Thread Synchronization

- It is necessary to synchronize the activities of the various threads
    - all threads of a process share the same address space and other resources
    - any alteration of a resource by one thread affects the other threads in the same process

Synchronization in Java

- Synchronization in java is the capability to control the access of multiple threads to any shared resource.

- Java Synchronization is better option where we want to allow only one thread to access the shared resource.

Thread Synchronization

Mutual Exclusive helps keep threads from interfering with one another while sharing data. This can be done by three ways in java:

- by synchronized method
- by synchronized block
- by static synchronization

Understanding the problem without Synchronization

```
Class Table{
 void printTable(int n){            //method not synchronized
  for(int i=1;i<=5;i++){
    System.out.println(n*i);
    try{
     Thread.sleep(400);
    }catch(Exception e){System.out.println(e);}
  }


 }
}
```

Thread creation

```
class MyThread1 extends Thread{

Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}
```

```
class MyThread2 extends Thread{

Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}
```

Output: 5 100 10 200 15 300 20 400 25 500

Cont…

```
class TestSynchronization1{
        public static void main(String args[]){
        Table obj = new Table();//only one object
        MyThread1 t1=new MyThread1(obj);
        MyThread2 t2=new MyThread2(obj);
        t1.start();
        t2.start();
        }
}
```

```
Output:
 5
100
10
200
15
300
20
400
25
500
```

Java synchronized method

- If you declare any method as synchronized, it is known as synchronized method.
- Synchronized method is used to lock an object for any shared resource.
- When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

Synchronized Method

```
//example of java
class Table{
 synchronized void printTable(int n){//synchronized method
   for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){System.out.println(e);}
   }
   }
}
```

```
Output:
 5
 10
15
20
25
100
200
300
400
 500
```

Synchronized block in java

- Synchronized block can be used to perform synchronization on any specific resource of the method.
- Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.
- If you put all the codes of the method in the synchronized block, it will work same as the synchronized method.

Points to remember for Synchronized block

- Synchronized block is used to lock an object for any shared resource.
- Scope of synchronized block is smaller than the method.

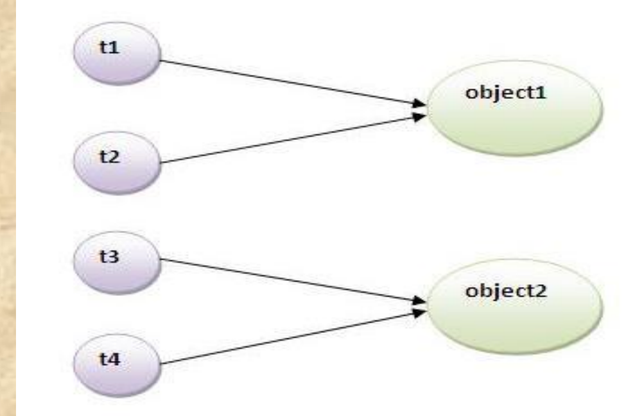
Synchronized Block

```
class Table{
  void printTable(int n){
  synchronized(this){//synchronized block
    for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){System.out.println(e);}
    }
   }
 }//end of the method
}
```

```
Output:
 5
 10
15
20
25
100
200
300
400
 500
```

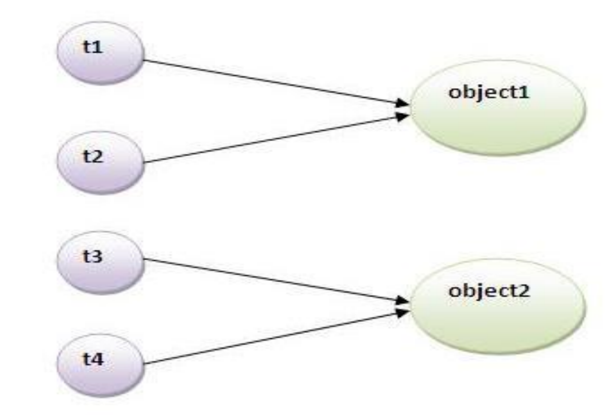
Static synchronization

- If you make any static method as synchronized, the lock will be on the class not on object.



- In case of synchronized method and synchronized block there cannot be interference between t1 and t2 or t3 and t4 because t1 and t2 both refers to a common object that have a single lock.

Static synchronization



- But there can be interference between t1 and t3 or t2 and t4 because t1 acquires another lock and t3 acquires another lock.I want no interference between t1 and t3 or t2 and t4.Static synchronization solves this problem.

Cont…

```
Class Table{
 synchronized static void printTable(int n){
            //method not synchronized
  for(int i=1;i<=5;i++){
    System.out.println(n*i);
    try{
     Thread.sleep(400);
    }catch(Exception e){System.out.println(e);}
  }


 }
}
```

Thread creation

```
class MyThread1 extends Thread
{
public void run(){
Table.printTable(1);
}
}
```

```
class MyThread2 extends Thread
{
public void run(){
Table.printTable(5);
}
}
```

```
class MyThread3 extends Thread
{
public void run(){
Table.printTable(10);
}
}
```

```
class MyThread4 extends Thread
{
public void run(){
Table.printTable(15);
}
}
```

Output: 5 100 10 200 15 300 20 400 25 500

Cont…

```
public class TestSynchronization4{
public static void main(String t[]){
MyThread1 t1=new MyThread1();
MyThread2 t2=new MyThread2();
MyThread3 t3=new MyThread3();
MyThread4 t4=new MyThread4();
t1.start();
t2.start();
t3.start();
t4.start();
}
}
```

```
Output:   Output:
1         10
2         20
3         30
4         40
5         50

5         15
10        30
15        45
20        60
25        75
```