Key Methods in the Java ExecutorService

(Part 2)

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
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
  - These methods also manage the lifecycle of tasks & the Executor Service itself
Key Methods in the ExecutorService Interface: Lifecycle Management
An ExecutorService instance can be in one of three states:

- **Running**
  - `new*ThreadPool()`
  - `shutdown()` / `shutdownNow()`
- **Shutting down**
- **Terminated**
An ExecutorService instance can be in one of three states:

- **Running**
  - After being created via a factory method

Key Methods in the ExecutorService Interface:

- `new*ThreadPool()`
- `shutdown()`/
  - `shutdownNow()`
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:

- `new*ThreadPool()`
- `shutdown() / shutdownNow()`
An ExecutorService instance can be in one of three states:

- Running
- Shutting down
- Terminated
  - After all tasks have completed
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

```java
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

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

```java
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
public interface ExecutorService extends Executor {

    // graceful shutdown
    void shutdown();

    // abrupt shutdown
    List<Runnable> shutdownNow();
}
```

See [docs.oracle.com/javase/tutorial/essential/concurrency/interrupt.html](https://docs.oracle.com/javase/tutorial/essential/concurrency/interrupt.html)
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
public interface ExecutorService
extends Executor {

  ...  
  void shutdown();

  List<Runnable> shutdownNow();

  ...
```

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

See weblogs.java.net/blog/2009/03/02/cancelling-tasks-threadinterrupt-fragility
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
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 RejectedExecutionExceptionHandler

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/RejectedExecutionHandler.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/RejectedExecutionHandler.html)
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
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.

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/RejectedExecutionException.html
Clients of ExecutorService can query the status of a shutdown & wait for termination to finish.

```java
public interface ExecutorService extends Executor {
    ...

    boolean isShutdown();

    boolean isTerminated();

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

```java
public interface ExecutorService extends Executor {
    ...
    boolean isShUTDOWN();
    boolean isTerminated();
    boolean awaitTermination
        (long timeout,
         TimeUnit unit) ...;
```
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

```java
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

```java
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

```java
public interface ExecutorService
    extends Executor {

    ...  
    boolean isShutdown();
    
    boolean isTerminated();
    
    boolean awaitTermination(
        long timeout,
        TimeUnit unit) ...;

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

See [www.baeldung.com/java-executor-service-tutorial](http://www.baeldung.com/java-executor-service-tutorial)
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

```java
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)](en.wikipedia.org/wiki/Barrier_(computer_science))
End of Key Methods in the Java ExecutorService (Part 2)