Use of join() in thread in java

**When we invoke the join() method on a thread, the calling thread goes into a waiting state until the referenced thread terminates.**

**t.join(); causes the current thread to pause execution until t's thread terminates.**

**wait(): Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object**.

**notify(): Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened**.

**notifyAll():Wakes up all threads that are waiting on this object's monitor. A thread waits on an object's monitor by calling one of the wait methods.**

**The join() method waits for a thread to die. While wait() and notify() are used for inter thread communication**.

**Use Cases for join() method**

**Use Case-1**: **Two threads are running, we want to execute third method after completion of both the threads.**

**public class** TestJoin2 {  
  
 **public void** sleep(**long** time) {  
 **try** {  
 TimeUnit.***SECONDS***.sleep(time);  
 } **catch** (InterruptedException e) {  
 **throw new** RuntimeException(e);  
 }  
 }  
  
 **public void** m1(**long** time) {  
 System.***out***.println(Thread.*currentThread*().getName() + " started ...");  
 sleep(time);  
 System.***out***.println(Thread.*currentThread*().getName() + " completed ...");  
 }  
  
 **public void** m2(**long** time) {  
 System.***out***.println(Thread.*currentThread*().getName() + " started ...");  
 sleep(time);  
 System.***out***.println(Thread.*currentThread*().getName() + " completed ...");  
 }  
  
 **public void** check() {  
 Instant start = Instant.*now*();  
 Runnable r1 = () -> m1(7);  
 Runnable r2 = () -> m2(5);  
 Thread t1 = **new** Thread(r1);  
 Thread t2 = **new** Thread(r2);  
 t1.start();  
 t2.start();  
  
 **try** {  
 t1.join();  
 t2.join();  
 } **catch** (InterruptedException e) {  
 **throw new** RuntimeException(e);  
 }

// Third operation below  
 Instant end = Instant.*now*();  
 **long** duration = Duration.*between*(start,end).getSeconds();  
 System.***out***.println("Duration : "+duration);  
 System.***out***.println("All OPS completed ...");  
 }  
  
 **public static void** main(String[] args) {  
  
 **new** TestJoin2().check();  
  
 }  
}

**OUTPUT**

Thread-0 started ...

Thread-1 started ...

Thread-1 completed ...

Thread-0 completed ...

Duration : 7

All OPS completed ...

**Use Case-2**: **Two threads are running, we want to execute third method whether two threads complete or not.**

**public class** TestJoin2 {  
  
 **public void** sleep(**long** time) {  
 **try** {  
 TimeUnit.***SECONDS***.sleep(time);  
 } **catch** (InterruptedException e) {  
 **throw new** RuntimeException(e);  
 }  
 }  
  
 **public void** m1(**long** time) {  
 System.***out***.println(Thread.*currentThread*().getName() + " started ...");  
 sleep(time);  
 System.***out***.println(Thread.*currentThread*().getName() + " completed ...");  
 }  
  
 **public void** m2(**long** time) {  
 System.***out***.println(Thread.*currentThread*().getName() + " started ...");  
 sleep(time);  
 System.***out***.println(Thread.*currentThread*().getName() + " completed ...");  
 }  
  
 **public void** check() {  
 Instant start = Instant.*now*();  
 Runnable r1 = () -> m1(7);  
 Runnable r2 = () -> m2(9);  
 Thread t1 = **new** Thread(r1);  
 Thread t2 = **new** Thread(r2);  
 t1.start();  
 t2.start();  
  
 **try** {  
 t1.join(2000);  
 t2.join(2000);  
 } **catch** (InterruptedException e) {  
 **throw new** RuntimeException(e);  
 }  
  
 // Third operation below  
 Instant end = Instant.*now*();  
 **long** duration = Duration.*between*(start,end).getSeconds();  
 System.***out***.println("Duration : "+duration);  
 System.***out***.println("All OPS completed ...");  
 }  
  
 **public static void** main(String[] args) {  
  
 **new** TestJoin2().check();  
  
 }  
}

**Use Case-3: Use join()method so that two threads will run alternately.**

**public class** CoinTossGame {  
 **public synchronized void** toss(String name) {  
 **try** {  
 System.***out***.println("---------------------START--------------------");  
 System.***out***.println(name + " is playing the game");  
 Thread.*sleep*(2000);  
 System.***out***.println(name + " completed the game...");  
 System.***out***.println("---------------------OVER---------------------**\n**");  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
}

**public class** TestJoin3 {  
  
 **public void** sleep(**long** time) {  
 **try** {  
 TimeUnit.***SECONDS***.sleep(time);  
 } **catch** (InterruptedException e) {  
 **throw new** RuntimeException(e);  
 }  
 }  
  
 **public void** m1(CoinTossGame game) {  
 **while** (**true**) {  
 game.toss("Player 1");  
 **try** {  
 Thread.*currentThread*().join(1000);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 }  
  
 **public void** m2(CoinTossGame game) {  
 **while** (**true**) {  
 game.toss("Player 2");  
 **try** {  
 Thread.*currentThread*().join(1000);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }

**public void** check() {  
 CoinTossGame game = **new** CoinTossGame();  
 Runnable r1 = () -> m1(game);  
 Runnable r2 = () -> m2(game);  
  
 **new** Thread(r1).start();  
 **new** Thread(r2).start();  
 }  
  
 **public static void** main(String[] args) {  
 **new** TestJoin3().check();  
 }  
}

**OUTPUT**

---------------------START--------------------

Player 1 is playing the game

Player 1 completed the game...

---------------------OVER---------------------

---------------------START--------------------

Player 2 is playing the game

Player 2 completed the game...

---------------------OVER---------------------

---------------------START--------------------

Player 1 is playing the game

Player 1 completed the game...

---------------------OVER---------------------

Question: **Suppose there are two threads, both are running, I want to execute a method after the completion of two threads. How will you do it ?**

There is a another way to perform third task after the completion of two threads.  
**public class** TestTaskThread3 {  
 **public static void** main(String[] args) **throws** Exception {  
 Thread th1 = **new** Thread(**new** TaskThread1(**"Alice"**, 1));  
 Thread th2 = **new** Thread(**new** TaskThread1(**"Bob"**, 2));  
 th1.start();  
 th2.start();  
 **boolean flag = true;  
 while (flag) {  
 if (!th1.isAlive() && !th2.isAlive()) {  
 flag = false;  
 System.*out*.println("All operations completed ...");  
 }  
 }** }  
}

Or

Thread th1 = **new** Thread(**new** TaskThread1(**"Alice"**, 1));  
Thread th2 = **new** Thread(**new** TaskThread1(**"Bob"**, 2));  
th1.start();  
th2.start();  
t1.join();  
t2.join();

**System.*out*.println("All operations completed ...");**

OR

**boolean** flag = **true**;  
**while** (flag) {  
 **if** (t1.isAlive() && t2.isAlive()) {  
 flag = **true**;  
 } **else** flag = **false**;  
}  
System.***out***.println(**"All operations completed ..."**);

**Use of Thread.yield**

**This static method is essentially used to notify the system that the current thread is willing to "give up the CPU" for a while.** We should try to avoid using Thread.yield() method.

**What is the difference between sleep() and yield()**

**Sleep is used to delay the execution for a period of** time, and sleeping thread is guaranteed for sleep at least the specified time. But it not guaranteed that newly weak up thread will actually returning to running.

Yield is used to get the running thread into out of runnable state with the same priority.

**isAlive()**

**The isAlive method is generally used to test if a thread has started.** Right after creating a Thread object, the thread object is in the "**New Thread**" state.

**DEFINITION OF THE isAlive() METHOD:**

The isAlive method returns true if the thread has been started and not stopped.

**There is a difference between join() and sleep()**

**join( )** method of java waits till the thread on which it is called terminates. The isAlive( ) method of java returns true only if the thread is still running. **join()** will wait until the timeout expires or the thread finishes. **sleep()** will just wait for the specified amount of time unless interrupted. So it is perfectly possible for join() to return much faster than the specified time. **join()** waits for something meaningful while **sleep()** just sits there doing nothing.

**\*\* Situation \*\***

Two threads, one by one thread, it means one operation will be performed by only one thread. Once that thread completes that operation, another thread has to start the operation. In this case, you have to use "Thread.join()" method. But there is a trick in this case, see below.

**th1.start();**

**incorrect Implementation 🡺 th2.start();**

**th1.join();**

**th2.join();**

th1.start();

th1.join(); 🡸 **correct implementation**

th2.start();

th2.join();