| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Timer.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/util/StringTokenizer.html)   [**NEXT CLASS**](http://docs.google.com/java/util/TimerTask.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/Timer.html)    [**NO FRAMES**](http://docs.google.com/Timer.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | [CONSTR](#3znysh7) | [METHOD](#2et92p0) | DETAIL: FIELD | [CONSTR](#3dy6vkm) | [METHOD](#3rdcrjn) |

## **java.util**

Class Timer

[java.lang.Object](http://docs.google.com/java/lang/Object.html)  
 **java.util.Timer**

public class **Timer**extends [Object](http://docs.google.com/java/lang/Object.html)

A facility for threads to schedule tasks for future execution in a background thread. Tasks may be scheduled for one-time execution, or for repeated execution at regular intervals.

Corresponding to each Timer object is a single background thread that is used to execute all of the timer's tasks, sequentially. Timer tasks should complete quickly. If a timer task takes excessive time to complete, it "hogs" the timer's task execution thread. This can, in turn, delay the execution of subsequent tasks, which may "bunch up" and execute in rapid succession when (and if) the offending task finally completes.

After the last live reference to a Timer object goes away *and* all outstanding tasks have completed execution, the timer's task execution thread terminates gracefully (and becomes subject to garbage collection). However, this can take arbitrarily long to occur. By default, the task execution thread does not run as a *daemon thread*, so it is capable of keeping an application from terminating. If a caller wants to terminate a timer's task execution thread rapidly, the caller should invoke the timer's cancel method.

If the timer's task execution thread terminates unexpectedly, for example, because its stop method is invoked, any further attempt to schedule a task on the timer will result in an IllegalStateException, as if the timer's cancel method had been invoked.

This class is thread-safe: multiple threads can share a single Timer object without the need for external synchronization.

This class does *not* offer real-time guarantees: it schedules tasks using the Object.wait(long) method.

Implementation note: This class scales to large numbers of concurrently scheduled tasks (thousands should present no problem). Internally, it uses a binary heap to represent its task queue, so the cost to schedule a task is O(log n), where n is the number of concurrently scheduled tasks.

Implementation note: All constructors start a timer thread.

**Since:** 1.3 **See Also:**[TimerTask](http://docs.google.com/java/util/TimerTask.html), [Object.wait(long)](http://docs.google.com/java/lang/Object.html#wait(long))

| **Constructor Summary** | |
| --- | --- |
| [**Timer**](http://docs.google.com/java/util/Timer.html#Timer())()            Creates a new timer. |
| [**Timer**](http://docs.google.com/java/util/Timer.html#Timer(boolean))(boolean isDaemon)            Creates a new timer whose associated thread may be specified to run as a daemon. |
| [**Timer**](http://docs.google.com/java/util/Timer.html#Timer(java.lang.String))([String](http://docs.google.com/java/lang/String.html) name)            Creates a new timer whose associated thread has the specified name. |
| [**Timer**](http://docs.google.com/java/util/Timer.html#Timer(java.lang.String,%20boolean))([String](http://docs.google.com/java/lang/String.html) name, boolean isDaemon)            Creates a new timer whose associated thread has the specified name, and may be specified to run as a daemon. |

| **Method Summary** | |
| --- | --- |
| void | [**cancel**](http://docs.google.com/java/util/Timer.html#cancel())()            Terminates this timer, discarding any currently scheduled tasks. |
| int | [**purge**](http://docs.google.com/java/util/Timer.html#purge())()            Removes all cancelled tasks from this timer's task queue. |
| void | [**schedule**](http://docs.google.com/java/util/Timer.html#schedule(java.util.TimerTask,%20java.util.Date))([TimerTask](http://docs.google.com/java/util/TimerTask.html) task, [Date](http://docs.google.com/java/util/Date.html) time)            Schedules the specified task for execution at the specified time. |
| void | [**schedule**](http://docs.google.com/java/util/Timer.html#schedule(java.util.TimerTask,%20java.util.Date,%20long))([TimerTask](http://docs.google.com/java/util/TimerTask.html) task, [Date](http://docs.google.com/java/util/Date.html) firstTime, long period)            Schedules the specified task for repeated *fixed-delay execution*, beginning at the specified time. |
| void | [**schedule**](http://docs.google.com/java/util/Timer.html#schedule(java.util.TimerTask,%20long))([TimerTask](http://docs.google.com/java/util/TimerTask.html) task, long delay)            Schedules the specified task for execution after the specified delay. |
| void | [**schedule**](http://docs.google.com/java/util/Timer.html#schedule(java.util.TimerTask,%20long,%20long))([TimerTask](http://docs.google.com/java/util/TimerTask.html) task, long delay, long period)            Schedules the specified task for repeated *fixed-delay execution*, beginning after the specified delay. |
| void | [**scheduleAtFixedRate**](http://docs.google.com/java/util/Timer.html#scheduleAtFixedRate(java.util.TimerTask,%20java.util.Date,%20long))([TimerTask](http://docs.google.com/java/util/TimerTask.html) task, [Date](http://docs.google.com/java/util/Date.html) firstTime, long period)            Schedules the specified task for repeated *fixed-rate execution*, beginning at the specified time. |
| void | [**scheduleAtFixedRate**](http://docs.google.com/java/util/Timer.html#scheduleAtFixedRate(java.util.TimerTask,%20long,%20long))([TimerTask](http://docs.google.com/java/util/TimerTask.html) task, long delay, long period)            Schedules the specified task for repeated *fixed-rate execution*, beginning after the specified delay. |

| **Methods inherited from class java.lang.**[**Object**](http://docs.google.com/java/lang/Object.html) |
| --- |
| [clone](http://docs.google.com/java/lang/Object.html#clone()), [equals](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)), [finalize](http://docs.google.com/java/lang/Object.html#finalize()), [getClass](http://docs.google.com/java/lang/Object.html#getClass()), [hashCode](http://docs.google.com/java/lang/Object.html#hashCode()), [notify](http://docs.google.com/java/lang/Object.html#notify()), [notifyAll](http://docs.google.com/java/lang/Object.html#notifyAll()), [toString](http://docs.google.com/java/lang/Object.html#toString()), [wait](http://docs.google.com/java/lang/Object.html#wait()), [wait](http://docs.google.com/java/lang/Object.html#wait(long)), [wait](http://docs.google.com/java/lang/Object.html#wait(long,%20int)) |

| **Constructor Detail** |
| --- |

### Timer

public **Timer**()

Creates a new timer. The associated thread does *not* run as a daemon.

**See Also:**[Thread](http://docs.google.com/java/lang/Thread.html), [cancel()](http://docs.google.com/java/util/Timer.html#cancel())

### Timer

public **Timer**(boolean isDaemon)

Creates a new timer whose associated thread may be specified to run as a daemon. A daemon thread is called for if the timer will be used to schedule repeating "maintenance activities", which must be performed as long as the application is running, but should not prolong the lifetime of the application.

**Parameters:**isDaemon - true if the associated thread should run as a daemon.**See Also:**[Thread](http://docs.google.com/java/lang/Thread.html), [cancel()](http://docs.google.com/java/util/Timer.html#cancel())

### Timer

public **Timer**([String](http://docs.google.com/java/lang/String.html) name)

Creates a new timer whose associated thread has the specified name. The associated thread does *not* run as a daemon.

**Parameters:**name - the name of the associated thread **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if name is null**Since:** 1.5 **See Also:**[Thread.getName()](http://docs.google.com/java/lang/Thread.html#getName()), [Thread.isDaemon()](http://docs.google.com/java/lang/Thread.html#isDaemon())

### Timer

public **Timer**([String](http://docs.google.com/java/lang/String.html) name,  
 boolean isDaemon)

Creates a new timer whose associated thread has the specified name, and may be specified to run as a daemon.

**Parameters:**name - the name of the associated threadisDaemon - true if the associated thread should run as a daemon **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if name is null**Since:** 1.5 **See Also:**[Thread.getName()](http://docs.google.com/java/lang/Thread.html#getName()), [Thread.isDaemon()](http://docs.google.com/java/lang/Thread.html#isDaemon())

| **Method Detail** |
| --- |

### schedule

public void **schedule**([TimerTask](http://docs.google.com/java/util/TimerTask.html) task,  
 long delay)

Schedules the specified task for execution after the specified delay.

**Parameters:**task - task to be scheduled.delay - delay in milliseconds before task is to be executed. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if delay is negative, or delay + System.currentTimeMillis() is negative. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if task was already scheduled or cancelled, or timer was cancelled.

### schedule

public void **schedule**([TimerTask](http://docs.google.com/java/util/TimerTask.html) task,  
 [Date](http://docs.google.com/java/util/Date.html) time)

Schedules the specified task for execution at the specified time. If the time is in the past, the task is scheduled for immediate execution.

**Parameters:**task - task to be scheduled.time - time at which task is to be executed. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if time.getTime() is negative. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if task was already scheduled or cancelled, timer was cancelled, or timer thread terminated.

### schedule

public void **schedule**([TimerTask](http://docs.google.com/java/util/TimerTask.html) task,  
 long delay,  
 long period)

Schedules the specified task for repeated *fixed-delay execution*, beginning after the specified delay. Subsequent executions take place at approximately regular intervals separated by the specified period.

In fixed-delay execution, each execution is scheduled relative to the actual execution time of the previous execution. If an execution is delayed for any reason (such as garbage collection or other background activity), subsequent executions will be delayed as well. In the long run, the frequency of execution will generally be slightly lower than the reciprocal of the specified period (assuming the system clock underlying Object.wait(long) is accurate).

Fixed-delay execution is appropriate for recurring activities that require "smoothness." In other words, it is appropriate for activities where it is more important to keep the frequency accurate in the short run than in the long run. This includes most animation tasks, such as blinking a cursor at regular intervals. It also includes tasks wherein regular activity is performed in response to human input, such as automatically repeating a character as long as a key is held down.

**Parameters:**task - task to be scheduled.delay - delay in milliseconds before task is to be executed.period - time in milliseconds between successive task executions. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if delay is negative, or delay + System.currentTimeMillis() is negative. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if task was already scheduled or cancelled, timer was cancelled, or timer thread terminated.

### schedule

public void **schedule**([TimerTask](http://docs.google.com/java/util/TimerTask.html) task,  
 [Date](http://docs.google.com/java/util/Date.html) firstTime,  
 long period)

Schedules the specified task for repeated *fixed-delay execution*, beginning at the specified time. Subsequent executions take place at approximately regular intervals, separated by the specified period.

In fixed-delay execution, each execution is scheduled relative to the actual execution time of the previous execution. If an execution is delayed for any reason (such as garbage collection or other background activity), subsequent executions will be delayed as well. In the long run, the frequency of execution will generally be slightly lower than the reciprocal of the specified period (assuming the system clock underlying Object.wait(long) is accurate).

Fixed-delay execution is appropriate for recurring activities that require "smoothness." In other words, it is appropriate for activities where it is more important to keep the frequency accurate in the short run than in the long run. This includes most animation tasks, such as blinking a cursor at regular intervals. It also includes tasks wherein regular activity is performed in response to human input, such as automatically repeating a character as long as a key is held down.

**Parameters:**task - task to be scheduled.firstTime - First time at which task is to be executed.period - time in milliseconds between successive task executions. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if time.getTime() is negative. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if task was already scheduled or cancelled, timer was cancelled, or timer thread terminated.

### scheduleAtFixedRate

public void **scheduleAtFixedRate**([TimerTask](http://docs.google.com/java/util/TimerTask.html) task,  
 long delay,  
 long period)

Schedules the specified task for repeated *fixed-rate execution*, beginning after the specified delay. Subsequent executions take place at approximately regular intervals, separated by the specified period.

In fixed-rate execution, each execution is scheduled relative to the scheduled execution time of the initial execution. If an execution is delayed for any reason (such as garbage collection or other background activity), two or more executions will occur in rapid succession to "catch up." In the long run, the frequency of execution will be exactly the reciprocal of the specified period (assuming the system clock underlying Object.wait(long) is accurate).

Fixed-rate execution is appropriate for recurring activities that are sensitive to *absolute* time, such as ringing a chime every hour on the hour, or running scheduled maintenance every day at a particular time. It is also appropriate for recurring activities where the total time to perform a fixed number of executions is important, such as a countdown timer that ticks once every second for ten seconds. Finally, fixed-rate execution is appropriate for scheduling multiple repeating timer tasks that must remain synchronized with respect to one another.

**Parameters:**task - task to be scheduled.delay - delay in milliseconds before task is to be executed.period - time in milliseconds between successive task executions. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if delay is negative, or delay + System.currentTimeMillis() is negative. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if task was already scheduled or cancelled, timer was cancelled, or timer thread terminated.

### scheduleAtFixedRate

public void **scheduleAtFixedRate**([TimerTask](http://docs.google.com/java/util/TimerTask.html) task,  
 [Date](http://docs.google.com/java/util/Date.html) firstTime,  
 long period)

Schedules the specified task for repeated *fixed-rate execution*, beginning at the specified time. Subsequent executions take place at approximately regular intervals, separated by the specified period.

In fixed-rate execution, each execution is scheduled relative to the scheduled execution time of the initial execution. If an execution is delayed for any reason (such as garbage collection or other background activity), two or more executions will occur in rapid succession to "catch up." In the long run, the frequency of execution will be exactly the reciprocal of the specified period (assuming the system clock underlying Object.wait(long) is accurate).

Fixed-rate execution is appropriate for recurring activities that are sensitive to *absolute* time, such as ringing a chime every hour on the hour, or running scheduled maintenance every day at a particular time. It is also appropriate for recurring activities where the total time to perform a fixed number of executions is important, such as a countdown timer that ticks once every second for ten seconds. Finally, fixed-rate execution is appropriate for scheduling multiple repeating timer tasks that must remain synchronized with respect to one another.

**Parameters:**task - task to be scheduled.firstTime - First time at which task is to be executed.period - time in milliseconds between successive task executions. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if time.getTime() is negative. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if task was already scheduled or cancelled, timer was cancelled, or timer thread terminated.

### cancel

public void **cancel**()

Terminates this timer, discarding any currently scheduled tasks. Does not interfere with a currently executing task (if it exists). Once a timer has been terminated, its execution thread terminates gracefully, and no more tasks may be scheduled on it.

Note that calling this method from within the run method of a timer task that was invoked by this timer absolutely guarantees that the ongoing task execution is the last task execution that will ever be performed by this timer.

This method may be called repeatedly; the second and subsequent calls have no effect.

### purge

public int **purge**()

Removes all cancelled tasks from this timer's task queue. *Calling this method has no effect on the behavior of the timer*, but eliminates the references to the cancelled tasks from the queue. If there are no external references to these tasks, they become eligible for garbage collection.

Most programs will have no need to call this method. It is designed for use by the rare application that cancels a large number of tasks. Calling this method trades time for space: the runtime of the method may be proportional to n + c log n, where n is the number of tasks in the queue and c is the number of cancelled tasks.

Note that it is permissible to call this method from within a a task scheduled on this timer.

**Returns:**the number of tasks removed from the queue.**Since:** 1.5

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Timer.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/util/StringTokenizer.html)   [**NEXT CLASS**](http://docs.google.com/java/util/TimerTask.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/Timer.html)    [**NO FRAMES**](http://docs.google.com/Timer.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | [CONSTR](#3znysh7) | [METHOD](#2et92p0) | DETAIL: FIELD | [CONSTR](#3dy6vkm) | [METHOD](#3rdcrjn) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).