

# Java ExecutorService: Key Methods

**Douglas C. Schmidt**

**[d.schmidt@vanderbilt.edu](mailto:d.schmidt@vanderbilt.edu)**

**[www.dre.vanderbilt.edu/~schmidt](http://www.dre.vanderbilt.edu/~schmidt)**



**Professor of Computer Science**













**Institute for Software  
Integrated Systems**

**Vanderbilt University  
Nashville, Tennessee, USA**




# Learning Objectives in this Part of the Lesson

- Recognize the powerful features defined in the Java ExecutorService interface
- Understand other interfaces related to ExecutorService
- Know the key methods provided by ExecutorService

<<Java Interface>>  <b>ExecutorService</b>	
	<code>shutdown():void</code>
	<code>shutdownNow():List&lt;Runnable&gt;</code>
	<code>isShutdown():boolean</code>
	<code>isTerminated():boolean</code>
	<code>awaitTermination(long,TimeUnit):boolean</code>
	<code>submit(Callable&lt;T&gt;):Future&lt;T&gt;</code>
	<code>submit(Runnable,T):Future&lt;T&gt;</code>
	<code>submit(Runnable):Future&lt;?&gt;</code>
	<code>invokeAll(Collection&lt;? extends Callable&lt;T&gt;&gt;):List&lt;Future&lt;T&gt;&gt;</code>
	<code>invokeAny(Collection&lt;? extends Callable&lt;T&gt;&gt;)</code>
	<code>invokeAny(Collection&lt;? extends Callable&lt;T&gt;&gt;,long,TimeUnit)</code>

# Learning Objectives in this Part of the Lesson

- Recognize the powerful features defined in the Java ExecutorService interface
- Understand other interfaces related to ExecutorService
- Know the key methods provided by ExecutorService
  - These methods submit 1+ tasks for asynchronous execution & manage the lifecycle of tasks & the Executor Service itself

<<Java Interface>>  <b>ExecutorService</b>
<ul style="list-style-type: none"><li>• shutdown():void</li><li>• shutdownNow():List&lt;Runnable&gt;</li><li>• isShutdown():boolean</li><li>• isTerminated():boolean</li><li>• awaitTermination(long,TimeUnit):boolean</li><li>• submit(Callable&lt;T&gt;):Future&lt;T&gt;</li><li>• submit(Runnable,T):Future&lt;T&gt;</li><li>• submit(Runnable):Future&lt;?&gt;</li><li>• invokeAll(Collection&lt;? extends Callable&lt;T&gt;&gt;):List&lt;Future&lt;T&gt;&gt;</li><li>• invokeAny(Collection&lt;? extends Callable&lt;T&gt;&gt;)</li><li>• invokeAny(Collection&lt;? extends Callable&lt;T&gt;&gt;,long,TimeUnit)</li></ul>

---

# Key Methods in the ExecutorService Interface: Task Execution

# Key Methods in the ExecutorService Interface

---

- ExecutorService can execute individual tasks

```
public interface ExecutorService
    extends Executor {
    // Inherited from Executor
    void execute(Runnable command);

    <T> Future<T> submit
        (Callable<T> task);
    ...
}
```

# Key Methods in the ExecutorService Interface

---

- ExecutorService can execute individual tasks
  - execute() runs one-way tasks that return void



```
public interface ExecutorService
    extends Executor {

    // Inherited from Executor
    void execute(Runnable command);

    <T> Future<T> submit
        (Callable<T> task);

    ...
}
```

---

However, this method isn't very useful/common in practice

# Key Methods in the ExecutorService Interface

- ExecutorService can execute individual tasks
  - `execute()` runs one-way tasks that return void
  - `submit()` runs two-way async tasks that return a value via a future



```
public interface ExecutorService
    extends Executor {

    // Inherited from Executor
    void execute(Runnable command);

    <T> Future<T> submit
        (Callable<T> task);

    ...
}
```

This method is the most useful/common in practice

# Key Methods in the ExecutorService Interface

---

- ExecutorService can execute individual tasks
  - `execute()` runs one-way tasks that return void
  - `submit()` runs two-way async tasks that return a value via a future
  - Supports the “synchronous future” processing model

```
public interface ExecutorService
    extends Executor {

    // Inherited from Executor
    void execute(Runnable command);

    <T> Future<T> submit
        (Callable<T> task);

    ...
}
```



# Key Methods in the ExecutorService Interface

---

- ExecutorService can execute individual tasks
  - `execute()` runs one-way tasks that return void
  - `submit()` runs two-way async tasks that return a value via a future
    - Supports the “synchronous future” processing model
    - `Future.get()` can block until task completes successfully

```
public interface ExecutorService
    extends Executor {

    // Inherited from Executor
    void execute(Runnable command);

    <T> Future<T> submit
        (Callable<T> task);

    ...
}
```

# Key Methods in the ExecutorService Interface

---

- ExecutorService can execute individual tasks

- `execute()` runs one-way tasks that return void

- `submit()` runs two-way async tasks that return a value via a future

- Supports the “synchronous future” processing model

- `Future.get()` can block until task completes successfully

- After which point `get()` returns the task’s result

```
public interface ExecutorService
    extends Executor {

    // Inherited from Executor
    void execute(Runnable command);

    <T> Future<T> submit
        (Callable<T> task);

    ...
}
```

# Key Methods in the ExecutorService Interface

---

- ExecutorService can execute individual tasks
  - `execute()` runs one-way tasks that return void
  - `submit()` runs two-way async tasks that return a value via a future
  - `submit()` can also run one-way async tasks that return no value

```
public interface ExecutorService
    extends Executor {
    // Inherited from Executor
    void execute(Runnable command);

    <T> Future<T> submit
        (Callable<T> task);

    <T> Future<T> submit
        (Runnable task);
    ...
}
```

# Key Methods in the ExecutorService Interface

- ExecutorService can execute individual tasks
  - `execute()` runs one-way tasks that return void
  - `submit()` runs two-way async tasks that return a value via a future
  - `submit()` can also run one-way async tasks that return no value
    - It is possible to cancel this computation, however

```
public interface ExecutorService
    extends Executor {

    // Inherited from Executor
    void execute(Runnable command);

    <T> Future<T> submit
        (Callable<T> task);

    <T> Future<T> submit
        (Runnable task);
    ...
}
```



# Key Methods in the ExecutorService Interface

---

- ExecutorService can also execute groups of tasks

```
public interface ExecutorService
    extends Executor {

    ...

    <T> List<Future<T>> invokeAll
        (Collection<? extends
         Callable<T>> tasks) ...;

    <T> T invokeAny
        (Collection<? extends
         Callable<T>> tasks) ...;

    ...
}
```

# Key Methods in the ExecutorService Interface

- ExecutorService can also execute groups of tasks

```
public interface ExecutorService
    extends Executor {

    ...

    <T> List<Future<T>> invokeAll
        (Collection<? extends
         Callable<T>> tasks) ...;

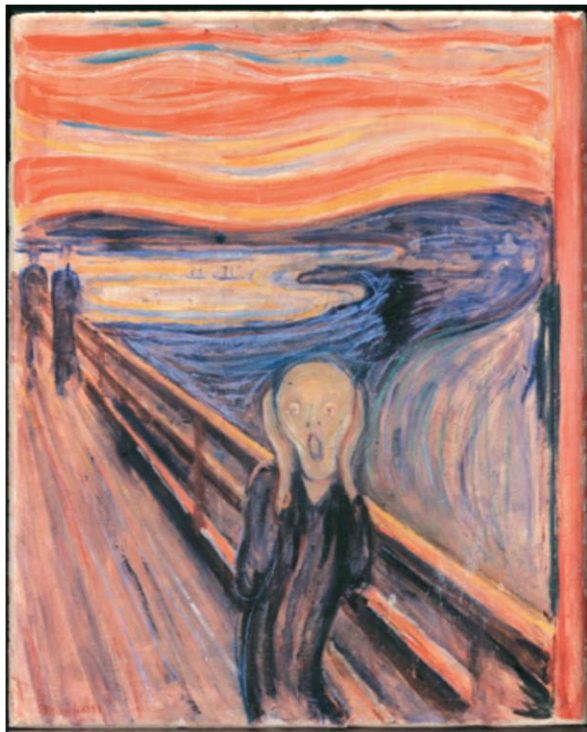
    <T> T invokeAny
        (Collection<? extends
         Callable<T>> tasks) ...;

    ...
}
```

*Groups of tasks can be passed to these methods as collection parameters*

# Key Methods in the ExecutorService Interface

- ExecutorService can also execute groups of tasks



```
public interface ExecutorService  
    extends Executor {
```

```
    ...  
    <T> List<Future<T>> invokeAll  
        (Collection<? extends  
            Callable<T>> tasks) ...;  
  
    <T> T invokeAny  
        (Collection<? extends  
            Callable<T>> tasks) ...;  
    ...
```

Don't modify collection param while invokeAll() or invokeAny() are running!!!

# Key Methods in the ExecutorService Interface

- ExecutorService can also execute groups of tasks
- Returns a list of futures when all tasks complete

```
public interface ExecutorService
    extends Executor {

    ...

    <T> List<Future<T>> invokeAll
        (Collection<? extends
         Callable<T>> tasks) ...;

    <T> T invokeAny
        (Collection<? extends
         Callable<T>> tasks) ...;

    ...
}
```



# Key Methods in the ExecutorService Interface

- ExecutorService can also execute groups of tasks
- Returns a list of futures when all tasks complete

```
public interface ExecutorService
    extends Executor {

    ...

    <T> List<Future<T>> invokeAll
        (Collection<? extends
         Callable<T>> tasks) ...;

    <T> T invokeAny
        (Collection<? extends
         Callable<T>> tasks) ...;

    ...
}
```

*All futures returned in  
this list are "done"!*

Futures are used to indicate whether task terminate normally or exceptionally

# Key Methods in the ExecutorService Interface

- ExecutorService can also execute groups of tasks

- Returns a list of futures when all tasks complete
- Return the result of *one* successful completion

```
public interface ExecutorService
    extends Executor {

    ...

    <T> List<Future<T>> invokeAll
        (Collection<? extends
         Callable<T>> tasks) ...;

    <T> T invokeAny
        (Collection<? extends
         Callable<T>> tasks) ...;

    ...
}
```

Useful for concurrent algorithms that just want the result that completes first

# Key Methods in the ExecutorService Interface

- ExecutorService can also execute groups of tasks
  - Returns a list of futures when all tasks complete
  - Return the result of *one* successful completion
    - Cancel uncompleted tasks



```
public interface ExecutorService
    extends Executor {

    ...

    <T> List<Future<T>> invokeAll
        (Collection<? extends
         Callable<T>> tasks) ...;

    <T> T invokeAny
        (Collection<? extends
         Callable<T>> tasks) ...;

    ...
}
```

# Key Methods in the ExecutorService Interface

- ExecutorService can also execute groups of tasks

- Returns a list of futures when all tasks complete

- Return the result of *one* successful completion

- Cancel uncompleted tasks

- Ignore other completed task results



Ignore

```
public interface ExecutorService  
    extends Executor {
```

```
    ...
```

```
<T> List<Future<T>> invokeAll  
    (Collection<? extends  
        Callable<T>> tasks) ...;
```

```
<T> T invokeAny  
    (Collection<? extends  
        Callable<T>> tasks) ...;
```

```
    ...
```

# Key Methods in the ExecutorService Interface

- ExecutorService can also execute groups of tasks

- Returns a list of futures when all tasks complete
- Return the result of *one* successful completion

*These methods block the calling thread until they are finished, which may be non-intuitive..*

```
public interface ExecutorService  
    extends Executor {
```

```
    ...  
    <T> List<Future<T>> invokeAll  
        (Collection<? extends  
            Callable<T>> tasks) ...;
```

```
    <T> T invokeAny  
        (Collection<? extends  
            Callable<T>> tasks) ...;
```

```
    ...
```



# Key Methods in the ExecutorService Interface

- ExecutorService can also execute groups of tasks

- Returns a list of futures when all tasks complete
- Return the result of *one* successful completion

*These overloaded methods block for up to a given amount of time*



```
public interface ExecutorService
    extends Executor {

    ...

    <T> List<Future<T>> invokeAll
        (Collection<? extends
            Callable<T>> tasks,
            long timeout, TimeUnit unit)
        ...;

    <T> T invokeAny(Collection<?
        extends Callable<T>> tasks,
            long timeout, TimeUnit unit)
        ...;

    ...
}
```

# Key Methods in the ExecutorService Interface

- ExecutorService can also execute groups of tasks

- Returns a list of futures when all tasks complete
- Return the result of *one* successful completion

```
public interface ExecutorService
    extends Executor {

    ...

    <T> List<Future<T>> invokeAll
        (Collection<? extends
        Callable<T>> tasks,
        long timeout, TimeUnit unit)
        ...;

    <T> T invokeAny(Collection<?
        extends Callable<T>> tasks,
        long timeout, TimeUnit unit)
        ...;

    ...
}
```

*If method didn't time out, each task completed, whereas if it did time out, some tasks will not have completed.*

Task that have not completed are cancelled if timeout occurs.

# Key Methods in the ExecutorService Interface

- ExecutorService can also execute groups of tasks

- Returns a list of futures when all tasks complete
- Return the result of *one* successful completion

```
public interface ExecutorService
    extends Executor {

    ...

    <T> List<Future<T>> invokeAll
        (Collection<? extends
         Callable<T>> tasks,
         long timeout, TimeUnit unit)
        ...;

    <T> T invokeAny(Collection<?
        extends Callable<T>> tasks,
        long timeout, TimeUnit unit)
        ...;

    ...
}
```

*TimeoutException is thrown if timeout elapses*

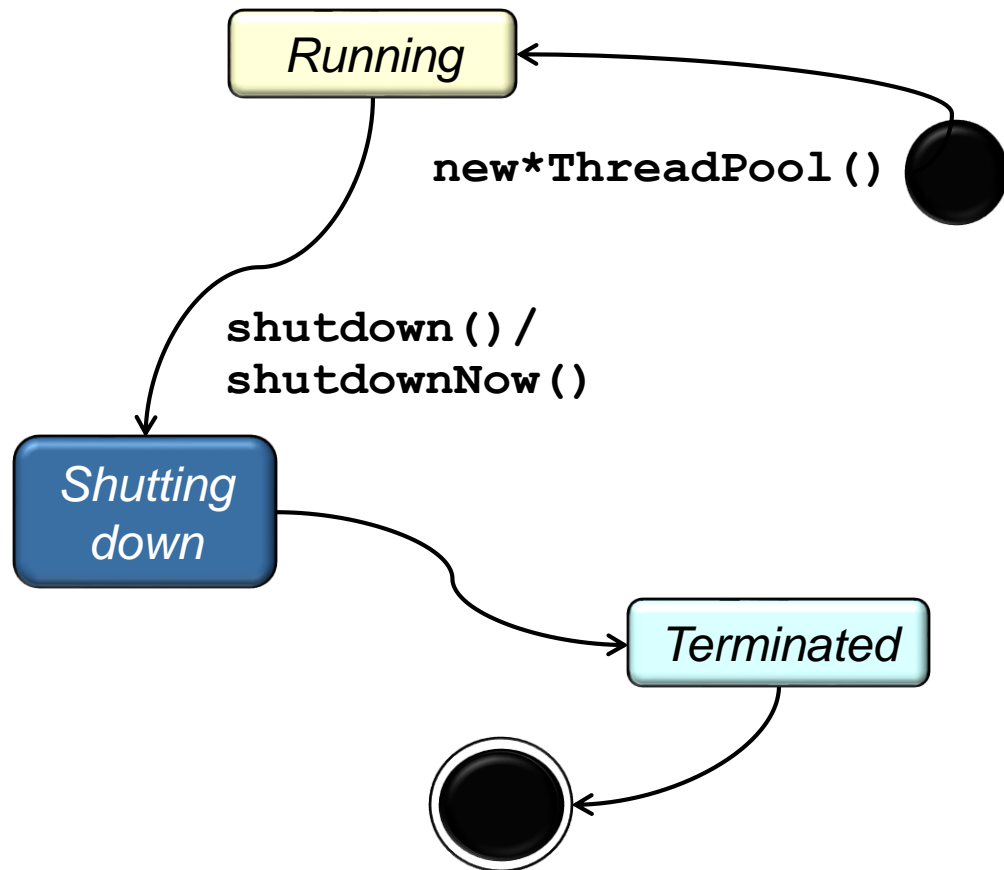


---

# Key Methods in the ExecutorService Interface: Lifecycle Management

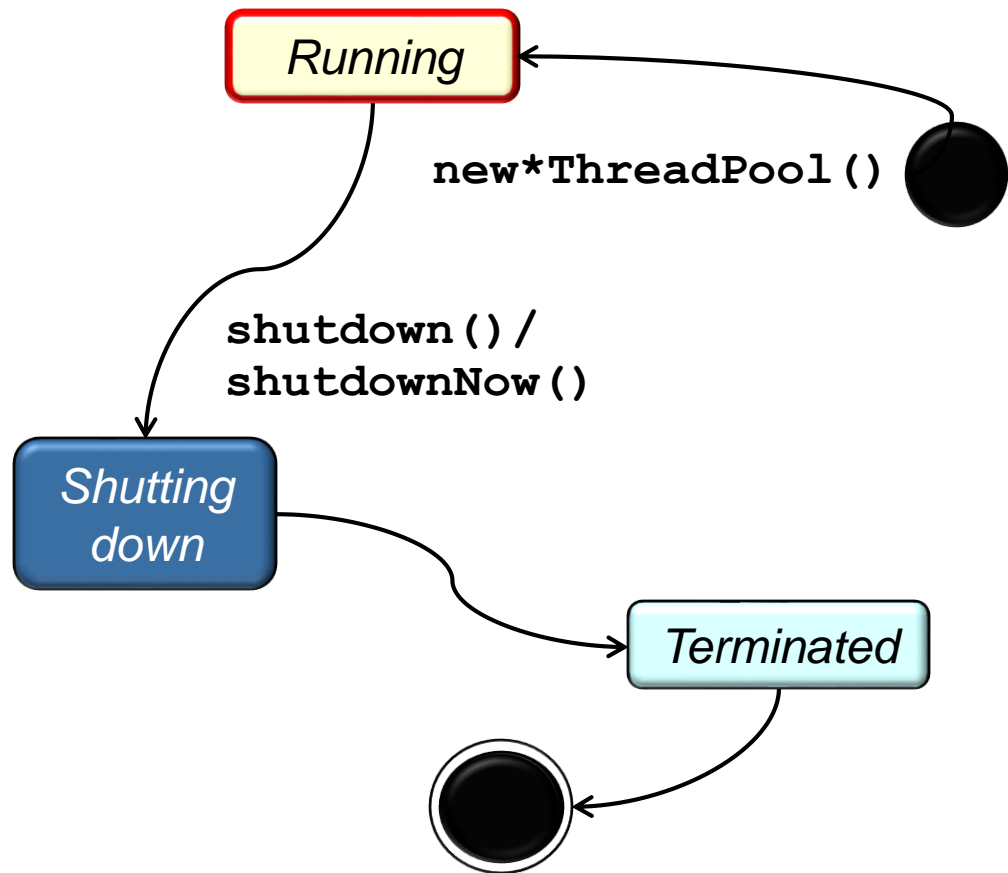
# Key Methods in the ExecutorService Interface

- An ExecutorService instance can be in one of three states



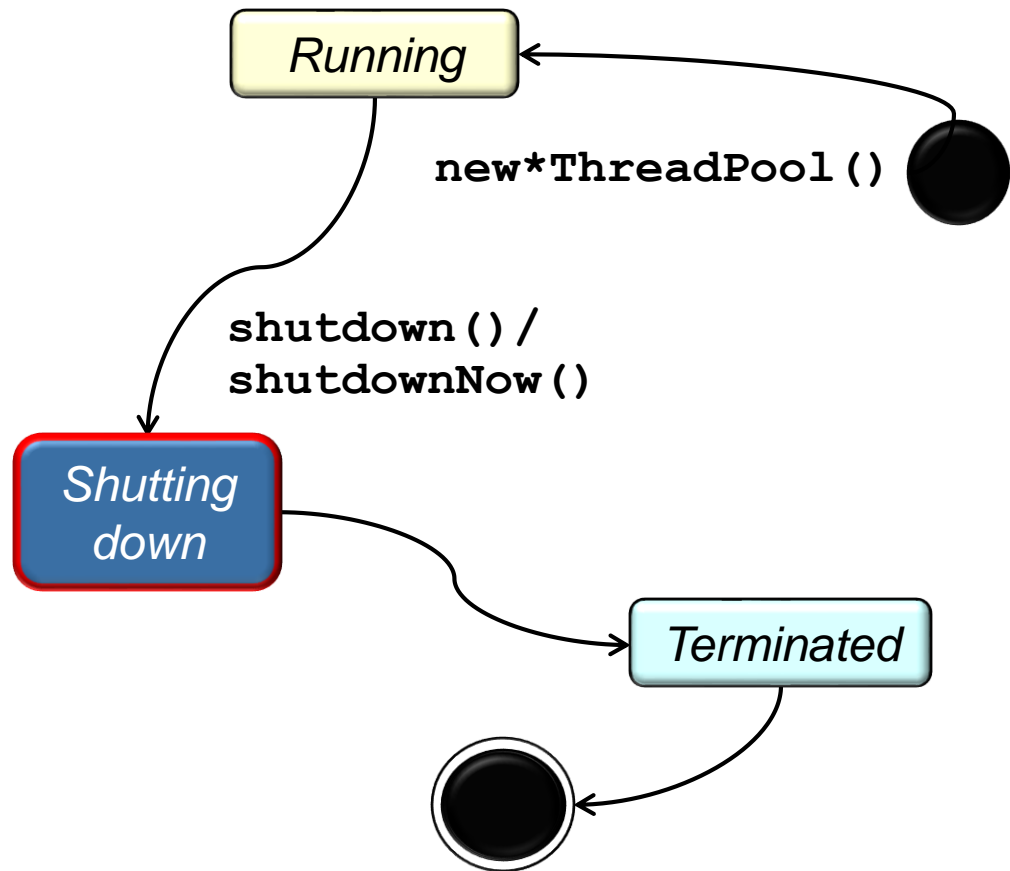
# Key Methods in the ExecutorService Interface

- An ExecutorService instance can be in one of three states
  - Running
    - After being created via a factory method



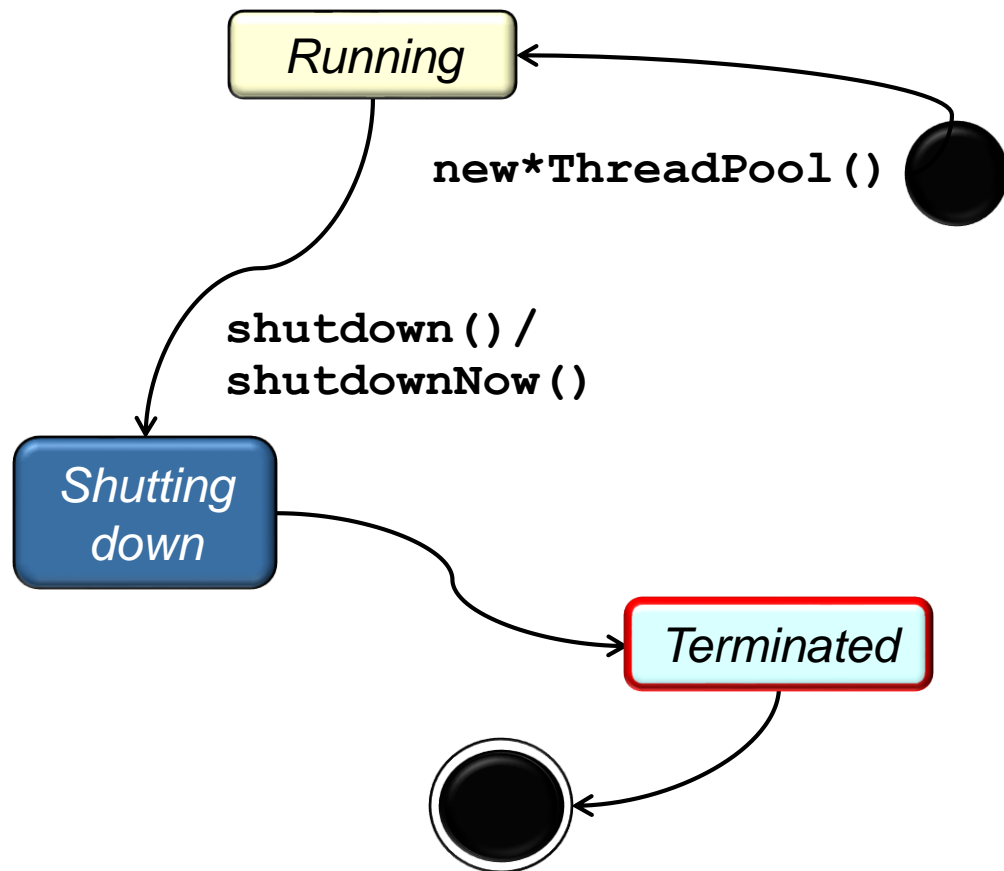
# Key Methods in the ExecutorService Interface

- An ExecutorService instance can be in one of three states
  - Running
  - Shutting down
    - After being shut down gracefully or abruptly



# Key Methods in the ExecutorService Interface

- An ExecutorService instance can be in one of three states
  - Running
  - Shutting down
  - Terminated
    - After all tasks have completed



# Key Methods in the ExecutorService Interface

- An ExecutorService client can initiate shutdown operations to manage its lifecycle

```
public interface ExecutorService  
    extends Executor {  
  
    ...  
    void shutdown();  
  
    List<Runnable> shutdownNow();  
    ...  
}
```



# Key Methods in the ExecutorService Interface

---

- An ExecutorService client can initiate shutdown operations to manage its lifecycle
- Performs “graceful shutdown” that completes active tasks



```
public interface ExecutorService
    extends Executor {

    ...
    void shutdown();

    List<Runnable> shutdownNow();
    ...
}
```

# Key Methods in the ExecutorService Interface

- An ExecutorService client can initiate shutdown operations to manage its lifecycle
- Performs “graceful shutdown” that completes active tasks
- But ignores new tasks & doesn’t process waiting tasks

```
public interface ExecutorService  
    extends Executor {  
    ...  
    void shutdown();  
  
    List<Runnable> shutdownNow();  
    ...  
}
```





# Key Methods in the ExecutorService Interface

- An ExecutorService client can initiate shutdown operations to manage its lifecycle
  - Performs “graceful shutdown” that completes active tasks
  - Performs “abrupt shutdown” that cancels active tasks & doesn’t process waiting tasks

```
public interface ExecutorService
    extends Executor {

    ...
    void shutdown();

    List<Runnable> shutdownNow();
    ...
}
```



# Key Methods in the ExecutorService Interface

- An ExecutorService client can initiate shutdown operations to manage its lifecycle
  - Performs “graceful shutdown” that completes active tasks
  - Performs “abrupt shutdown” that cancels active tasks & doesn’t process waiting tasks
    - Active tasks are cancelled by posting an interrupt request to executor thread(s)

```
public interface ExecutorService
    extends Executor {

    ...

    void shutdown() ;

    List<Runnable> shutdownNow() ;

    ...
}
```



# Key Methods in the ExecutorService Interface

- An ExecutorService client can initiate shutdown operations to manage its lifecycle
  - Performs “graceful shutdown” that completes active tasks
  - Performs “abrupt shutdown” that cancels active tasks & doesn’t process waiting tasks
  - Active tasks are cancelled by posting an interrupt request to executor thread(s)

*Java interrupt requests are “voluntary” & require cooperation between threads*

```
public interface ExecutorService
    extends Executor {

    ...

    void shutdown();

    List<Runnable> shutdownNow();

    ...
}
```



See [weblogs.java.net/blog/2009/03/02/cancelling-tasks-threadinterrupt-fragility](http://weblogs.java.net/blog/2009/03/02/cancelling-tasks-threadinterrupt-fragility)

# Key Methods in the ExecutorService Interface

- An ExecutorService client can initiate shutdown operations to manage its lifecycle
  - Performs “graceful shutdown” that completes active tasks
  - Performs “abrupt shutdown” that cancels active tasks & doesn’t process waiting tasks
    - Active tasks are cancelled by posting an interrupt request to executor thread(s)
    - Returns waiting tasks

```
public interface ExecutorService
    extends Executor {

    ...

    void shutdown();

    List<Runnable> shutdownNow();

    ...
}
```



# Key Methods in the ExecutorService Interface

- An ExecutorService client can initiate shutdown operations to manage its lifecycle
  - Performs “graceful shutdown” that completes active tasks
  - Performs “abrupt shutdown” that cancels active tasks & doesn’t process waiting tasks
- Tasks submitted after an Executor Service is shut down are dealt with by RejectedExceptionHandler

## Interface RejectedExecutionHandler

### All Known Implementing Classes:

`ThreadPoolExecutor.AbortPolicy`,  
`ThreadPoolExecutor.CallerRunsPolicy`,  
`ThreadPoolExecutor.DiscardOldestPolicy`,  
`ThreadPoolExecutor.DiscardPolicy`

---

```
public interface RejectedExecutionHandler
```

A handler for tasks that cannot be executed by a `ThreadPoolExecutor`.

# Key Methods in the ExecutorService Interface

- An ExecutorService client can initiate shutdown operations to manage its lifecycle
  - Performs “graceful shutdown” that completes active tasks
  - Performs “abrupt shutdown” that cancels active tasks & doesn’t process waiting tasks
- Tasks submitted after an Executor Service is shut down are dealt with by RejectedExceptionHandler
  - Can silently discard task or throw RejectedExecutionException

## Class RejectedExecutionException

```
java.lang.Object
    java.lang.Throwable
        java.lang.Exception
            java.lang.RuntimeException
                java.util.concurrent.RejectedExecutionException
```

### All Implemented Interfaces:

Serializable

---

```
public class RejectedExecutionException
    extends RuntimeException
```

Exception thrown by an Executor when a task cannot be accepted for execution.

# Key Methods in the ExecutorService Interface

---

- Clients of ExecutorService can query the status of a shutdown & wait for termination to finish

```
public interface ExecutorService
    extends Executor {

    ...

    boolean isShutdown();

    boolean isTerminated();

    boolean awaitTermination
        (long timeout,
         TimeUnit unit) ...;
```

# Key Methods in the ExecutorService Interface

---

- Clients of ExecutorService can query the status of a shutdown & wait for termination to finish
- True if executor shut down
  - i.e., in “shutting down” state

```
public interface ExecutorService
    extends Executor {

    ...

    boolean isShutdown();

    boolean isTerminated();

    boolean awaitTermination
        (long timeout,
         TimeUnit unit) ...;
```



# Key Methods in the ExecutorService Interface

---

- Clients of ExecutorService can query the status of a shutdown & wait for termination to finish
  - True if executor shut down
  - True if all tasks have completed after executor was shut down
    - i.e., in “terminated” state

```
public interface ExecutorService
    extends Executor {

    ...

    boolean isShutdown();

    boolean isTerminated();

    boolean awaitTermination
        (long timeout,
         TimeUnit unit) ...;
```

# Key Methods in the ExecutorService Interface

---

- Clients of ExecutorService can query the status of a shutdown & wait for termination to finish
  - True if executor shut down
  - True if all tasks have completed after executor was shut down
  - Blocks until all tasks complete

```
public interface ExecutorService
    extends Executor {

    ...

    boolean isShutdown();

    boolean isTerminated();

    boolean awaitTermination
        (long timeout,
         TimeUnit unit) ...;
```

# Key Methods in the ExecutorService Interface

- Clients of ExecutorService can query the status of a shutdown & wait for termination to finish
  - True if executor shut down
  - True if all tasks have completed after executor was shut down
  - Blocks until all tasks complete

```
public interface ExecutorService
    extends Executor {

    ...

    boolean isShutdown();

    boolean isTerminated();

    boolean awaitTermination
        (long timeout,
         TimeUnit unit) ...;
```

*shutdownNow() might reduce the blocking time for awaitTermination()*

# Key Methods in the ExecutorService Interface

- Clients of ExecutorService can query the status of a shutdown & wait for termination to finish
  - True if executor shut down
  - True if all tasks have completed after executor was shut down
- Blocks until all tasks complete



```
public interface ExecutorService
    extends Executor {

    ...

    boolean isShutdown();

    boolean isTerminated();

    boolean awaitTermination
        (long timeout,
         TimeUnit unit) ...;
```

*shutdown\*() & awaitTermination()  
provide barrier synchronization*

See [en.wikipedia.org/wiki/Barrier\\_\(computer\\_science\)](https://en.wikipedia.org/wiki/Barrier_(computer_science))

---

# End of Java Executor Service: Key Methods