.print("Hi ");
  }
}

class MyThread extends Thread
{ Hello h2;
   X x;
  MyThread(Hello h3,X x)
   { h2=h3; this.x=x;
     }
  public void run()
    { h2.hi(x); }
}
```

```
class SynchronizedBlockDemo

{

public static void main(String[] args)

{

X x1=new X();

Hello h1 =new Hello();

MyThread t1 = new MyThread(h1,x1);

MyThread t2 = new MyThread(h1,x1);

t1.start();

t2.start();

}

}
```

Example : To get class level lock we have to declare synchronized block.

```
class Hello
{
  public void hi()
   {
    System.out.print("Hello ");
   System.out.print("Hello 1 ");
  System.out.print("Hello 2");
  synchronized (Hello.class)  //synchronized Block for class level lock
   {
   for(int i=0;i<10;i++)
     {
    System.out.print(i);
    try { Thread.sleep(2000); }
    catch (InterruptedException e) { }
      }
    }
  }
}

class MyThread extends Thread
{ Hello h;
  MyThread(Hello h)
   { this.h=h; }
  public void run()
    { h.hi(); }
}

class SynchronizedBlockDemo
{
public static void main(String[] args)
{
Hello h =new Hello();
Hello h1 =new Hello();
MyThread t1 = new MyThread(h);
MyThread t2 = new MyThread(h1);
t1.start();
t2.start();
}
}
```

The statements which present inside synchronized method and synchronized block are called synchronized statements

Inter thread communication(wait(),notify(), notifyAll())

- Two threads can communicate with each other by using wait(), notify()and notifyAll() methods.
- The thread which is excepting updation it has to cal lwait() method and the thread which is performing updation it has to call notify() method. After getting notification the waiting thread will get those updations.
- wait(), notify() and notifyAll() methods are available in Object class but not in Thread class because thread can call these methods on any common object.
- To call wait(), notify() and notifyAll() methods compulsory the current thread should be owner of that object that is current thread should has lock of that object that is current thread should be in synchronized area. Hencewecancallwait(), notify() and notifyAll( ) methods only from synchronized area otherwise we will get runtime exception saying IllegalMonitorStateException.
- Once a thread calls wait() method on the given object 1st it releases the lock of that object immediately and entered into waiting state.
- Once a thread calls notify() (or) notifyAll() methods it releases the lock of that object but may not immediately.
- Except these (wait(),notify(),notifyAll()) methods there is no other place(method) where the lock release will be happen.

public final void wait() throws InterruptedException
public final native void wait(long ms) throws InterruptedException
public final void wait(long ms,int ns) throws InterruptedException
public final native void notify()
public final void notifyAll()

Example1:

```
class ThreadA  {
 public static void main(String[] args) throws InterruptedException
 {
  ThreadB b=new ThreadB();
   b.start();
   synchronized(b)
    {
    System.out.println("main Thread calling wait() method");    //step-1 b.wait();
    System.out.println("main Thread got notification call");   //step-4 System.out.println(b.total);
    }
}}

class  ThreadB extends Thread
{
   int total=0;
   public void run()
   {
```

```
 synchronized(this)
  {
  System.out.println("child thread starts calcuation"); //step-2
  for(inti=0;i<=100;i++)
 { total=total+i; }

System.out.println("child thread giving notification call");//step-3
this.notify();
}
}}
```

notify vs notifyAll() :

- We can use notify() method to give notification for only one thread. If multiple threads are waiting then only one thread will get the chance and remaining threads has to wait for further notification. But which thread will be notify(inform) we can't expect exactly itdependson JVM.
- We can use notifyAll() method to give the notification for all waiting threads. All waiting threads will be notified and will be executed one by one.