Java ExecutorService Related Interfaces

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

```
<<Java Interface>>
Future<V>
- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long, TimeUnit)

<<Java Interface>>
Callable<V>
- call()

<<Java Interface>>
Runnable
- run():void
```

```
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.
```

Interface ExecutorService

All Superinterfaces:
Executor

All Known Subinterfaces:
ScheduledExecutorService

All Known Implementing Classes:
AbstractExecutorService, ForkJoinPool, ScheduledThreadPoolExecutor, ThreadPoolExecutor
Overview of Interfaces Related To ExecutorService
Overview of Interfaces Related to ExecutorService

- ExecutorService uses several other interfaces to manage task lifecycles

```
<<Java Interface>>
Runnable
run():void

<<Java Interface>>
Future<V>
cancel(boolean):boolean
isCancelled():boolean
isDone():boolean
get()
get(long, TimeUnit)

<<Java Interface>>
Callable<V>
call()
```
Overview of Interfaces Related to ExecutorService

- ExecutorService uses two interfaces to define tasks

```
<<Java Interface>>
Runnable
- run(): void

<<Java Interface>>
Callable<V>
- call()
```
Overview of Interfaces Related to ExecutorService

ExecutorService uses two interfaces to define tasks, 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 Interfaces Related to ExecutorService

- ExecutorService uses two interfaces to define tasks, 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

Callable is also a functional interface
Overview of Interfaces Related to ExecutorService

- ExecutorService uses two interfaces to define tasks, e.g.
  - Runnable
  - Callable
    - A “two-way” task that returns a result
    - Typically used to run two-way async tasks
Overview of Interfaces Related to ExecutorService

- ExecutorService uses two interfaces to define tasks, e.g.
  -Runnable
  -Callable
  -A “two-way” task that returns a result
  -Typically used to run two-way async tasks
  -Implements the Active Object pattern

Decouples the thread that invokes a method from the thread that executes the method

See en.wikipedia.org/wiki/Active_object
Overview of Interfaces Related to ExecutorService

- The **Future** interface is used by ExecutorService to represent & control a task’s lifecycle

```
<<Java Interface>>

Future<V>

- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long, TimeUnit)
```

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 Interfaces Related to ExecutorService

1. Asynchronous computation

2. Result obtained only after the computation completes

See en.wikipedia.org/wiki/Futures_and_promises
Overview of Interfaces Related to ExecutorService

- A Java future defines a proxy to represent & control the result of an async computation.

```java
Result get_result ()
begin
  ## Suspend calling thread until result is available.
  if (result == NULL) then
    thread.wait ();
  return result;
end
```

Table tent #'s are a human-known-use of futures!

See www.citygrafx.com/table-numbers-table-markers
Overview of Interfaces Related to ExecutorService

- A Java future defines a proxy to represent & control the result of an async computation.

```java
Result get_result ()
begin
  ## Suspend calling thread until result is available.
  if (result == NULL) then
    thread.wait ();
  return result;
end
```

Table tent #'s are a human-known-use of futures!

- e.g., McDonald’s vs Wendy’s model of preparing fast food
Overview of Interfaces Related to ExecutorService

- Programs use the **Future** interface to represent & control a task’s lifecycle via its five methods

```
Callable callable
Callable callable
Callable callable
WorkQueue
3.offer()
4.take()
5.run()
submit()
ThreadPoolExecutor

Fixed WorkerThreads

Future BigFraction

ResultType result = future.get();

Thread (main thread)

<<Java Interface>>

Future<V>

cancel(boolean): boolean
isCancelled(): boolean
isDone(): boolean
get()
get(long, TimeUnit)
```
Overview of Interfaces Related to ExecutorService

- Programs use the **Future** interface to represent & control a task’s lifecycle via its five methods, e.g.
- Can be used to retrieve a two-way task’s result

```
ResultType result = future.get();
```

```
Future<
  
  cancel(boolean):boolean
  
  isCancelled():boolean
  
  isDone():boolean
  
  get():
  
  get(long, TimeUnit)
```
Overview of Interfaces Related to ExecutorService

- Programs use the **Future** interface to represent & control a task’s lifecycle via its five methods, e.g.
  - Can be used to retrieve a two-way task’s result
  - Can be tested for completion

```
<<Java Interface>>
Future<V>
- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long, TimeUnit)
```

Diagram:
- ThreadPoolExecutor
- WorkQueue
- Callable
- Future
- BigFraction
- Callable
- Thread (main thread)
- Submit
- Take
- Run
- IsDone

```
1. submit(task) Callable
2. Return future
3. offer() submit() Callable
4. take() Callable
5. run() Callable
       Fixed WorkerThreads
       run()
6. isDone() Future
```

16
Overview of Interfaces Related to ExecutorService

- Programs use the **Future** interface to represent & control a task’s lifecycle via its five methods, e.g.
  - Can be used to retrieve a two-way task’s result
  - Can be tested for completion
  - Can be tested for cancellation & cancelled

```java
<<Java Interface>>

*Future<V>*
- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long, TimeUnit)
```

```
1. submit(task)
2. Return future
3. offer()
4. take()
5. run()
6. cancel()
```

![Diagram of ThreadPoolExecutor with WorkQueue, callable, Fixed WorkerThreads, submit(), run(), take(), cancel() methods, and BigFraction example.](image)
Overview of Interfaces Related to ExecutorService

- ExecutorService.submit() can initiate an async computation in Java.

```java
ExecutorService executorService = Executors.newFixedThreadPool(sMAX_THREADS);
```

See next part of this lesson on “Java ExecutorService: Key Methods”
Overview of Interfaces Related to ExecutorService

- ExecutorService.submit() can initiate an async computation in Java.

```java
Callable<BigFraction> task = () -> {
    BigFraction bf1 = new BigFraction(f1);
    BigFraction bf2 = new BigFraction(f2);
    return bf1.multiply(bf2);
};
Future<BigFraction> future = mExecutorService.submit(task);
```
Overview of Interfaces Related to ExecutorService

- ExecutorService.submit() can initiate an async computation in Java.
- It returns a Future implemented as a FutureTask

```
Callable<BigFraction> task = () -> {
    BigFraction bf1 = new BigFraction(f1);
    BigFraction bf2 = new BigFraction(f2);
    return bf1.multiply(bf2);
};

Future<BigFraction> future = mExecutorService.submit(task);
```

See upcoming lesson on “Java FutureTask”
Overview of Interfaces Related to ExecutorService

- ExecutorService.submit() can initiate an async computation in Java.
- It returns a Future implemented as a FutureTask.
- Async computation runs in a worker thread.

Callable<BigFraction> task = () -> {
    BigFraction bf1 = new BigFraction(f1);
    BigFraction bf2 = new BigFraction(f2);
    return bf1.multiply(bf2);
};

Future<BigFraction> future = mExecutorService.submit(task);

See upcoming part of this lesson on “Overview of Java ThreadPoolExecutor”
Overview of Interfaces Related to ExecutorService

• When the async computation completes the future is triggered & the result is available

BigFraction result = future.get();

1. submit (task)
2. Return future
3. offer()
4. take()
5. run()

ThreadPoolExecutor

ExecutorService

Callable
Callable
Callable
Callable

WorkQueue

submit()

6. get()
Overview of Interfaces Related to ExecutorService

- When the async computation completes the future is triggered & the result is available
- `get()` can block or (timed-)poll

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html#get
Overview of Interfaces Related to ExecutorService

- When the async computation completes the future is triggered & the result is available
  - `get()` can block or (timed-)poll
  - Results can occur in a different order than the original calls were made
Overview of Interfaces Related to ExecutorService

- Futures are applied in the **Active Object** pattern

---

**Client thread**

![Diagram of Client thread and Future]

**Service thread**

![Diagram of Service thread and Future]

**Future**

- **get_result**
- **write_result**

---

1. Async computation runs in a pool thread

2. Client uses future to obtain a result only after async computation completes

---

See [en.wikipedia.org/wiki/Active_object](en.wikipedia.org/wiki/Active_object)
Other variants of Future are applied by implementations of the ExecutorService.
Other variants of Future are applied by implementations of the ExecutorService, e.g.

**RunnableFuture**
- Execution of run() method will complete the future & allow access to its results

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/RunnableFuture.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/RunnableFuture.html)
Overview of Interfaces Related to ExecutorService

- Other variants of Future are applied by implementations of the ExecutorService, e.g.
  - RunnableFuture
  - FutureTask
    - Conveys result from thread running a task to thread(s) retrieving result

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/FutureTask.html
Overview of Interfaces Related to ExecutorService

- A CompletableFuture is an implementation of Future that supports dependent actions triggered upon an async operation 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)

CompletableFuture isn’t part of the Java Executor framework
Overview of Interfaces Related to ExecutorService

- A CompletableFuture is an implementation of Future that supports dependent actions triggered upon an async operation completion
- This topic is covered elsewhere

See [www.dre.vanderbilt.edu/~schmidt/DigitalLearning](http://www.dre.vanderbilt.edu/~schmidt/DigitalLearning)
Overview of Interfaces Related to ExecutorService

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

See src/share/classes/java/util/concurrent
Overview of Interfaces Related to ExecutorService

- ExecutorService also forms the basis for key Java Executor framework subclasses
- We cover these subclasses later in this course
End of Java Executor Service Related Interfaces