

Basics of Java ▾

OOPS Concepts ▾

String Handling ▾

Exception Handling ▾

Java Multithreading ▾

Introduction to Multithreading

Thread Class

Creating a thread

Joining threads

Sleeping Thread in Java

Naming Thread in Java

Thread Priority in Java

Daemon Thread in Java

Synchronization

Interthread Communication

Thread group

Advanced Topics ▾

Collection Framework ▾

Java GUI ▾

Reflection API ▾

RMI Application ▾

Inner Class ▾

Wrapper Class ▾

File Handling ▾

List ▾

Set ▾

Map ▾

Queue & Deque ▾

JDBC ▾

Layout Managers ▾

ADVERTISEMENT

Joining threads in Java

ADVERTISEMENT

Sometimes one thread needs to know when other thread is terminating. In java, **isAlive()** and **join()** are two different methods that are used to check whether a thread has finished its execution or not.

The **isAlive()** method returns **true** if the thread upon which it is called is still running otherwise it returns **false**.

```
final boolean isAlive()
```

But, **join()** method is used more commonly than **isAlive()**. This method waits until the thread on which it is called terminates.

```
final void join() throws InterruptedException
```

Using **join()** method, we tell our thread to wait until the specified thread completes its execution. There are overloaded versions of **join()** method, which allows us to specify time for which you want to wait for the specified thread to terminate.

ADVERTISEMENT

Java **isAlive** method

Lets take an example and see how the **isAlive()** method works. It returns true if thread status is live, false otherwise.

```
public class MyThread extends Thread
{
    public void run()
    {
        System.out.println("r1 ");
        try {
            Thread.sleep(500);
        }
        catch(InterruptedException ie)
        {
            // do something
        }
        System.out.println("r2 ");
    }
    public static void main(String[] args)
    {
        MyThread t1=new MyThread();
        MyThread t2=new MyThread();
        t1.start();
        t2.start();
    }
}
```

OUTPUT:

```
r1
true
true
r1
r2
r2
```

Example of thread without **join()** method

If we run a thread without using **join()** method then the execution of thread cannot be predict. Thread scheduler schedules the execution of thread.

```
public class MyThread extends Thread
{
    public void run()
    {
        System.out.println("r1 ");
        try {
            Thread.sleep(500);
        }
        catch(InterruptedException ie){ }
        System.out.println("r2 ");
    }
    public static void main(String[] args)
    {
        MyThread t1=new MyThread();
        MyThread t2=new MyThread();
        t1.start();
        t2.start();
    }
}
```

OUTPUT:

```
r1
r1
r2
r2
```

In this above program two thread **t1** and **t2** are created. **t1** starts first and after printing "r1" on console thread **t1** goes to sleep for 500 ms. At the same time Thread **t2** will start its process and print "r1" on console and then go into sleep for 500 ms. Thread **t1** will wake up from sleep and print "r2" on console similarly thread **t2** will wake up from sleep and print "r2" on console. So you will get output like **r1 r1 r2 r2**

Example of thread with **join()** method

In this example, we are using **join()** method to ensure that thread finished its execution before starting other thread. It is helpful when we want to executes multiple threads based on our requirement.

```
public class MyThread extends Thread
{
    public void run()
    {
        System.out.println("r1 ");
        try {
            Thread.sleep(500);
        }
        catch(InterruptedException ie){ }
        System.out.println("r2 ");
    }
    public static void main(String[] args)
    {
        MyThread t1=new MyThread();
        MyThread t2=new MyThread();
        t1.start();

        try{
            t1.join(); //Waiting for t1 to finish
        }catch(InterruptedException ie){}
    }
}
```

OUTPUT:

```
r1
r2
r1
r2
```

In this above program **join()** method on thread **t1** ensures that **t1** finishes its process before thread **t2** starts.

Specifying time with **join()**

If in the above program, we specify time while using **join()** with **t1**, then **t1** will execute for that time, and then **t2** will join it.

```
public class MyThread extends Thread
{
    MyThread(String str){
        super(str);
    }

    public void run()
    {
        System.out.println(Thread.currentThread().getName());
    }
    public static void main(String[] args)
    {
        MyThread t1=new MyThread("first thread");
        MyThread t2=new MyThread("second thread");
        t1.start();

        try{
            t1.join(1500); //Waiting for t1 to finish
        }catch(InterruptedException ie){}
    }
}
```

Doing so, initially **t1** will execute for 1.5 seconds, after which **t2** will join it.

← Prev

Next →

ADVERTISEMENT