Advanced Java Completable Future Features: Applying Completion Stage 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)
- Apply these methods
  - supplyAsync(), thenCompose(), & thenApplyAsync()
  - thenAccept() & acceptEither()

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8)
Applying Completable Future Completion Stage Methods
We show key completion stage methods via the `testFractionMultiplications1()` method that multiplies big fractions using a stream of `CompletableFuture`

```java
static void testFractionMultiplications1() {
    ...
    Stream.generate(() -> makeBigFraction(new Random(), false))
        .limit(sMAX_FRACTIONS)
        .map(reduceAndMultiplyFraction)
        .collect(FuturesCollector.toFuture())
        .thenAccept(ex8::sortAndPrintList);
}
```

Return a single future to a list of big fractions being reduced & multiplied asynchronously

See lesson on "Advanced Java Completable Future Features: Implementing FuturesCollector"
We show key completion stage methods via the testFractionMultiplications1() method that multiplies big fractions using a stream of CompletableFutures.

```java
static void testFractionMultiplications1() {
    ...
    Stream.generate(() -> makeBigFraction(new Random(), false))
        .limit(sMAX_FRACTIONS)
        .map(reduceAndMultiplyFraction)
        .collect(FuturesCollector.toFuture())
        .thenAccept(ex8::sortAndPrintList);
}
```

Sort & print results when all async computations complete.
We show key completion stage methods via the testFractionMultiplications1() method that multiplies big fractions using a stream of CompletableFutures.

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

    quickSortF.acceptEither(mergeSortF, sortedList -> sortedList.forEach(frac -> display(frac.toMixedString())));
} ...
```

Sort & print a list of reduced & multiplied big fractions
ApplyingCompletableFutureCompletionStageMethods

- We show key completion stage methods via the testFractionMultiplications1() method that multiplies big fractions using a stream of CompletableFutures

```java
static void sortAndPrintList(List<BigFraction> list) {

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

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

  quickSortF.acceptEither(mergeSortF, sortedList ->
      sortedList.forEach(frac -> display(frac.toMixedString())));
}
```

Asynchronously apply quick sort & merge sort!
Applying CompletableFuture Completion Stage Methods

- We show key completion stage methods via the testFractionMultiplications1() method that multiplies big fractions using a stream of CompletableFutures.

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

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

    quickSortF.acceptEither(mergeSortF, sortedList ->
            sortedList.forEach(frac -> display(frac.toMixedString()));
    ); ...
```
Applying Completable Future Completion Stage Methods

- We show key completion stage methods via the testFractionMultiplications1() method that multiplies big fractions using a stream of CompletableFutures

```java
static void sortAndPrintList(List<BigFraction> list) {

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

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

    quickSortF.acceptEither(mergeSortF, sortedList ->
        sortedList.forEach(frac -> display(frac.toMixedString())));
}
```

If future is already completed the action runs in the thread that registered the action
Applying CompletableFuture Completion Stage Methods

- We show key completion stage methods via the testFractionMultiplications1() method that multiplies big fractions using a stream of CompletableFuture

```java
static void sortAndPrintList(List<BigFraction> list) {

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

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

    quickSortF.acceptEither(mergeSortF, sortedList ->
        sortedList.forEach(frac -> display(frac.toMixedString())));
}
```

Otherwise, the action runs in the thread in which the previous stage ran
Applying Completable Future Completion Stage Methods

- We show key completion stage methods via the testFractionMultiplications1() method that multiplies big fractions using a stream of CompletableFutures.

```java
static void sortAndPrintList(List<BigFraction> list) {

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

    quickSortF.acceptEither(mergeSortF, sortedList ->
        sortedList.forEach(frac -> display(frac.toMixedString())));
}
```

`acceptEither()` does not cancel the second future after the first one completes.
End of Advanced Java
CompletableFuture Features:
Applying Completion
Stage Methods (Part 2)