Java ExecutorService:

Key Methods

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

```java
<<Java Interface>>
ExecutorService

- shutdown():void
- shutdownNow():List<Runnable>
- isShutdown():boolean
- isTerminated():boolean
- awaitTermination(long, TimeUnit):boolean
- submit(Callable<T>):Future<T>
- submit(Runnable,T):Future<T>
- submit(Runnable):Future<?>
- invokeAll(Collection<? extends Callable<T>>):List<Future<T>>
- invokeAny(Collection<? extends Callable<T>>)
- invokeAny(Collection<? extends Callable<T>>,long,TimeUnit)
```
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
<<Java Interface>>

ExecutorService

- shutdown():void
- shutdownNow():List<Runnable>
- isShutdown():boolean
- isTerminated():boolean
- awaitTermination(long, TimeUnit):boolean
- submit(Callable<T>):Future<T>
- submit(Runnable,T):Future<T>
- submit(Runnable):Future<?>
- invokeAll(Collection<? extends Callable<T>>):List<Future<T>>
- invokeAny(Collection<? extends Callable<T>>)
- invokeAny(Collection<? extends Callable<T>>,long, TimeUnit)
```
Key Methods in the ExecutorService Interface:
Task Execution
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);
    ...
}
```

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

```java
define 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

```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 async tasks that return a value via a future
  - Supports the “synchronous future” processing model
  - `Future.get()` can block 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 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

```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 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

```java
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

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

```java
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

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

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

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#invokeAll](http://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 all tasks complete

```java
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

```java
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

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,
                        long timeout, TimeUnit unit);

    ...;

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

    ...;

    ...

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

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

Key Methods in the ExecutorService Interface
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

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.

```java
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)
    ...
    ...
```
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

• ExecutorService can also execute groups of tasks
• Returns a list of futures when all tasks complete
• Return the result of one successful completion
Key Methods in the ExecutorService Interface:
Lifecycle Management
An ExecutorService instance can be in one of three states:

- **Running**
- **Shutting down**
- **Terminated**

Key Methods in the ExecutorService Interface:

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

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

2. **Shutting down**
   - `shutdown() / shutdownNow()`

3. **Terminated**

Key Methods in the ExecutorService Interface:
- `newThreadPool()`
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

```
new*ThreadPool()

shutdown() / shutdownNow()
```
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:
- `new*ThreadPool()`
- `shutdown()` / `shutdownNow()`
• An ExecutorService client can initiate shutdown operations to manage its lifecycle

```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

```java
class 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.

But ignores new tasks & doesn’t process waiting tasks.

**Public Interface**

```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

```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.
- Active tasks are cancelled by posting an interrupt request to executor thread(s).

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

See [docs.oracle.com/javase/tutorial/essential/concurrency/interrupt.html](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
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

```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
- Tasks submitted after an Executor Service is shut down are dealt with by RejectedExecutionHandler

### 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 RejectedExecutionHandler
  - Can silently discard task or throw RejectedExecutionException

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

```java
public interface ExecutorService
    extends Executor {

    // Key methods for querying the shutdown status
    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

```
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

```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)
End of Java Executor Service: Key Methods