How to perform single task by multiple threads?

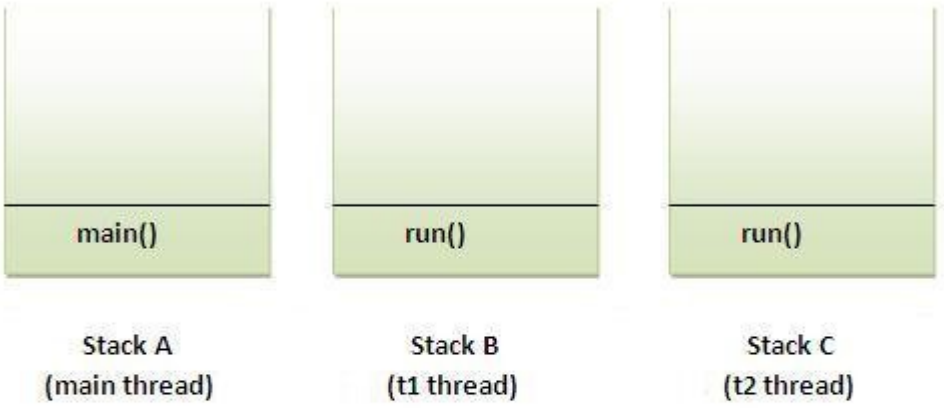
If you have to perform single task by many threads, have only one run() method.For example:

Program of performing single task by multiple threads

```
1.  class TestMultitasking1 extends Thread{
2.   public void run(){
3.     System.out.println("task one");
4.   }
5.   public static void main(String args[]){
6.    TestMultitasking1 t1=new TestMultitasking1();
7.    TestMultitasking1 t2=new TestMultitasking1();
8.
9.
10.  t1.start();
11.  t2.start();
12.
13. }
14.}
```

```
Output:
       task one
       task one
```

Note: Each thread run in a separate callstack.



How to perform multiple tasks by multiple threads (multitasking in multithreading)?

If you have to perform multiple tasks by multiple threads,have multiple run() methods.For example:

Program of performing two tasks by two threads

```
1.  class Simple1 extends Thread{
2.   public void run(){
3.     System.out.println("task one");
4.   }
5.  }
6.
7.  class Simple2 extends Thread{
8.   public void run(){
9.     System.out.println("task two");
10. }
11.}
12.
13. class TestMultitasking3{
14. public static void main(String args[]){
15.  Simple1 t1=new Simple1();
16.  Simple2 t2=new Simple2();
17.
18.  t1.start();
19.  t2.start();
20. }
21.}
```

```
Output:task one
       task two
```

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.

Why use Synchronization

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

Types of Synchronization

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

Here, we will discuss only thread synchronization.

Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive
    1. Synchronized method.
    2. Synchronized block.
    3. static synchronization.
2. Cooperation (Inter-thread communication in java)

Mutual Exclusive

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

1. by synchronized method
2. by synchronized block
3. by static synchronization

Concept of Lock in Java

Synchronization is built around an internal entity known as the lock or monitor. Every object has an lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.

From Java 5 the package java.util.concurrent.locks contains several lock implementations.

Understanding the problem without Synchronization

In this example, there is no synchronization, so output is inconsistent. Let's see the example:

1. Class Table{

```
2.
3.  void printTable(int n){//method not synchronized
4.    for(int i=1;i<=5;i++){
5.      System.out.println(n*i);
6.      try{
7.       Thread.sleep(400);
8.      }catch(Exception e){System.out.println(e);}
9.    }
10.
11. }
12.}
13.
14.class MyThread1 extends Thread{
15.Table t;
16.MyThread1(Table t){
17.this.t=t;
18.}
19.public void run(){
20.t.printTable(5);
21.}
22.
23.}
24.class MyThread2 extends Thread{
25.Table t;
26.MyThread2(Table t){
27.this.t=t;
28.}
29.public void run(){
30.t.printTable(100);
31.}
32.}
33.
34.class TestSynchronization1{
35.public static void main(String args[]){
36.Table obj = new Table();//only one object
37.MyThread1 t1=new MyThread1(obj);
38.MyThread2 t2=new MyThread2(obj);
39.t1.start();
40.t2.start();
41.}
42.}
```

```
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.

```
//example of java synchronized method
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);}
   }

 }
}

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);
}
}

public class TestSynchronization2{
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
        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.

Syntax to use synchronized block

1. synchronized(object reference expression) {
2. //code block
3. }

Example of synchronized block

Let's see the simple example of synchronized block.

Program of synchronized block

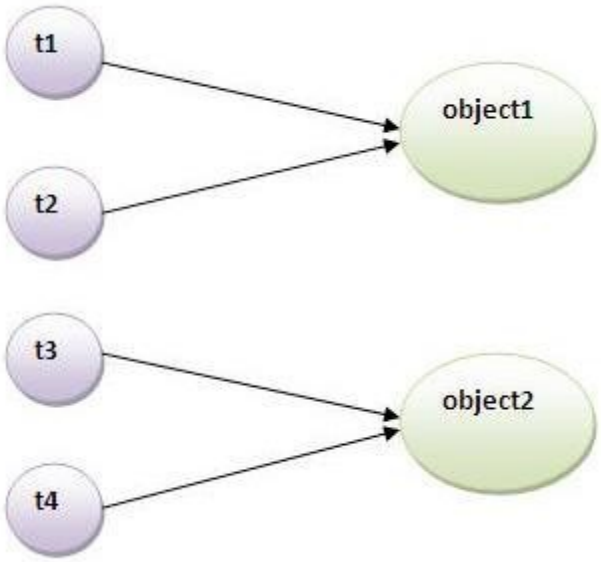
```
class Table{

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
}

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);
}
}

public class TestSynchronizedBlock1{
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
        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.



Problem without static synchronization

Suppose there are two objects of a shared class(e.g. Table) named object1 and object2.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.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.

Example of static synchronization

In this example we are applying synchronized keyword on the static method to perform static synchronization.

```
Class Table{

synchronized static void printTable(in n){
for(int i=1;i<=10;i++){
System.out.println(n*i);
try{
Thread.sleep(400);
}catch(Exception e){}
}
}
}

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

class MyThread2 extend Thread{
public void run(){
```

```
Table.printTable(10);
}
}

class MyThread3 extends Thread{
public void run(){
Table.printTable(100);
}
}



class MyThread4 extends Thread{
public void run(){
Table.printTable(1000);
}
}

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();
}
}
```

Test it Now

```
Output: 1
        2
        3
        4
        5
        6
        7
        8
        9
        10
        10
        20
        30
        40
        50
        60
        70
        80
        90
        100
        100
        200
        300
        400
```

500
600
700
800
900
1000
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000