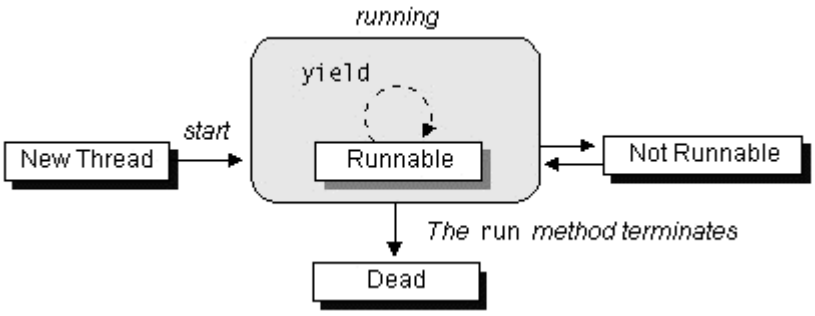
Java Concurrency – yield(), sleep() and join() methods

We can prevent the execution of a thread by using one of the following methods of Thread class.

1. **yield():** Suppose there are three threads t1, t2, and t3. Thread t1 gets the processor and starts its execution and thread t2 and t3 are in Ready/Runnable state. Completion time for thread t1 is 5 hour and completion time for t2 is 5 minutes. Since t1 will complete its execution after 5 hours, t2 has to wait for 5 hours to just finish 5 minutes job. In such scenarios where one thread is taking too much time to complete its execution, we need a way to prevent execution of a thread in between if something important is pending. yeild() helps us in doing so.
   yield() basically means that the thread is not doing anything particularly important and if any other threads or processes need to be run, they should run. Otherwise, the current thread will continue to run.



Use of yield method:

- Whenever a thread calls java.lang.Thread.yield method, it gives hint to the thread scheduler that it is ready to pause its execution. Thread scheduler is free to ignore this hint.

- If any thread executes yield method , thread scheduler checks if there is any thread with same or high priority than this thread. If processor finds any thread with higher or same priority then it will move the current thread to Ready/Runnable state and give processor to other thread and if not – current thread will keep executing.

Syntax:

```
public static native void yield()
```

```
// Java program to illustrate yield() method
// in Java
import java.lang.*;

// MyThread extending Thread
class MyThread extends Thread
{
    public void run()
    {
        for (int i=0; i<5 ; i++)
            System.out.println(Thread.currentThread().getName()
                               + " in control");
    }
}

// Driver Class
public class yieldDemo
{
    public static void main(String[]args)
    {
        MyThread t = new MyThread();
        t.start();

        for (int i=0; i<5; i++)
        {
            // Control passes to child thread
            Thread.yield();

            // After execution of child Thread
            // main thread takes over
            System.out.println(Thread.currentThread().getName()
                               + " in control");
        }
    }
}
```

Run on IDE

Output:

```
Thread-0 in control
Thread-0 in control
Thread-0 in control
Thread-0 in control
Thread-0 in control
main in control
main in control
```

```
main in control
main in control
main in control
```

Output may vary in machine to machine but chances of execution of yield() thread first is higher than the other thread because main thread is always pausing its execution and giving chance to child thread(with same priority).

Note:

- Once a thread has executed yield method and there are many threads with same priority is waiting for processor, then we can't specify which thread will get execution chance first.
- The thread which executes the yield method will enter in the Runnable state from Running state.
- Once a thread pauses its execution, we can't specify when it will get chance again it depends on thread scheduler.
- Underlying platform must provide support for preemptive scheduling if we are using yield method.

2. **sleep():** This method causes the currently executing thread to sleep for the specified number of milliseconds, subject to the precision and accuracy of system timers and schedulers.

Syntax:

```
//  sleep for the specified number of milliseconds
public static void sleep(long millis) throws InterruptedException

//sleep for the specified number of milliseconds plus nano seconds
public static void sleep(long millis, int nanos)
                        throws InterruptedException
```

```
// Java program to illustrate
// sleep() method in Java
import java.lang.*;

public class SleepDemo implements Runnable
{
    Thread t;
    public void run()
    {
        for (int i = 0; i < 4; i++)
        {
            System.out.println(Thread.currentThread().getName()
                                                    + "   " + i);
            try
            {
                // thread to sleep for 1000 milliseconds
                Thread.sleep(1000);
            }

            catch (Exception e)
```

```
                {
                    System.out.println(e);
                }
            }
        }

        public static void main(String[] args) throws Exception
        {
            Thread t = new Thread(new SleepDemo());

            // call run() function
            t.start();

            Thread t2 = new Thread(new SleepDemo());

            // call run() function
            t2.start();
        }
}
```

Run on IDE

Output:

```
Thread-0  0
Thread-1  0
Thread-0  1
Thread-1  1
Thread-0  2
Thread-1  2
Thread-0  3
Thread-1  3
```

Note:

- Based on the requirement we can make a thread to be in sleeping state for a specified period of time
- Sleep() causes the thread to definitely stop executing for a given amount of time; if no other thread or process needs to be run, the CPU will be idle (and probably enter a power saving mode).

yield() vs sleep()

yield:() indicates that the thread is not doing anything particularly important and if any other threads or processes need to be run, they can. Otherwise, the **current thread will continue to run.**

sleep(): causes the thread to definitely stop executing for a given amount of time; if no other thread or process needs to be run, **the CPU will be idle** (and probably enter a power saving mode).

3. **join():** The join() method of a Thread instance is used to join the start of a thread's execution to end of other thread's execution such that a thread does not start running until another thread ends. If join() is called on a Thread instance, the currently running thread will block until

the Thread instance has finished executing.

The join() method waits at most this much milliseconds for this thread to die. A timeout of 0 means to wait forever

Syntax:

```
// waits for this thread to die.
public final void join() throws InterruptedException

// waits at most this much milliseconds for this thread to die
public final void join(long millis)
              throws InterruptedException

// waits at most milliseconds plus nanoseconds for this thread to die.
The java.lang.Thread.join(long millis, int nanos)
```

```
// Java program to illustrate join() method in Java
import java.lang.*;

public class JoinDemo implements Runnable
{
    public void run()
    {
        Thread t = Thread.currentThread();
        System.out.println("Current thread: "
                           + t.getName());

        // checks if current thread is alive
        System.out.println("Is Alive? "
                           + t.isAlive());
    }

    public static void main(String args[]) throws Exception
    {
        Thread t = new Thread(new JoinDemo());
        t.start();

        // Waits for 1000ms this thread to die.
        t.join(1000);

        System.out.println("\nJoining after 1000"+
                           " mili seconds: \n");
        System.out.println("Current thread: " +
                                t.getName());


        // Checks if this thread is alive
        System.out.println("Is alive? " + t.isAlive());
    }
}
```

Run on IDE

Output:

```
Current thread: Thread-0
Is Alive? true
```

```
Joining after 1000 mili seconds:

Current thread: Thread-0
Is alive? false
```

Note:

- If any executing thread t1 calls join() on t2 i.e; t2.join() immediately t1 will enter into waiting state until t2 completes its execution.
- Giving a timeout within join(), will make the join() effect to be nullified after the specific timeout.

References: StackOverflow

This article is contributed by **Dharmesh**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

GATE CS Corner Company Wise Coding Practice

Java Java-Multithreading

Login to Improve this Article

Recommended Posts:

isAlive() and join() methods of Thread Class in Java
Inter-thread Communication in Java
Main thread in Java
Java.lang.Thread class in Java
Multithreading in Java
Joining Threads in Java
BigDecimal Class in Java
Swap corner words and reverse middle characters
String Literal Vs String Object in Java
Download web page using Java