Basic Java CompletableFuture Features

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

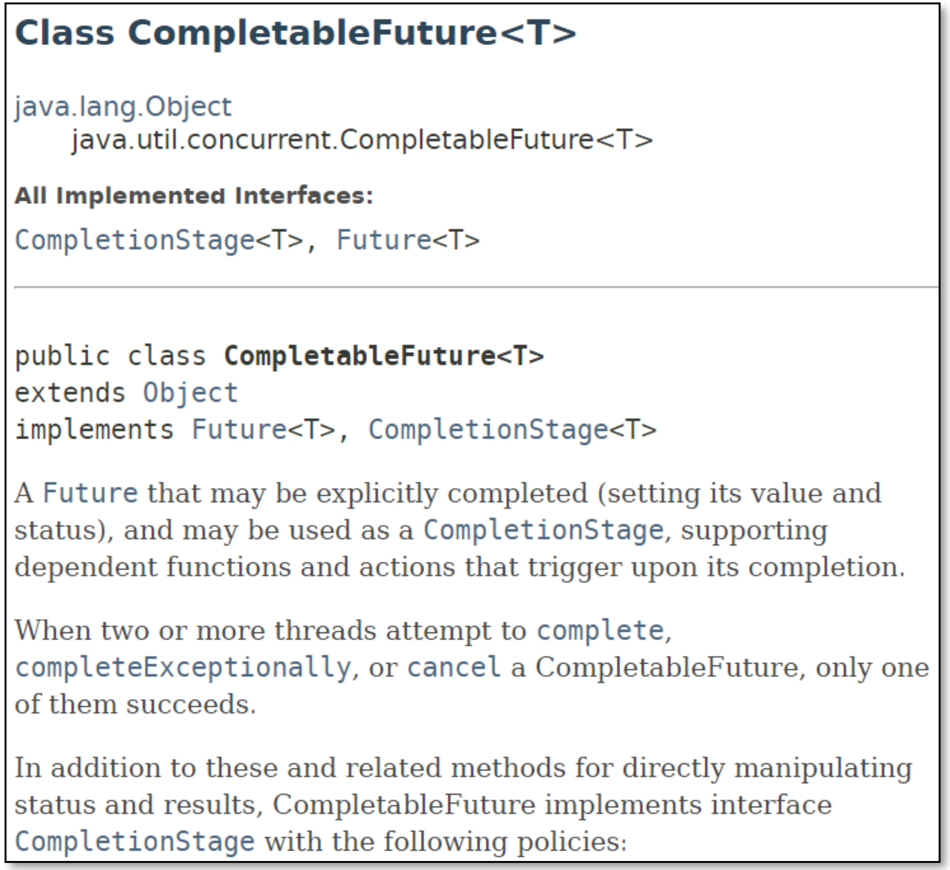
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand the basic completable futures features



Class CompletableFuture<T>

java.lang.Object
    java.util.concurrent.CompletableFuture<T>

All Implemented Interfaces:

CompletionStage<T>, Future<T>

```
public class CompletableFuture<T>
extends Object
implements Future<T>, CompletionStage<T>
```

A Future that may be explicitly completed (setting its value and status), and may be used as a CompletionStage, supporting dependent functions and actions that trigger upon its completion.

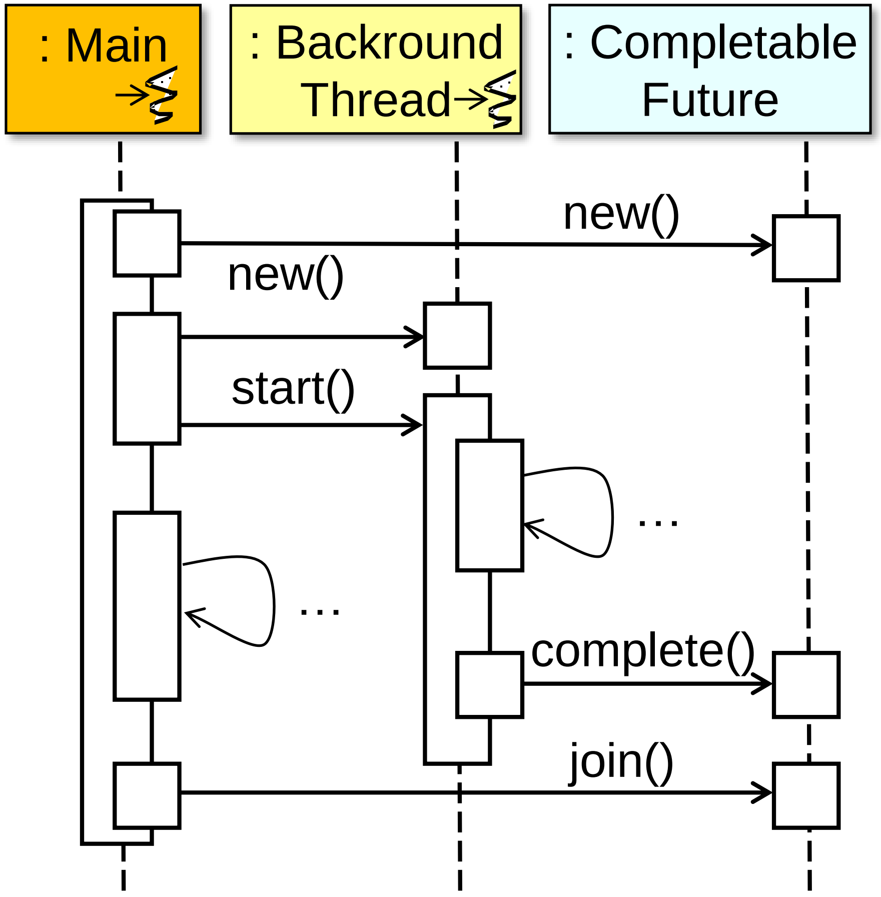
When two or more threads attempt to complete, completeExceptionally, or cancel a CompletableFuture, only one of them succeeds.

In addition to these and related methods for directly manipulating status and results, CompletableFuture implements interface CompletionStage with the following policies:

Basic CompletableFuture Features

Basic CompletableFuture Features

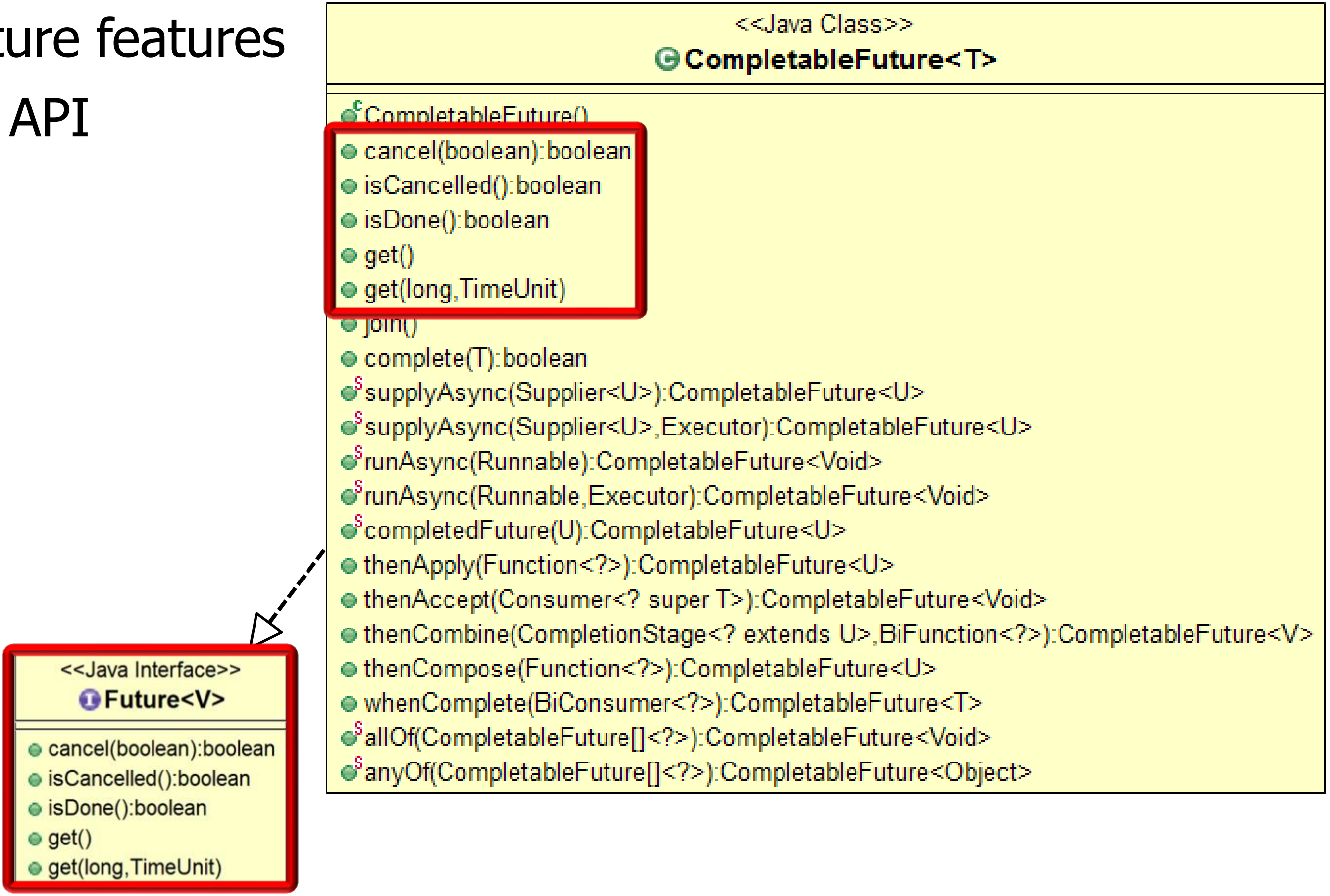
- Basic CompletableFuture features



See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8

Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API



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

Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API
    - Can (time-) block & poll

```
String f1 = "62675744/15668936";
String f2 = "609136/913704";

ForkJoinTask<BigFraction> f =
  commonPool().submit(() -> {
    BigFraction bf1 =
      new BigFraction(f1);
    BigFraction bf2 =
      new BigFraction(f2);
    return bf1.multiply(bf2);
  });
...
BigFraction result = f.get();
// f.get(10, MILLISECONDS);
// f.get(0, 0);
```

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

Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API
    - Can (time-) block & poll
    - Can be cancelled & tested if cancelled/done

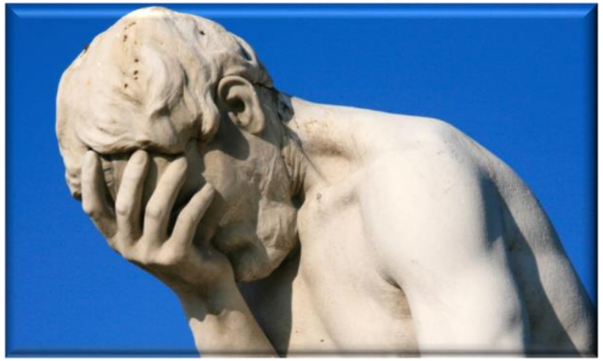
```
String f1 = "62675744/15668936";
String f2 = "609136/913704";

ForkJoinTask<BigFraction> f =
  commonPool().submit(() -> {
    BigFraction bf1 =
      new BigFraction(f1);
    BigFraction bf2 =
      new BigFraction(f2);
    return bf1.multiply(bf2);
  });
 ...
 if (!(f.isDone()
       || !f.isCancelled()))
   f.cancel();
```

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

Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API
    - Can (time-) block & poll
    - Can be cancelled & tested if cancelled/done
      - cancel() doesn't interrupt the computation by default..



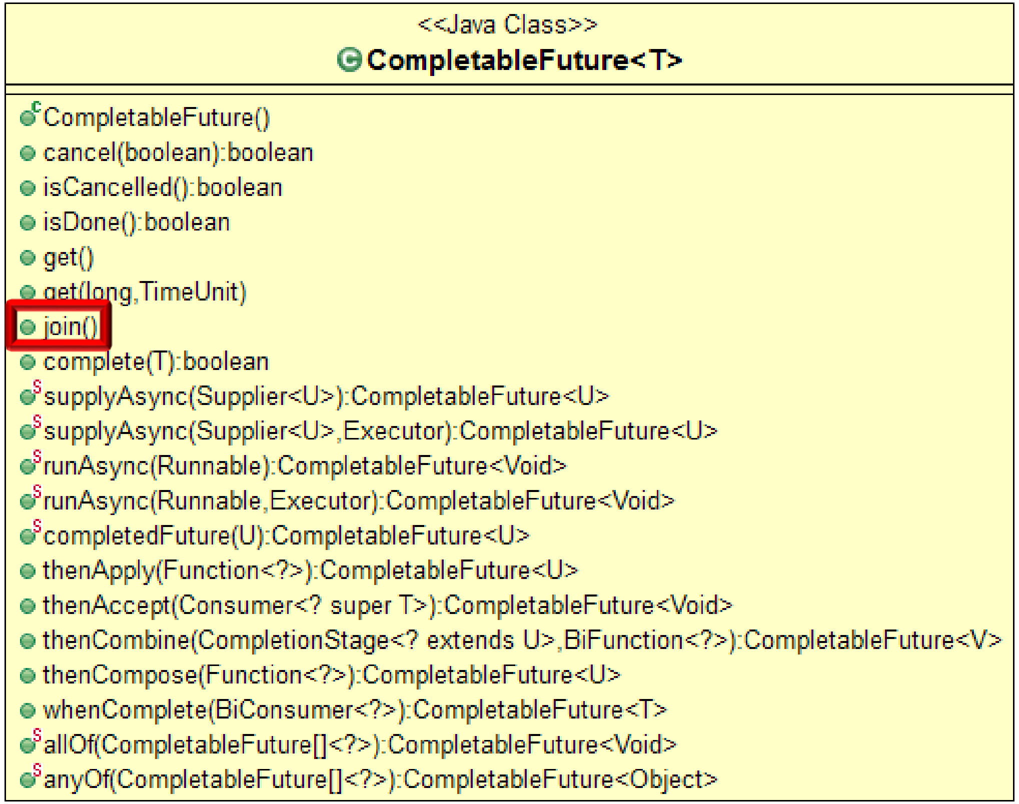
```
String f1 = "62675744/15668936";
String f2 = "609136/913704";

ForkJoinTask<BigFraction> f =
   commonPool().submit(() -> {
     BigFraction bf1 =
       new BigFraction(f1);
     BigFraction bf2 =
       new BigFraction(f2);
     return bf1.multiply(bf2);
   });
 ...
 if (!(f.isDone()
         || !f.isCancelled()))
   f.cancel();
```

See www.nurkiewicz.com/2015/03/completablefuture-cant-be-interrupted.html

Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API
  - Define a join() method



<<Java Class>>
CompletableFuture<T>

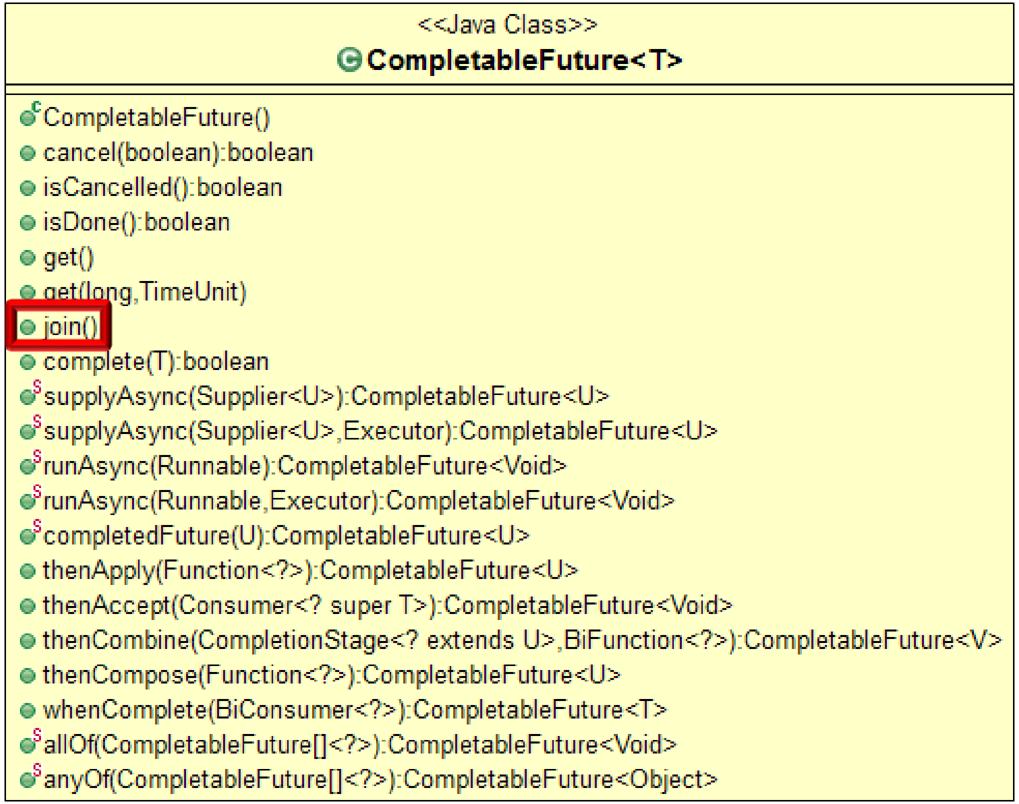
- CompletableFuture()
- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long,TimeUnit)
- join()
- complete(T):boolean
- supplyAsync(Supplier<U>):CompletableFuture<U>
- supplyAsync(Supplier<U>,Executor):CompletableFuture<U>
- runAsync(Runnable):CompletableFuture<Void>
- 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 docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#join

Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API
  - Define a join() method
    - Behaves like get() *without* using checked exceptions

```
futures
   .stream()
   .map(CompletableFuture
          ::join)
   .collect(toList())
```


<<Java Class>>
CompletableFuture<T>

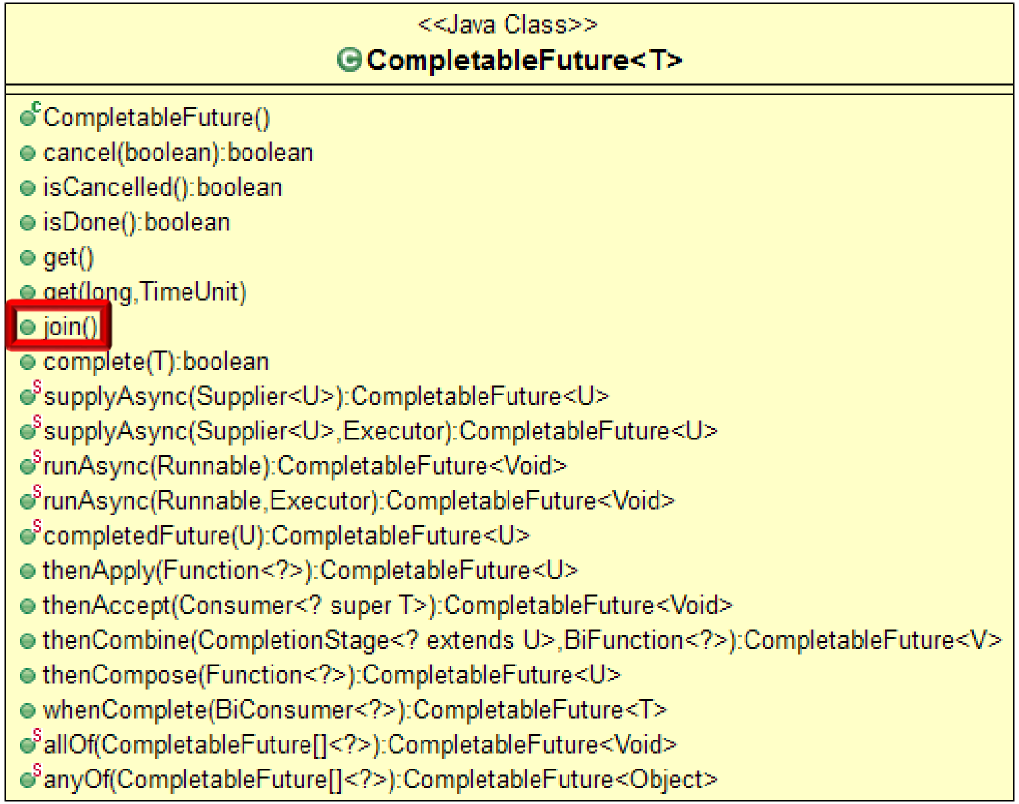
- CompletableFuture()
- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long,TimeUnit)
- join()
- complete(T):boolean
- supplyAsync(Supplier<U>):CompletableFuture<U>
- supplyAsync(Supplier<U>,Executor):CompletableFuture<U>
- runAsync(Runnable):CompletableFuture<Void>
- 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>

CompletableFuture::join can be used as a method reference in a Java stream

Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API
  - Define a join() method
    - Behaves like get() *without* using checked exceptions

```
futures
  .stream()
  .map(future
    -> try { future.get();
    } catch (Exception e){
    })
  .collect(toList())
```



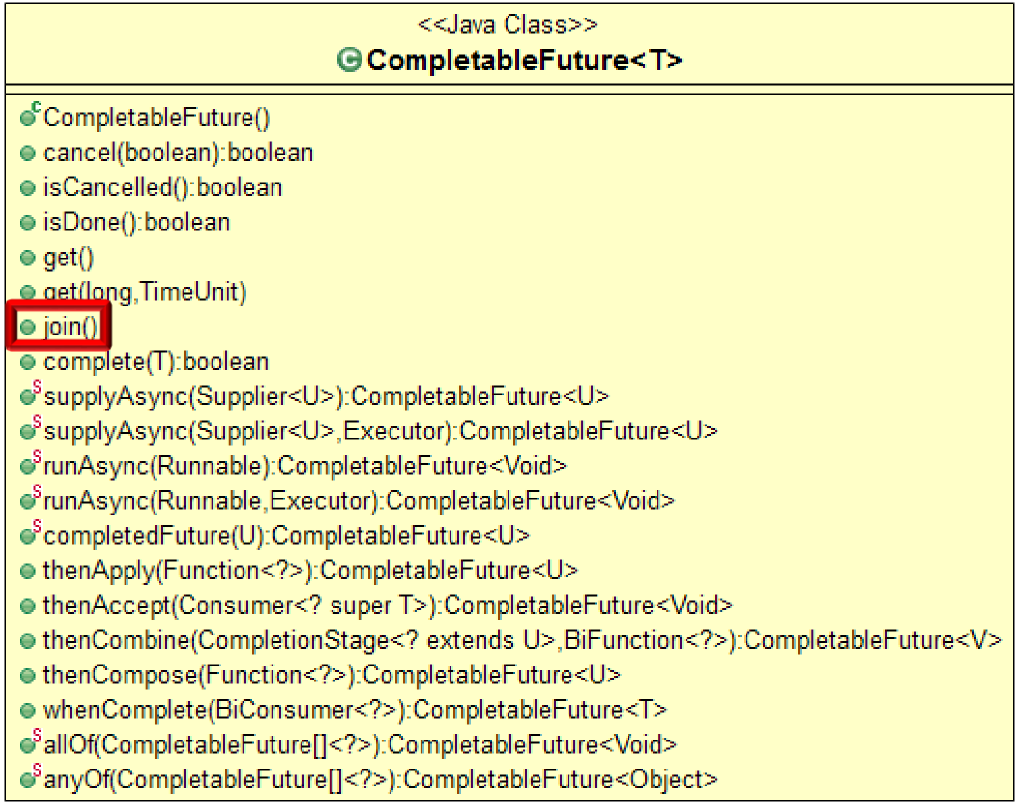
<<Java Class>>
CompletableFuture<T>

- 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>

Mixing checked exceptions & Java streams is ugly..

Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API
  - Define a join() method
    - Behaves like get() *without* using checked exceptions
    - There is no timed version of join()

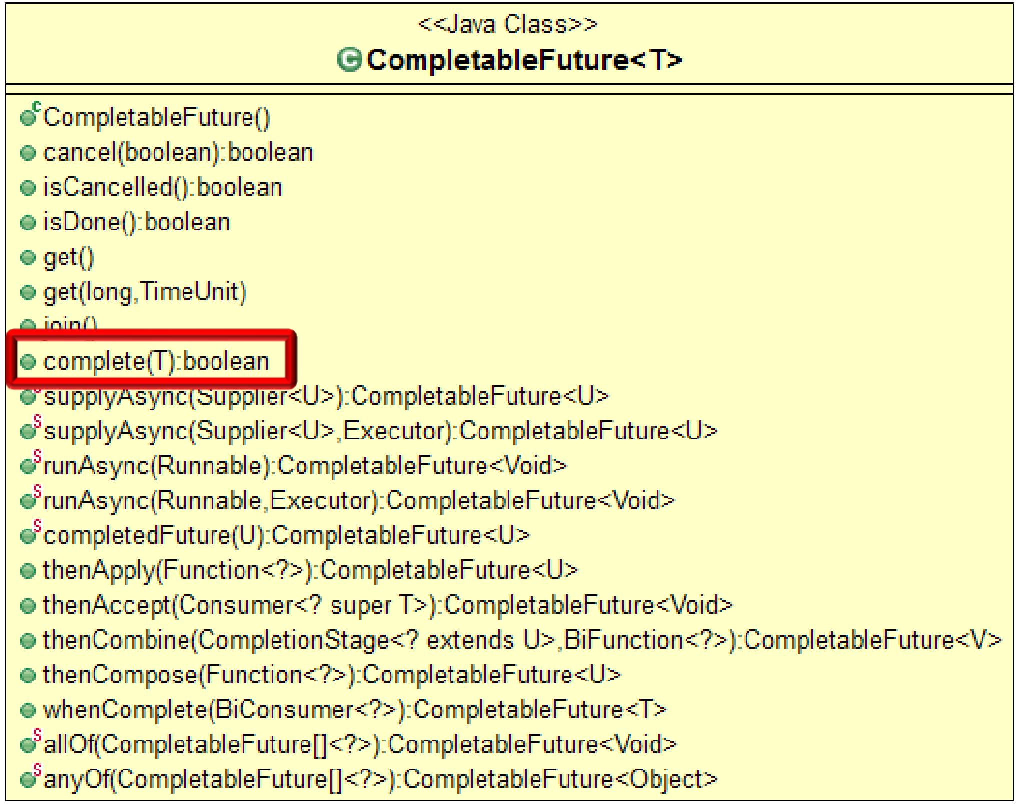


```
<<Java Class>>
CompletableFuture<T>

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>
```

Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API
  - Define a join() method
  - Can be completed explicitly


<<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 docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#complete

Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API
  - Define a join() method
  - Can be completed explicitly
    - i.e., sets result returned by get()/join() to a given value

```
CompletableFuture<...> future =
  new CompletableFuture<>();

new Thread (() -> {
  ...
  future.complete(...);
}).start();

...
System.out.println(future.join());
```

Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API
  - Define a join() method
  - Can be completed explicitly
    - i.e., sets result returned by get()/join() to a given value

```
CompletableFuture<...> future =
  new CompletableFuture<>();

new Thread (() -> {
    ...
    future.complete(...);
}).start();

...
System.out.println(future.join());
```

Create an incomplete future

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

Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API
  - Define a join() method
  - Can be completed explicitly
    - i.e., sets result returned by get()/join() to a given value

```
CompletableFuture<...> future =
  new CompletableFuture<>();

new Thread (() -> {
   ...
   future.complete(...);
}).start();

...
System.out.println(future.join());
```

Create/start a new thread that runs concurrently with the main thread

See docs.oracle.com/javase/8/docs/api/java/lang/Thread.html

Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API
  - Define a join() method
  - Can be completed explicitly
    - i.e., sets result returned by get()/join() to a given value

```
CompletableFuture<...> future =
  new CompletableFuture<>();

new Thread (() -> {
  ...
  future.complete(...);
}).start();

...
System.out.println(future.join());
```

After complete() is done calls to join() will unblock

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#complete

Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API
  - Define a join() method
  - Can be completed explicitly
    - i.e., sets result returned by get()/join() to a given value

```
CompletableFuture<...> future =
  new CompletableFuture<>();

final CompletableFuture<Long> zero
  = CompletableFuture
      .completedFuture(0L);

new Thread (() -> {
  ...
  future.complete(zero.join());
}).start();

...
System.out.println(future.join());
```

A completable future can be initialized to a value/constant

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#completedFuture

End of Basic Java CompletableFuture Features