Overview of Java Futures (Part 1)

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

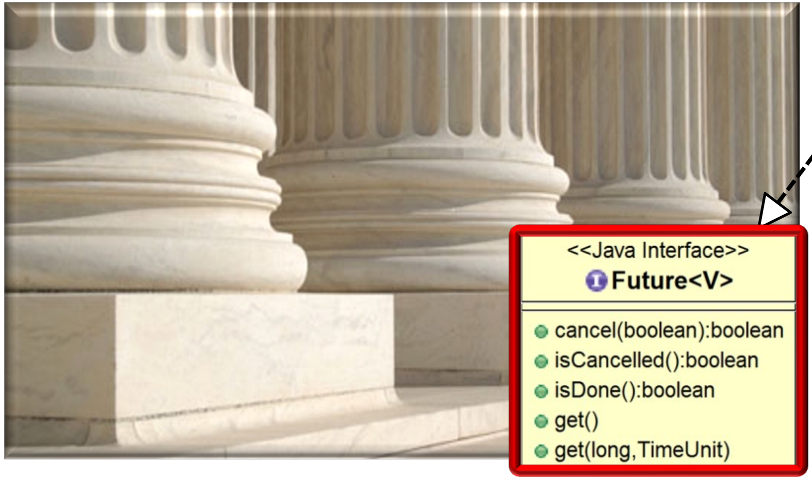
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

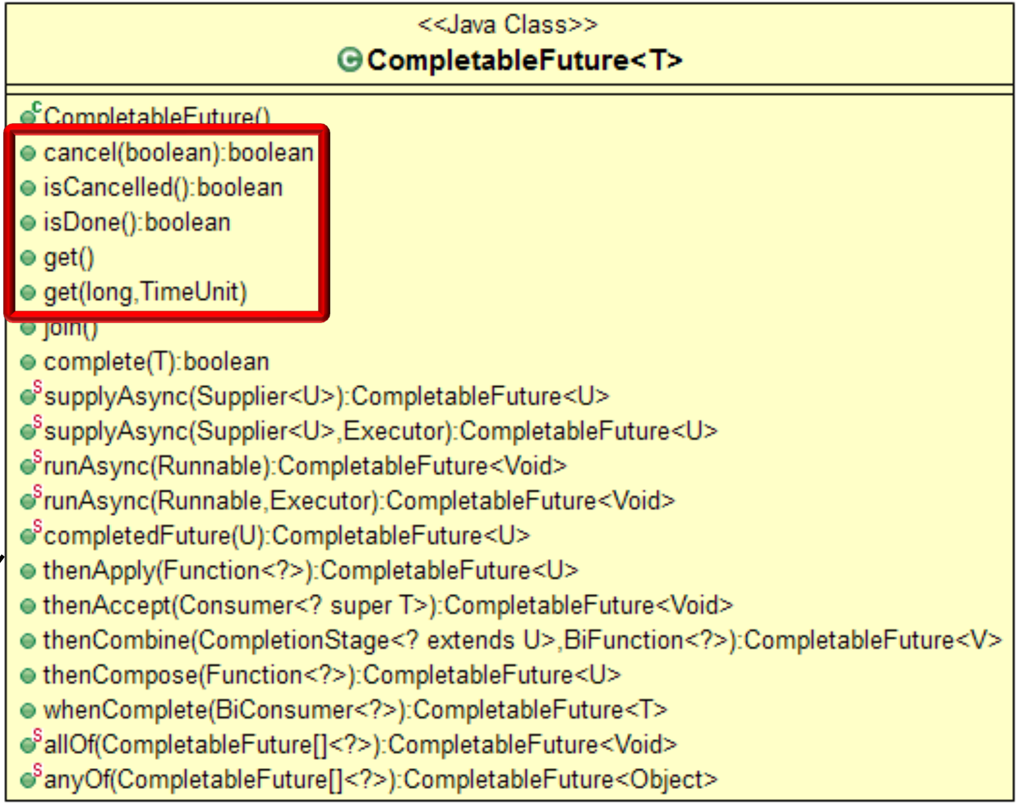
Learning Objectives in this Part of the Lesson

- Motivate the need for Java futures by understanding the pros & cons of synchrony & asynchrony
- Understand that Java futures provide the foundation for Java completable futures



<<Java Interface>>
Future<V>

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



<<Java Class>>
CompletableFuture<T>

- CompletableFuture()
- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long,TimeUnit)
- join()
- complete(T):boolean
- supplyAsync(Supplier<U>):CompletableFuture<U>
- supplyAsync(Supplier<U>,Executor):CompletableFuture<U>
- runAsync(Runnable):CompletableFuture<Void>
- runAsync(Runnable,Executor):CompletableFuture<Void>
- completedFuture(U):CompletableFuture<U>
- thenApply(Function<?>):CompletableFuture<U>
- thenAccept(Consumer<? super T>):CompletableFuture<Void>
- thenCombine(CompletionStage<? extends U>,BiFunction<?>):CompletableFuture<V>
- thenCompose(Function<?>):CompletableFuture<U>
- whenComplete(BiConsumer<?>):CompletableFuture<T>
- allOf(CompletableFuture[]<?>):CompletableFuture<Void>
- anyOf(CompletableFuture[]<?>):CompletableFuture<Object>

See en.wikipedia.org/wiki/Java_version_history

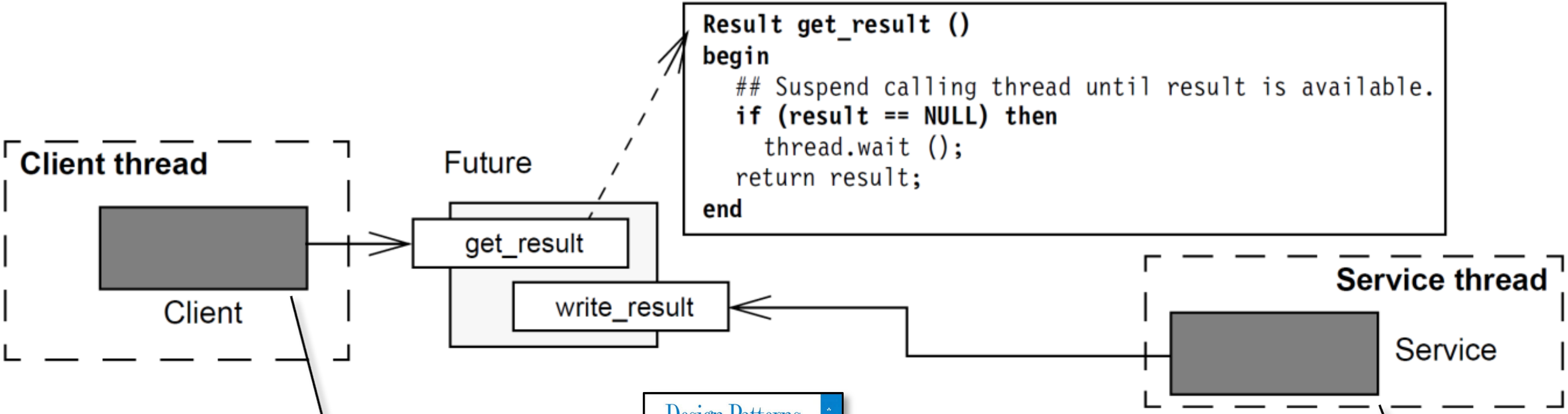
Learning Objectives in this Part of the Lesson

- Motivate the need for Java futures by understanding the pros & cons of synchrony & asynchrony
- Understand that Java futures provide the foundation for Java completable futures
  - Recognize a human known use of Java futures

A Human Known Use of Java Futures

A Human Known Use of Java Futures

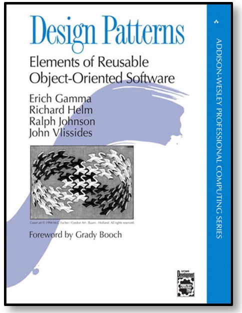
- A future is essentially a proxy that represents the result(s) of an async call

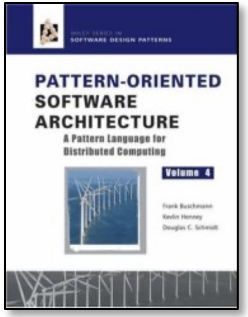


```
Result get_result ()
begin
   ## Suspend calling thread until result is available.
   if (result == NULL) then
     thread.wait ();
   return result;
end
```

1. The async computation runs

2. Result obtained only after async computation completes

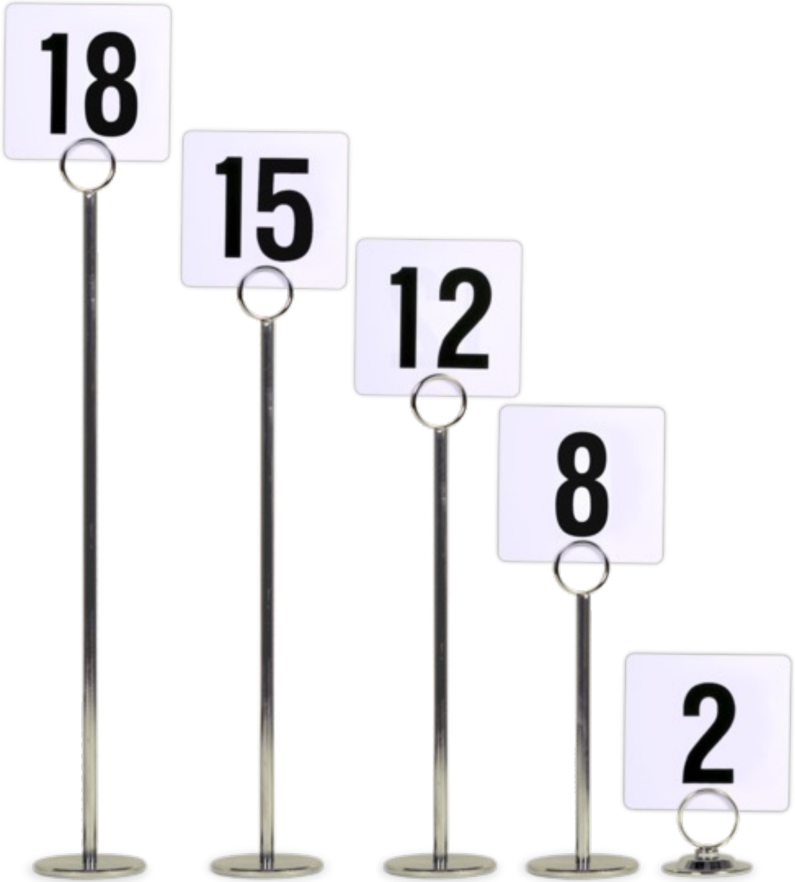

Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES


PATTERN-ORIENTED SOFTWARE ARCHITECTURE
A Pattern Language for Distributed Computing
Volume 4

See en.wikipedia.org/wiki/Proxy_pattern & en.wikipedia.org/wiki/Futures_and_promises

A Human Known Use of Java Futures

- Table tent #'s & table # stands a human-known-use of futures in restaurants!



See www.citygrafx.com/table-numbers-table-markers

A Human Known Use of Java Futures

- Table tent #'s & table # stands a human-known-use of futures in restaurants!
  - e.g., McDonald's vs Wendy's model of preparing fast food

A Human Known Use of Java Futures

- Table tent #'s & table # stands a human-known-use of futures in restaurants!
  - e.g., McDonald's vs Wendy's model of preparing fast food

McDonald's historically 'cached' food in heatlamps & performed "synchronous" transactions

See www.latimes.com/archives/la-xpm-1998-mar-27-fi-33150-story.html

A Human Known Use of Java Futures

- Table tent #'s & table # stands a human-known-use of futures in restaurants!
  - e.g., McDonald's vs Wendy's model of preparing fast food

QUALITY IS OUR RECIPE

WENDY'S®

OLD FASHIONED HAMBURGERS®

Wendy's historically cooked food to order & performed "asynchronous" transactions with various futures

See www.wendys.com/csr-what-we-value/food/quality/fresh

End of Overview of Java Futures (Part 1)

Overview of Java Futures (Part 2)

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

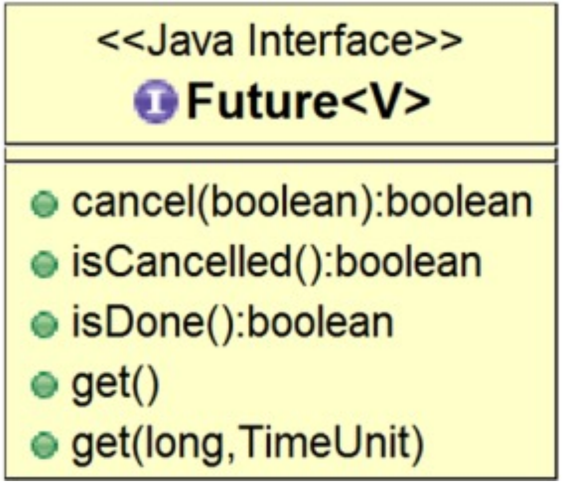
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Motivate the need for Java futures by understanding the pros & cons of synchrony & asynchrony
- Understand that Java futures provide the foundation for Java completable futures
  - Recognize a human known use of Java futures
  - Know all the methods in the Future interface



<<Java Interface>>

Future<V>

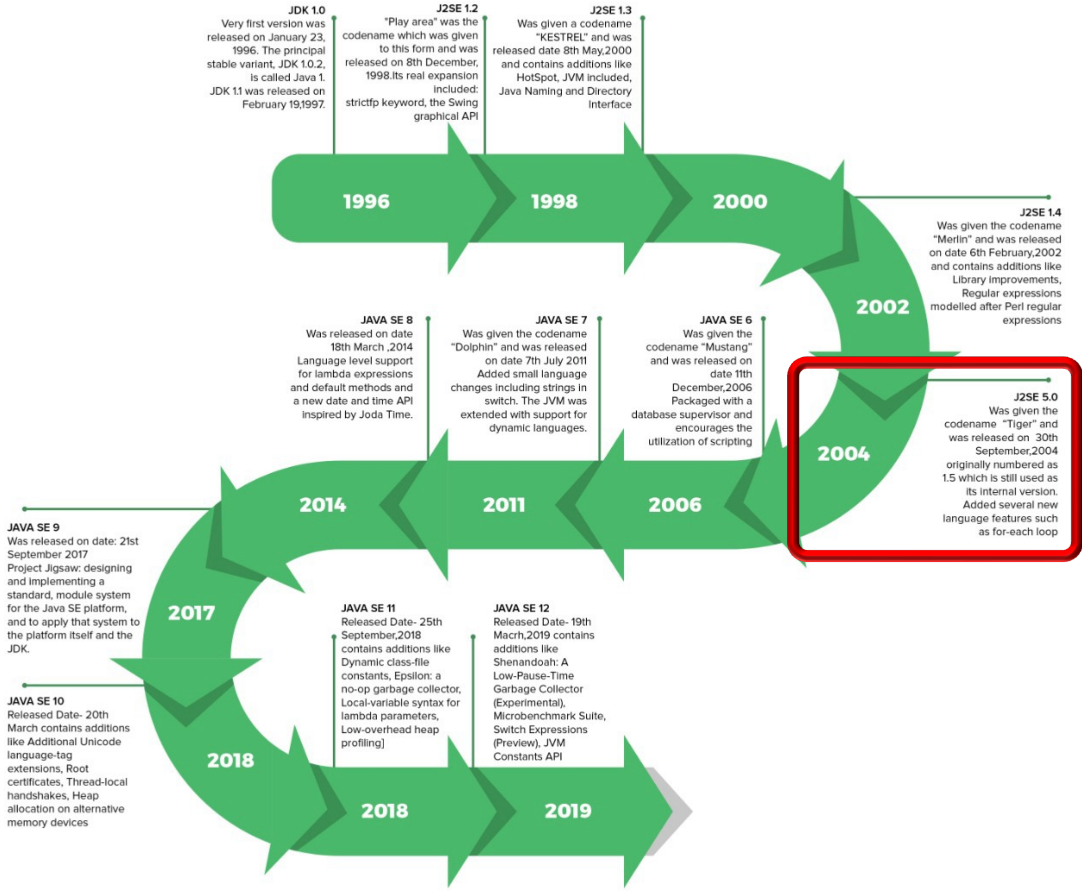
- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long,TimeUnit)

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html

Overview of the Java Future API

Overview of the Java Future API

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



JDK 1.0
Very first version was released on January 23, 1996. The principal stable variant, JDK 1.0.2, is called Java 1. JDK 1.1 was released on February 19,1997.

J2SE 1.2
"Play area" was the codename which was given to this form and was released on 8th December, 1998.Its real expansion included: strictfp keyword, the Swing graphical API

J2SE 1.3
Was given a codename "KESTREL" and was released date 8th May,2000 and contains additions like HotSpot, JVM included, Java Naming and Directory Interface

J2SE 1.4
Was given the codename "Merlin" and was released on date 6th February,2002 and contains additions like Library improvements, Regular expressions modelled after Perl regular expressions

J2SE 5.0
Was given the codename "Tiger" and was released on 30th September,2004 originally numbered as 1.5 which is still used as its internal version. Added several new language features such as for-each loop

JAVA SE 6
Was given the codename "Mustang" and was released on date 11th December,2006 Packaged with a database supervisor and encourages the utilization of scripting

JAVA SE 7
Was given the codename "Dolphin" and was released on date 7th July 2011 Added small language changes including strings in switch. The JVM was extended with support for dynamic languages.

JAVA SE 8
Was released on date 18th March ,2014 Language level support for lambda expressions and default methods and a new date and time API inspired by Joda Time.

JAVA SE 9
Was released on date: 21st September 2017 Project Jigsaw: designing and implementing a standard, module system for the Java SE platform, and to apply that system to the platform itself and the JDK.

JAVA SE 10
Released Date- 20th March contains additions like Additional Unicode language-tag extensions, Root certificates, Thread-local handshakes, Heap allocation on alternative memory devices

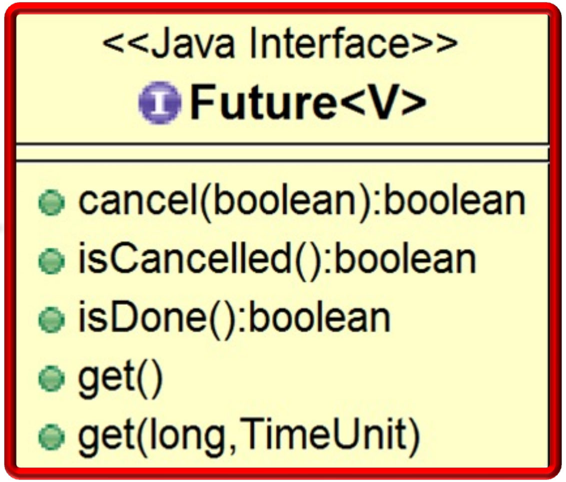
JAVA SE 11
Released Date- 25th September,2018 contains additions like Dynamic class-file constants, Epsilon: a no-op garbage collector, Local-variable syntax for lambda parameters, Low-overhead heap profiling]

JAVA SE 12
Released Date- 19th Macrh,2019 contains additions like Shenandoah: A Low-Pause-Time Garbage Collector (Experimental), Microbenchmark Suite, Switch Expressions (Preview), JVM Constants API

See www.geeksforgeeks.org/the-complete-history-of-java-programming-language

Overview of the Java Future API

- A **Future** represents the result of an asynchronous computation



<<Java Interface>>

Future<V>

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

Methods are provided to check if the computation is complete, to wait for its completion, & to retrieve the result

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html

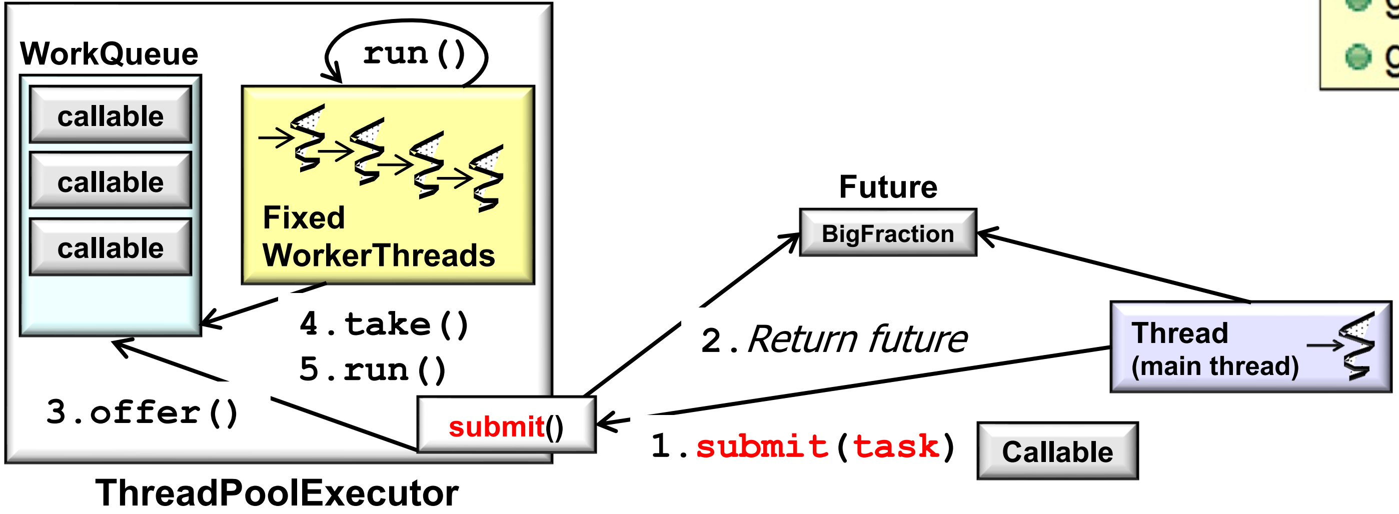
Overview of the Java Future API

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

<<Java Interface>>

Future<V>

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



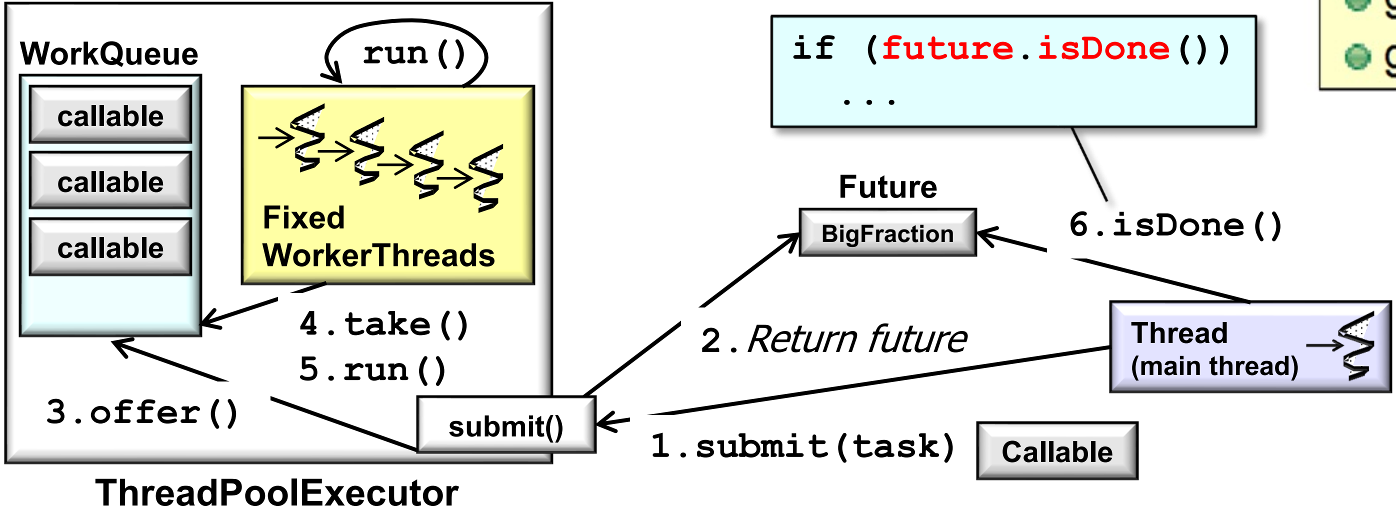
Overview of the Java Future API

- Java Future methods can manage a task's lifecycle after it's submitted to run asynchronously, e.g.
  - A future can be tested for completion

<<Java Interface>>

Future<V>

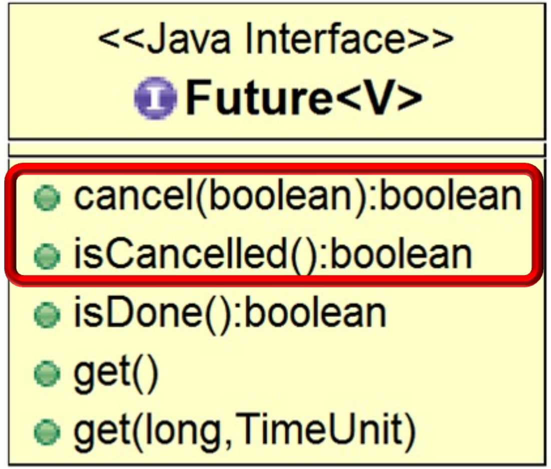
- cancel(boolean):boolean
- isCancelled():boolean
- **isDone():boolean**
- get()
- get(long,TimeUnit)

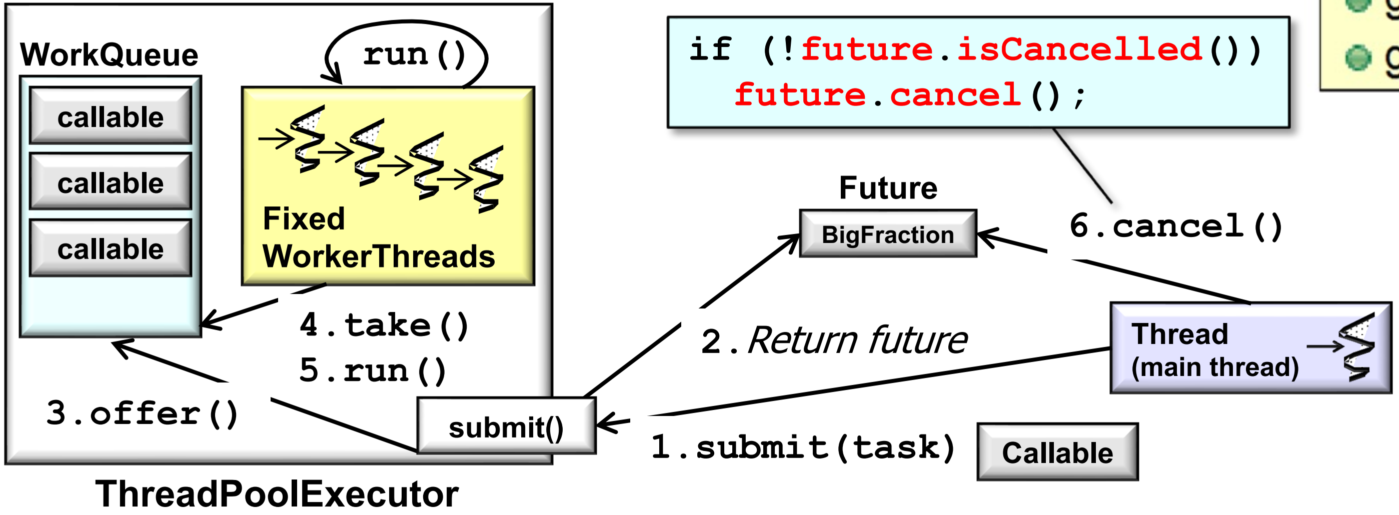


```
if (future.isDone())
  ...
```

Overview of the Java Future API

- Java Future methods can manage a task's lifecycle after it's submitted to run asynchronously, e.g.
  - A future can be tested for completion
  - A future be tested for cancellation & cancelled





```
if (!future.isCancelled())
  future.cancel();
```

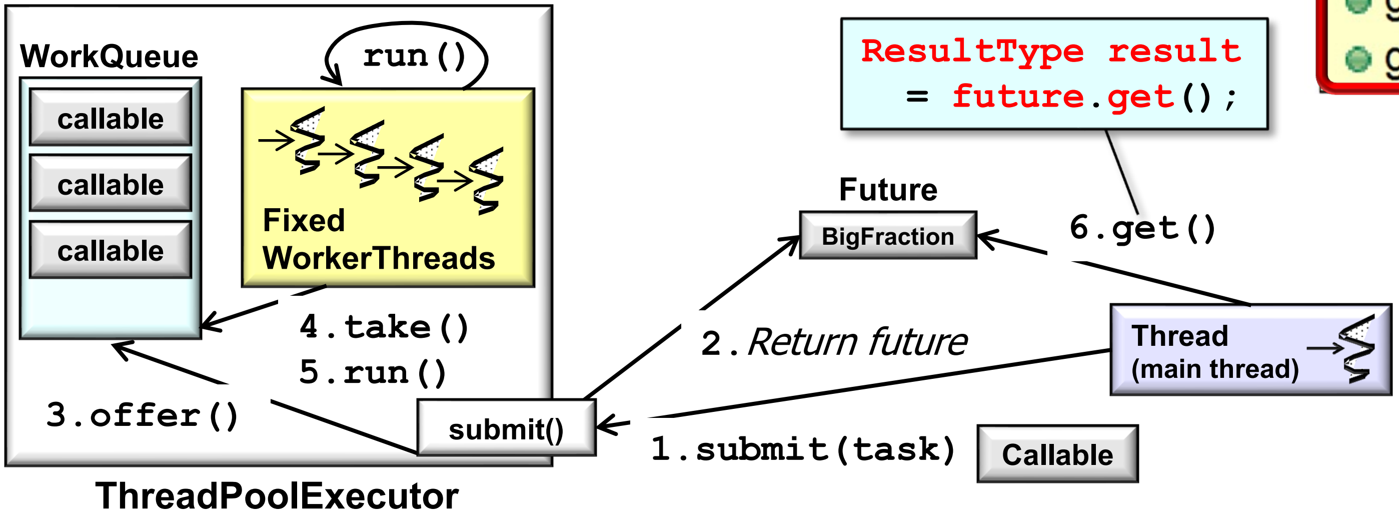
Overview of the Java Future API

- Java Future methods can manage a task’s lifecycle after it’s submitted to run asynchronously, e.g.
  - A future can be tested for completion
  - A future be tested for cancellation & cancelled
  - A future can retrieve a two-way task’s result

<<Java Interface>>

Future<V>

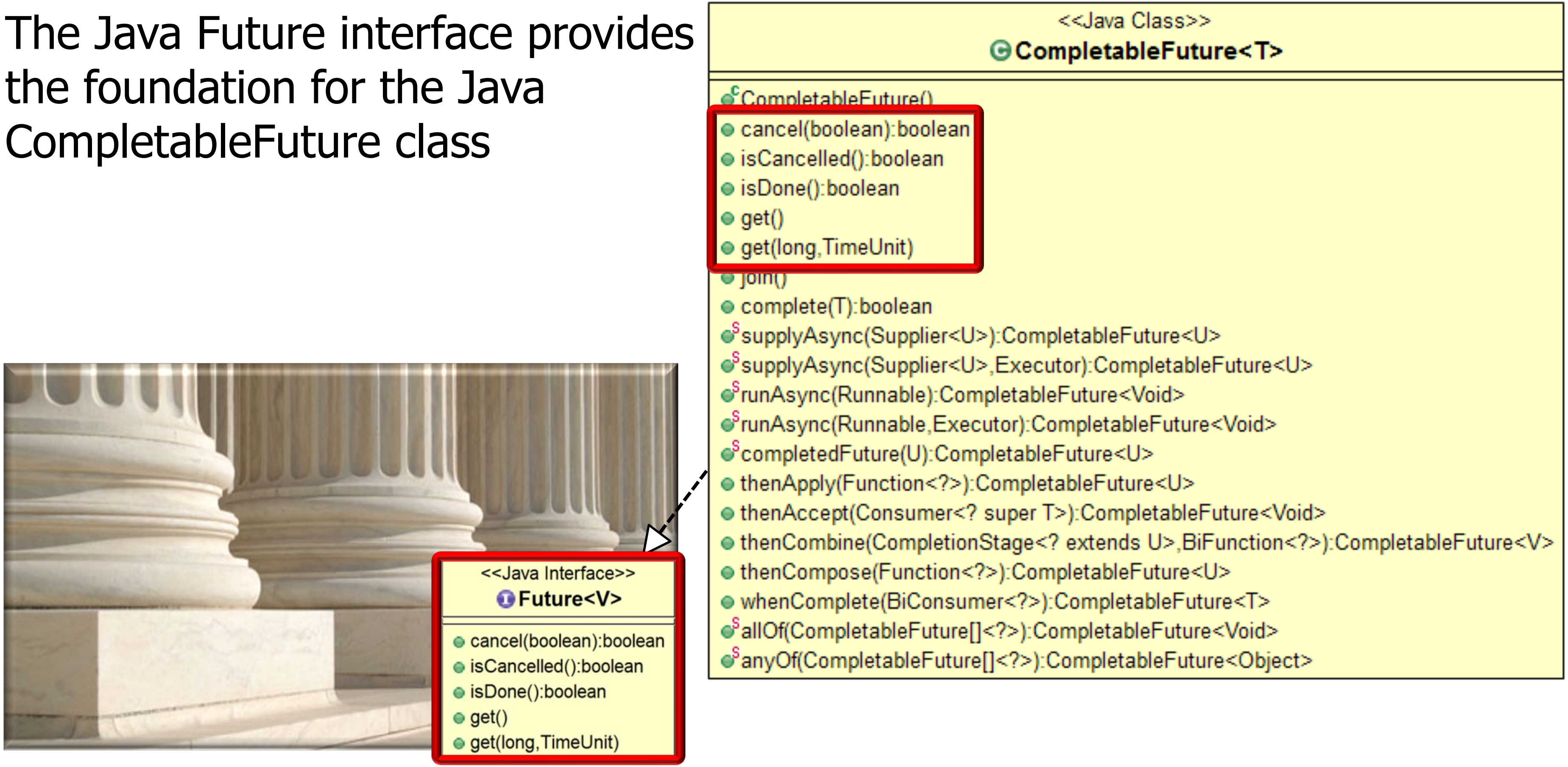
- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long,TimeUnit)



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

Overview of the Java Future API

- The Java Future interface provides the foundation for the Java CompletableFuture class



<<Java Class>>

CompletableFuture<T>

- CompletableFuture()
- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long,TimeUnit)
- join()
- complete(T):boolean
- supplyAsync(Supplier<U>):CompletableFuture<U>
- supplyAsync(Supplier<U>,Executor):CompletableFuture<U>
- runAsync(Runnable):CompletableFuture<Void>
- runAsync(Runnable,Executor):CompletableFuture<Void>
- completedFuture(U):CompletableFuture<U>
- thenApply(Function<?>):CompletableFuture<U>
- thenAccept(Consumer<? super T>):CompletableFuture<Void>
- thenCombine(CompletionStage<? extends U>,BiFunction<?>):CompletableFuture<V>
- thenCompose(Function<?>):CompletableFuture<U>
- whenComplete(BiConsumer<?>):CompletableFuture<T>
- allOf(CompletableFuture[]<?>):CompletableFuture<Void>
- anyOf(CompletableFuture[]<?>):CompletableFuture<Object>

<<Java Interface>>

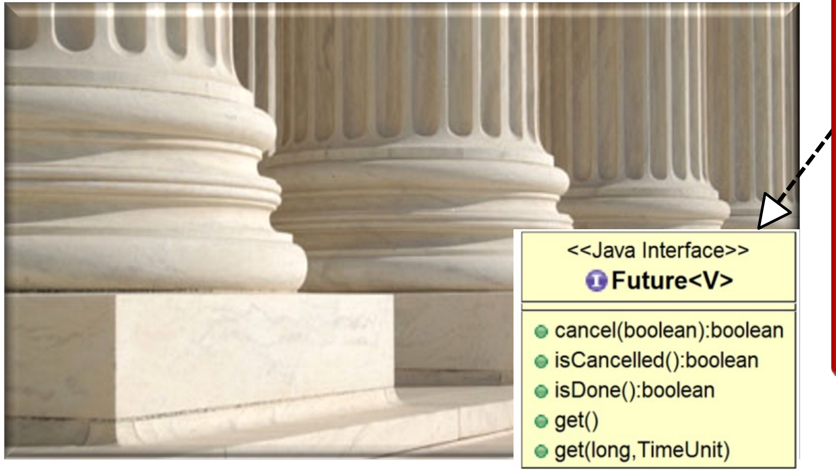
Future<V>

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

See en.wikipedia.org/wiki/Java_version_history

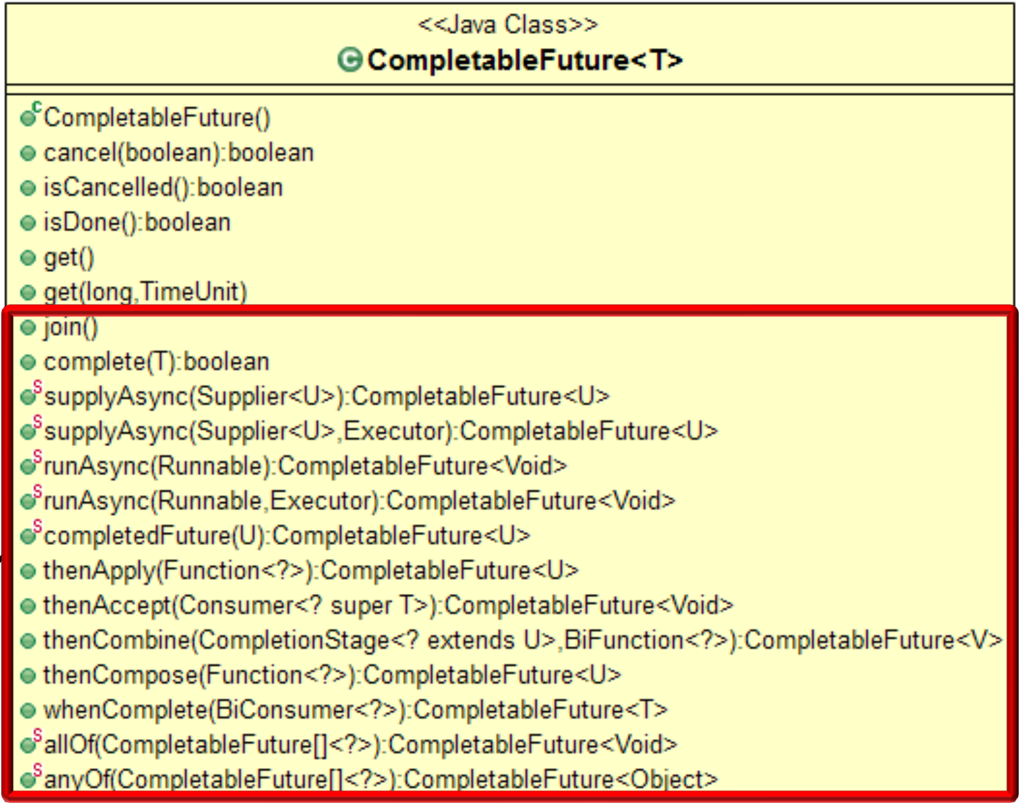
Overview of the Java Future API

- The Java Future interface provides the foundation for the Java CompletableFuture class
  - However, the CompletableFuture class defines dozens of methods & more powerful capabilities



<<Java Interface>>
Future<V>

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



<<Java Class>>
CompletableFuture<T>

- CompletableFuture()
- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long,TimeUnit)
- join()
- complete(T):boolean
- supplyAsync(Supplier<U>):CompletableFuture<U>
- supplyAsync(Supplier<U>,Executor):CompletableFuture<U>
- runAsync(Runnable):CompletableFuture<Void>
- runAsync(Runnable,Executor):CompletableFuture<Void>
- completedFuture(U):CompletableFuture<U>
- thenApply(Function<?>):CompletableFuture<U>
- thenAccept(Consumer<? super T>):CompletableFuture<Void>
- thenCombine(CompletionStage<? extends U>,BiFunction<?>):CompletableFuture<V>
- thenCompose(Function<?>):CompletableFuture<U>
- whenComplete(BiConsumer<?>):CompletableFuture<T>
- allOf(CompletableFuture[]<?>):CompletableFuture<Void>
- anyOf(CompletableFuture[]<?>):CompletableFuture<Object>

See upcoming lessons on the completable futures framework

End of Overview of Java Futures (Part 2)