Thread

END

Q What do you understand by thread-safety? Why is it required? How to achieve it.

Thread-Safety in Java:— Any programming statements that are executing by many threads simultaneously. In other words we can say that thread-safe code is a code that will work even if many threads are executing it simultaneously.

05 Sunday

There are three ways to achieve this:—

(i) Synchronization
(ii) Use of locks
(iii) Using thread-safe collection classes.

(i) Synchronization → Only one thread can access a code at a time. This is performed by programmer in order to safety of a shared resource or block or function.

Let us take an example that we have a method m1() and two threads are trying to access it simultaneously, then we have to put synchronized keyword while declaring that method as follows:

```
Synchronized void m1(int i)
{
for( int j=0; j<100; j++)
{
System.out.print(j + " ");
}
}
```

only one can access at a time

Thread2

Thread1

by placing synchronized keyword before void m1(int i), we have locked this method for accessing simultaneously by more than one thread. Only 1 thread can get access to the method m1() at a

time. If one thread performs its work then another will work on that method.

In this way we have specified a lock, so that other threads are in waiting condition while one thread is trying to access this method.

"Synchronized" Keyword :— This this a modifier used to solve data inconcistency problem occured due to more applications are trying to modify or access a code or function.

eg:- Your Bank A/c balance should not be updated at a time by two sources such as ATM, Mobile, Laptop etc. Only one source can aquire lock to change it at a time.

→ Synchronized is applicable only for methods and blocks.

→ Synchronized is not applicable for classes and variables.

⇒ Synchronized keyword in java creates a block of code known as critical section.

⇒ To enter a critical section, thread need to obtain the corresponding object's lock.

Let us take an example to see the difference between normal and synchronized methods:-

Program without synchronized keyword:-

```
class A
{
synchronized void addnew(int i)
{
    Thread t = Thread.currentThread();
    for(int n=1; n<=5; n++)
    {
    System.out.println(t.getName() + "-" +
                       (i+n));
    }
}
}

Class B extends Thread
{
    A a1 = new A();
    public void run()
    {
        a1.addnew(100);
```

Notes

```
class Syntest{
public static void main(String args[])
{
    B b = new B();
Thread t1 = new Thread(b);
Thread t2 = new Thread(b);
t1.setName("T1");
t2.setName("T2");
t1.start();
t2.start();
}
}
```

Execution of this program is as follows:-

We have created two threads on the same object — t1 and t2 and started both of them to run. The output is in consistent as:-

```
T1 - 101
T2 - 101
T2 - 102
T1 - 102
```

→ without synchronized addnew()

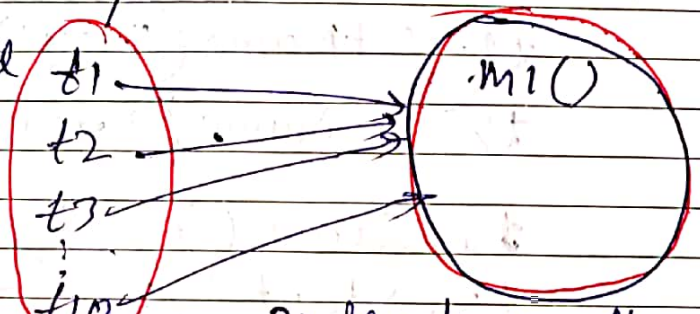
If we put synchronized keyword before addnew() method then the output will be normal.

Synchronization solves the problem of inconsistency but it also has some problems as:—

It increases the waiting time.

Suppose we have 10 threads t1,---t10 trying to access a synchronized method m1()

Suppose one thread takes 5 seconds to work on m1() then others will have to wait for its turn to come and hence t10 gets its turn after around 45 seconds waiting time.



20 seconds one thread then others are waiting

— Effects performance of the system.

Hence few lines of code can be synchronized to maintain the performance of the system wherever very necessary.

Notes

```
Synchronized (object)
{
==========
}
```

Today we will see the uses of Sleep() method in java -

In java multi-threading we can use Sleep method to pause the execution of a thread for specified milli-seconds. It causes the currently running thread to pause or block for atleast the specified number of milli-seconds.

One thing is very important to note that this statement (method) can arise exception, so that we have to call this function in try block only and catch block handles the exception.

Example :- A.java

```
class A extends Thread
{
    public void run()
    {
        for(int i=0; i<=10; i++)
        {
```

```
try {
        if (i==2) sleep(1000);
    }
    catch(Exception e) {e.printStackTrace();}
    } //for closed
    System.out.println(" A:"+i);
} //closed run()
public static void main(String args[])
    {
        A a = new A();
        a.start();
    }
} //closed class A
```

Output:— A:0
—1000 ms pause—
A:1
A:2
A:3
- - -
A:10

Note:- The type of exception sleep() method is throwing is — **InterruptedException**

Notes

Use of stop() in java multi-threading :—

stop() method kills the thread on execution.

APRIL

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

MAY

S	M	T	W	T	F	S
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

Let us see with the following example:-

```
class A extends Thread
{
    public void run()
    {
        for(int i=1; i<=10; i++)
        {
            if(i==5) stop();
            System.out.println("A:"+i);
        }
    }
public static void main(String args[])
{
    A a = new A();
    a.start();
}
```

After Compiling & Running the o/p will be as follows:-

```
A:1
A:2
A:3
A:4
```

As stop() method is called the thread is killed, after that no any statement will run.

24 Friday April

Q) Can we start a thread twice? Ans → NO

as in the example below:— A.java

```
class A extends Thread
{
    public void run()
    {
        System.out.println("Thread A");
    }
    public static void main(String a[])
    {
        A a = new A();
        a.start();
        //a.start();
    }
}
```

We cannot call start() more than one times because it throws IllegalThreadStateException.

You can try the above example and see the particular error on compilation.

Notes