**Java synchronized keyword**

By Lokesh Gupta | Filed Under: [Multi Threading](https://howtodoinjava.com/java/multi-threading/)

**Java synchronized keyword** marks a block or method a critical section. A critical section is where one and only one thread is executing at a time, and the thread holds the lock for the synchronized section.

**synchronized** keyword helps in writing [concurrent](https://howtodoinjava.com/java-concurrency-tutorial/) parts of the applications, to protect shared resources within this block.

The synchronized keyword can be use with –

* a code block
* a method

**1. Java synchronized block**

**1.1. Syntax**

The general syntax for writing a synchronized block is as follows. Here **lockObject** is a reference to an object whose lock associates with the monitor that the synchronized statements represent.

|  |
| --- |
| Syntax |
| synchronized( lockObject )  {     // synchronized statements  } |

**1.2. Internal working**

When a thread wants to execute synchronized statements inside the synchronized block, it MUST acquire the lock on lockObject‘s monitor. At a time, only one thread can acquire the monitor of a lock object. So all other threads must wait till this thread, currently acquired the lock, finish it’s execution.

In this way, synchronized keyword guarantees that only one thread will be executing the synchronized block statements at a time, and thus prevent multiple threads from corrupting the shared data inside the block.

Keep in mind that if a thread is put on sleep (using sleep() method) then it does not release the lock. At this sleeping time, no thread will be executing the synchronized block statements.

Java synchronization will throw **NullPointerException** if lock object used in 'synchronized (lock)' is null.

**1.3. Java synchronized block example**

Java program to demonstrate the usage of synchronized block. In given example, we have a MathClass with a method printNumbers(). This method will print the numbers starting from 1 to the argument number N.

Notice that the code in printNumbers() method is inside synchronized block.

|  |
| --- |
| MathClass.java |
| public class MathClass  {      void printNumbers(int n) throws InterruptedException      {          synchronized (this)          {              for (int i = 1; i <= n; i++)              {                  System.out.println(Thread.currentThread().getName() + " :: "+  i);                  Thread.sleep(500);              }          }      }  } |

I have created two threads which start executing the printNumbers() method exactly at same time. Due to block being synchronized, only one thread is allowed access and other thread has to wait until first thread is finished.

|  |
| --- |
| Main.java |
| public class Main  {      public static void main(String args[])      {          final MathClass mathClass = new MathClass();            //first thread          Runnable r = new Runnable()          {              public void run()              {                  try {                      mathClass.printNumbers(3);                  } catch (InterruptedException e) {                      e.printStackTrace();                  }              }          };            new Thread(r, "ONE").start();          new Thread(r, "TWO").start();      }  } |

Program output.

|  |
| --- |
| Console |
| ONE :: 1  ONE :: 2  ONE :: 3    TWO :: 1  TWO :: 2  TWO :: 3 |

**2. Java synchronized method**

**2.1. Syntax**

The general syntax for writing a synchronized method is as follows. Here **lockObject** is a reference to an object whose lock associates with the monitor that the synchronized statements represent.

|  |
| --- |
| Syntax |
| <access modifier> synchronized method( parameters )  {      // synchronized code  } |

**2.2. Internal working**

Similar to synchronized block, a thread MUST acquire the lock on the associated monitor object with synchronized method. In case of synchronized method, the lock object is –

* **‘.class’ object** – if the method is static.
* **‘this’ object** – if the method is not static. ‘this’ refer to reference to current object in which synchronized method is invoked.

Read More : [Object level lock vs Class level lock in Java](https://howtodoinjava.com/java/multi-threading/object-vs-class-level-locking/)

Java synchronized keyword is **re-entrant** in nature it means if a 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.

**2.3. Java synchronized method example**

Similar to synchronized block example, we can apply synchronized keyword at printNumber()method and it will make the method as synchronized. Now if we again run the example, we will get the similar output.

|  |
| --- |
| MathClass.java |
| public class MathClass  {      synchronized void printNumbers(int n) throws InterruptedException      {          for (int i = 1; i <= n; i++)          {              System.out.println(Thread.currentThread().getName() + " :: "+  i);              Thread.sleep(500);          }      }  } |

Program output.

|  |
| --- |
| Console |
| ONE :: 1  ONE :: 2  ONE :: 3    TWO :: 1  TWO :: 2  TWO :: 3 |

Drop me your questions in comments.

Happy Learning !!