The Java ExecutorService
(Part 1)

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

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

- Recognize the more powerful features provided by the Java ExecutorService interface & its related interfaces/classes

**Interface ExecutorService**

**All Superinterfaces:**
- Executor

**All Known Subinterfaces:**
- ScheduledExecutorService

**All Known Implementing Classes:**
- AbstractExecutorService, ForkJoinPool, ScheduledThreadPoolExecutor, ThreadPoolExecutor

```java
public interface ExecutorService
extends Executor

An Executor that provides methods to manage termination and methods that can produce a Future for tracking progress of one or more asynchronous tasks.

An ExecutorService can be shut down, which will cause it to reject new tasks. Two different methods are provided for shutting down an ExecutorService. The shutdown() method will allow previously submitted tasks to execute before terminating, while the shutdownNow() method prevents waiting tasks from starting and attempts to stop currently executing tasks. Upon termination, an executor has no tasks actively executing, no tasks awaiting execution, and no new tasks can be submitted. An unused ExecutorService should be shut down to allow reclamation of its resources.
```
Overview of the ExecutorService Interface
Overview of the ExecutorService Interface

- Extends Executor to submit tasks, produce futures for asynchronous tasks, & manage termination of tasks & worker threads in pools

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html
Overview of the ExecutorService Interface

- The ExecutorService is used with other interfaces
Overview of the ExecutorService Interface

- The ExecutorService is used with other interfaces, e.g.
  - **Runnable**
    - A “one-way” task that does not return a result

See [docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html](docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html)
Overview of the ExecutorService Interface

• The ExecutorService is used with other interfaces, e.g.
  • Runnable
  • **Callable**
    • A “two-way” task that returns a result

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/Callable.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/Callable.html)
Overview of the ExecutorService Interface

- The ExecutorService is used with other interfaces, e.g.
  - **Runnable**
  - **Callable**
    - A “two-way” task that returns a result
    - Typically used to run asynchronous tasks
Overview of the ExecutorService Interface

- The ExecutorService is used with other interfaces, e.g.
  - Runnable
  - Callable
    - A “two-way” task that returns a result
    - Typically used to run asynchronous tasks
  - Implements the Active Object pattern
    - Decouples method execution from method invocation for objects that each reside in their own thread(s) of control

See en.wikipedia.org/wiki/Active_object
Overview of the ExecutorService Interface

- The ExecutorService is used with other interfaces, e.g.
  - Runnable
  - Callable
  - **Future**
    - Represents the result of an asynchronous two-way task

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html)
Overview of the ExecutorService Interface

- The ExecutorService is used with other interfaces, e.g.
  - Runnable
  - Callable
  - Future
    - Represents the result of an asynchronous two-way task
    - Can be canceled & tested to see if task is done

<<Java Interface>>

Future<V>

- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long, TimeUnit)
Overview of the ExecutorService Interface

• The ExecutorService is used with other interfaces, e.g.
  • Runnable
  • Callable

• Future
  • Represents the result of an asynchronous two-way task
    • Can be canceled & tested to see if task is done
  • Used to implement the Active Object pattern when used with a callable

See en.wikipedia.org/wiki/Active_object
Overview of the ExecutorService Interface

- The ExecutorService is used with other interfaces, e.g.
  - Runnable
  - Callable
  - Future
    - Represents the result of an asynchronous two-way task
      - Can be canceled & tested to see if task is done
      - Used to implement the Active Object pattern when used with a callable
    - Other Future variants implement ExecutorService
Overview of the ExecutorService Interface

• The ExecutorService is used with other interfaces, e.g.
  • Runnable
  • Callable
  • Future
    • Represents the result of an asynchronous two-way task
      • Can be canceled & tested to see if task is done
      • Used to implement the Active Object pattern when used with a callable
    • Other Future variants implement ExecutorService
      • FutureTask
        • Conveys the result from the thread executing a computation to the thread(s) retrieving the result

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/FutureTask.html
Overview of the ExecutorService Interface

• The ExecutorService is used with other interfaces, e.g.
  • Runnable
  • Callable
  • Future
    • Represents the result of an asynchronous two-way task
      • Can be canceled & tested to see if task is done
      • Used to implement the Active Object pattern when used with a callable
    • Other Future variants implement ExecutorService
      • FutureTask
      • RunnableTask
        • Successful execution of the run() method causes completion of the future & allows access to its results

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/RunnableFuture.html
Overview of the ExecutorService Interface

• The ExecutorService is used with other interfaces, e.g.
  • Runnable
  • Callable
  • Future
  • CompletableFuture
    • A Future variant that supports dependent functions & actions that trigger upon its completion

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html)
Overview of the ExecutorService Interface

- The ExecutorService is used with other interfaces, e.g.
  - Runnable
  - Callable
  - Future
  - CompletableFuture
    - A Future variant that supports dependent functions & actions that trigger upon its completion

CompletableFuture isn’t strictly part of the Java Executor framework, but they can be used together, so we’ll cover it in a later lesson.
Overview of the ExecutorService Interface

- ExecutorService also forms the basis for key Java Executor framework subclasses

See libcore/luni/src/main/java/java/util/concurrent/
End of Overview of Java ExecutorService (Part 1)
The Java ExecutorService

(Part 2)

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

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

- Recognize the more powerful features provided by the Java ExecutorService interface & its related interfaces/classes
- Know the key methods provided by the Java ExecutorService
Key Methods in the ExecutorService Interface (Part 1)
Key Methods in the ExecutorService Interface

- ExecutorService can execute individual tasks

```java
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

```java
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 asynchronous tasks

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

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

6
Key Methods in the ExecutorService Interface

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

```java
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 asynchronous tasks
  - Supports “synchronous future” processing model
  - `Future.get()` blocks until task completes successfully

```java
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 asynchronous tasks
    • Supports “synchronous future” processing model
    • `Future.get()` blocks until task completes successfully
      • After which point `get()` returns the task’s result

```java
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 also execute groups of tasks

```java
public interface ExecutorService
    extends Executor {
    ...
    <T> List<Future<T>> invokeAll
        (Collection<? extends Callable<T>> tasks) ...;

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

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

Don’t modify collection while the invokeAll() or invokeAny() methods are in progress!!!
Key Methods in the ExecutorService Interface

- ExecutorService can also execute groups of tasks
- Returns a list of futures when 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 the futures returned in this list are “done”!

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#invokeAll](docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#invokeAll)
Key Methods in the ExecutorService Interface

- ExecutorService can also execute groups of tasks
- Returns a list of futures when 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) ...;

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

    ...
```

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 tasks complete
- Return the result of *one* successful completion
- Cancel uncompleted tasks

```java
class ExecutorService extends Executor {
    ...
    <T> List<Future<T>> invokeAll
        (Collection<? extends Callable<T>> tasks) ...;

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

    <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 tasks complete
- Return the result of one successful completion
- Cancel uncompleted tasks
- Ignore other completed task results

```
public interface ExecutorService
  extends Executor {

  ...;

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

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

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

  ...
```
Key Methods in the ExecutorService Interface (Part 2)
Key Methods in the ExecutorService Interface

- ExecutorService can initiate shutdown operations

```java
public interface ExecutorService extends Executor {

    ... 
    void shutdown();

    List<Runnable> shutdownNow();
    ...
```
Key Methods in the ExecutorService Interface

- ExecutorService can initiate shutdown operations
- Perform “orderly shutdown” that completes existing tasks

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

• ExecutorService can initiate shutdown operations

  • Perform “orderly shutdown” that completes existing tasks
    • But ignores new ones

```java
public interface ExecutorService extends Executor {

  void shutdown();

  List<Runnable> shutdownNow();

  ...
```

18
Key Methods in the ExecutorService Interface

- ExecutorService can initiate shutdown operations
- Perform “orderly shutdown” that completes existing tasks
- Attempt to stop active tasks & don’t process waiting tasks

```java
public interface ExecutorService extends Executor {
    ...
    void shutdown();

    List<Runnable> shutdownNow();
    ...
}
```
Key Methods in the ExecutorService Interface

- ExecutorService can initiate shutdown operations
  - Perform “orderly shutdown” that completes existing tasks
  - Attempt to stop active tasks & don’t process waiting tasks
  - Returns waiting tasks

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

- ExecutorService can query shutdown status & wait for termination to finish

```java
public interface ExecutorService
    extends Executor {

    ... 
    boolean isShutdown();

    boolean isTerminated();

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

Key Methods in the ExecutorService Interface

- ExecutorService can query shutdown status & wait for termination to finish
- True if Executor shut down

```java
public interface ExecutorService extends Executor {
    ...
    boolean isShutdown();
    boolean isTerminated();
    boolean awaitTermination
        (long timeout,
         TimeUnit unit) ...;
}
```
Key Methods in the ExecutorService Interface

- ExecutorService can query shutdown status & wait for termination to finish
  - True if Executor shut down
  - True if all tasks completed after shut down

```java
public interface ExecutorService extends Executor {
    ...
    boolean isShutdown();

    boolean isTerminated();

    boolean awaitTermination(
        long timeout,
        TimeUnit unit)
    ...;
```
Key Methods in the ExecutorService Interface

- ExecutorService can query shutdown status & wait for termination to finish
  - True if Executor shut down
  - True if all tasks completed after shutdown
  - Blocks until all tasks complete

```java
public interface ExecutorService extends Executor {
    ... 
    boolean isShutdown();

    boolean isTerminated();

    boolean awaitTermination
        (long timeout, TimeUnit unit) ...;
}
```
shutdown() & awaitTermination() can be used together to provide “exit barrier” synchronization
End of Overview of Java ExecutorService (Part 2)