

Stack Overflow is a question and answer site for professional and enthusiast programmers. It's 100% free, no registration required.

Take the 2-minute tour x

## Difference between wait() and sleep()

What is the difference between a `wait()` and `sleep()` in Threads?

Is my understanding that a `wait()` -ing Thread is still in running mode and uses CPU cycles but a `sleep()` -ing does not consume any CPU cycles correct?

Why do we have *both* `wait()` and `sleep()` : how does their implementation vary at a lower level?

java multithreading sleep wait

edited Jun 15 '13 at 10:35



Raedwald

10.8k

7

41

77

asked Jun 24 '09 at 6:48



Geek

4,347

10

38

57

10 very good question. semantics of both are easy to confuse. – [Andreas Petersson](#) Jun 24 '09 at 7:25

Very nice questions but they are 2 in one. Why do we have both is not the same as how they can (and not are!) implemented at a lower level. I've answered to that too. – [estani](#) Apr 19 '12 at 10:39

Suppose a thread A is in a synchronized block, and while it is in the cpu from this thread is taken and given to another thread B. now in which state Thread A will go, will the other threads waiting on this synchronized block come inside now? – [Peter](#) Aug 17 '13 at 8:22

Here's a good article describing it: [qat.com/using-waitnotify-instead-thread-sleep-java](http://qat.com/using-waitnotify-instead-thread-sleep-java) – [Triton Man](#) Sep 27 '13 at 13:59

its EXCATLY the opposite - `sleep` "uses" all of its available CPU-cycles but since the thread will be in "WAITING"-state these can be yielded if necessary - in fact most operating systems automatically yield the cycles *IF* it is possible, hence your thread will not create any actual CPU-load ... it will do so on older operating systems, though. `Object.wait()`, on the other hand *NEVER* uses any cycles (while being unnotified) because thats realized via software-interrupts in many cases - private, transient & transparent locks, implemented by the JVM. `Thread.sleep` is bad practise. – [specializt](#) Dec 1 '14 at 11:16

## 27 Answers

A `wait` can be "woken up" by another process calling `notify` on the monitor which is being waited on whereas a `sleep` cannot. Also a `wait` (and `notify`) must happen in a block synchronized on the monitor object whereas `sleep` does not:

```
Object mon = ...;
synchronized (mon) {
    mon.wait();
}
```

At this point the currently executing thread waits *and releases the monitor*. Another thread may do

```
synchronized (mon) { mon.notify(); }
```

(On the same `mon` object) and the first thread (assuming it is the only thread waiting on the monitor) will wake up.


You can also call `notifyAll` if more than one thread is waiting on the monitor - this will wake *all of them up*. However, only one of the threads will be able to grab the monitor (remember that the `wait` is in a `synchronized` block) and carry on - the others will then be blocked until they can acquire the monitor's lock.

Another point is that you call `wait` on `Object` itself (i.e. you wait on an object's monitor) whereas you call `sleep` on `Thread`.

Yet another point is that you can get *spurious wakeups* from `wait` (i.e. the thread which is waiting resumes for no apparent reason). You should **always** `wait` **whilst spinning on some condition** as follows:

```
synchronized {
    while (!condition) { mon.wait(); }
}
```

edited Jun 16 '13 at 23:02

 Sk8erPeter  
2,570 3 27 41

answered Jun 24 '09 at 6:50

 oxbow\_lakes  
74.5k 29 216 365

- 
- 8 A sleeping Thread can also be woken up by notify(). ? – [Geek](#) Jun 24 '09 at 6:51
- 
- 58 No, it cannot. It can only be interrupted. – [Peter Štibrány](#) Jun 24 '09 at 6:53
- 
- 7 No - a sleeping thread cannot be woken by notify – [oxbow\\_lakes](#) Jun 24 '09 at 6:54
- 
- 11 @Geek - why in the world do you say wait() wastes CPU cycles? – [Robert Munteanu](#) Jun 24 '09 at 7:04
- 
- 6 Interruption is intended as a mechanism to gently encourage a thread to stop running entirely and cancel remaining operations. wait / notify are typically used to wait for some other thread to accomplish a task, or to wait until a certain condition is satisfied. – [Louis Wasserman](#) Jul 26 '12 at 11:40
- 

One key difference not yet mentioned is that while sleeping a Thread does *not* release the locks it holds, while waiting releases the lock on the object that wait() is called on.

```
synchronized(LOCK) {
    Thread.sleep(1000); // LOCK is held
}
```

```
synchronized(LOCK) {
    LOCK.wait(); // LOCK is not held
}
```

edited Jun 24 '09 at 7:24

answered Jun 24 '09 at 7:06

Robert Munteanu  
34k 21 123 215

- 
- 45 Waiting only releases the lock for the object you call wait() on. It doesn't release any *other* locks. – [Jon Skeet](#) Jun 24 '09 at 7:18
- 
- @Jon Skeet: thanks, corrected. – [Robert Munteanu](#) Jun 24 '09 at 7:20
- 
- 7 You don't actually need to call sleep from within a lock - locks and wait/notify go hand in hand but locks and sleep are unrelated. – [oxbow\\_lakes](#) Jun 24 '09 at 7:21
- 
- 4 @oxbow\_lakes - I'd say that you should not sleep with locks, there are few uses cases for that. Just wanted to point out the differences. – [Robert Munteanu](#) Jun 24 '09 at 7:23
- 
- 1 @RobertMunteanu, Your answer misleadingly claims that sleep holds java locks, but it doesn't. To have a fair comparison, we would compare synchronized(OUTER\_LOCK){ Thread.sleep(1000); } with synchronized(OUTER\_LOCK){ synchronized(LOCK){LOCK.wait();} } and we can see that both instructions don't release the OUTER\_LOCK . If there's any difference, we can say that sleep doesn't explicitly **use** java locks, but the question is asking about quote "how does their implementation vary at a lower level?" unquote. – [Pacerier](#) Aug 11 '14 at 7:56
- 

I found [this link](#) helpful (which references [this post](#)). It puts the difference between sleep() , wait() , and yield() in human terms. (in case the links ever go dead I've included the post below with additional markup)

It all eventually makes its way down to the OS's scheduler, which hands out timeslices to processes and threads.

sleep(n) says **"I'm done with my timeslice, and please don't give me another one for at least n milliseconds."** The OS doesn't even try to schedule the sleeping thread until requested time has passed.

yield() says **"I'm done with my timeslice, but I still have work to do."** The OS is free to immediately give the thread another timeslice, or to give some other thread or process the CPU the yielding thread just gave up.

.wait() says **"I'm done with my timeslice. Don't give me another timeslice until someone calls notify()"**. As with sleep() , the OS won't even try to schedule your task

unless someone calls `notify()` (or one of a few other wakeup scenarios occurs).

Threads also lose the remainder of their timeslice when they perform blocking IO and under a few other circumstances. If a thread works through the entire timeslice, the OS forcibly takes control roughly as if `yield()` had been called, so that other processes can run.

You rarely need `yield()`, but if you have a compute-heavy app with logical task boundaries, inserting a `yield()` *might* improve system responsiveness (at the expense of time — context switches, even just to the OS and back, aren't free). Measure and test against goals you care about, as always.

answered Aug 5 '11 at 19:32

 **E-rich**  
2,278 4 23 53

Yield is basically platform-dependent... [javamex.com/tutorials/threads/yield.shtml](http://javamex.com/tutorials/threads/yield.shtml) — [Pacerier](#) Aug 10 '14 at 5:04

There are a lot of answers here but I couldn't find the semantic distinction mentioned on any.

It's not about the thread itself; both methods are required as they support very different use-cases.

`sleep()` sends the Thread to sleep as it was before, it just packs the context and stops executing for a predefined time. So in order to wake it up before the due time, you need to know the Thread reference. This is not a common situation in a multi-threaded environment. It's mostly used for time-synchronization (e.g. wake in exactly 3.5 seconds) and/or hard-coded fairness (just sleep for a while and let others threads work).


`wait()`, on the contrary, is a thread (or message) synchronization mechanism that allows you to notify a Thread of which you have no stored reference (nor care). You can think of it as a publish-subscribe pattern (`wait` == subscribe and `notify()` == publish). Basically using `notify()` you are sending a message (that might even not be received at all and normally you don't care).

To sum up, you normally use `sleep()` for time-synchronization and `wait()` for multi-thread-synchronization.

They could be implemented in the same manner in the underlying OS, or not at all (as previous versions of Java had no real multithreading; probably some small VMs doesn't do that either). Don't forget Java runs on a VM, so your code will be transformed in something different according to the VM/OS/HW it runs on.

edited Jul 21 '14 at 8:22

answered Apr 19 '12 at 10:38

 **estani**  
1,596 10 14

There are some difference key notes i conclude after working on wait and sleep, first take a look on sample using wait() and sleep():

**Example1:** using `wait()` and `sleep()`:

```
synchronized(HandObject) {
    while(isHandFree() == false) {
        /* Hand is still busy on happy coding or something else, please wait */
        HandObject.wait();
    }
}

/* Get lock ^^, It is my turn, take a cup beer now */
while (beerIsAvailable() == false) {
    /* Beer is still coming, not available, Hand still hold glass to get beer,
    don't release hand to perform other task */
    Thread.sleep(5000);
}

/* Enjoy my beer now ^^ */
drinkBeers();

/* I have drink enough, now hand can continue with other task: continue coding */
setHandFreeState(true);
```

```
synchronized(HandObject) {  
    HandObject.notifyAll();  
}
```

Let clarify some key notes:

1. **Call on:**

- wait(): Call on current thread that hold HandObject Object
- sleep(): Call on Thread execute task get beer (is class method so affect on current running thread)

2. **Synchronized:**

- wait(): when synchronized multi thread access same Object (HandObject) (When need communication between more than one thread (thread execute coding, thread execute get beer) access on same object HandObject )
- sleep(): when waiting condition to continue execute (Waiting beer available)

3. **Hold lock:**

- wait(): release the lock for other object have chance to execute (HandObject is free, you can do other job)
- sleep(): keep lock for at least t times (or until interrupt) (My job still not finish, i'm continue hold lock and waiting some condition to continue)

4. **Wake-up condition:**

- wait(): until call notify(), notifyAll() from object
- sleep(): until at least time expire or call interrupt

5. And the last point is **use when** as [estani](#) indicate:

you normally use sleep() for time-synchronization and wait() for multi-thread-synchronization.

Please correct me if i'm wrong.

edited Jul 16 '14 at 7:41



NickyNick  
19 5

answered May 19 '12 at 6:59



NguyenDat  
2,147 1 24 35

This is a very simple question, because both these methods have a totally different use.

**The major difference is to wait to release the lock or monitor while sleep doesn't release any lock or monitor while waiting. Wait is used for inter-thread communication while sleep is used to introduce pause on execution.**

This was just a clear and basic explanation, if you want more than that then continue reading.

In case of `wait()` method thread goes in waiting state and it won't come back automatically until we call the `notify()` method (or `notifyAll()` if you have more than one thread in waiting state and you want to wake all of those threads). And you need synchronized or object lock or class lock to access the `wait()` or `notify()` or `notifyAll()` methods. And one more thing, the `wait()` method is used for inter-thread communication because if a thread goes in waiting state you'll need another thread to wake that thread.

But in case of `sleep()` this is a method which is used to hold the process for few seconds or the time you wanted. Because you don't need to provoke any `notify()` or `notifyAll()` method to get that thread back. Or you don't need any other thread to call back that thread. Like if you want something should happen after few seconds like in a game after user's turn you want the user to wait until the computer plays then you can mention the `sleep()` method.

And one more important difference which is asked often in interviews: `sleep()` belongs to `Thread` class and `wait()` belongs to `Object` class.

These are all the differences between `sleep()` and `wait()`.

And there is a similarity between both methods: they both are checked statement so you need try catch or throws to access these methods.

I hope this will help you.

edited Apr 20 '12 at 0:00



Charles Menguy

14.3k 10 39 74

answered Apr 19 '12 at 7:02



Vikas Gupta

965 2 13 25

`sleep` is a method of `Thread`, `wait` is a method of `Object`, so `wait/notify` is a technique of synchronizing shared data in Java (using [monitor](#)), but `sleep` is a simple method of thread to pause itself.

edited Nov 1 '12 at 12:33

answered Oct 30 '12 at 22:46



barn.gumbl

1,155 18 36

source : <http://www.jguru.com/faq/view.jsp?EID=47127>

`Thread.sleep()` sends the current thread into the "Not Runnable" state for some amount of time. The thread keeps the monitors it has acquired -- i.e. if the thread is currently in a synchronized block or method no other thread can enter this block or method. If another thread calls `t.interrupt()` it will wake up the sleeping thread.

Note that `sleep` is a static method, which means that it always affects the current thread (the one that is executing the `sleep` method). A common mistake is to call `t.sleep()` where `t` is a different thread; even then, it is the current thread that will sleep, not the `t` thread.

`t.suspend()` is deprecated. Using it is possible to halt a thread other than the current thread. A suspended thread keeps all its monitors and since this state is not interruptable it is deadlock prone.

`object.wait()` sends the current thread into the "Not Runnable" state, like `sleep()`, but with a twist. `Wait` is called on an object, not a thread; we call this object the "lock object." Before `lock.wait()` is called, the current thread must synchronize on the lock object; `wait()` then releases this lock, and adds the thread to the "wait list" associated with the lock. Later, another thread can synchronize on the same lock object and call `lock.notify()`. This wakes up the original, waiting thread. Basically, `wait()` / `notify()` is like `sleep()` / `interrupt()`, only the active thread does not need a direct pointer to the sleeping thread, but only to the shared lock object.

edited Jun 17 '13 at 10:45



Sk8erPeter

2,570 3 27 41

answered Nov 5 '11 at 3:34



om singh

81 1 1

- 2 you should have mentioned that you copied and pasted this explanation from an external site. Your post has been edited, but you should mark the source if you copy-paste in the future, as these are NOT your own words (so this is really unethical). – Sk8erPeter Jun 17 '13 at 10:45

Wait and sleep are two different things:

- In `sleep()` the thread stops working for the specified duration.
- In `wait()` the thread stops working until the object being waited-on is notified, generally by other threads.

edited May 24 '12 at 10:19



Zaki

3,624 3 17 32

answered Jun 24 '09 at 6:53



Itay Maman

16.4k 4 40 80

but you can interrupt a sleeping Thread. In that case `wait()` is redundant infact it wastes CPU cycles too :-(  
– Geek Jun 24 '09 at 6:54

- 6 Wait doesn't waste CPU cycles. – Peter Štibraný Jun 24 '09 at 6:55
- 1 @Peter - I think it does. It waits() for its chunk of CPU cycles and then the OS gives the CPU cycles to other Threads. I think this might be OS dependant, I am not sure. – Geek Jun 24 '09 at 6:58
- 2 It would be very poor implementation of `wait()` if it wasted CPU cycles. `wait/notify` is used quite a lot for interthread communication. – Peter Štibraný Jun 24 '09 at 7:00
- 1 Mate, you were right :-)) – Geek Jun 24 '09 at 7:02

**sleep()** is a method which is used to hold the process for few seconds or the time you wanted but in case of `wait()` method thread goes in waiting state and it won't come back automatically until we call the `notify()` or `notifyAll()`.

The **major difference** is that **wait()** releases the lock or monitor while `sleep()` doesn't releases any lock or monitor while waiting. Wait is used for inter-thread communication while sleep is used to introduce pause on execution, generally.

**Thread.sleep()** sends the current thread into the "Not Runnable" state for some amount of time. The thread keeps the monitors it has acquired — i.e. if the thread is currently in a synchronized block or method no other thread can enter this block or method. If another thread calls `t.interrupt()` it will wake up the sleeping thread. Note that `sleep` is a static method, which means that it always affects the current thread (the one that is executing the `sleep` method). A common mistake is to call `t.sleep()` where `t` is a different thread; even then, it is the current thread that will sleep, not the `t` thread.

**object.wait()** sends the current thread into the "Not Runnable" state, like `sleep()`, but with a twist. Wait is called on an object, not a thread; we call this object the "lock object." Before `lock.wait()` is called, the current thread must synchronize on the lock object; `wait()` then releases this lock, and adds the thread to the "wait list" associated with the lock. Later, another thread can synchronize on the same lock object and call `lock.notify()`. This wakes up the original, waiting thread. Basically, `wait()/notify()` is like `sleep()/interrupt()`, only the active thread does not need a direct pointer to the sleeping thread, but only to the shared lock object.

```
synchronized(LOCK) {
    Thread.sleep(1000); // LOCK is held
}

synchronized(LOCK) {
    LOCK.wait(); // LOCK is not held
}
```

Let categorize all above points :

Call on:

- **wait()**: Call on an object; current thread must synchronize on the lock object.
- **sleep()**: Call on a Thread; always currently executing thread.

Synchronized:

- **wait()**: when synchronized multiple threads access same Object one by one.
- **sleep()**: when synchronized multiple threads wait for sleep over of sleeping thread.

Hold lock:

- **wait()**: release the lock for other objects to have chance to execute.
- **sleep()**: keep lock for at least t times if timeout specified or somebody interrupt.

Wake-up condition:

- **wait()**: until call notify(), notifyAll() from object
- **sleep()**: until at least time expire or call interrupt().

Usage:

- **sleep()**: for time-synchronization and;
- **wait()**: for multi-thread-synchronization.

Ref:[diff sleep and wait](#)

answered Dec 23 '13 at 6:07



[Reegan Miranda](#)

739 2 9 21

In simple words, wait is wait Until some other thread invokes you whereas sleep is "dont execute next statement" for some specified period of time.

Moreover sleep is static method in Thread class and it operates on thread, whereas wait() is in Object class and called on an object.

Another point, when you call wait on some object, the thread involved synchronize the object and then waits. :)

edited Jun 24 '09 at 6:59

answered Jun 24 '09 at 6:51



[Ratnesh Maurya](#)

571 2 9

1 Why do you need both ? Why sleep() is not sufficient ? – [Geek](#) Jun 24 '09 at 6:53

Notify is used for communication between threads. To call wait, you need some object, synchronize on it, and then call wait on it. To be notified, you need other thread to synchronize on the *same* object, and call notify. – [Peter Štibrný](#) Jun 24 '09 at 6:55

The methods are used for different things.

```
Thread.sleep(5000); // Wait until the time has passed.
```

```
Object.wait(); // Wait until some other thread tells me to wake up.
```

Thread.sleep(n) *can* be interrupted, but Object.wait() *must* be notified. It's possible to specify the maximum time to wait: object.wait(5000) so it would be possible to use wait to, er, sleep but then you have to bother with locks.

Neither of the methods uses the cpu while sleeping/waiting.

The methods are implemented using native code, using similar constructs but not in the same way.

Look for yourself: [Is the source code of native methods available?](#) The file /src/share/vm/prims/jvm.cpp is the starting point...

answered May 11 '13 at 6:15



[KarlP](#)

3,379 12 34

Thread.sleep timing can also be set to indefinite. Object.wait timing can also be set to definite. This answer doesn't explain why we need 2 hammers that do the same thing. – [Pacerier](#) Aug 10 '14 at 5:09

- The method `wait(1000)` causes the current thread to sleep **up to one second**.
  - A thread could sleep less than 1 second if it receives the `notify()` or `notifyAll()` method call.
- The call to `sleep(1000)` causes the current thread to sleep for **exactly 1 second**.
  - Also **sleeping thread doesn't hold lock any resource**. But waiting thread does.

edited Jun 17 '13 at 8:33

answered May 17 '13 at 19:55

 [Sk8erPeter](#)  
2,570 3 27 41

 [Rupesh](#)  
678 3 14

1 `sleep(1000)` doesn't guaranty to sleep for exactly 1 second. It may may be interrupted before. – [Lucio](#) Jun 19 '14 at 14:59


`wait` and `sleep` methods are very different:

- `sleep` has no way of "waking-up",
- whereas `wait` has a way of "waking-up" during the wait period, by another thread calling `notify` or `notifyAll`.

Come to think about it, the names are confusing in that respect; however `sleep` is a standard name and `wait` is like the `WaitForSingleObject` or `WaitForMultipleObjects` in the Win API.

edited Jun 17 '13 at 8:52

answered Jun 24 '09 at 6:52

 [Sk8erPeter](#)  
2,570 3 27 41

 [Roe Adler](#)  
10.7k 15 67 112

1 But we can interrupt a `sleep` couldn't we? so what's the difference with that sleep/interrupt vs wait/notify? – [Pacerier](#) Feb 3 '12 at 9:01


One potential big difference between sleep/interrupt and wait/notify is that


- calling `interrupt()` during `sleep()` always throws an exception (e.g. `InterruptedException`), whereas
- calling `notify()` during `wait()` does not.

Generating an exception when not needed is inefficient. If you have threads communicating with each other at a high rate, then it would be generating a lot of exceptions if you were calling interrupt all the time, which is a total waste of CPU.

edited Jun 17 '13 at 8:53

answered May 18 '13 at 22:26

 [Sk8erPeter](#)  
2,570 3 27 41

 [Mark](#)  
11 1

+1, A valid point actually, though arguing on the *internals* of implementations may be more relevant to performance analysis... – [Pacerier](#) Aug 10 '14 at 5:16

In my opinion, the main difference between both mechanisms is that sleep/interrupt is the most basic way of handling threads, whereas **wait/notify is an abstraction aimed to do thread inter-communication easier**. This means that sleep/interrupt can do anything, but that this specific task is harder to do.

Why is wait/notify more suitable? Here are some personal considerations:


1. **It enforces centralization.** It allows to coordinate the communication between a group of threads with a single shared object. This simplifies the work a lot.
2. **It enforces synchronization.** Because it makes the programmer wrap the call to wait/notify in a synchronized block.



3. **It's independent of the thread origin and number.** With this approach you can add more threads arbitrarily without editing the other threads or keeping a track of the existing ones. If you used sleep/interrupt, first you would need to keep the references to the sleeping threads, and then interrupt them one by one, by hand.

An example from the real life that is good to explain this is a classic restaurant and the method that the personnel use to communicate among them: The waiters leave the customer requests in a central place (a cork board, a table, etc.), ring a bell, and the workers from the kitchen come to take such requests. Once that there is any course ready, the kitchen personnel ring the bell again so that the waiters are aware and take them to the customers.

answered Jan 30 '14 at 17:15

 [negora](#)  
81 1 2

Wait() and sleep() Differences?

Thread.sleep() Once its work completed then only its release the lock to everyone. until its never release the lock to anyone.

**Sleep()** take the key, its never release the key to anyone, when its work completed then only its release then only take the key waiting stage threads.

Object.wait() When its going to waiting stage, its will be release the key and its waiting for some of the seconds based on the parameter.

For Example:

you are take the coffee in yours right hand, you can take another anyone of the same hand, when will your put down then only take another object same type here. also. this is sleep() you sleep time you didn't any work, you are doing only sleeping.. same here also.

wait(). when you are put down and take another one mean while you are waiting , that's wait

you are play movie or anything in yours system same as player you can't play more than one at a time right, thats its here, when you close and choose another anyone movie or song mean while is called wait

edited May 6 '14 at 13:05

answered May 6 '14 at 12:51

 [VISALIG](#)  
11 3


1) wait() is a method of Object class.  
sleep() is a method of Thread class.


2) sleep() allows the thread to go to sleep state for x milliseconds.  
When a thread goes into sleep state it doesn't release the lock.

wait() allows thread to release the lock and goes to suspended state.  
The thread is only active when a notify() or notifyAll() method is called for the same object.

edited Nov 7 '14 at 7:51

answered Sep 9 '14 at 10:52

 [user2195963](#)  
16 5

 [OpenJDK](#)  
398 1 13

You are correct - Sleep() causes that thread to "sleep" and the CPU will go off and process other threads (otherwise known as context switching) whereas I believe Wait keeps the CPU processing the current thread.

We have both because although it may seem sensible to let other people use the CPU while you're not using it, actually there is an overhead to context switching - depending on how long the sleep is for, it can be more expensive in CPU cycles to switch threads than it is to simply have your thread doing nothing for a few ms.

Also note that sleep forces a context switch.

Also - in general it's not possible to control context switching - during the Wait the OS may (and will for longer waits) choose to process other threads.

edited Jun 24 '09 at 7:21



oxbow\_lakes

74.5k 29 216 365

answered Jun 24 '09 at 7:02



Justin

44.8k 20 118 231

- 2 wait() doesn't keep the CPU processing the current thread. It is like sleep in that it causes a context switch as well: [javamex.com/tutorials/threads/context\\_switch.shtml](http://javamex.com/tutorials/threads/context_switch.shtml). I've been asking for half a year all around stackoverflow and it seems like no one knows what's the difference between wait/notify vs sleep/interrupt. – Pacerier Jul 20 '12 at 10:15

Lets assume you are hearing songs.

As long as the current song is running, the next song wont play, i.e Sleep() called by next song

If you finish the song it will stop and until you select play button(notify()) it wont play, i.e wait() called by current song.

In this both cases songs going to Wait states.

edited Mar 21 '12 at 7:27



oers

10.5k 8 30 50

answered Mar 20 '12 at 12:29



pavan

29 2

wait() is given inside a synchronized method whereas sleep() is given inside a non-synchronized method because wait() method release the lock on the object but sleep() or yield() does release the lock() .

edited May 13 '13 at 14:48



alecxe

82.4k 15 64 137

answered May 11 '13 at 5:26



Aravind Mano

1 1

sleep() method causes the current thread to move from running state to block state for a specified time. If the current thread has the lock of any object then it keeps holding it, which means that other threads cannot execute any synchronized method in that class object.

wait() method causes the current thread to go into block state either for a specified time or until notify, but in this case the thread releases the lock of the object (which means that other threads can execute any synchronized methods of the calling object).

edited Jul 14 '13 at 20:35



dkar

636 4 16

answered Jul 14 '13 at 20:01



User10001

331 4 17

Actually, all this is clearly described in Java docs (but I realized this only after reading the answers).

<http://docs.oracle.com/javase/7/docs/api/index.html> :

wait() - The current thread must own this object's monitor. The thread releases ownership of this monitor and waits until another thread notifies threads waiting on this object's monitor to wake up either through a call to the notify method or the notifyAll method. The thread then waits until it can re-obtain ownership of the monitor and resumes execution.

sleep() - Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds, subject to the precision and accuracy of system timers and schedulers. The thread does not lose ownership of any monitors.

answered Aug 7 '13 at 22:34



TT\_

379 4 14

Here wait() will be in the waiting state till it notify by another Thread but where as sleep() will be having some time..after that it will automatically transfer to the Ready state...

answered Sep 23 '13 at 5:24



wait with a timeout value can wakeup upon timeout value elapsed or notify whichever is earlier (or interrupt as well), whereas, a sleep wakes up on timeout value elapsed or interrupt whichever is earlier. wait() with no timeout value will wait for ever until notified or interrupted

answered Sep 9 '14 at 11:33



wait releases the lock and sleep doesn't. A thread in waiting state is eligible for waking up as soon as notify or notifyAll is called. But in case of sleep the thread keeps the lock and it'll only be eligible once the sleep time is over.

answered Nov 7 '14 at 8:53



So if the thread is sleeping for 10 seconds and an interrupted exception happens ???? – [Geek](#) Nov 11 '14 at 23:20

Example about sleep doesn't release lock and wait does

Here there are two classes :

1. **Main** : Contains main method and two threads.
2. **Singleton** : This is singleton class with two static methods getInstance() and getInstance(boolean isWait).

```
public class Main {  
  
    private static Singleton singletonA = null;  
    private static Singleton singletonB = null;  
  
    public static void main(String[] args) throws InterruptedException {  
  
        Thread threadA = new Thread() {  
            @Override  
            public void run() {  
  
                singletonA = Singleton.getInstance(true);  
  
            }  
        };  
  
        Thread threadB = new Thread() {  
            @Override  
            public void run() {  
                singletonB = Singleton.getInstance();  
  
                while (singletonA == null) {  
                    System.out.println("SingletonA still null");  
                }  
  
                if (singletonA == singletonB) {  
                    System.out.println("Both singleton are same");  
                } else {  
                    System.out.println("Both singleton are not same");  
                }  
  
            }  
        };  
  
        threadA.start();  
        threadB.start();  
  
    }  
}
```

and

```

public class Singleton {

    private static Singleton _instance;

    public static Singleton getInstance() {

        if (_instance == null) {
            synchronized (Singleton.class) {
                if (_instance == null)
                    _instance = new Singleton();
            }
        }
        return _instance;
    }

    public static Singleton getInstance(boolean isWait) {

        if (_instance == null) {
            synchronized (Singleton.class) {
                if (_instance == null) {
                    if (isWait) {
                        try {
                            // Singleton.class.wait(500); // Using wait
                            Thread.sleep(500); // Using Sleep
                            System.out.println("_instance : "
                                + String.valueOf(_instance));
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                    _instance = new Singleton();
                }
            }
        }
        return _instance;
    }
}

```

Now run this example you will get below output :

```

_instance : null
Both singleton are same

```

Here Singleton instances created by threadA and threadB are same. It means threadB is waiting outside until threadA release it's lock.

Now change the Singleton.java by commenting Thread.sleep(500); method and uncommenting Singleton.class.wait(500); . Here because of Singleton.class.wait(500); method threadA will release all acquire locks and moves into the "Non Runnable" state, threadB will get change to enter in synchronized block.

Now run again :

```

SingletonA still null
SingletonA still null
SingletonA still null
_instance : com.omt.sleepwait.Singleton@10c042ab
SingletonA still null
SingletonA still null
SingletonA still null
Both singleton are not same

```

Here Singleton instances created by threadA and threadB are NOT same because of threadB got change to enter in synchronised block and after 500 milliseconds threadA started from it's last position and created one more Singleton object.

answered Dec 1 '14 at 11:06



Dhiral Pandya

2,409 1 19 22