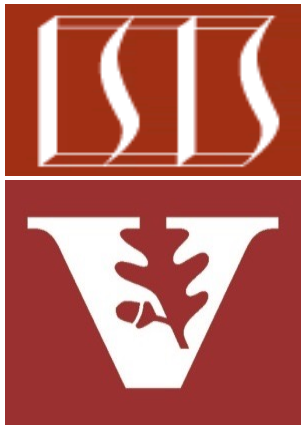
Advanced Java CompletableFuture Features: Two Stage Completion Methods (Part 1)

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt



Professor of Computer Science

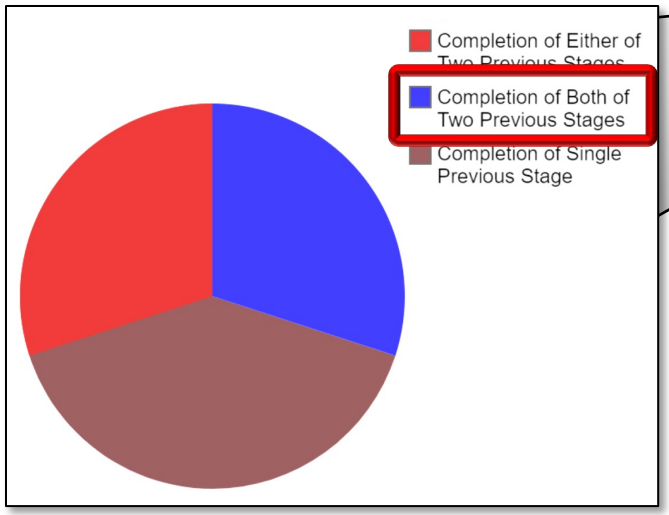
Institute for Software Integrated Systems

Vanderbilt University

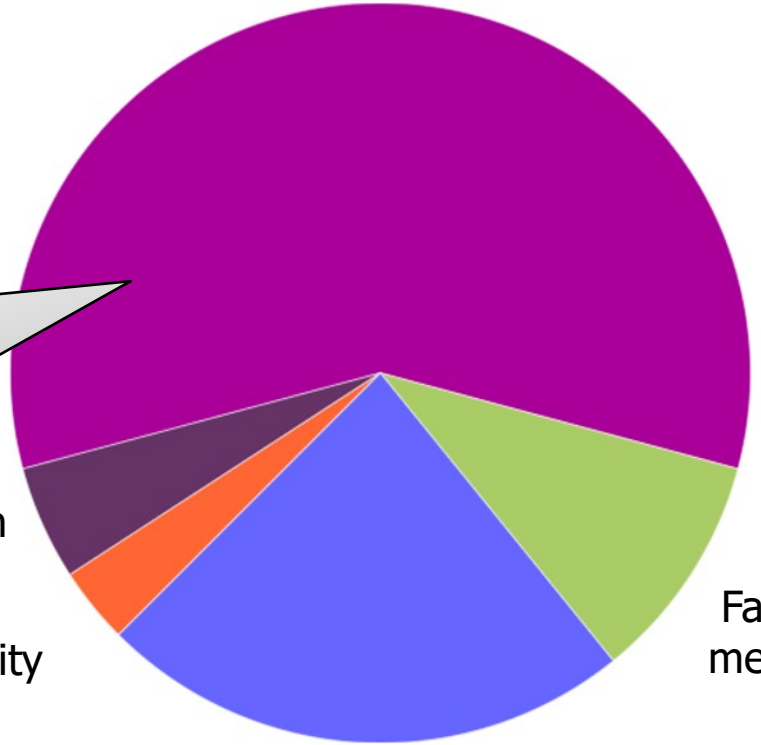
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand how completion stage methods chain dependent actions
- Know how to group these methods
- Single stage methods
- Two stage methods (and)



Completion stage methods



Exception methods

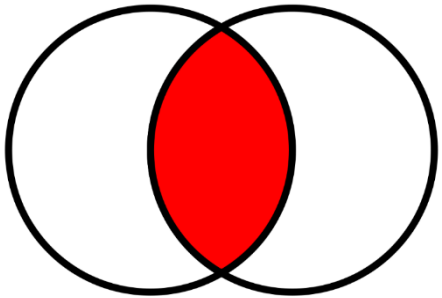
Arbitrary-arity methods

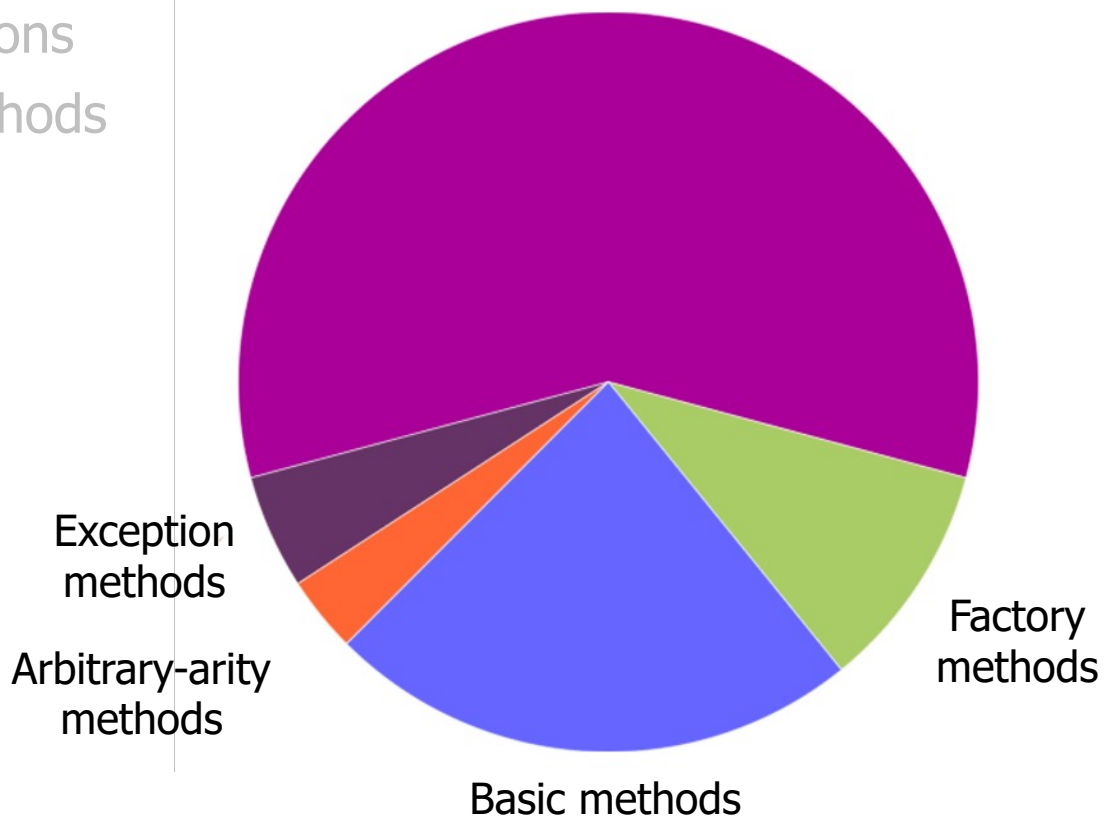
Factory methods

Basic methods

Learning Objectives in this Part of the Lesson

- Understand how completion stage methods chain dependent actions
- Know how to group these methods
- Single stage methods
- Two stage methods (and)



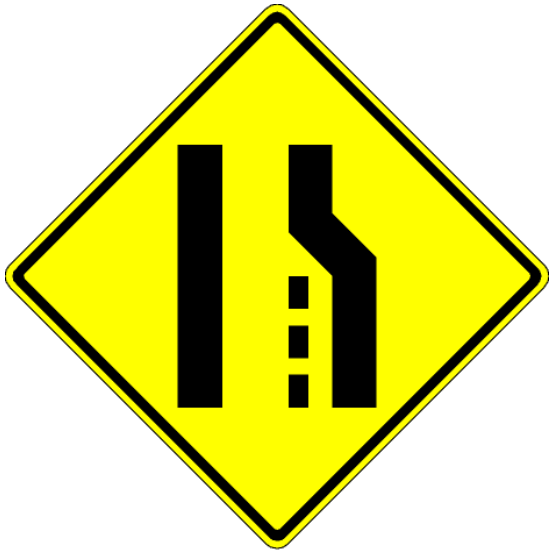


See en.wikipedia.org/wiki/Logical_conjunction

Methods Triggered by Completion of Both of Two Stages

Methods Triggered by Completion of Both of Two Stages

- Methods triggered by completion of both of two previous stages
  - thenCombine()

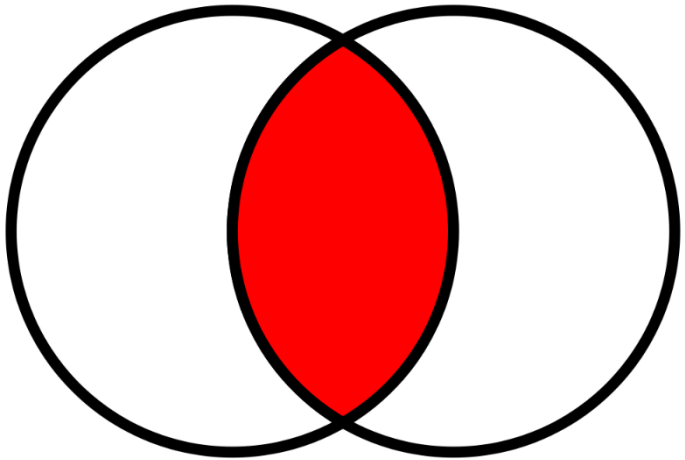


```
CompletableFuture<U> thenCombine
   (CompletionStage<? Extends U>
                              other,
    BiFunction<? super T,
               ? super U,
               ? extends V> fn)
{ ... }
```

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

Methods Triggered by Completion of Both of Two Stages

- Methods triggered by completion of both of two previous stages
  - thenCombine()
    - Applies a BiFunction action to two previous stages' results

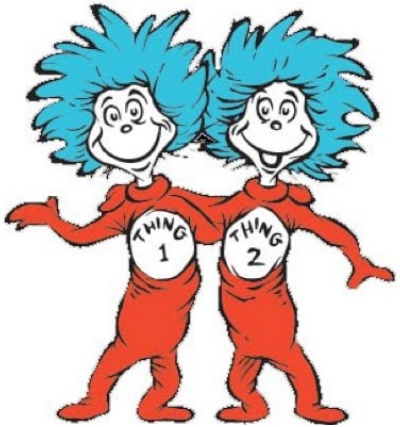


```
CompletableFuture<U> thenCombine
  (CompletionStage<? Extends U>
                     other,
   BiFunction<? super T,
              ? super U,
              ? extends V> fn)
{ ... }
```

See en.wikipedia.org/wiki/Logical_conjunction

Methods Triggered by Completion of Both of Two Stages

- Methods triggered by completion of both of two previous stages
  - thenCombine()
    - Applies a BiFunction action to two previous stages' results
      - Two futures are used here:




```
CompletableFuture<U> thenCombine
  (CompletionStage<? Extends U>
                     other,
   BiFunction<? super T,
              ? super U,
              ? extends V> fn)
{ ... }
```

Methods Triggered by Completion of Both of Two Stages

- Methods triggered by completion of both of two previous stages
  - thenCombine()
    - Applies a BiFunction action to two previous stages' results
      - Two futures are used here:
        - The future used to invoke thenCombine()
          - Not shown since it's not part of the method signature, but is implied since thenCombine() is a non-static method

```
CompletableFuture<U> thenCombine
  (CompletionStage<? Extends U>
                     other,
   BiFunction<? super T,
              ? super U,
              ? extends V> fn)
{ ... }
```

Methods Triggered by Completion of Both of Two Stages

- Methods triggered by completion of both of two previous stages
  - thenCombine()
    - Applies a BiFunction action to two previous stages' results
      - Two futures are used here:
        - The future used to invoke thenCombine()
        - The `other' future passed to thenCombine()

```
CompletableFuture<U> thenCombine
  (CompletionStage<? Extends U>
                   other,
   BiFunction<? super T,
              ? super U,
              ? extends V> fn)
{ ... }
```

Methods Triggered by Completion of Both of Two Stages

- Methods triggered by completion of both of two previous stages
  - thenCombine()
    - Applies a BiFunction action to two previous stages' results
    - Returns a future containing the result of the action

```
CompletableFuture<U> thenCombine
  (CompletionStage<? Extends U>
                        other,
   BiFunction<? super T,
              ? super U,
              ? extends V> fn)
{ ... }
```

Methods Triggered by Completion of Both of Two Stages

- Methods triggered by completion of both of two previous stages
  - thenCombine()
    - Applies a BiFunction action to two previous stages' results
    - Returns a future containing the result of the action

```
CompletableFuture<U> thenCombine
  (CompletionStage<? Extends U>
                     other,
   BiFunction<? super T,
              ? super U,
              ? extends V> fn)
{ ... }
```

thenCombine() essentially performs a simple "reduction"

Methods Triggered by Completion of Both of Two Stages

- Methods triggered by completion of both of two previous stages
  - thenCombine()
    - Applies a BiFunction action to two previous stages' results
    - Returns a future containing the result of the action
    - Used to "join" two paths of asynchronous execution

```
CompletableFuture<BF> compF1 =
    CompletableFuture
    .supplyAsync(() ->
       /* multiply two BFs. */);

CompletableFuture<BF> compF2 =
    CompletableFuture
    .supplyAsync(() ->
       /* divide two BFs. */);

compF1
  .thenCombine(compF2,
               BigFraction::add)

  .thenAccept(System.out::println);
```

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

Methods Triggered by Completion of Both of Two Stages

- Methods triggered by completion of both of two previous stages
  - thenCombine()
    - Applies a BiFunction action to two previous stages' results
    - Returns a future containing the result of the action
    - Used to "join" two paths of asynchronous execution

Asynchronously multiple & divide two big fractions

```
CompletableFuture<BF> compF1 =
    CompletableFuture
    .supplyAsync(() ->
       /* multiply two BFs. */);

CompletableFuture<BF> compF2 =
    CompletableFuture
    .supplyAsync(() ->
       /* divide two BFs. */);

compF1
   .thenCombine(compF2,
                BigFraction::add)

   .thenAccept(System.out::println);
```

Methods Triggered by Completion of Both of Two Stages

- Methods triggered by completion of both of two previous stages
  - thenCombine()
    - Applies a BiFunction action to two previous stages' results
    - Returns a future containing the result of the action
  - Used to "join" two paths of asynchronous execution

thenCombine()'s action is triggered only after its two associated futures complete

```
CompletableFuture<BF> compF1 =
    CompletableFuture
    .supplyAsync(() ->
       /* multiply two BFs. */);

CompletableFuture<BF> compF2 =
    CompletableFuture
    .supplyAsync(() ->
       /* divide two BFs. */);

compF1
   .thenCombine(compF2,
                BigFraction::add)

   .thenAccept(System.out::println);
```

Methods Triggered by Completion of Both of Two Stages

- Methods triggered by completion of both of two previous stages
  - thenCombine()
    - Applies a BiFunction action to two previous stages' results
    - Returns a future containing the result of the action
    - Used to "join" two paths of asynchronous execution

```
CompletableFuture<BF> compF1 =
    CompletableFuture
    .supplyAsync(() ->
       /* multiply two BFs. */);

CompletableFuture<BF> compF2 =
    CompletableFuture
    .supplyAsync(() ->
       /* divide two BFs. */);

compF1
   .thenCombineAsync(compF2,
                     aLongDurationBiFunction)

   .thenAccept(System.out::println);
```

thenCombineAsync() can be used if a long-duration BiFunction is applied

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

Methods Triggered by Completion of Both of Two Stages

- Methods triggered by completion of both of two previous stages
  - thenCombine()
    - Applies a BiFunction action to two previous stages' results
    - Returns a future containing the result of the action
    - Used to "join" two paths of asynchronous execution

```
CompletableFuture<BF> compF1 =
    CompletableFuture
    .supplyAsync(() ->
       /* multiply two BFs. */);

CompletableFuture<BF> compF2 =
    CompletableFuture
    .supplyAsync(() ->
       /* divide two BFs. */);

compF1
   .thenCombine(compF2,
                BigFraction::add)

   .thenAccept(System.out::println);
```

Print out the results

End of Advanced Java CompletableFuture Features: Two Stage Completion Methods (Part 1)