Overview of Java Futures

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

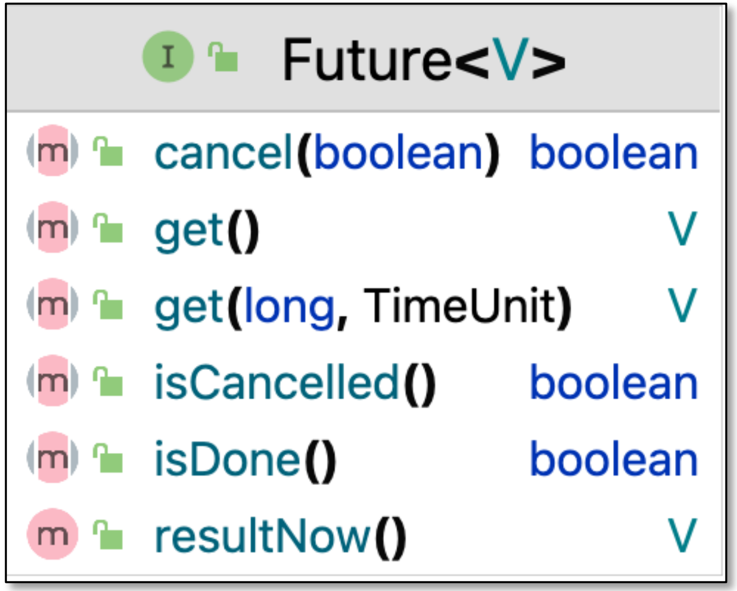
Professor of Computer Science

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

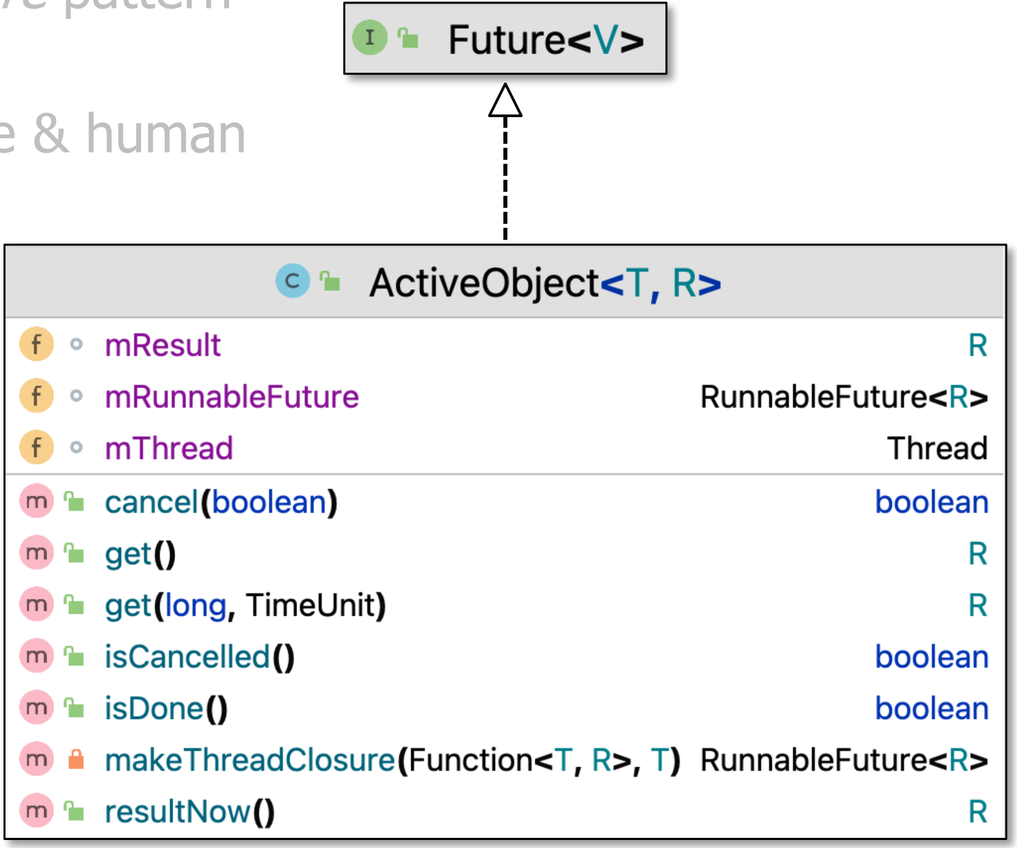


```
Future<V>
cancel(boolean)          boolean
get()                          V
get(long, TimeUnit)            V
isCancelled()            boolean
isDone()                 boolean
resultNow()                    V
```

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

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
  - The ActiveObject class from the ex16 case study is used as a running example

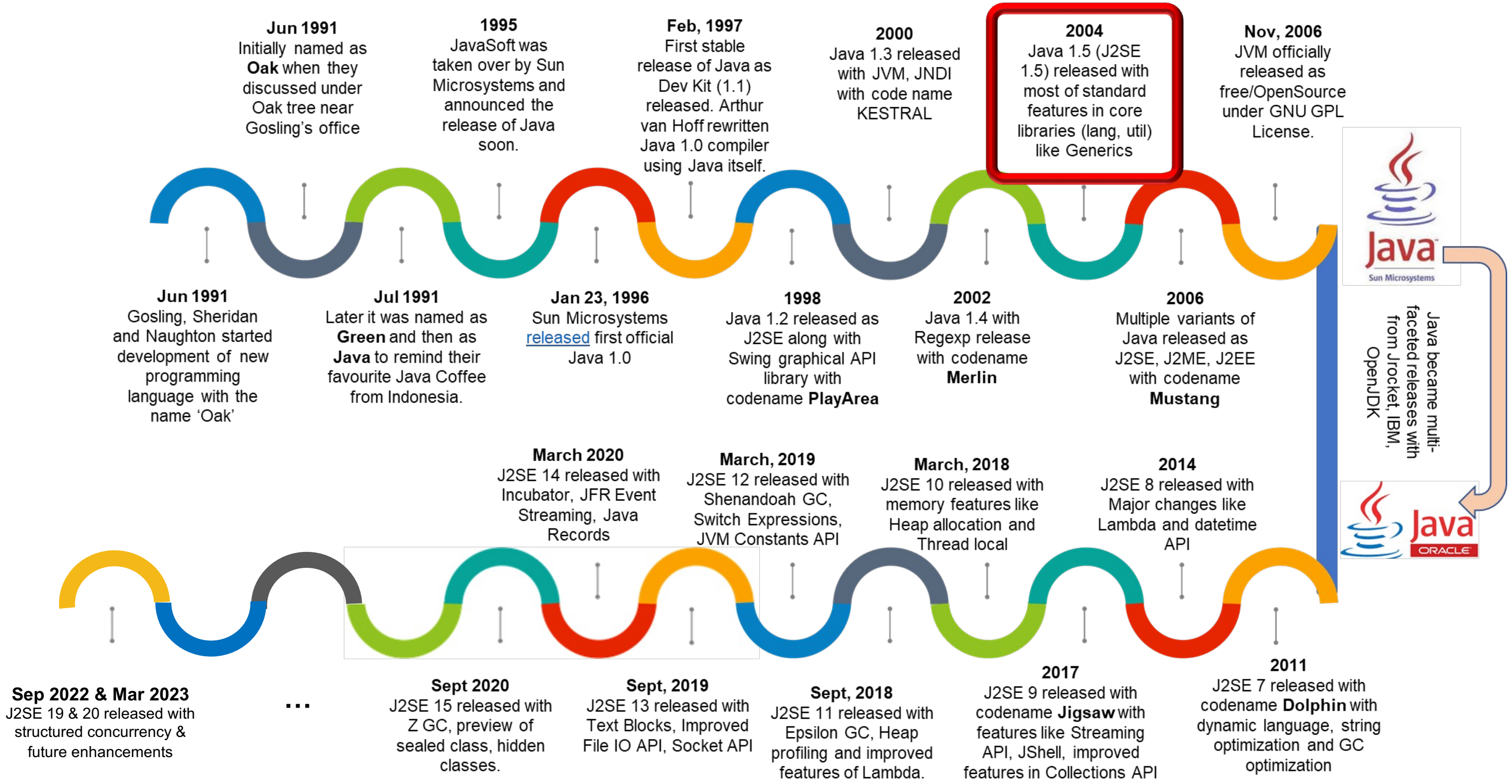


See ModernJava/blob/main/FP/ex16/src/main/java/utils/ActiveObject.java

Overview of the Modern Java Future API

Overview of the Modern Java Future API

- Java 5 added async call support via the Java Future interface



Jun 1991
Initially named as **Oak** when they discussed under Oak tree near Gosling's office

Jun 1991
Gosling, Sheridan and Naughton started development of new programming language with the name 'Oak'

Jul 1991
Later it was named as **Green** and then as **Java** to remind their favourite Java Coffee from Indonesia.

1995
JavaSoft was taken over by Sun Microsystems and announced the release of Java soon.

Jan 23, 1996
Sun Microsystems released first official Java 1.0

Feb, 1997
First stable release of Java as Dev Kit (1.1) released. Arthur van Hoff rewritten Java 1.0 compiler using Java itself.

1998
Java 1.2 released as J2SE along with Swing graphical API library with codename **PlayArea**

2000
Java 1.3 released with JVM, JNDI with code name KESTRAL

2002
Java 1.4 with Regexp release with codename **Merlin**

2004
Java 1.5 (J2SE 1.5) released with most of standard features in core libraries (lang, util) like Generics

2006
Multiple variants of Java released as J2SE, J2ME, J2EE with codename **Mustang**

Nov, 2006
JVM officially released as free/OpenSource under GNU GPL License.

Java became multi-faceted releases with from Jrocket, IBM, OpenJDK

2011
J2SE 7 released with codename **Dolphin** with dynamic language, string optimization and GC optimization

2014
J2SE 8 released with Major changes like Lambda and datetime API

2017
J2SE 9 released with codename **Jigsaw** with features like Streaming API, JShell, improved features in Collections API

March, 2018
J2SE 10 released with memory features like Heap allocation and Thread local

Sept, 2018
J2SE 11 released with Epsilon GC, Heap profiling and improved features of Lambda.

March, 2019
J2SE 12 released with Shenandoah GC, Switch Expressions, JVM Constants API

Sept, 2019
J2SE 13 released with Text Blocks, Improved File IO API, Socket API

March 2020
J2SE 14 released with Incubator, JFR Event Streaming, Java Records

Sept 2020
J2SE 15 released with Z GC, preview of sealed class, hidden classes.

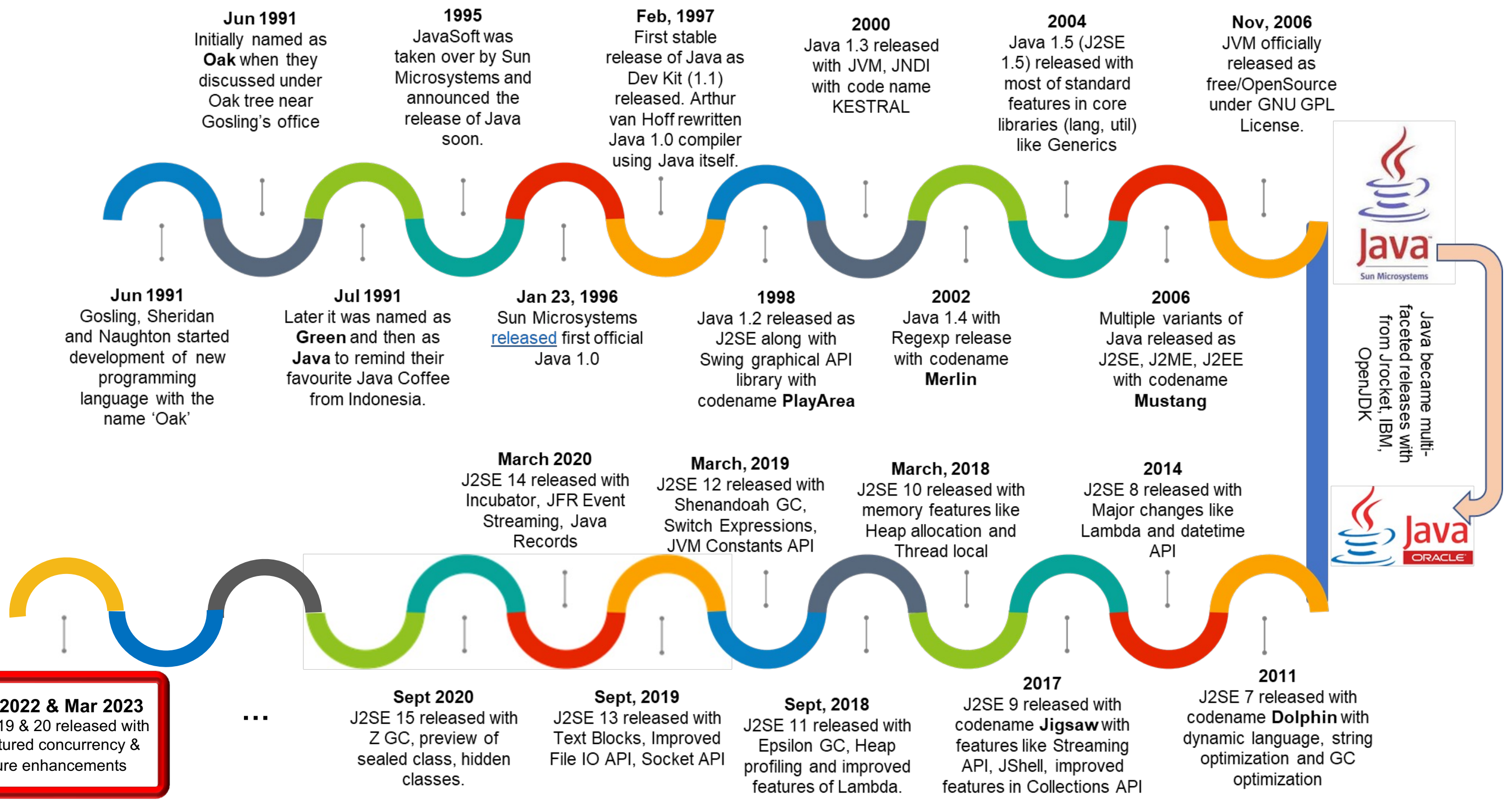
...

Sep 2022 & Mar 2023
J2SE 19 & 20 released with structured concurrency & future enhancements

See en.wikipedia.org/wiki/Java_version_history

Overview of the Modern Java Future API

- Java 19+ added several enhancements to the Java Future interface



Jun 1991
Initially named as **Oak** when they discussed under Oak tree near Gosling's office

Jun 1991
Gosling, Sheridan and Naughton started development of new programming language with the name 'Oak'

Jul 1991
Later it was named as **Green** and then as **Java** to remind their favourite Java Coffee from Indonesia.

1995
JavaSoft was taken over by Sun Microsystems and announced the release of Java soon.

Jan 23, 1996
Sun Microsystems released first official Java 1.0

Feb, 1997
First stable release of Java as Dev Kit (1.1) released. Arthur van Hoff rewritten Java 1.0 compiler using Java itself.

1998
Java 1.2 released as J2SE along with Swing graphical API library with codename **PlayArea**

2000
Java 1.3 released with JVM, JNDI with code name KESTRAL

2002
Java 1.4 with Regexp release with codename **Merlin**

2004
Java 1.5 (J2SE 1.5) released with most of standard features in core libraries (lang, util) like Generics

2006
Multiple variants of Java released as J2SE, J2ME, J2EE with codename **Mustang**

Nov, 2006
JVM officially released as free/OpenSource under GNU GPL License.

Java became multi-faceted releases with from Jrocket, IBM, OpenJDK

2011
J2SE 7 released with codename **Dolphin** with dynamic language, string optimization and GC optimization

2014
J2SE 8 released with Major changes like Lambda and datetime API

2017
J2SE 9 released with codename **Jigsaw** with features like Streaming API, JShell, improved features in Collections API

March, 2018
J2SE 10 released with memory features like Heap allocation and Thread local

Sept, 2018
J2SE 11 released with Epsilon GC, Heap profiling and improved features of Lambda.

March, 2019
J2SE 12 released with Shenandoah GC, Switch Expressions, JVM Constants API

Sept, 2019
J2SE 13 released with Text Blocks, Improved File IO API, Socket API

March 2020
J2SE 14 released with Incubator, JFR Event Streaming, Java Records

Sept 2020
J2SE 15 released with Z GC, preview of sealed class, hidden classes.

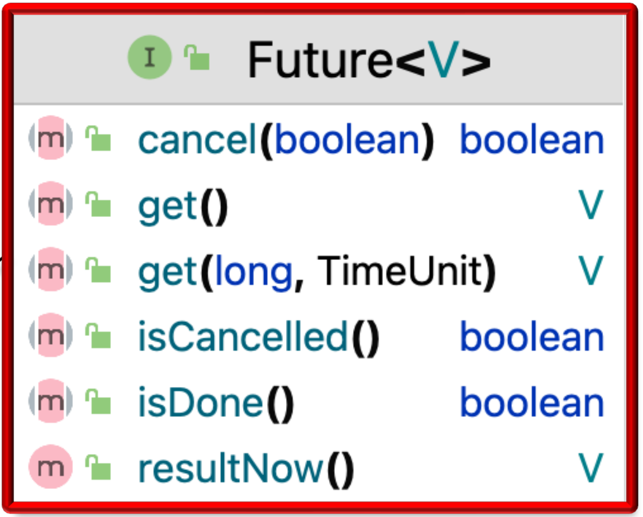
...

Sep 2022 & Mar 2023
J2SE 19 & 20 released with structured concurrency & future enhancements

See openjdk.org/jeps/437

Overview of the Modern Java Future API

- A **Future** provides a proxy to the result of an asynchronous computation



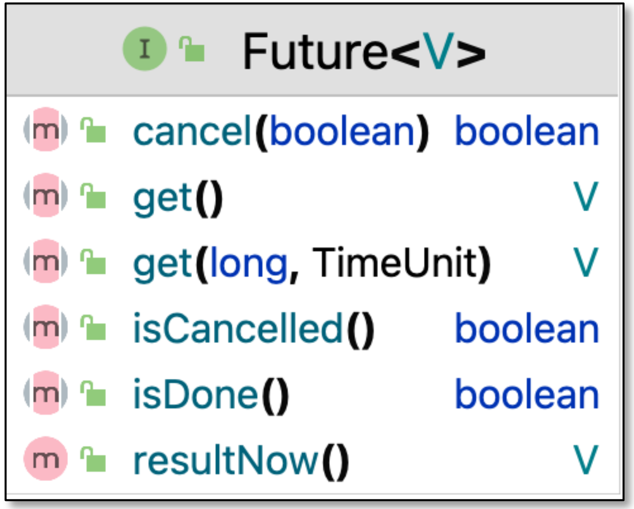
```
Future<V>
cancel(boolean)        boolean
get()                  V
get(long, TimeUnit)    V
isCancelled()          boolean
isDone()               boolean
resultNow()            V
```

Its methods check if an asynchronous computation is complete or canceled, cancel the computation if needed, wait for its completion, & retrieve the result (if any)

See [20/docs/api/java.base/java/util/concurrent/Future.html]()

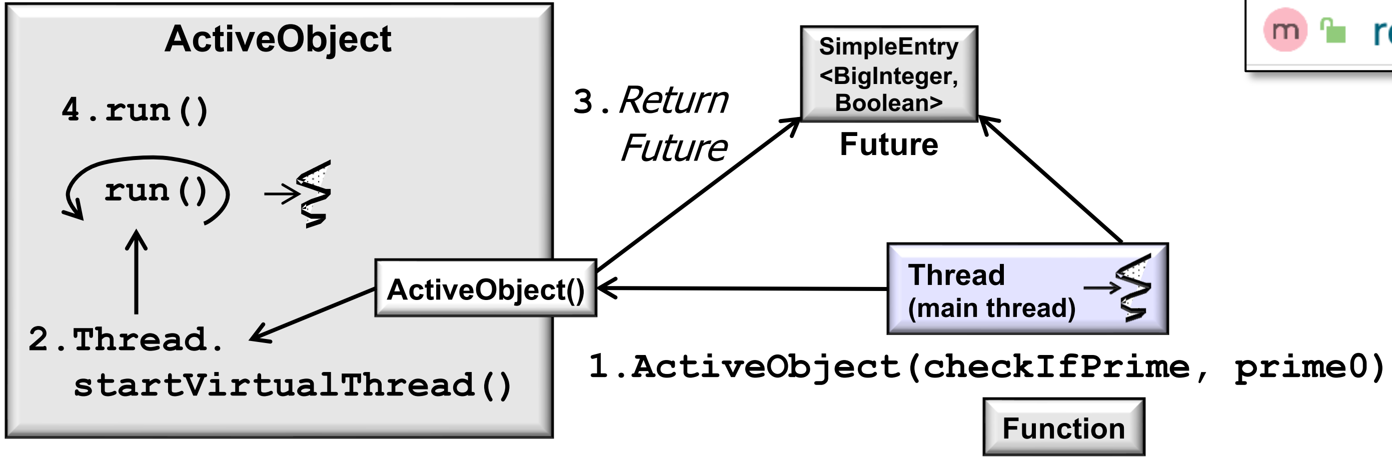
Overview of the Modern Java Future API

- Java Future methods manage a task's lifecycle after it's submitted to run asynchronously



Future<V>

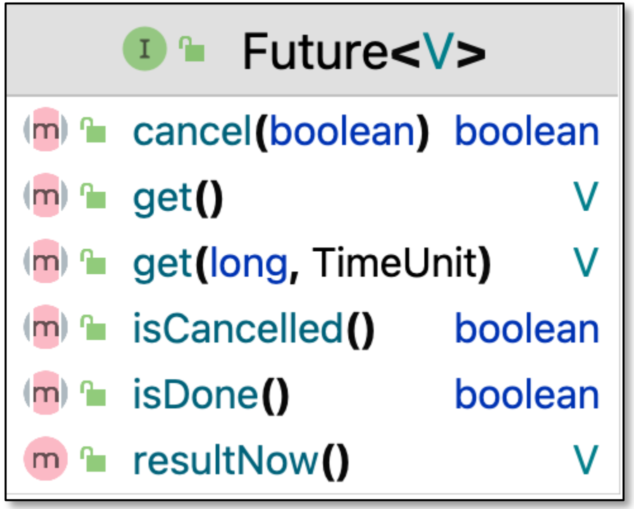
Method	Return
cancel(boolean)	boolean
get()	V
get(long, TimeUnit)	V
isCancelled()	boolean
isDone()	boolean
resultNow()	V



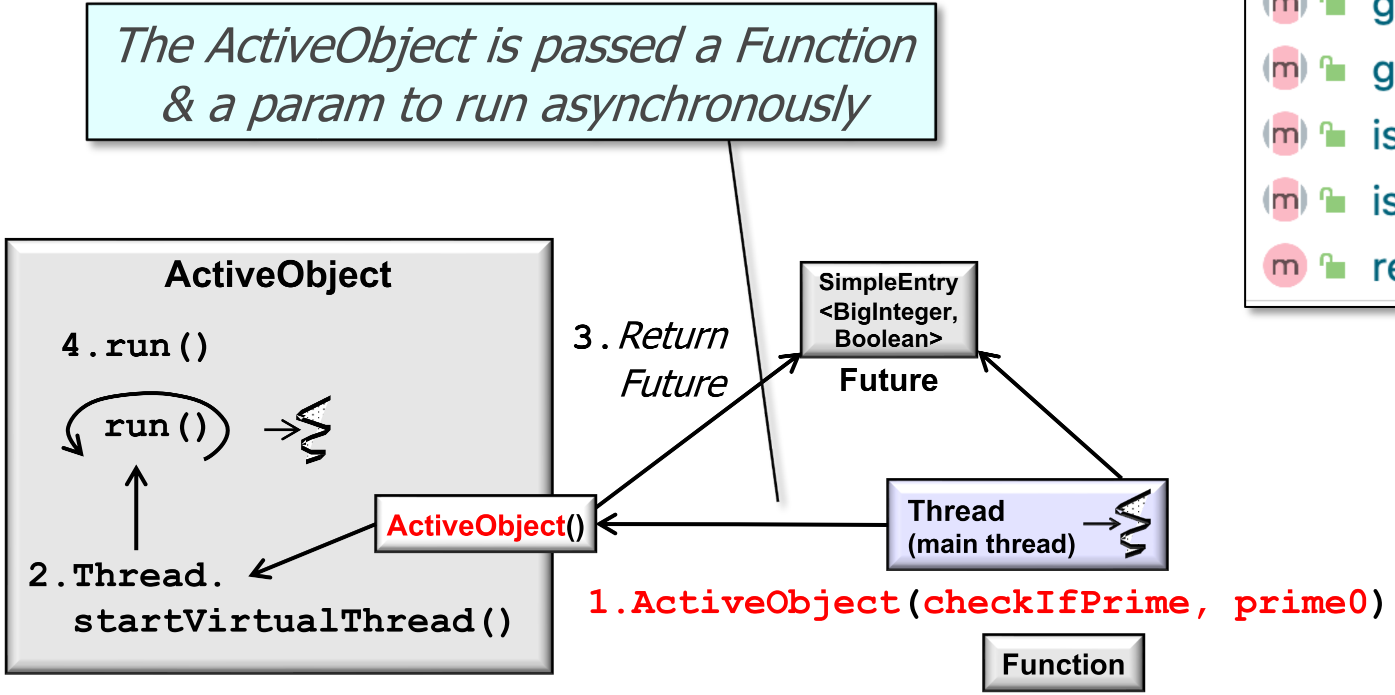
See en.wikipedia.org/wiki/Samsara_(Buddhism)

Overview of the Modern Java Future API

- Java Future methods manage a task's lifecycle after it's submitted to run asynchronously



Future<V>	
cancel(boolean)	boolean
get()	V
get(long, TimeUnit)	V
isCancelled()	boolean
isDone()	boolean
resultNow()	V

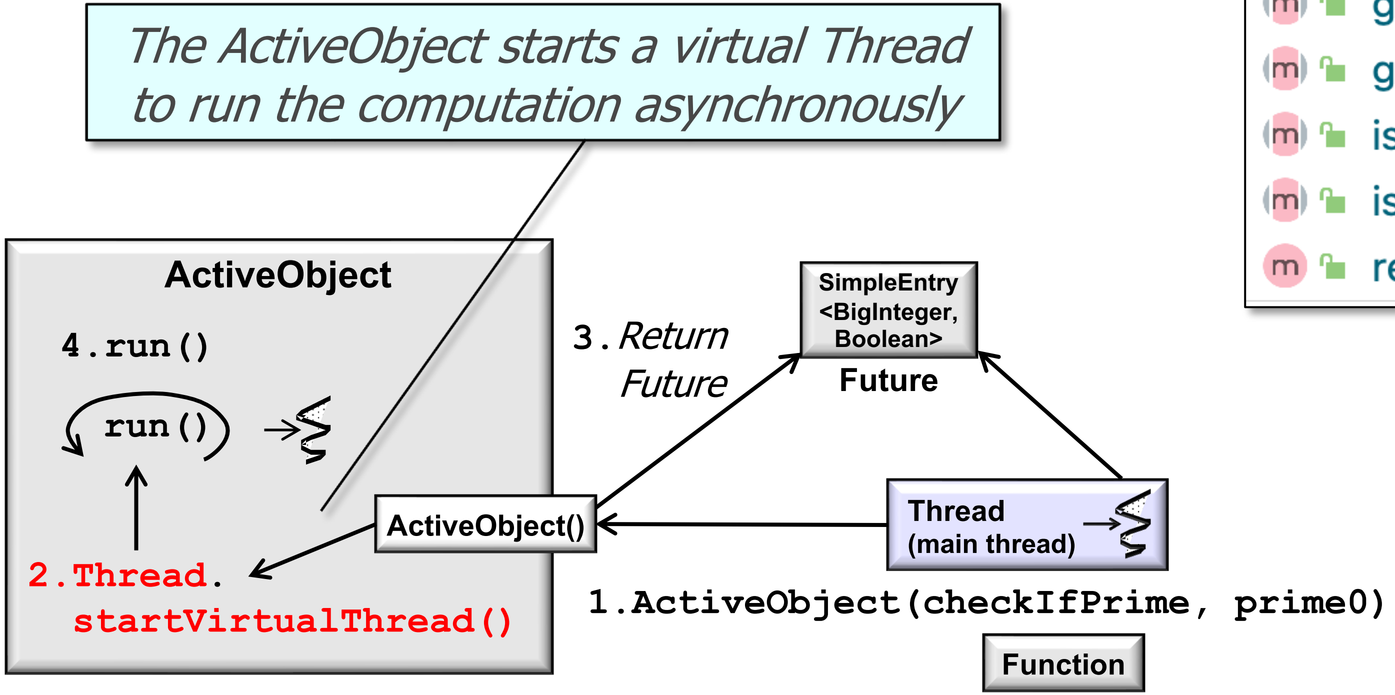


The ActiveObject is passed a Function & a param to run asynchronously
ActiveObject
4.run()
run()
2.Thread.startVirtualThread()
ActiveObject()
3.*Return Future*
SimpleEntry<BigInteger, Boolean>
Future
Thread (main thread)
1.ActiveObject(checkIfPrime, prime0)
Function

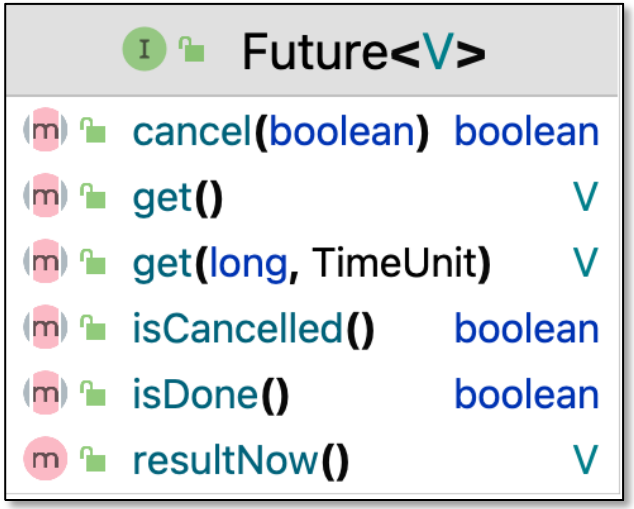
See ModernJava/blob/main/FP/ex16/src/main/java/utils/ActiveObject.java

Overview of the Modern Java Future API

- Java Future methods manage a task's lifecycle after it's submitted to run asynchronously



The ActiveObject starts a virtual Thread to run the computation asynchronously
ActiveObject
4.run()
run()
2.Thread.
startVirtualThread()
ActiveObject()
3. *Return Future*
SimpleEntry <BigInteger, Boolean>
Future
Thread (main thread)
1.ActiveObject(checkIfPrime, prime0)
Function



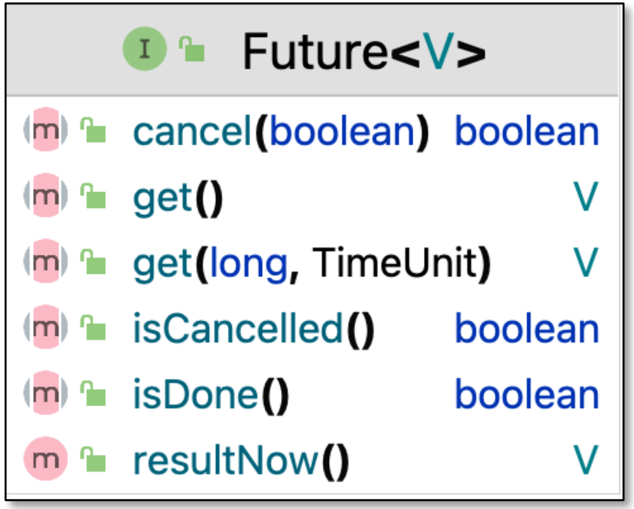
Future<V>

cancel(boolean)	boolean
get()	V
get(long, TimeUnit)	V
isCancelled()	boolean
isDone()	boolean
resultNow()	V

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

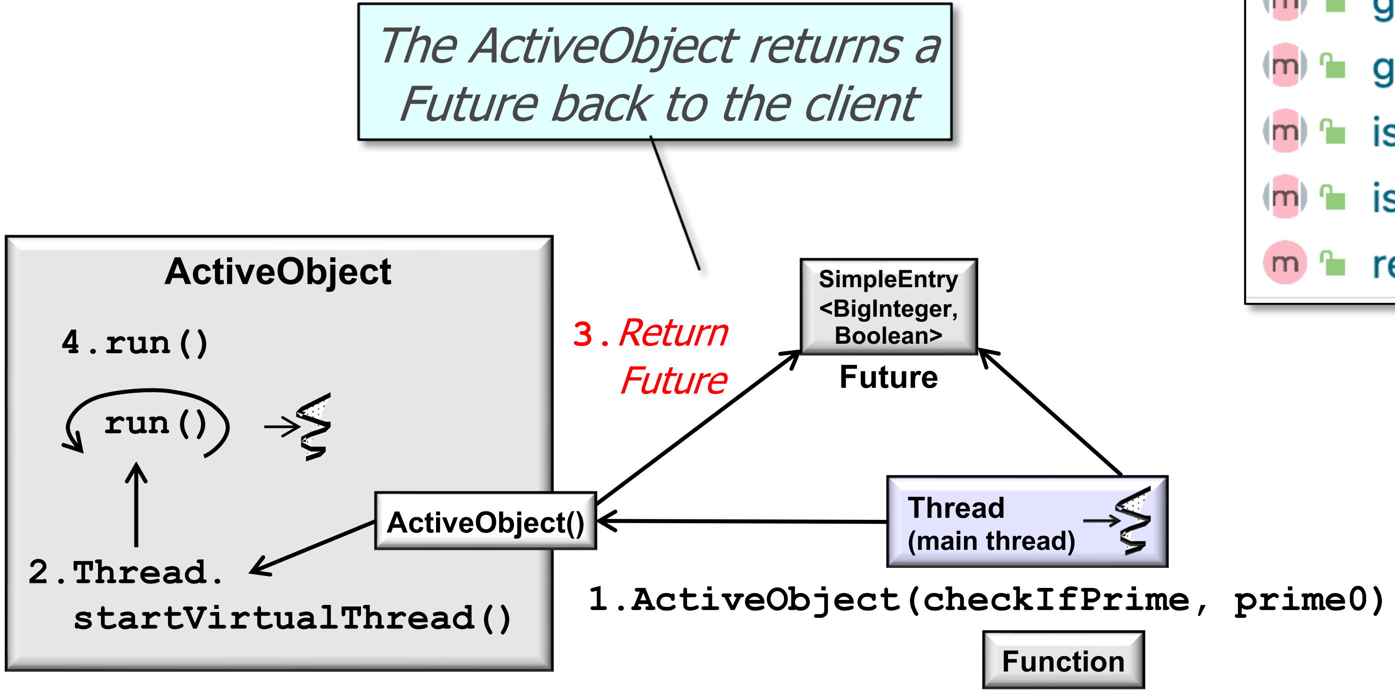
Overview of the Modern Java Future API

- Java Future methods manage a task's lifecycle after it's submitted to run asynchronously



Future<V>

cancel(boolean)	boolean
get()	V
get(long, TimeUnit)	V
isCancelled()	boolean
isDone()	boolean
resultNow()	V

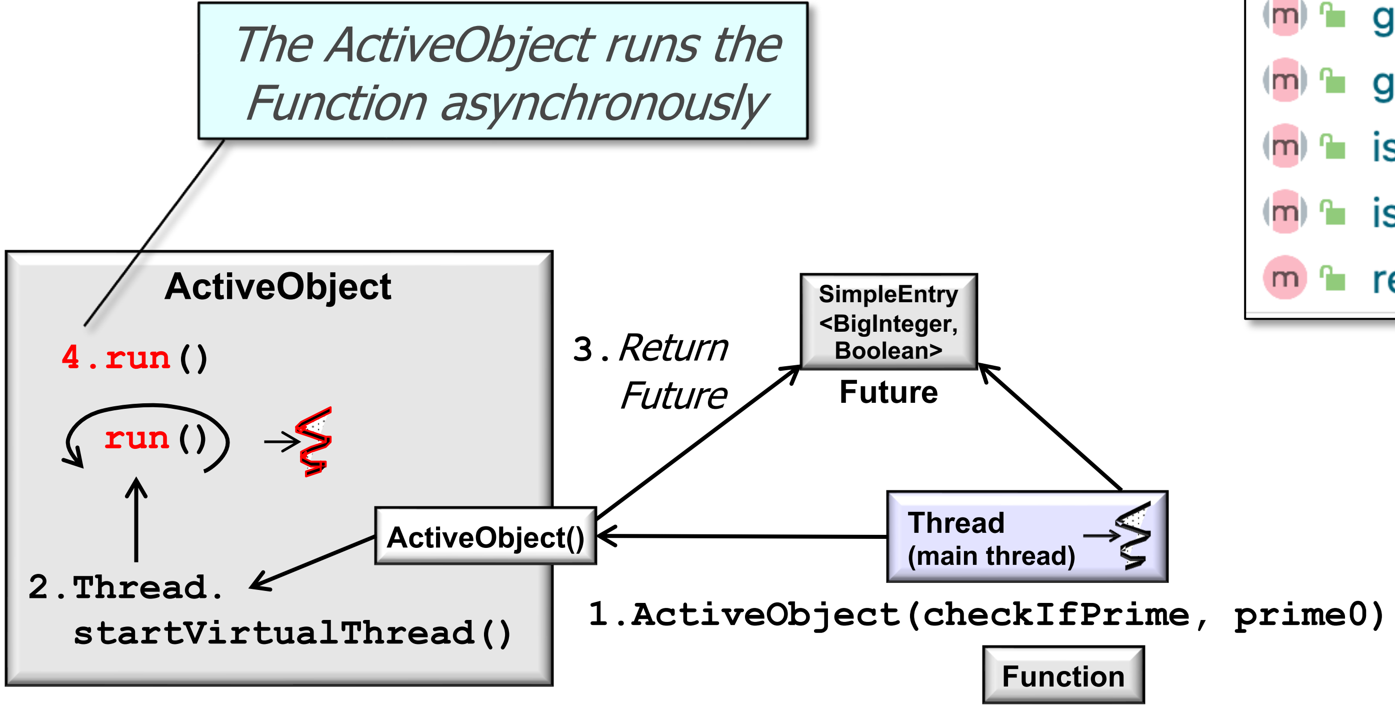


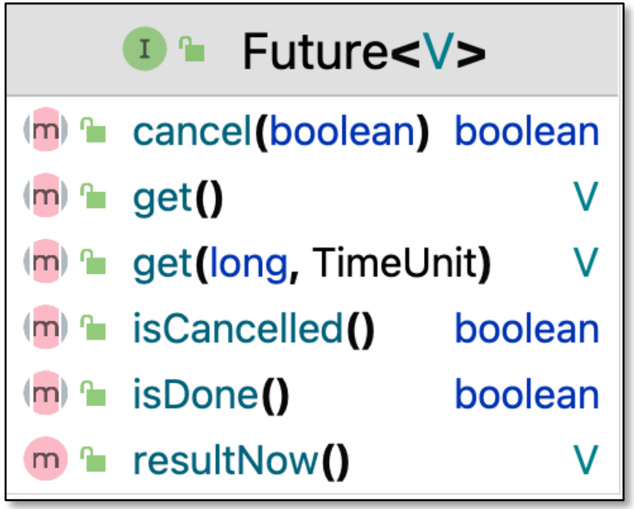
The ActiveObject returns a Future back to the client
ActiveObject
4.run()
run()
2.Thread.
startVirtualThread()
ActiveObject()
3.Return Future
SimpleEntry <BigInteger, Boolean>
Future
Thread (main thread)
1.ActiveObject(checkIfPrime, prime0)
Function

The ActiveObject class implements the Future interface

Overview of the Modern Java Future API

- Java Future methods manage a task's lifecycle after it's submitted to run asynchronously


The ActiveObject runs the Function asynchronously
ActiveObject
4.run()
run()
2.Thread.
startVirtualThread()
ActiveObject()
3.Return Future
SimpleEntry <BigInteger, Boolean>
Future
Thread (main thread)
1.ActiveObject(checkIfPrime, prime0)
Function



Future<V>

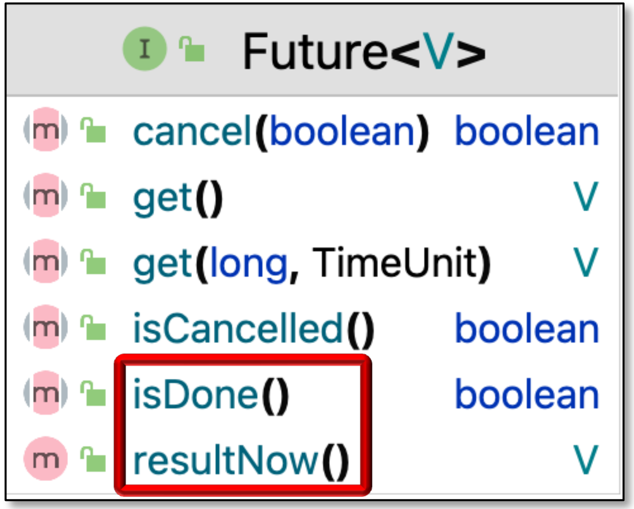
cancel(boolean)	boolean
get()	V
get(long, TimeUnit)	V
isCancelled()	boolean
isDone()	boolean
resultNow()	V

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

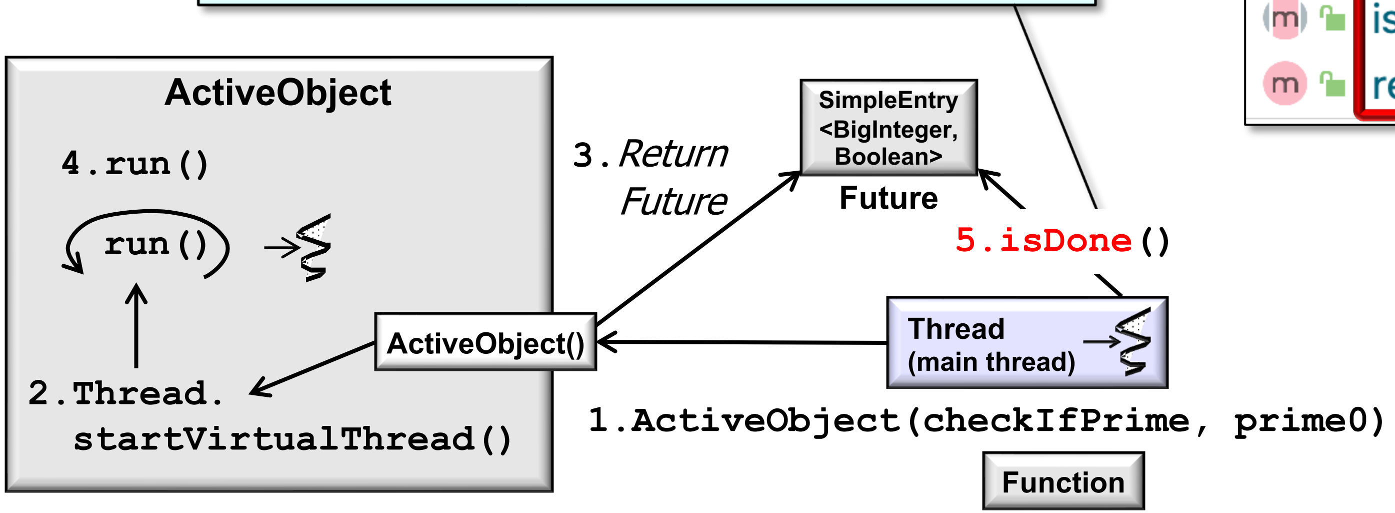
Overview of the Modern Java Future API

- Java Future methods manage a task's lifecycle after it's submitted to run asynchronously

```
if (future.isDone())
   result.add(future.resultNow());
```



Future<V>	
cancel(boolean)	boolean
get()	V
get(long, TimeUnit)	V
isCancelled()	boolean
isDone()	boolean
resultNow()	V

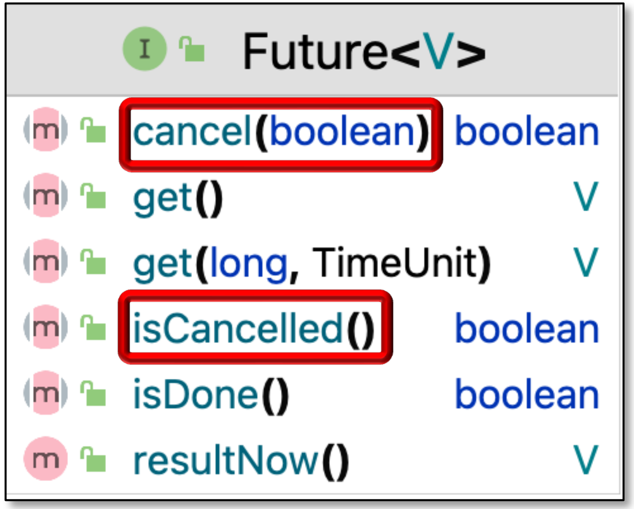


A Future can be tested for completion & obtained immediately

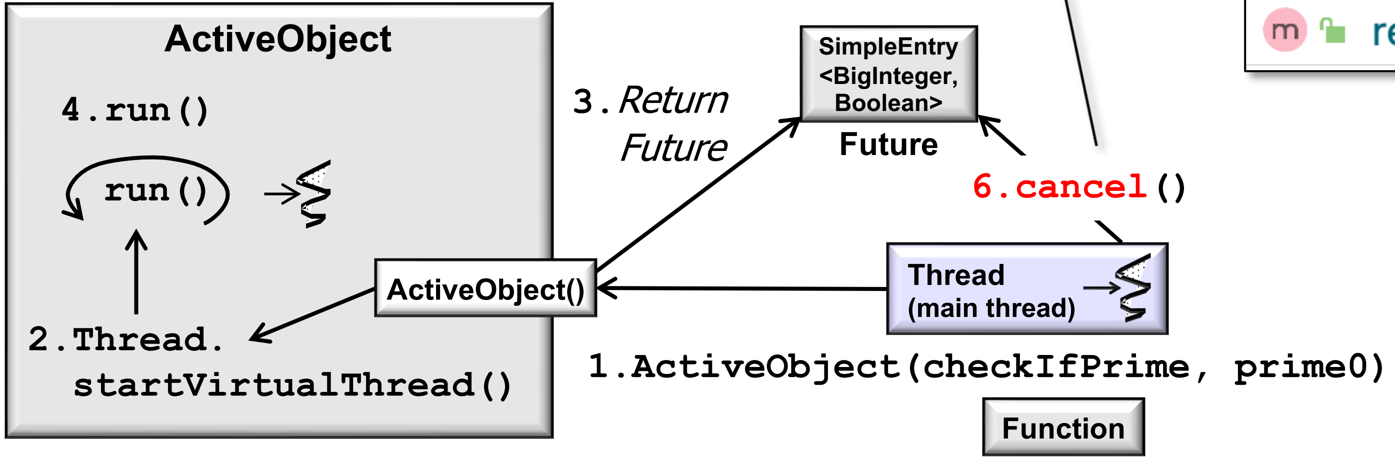
Overview of the Modern Java Future API

- Java Future methods manage a task's lifecycle after it's submitted to run asynchronously

```
if (!future.isDone() &&
     !future.isCancelled())
   future.cancel(true);
   ...
```



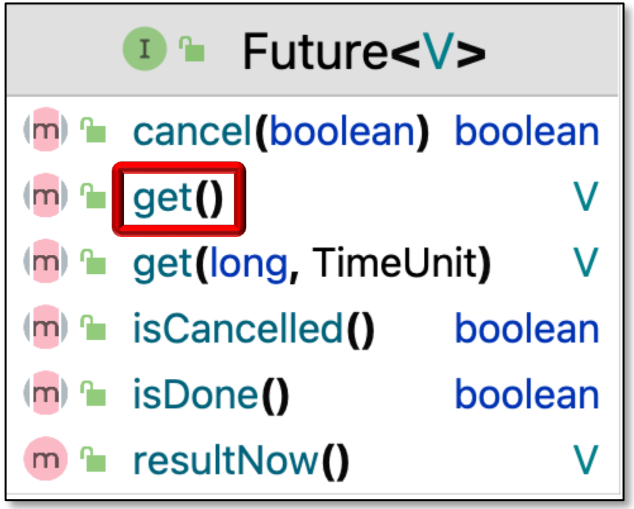
Future<V>	
cancel(boolean)	boolean
get()	V
get(long, TimeUnit)	V
isCancelled()	boolean
isDone()	boolean
resultNow()	V



A Future can be tested for cancelation & can be canceled

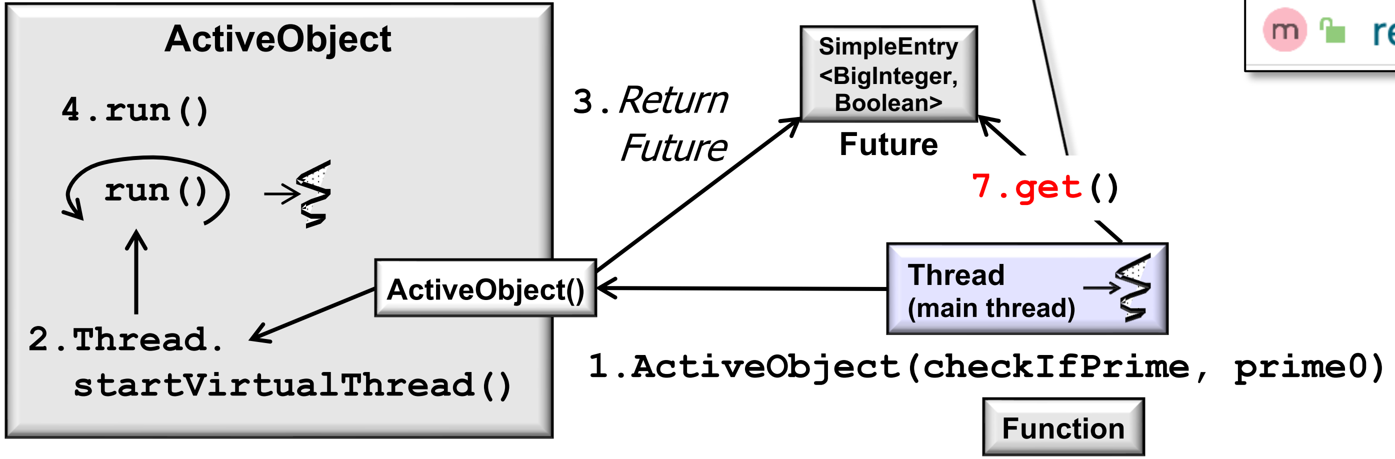
Overview of the Modern Java Future API

- Java Future methods manage a task's lifecycle after it's submitted to run asynchronously



Future<V>	
cancel(boolean)	boolean
get()	V
get(long, TimeUnit)	V
isCancelled()	boolean
isDone()	boolean
resultNow()	V

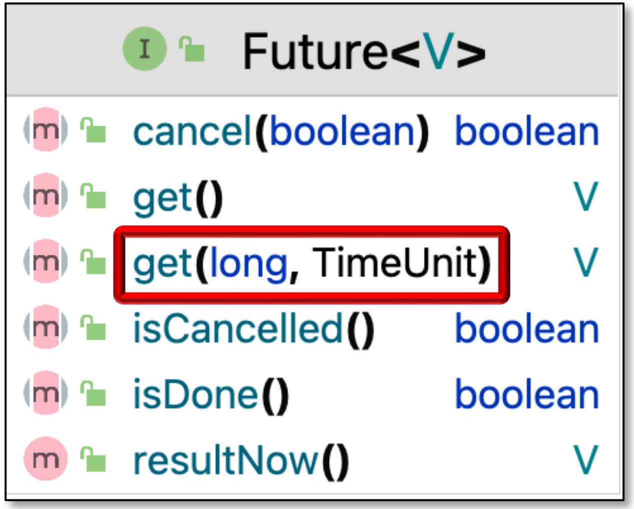
```
var result = future.get();
```



A Future can retrieve a two-way task's result in a blocking manner

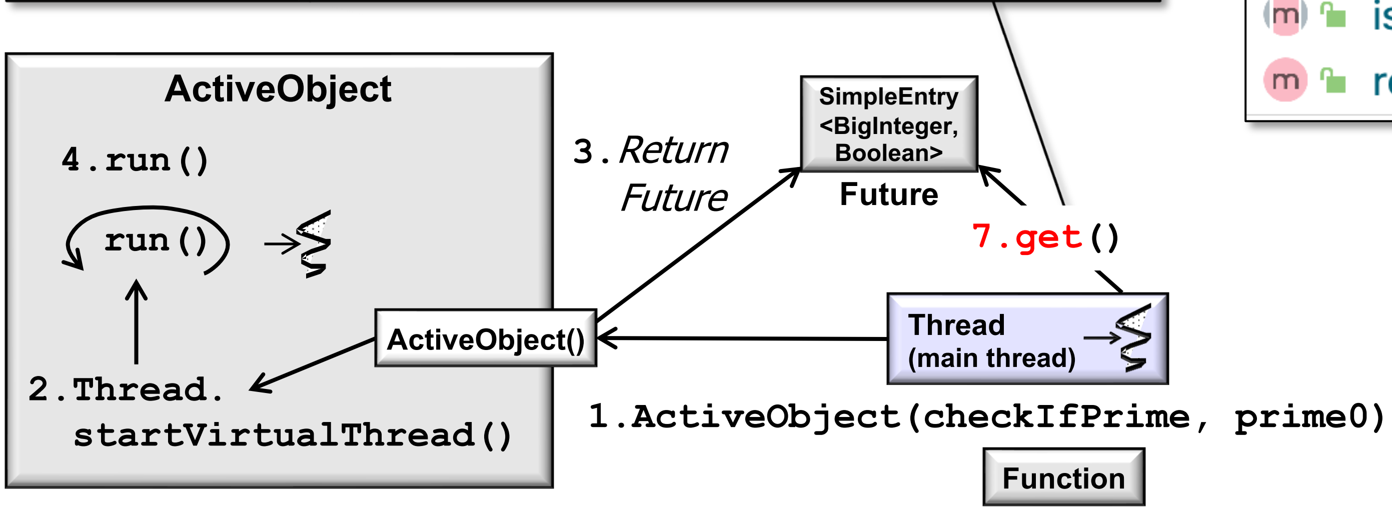
Overview of the Modern Java Future API

- Java Future methods manage a task's lifecycle after it's submitted to run asynchronously



Future<V>	
cancel(boolean)	boolean
get()	V
get(long, TimeUnit)	V
isCancelled()	boolean
isDone()	boolean
resultNow()	V

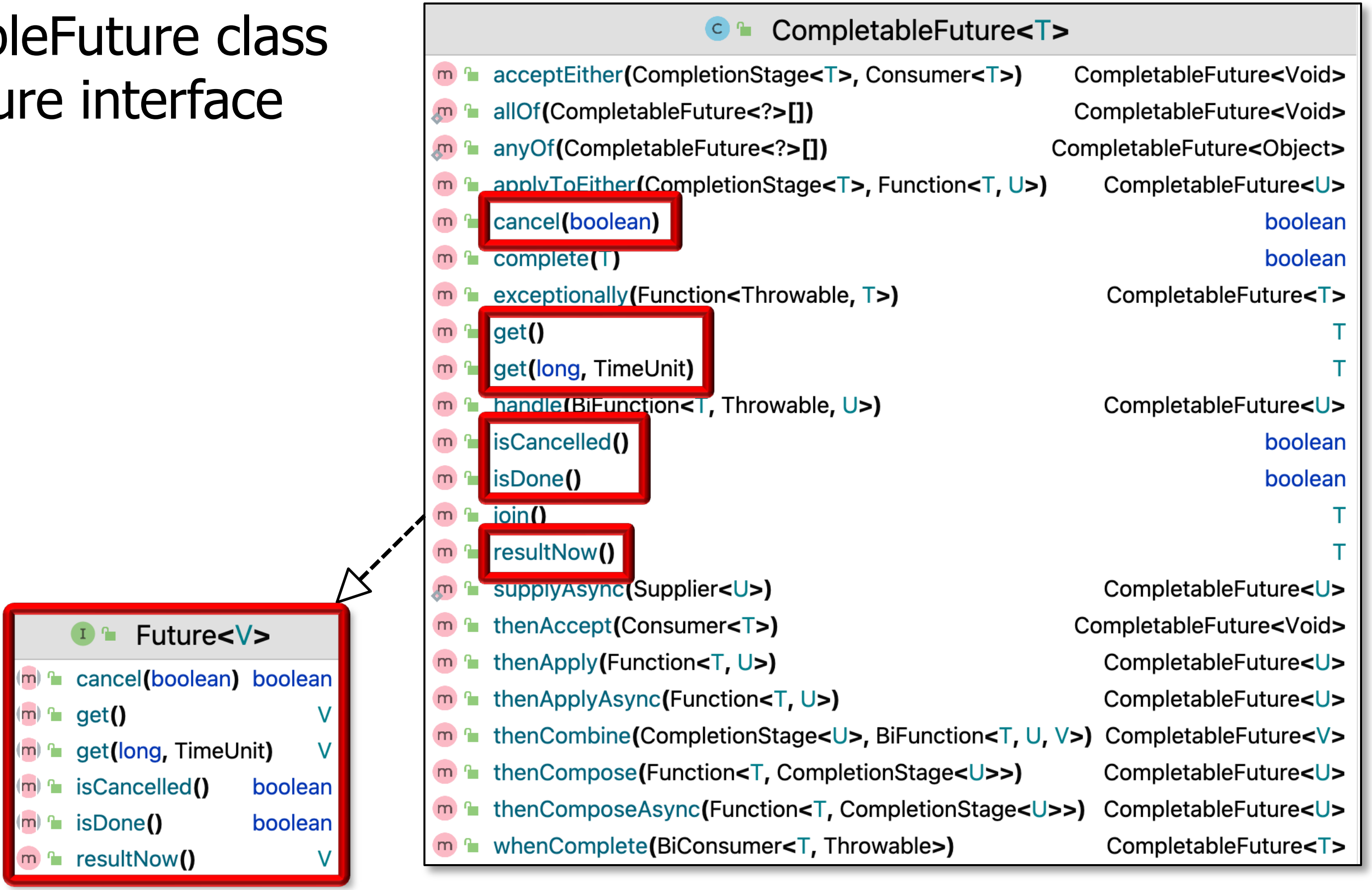
```
var result = future.get(waitTime, SECONDS);
```



A two-way task's result can also be obtained in a non-blocking or timed manner

Overview of the Modern Java Future API

- The Java CompletableFuture class implements the Future interface



CompletableFuture<T>

acceptEither(CompletionStage<T>, Consumer<T>)	CompletableFuture<Void>
allOf(CompletableFuture<?>[])	CompletableFuture<Void>
anyOf(CompletableFuture<?>[])	CompletableFuture<Object>
applyToEither(CompletionStage<T>, Function<T, U>)	CompletableFuture<U>
cancel(boolean)	boolean
complete(T)	boolean
exceptionally(Function<Throwable, T>)	CompletableFuture<T>
get()	T
get(long, TimeUnit)	T
handle(BiFunction<T, Throwable, U>)	CompletableFuture<U>
isCancelled()	boolean
isDone()	boolean
join()	T
resultNow()	T
supplyAsync(Supplier<U>)	CompletableFuture<U>
thenAccept(Consumer<T>)	CompletableFuture<Void>
thenApply(Function<T, U>)	CompletableFuture<U>
thenApplyAsync(Function<T, U>)	CompletableFuture<U>
thenCombine(CompletionStage<U>, BiFunction<T, U, V>)	CompletableFuture<V>
thenCompose(Function<T, CompletionStage<U>>)	CompletableFuture<U>
thenComposeAsync(Function<T, CompletionStage<U>>)	CompletableFuture<U>
whenComplete(BiConsumer<T, Throwable>)	CompletableFuture<T>

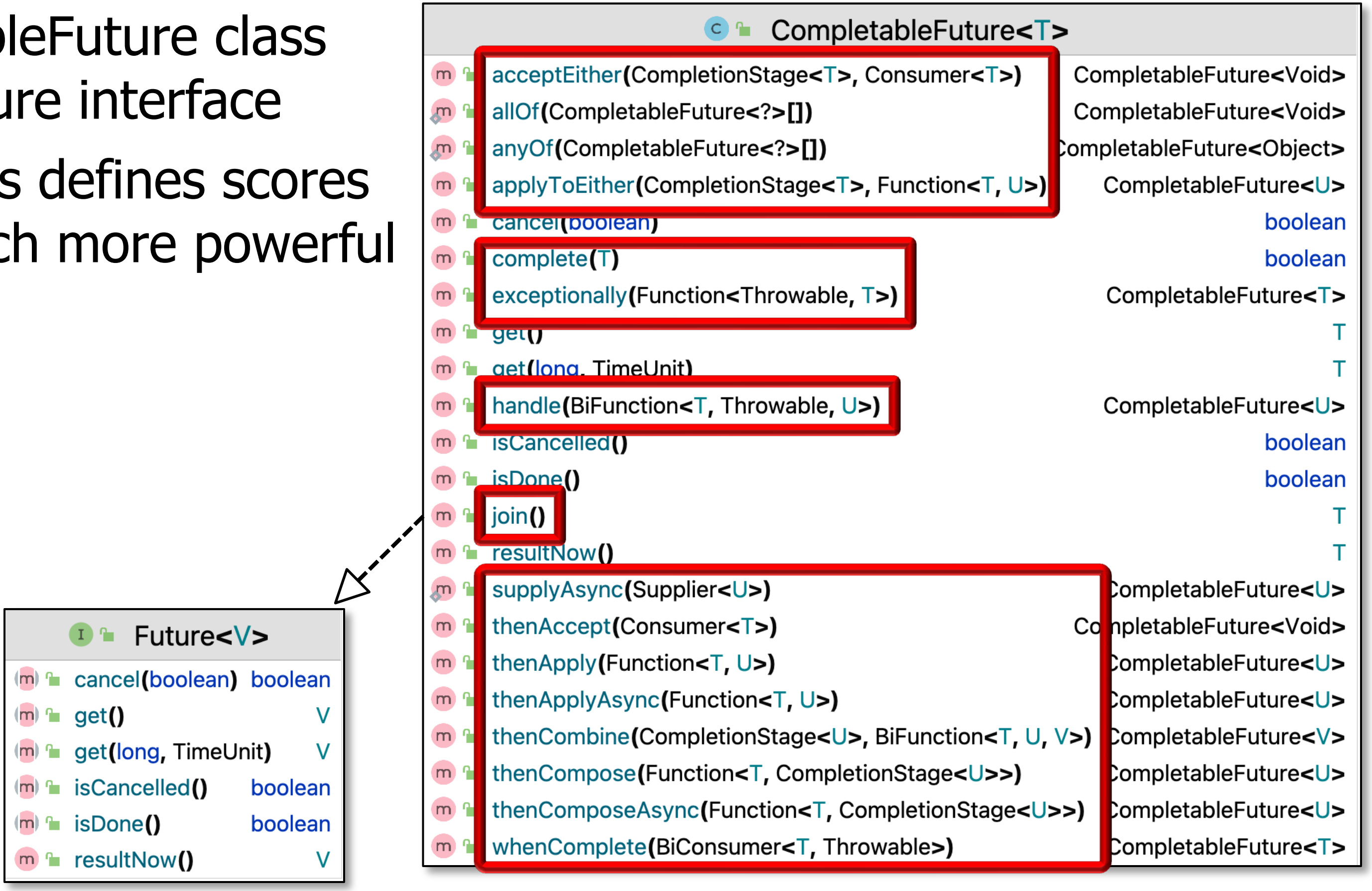
Future<V>

cancel(boolean)	boolean
get()	V
get(long, TimeUnit)	V
isCancelled()	boolean
isDone()	boolean
resultNow()	V

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

Overview of the Modern Java Future API

- The Java CompletableFuture class implements the Future interface
  - However, this class defines scores of methods & much more powerful capabilities



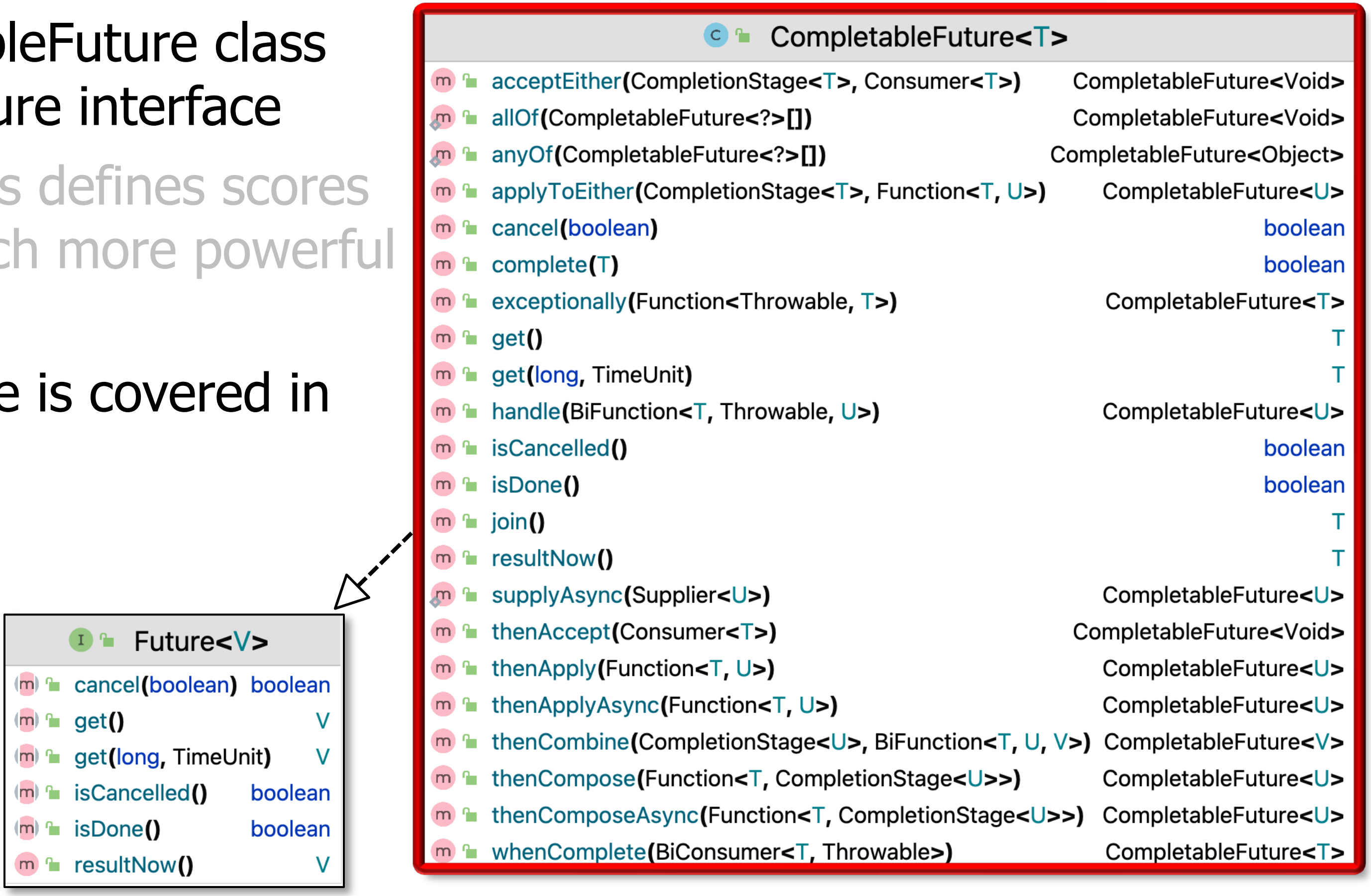
CompletableFuture<T>	
acceptEither(CompletionStage<T>, Consumer<T>)	CompletableFuture<Void>
allOf(CompletableFuture<?>[])	CompletableFuture<Void>
anyOf(CompletableFuture<?>[])	CompletableFuture<Object>
applyToEither(CompletionStage<T>, Function<T, U>)	CompletableFuture<U>
cancel(boolean)	boolean
complete(T)	boolean
exceptionally(Function<Throwable, T>)	CompletableFuture<T>
get()	T
get(long, TimeUnit)	T
handle(BiFunction<T, Throwable, U>)	CompletableFuture<U>
isCancelled()	boolean
isDone()	boolean
join()	T
resultNow()	T
supplyAsync(Supplier<U>)	CompletableFuture<U>
thenAccept(Consumer<T>)	CompletableFuture<Void>
thenApply(Function<T, U>)	CompletableFuture<U>
thenApplyAsync(Function<T, U>)	CompletableFuture<U>
thenCombine(CompletionStage<U>, BiFunction<T, U, V>)	CompletableFuture<V>
thenCompose(Function<T, CompletionStage<U>>)	CompletableFuture<U>
thenComposeAsync(Function<T, CompletionStage<U>>)	CompletableFuture<U>
whenComplete(BiConsumer<T, Throwable>)	CompletableFuture<T>

Future<V>	
cancel(boolean)	boolean
get()	V
get(long, TimeUnit)	V
isCancelled()	boolean
isDone()	boolean
resultNow()	V

See blog.knoldus.com/future-vs-completablefuture-1

Overview of the Modern Java Future API

- The Java CompletableFuture class implements the Future interface
  - However, this class defines scores of methods & much more powerful capabilities
  - CompletableFuture is covered in other courses



CompletableFuture<T>

acceptEither(CompletionStage<T>, Consumer<T>)	CompletableFuture<Void>
allOf(CompletableFuture<?>[])	CompletableFuture<Void>
anyOf(CompletableFuture<?>[])	CompletableFuture<Object>
applyToEither(CompletionStage<T>, Function<T, U>)	CompletableFuture<U>
cancel(boolean)	boolean
complete(T)	boolean
exceptionally(Function<Throwable, T>)	CompletableFuture<T>
get()	T
get(long, TimeUnit)	T
handle(BiFunction<T, Throwable, U>)	CompletableFuture<U>
isCancelled()	boolean
isDone()	boolean
join()	T
resultNow()	T
supplyAsync(Supplier<U>)	CompletableFuture<U>
thenAccept(Consumer<T>)	CompletableFuture<Void>
thenApply(Function<T, U>)	CompletableFuture<U>
thenApplyAsync(Function<T, U>)	CompletableFuture<U>
thenCombine(CompletionStage<U>, BiFunction<T, U, V>)	CompletableFuture<V>
thenCompose(Function<T, CompletionStage<U>>)	CompletableFuture<U>
thenComposeAsync(Function<T, CompletionStage<U>>)	CompletableFuture<U>
whenComplete(BiConsumer<T, Throwable>)	CompletableFuture<T>

Future<V>

cancel(boolean)	boolean
get()	V
get(long, TimeUnit)	V
isCancelled()	boolean
isDone()	boolean
resultNow()	V

See www.dre.vanderbilt.edu/~schmidt/cs253

End of Overview of Java Futures