Advanced Java CompletableFuture Features: Introducing Factory Methods

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

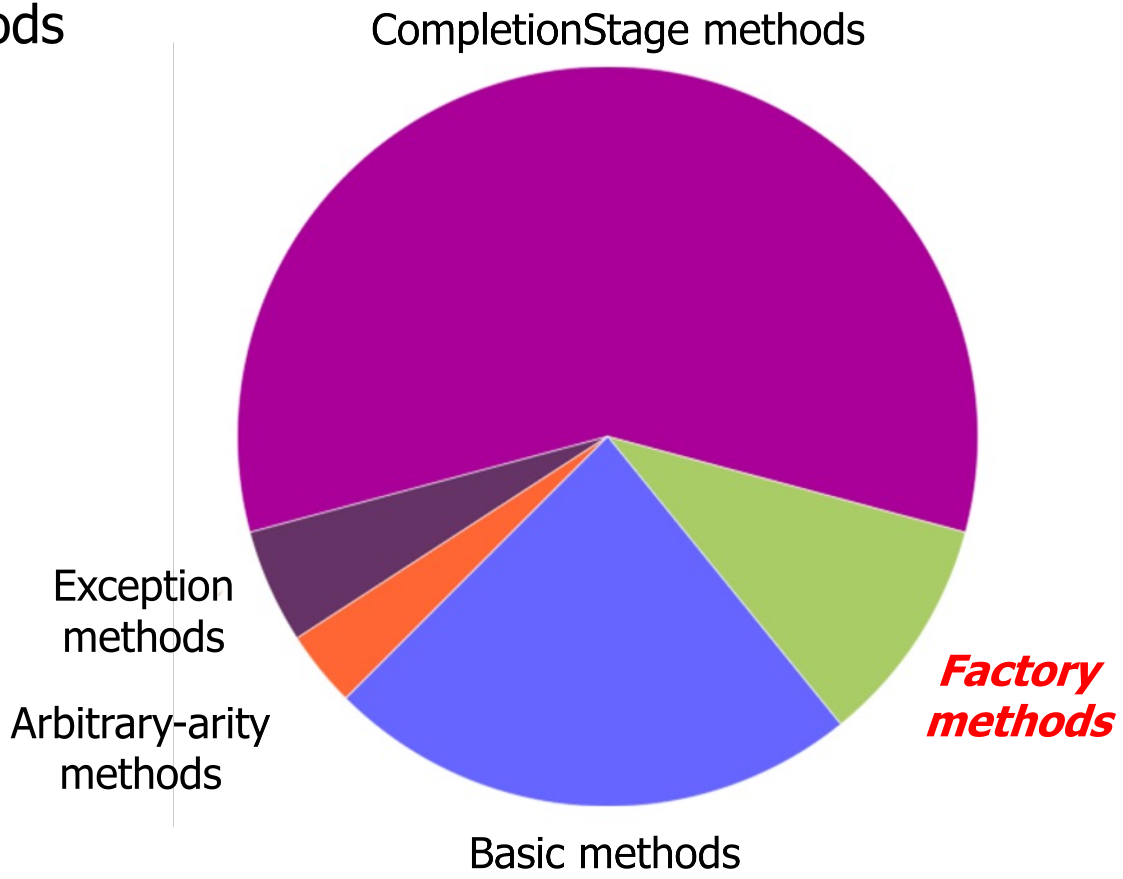
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand how factory methods initiate async computations

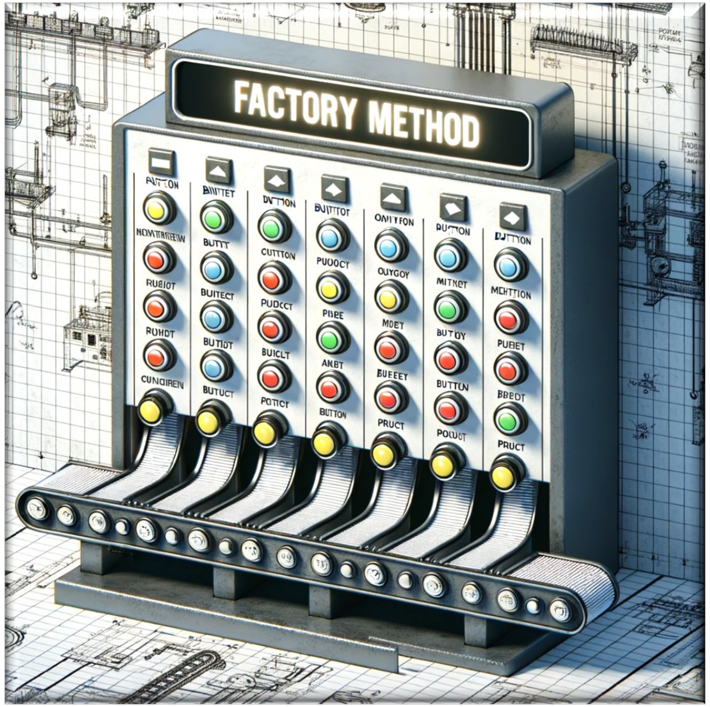


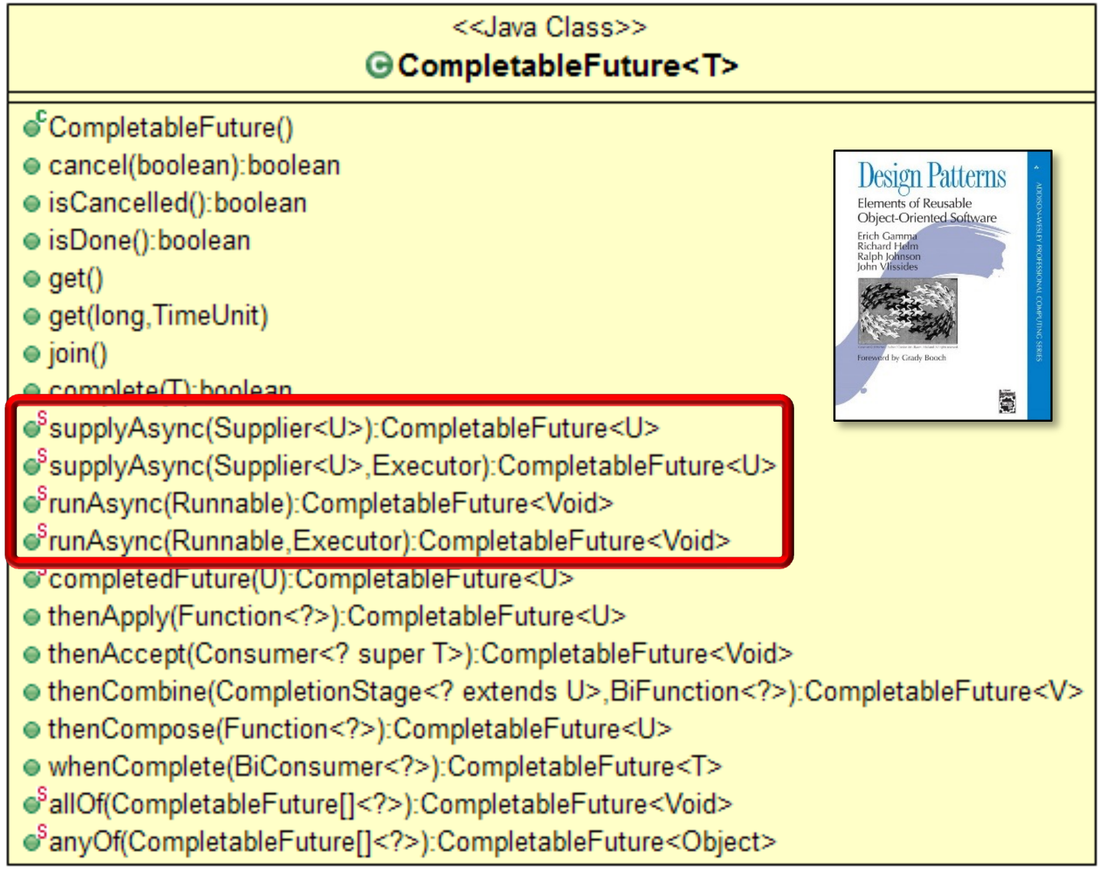
See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html

Factory Methods Initiate Async Computations

Factory Methods Initiate Async Computations

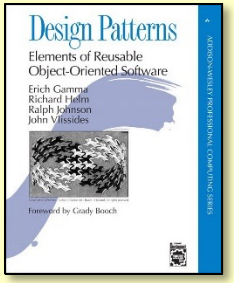
- Four factory methods initiate asynchronous computations





<<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>

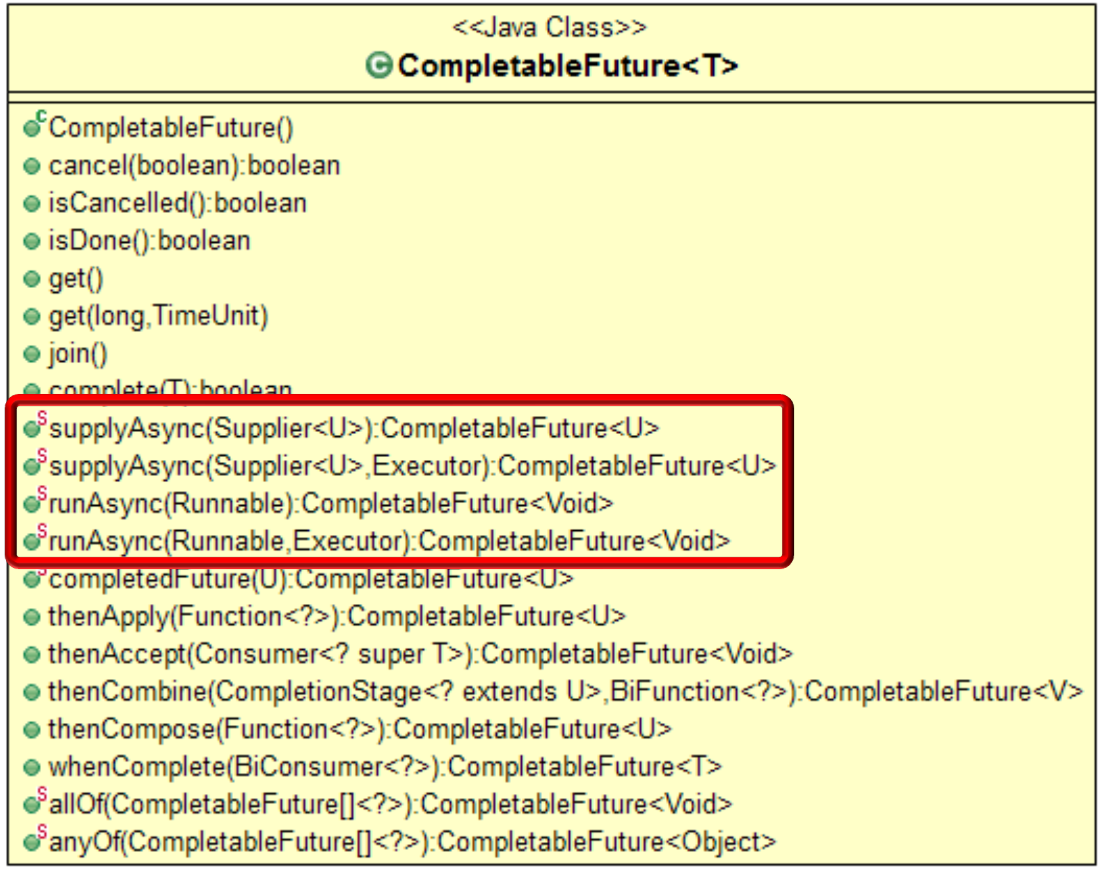


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

See en.wikipedia.org/wiki/Factory_method_pattern

Factory Methods Initiate Async Computations

- Four factory methods initiate asynchronous computations
  - These computations may or may not return a value



```
<<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>
```

Factory Methods Initiate Async Computations

- Four factory methods initiate asynchronous computations
  - These computations may or may not return a value
    - supplyAsync() allows two-way calls via a supplier

Methods	Params	Returns	Behavior
supply Async	Supplier	Completable Future with result of Supplier	Asynchronously run supplier in common fork/ join pool
supply Async	Supplier, Executor	Completable Future with result of Supplier	Asynchronously run supplier in given executor context

See docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html

Factory Methods Initiate Async Computations

- Four factory methods initiate asynchronous computations
  - These computations may or may not return a value
    - supplyAsync() allows two-way calls via a supplier
      - Can be passed params

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

CompletableFuture<BigFraction> future
  = CompletableFuture
      .supplyAsync(() -> {
        BigFraction bf1 =
          new BigFraction(f1);
        BigFraction bf2 =
          new BigFraction(f2);

        return bf1.multiply(bf2);
      });
```

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

Factory Methods Initiate Async Computations

- Four factory methods initiate asynchronous computations
  - These computations may or may not return a value
    - supplyAsync() allows two-way calls via a supplier
      - Can be passed params

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

CompletableFuture<BigFraction> future
  = CompletableFuture
      .supplyAsync(() -> {
        BigFraction bf1 =
          new BigFraction(f1);
        BigFraction bf2 =
          new BigFraction(f2);

        return bf1.multiply(bf2);
      });
```

Params are passed as "effectively final" objects to the supplier lambda

See javarevisited.blogspot.com/2015/03/what-is-effectively-final-variable-of.html

Factory Methods Initiate Async Computations

- Four factory methods initiate asynchronous computations
  - These computations may or may not return a value
    - supplyAsync() allows two-way calls via a supplier
      - Can be passed params
      - Returns a value

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

CompletableFuture<BigFraction> future
  = CompletableFuture
      .supplyAsync(() -> {
        BigFraction bf1 =
          new BigFraction(f1);
        BigFraction bf2 =
          new BigFraction(f2);

        return bf1.multiply(bf2);
      });
```

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

Factory Methods Initiate Async Computations

- Four factory methods initiate asynchronous computations
  - These computations may or may not return a value
    - supplyAsync() allows two-way calls via a supplier
    - runAsync() enables one-way calls via a runnable

Methods	Params	Returns	Behavior
runAsync	Runnable	CompletableFuture with result of Void	Asynchronously run runnable in common fork/join pool
runAsync	Runnable, Executor	CompletableFuture with result of Void	Asynchronously run runnable in given executor context

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

Factory Methods Initiate Async Computations

- Four factory methods initiate asynchronous computations
  - These computations may or may not return a value
    - supplyAsync() allows two-way calls via a supplier
    - runAsync() enables one-way calls via a runnable
      - Can be passed params

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

CompletableFuture<Void> future
  = CompletableFuture
        .runAsync(() -> {
          BigFraction bf1 =
            new BigFraction(f1);
          BigFraction bf2 =
            new BigFraction(f2);

          System.out.println
            (bf1.multiply(bf2)
                .toMixedString());
        });
```

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

Factory Methods Initiate Async Computations

- Four factory methods initiate asynchronous computations
  - These computations may or may not return a value
    - supplyAsync() allows two-way calls via a supplier
    - runAsync() enables one-way calls via a runnable
      - Can be passed params
      - Returns no value

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

CompletableFuture<Void> future
  = CompletableFuture
      .runAsync(() -> {
        BigFraction bf1 =
          new BigFraction(f1);
        BigFraction bf2 =
          new BigFraction(f2);

        System.out.println
          (bf1.multiply(bf2)
              .toMixedString());
      });
```

"Void" is not a value!

See www.baeldung.com/java-void-type

Factory Methods Initiate Async Computations

- Four factory methods initiate asynchronous computations
  - These computations may or may not return a value
    - supplyAsync() allows two-way calls via a supplier
    - runAsync() enables one-way calls via a runnable
      - Can be passed params
      - Returns no value

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

CompletableFuture<Void> future
  = CompletableFuture
      .runAsync(() -> {
        BigFraction bf1 =
          new BigFraction(f1);
        BigFraction bf2 =
          new BigFraction(f2);

        System.out.println
          (bf1.multiply(bf2)
              .toMixedString())
      });
```

Any output must therefore come from "side-effects"

See en.wikipedia.org/wiki/Side_effect_(computer_science)

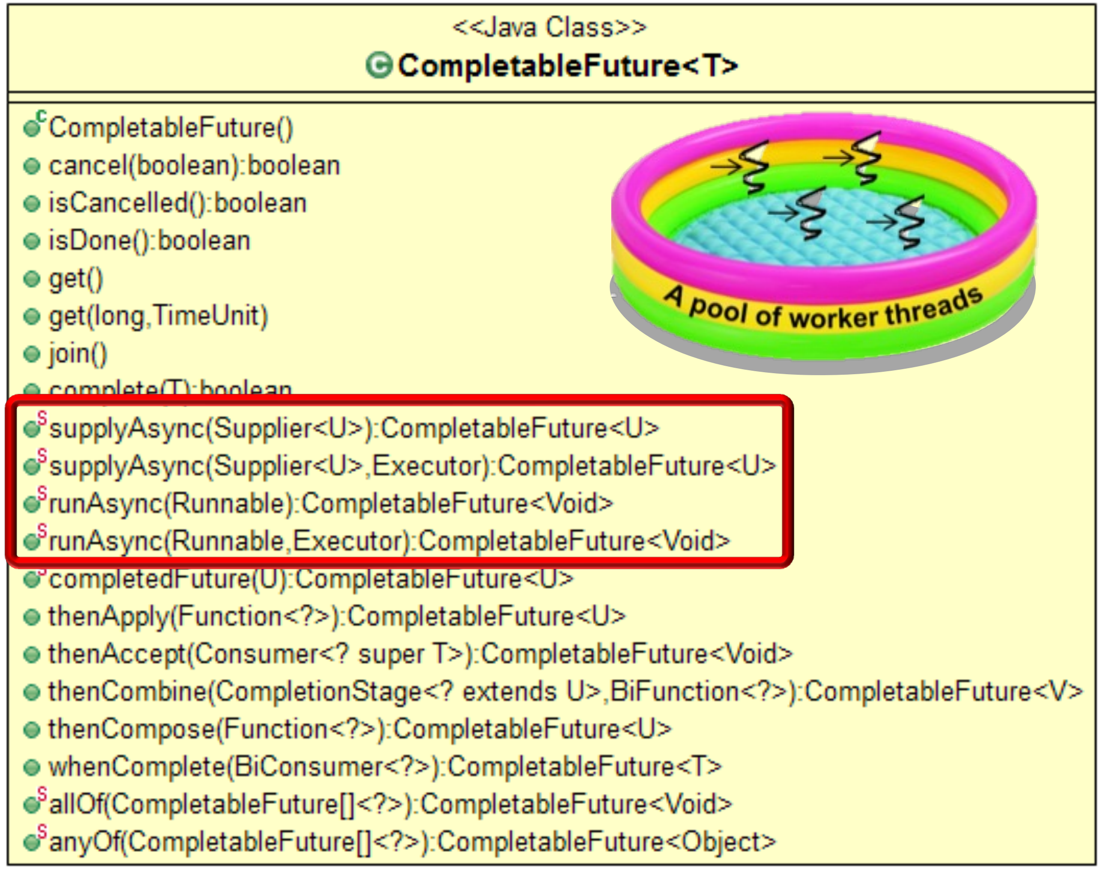
Factory Methods Initiate Async Computations

- Four factory methods initiate asynchronous computations
  - These computations may or may not return a value
    - supplyAsync() allows two-way calls via a supplier
    - runAsync() enables one-way calls via a runnable

supplyAsync() is more commonly used than runAsync() in practice

Factory Methods Initiate Async Computations

- Four factory methods initiate asynchronous computations
  - These computations may or may not return a value
  - Asynchronous functionality runs in a thread pool



<<Java Class>>
CompletableFuture<T>

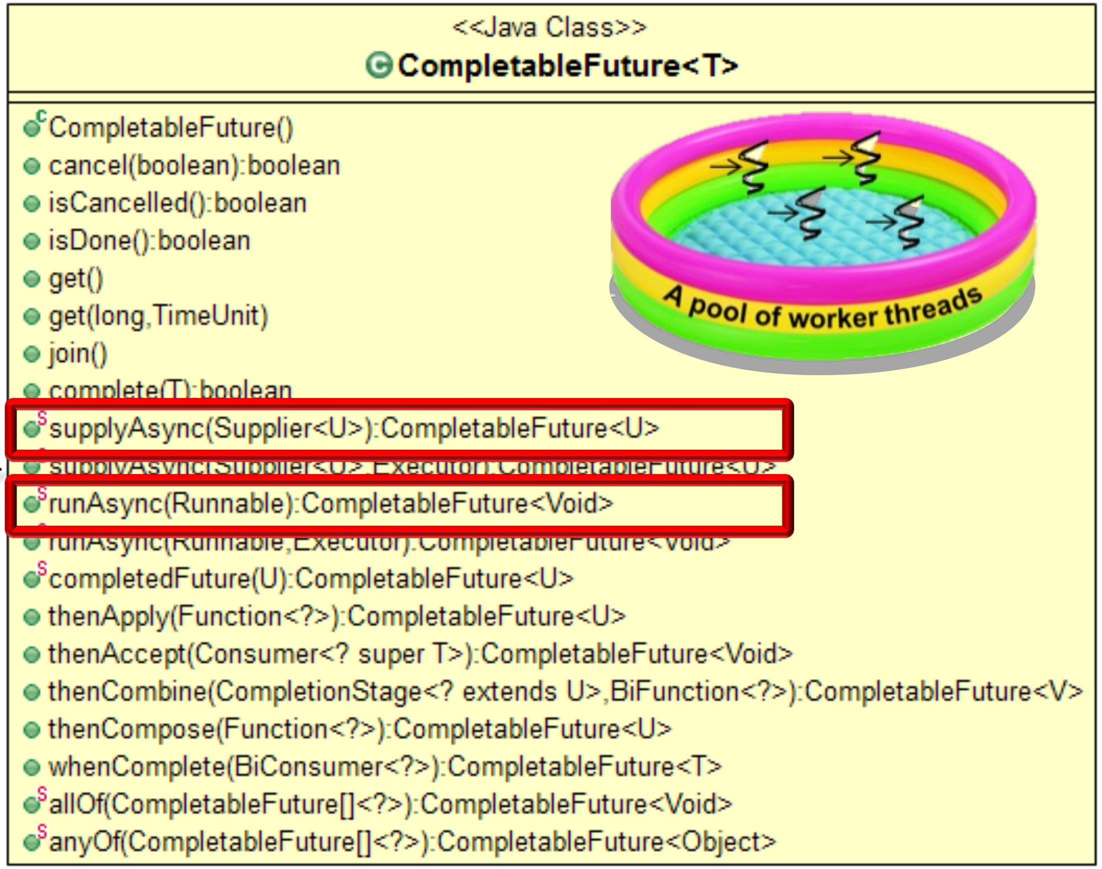
- CompletableFuture()
- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long,TimeUnit)
- join()
- complete(T):boolean
- supplyAsync(Supplier<U>):CompletableFuture<U>
- supplyAsync(Supplier<U>,Executor):CompletableFuture<U>
- runAsync(Runnable):CompletableFuture<Void>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

Help make programs more *elastic* by leveraging a pool of worker threads

Factory Methods Initiate Async Computations

- Four factory methods initiate asynchronous computations
  - These computations may or may not return a value
  - Asynchronous functionality runs in a thread pool

By default, the common fork-join pool is used



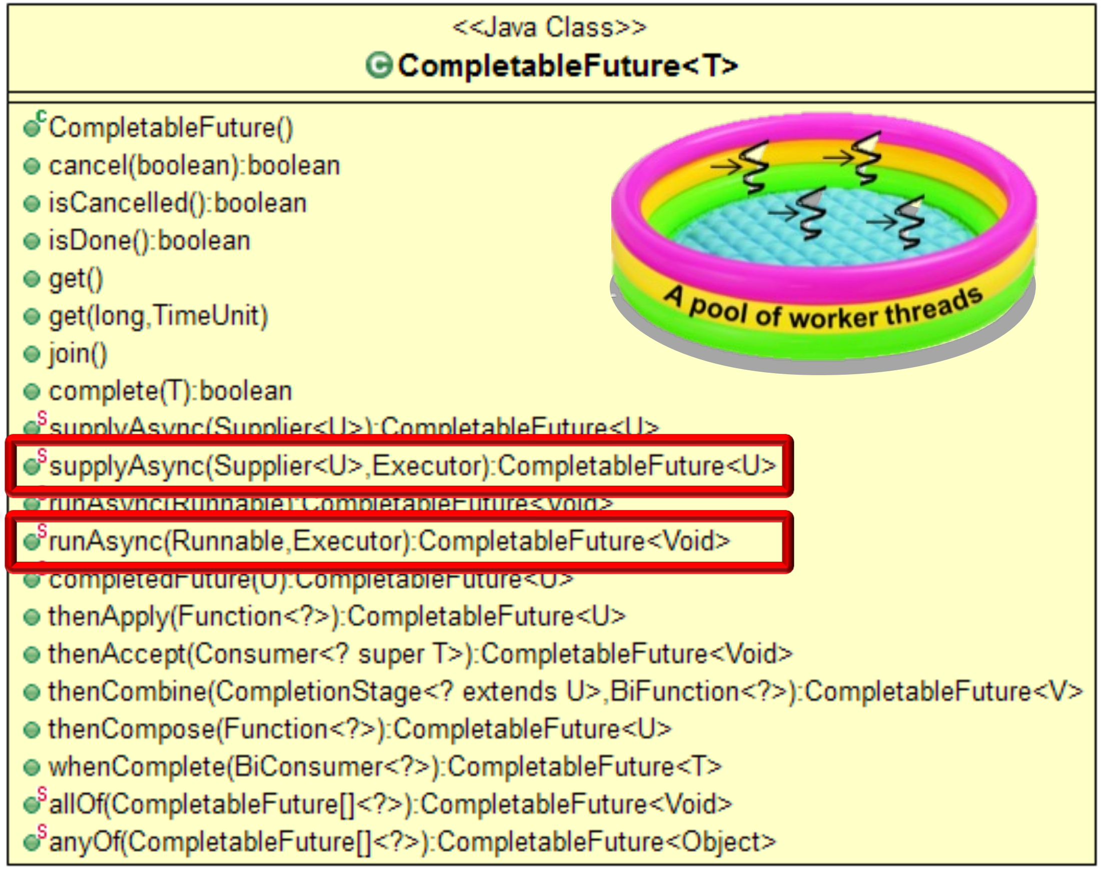
<<Java Class>>
CompletableFuture<T>

```
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 dzone.com/articles/common-fork-join-pool-and-streams

Factory Methods Initiate Async Computations

- Four factory methods initiate asynchronous computations
  - These computations may or may not return a value
  - Asynchronous functionality runs in a thread pool


<<Java Class>>
CompletableFuture<T>

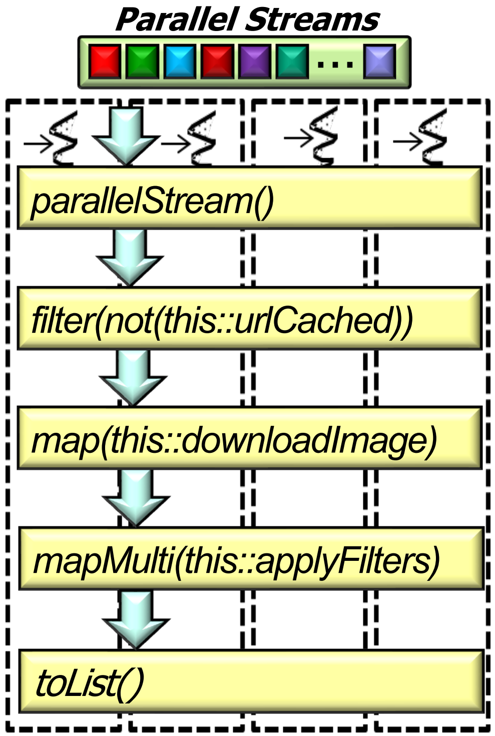
CompletableFuture()
cancel(boolean):boolean
isCancelled():boolean
isDone():boolean
get()
get(long,TimeUnit)
join()
complete(T):boolean
supplyAsync(Supplier<U>):CompletableFuture<U>
supplyAsync(Supplier<U>,Executor):CompletableFuture<U>
runAsync(Runnable):CompletableFuture<Void>
runAsync(Runnable,Executor):CompletableFuture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

However, a pre- or user-defined thread pool can also be given

See www.baeldung.com/thread-pool-java-and-guava

Factory Methods Initiate Async Computations

- Four factory methods initiate asynchronous computations
  - These computations may or may not return a value
  - Asynchronous functionality runs in a thread pool
    - In contrast, Java parallel streams use the common fork-join pool



See lesson on "*Java Parallel Stream Internals: Parallel Processing via the Common Fork-Join Pool*"

End of Advanced Java CompletableFuture Features: Introducing Factory Methods