Key Methods in the Java ExecutorService (Part 2)

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

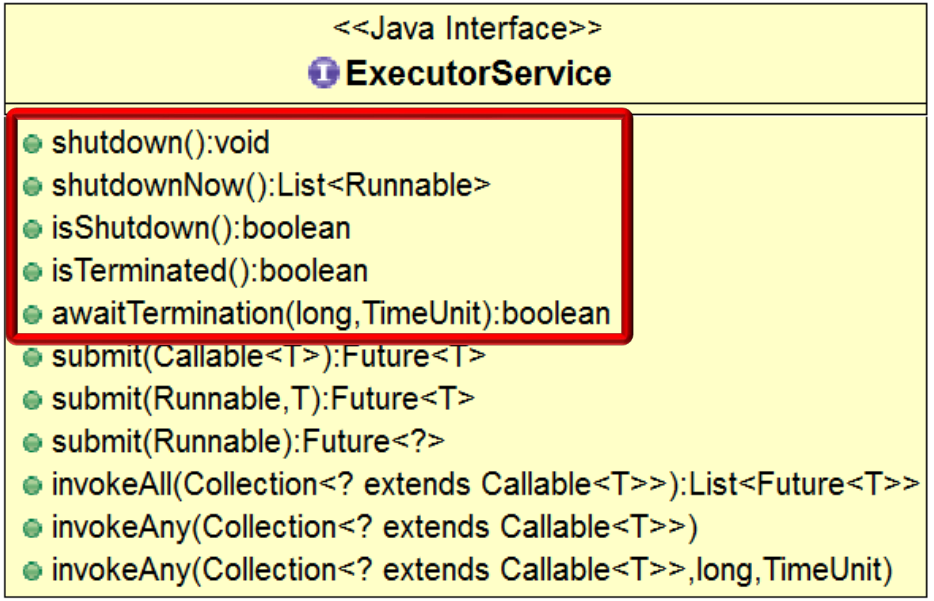
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

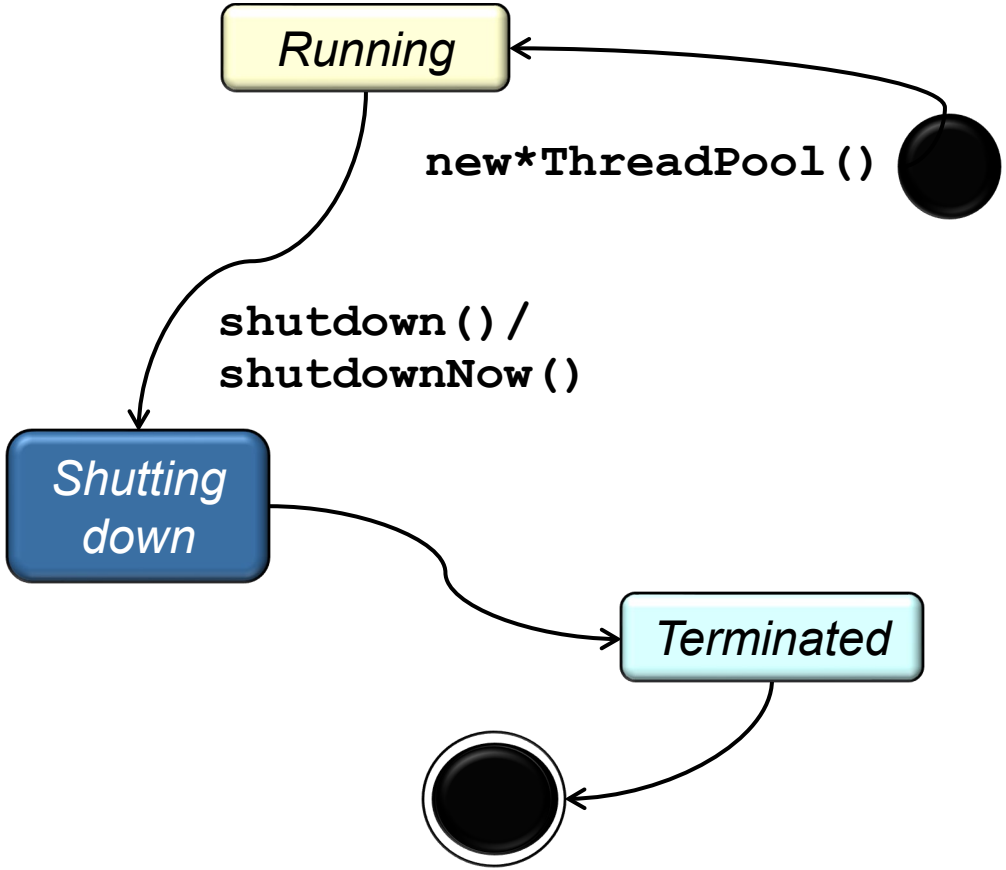
- Recognize the powerful features defined in the Java ExecutorService interface
- Understand other interfaces related to ExecutorService
- Know the key methods provided by ExecutorService
  - These methods submit 1+ tasks for asynchronous execution
  - These methods also manage the lifecycle of tasks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



Key Methods in the ExecutorService Interface: Lifecycle Management

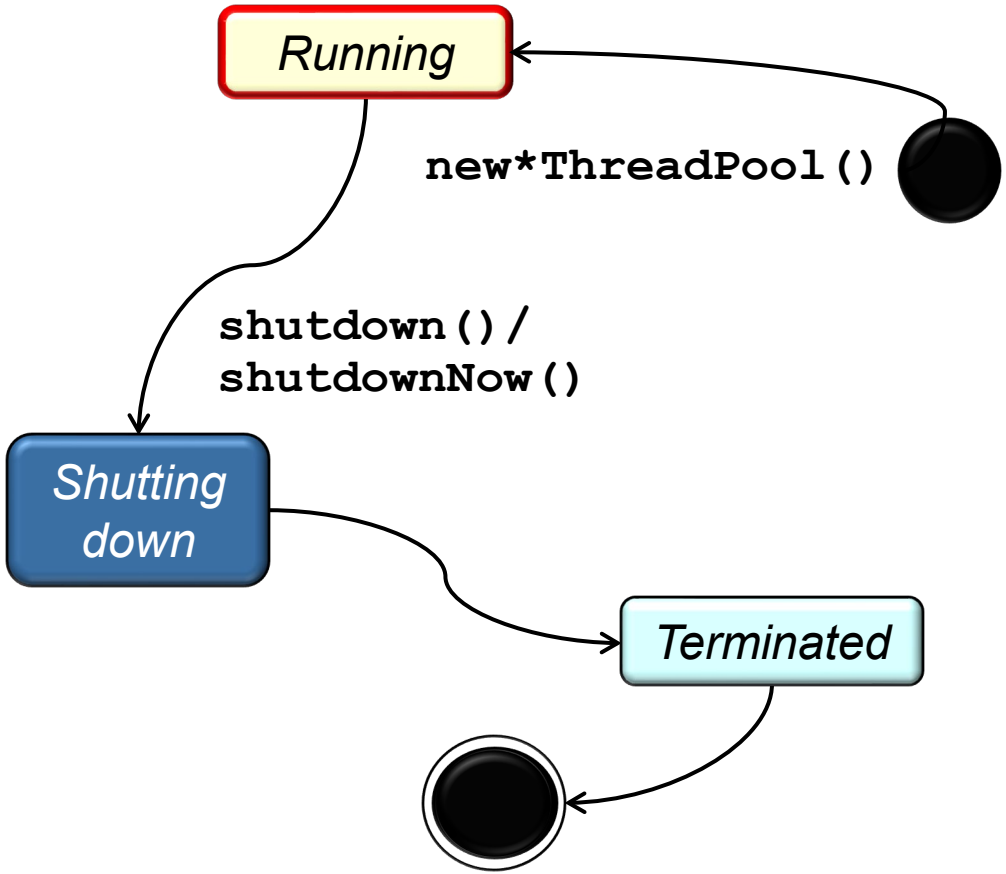
Key Methods in the ExecutorService Interface

- An ExecutorService instance can be in one of three states



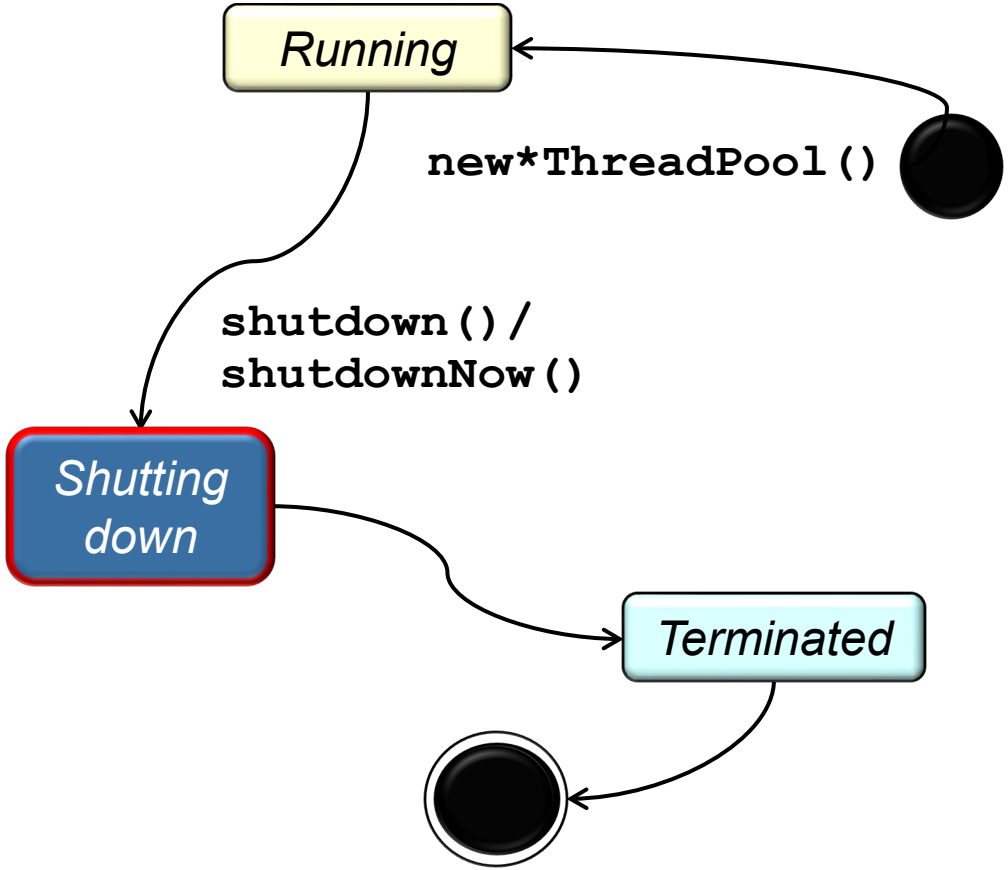
Key Methods in the ExecutorService Interface

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



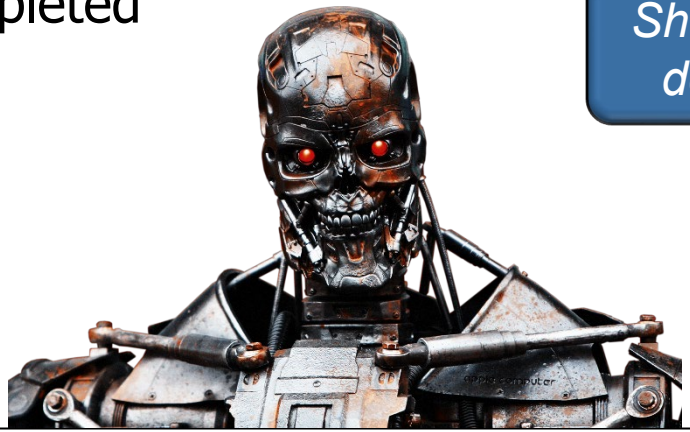
Key Methods in the ExecutorService Interface

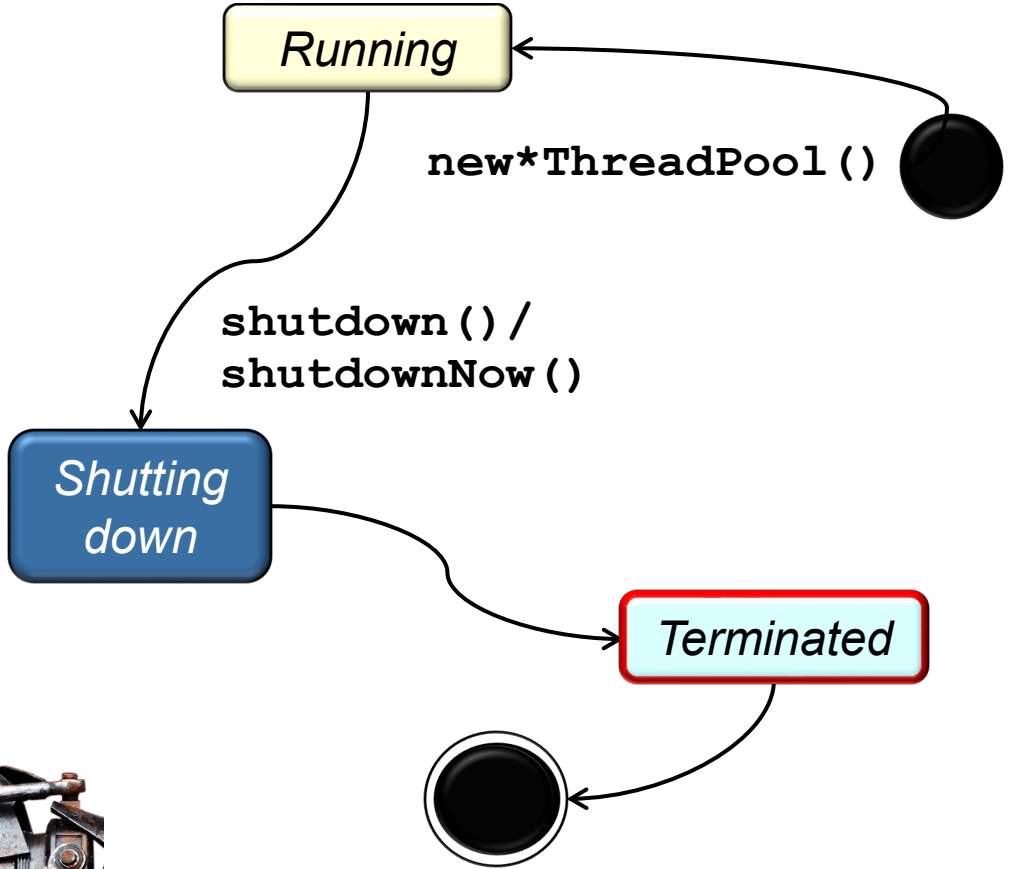
- An ExecutorService instance can be in one of three states
  - Running
  - Shutting down
    - After being shut down gracefully or abruptly



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();
  ...
```

See 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)

```
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

Key Methods in the ExecutorService Interface

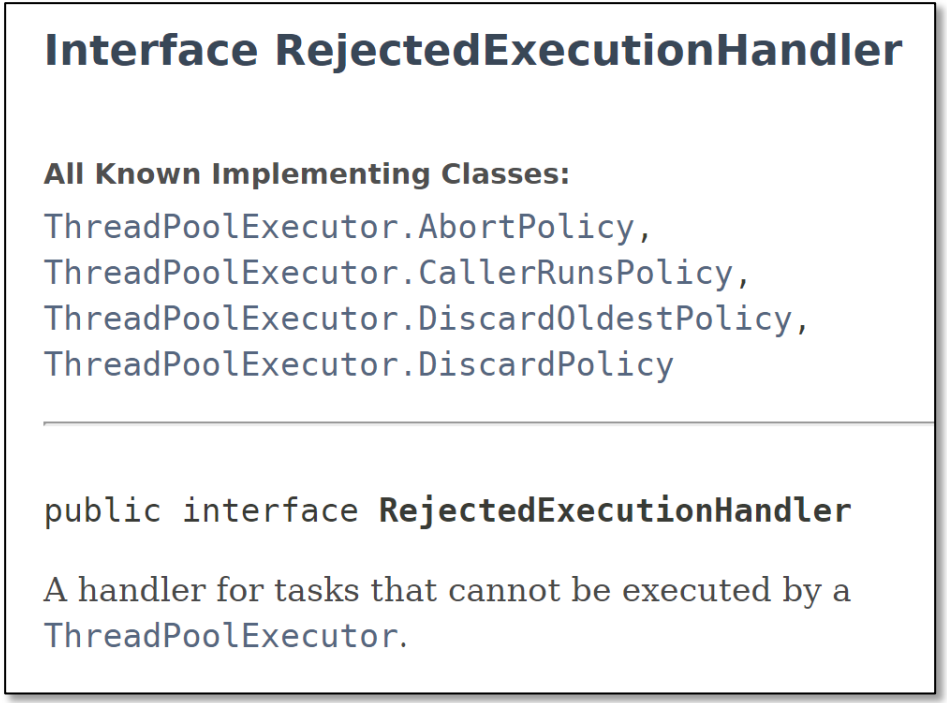
- An ExecutorService client can initiate shutdown operations to manage its lifecycle
  - Performs "graceful shutdown" that completes active tasks
  - Performs "abrupt shutdown" that cancels active tasks & doesn't process waiting tasks
    - Active tasks are cancelled by posting an interrupt request to 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`.

See 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.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

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()

See 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


```
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)

End of Key Methods in the Java ExecutorService (Part 2)