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 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
    • Two stage methods (and)
    • Two stage methods (or)

Completion stage methods

Exception methods
Arbitrary-arity methods
Basic methods
Factory methods
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](https://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:
    - The future used to invoke `acceptEither()`
    - The `other` future passed to `acceptEither()`

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
    - Returns a future to Void

```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
    - 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 two completable futures that will contain the results of sorting the list using two different algorithms in two different threads
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

```
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())));
```
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.
End of Advanced Java CompletableFuture Features: Two Stage Completion Methods (Part 2)