Advanced Java CompletableFuture Features:

Single Stage Completion 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 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
Methods Triggered by Completion of a Single Stage
Methods Triggered by Completion of a Single Stage

- Methods triggered by completion of a single previous stage
- `thenApply()`
Methods Triggered by Completion of a Single Stage

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

CompletetableFuture\(<U>\) \(\text{thenApply}\)

\[(\text{Function}\langle? \text{super} \ T, \text{? extends} \ U\rangle \ \text{fn})\]

\{
  ...  
\}

See [docs.oracle.com/javase/8/docs/api/java/util/function/Function.html](docs.oracle.com/javase/8/docs/api/java/util/function/Function.html)
Methods Triggered by Completion of a Single Stage

- 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

```java
CompletableFuture<U> thenApply
    (Function<? super T,
        ? extends U> fn)

{ ... }
```
Methods Triggered by Completion of a Single Stage

• 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

```java
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](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8)
Methods Triggered by Completion of a Single Stage

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

```java
CompletableFuture<U> thenCompose
  (Function<? super T, ? extends CompletionStage<U>> fn)
  
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#thenCompose
Methods Triggered by Completion of a Single Stage

- 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)
{ ... }

See docs.oracle.com/javase/8/docs/api/java/util/function/Function.html
Methods Triggered by Completion of a Single Stage

- 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 Stage

- 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

See [dzone.com/articles/understanding-flatmap](https://dzone.com/articles/understanding-flatmap)
Methods Triggered by Completion of a Single Stage

- 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

Function\(<BF, \\
\text{CompletableFuture}\(<BF>> \\
\text{reduceAndMultiplyFractions} = \\
\text{unreduced} \rightarrow \text{CompletableFuture} \\
.\text{supplyAsync} \\
(()) \rightarrow BF.\text{reduce}(\text{unreduced}))

.\text{thenCompose} \\
(\text{reduced} \rightarrow \text{CompletableFuture} \\
.\text{supplyAsync}(() \rightarrow \\
\text{reduced}.\text{multiply}(...)))

...
Methods Triggered by Completion of a Single Stage

- 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

```java
Function<BF,
  CompletableFuture<BF>>
reduceAndMultiplyFractions =
unreduced -> CompletableFuture.supplyAsync
  (() -> BF.reduce(unreduced))
  .thenCompose
  (reduced -> CompletableFuture.supplyAsync(() ->
    reduced.multiply(...)));
```

This function reduces & multiplies big fractions

See [docs.oracle.com/javase/8/docs/api/java/util/function/Function.html](http://docs.oracle.com/javase/8/docs/api/java/util/function/Function.html)
Methods Triggered by Completion of a Single Stage

- 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

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

Reduce big fraction asynchronously & return a completable future

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

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#supplyAsync
Methods Triggered by Completion of a Single Stage

- 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

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

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

supplyAsync() returns a future, but thenCompose() “flattens” this future
```

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#supplyAsync](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#supplyAsync)
Methods Triggered by Completion of a Single Stage

- 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()

```java
Function<BF, CompletableFuture<CompletableFuture<BF>>>
    reduceAndMultiplyFractions = unreduced -> CompletableFuture.of(BF.reduce(unreduced))
    .thenApply(reduced -> CompletableFuture.of(reduced.multiply(...)));
```

Nesting is unwieldy!
Methods Triggered by Completion of a Single Stage

- 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()

```
Function<BF, CompletableFuture<BF>>
reduceAndMultiplyFractions = unreduced -> CompletableFuture
  .supplyAsync(() -> BF.reduce(unreduced))
  .thenApplyAsync(reduced -> reduced.multiply(...));
```

*Flattening is more concise!*
Methods Triggered by Completion of a Single Stage

- 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()

```java
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 Stage

- 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()
    .thenCompose((() ->
      longRunnerReturnsCF())
    .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 Stage

- 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()`

```java
CompletableFuture<Integer> countF =
  .supplyAsync()
  .thenComposeAsync()
    ((() ->
      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 Stage

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

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

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#thenAccept](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#thenAccept)
Methods Triggered by Completion of a Single Stage

- 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 Stage

- 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

See en.wikipedia.org/wiki/Callback_(computer_programming)
Methods Triggered by Completion of a Single Stage

- 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

```java
CompletableFuture<Void>
    .thenAccept
    (Consumer<? super T> action)
    { ... }
```
Methods Triggered by Completion of a Single Stage

- 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

```java
BigFraction unreduced = BigFraction
    .valueOf(new BigInteger("..."),
             new BigInteger("..."),
             false); // Don’t reduce!

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

CompletableFuture
    .supplyAsync(reduce)
    .thenApply(BigFraction
                ::toMixedString)
    .thenAccept(System.out::println);
```

- `thenApply()` returns a string future that `thenAccept()` prints when it completes

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8](http://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8)
Methods Triggered by Completion of a Single Stage

- 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("..."),
  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 Stage

- 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!”

See [dzone.com/articles/callback-hell](dzone.com/articles/callback-hell)
End of Advanced Java
CompletableFuture Features:
Single Stage Completion Methods