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 critical section

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

### **Thread Synchronization**

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

1. Mutual Exclusive: one process/thread can enter critical section
   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 a 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:

Without any interruption No sleep() method

class Table {

public void printTable(int n){

for(int i=1;i<=5;i++){

System.out.println(n\*i);

}

}

}

class Thread1 extends Thread{

Table t;

Thread1(Table t){

this.t=t;

}

public void run(){

t.printTable(5);

}

}

class Thread2 extends Thread{

Table t;

Thread2(Table t){

this.t=t;

}

public void run(){

t.printTable(100);

}

}

class SyncTable{

public static void main(String args[]){

Table obj = new Table();

Thread1 t1 = new Thread1(obj);

Thread2 t2 = new Thread2(obj);

t1.start();

t2.start();

}

}

Output

5

10

15

20

25

100

200

300

400

500

Without Synchronization But with sleep

class Table {

public void printTable(int n){

for(int i=1;i<=5;i++){

System.out.println(n\*i);

try{

Thread.sleep(100);

}

catch(Exception e){

System.out.println(e);

}

}

}

}

class Thread1 extends Thread{

Table t;

Thread1(Table t){

this.t=t;

}

public void run(){

t.printTable(5);

}

}

class Thread2 extends Thread{

Table t;

Thread2(Table t){

this.t=t;

}

public void run(){

t.printTable(100);

}

}

class SyncTable{

public static void main(String args[]){

Table obj = new Table();

Thread1 t1 = new Thread1(obj);

Thread2 t2 = new Thread2(obj);

t1.start();

t2.start();

}

}

Output

5

100

10

200

15

300

400

20

500

25

With Synchronization

class Table {

public synchronized void printTable(int n){

for(int i=1;i<=5;i++){

System.out.println(n\*i);

try{

Thread.sleep(100);

}

catch(Exception e){

System.out.println(e);

}

}

}

}

class Thread1 extends Thread{

Table t;

Thread1(Table t){

this.t=t;

}

public void run(){

t.printTable(5);

}

}

class Thread2 extends Thread{

Table t;

Thread2(Table t){

this.t=t;

}

public void run(){

t.printTable(100);

}

}

class SyncTable{

public static void main(String args[]){

Table obj = new Table();

Thread1 t1 = new Thread1(obj);

Thread2 t2 = new Thread2(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**

**synchronized** (object reference expression) {

  //code block

}

class Table{

public void PrintTable(int n){

synchronized(this){

for(int i=1;i<=5;i++){

System.out.println(n\*i);

try{

Thread.sleep(300);

}

catch(Exception e){

System.out.println(e);

}

}

}

}

}

class Thread1 extends Thread{

Table t;

Thread1(Table t){

this.t = t;

}

public void run(){

t.PrintTable(5);

}

}

class Thread2 extends Thread{

Table t;

Thread2(Table t){

this.t = t;

}

public void run(){

t.PrintTable(100);

}

}

class SynBlock {

public static void main(String args[]){

Table obj = new Table();

Thread1 t1 = new Thread1(obj);

Thread2 t2 = new Thread2(obj);

t1.start();

t2.start();

}

}

Output

5

10

15

20

25

100

200

300

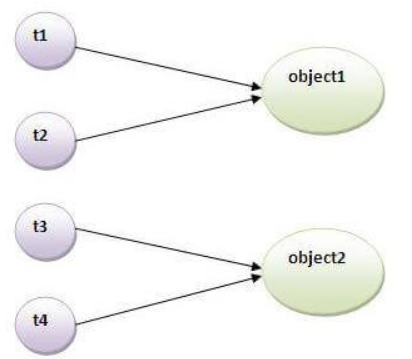
400

500

Problems with above 2 methods of synchronization

# 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.

class Table{

public void PrintTable(int n){

synchronized(this){

for(int i=1;i<=5;i++){

System.out.println(n\*i);

try{

Thread.sleep(300);

}

catch(Exception e){

System.out.println(e);

}

}

}

}

}

class Thread1 extends Thread{

Table t;

Thread1(Table t){

this.t = t;

}

public void run(){

t.PrintTable(5);

}

}

class Thread2 extends Thread{

Table t;

Thread2(Table t){

this.t = t;

}

public void run(){

t.PrintTable(10);

}

}

class Thread3 extends Thread{

Table t;

Thread3(Table t){

this.t = t;

}

public void run(){

t.PrintTable(100);

}

}

class Thread4 extends Thread{

Table t;

Thread4(Table t){

this.t = t;

}

public void run(){

t.PrintTable(1000);

}

}

class SynBlock {

public static void main(String args[]){

Table obj1 = new Table();

Table obj2 = new Table();

Thread1 t1 = new Thread1(obj1);

Thread2 t2 = new Thread2(obj1);

Thread3 t3 = new Thread3(obj2);

Thread4 t4 = new Thread4(obj2);

t1.start();

t2.start();

t3.start();

t4.start();

}

}

Output

5

100

10

200

15

300

20

400

25

500

10

1000

20

2000

30

3000

40

4000

50

5000

After applying Static synchronization

class Table{

public static synchronized void PrintTable(int n){

//synchronized(Table.class){

for(int i=1;i<=5;i++){

System.out.println(n\*i);

try{

Thread.sleep(300);

}

catch(Exception e){

System.out.println(e);

}

}

//}

}

}

class Thread1 extends Thread{

Table t;

Thread1(Table t){

this.t = t;

}

public void run(){

t.PrintTable(5);

}

}

class Thread2 extends Thread{

Table t;

Thread2(Table t){

this.t = t;

}

public void run(){

t.PrintTable(10);

}

}

class Thread3 extends Thread{

Table t;

Thread3(Table t){

this.t = t;

}

public void run(){

t.PrintTable(100);

}

}

class Thread4 extends Thread{

Table t;

Thread4(Table t){

this.t = t;

}

public void run(){

t.PrintTable(1000);

}

}

class SynBlock {

public static void main(String args[]){

Table obj1 = new Table();

Table obj2 = new Table();

Thread1 t1 = new Thread1(obj1);

Thread2 t2 = new Thread2(obj1);

Thread3 t3 = new Thread3(obj2);

Thread4 t4 = new Thread4(obj2);

t1.start();

t2.start();

t3.start();

t4.start();

}

}

Output

5

10

15

20

25

1000

2000

3000

4000

5000

100

200

300

400

500

10

20

30

40

50