Advanced Java CompletableFuture Features: Two Stage Completion Methods (Part 2)

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)
• Two stage methods (or)

See en.wikipedia.org/wiki/Logical_disjunction
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)
- Two stage methods (or)

See [en.wikipedia.org/wiki/Logical_disjunction](en.wikipedia.org/wiki/Logical_disjunction)
Methods Triggered by Completion of Two Stages
Methods Triggered by Completion of Either of Two Stages

- Methods triggered by completion of either of two previous stages
- `acceptEither()`

```java
CompletableFuture<Void> acceptEither(CompletionStage<? Extends T> other, Consumer<? super T> action)
{
    ...}
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#acceptEither
Methods Triggered by Completion of Either of Two Stages

- Methods triggered by completion of either of two previous stages
  - `acceptEither()`
    - Applies a consumer action that handles either of the previous stages' results

```java
CompletableFuture<Void> acceptEither(
    CompletionStage<? Extends T> other,
    Consumer<? super T> action)
{
    ...
}
```

See [en.wikipedia.org/wiki/Logical_disjunction](en.wikipedia.org/wiki/Logical_disjunction)
Methods Triggered by Completion of Either of Two Stages

- Methods triggered by completion of either of two previous stages
  - `acceptEither()`
    - Applies a consumer action that handles either of the previous stages' results
  - Two futures are used here:
    ```java
    CompletableFuture<Void> acceptEither(
        CompletionStage<? Extends T> other,
        Consumer<? super T> action)
    { ... }
    ```
Methods Triggered by Completion of Either of Two Stages

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

```java
CompletableFuture<Void> acceptEither
    (CompletionStage<? Extends T>
        other,
        Consumer<? super T> action)
{ ... }
```
Methods Triggered by Completion of Either of Two Stages

- Methods triggered by completion of either of two previous stages
  - acceptEither()
    - Applies a consumer action that handles either of the previous stages' results
  - Two futures are used here:
    - The future used to invoke acceptEither()
    - The `other' future passed to acceptEither()
Methods Triggered by Completion of Either of Two Stages

• Methods triggered by completion of either of two previous stages
  • acceptEither()
    • Applies a consumer action that handles either of the previous stages' results
  • Returns a future to Void

```java
CompletableFuture<Void> acceptEither(CompletionStage<? Extends T> other,
          Consumer<? super T> action)
{
  ... 
}
```

See www.baeldung.com/java-void-type
Methods Triggered by Completion of Either of Two Stages

- Methods triggered by completion of either of two previous stages
  - `acceptEither()`
    - Applies a consumer action that handles either of the previous stages' results
    - Returns a future to Void
    - Often used at the end of a chain of completion stages

```java
CompletableFuture<List<BigFraction>> quickSortF = CompletableFuture.supplyAsync(() ->
    quickSort(list));

CompletableFuture<List<BigFraction>> mergeSortF = CompletableFuture.supplyAsync(() ->
    mergeSort(list));
```
## Methods Triggered by Completion of Either of Two Stages

- Methods triggered by completion of either of two previous stages
  - `acceptEither()`
    - Applies a consumer action that handles either of the previous stages' results
    - Returns a future to Void
    - Often used at the end of a chain of completion stages

```java
CompletableFuture<List<BigFraction>> quickSortF = CompletableFuture.supplyAsync(() -> quickSort(list));
CompletableFuture<List<BigFraction>> mergeSortF = CompletableFuture.supplyAsync(() -> mergeSort(list));
```

Create a pair of `CompletableFuture` objects that will contain the results of sorting the list using two different algorithms in two different threads.

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8)
Methods Triggered by Completion of Either of Two Stages

- Methods triggered by completion of either of two previous stages
  - `acceptEither()`
    - Applies a consumer action that handles either of the previous stages' results
    - Returns a future to Void
    - Often used at the end of a chain of completion stages

```java
CompletableFuture<List<BigFraction>> quickSortF = CompletableFuture.supplyAsync(() -> quickSort(list));
CompletableFuture<List<BigFraction>> mergeSortF = CompletableFuture.supplyAsync(() -> mergeSort(list));
quickSortF.acceptEither(mergeSortF, results -> results.forEach(fraction -> System.out.println(fraction.toMixedString())));
```

This method is invoked when either `quickSortF` or `mergeSortF` complete.
Methods Triggered by Completion of Either of Two Stages

- Methods triggered by completion of either of two previous stages
  - `acceptEither()`
    - Applies a consumer action that handles either of the previous stages' results
    - Returns a future to Void
    - Often used at the end of a chain of completion stages

```java
CompletableFuture<List<BigFraction>> quickSortF = CompletableFuture.supplyAsync(() -> quickSort(list));
CompletableFuture<List<BigFraction>> mergeSortF = CompletableFuture.supplyAsync(() -> mergeSort(list));
quickSortF.acceptEither(mergeSortF, results -> results.forEach(fraction -> System.out.println(fraction.toMixedString())));
```
Methods Triggered by Completion of Either of Two Stages

- Methods triggered by completion of either of two previous stages
  - `acceptEither()`
    - Applies a consumer action that handles either of the previous stages' results
    - Returns a future to Void
    - Often used at the end of a chain of completion stages

```java
CompletableFuture<List<BigFraction>> quickSortF = CompletableFuture.supplyAsync(() ->
    quickSort(list));
CompletableFuture<List<BigFraction>> mergeSortF = CompletableFuture.supplyAsync()
    mergeSort(list);
quickSortF.acceptEither
    (mergeSortF, results -> results
    .forEach(fraction ->
        System.out.println(fraction
            .toMixedString())));
```

`acceptEither()` does *not* cancel the second future after the first one completes
Methods Triggered by Completion of Either of Two Stages

- Methods triggered by completion of either of two previous stages
- `acceptEither()`
  - Applies a consumer action that handles either of the previous stages' results
  - Returns a future to Void
  - Often used at the end of a chain of completion stages

```java
CompletableFuture<List<BigFraction>> quickSortF = CompletableFuture.supplyAsync(() ->
    quickSort(list));

CompletableFuture<List<BigFraction>> mergeSortF = CompletableFuture.supplyAsync(() ->
    mergeSort(list));

quickSortF.acceptEitherAsync(mergeSortF, results -> aLongDurationConsumer(results))
```

`acceptEitherAsync()` can be used if a long-duration Consumer is applied

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#acceptEitherAsync
End of Advanced Java CompletableFuture Features: Two Stage Completion Methods (Part 2)