*** Synchronization :

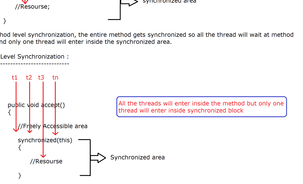
* Synchronization is a technique through which we can **control multiple threads but accepting only one thread at a time.**

* We can achieve synchronization by using "**synchronized**" keyword.

* Synchronized area is a **restricted area** so, only one thread is allowed to enter inside the synchronized area for a **single object.**

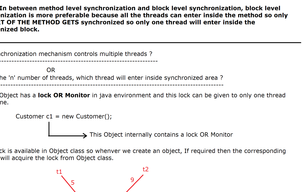
* It is divided into two categories :
  1) Method Level Synchronization
  2) Block Level Synchronization

Method Level Synchronization :



* In method level synchronization, the entire method gets synchronized so all the thread will wait at method level and only one thread will enter inside the synchronized area.

2) Block Level Synchronization :



*In block level synchronization the entire method does not get synchronized, only the part of the method gets synchronized so all the thread will enter inside the method but only one thread will enter inside the synchronized block.

Note :- In between method level synchronization and block level synchronization, block level synchronization is more preferable because all the threads can enter inside the method so only the PART OF THE METHOD GETS synchronized so only one thread will enter inside the synchronized block.

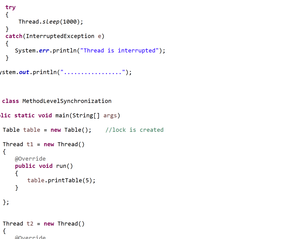
How synchronization mechanism controls multiple threads ?

OR

Among the 'n' number of threads, which thread will enter inside synchronized area ?

* Every Object has a **lock OR Monitor** in java environment and this lock can be given to only one thread at a time.

* This lock is available in Object class so wherever we create an object, If required then the corresponding object will acquire the lock from Object class.



The thread who acquires the lock from the corresponding object will enter inside the synchronized area, it will complete its task without any disturbance because at a time there will be only one thread inside the synchronized area(for single Object). *This is known as Thread-safety in java.

The thread which is inside the synchronized area, after completion of its task while going back will release the lock so the other threads (which are waiting outside for the lock) will get a chance to enter inside the synchronized area by again taking the lock from the object and submitting it to the synchronization mechanism.

This is how synchronization mechanism controls multiple Threads.

Note :- Synchronization logic can be done by senior programmers in the real time industry because due to poor synchronization there may be chance of getting deadlock.

//Program on Method Level Synchronization :

```
package com.ravi.synchronization;

class Table
{
  public synchronized void printTable(int num)
  {
    for(int i=1; i<=10; i++)
    {
       System.out.println(num+" X "+i+" = "+(num*i));

       try
       {
           Thread.sleep(1000);
       }
       catch(InterruptedException e)
       {
          System.err.println("Thread is interrupted");
       }
    }
    System.out.println("---------------");
  }
}

public class MethodLevelSynchronization
{
   public static void main(String[] args)
   {
       Table table = new Table();    //lock is created

       Thread t1 = new Thread()
       {
          @Override
          public void run()
          {
             table.printTable(5);
          }

       };


       Thread t2 = new Thread()
       {
          @Override
          public void run()
          {
             table.printTable(9);
          }

       };

       t1.start();  t2.start();

   }

}
```

//Program on block level synchronization :

```
package com.ravi.synchronization;

class PrintThread
{
   public void printThreadName()
   {
       String name = Thread.currentThread().getName();
       System.out.println("Thread Name is :"+name);

       synchronized(this)
       {
          for(int i=1; i<=10; i++)
          {
             System.out.println(i+" by "+name+" thread");
             try
             {
                Thread.sleep(500);
             }
             catch(InterruptedException e)
             {
                System.err.println("Thread is Interrupted");
             }
          }
          System.out.println("Synchronized block ended!!");

       }
   }
}

public class BlockLevelSynchronization
{
   public static void main(String[] args)
   {
       PrintThread pt = new PrintThread();

       Runnable r1 = () -> pt.printThreadName();


       Thread t1 = new Thread(r1,"Child1");
       Thread t2 = new Thread(r1,"Child2");

       t1.start();  t2.start();
   }
}
```
