**Object level Locking vs. Class level Locking in Java**

In java there are two types of locks:

Class Level

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.

Object Level

In case of Static methods the lock is always checked on class but in case of instance methods the lock is always checked on object.

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

Example:

show1() is non static and show() is static. Now, show() is called by class name (or by object) and show1() is called by object, then both methods can accessed simultaneously by two threads.

class Shared{

static int x;

static synchronized void show(String s,int a){

x=a;

System.out.println("Starting in method "+s+" "+x);

try{

Thread.sleep(2000);

}

catch(Exception e){ }

System.out.println("Ending from method "+s+" "+x);

}

synchronized void show1(String s,int a){

x=a;

System.out.println("Starting show1 "+s);

try{

Thread.sleep(2000);

}

catch(Exception e){ }

System.out.println("Ending from show1 "+s);

}

}

class CustomThread extends Thread{

Shared s;

public CustomThread(Shared s,String str){

super(str);

this.s=s;

start();

}

public void run(){

Shared.show(Thread.currentThread().getName(),10);

}

}

class CustomThread1 extends Thread{

Shared s;

public CustomThread1(Shared s,String str){

super(str);

this.s=s;

start();

}

public void run(){

s.show1(Thread.currentThread().getName(),20);

}

}

public class RunSync {

public static void main(String[] args) {

Shared sh=new Shared();

CustomThread t1=new CustomThread(sh,"one");

CustomThread1 t2=new CustomThread1(sh,"two");

}

}

Output:

Starting in method one 10

Starting show1 two

Ending from method one 20

Ending from show1 two

**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 :

|  |
| --- |
| 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          }      }  } |

## **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.

|  |
| --- |
| public class DemoClass  {      public synchronized static void demoMethod(){}  }    or    public class DemoClass  {      public void demoMethod(){          synchronized (DemoClass.class)          {              //other thread safe code          }      }  }    or    public class DemoClass  {      private final static Object lock = new Object();      public void demoMethod(){          synchronized (lock)          {              //other thread safe code          }      }  } |