Key Methods in the Java ExecutorService
(Part 1)

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

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shutdown()</td>
<td>void</td>
</tr>
<tr>
<td>shutdownNow()</td>
<td>List&lt;Runnable&gt;</td>
</tr>
<tr>
<td>isShutdown()</td>
<td>boolean</td>
</tr>
<tr>
<td>isTerminated()</td>
<td>boolean</td>
</tr>
<tr>
<td>awaitTermination(long, TimeUnit)</td>
<td>boolean</td>
</tr>
<tr>
<td>submit(Callable&lt;T&gt;)</td>
<td>Future&lt;T&gt;</td>
</tr>
<tr>
<td>submit(Runnable)</td>
<td>Future&lt;?&gt;</td>
</tr>
<tr>
<td>invokeAll(Collection&lt;? extends Callable&lt;T&gt;&gt;&gt;)</td>
<td>List&lt;Future&lt;T&gt;&gt;</td>
</tr>
<tr>
<td>invokeAny(Collection&lt;? extends Callable&lt;T&gt;&gt;))</td>
<td></td>
</tr>
<tr>
<td>invokeAny(Collection&lt;? extends Callable&lt;T&gt;&gt;, long, TimeUnit)</td>
<td></td>
</tr>
</tbody>
</table>
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
Key Methods in the ExecutorService Interface: Task Execution
Key Methods in the ExecutorService Interface

- ExecutorService executes 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 executes tasks
- `execute()` runs one-way tasks whose `run()` hook method returns `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 executes tasks
  - `execute()` runs one-way tasks whose `run()` hook method returns `void`
- submit() runs two-way async tasks whose `call()` hook method returns a value

```java
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 executes tasks
  - `execute()` runs one-way tasks whose `run()` hook method returns void
- submit() runs two-way async tasks whose `call()` hook method returns a value
  - This value can be obtained via a future

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

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

    ...}
```

<<Java Interface>>

`Future<V>`

- `cancel(boolean):boolean`
- `isCancelled():boolean`
- `isDone():boolean`
- `get()`
- `get(long, TimeUnit)`
Key Methods in the ExecutorService Interface

- ExecutorService executes tasks
  - `execute()` runs one-way tasks whose `run()` hook method returns `void`
- submit() runs two-way async tasks whose `call()` hook method returns a value
  - This value can be obtained 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 executes tasks
  - `execute()` runs one-way tasks whose `run()` hook method returns `void`
- submit() runs two-way async tasks whose `call()` hook method returns a value
  - This value can be obtained via a future
- Future.get() can block until task completes successfully

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

    <T> Future<T> submit (Callable<T> task);
    ...
}
```
• ExecutorService executes tasks
  • `execute()` runs one-way tasks whose `run()` hook method returns `void`
• `submit()` runs two-way async tasks whose `call()` hook method returns a value
  • This value can be obtained via a future
• `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 executes tasks
  - `execute()` runs one-way tasks whose `run()` hook method returns `void`
  - `submit()` runs two-way async tasks whose `call()` hook method returns a value
  - `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` executes tasks
  - `execute()` runs one-way tasks whose `run()` hook method returns void
- `submit()` runs two-way async tasks whose `call()` hook method returns a value
  - `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
public interface ExecutorService extends Executor {
    ...
    <T> List<Future<T>> invokeAll
        (Collection<? extends Callable<T>> tasks) ...;

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

ExecutiveService
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

```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](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

```java
public interface ExecutorService
    extends Executor {

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

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

    ...
```

Ignore
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...
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

```java
public interface ExecutorService
    extends Executor {

    // ... (omitted for brevity)

    <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)
    ...

    // ... (omitted for brevity)
```
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

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

```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)
    ...;

    ...;

    TimeoutException is thrown if timeout elapses
```
End of Key Methods in the Java ExecutorService (Part 1)