Java FutureTask: Introduction

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

• Understand how Java FutureTask provides a cancellable asynchronous computation that implements the Future & Runnable interfaces

```
Class FutureTask<V>

java.lang.Object
java.util.concurrent.FutureTask<V>

Type Parameters:
V - The result type returned by this FutureTask's get methods

All Implemented Interfaces:
Runnable, Future<V>, RunnableFuture<V>

public class FutureTask<V>
extends Object
implements RunnableFuture<V>

A cancellable asynchronous computation. This class provides a base implementation of Future, with methods to start and cancel a computation, query to see if the computation is complete, and retrieve the result of the computation. The result can only be retrieved when the computation has completed; the get methods will block if the computation has not yet completed. Once the computation has completed, the computation cannot be restarted or cancelled (unless the computation is invoked using runAndReset()).

A FutureTask can be used to wrap a Callable or Runnable object. Because FutureTask implements Runnable, a FutureTask can be submitted to an Executor for execution.
```
Overview of Java FutureTask
Java FutureTask conveys the result from a thread running an asynchronous computation to thread(s) that want to process the result.

**Overview of Java FutureTask**

**Class FutureTask<V>**

```java
java.lang.Object
java.util.concurrent.FutureTask<V>
```

**Type Parameters:**

- V - The result type returned by this FutureTask's get methods

**All Implemented Interfaces:**

- Runnable, Future<V>, RunnableFuture<V>

```java
public class FutureTask<V>
extends Object
implements RunnableFuture<V>
```

A cancellable asynchronous computation. This class provides a base implementation of Future, with methods to start and cancel a computation, query to see if the computation is complete, and retrieve the result of the computation. The result can only be retrieved when the computation has completed; the get methods will block if the computation has not yet completed. Once the computation has completed, the computation cannot be restarted or cancelled (unless the computation is invoked using runAndReset()).

A FutureTask can be used to wrap a Callable or Runnable object. Because FutureTask implements Runnable, a FutureTask can be submitted to an Executor for execution.

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/FutureTask.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/FutureTask.html)
Overview of Java FutureTask

- FutureTask implements RunnableFuture & provides several capabilities
Overview of Java FutureTask

- FutureTask implements RunnableFuture & provides several capabilities, e.g.
  - Start & cancel a computation that can run asynchronously

FutureTask computations are often started/run by a Java ThreadPoolExecutor
Overview of Java FutureTask

- FutureTask implements RunnableFuture & provides several capabilities, e.g.
  - Start & cancel a computation that can run asynchronously
  - Query to see if computation completed or was cancelled

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

<<Java Interface>>
RunnableFuture<V>
- run():void

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

<<Java Class>>
FutureTask<V>
- FutureTask(Callable<V>)
- FutureTask(Runnable V)
- isCancelled():boolean
- isDone():boolean
- cancel(boolean):boolean
- get()
- get(long, TimeUnit)
- run():void
```
Overview of Java FutureTask

- FutureTask implements RunnableFuture & provides several capabilities, e.g.
  - Start & cancel a computation that can run asynchronously
  - Query to see if computation completed or was cancelled
  - Get result of computation
End of Java FutureTask: Introduction