Overview of Java FutureTask

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

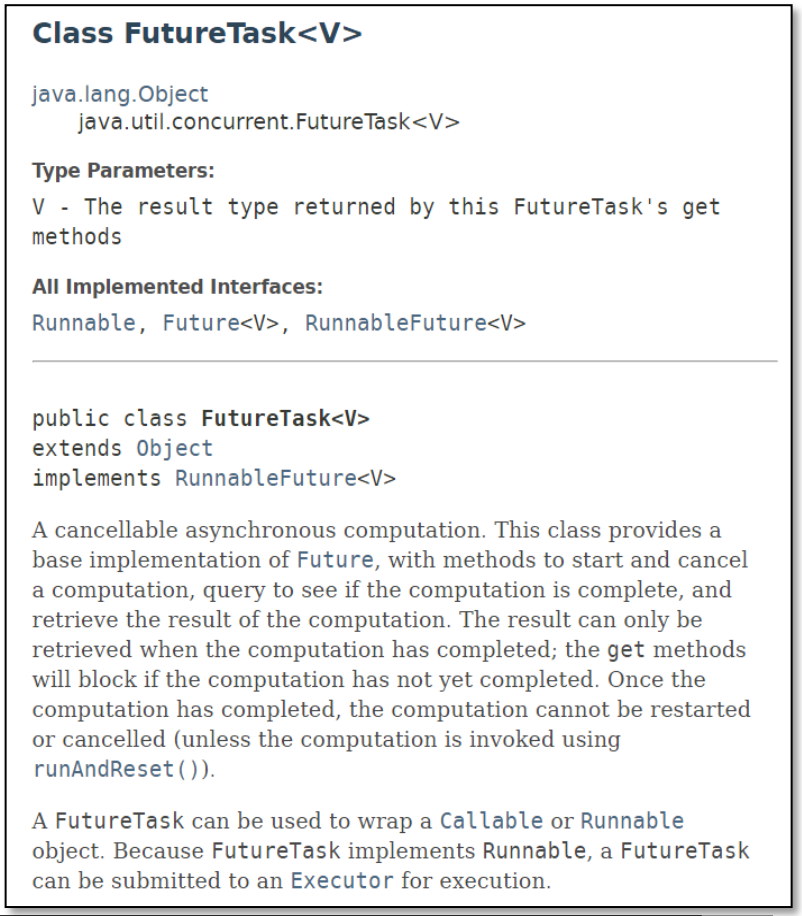
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand the need for the *Future* pattern & Java Future interface
- Recognize the lifecycle of a Future & human known uses of the *Future* pattern
- Know the key methods in the modern Java Future interface
- Learn how to implement a Future via the FutureTask class
  - FutureTask conveys the result from a Thread running an async computation to a Thread that wants to process the result



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()`).

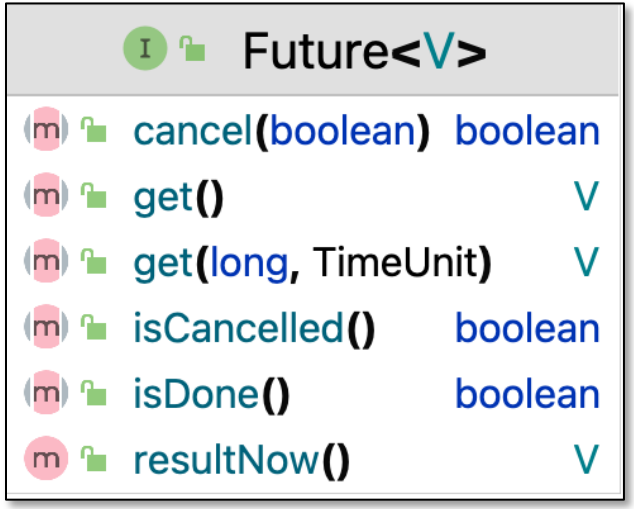
A `FutureTask` can be used to wrap a `Callable` or `Runnable` object. Because `FutureTask` implements `Runnable`, a `FutureTask` can be submitted to an `Executor` for execution.

See javase/20/docs/api/java.base/java/util/concurrent/FutureTask.html

Overview of Java FutureTask

Overview of Java FutureTask

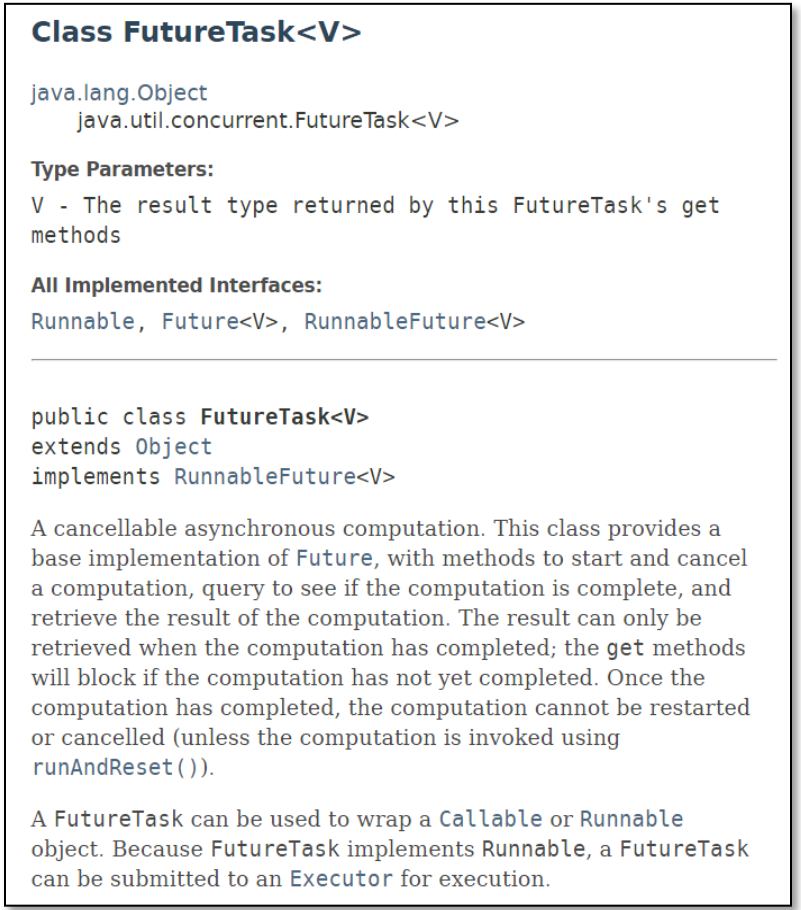
- The Java Future is an interface, so it must be implemented by a class before it can be used



See 20/docs/api/java.base/java/util/concurrent/Future.html

Overview of Java FutureTask

- The Java FutureTask class implements Future (indirectly) & provides a cancelable async computation



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>

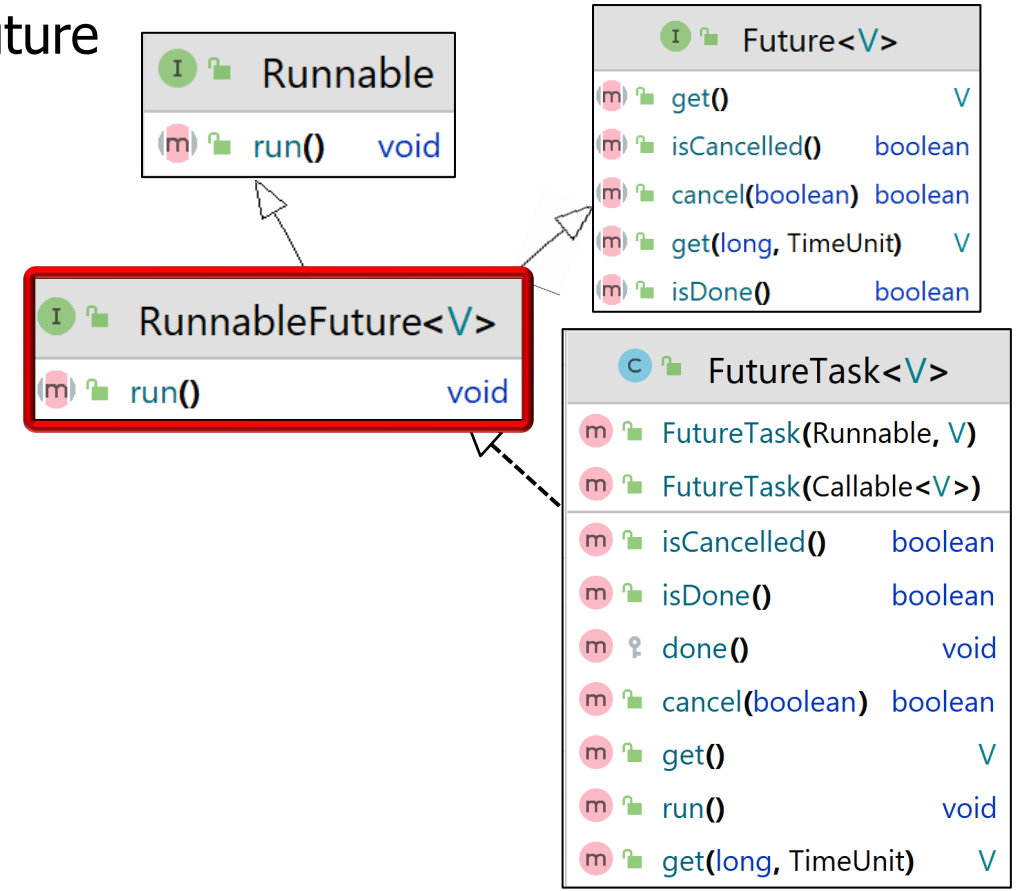
A cancellable asynchronous computation. This class provides a base implementation of Future, with methods to start and cancel a computation, query to see if the computation is complete, and retrieve the result of the computation. The result can only be retrieved when the computation has completed; the 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.

See javase/20/docs/api/java.base/java/util/concurrent/FutureTask.html

Overview of Java FutureTask

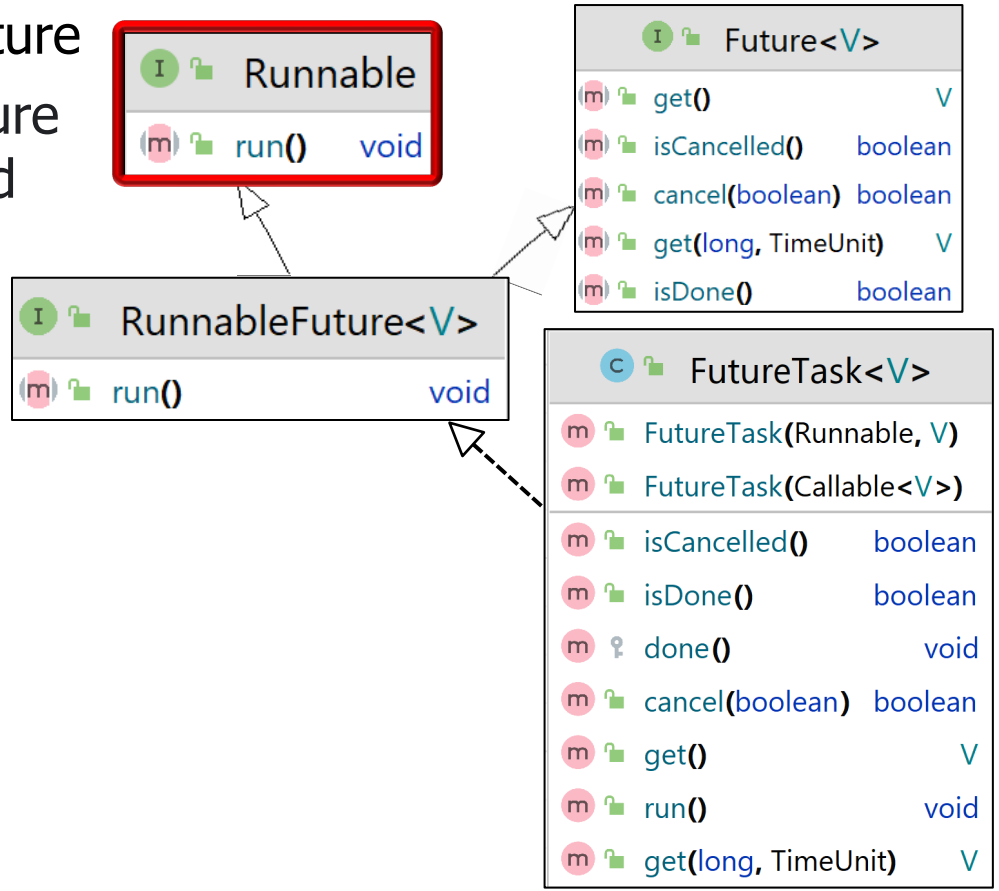
- FutureTask implements RunnableFuture



See javase/20/docs/api/java.base/java/util/concurrent/RunnableFuture.html

Overview of Java FutureTask

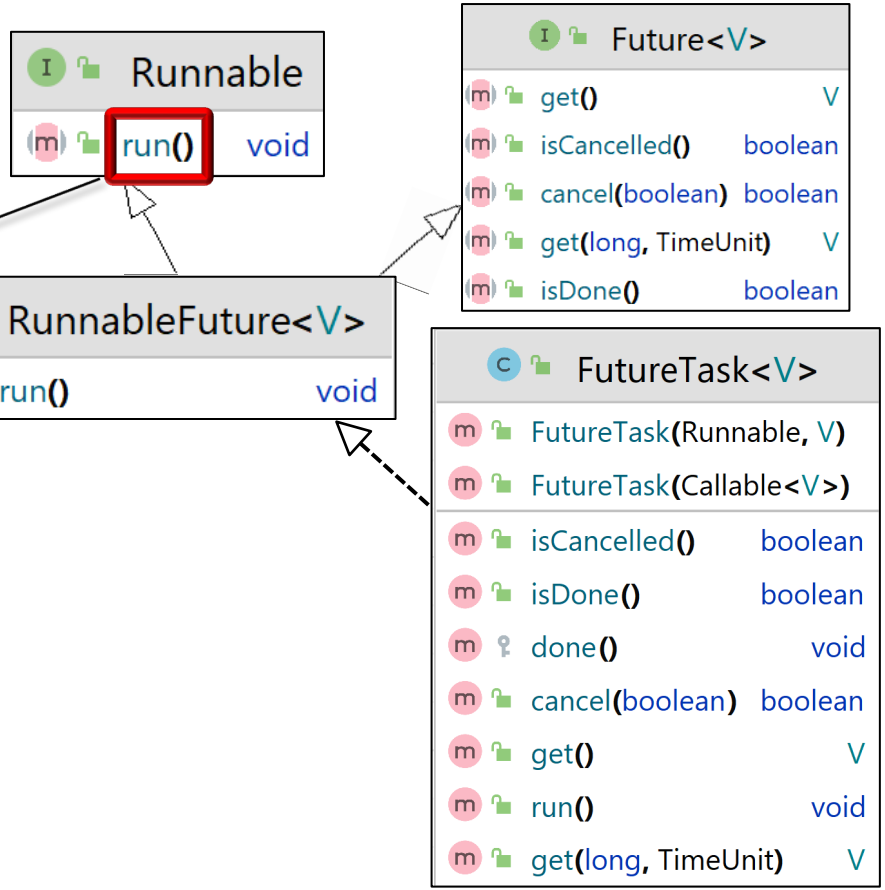
- FutureTask implements RunnableFuture
  - By implementing Runnable, a Future Task can be submitted to a Thread or Executor for execution



See javase/20/docs/api/java.base/java/lang/Runnable.html

Overview of Java FutureTask

- FutureTask implements RunnableFuture
  - By implementing Runnable, a Future Task can be submitted to a Thread or Executor for execution

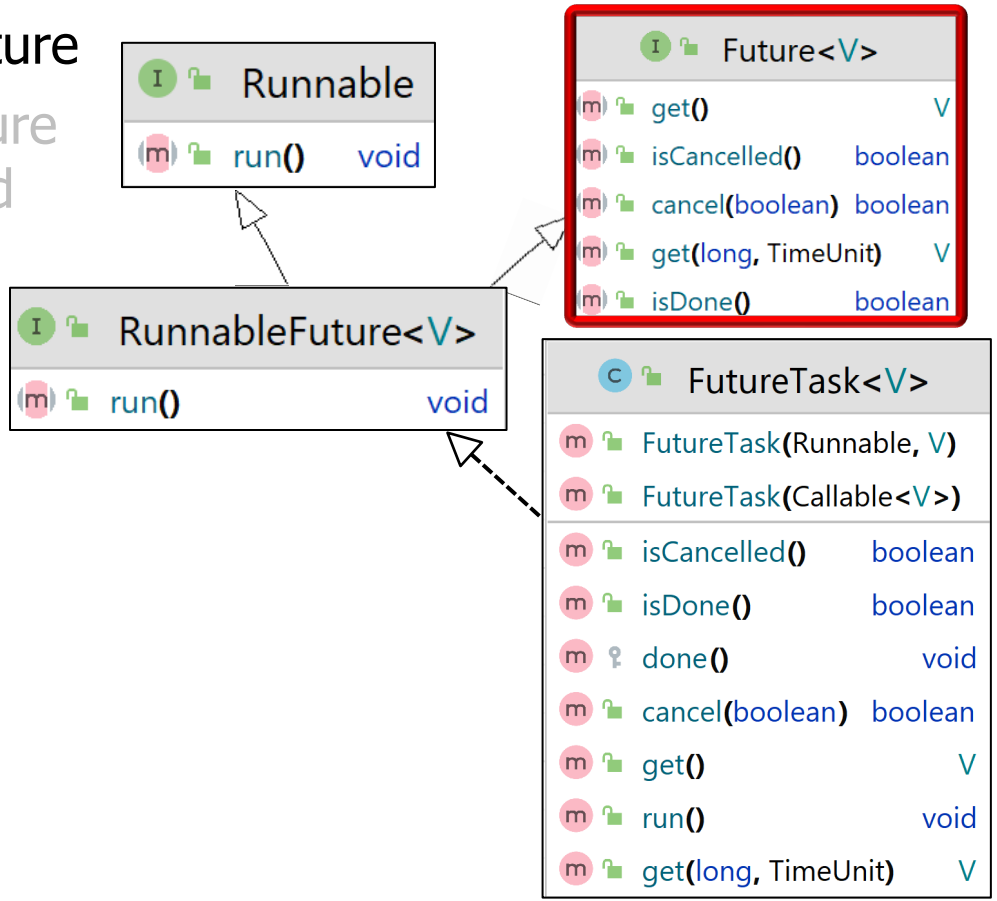


The run() hook method is called back in the right concurrency context

See javase/20/docs/api/java.base/java/lang/Runnable.html#run

Overview of Java FutureTask

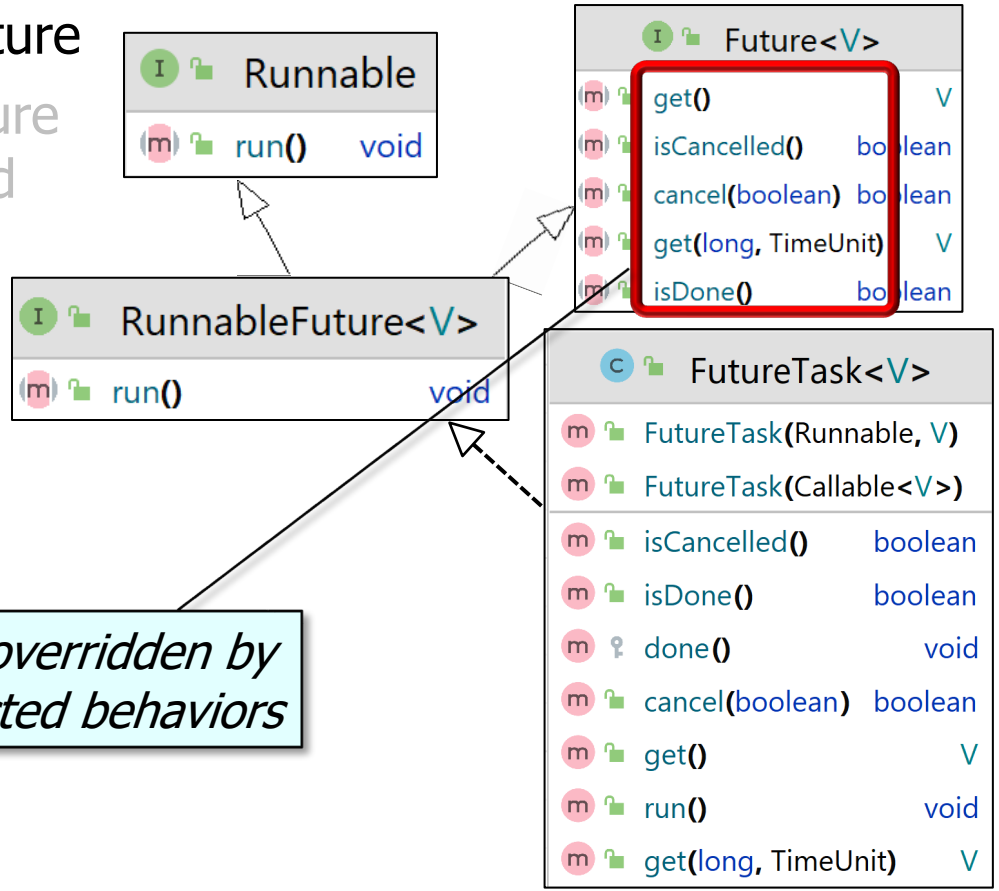
- FutureTask implements RunnableFuture
  - By implementing Runnable, a Future Task can be submitted to a Thread or Executor for execution
  - By implementing Future, a Future Task can process the results of an asynchronous computation



See 20/docs/api/java.base/java/util/concurrent/Future.html

Overview of Java FutureTask

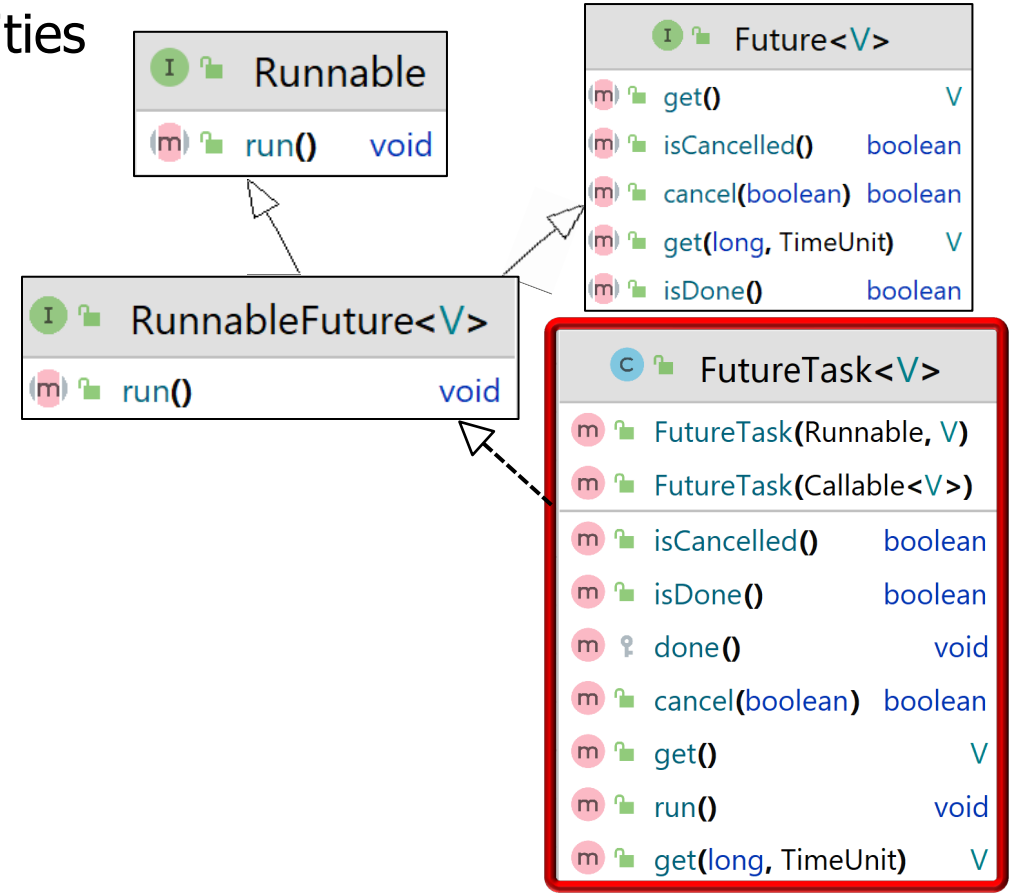
- FutureTask implements RunnableFuture
  - By implementing Runnable, a Future Task can be submitted to a Thread or Executor for execution
  - By implementing Future, a Future Task can process the results of an asynchronous computation



All the Future methods are overridden by FutureTask to do their expected behaviors

Overview of Java FutureTask

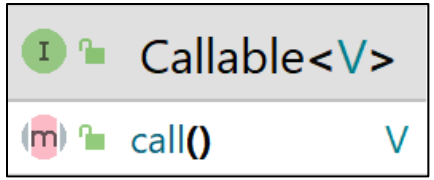
- FutureTask provides several capabilities

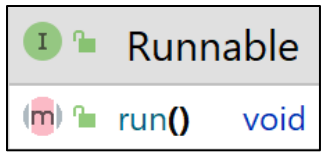


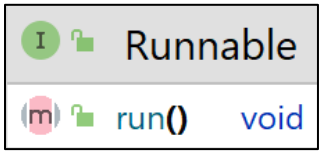
See www.geeksforgeeks.org/future-and-futuretask-in-java

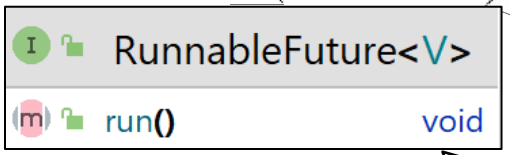
Overview of Java FutureTask

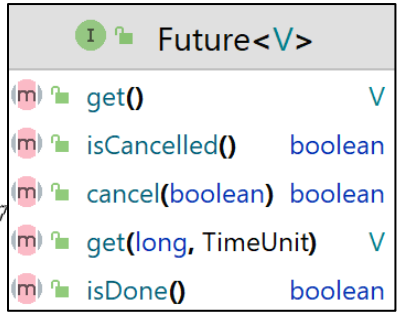
- FutureTask provides several capabilities, e.g.
  - It wraps Callable or Runnable
    - i.e., these computations can now run asynchronously

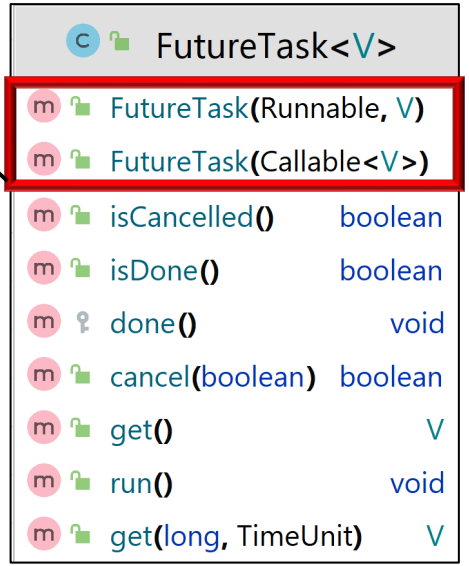








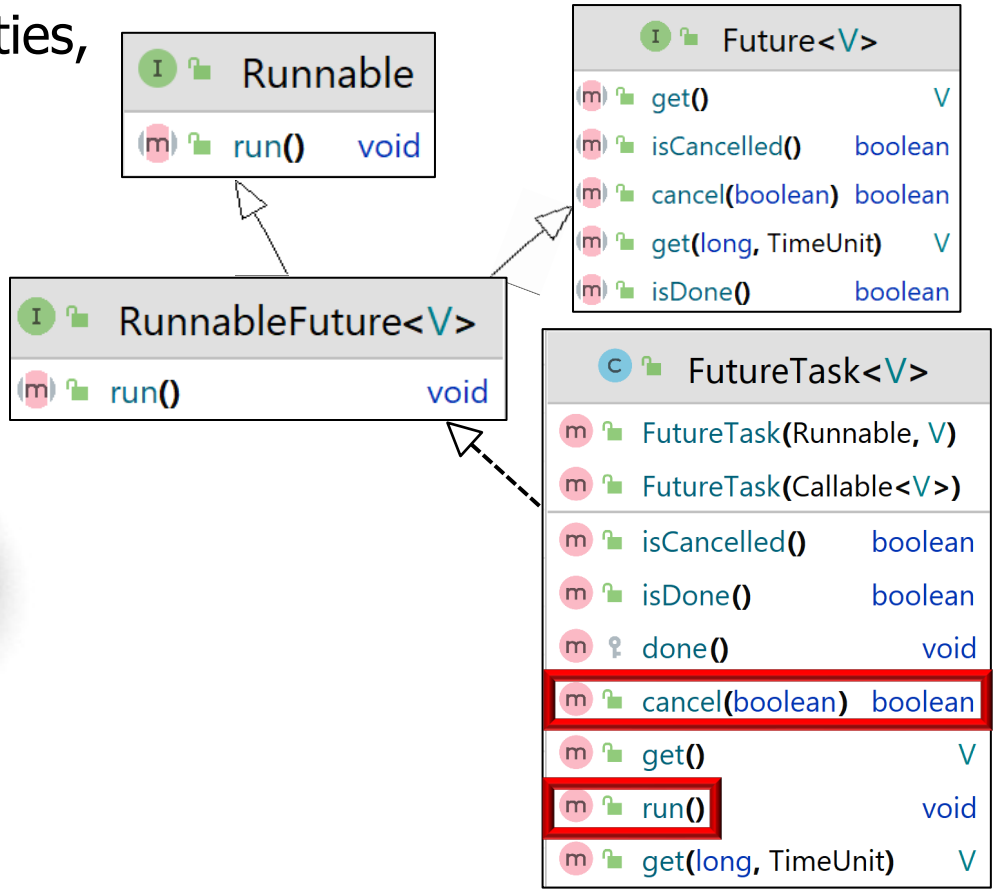




See javase/20/docs/api/java.base/java/util/concurrent/Callable.html

Overview of Java FutureTask

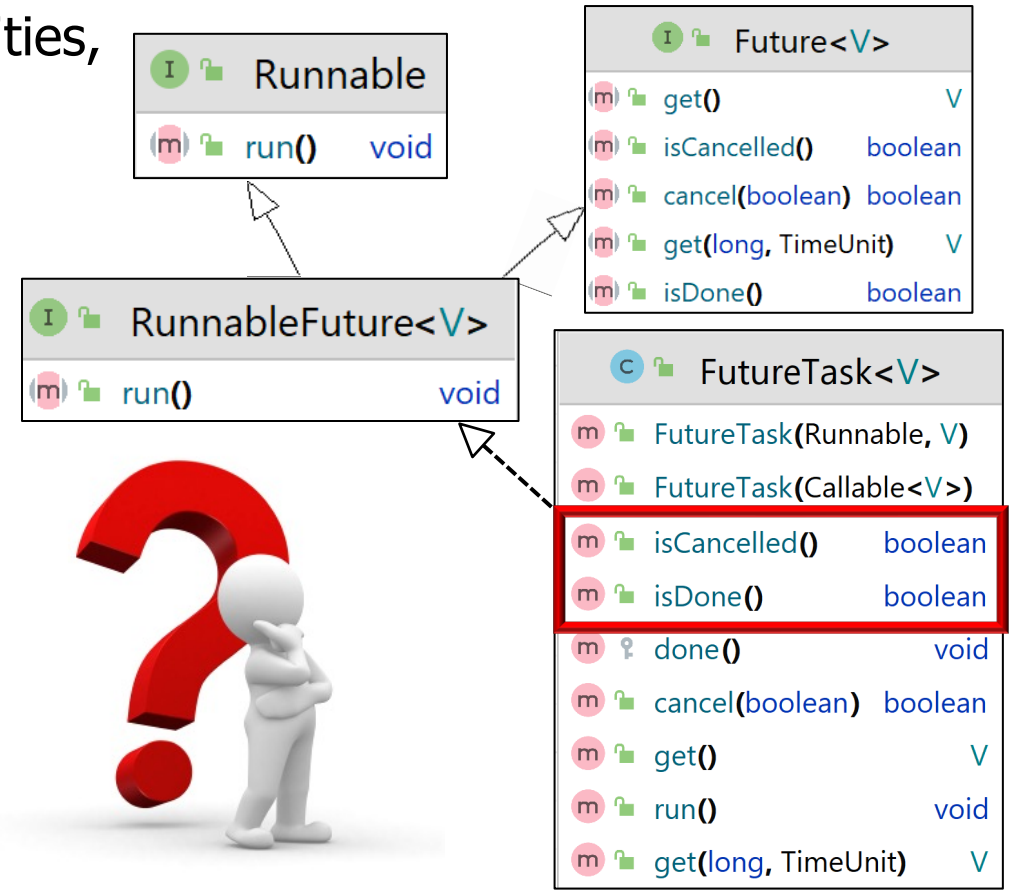
- FutureTask provides several capabilities, e.g.
  - It wraps Callable or Runnable
  - Start & cancel a computation that can run asynchronously



Async FutureTask computations can run via a Java Thread or some Executor

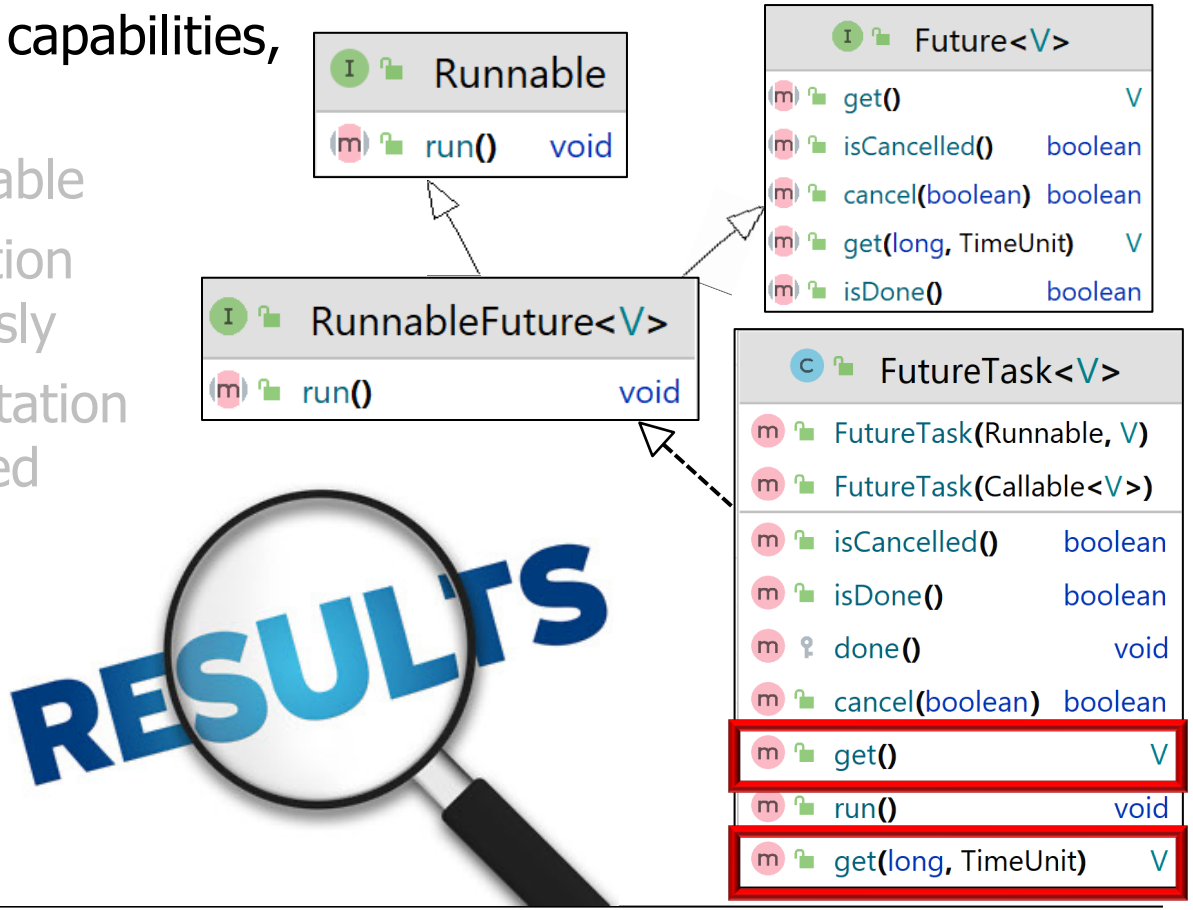
Overview of Java FutureTask

- FutureTask provides several capabilities, e.g.
  - It wraps Callable or Runnable
  - Start & cancel a computation that can run asynchronously
  - Query if the async computation completed or was cancelled



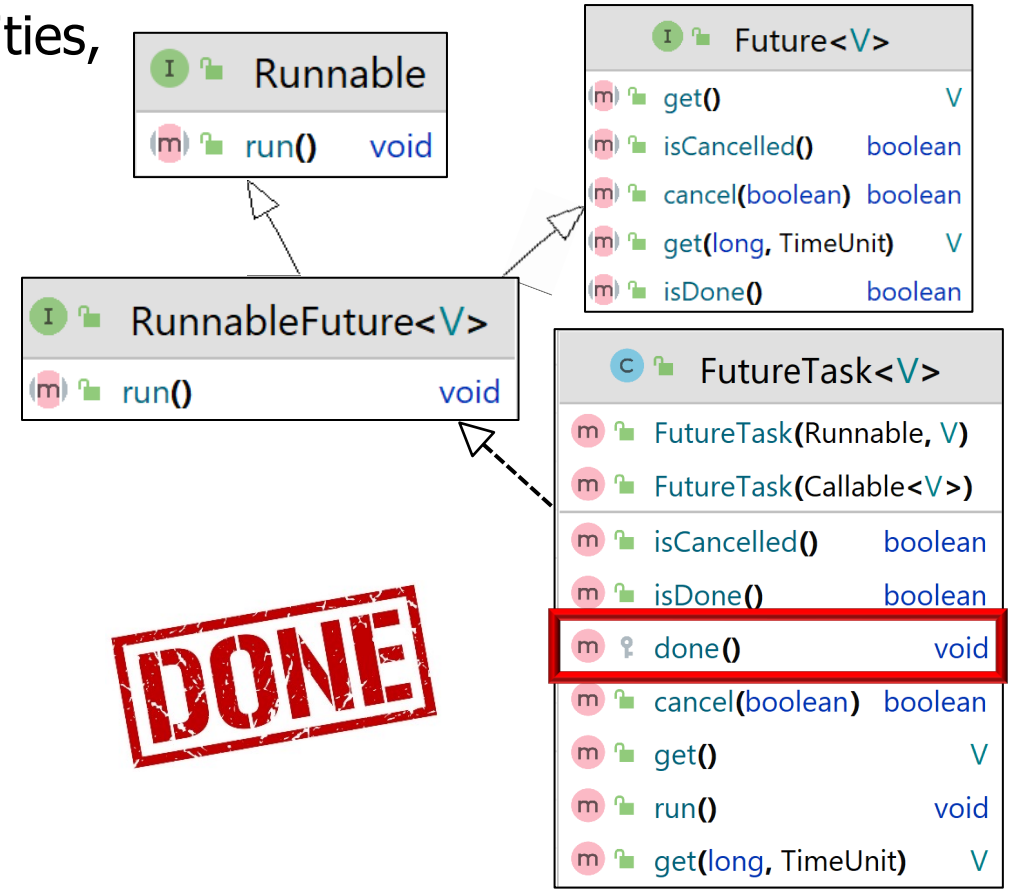
Overview of Java FutureTask

- FutureTask provides several capabilities, e.g.
  - It wraps Callable or Runnable
  - Start & cancel a computation that can run asynchronously
  - Query if the async computation completed or was cancelled
  - Get the result of an async computation



Overview of Java FutureTask

- FutureTask provides several capabilities, e.g.
  - It wraps Callable or Runnable
  - Start & cancel a computation that can run asynchronously
  - Query if the async computation completed or was cancelled
  - Get the result of an async computation
  - Hook method invoked when task transitions to isDone state



Subclasses may override this hook method, e.g., to invoke completion callbacks

Applying Java FutureTask

Applying Java FutureTask

- The ActiveObject class uses FutureTask to define a closure that applies a Function param within a virtual Thread object

```
RunnableFuture<R> makeThreadClosure
          (Function<T, R> function,
           T n) {

  var runnableFuture = new
      FutureTask<R>(() -> {
        return mResult = function
                  .apply(n);
  });

  mThread = Thread
     .startVirtualThread
        (runnableFuture);

  return runnableFuture;
}
```

See ModernJava/blob/main/FP/ex16/src/main/java/utils/ActiveObject.java

Applying Java FutureTask

- The ActiveObject class uses FutureTask to define a closure that applies a Function param within a virtual Thread object

This factory method is passed a 'function' & a parameter to apply this 'function' to

```
RunnableFuture<R> makeThreadClosure
        (Function<T, R> function,
         T n) {

  var runnableFuture = new
      FutureTask<R>(() -> {
        return mResult = function
                  .apply(n);
  });

  mThread = Thread
     .startVirtualThread
        (runnableFuture);

  return runnableFuture;
}
```

See javase/20/docs/api/java.base/java/util/function/Function.html

Applying Java FutureTask

- The ActiveObject class uses FutureTask to define a closure that applies a Function param within a virtual Thread object

```
RunnableFuture<R> makeThreadClosure
          (Function<T, R> function,
           T n) {

  var runnableFuture = new
      FutureTask<R>(() -> {
        return mResult = function
                  .apply(n);
  });

  mThread = Thread
     .startVirtualThread
        (runnableFuture);

  return runnableFuture;
}
```

Create a FutureTask that defines a Callable closure that applies the function' param to the param 'n'

See javase/20/docs/api/java.base/java/util/concurrent/FutureTask.html#FutureTask

Applying Java FutureTask

- The ActiveObject class uses FutureTask to define a closure that applies a Function param within a virtual Thread object

Create & return a new virtual Thread whose Runnable param is the Future Task that executes asynchronously

```
RunnableFuture<R> makeThreadClosure
          (Function<T, R> function,
           T n) {

  var runnableFuture = new
      FutureTask<R>(() -> {
        return mResult = function
                  .apply(n);
  });

  mThread = Thread
     .startVirtualThread
        (runnableFuture);

  return runnableFuture;
}
```

See javase/20/docs/api/java.base/java/lang/Thread.html#startVirtualThread

Applying Java FutureTask

- The ActiveObject class uses FutureTask to define a closure that applies a Function param within a virtual Thread object

Return the RunnableFuture, which is stored in the mRunnableField by the ActiveObject constructor

```
RunnableFuture<R> makeThreadClosure
          (Function<T, R> function,
           T n) {

  var runnableFuture = new
      FutureTask<R>(() -> {
        return mResult = function
                  .apply(n);
  });

  mThread = Thread
     .startVirtualThread
        (runnableFuture);

  return runnableFuture;
}
```

Applying Java FutureTask

- The ActiveObject class overrides all the Future methods it inherits

```
public boolean cancel
   (boolean mayInterruptIfRunning) {
   return mRunnableFuture
    .cancel(mayInterruptIfRunning);
}

public boolean isCancelled() {
   return mRunnableFuture
     .isCancelled();
}

public R get() ... {
   return mRunnableFuture.get();
} ...
```

Applying Java FutureTask

- The ActiveObject class overrides all the Future methods it inherits
  - & forwards them all to the mRunnableFuture field

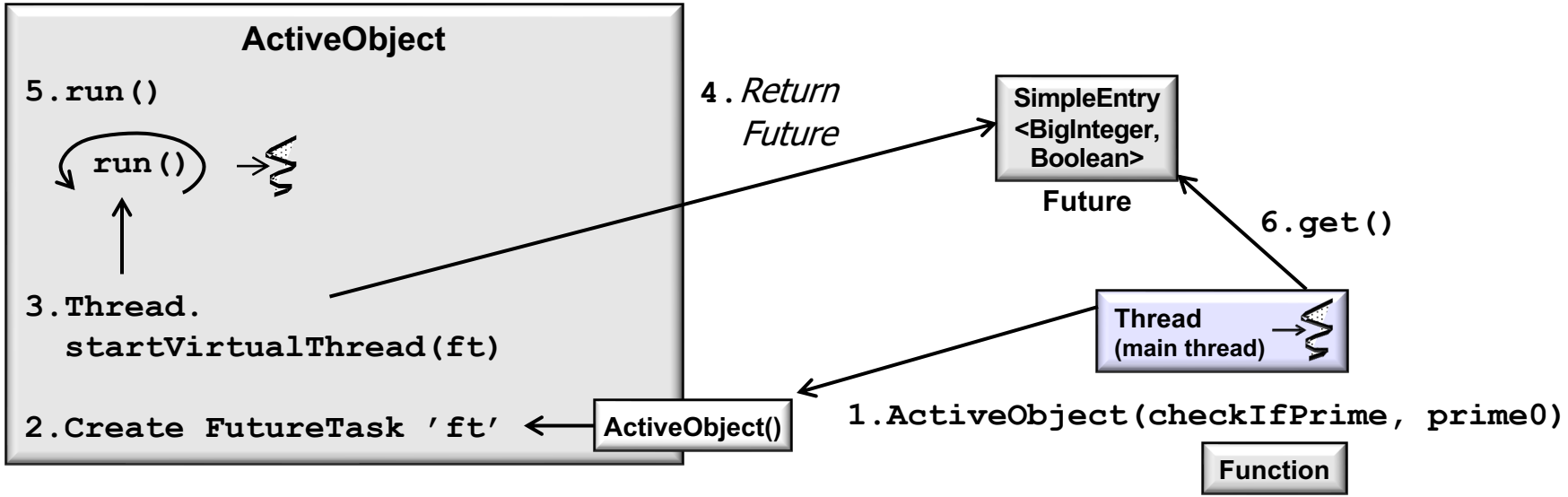
```
public boolean cancel
   (boolean mayInterruptIfRunning) {
   return mRunnableFuture
    .cancel(mayInterruptIfRunning);
}

public boolean isCancelled() {
   return mRunnableFuture
     .isCancelled();
}

public R get() ... {
   return mRunnableFuture.get();
} ...
```

Applying Java FutureTask

- The next part of this lesson shows how the Java FutureTask class can be combined with the Java Future interface & the *Active Object* pattern



See github.com/douglascraigschmidt/ModernJava/tree/main/FP/ex16

End of Overview of Java FutureTask