### Advanced Java Completable Future Features: Single Stage Completion Methods (Part 2)

Douglas C. Schmidt
<a href="mailto:d.schmidt@vanderbilt.edu">d.schmidt@vanderbilt.edu</a>
www.dre.vanderbilt.edu/~schmidt



**Professor of Computer Science** 

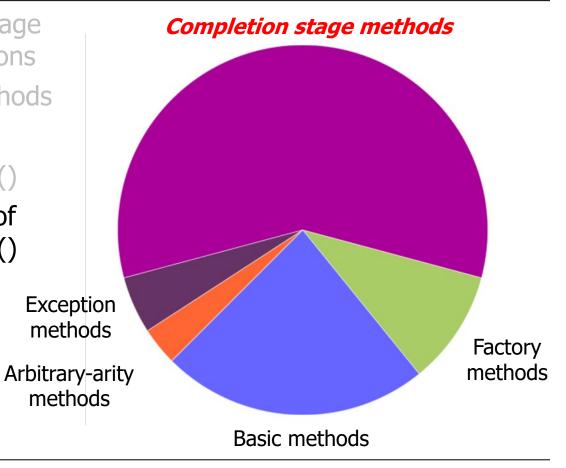
**Institute for Software Integrated Systems** 

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, e.g.
  - thenApply() & thenCompose()
  - thenAccept() & comparison of thenApply() & thenCompose()



- Methods triggered by completion CompletableFuture<Void>
- of a single previous stage
  - thenAccept()

```
thenAccept
    (Consumer<? super T> action)
{ . . . }
```

- Methods triggered by completion of a single previous stage
  - thenAccept()
    - Applies a consumer action to handle previous stage's result

```
CompletableFuture<Void>
    thenAccept
        (Consumer<? super T> action)
{ ... }
```

- Methods triggered by completion of a single previous stage
  - thenAccept()
    - Applies a consumer action to handle previous stage's result

CompletableFuture<Void>
 thenAccept
 (Consumer<? super T> action)
{ ... }

This action behaves as a "callback" with a side-effect



See en.wikipedia.org/wiki/Callback\_(computer\_programming)

- Methods triggered by completion of a single previous stage
  - thenAccept()
    - Applies a consumer action to handle previous stage's result
    - Returns a future to Void

```
CompletableFuture<Void>
    thenAccept
       (Consumer<? super T> action)
{ ... }
```

- BigFraction unreduced = BigFraction Methods triggered by completion
  - .valueOf(new BigInteger("..."), of a single previous stage
  - new BigInteger("..."), thenAccept()
  - false); // Don't reduce! Applies a consumer action to
    - handle previous stage's result Supplier<BigFraction> reduce = () -> BigFraction.reduce(unreduced);

CompletableFuture

.thenAccept(System.out::println);

Returns a future to Void

Often used at the end of a

- chain of completion stages .supplyAsync(reduce) .thenApply(BigFraction
- ::toMixedString)
- thenApply() returns a string future that thenAccept() prints when it completes

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

- Methods triggered by completion of a single previous stage
  - thenAccept()
    - Applies a consumer action to handle previous stage's result
    - Returns a future to Void
    - Often used at the end of a chain of completion stages

```
Supplier<BigFraction> reduce = ()
-> BigFraction.reduce(unreduced);
```

```
.supplyAsync(reduce)
```

CompletableFuture

.thenAccept(System.out::println);

println() is a callback that has a side-effect (i.e., printing the mixed string)

- Methods triggered by completion of a single previous stage
  - thenAccept()
    - Applies a consumer action to handle previous stage's result
    - Returns a future to Void
    - Often used at the end of a chain of completion stages
    - May lead to "callback hell" if used excessively!

```
function register()
   Swing a "'y
        LE 45 70072 'user_name'13 4
           Af 18 POST( 'unor ganastird new' 1) 4
               M (0_F007) 'user_panaword_new') --- 4_F007('oner_panaword_nepont')) {
                   if (strlen(5_POST('user_password_new')) > 5) (
                       if (strian)5 POST('usor name')) < 65 & strict(5 POST('usor name')) > 15 (
                            if (prog_match('/'[s-8\d][2.64(8/1', d_Pos2('oser_namo'))) {
                                Foser - read user's room, oser case, $11
                               if (timestforerf'uner name is) {
                                   Af (d_POST('user_enell')) (
                                        if (strick() POST['osor_small']) < 65) (
                                           of (filter warts 2007) near small"], Filter Walsdare Bearly) (
                                                $_$ESSIONE'mag'] - 'Sou are now registered so-please login's
                                                header('Location: ' . I_SHRVER('PHP_SHLF'));
                                             else Tong - 'The must provide a valid beall address';
                                        5. else Imag + "Enell bust by less than 60 characters";
                                    | else from - "finall cannot be empty";
                                5 else deng - Tourname already exists ;
                            ) else fines a 'Uncertang much be only a-r. A-2, 5-3's
                        ) else Tmag - 'Ouername must be between 7 and 64 characters';
                    I also from " "Pearword must be at lesst & chargeters";
                5 else Smay - 'Passwords do not match's
            | else Snay = 'Empty Pensword';
        ) also frap - 'Empty Unormano's
        E_SESSION['mog'] - Snogs
   return register form();
```

have similar method signatures

thenApply() & thenCompose()

? extends

CompletionStage<U>> fn)

(Function<? super T,

1.5

{ . . . }

 Unlike thenApply(), however, thenCompose() avoids unwieldy nesting of futures

```
CompletableFuture<BF>>>
reduceAndMultiplyFractions =
 unreduced -> CompletableFuture
  .supplyAsync
    (() -> BF.reduce(unreduced))
```

.supplyAsync(() ->

reduced.multiply(...));

Function<BF, CompletableFuture<</pre>

```
Nesting is
unwieldy!
               .thenApply
                 (reduced -> CompletableFuture
```

Function < BF,

 Unlike thenApply(), however, thenCompose() avoids unwieldy nesting of futures

```
Eliminates the nesting of futures via "flattening"!
```

```
CompletableFuture<BF>>
reduceAndMultiplyFractions =
  unreduced -> CompletableFuture
  .supplyAsync
  (() -> BF.reduce(unreduced))
```

```
. thenCompose
```

```
(reduced -> CompletableFuture
  .supplyAsync(() ->
```

reduced.multiply(...));

• • •

- Unlike thenApply(), however, thenCompose() avoids unwieldy nesting of futures
  - thenApplyAsync() can often be used to replace nesting of thenCompose(supplyAsync())

- . s
  - .thenApplyAsync(reduced
     -> reduced.multiply(...));

(() -> BF.reduce(unreduced))

- Unlike thenApply(), however, thenCompose() avoids unwieldy nesting of futures
  - thenApplyAsync() can often be used to replace nesting of thenCompose(supplyAsync())
  - thenCompse() can also avoid calling join() when flattening nested completable futures

```
CompletableFuture<Integer> countF =
  .CompletableFuture
      .supplyAsync
         (() ->
          longRunnerReturnsCF())
      .thenCompose
        (Function.identity())
 supplyAsync() will return a
   CompletableFuture to a
  CompletableFuture here!!
```

- Unlike thenApply(), however, thenCompose() avoids unwieldy nesting of futures
  - thenApplyAsync() can often be used to replace nesting of thenCompose(supplyAsync())
  - thenCompse() can also avoid calling join() when flattening nested completable futures

```
CompletableFuture<Integer> countF =
  .CompletableFuture
     .supplyAsync
        (() ->
         longRunnerReturnsCF())
     . thenCompose
       (Function.identity())
```

This idiom flattens the return value to "just" one CompletableFuture!

- Unlike thenApply(), however, thenCompose() avoids unwieldy nesting of futures
  - thenApplyAsync() can often be used to replace nesting of thenCompose(supplyAsync())
  - thenCompse() can also avoid calling join() when flattening nested completable futures
  - thenComposeAsync() can avoid calling supplyAsync() again in a chain

CompletableFuture<Integer> countF =
 .CompletableFuture
 supplyAsync

- .supplyAsync
  - longRunnerReturnsCF())

.thenComposeAsync
 (this::longerBlockerReturnsCF)

Runs longBlockerReturnsCF() in a common fork-join pool thread

### End of Advanced Java CompletableFuture Features: Single Stage Completion Methods (Part 2)