**1. Object level lock in Java**

**Object level lock** is mechanism when we 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**.

Object level locking can be done as below :

|  |
| --- |
| Various ways for object level locking |
| public class DemoClass  {      public synchronized void demoMethod(){}  }    or    public class DemoClass  {      public void demoMethod(){          synchronized (this)          {              //other thread safe code          }      }  }    or    public class DemoClass  {      private final Object lock = new Object();      public void demoMethod(){          synchronized (lock)          {              //other thread safe code          }      }  } |

**2. Class level lock in Java**

**Class level lock** prevents multiple threads to enter in synchronized block in any of all available instances of the class 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.

Class level locking should always be done **to make static data thread safe**. As we know that [**static**](https://howtodoinjava.com/java/basics/java-static-keyword/) keyword associate data of methods to class level, so use locking at static fields or methods to make it on class level.

|  |
| --- |
| Various ways for class level locking |
| public class DemoClass  {      //Method is static      public synchronized static void demoMethod(){        }  }    or    public class DemoClass  {      public void demoMethod()      {          //Acquire lock on .class reference          synchronized (DemoClass.class)          {              //other thread safe code          }      }  }    or    public class DemoClass  {      private final static Object lock = new Object();        public void demoMethod()      {          //Lock object is static          synchronized (lock)          {              //other thread safe code          }      }  } |

**3. Object level lock vs class level lock – 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 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 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 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](https://howtodoinjava.com/java/exception-handling/how-to-effectively-handle-nullpointerexception-in-java/) if object used in 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 synchronized keyword with constructor. It is 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.
10. Do not use String literals because they might be referenced else where in the application and can cause deadlock. String objects created with new keyword can be used safely. But as a best practice, create a new **private** scoped Object instance OR lock on the shared variable itself which we want to protect.