# Thread Scheduler in Java

**Thread scheduler** in java is the part of the JVM that **decides which thread should run**.

There is no guarantee that which runnable thread will be chosen to run by the thread scheduler.

Only one thread at a time can run in a single process.

# Sleep method in java

The sleep() method of Thread class is used to sleep a thread for the specified amount of time.

## **Syntax of sleep() method in java**

The Thread class provides two methods for sleeping a thread:

* public static void sleep(long miliseconds)throws InterruptedException
* public static void sleep(long miliseconds, int nanos)throws InterruptedException

## **Example of sleep method in java**

1. **class** TestSleepMethod1 **extends** Thread{
2. **public** **void** run(){
3. **for**(**int** i=1;i<5;i++)
4. {
5. **Try** { Thread.sleep(500); }
6. **catch**(InterruptedException e){System.out.println(e);}
7. System.out.println(i);
8. }
9. }
10. **public** **static** **void** main(String args[]){
11. TestSleepMethod1 t1=**new** TestSleepMethod1();
12. TestSleepMethod1 t2=**new** TestSleepMethod1();
14. t1.start();
15. t2.start();
17. }
18. }

Output:

1

1

2

2

3

3

4

4

As you know well that at a time only one thread is executed. If you sleep a thread for the specified time,the thread shedular picks up another thread and so on.

Can we start a thread twice

No. After starting a thread, it can never be started again. If you does so, an *IllegalThreadStateException* is thrown. In such case, thread will run once but for second time, it will throw exception.

Let's understand it by the example given below:

1. **public** **class** TestThreadTwice1 **extends** Thread{
2. **public** **void** run(){
3. System.out.println("running...");
4. }
5. **public** **static** **void** main(String args[]){
6. TestThreadTwice1 t1=**new** TestThreadTwice1();
7. t1.start();
8. t1.start();
9. }
10. }

running

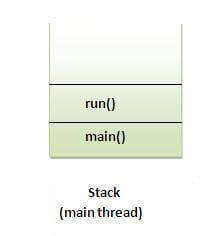
Exception in thread "main" java.lang.IllegalThreadStateException

# What if we call run() method directly instead start() method?

|  |
| --- |
| * Each thread starts in a separate call stack. * Invoking the run() method from main thread, the run() method goes onto the current call stack rather than at the beginning of a new call stack. |

1. **class** TestCallRun1 **extends** Thread{
2. **public** **void** run(){
3. System.out.println("running...");
4. }
5. **public** **static** **void** main(String args[]){
6. TestCallRun1 t1=**new** TestCallRun1();
7. t1.run();//fine, but does not start a separate call stack
8. }
9. }

Output:running...

 ***Problem if you direct call run() method***

1. **class** TestCallRun2 **extends** Thread{
2. **public** **void** run(){
3. **for**(**int** i=1;i<5;i++){
4. **try**{Thread.sleep(500);}**catch**(InterruptedException e){System.out.println(e);}
5. System.out.println(i);
6. }
7. }
8. **public** **static** **void** main(String args[]){
9. TestCallRun2 t1=**new** TestCallRun2();
10. TestCallRun2 t2=**new** TestCallRun2();
12. t1.run();
13. t2.run();
14. }
15. }

Output:1

2

3

4

5

1

2

3

4

5

|  |
| --- |
| As you can see in the above program that there is no context-switching because here t1 and t2 will be treated as normal object not thread object. |

# Naming Thread and Current Thread

## **Naming Thread**

The Thread class provides methods to change and get the name of a thread. By default, each thread has a name i.e. thread-0, thread-1 and so on. By we can change the name of the thread by using setName() method. The syntax of setName() and getName() methods are given below:

1. **public String getName():** is used to return the name of a thread.
2. **public void setName(String name):** is used to change the name of a thread.

## **Example of naming a thread**

1. **class** TestMultiNaming1 **extends** Thread{
2. **public** **void** run(){
3. System.out.println("running...");
4. }
5. **public** **static** **void** main(String args[]){
6. TestMultiNaming1 t1=**new** TestMultiNaming1();
7. TestMultiNaming1 t2=**new** TestMultiNaming1();
8. System.out.println("Name of t1:"+t1.getName());
9. System.out.println("Name of t2:"+t2.getName());
11. t1.start();
12. t2.start();
14. t1.setName("Sonoo Jaiswal");
15. System.out.println("After changing name of t1:"+t1.getName());
16. }
17. }

Output:Name of t1:Thread-0

Name of t2:Thread-1

id of t1:8

running...

After changeling name of t1:Sonoo Jaiswal

running...

## **Current Thread**

The currentThread() method returns a reference of currently executing thread.

1. **public** **static** Thread currentThread()

### **Example of currentThread() method**

1. **class** TestMultiNaming2 **extends** Thread{
2. **public** **void** run(){
3. System.out.println(Thread.currentThread().getName());
4. }
5. **public** **static** **void** main(String args[]){
6. TestMultiNaming2 t1=**new** TestMultiNaming2();
7. TestMultiNaming2 t2=**new** TestMultiNaming2();
9. t1.start();
10. t2.start();
11. }
12. }

Output:Thread-0

Thread-1