Overview of Advanced Java 8 CompletableFuture Features (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

- Understand advanced features of completable futures

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.

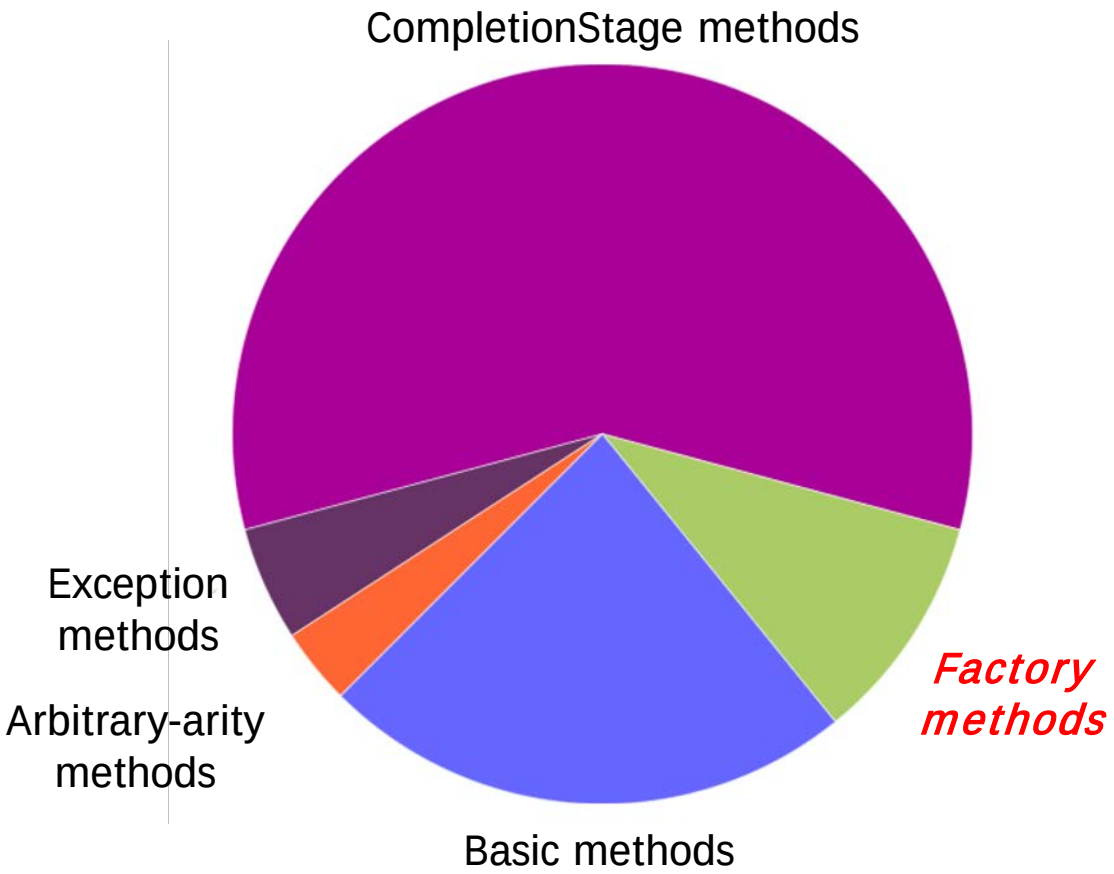
When two or more threads attempt to `complete`, `completeExceptionally`, or `cancel` a CompletableFuture, only one of them succeeds.

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

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

Learning Objectives in this Part of the Lesson

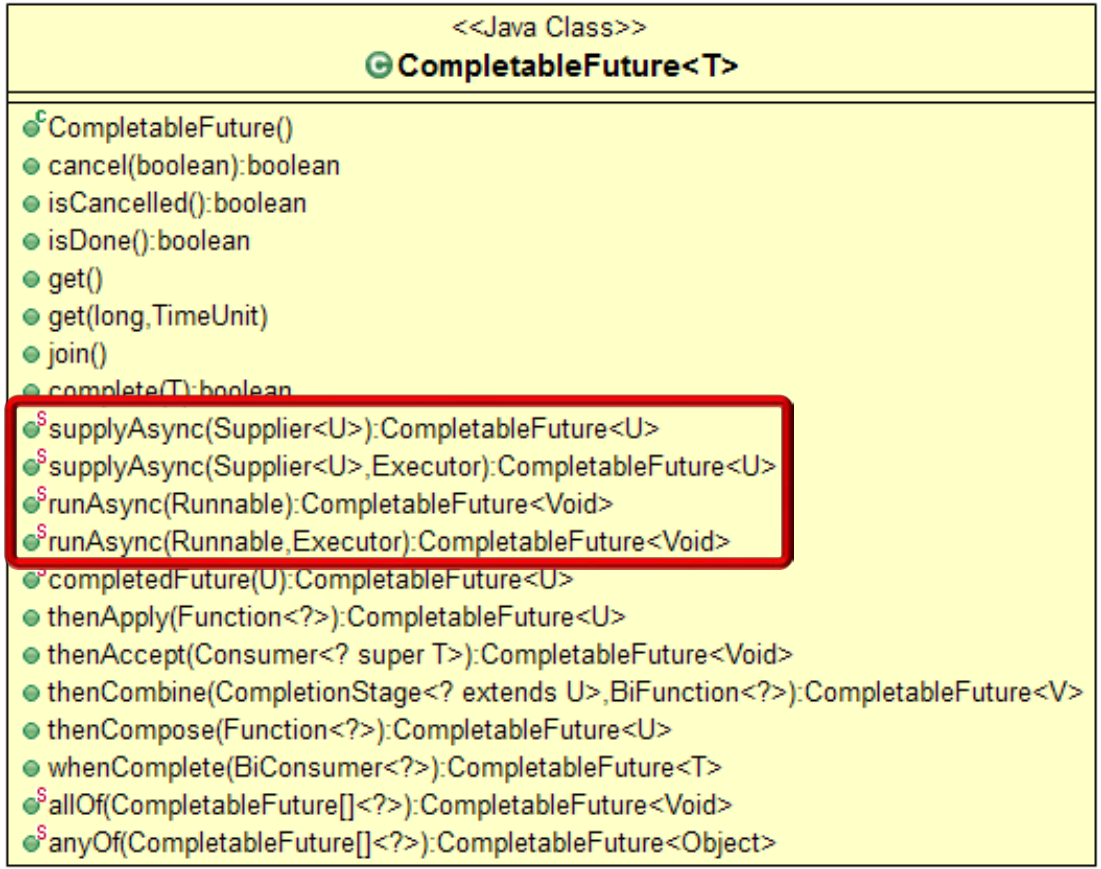
- Understand advanced features of completable futures, e.g.
  - Factory methods that initiate async computations



Factory Methods Initiate Async Computations

Factory Methods Initiate Async Computations

- Four factory methods initiate async computations



Factory Methods Initiate Async Computations

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

Methods	Params	Returns	Behavior
supplyAsync	Supplier	CompletableFuture with result of Supplier	Asynchronously run supplier in common fork/join pool
supplyAsync	Supplier, Executor	CompletableFuture with result of Supplier	Asynchronously run supplier in given executor pool

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

Factory Methods Initiate Async Computations

- Four factory methods initiate async 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 async 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);
      });
```

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

supplyAsync() is a more concise version of Callable.call()

Factory Methods Initiate Async Computations

- Four factory methods initiate async 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 pool

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

Factory Methods Initiate Async Computations

- Four factory methods initiate async 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, but returns no values

```
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 async 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, but returns no values

```
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!

Factory Methods Initiate Async Computations

- Four factory methods initiate async computations
  - These computations may or may not return a value
  - Async 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>

Factory Methods Initiate Async Computations

- Four factory methods initiate async computations
  - These computations may or may not return a value
  - Async 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>

This thread pool defaults to common fork-join pool, but can be given explicitly

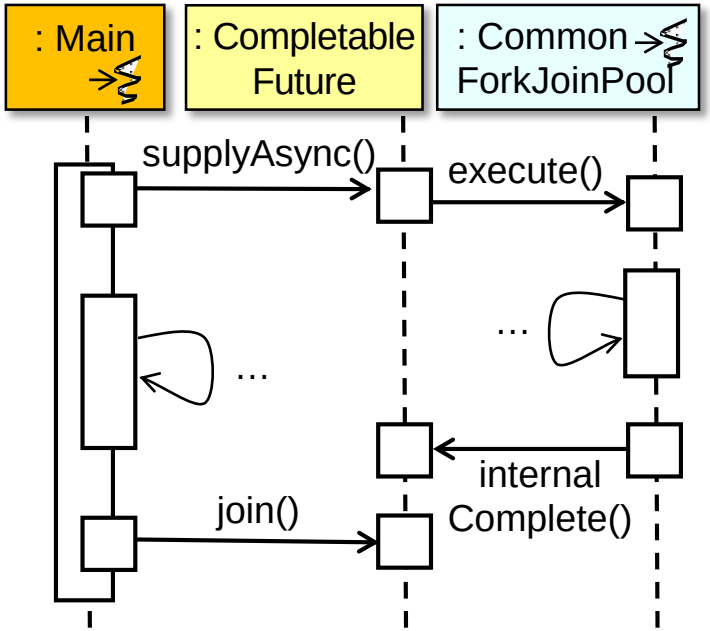
Applying Completable Future Factory Methods

Applying Completable Future Factory Methods

- Using supplyAsync() to multiply big fractions

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



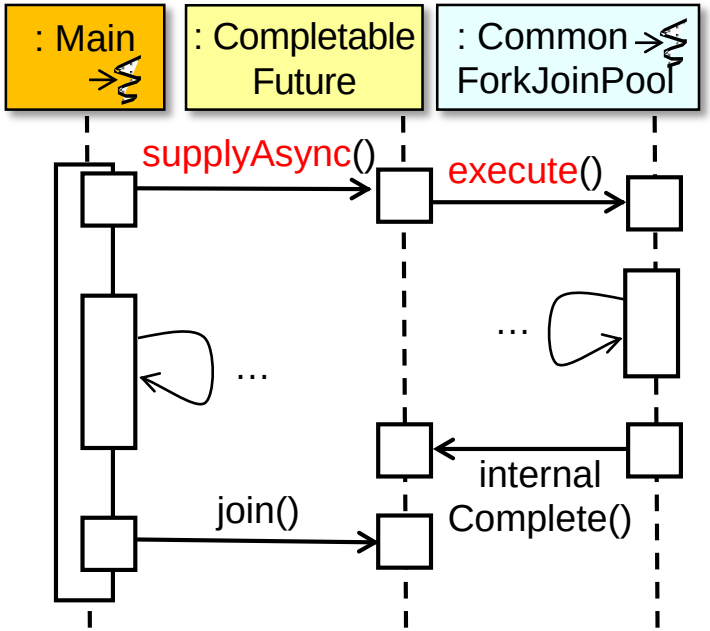
See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8

Applying Completable Future Factory Methods

- Using supplyAsync() to multiply big fractions

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



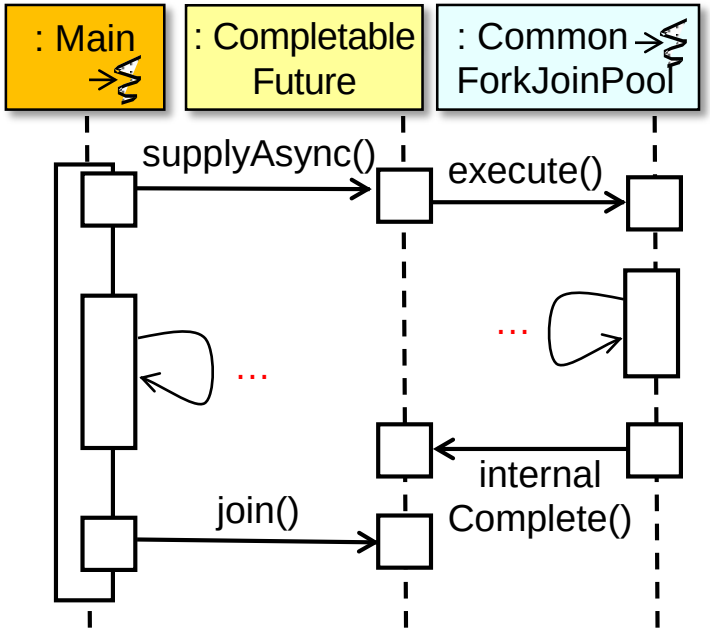
Schedule computation for execution on the fork-join pool

Applying Completable Future Factory Methods

- Using supplyAsync() to multiply big fractions

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



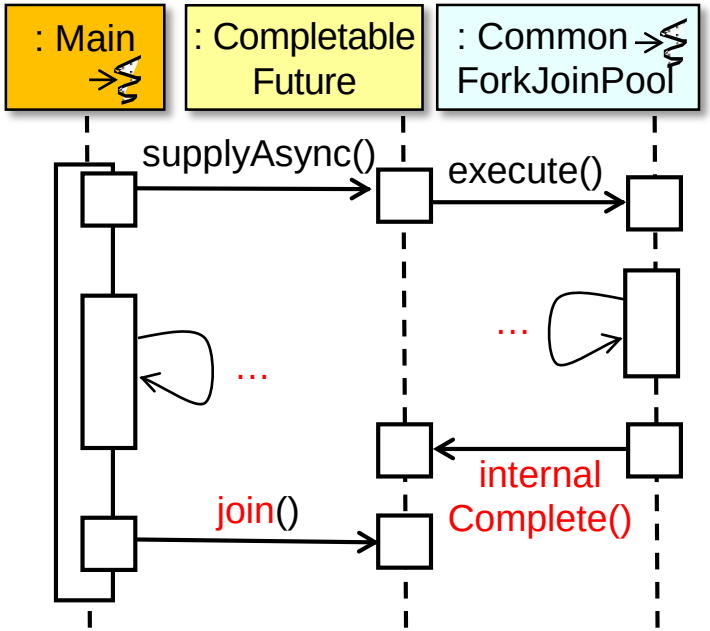
These computations run concurrently

Applying Completable Future Factory Methods

- Using supplyAsync() to multiply big fractions

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



join() blocks until result is complete

Applying Completable Future Factory Methods

- Using supplyAsync() to multiply big fractions

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

Java threads are *not* explicitly used in this example!

End of Overview of Advanced Java 8 Completable Future Features (Part 1)