Source: <http://www.journaldev.com/1050/java-timer-timertask-example>

**Java Timer TimerTask Example**

Java **java.util.Timer** is a utility class that can be used to schedule a thread to be executed at certain time in future. **Java Timer** class can be used to schedule a task to be run one-time or to be run at regular intervals.

### Java TimerTask

**java.util.TimerTask** is an [**abstract class**](http://www.journaldev.com/1582/abstract-class-in-java) that implements Runnable interface and we need to extend this class to create our own **TimerTask** that can be scheduled using java Timer class.

## Java Timer Example

Java Timer class is thread safe and multiple threads can share a single **Timer** object without need for external synchronization. Timer class uses **java.util.TaskQueue** to add tasks at given regular interval and at any time there can be only one thread running the TimerTask, for example if you are creating a Timer to run every 10 seconds but single thread execution takes 20 seconds, then Timer object will keep adding tasks to the queue and as soon as one thread is finished, it will notify the queue and another thread will start executing.

Java Timer class uses Object [wait and notify](http://www.journaldev.com/1037/java-thread-wait-notify-and-notifyall-example) methods to schedule the tasks.

Here is a simple program for Java Timer and TimerTask example.

package com.journaldev.threads;

import java.util.Date;

import java.util.Timer;

import java.util.TimerTask;

public class MyTimerTask extends TimerTask {

@Override

public void run() {

System.out.println("Timer task started at:"+new Date());

completeTask();

System.out.println("Timer task finished at:"+new Date());

}

private void completeTask() {

try {

//assuming it takes 20 secs to complete the task

Thread.sleep(20000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

public static void main(String args[]){

TimerTask timerTask = new MyTimerTask();

//running timer task as daemon thread

Timer timer = new Timer(true);

timer.scheduleAtFixedRate(timerTask, 0, 10\*1000);

System.out.println("TimerTask started");

//cancel after sometime

try {

Thread.sleep(120000);

} catch (InterruptedException e) {

e.printStackTrace();

}

timer.cancel();

System.out.println("TimerTask cancelled");

try {

Thread.sleep(30000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

Notice that one thread execution will take 20 seconds but Java Timer object is scheduled to run the task every 10 seconds. Here is the output of the program:

TimerTask started

Timer task started at:Wed Dec 26 19:16:39 PST 2012

Timer task finished at:Wed Dec 26 19:16:59 PST 2012

Timer task started at:Wed Dec 26 19:16:59 PST 2012

Timer task finished at:Wed Dec 26 19:17:19 PST 2012

Timer task started at:Wed Dec 26 19:17:19 PST 2012

Timer task finished at:Wed Dec 26 19:17:39 PST 2012

Timer task started at:Wed Dec 26 19:17:39 PST 2012

Timer task finished at:Wed Dec 26 19:17:59 PST 2012

Timer task started at:Wed Dec 26 19:17:59 PST 2012

Timer task finished at:Wed Dec 26 19:18:19 PST 2012

Timer task started at:Wed Dec 26 19:18:19 PST 2012

TimerTask cancelled

Timer task finished at:Wed Dec 26 19:18:39 PST 2012

The output confirms that if a task is already executing, Timer will wait for it to finish and once finished, it will start again the next task from the queue.

Java Timer object can be created to run the associated tasks as a daemon thread. Timer *cancel()* method is used to terminate the timer and discard any scheduled tasks, however it doesn’t interfere with the currently executing task and let it finish. If the timer is run as [daemon thread](http://www.journaldev.com/1072/java-daemon-thread-example), whether we cancel it or not, it will terminate as soon as all the user threads are finished executing.

Timer class contains several **schedule**() methods to schedule a task to run once at given date or after some delay. There are several **scheduleAtFixedRate**() methods to run a task periodically with certain interval.

While scheduling tasks using Timer, you should make sure that time interval is more than normal thread execution, otherwise tasks queue size will keep growing and eventually task will be executing always. That’s all for a quick roundup on Java Timer and Java TimerTask.