## Advanced Java CompletableFuture Features: Single Stage Completion Methods

Douglas C. Schmidt

<u>d.schmidt@vanderbilt.edu</u>

www.dre.vanderbilt.edu/~schmidt



**Professor of Computer Science** 

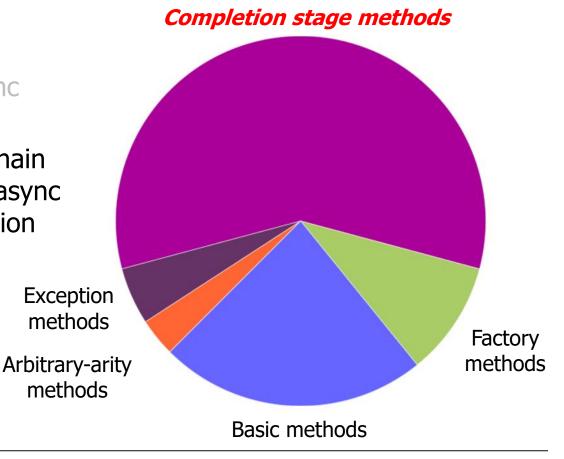
**Institute for Software Integrated Systems** 

Vanderbilt University Nashville, Tennessee, USA



#### Learning Objectives in this Part of the Lesson

- Understand advanced features of completable futures, e.g.
  - Factory methods initiate async computations
  - Completion stage methods chain together actions to perform async result processing & composition
    - Method grouping
    - Single stage methods



- of a single previous stage
   thenApply()

- Methods triggered by completion of a single previous stage
  - thenApply()
    - Applies a function action to the previous stage's result

- Methods triggered by completion of a single previous stage
  - thenApply()
    - Applies a function action to the previous stage's result
    - Returns a future containing the result of the action

- Methods triggered by completion of a single previous stage
  - thenApply()
    - Applies a function action to the previous stage's result
    - Returns a future containing the result of the action
    - Used for a quick sync action that returns a value rather than a future

BigFraction unreduced = BigFraction .valueOf(new BigInteger("..."),

new BigInteger("..."),

false); // Don't reduce!

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

CompletableFuture . supplyAsync (reduce)

. thenApply (BigFraction ::toMixedString)

e.g., toMixedString() returns a string value

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

- CompletableFuture<U> thenCompose Methods triggered by completion
  - of a single previous stage
  - thenApply()
  - thenCompose()

- extends CompletionStage<U>> fn)
- **{ ... }**

(Function<? super T,

- Methods triggered by completion of a single previous stage
  - thenApply()
  - thenCompose()
    - Applies a function action to the previous stage's result

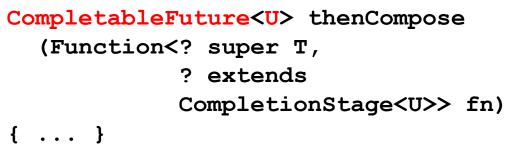
```
CompletableFuture<U> thenCompose (Function<? super T, ? extends
```

CompletionStage<U>> fn)

- Methods triggered by completion of a single previous stage
  - thenApply()
  - thenCompose()
    - Applies a function action to the previous stage's result
    - Returns a future containing result of the action directly
      - *i.e., not* a nested future

- Methods triggered by completion of a single previous stage
  - thenApply()
  - thenCompose()
    - Applies a function action to the previous stage's result
    - Returns a future containing result of the action directly
      - *i.e., not* a nested future

thenCompose() is similar to flatMap() on a Stream or Optional







- Methods triggered by completion of a single previous stage
  - thenApply()
  - thenCompose()
    - Applies a function action to the previous stage's result
    - Returns a future containing result of the action directly
    - Used for a longer async action that returns a future

.thenCompose
 (reduced -> CompletableFuture
 .supplyAsync(() ->
 reduced.multiply(...));

- Methods triggered by completion of a single previous stage
  - thenApply()
  - thenCompose()
    - Applies a function action to the previous stage's result
    - Returns a future containing result of the action directly
    - Used for a longer async action that returns a future

This function reduces & multiplies big fractions

```
.thenCompose
  (reduced -> CompletableFuture
    .supplyAsync(() ->
    reduced.multiply(...)));
```

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

- Methods triggered by completion of a single previous stage
  - thenApply()
  - thenCompose()
    - Applies a function action to the previous stage's result
    - Returns a future containing result of the action directly
    - Used for a longer async action that returns a future

Reduce big fraction asynchronously & return a completable future

.thenCompose
 (reduced -> CompletableFuture
 .supplyAsync(() ->
 reduced.multiply(...)));

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

- Methods triggered by completion Function<br/>
  of a single previous stage Completal
  - thenApply()
  - thenCompose()
    - Applies a function action to the previous stage's result
    - Returns a future containing result of the action directly
    - Used for a longer async action that returns a future

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

.thenCompose
/(reduced -> CompletableFuture
.supplyAsync(() ->

reduced.multiply(...));

supplyAsync() returns a future, but thenCompose() "flattens" this future

Nesting is

unwieldy!

- Methods triggered by completion of a single previous stage
  - thenApply()
  - thenCompose()
    - Applies a function action to the previous stage's result
    - Returns a future containing result of the action directly
    - Used for a longer async action that returns a future
    - Avoids unwieldy nesting of futures à la thenApply()

```
.thenApply
  (reduced -> CompletableFuture
    .supplyAsync(() ->
    reduced.multiply(...)));
```

Flattening is

more concise!

Function < BF,

- Methods triggered by completion of a single previous stage
  - thenApply()
  - thenCompose()
    - Applies a function action to the previous stage's result
    - Returns a future containing result of the action directly
    - Used for a longer async action that returns a future
    - Avoids unwieldy nesting of futures à la thenApply()

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

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

thenApplyAsync() can often replace thenCompose(supplyAsync()) nestings

- Methods triggered by completion of a single previous stage
  - thenApply()
  - thenCompose()
    - Applies a function action to the previous stage's result
    - Returns a future containing result of the action directly
    - Used for a longer async action that returns a future
    - Avoids unwieldy nesting of futures à la thenApply()

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

Can be used to avoid calling join() when flattening nested completable futures

- Methods triggered by completion of a single previous stage
  - thenApply()
  - thenCompose()
    - Applies a function action to the previous stage's result
    - Returns a future containing result of the action directly
    - Used for a longer async action that returns a future
    - Avoids unwieldy nesting of futures à la thenApply()

.thenCompose
 (Function.identity())

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

Can be used to avoid calling join() when flattening nested completable futures

- Methods triggered by completion of a single previous stage
  - thenApply()
  - thenCompose()
    - Applies a function action to the previous stage's result

Returns a future containing

- result of the action directlyUsed for a longer async action
- Avoids unwieldy nesting of futures à la thenApply()

that returns a future

CompletableFuture<Integer> countF = .CompletableFuture

- .supplyAsync (() ->
  - longRunnerReturnsCF())

.thenComposeAsync
 (this::longBlockerReturnsCF)

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

thenComposeAsync() can be used to avoid calling supplyAsync() again in a chain

- Methods triggered by completion of a single previous stage
  - thenApply()
  - thenCompose()
  - thenAccept()

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

- Methods triggered by completion of a single previous stage
  - thenApply()
  - thenCompose()
  - 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
  - thenApply()
  - thenCompose()
  - 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

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

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

- Methods triggered by completion of a single previous stage
  - thenApply()
  - thenCompose()
  - 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

BigFraction unreduced = BigFraction
 .valueOf(new BigInteger("..."),

.valueOf(new BigInteger("..."),
 new BigInteger("..."),

new BigInteger("..."),
false); // Don't reduce!

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

CompletableFuture .supplyAsync(reduce)

.thenAccept(System.out::println);

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
  - thenApply()
  - thenCompose()
  - 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

new BigInteger("..."),
false); // Don't reduce!

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

CompletableFuture .supplyAsync(reduce)

.thenApply(BigFraction

::toMixedString)

.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
  - thenApply()
  - thenCompose()
  - 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!"

```
function register()
    if (leapty(% FOOTy)
        Select a 77's
        if (8 FOST) weer name (1) (
           LE 15 POSTI 'mery password new' 19 &
                If (8_POST) went_persond_how'l was 8_POST('nest_personned_nepert'l) (
                    if (strlengs Post( uses passwood new'l) > 5) (
                        if (etrian); F0077 'over name' [] < 65 44 strien() F0071 'over name' [] > 15 (
                            if (prog_match('/'[x-s\d](7.64)5/1', 6_9002('usor_name'))) {
                                Front - read user'd FORT 'user come the
                                if (tiesekt@unnyf'menr asso' [13 6
                                    If (d_9007) 'user_seeil' 15 (
                                        Af [stries($ PODT] near meall | ] 4 113 4
                                            AF (filter_verts_bont; soor_mail'], Filter_validate_mail) (
                                                 P_SECRIOR['mag'] - 'Sec are now requirered so please login's
                                                 headect 'Location: ' . N. SERVERS '2009 SECP'11:
                                              else Hong a 'five must provide a valid small address';
                                         } else burg + "Enell burt be less than 64 characters";
                                    ) else fong + "Enall cannot be empty";
                                ) else dong - 'Tourname already sicists';
                            ) else from - 'Consume must be only a-s, A-I, S-3';
                        ) else thon - 'Durensus much be between 2 and 66 characters's
                    | else from - "Dissword must be at least & chargeters's
                ) else Snoy - 'Patawords do not match's
            | also fray - 'Emply Pensword's
         alse from " 'fruty lineranc's
        speed - I'pen' pertinent
    return register_form();
```

### End of Advanced Java CompletableFuture Features: Single Stage Completion Methods