Overview of Java Scheduled ExecutorService Policies

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
• Recognize the key methods provided by the Java ScheduledExecutorService interface & its related interfaces/classes
• Understand the policies defined by the ScheduledThreadPoolExecutor

Class ScheduledThreadPoolExecutor
java.lang.Object
    java.util.concurrent.AbstractExecutorService
    java.util.concurrent.ThreadPoolExecutor
    java.util.concurrent.ScheduledThreadPoolExecutor

All Implemented Interfaces:
Executor, ExecutorService, ScheduledExecutorService

class ScheduledThreadPoolExecutor
extends ThreadPoolExecutor
implements ScheduledExecutorService

A ThreadPoolExecutor that can additionally schedule commands to run after a given delay, or to execute periodically. This class is preferable to Timer when multiple worker threads are needed, or when the additional flexibility or capabilities of ThreadPoolExecutor (which this class extends) are required.

Delayed tasks execute no sooner than they are enabled, but without any real-time guarantees about when, after they are enabled, they will commence. Tasks scheduled for exactly the same execution time are enabled in first-in-first-out (FIFO) order of submission.
Overview of ScheduledThreadPoolExecutor & Its Policies
Overview of ScheduledThreadPoolExecutor & Its Policies

- ScheduledExecutorService must be implemented before it is useful (like any interface)

```java
<<Java Interface>>

ScheduledExecutorService

- schedule(Runnable, long, TimeUnit)
- schedule(Callable<V>, long, TimeUnit)
- scheduleAtFixedRate(Runnable, long, long, long, TimeUnit)
- scheduleWithFixedDelay(Runnable, long, long, long, TimeUnit)
```

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ScheduledExecutorService.html](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ScheduledExecutorService.html)
Overview of ScheduledThreadPoolExecutor & Its Policies

- ScheduledExecutorService must be implemented before it is useful (like any interface)
- The Executors utility class defines several factory methods that create implementations

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executors.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executors.html)
Overview of ScheduledThreadPoolExecutor & Its Policies

- ScheduledExecutorService must be implemented before it is useful (like any interface)
- The Executors utility class defines several factory methods that create implementations
- These implementations are based (directly or indirectly) on the ScheduledThreadPoolExecutor

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ScheduledThreadPoolExecutor.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/ScheduledThreadPoolExecutor.html)
Overview of ScheduledThreadPoolExecutor & Its Policies

- This class is implemented via several interfaces & classes

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ScheduledThreadPoolExecutor.html
Overview of ScheduledThreadPoolExecutor & Its Policies

- This class is implemented via several interfaces & classes

Schedules commands that run after a delay and/or executes them periodically
Overview of ScheduledThreadPoolExecutor & Its Policies

- Apps that need to change default policies need to access this class directly.

```java
<<Java Class>>
 ScheduledThreadPoolExecutor

- ScheduledThreadPoolExecutor(int)
- ScheduledThreadPoolExecutor(int, ThreadFactory)
- ScheduledThreadPoolExecutor(int, RejectedExecutionHandler)
- ScheduledThreadPoolExecutor(int, ThreadFactory, RejectedExecutionHandler)
- schedule(Runnable, long, TimeUnit)
- schedule(Callable<V>, long, TimeUnit)
- scheduleAtFixedRate(Runnable, long, long, TimeUnit)
- scheduleWithFixedDelay(Runnable, long, long, TimeUnit)
- execute(Runnable): void
- submit(Runnable)
- submit(Runnable, T)
- submit(Callable<T>)

- setContinueExistingPeriodicTasksAfterShutdownPolicy(boolean): void
- getContinueExistingPeriodicTasksAfterShutdownPolicy(): boolean
- setExecuteExistingDelayedTasksAfterShutdownPolicy(boolean): void
- getExecuteExistingDelayedTasksAfterShutdownPolicy(): boolean
- setRemoveOnCancelPolicy(boolean): void
- getRemoveOnCancelPolicy(): boolean
- shutdown(): void
- shutdownNow(): List<Runnable>
- getQueue(): BlockingQueue<Runnable>
```
Overview of ScheduledThreadPoolExecutor & Its Policies

- Apps that need to change default policies need to access this class directly, e.g.
- Set/get policy to continue running periodic tasks even when executor shuts down (defaults to true)
Overview of ScheduledThreadPoolExecutor & Its Policies

- Apps that need to change default policies need to access this class directly, e.g.
  - Set/get policy to continue running periodic tasks even when executor shuts down (defaults to true)
  - Set/get policy to run delayed tasks even when the executor has been shutdown (defaults to true)
Overview of ScheduledThreadPoolExecutor & Its Policies

• Apps that need to change default policies need to access this class directly, e.g.
  • Set/get policy to continue running periodic tasks even when executor shuts down (defaults to true)
  • Set/get policy to run delayed tasks even when the executor has been shutdown (defaults to true)
  • Set/gets the policy to immediately remove tasks from work queue when they are cancelled (defaults to false)
Oddly, all three of these policies are set “backwards” by default.

### Overview of ScheduledThreadPoolExecutor & Its Policies

- Apps that need to change default policies need to access this class directly.

<table>
<thead>
<tr>
<th>Java Class</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScheduledThreadPoolExecutor</td>
<td>setContinueExistingPeriodicTasksAfterShutdownPolicy(boolean):void</td>
</tr>
<tr>
<td></td>
<td>getContinueExistingPeriodicTasksAfterShutdownPolicy():boolean</td>
</tr>
<tr>
<td></td>
<td>setExecuteExistingDelayedTasksAfterShutdownPolicy(boolean):void</td>
</tr>
<tr>
<td></td>
<td>getExecuteExistingDelayedTasksAfterShutdownPolicy():boolean</td>
</tr>
<tr>
<td></td>
<td>setRemoveOnCancelPolicy(boolean):void</td>
</tr>
<tr>
<td></td>
<td>getRemoveOnCancelPolicy():boolean</td>
</tr>
<tr>
<td></td>
<td>shutdown():void</td>
</tr>
<tr>
<td></td>
<td>shutdownNow():List&lt;Runnable&gt;</td>
</tr>
<tr>
<td></td>
<td>getQueue():BlockingQueue&lt;Runnable&gt;</td>
</tr>
</tbody>
</table>

- Set/get policy to continue running existing periodic tasks even when executor has been shutdown
- Set/get the policy to run existing delayed tasks even when executor has been shutdown
- Set/gets the policy to immediately remove tasks from work queue when they are cancelled
Overview of ScheduledThreadPoolExecutor & Its Policies

- Don’t try to access a ScheduledThreadPoolExecutor by casting the result of Executors.newSingleThreadScheduledExecutor()

```
ScheduledThreadPoolExecutor schedTPExec =
    (ScheduledThreadPoolExecutor) Executors.newSingleThreadScheduledExecutor();
```

See [stackoverflow.com/questions/10200230/creatingscheduledthreadpoolexecutor-using-executors](https://stackoverflow.com/questions/10200230/creatingscheduledthreadpoolexecutor-using-executors)
Overview of ScheduledThreadPoolExecutor & Its Policies

- Don’t try to access a ScheduledThreadPoolExecutor by casting the result of Executors.newSingleThreadScheduledExecutor()

```java
ScheduledThreadPoolExecutor schedTPExec =
   (ScheduledThreadPoolExecutor)
   Executors.newSingleThreadScheduledExecutor();
```

`newSingleThreadScheduledExecutor()` doesn’t return `ScheduledThreadPoolExecutor`!

```java
class Executors {
...

    public static ScheduledExecutorService
        newSingleThreadScheduledExecutor() {
        return new DelegatedScheduledExecutorService
            (new ScheduledThreadPoolExecutor(1));
    }
}
```

See [java/util/concurrent/Executors.java#Executors.newSingleThreadScheduledExecutor](java/util/concurrent/Executors.java#Executors.newSingleThreadScheduledExecutor)
Overview of ScheduledThreadPoolExecutor & Its Policies

- Therefore, if you need a single-threaded ScheduledThreadPoolExecutor with non-default policies you’ll need to make one yourself

```java
ScheduledThreadPoolExecutor exec =
    new ScheduledThreadPoolExecutor(1);

or

ScheduledThreadPoolExecutor exec = (ScheduledThreadPoolExecutor)
    Executors.newScheduledThreadPool(1);
```

```java
exec.setRemoveOnCancelPolicy(true);
exec.setContinueExistingPeriodicTasksAfterShutdownPolicy(false);
exec.setExecuteExistingDelayedTasksAfterShutdownPolicy(false);
```

See lesson on “The Java ScheduledExecutorService: Implementing TimedMemoizerEx”
Therefore, if you need a single-threaded ScheduledThreadPoolExecutor with non-default policies you’ll need to make one yourself:

```java
ScheduledThreadPoolExecutor exec =
    new ScheduledThreadPoolExecutor(1);
```

Or

```java
ScheduledThreadPoolExecutor exec = (ScheduledThreadPoolExecutor)
    Executors.newScheduledThreadPool(1);
```

```java
exec.setRemoveOnCancelPolicy(true);
exec.setContinueExistingPeriodicTasksAfterShutdownPolicy(false);
exec.setExecuteExistingDelayedTasksAfterShutdownPolicy(false);
```

Therefore, if you need a single-threaded ScheduledThreadPoolExecutor with non-default policies you’ll need to make one yourself

```java
ScheduledThreadPoolExecutor exec =
    new ScheduledThreadPoolExecutor(1);
```

Or

```java
ScheduledThreadPoolExecutor exec = (ScheduledThreadPoolExecutor) Executors.newScheduledThreadPool(1);
```

Set default policies to something more sensible

```java
exec.setRemoveOnCancelPolicy(true);
exec.setContinueExistingPeriodicTasksAfterShutdownPolicy(false);
exec.setExecuteExistingDelayedTasksAfterShutdownPolicy(false);
```
End of Overview of Java ScheduledExecutor Service Policies