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.

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.      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.    TestSleepMethod1 t1=new TestSleepMethod1();
10.   TestSleepMethod1 t2=new TestSleepMethod1();
11.
12.   t1.start();
13.   t2.start();
14.  }
15. }
```

Output:

```
1
1
2
2
3
3
4
4
```

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. }
```

Test it Now

```
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.

```
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. }
```

Test it Now

```
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();
11.
12.   t1.run();
13.   t2.run();
14.  }
15. }
```

```
Output:1
       2
       3
       4
       5
       1
       2
       3
       4
       5
```

The join() method

The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.

Syntax:

```
public void join()throws InterruptedException

public void join(long milliseconds)throws InterruptedException
```

Example of join() method

```
1.  class TestJoinMethod1 extends Thread{
2.   public void run(){
3.    for(int i=1;i<=5;i++){
```

```
    try{
     Thread.sleep(500);
    }catch(Exception e){System.out.println(e);}
    System.out.println(i);
    }
   }
public static void main(String args[]){
 TestJoinMethod1 t1=new TestJoinMethod1();
 TestJoinMethod1 t2=new TestJoinMethod1();
 TestJoinMethod1 t3=new TestJoinMethod1();
 t1.start();
 try{
  t1.join();
 }catch(Exception e){System.out.println(e);}

 t2.start();
 t3.start();
 }
}
```

Test it Now

```
Output:1
       2
       3
       4
       5
       1
       1
       2
       2
       3
       3
       4
       4
       5
       5
```

As you can see in the above example,when t1 completes its task then t2 and t3 starts executing.

Example of join(long miliseconds) method

```
class TestJoinMethod2 extends Thread{
 public void run(){
  for(int i=1;i<=5;i++){
   try{
    Thread.sleep(500);
   }catch(Exception e){System.out.println(e);}
   System.out.println(i);
  }
```

```
    }
public static void main(String args[]){
 TestJoinMethod2 t1=new TestJoinMethod2();
 TestJoinMethod2 t2=new TestJoinMethod2();
 TestJoinMethod2 t3=new TestJoinMethod2();
 t1.start();
 try{
  t1.join(1500);
 }catch(Exception e){System.out.println(e);}

 t2.start();
 t3.start();
 }
}
```

Test it Now

```
Output:1
       2
       3
       1
       4
       1
       2
       5
       2
       3
       3
       4
       4
       5
       5
```

In the above example,when t1 is completes its task for 1500 miliseconds(3 times) then t2 and t3 starts executing.

---

getName(),setName(String) and getId() method:

```
public String getName()

public void setName(String name)

public long getId()
```

```
class TestJoinMethod3 extends Thread{
  public void run(){
   System.out.println("running...");
  }
 public static void main(String args[]){
  TestJoinMethod3 t1=new TestJoinMethod3();
```

```
7.    TestJoinMethod3 t2=new TestJoinMethod3();
8.    System.out.println("Name of t1:"+t1.getName());
9.    System.out.println("Name of t2:"+t2.getName());
10.   System.out.println("id of t1:"+t1.getId());
11.
12.   t1.start();
13.   t2.start();
14.
15.   t1.setName("Sonoo Jaiswal");
16.   System.out.println("After changing name of t1:"+t1.getName());
17.  }
18. }
```

Test it Now

```
Output:Name of t1:Thread-0
       Name of t2:Thread-1
       id of t1:8
       running...
       After changling name of t1:Sonoo Jaiswal
       running...
```

The currentThread() method:

The currentThread() method returns a reference to the currently executing thread object.

Syntax:

public static Thread currentThread()

Example of currentThread() method

```
1.  class TestJoinMethod4 extends Thread{
2.   public void run(){
3.    System.out.println(Thread.currentThread().getName());
4.   }
5.   }
6.   public static void main(String args[]){
7.    TestJoinMethod4 t1=new TestJoinMethod4();
8.    TestJoinMethod4 t2=new TestJoinMethod4();
9.
10.   t1.start();
11.   t2.start();
12.  }
13. }
```

Test it Now

```
Output:Thread-0
       Thread-1
```

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());
10.
11.   t1.start();
12.   t2.start();
13.
14.   t1.setName("Sonoo Jaiswal");
15.   System.out.println("After changing name of t1:"+t1.getName());
16.  }
17. }
```

Test it Now

```
Output:Name of t1:Thread-0
       Name of t2:Thread-1
       id of t1:8
       running...
       After changeling name of t1:Sonoo Jaiswal
       running...
```

Priority of a Thread (Thread Priority):

Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

3 constants defined in Thread class:

1. public static int MIN_PRIORITY
2. public static int NORM_PRIORITY
3. public static int MAX_PRIORITY

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

Example of priority of a Thread:

```
1.  class TestMultiPriority1 extends Thread{
2.   public void run(){
3.    System.out.println("running thread name is:"+Thread.currentThread().getName());
4.    System.out.println("running thread priority is:"+Thread.currentThread().getPriority());
5.
6.    }
7.   public static void main(String args[]){
8.    TestMultiPriority1 m1=new TestMultiPriority1();
9.    TestMultiPriority1 m2=new TestMultiPriority1();
10.  m1.setPriority(Thread.MIN_PRIORITY);
11.  m2.setPriority(Thread.MAX_PRIORITY);
12.  m1.start();
13.  m2.start();
14.
15. }
16. }
```

Test it Now

```
Output:running thread name is:Thread-0
       running thread priority is:10
       running thread name is:Thread-1
       running thread priority is:1
```