The Java FutureTask

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

- Understand how the Java FutureTask conveys the result from the thread executing a computation to thread(s) retrieving the result.

**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.
Learning Objectives in this Part of the Lesson

• Understand how the Java FutureTask conveys the result from the thread executing a computation to thread(s) retrieving the result

• Know how the Memoizer class uses FutureTask to speed up programs by caching results of expensive function calls & returning the cached result when the same inputs occur again
Overview of Java FutureTask
Overview of Java FutureTask

- FutureTask conveys the result from the thread executing a computation to thread(s) retrieving the result.

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>
```

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

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/FutureTask.html
FutureTask implements RunnableFuture & provides several capabilities, e.g.

- Start & cancel a computation that runs asynchronously
Overview of Java FutureTask

- FutureTask implements RunnableFuture & provides several capabilities, e.g.
  - Start & cancel a computation that runs asynchronously
  - Query to see if computation is complete or has been cancelled
Overview of Java FutureTask

- FutureTask implements RunnableFuture & provides several capabilities, e.g.
  - Start & cancel a computation that runs asynchronously
  - Query to see if computation is complete or has been cancelled
  - Retrieve the result of the computation

```java
<<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

- Key methods in FutureTask

```
class FutureTask<V> implements RunnableFuture<V> {
    public FutureTask (Callable<V> callable) {...}

    public void run() {...}

    public boolean isCancelled() {...}

    public boolean isDone() {...}

    public boolean cancel(boolean mayInterruptIfRunning) {...}

    public V get() {...}

    protected void done() {...}
    ...
}
```

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

• Key methods in FutureTask, e.g.
  Constructs a futuretask with a Callable to run later

```java
class FutureTask<V> implements RunnableFuture<V> {
    public FutureTask (Callable<V> callable) {...}

    public void run() {...}

    public boolean isCancelled() {...}

    public boolean isDone() {...}

    public boolean cancel(boolean mayInterruptIfRunning) {...}

    public V get() {...}

    protected void done() {...}
    ...
}
```
Overview of Java FutureTask

- Key methods in FutureTask, e.g.
  - Constructs a futuretask with a Callable to run later
  - Runs the task

```java
class FutureTask<V> implements RunnableFuture<V> {
    public FutureTask(
        Callable<V> callable) {...}

    public void run() {...}

    public boolean isCancelled() {...}

    public boolean isDone() {...}

    public boolean cancel(boolean mayInterruptIfRunning) {...}

    public V get() {...}

    protected void done() {...}
...
```

Sets the futuretask to the result of its computation unless it’s been cancelled
Overview of Java FutureTask

- Key methods in FutureTask, e.g.
  - Constructs a future task with a Callable to run later
  - Runs the task
  - Checks whether the future task has finished normally or was cancelled

```java
class FutureTask<V> implements RunnableFuture<V> {
    public FutureTask(
        Callable<V> callable) {...}

    public void run() {...}

    public boolean isCancelled() {...}

    public boolean isDone() {...}

    public boolean cancel(boolean mayInterruptIfRunning) {...}

    public V get() {...}

    protected void done() {...}

    ...
```
Overview of Java FutureTask

- Key methods in FutureTask, e.g.
  - Constructs a futuretask with a Callable to run later
  - Runs the task
  - Checks whether the futuretask has finished normally or was cancelled
  - Cancels a futuretask

```java
class FutureTask<V> implements RunnableFuture<V> {
    public FutureTask
        (Callable<V> callable) {...}

    public void run() {...}

    public boolean isCancelled() {...}

    public boolean isDone() {...}

    public boolean cancel(boolean mayInterruptIfRunning) {...}

    public V get() {...}

    protected void done() {...}

    ...
```
Overview of Java FutureTask

- Key methods in FutureTask, e.g.
  - Constructs a futuretask with a Callable to run later
  - Runs the task
  - Checks whether the futuretask has finished normally or was cancelled
  - Cancels a futuretask
  - Gets the result

```java
class FutureTask<V> implements RunnableFuture<V> {
    public FutureTask
        (Callable<V> callable) {...}

    public void run() {...}

    public boolean isCancelled() {...}

    public boolean isDone() {...}

    public boolean cancel(boolean mayInterruptIfRunning) {...}

    public V get() {...}

    protected void done() {...}
    ...
}
```

This call blocks until the task is finished its computations – there’s also a timed version
Overview of Java FutureTask

- Key methods in FutureTask, e.g.
  - Constructs a futuretask with a Callable to run later
  - Runs the task
  - Checks whether the futuretask has finished normally or was cancelled
  - Cancels a futuretask
  - Gets the result
  - Hook method called back after run() returns

```java
class FutureTask<V> implements RunnableFuture<V> {
    public FutureTask(Callable<V> callable) {...}

    public void run() {...}

    public boolean isCancelled() {...}

    public boolean isDone() {...}

    public boolean cancel(boolean mayInterruptIfRunning) {...}

    public V get() {...}

    protected void done() {...}

    ...
}
```

Subclasses may override this method to invoke completion callbacks or perform bookkeeping
Applying FutureTask to the Memoizer Class
Applying FutureTask to the Memoizer Class

- Memoization is an optimization technique used to speed up programs by caching the results of expensive function calls & returning the cached result when the same inputs occur again.

See en.wikipedia.org/wiki/Memoization
Applying FutureTask to the Memoizer Class

- This class defines a "memoizing" cache that maps a key to the value produced by a function

This class is based on "Java Concurrency in Practice" by Brian Goetz et al.

See github.com/douglascraigschmidt/POSA/blob/master/ex/M4/Primes/PrimeExecutorCompletionService/app/src/main/java/vandy/mooc/prime/utils/Memoizer.java
Applying FutureTask to the Memoizer Class

- This class defines a "memoizing" cache that maps a key to the value produced by a function.
- If a value has been computed for a key it is returned rather than calling the function to compute it again.
Applying FutureTask to the Memoizer Class

• This class defines a "memoizing" cache that maps a key to the value produced by a function
  • If a value has been computed for a key it is returned rather than calling the function to compute it again
  • A ConcurrentHashMap is used to minimize synchronization overhead

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html
Applying FutureTask to the Memoizer Class

- This class defines a "memoizing" cache that maps a key to the value produced by a function.
  - If a value has been computed for a key it is returned rather than calling the function to compute it again.
  - A ConcurrentHashMap is used to minimize synchronization overhead.
  - FutureTask is used to ensure only one call to the function is run when a key & value is first added to the cache.

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/FutureTask.html
• This class defines a "memoizing" cache that maps a key to the value produced by a function
  • If a value has been computed for a key it is returned rather than calling the function to compute it again
• By extending Function an instance of Memoizer can be used transparently where ever a Function is expected

```java
Function<Long, Long> primeMemoizer = new Memoizer<>
    (PrimeCheckers::bruteForceChecker);

new Random().longs(count, 0, Integer.MAX_VALUE)
    .mapToObj(ranNum ->
        new PrimeCallable(ranNum, primeMemoizer))
... 
```

See [docs.oracle.com/javase/8/docs/api/java/util/function/Function.html](docs.oracle.com/javase/8/docs/api/java/util/function/Function.html)
 Memoizer uses FutureTask to ensure only one call to the function is run when a key & value is first added to the cache

```java
public class Memoizer<K, V> implements Function<K, V> {

    /** Associate a key with a value produced by a function. A Future is used to ensure the function is called once. */
    private final ConcurrentMap<K, Future<V>> mCache =
            new ConcurrentHashMap<>();

    /** This function produces a value based on the key. */
    private final Function<K, V> mF;

    /** Constructor initializes the function field. */
    public Memoizer(Function<K, V> function) {
        mF = function;
    }

    ...
```
Memoizer uses FutureTask to ensure only one call to the function is run when a key & value is first added to the cache.

```java
public class Memoizer<K, V> implements Function<K, V> {
    ...
    public V apply(final K key) {
        // Try to find the key in the cache.
        Future<V> f = mCache.get(key);

        if (f == null) {
            // This FutureTask's run() method computes the value.
            FutureTask<V> ft = new FutureTask<>(() -> mF.apply(key));

            f = mCache.putIfAbsent(key, ft); // Atomically add ft.

            if (f == null) { // If this is the first-time-in then
                f = ft; ft.run(); // compute the function
            }
        }
    }
    ...
}
Memoizer uses FutureTask to ensure only one call to the function is run when a key & value is first added to the cache.

```java
class Memoizer<K, V> implements Function<K, V> {
    ...
    public V apply(final K key) {
        ...
        try {
            // Return the result of the future, which will block
            // if the futureTask hasn't finished running yet.
            return f.get();
        } catch (Exception e) {
            // Try to remove key from the cache if exception occurs.
            mCache.remove(key, f);
            throw e; // Rethrow the exception.
        }
    }
}
```
End of Java FutureTask