Advanced Java Completable Future Features: Applying Completion Stage 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
    • Two stage methods (and)
    • Two stage methods (or)
  • Apply these methods

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8)
Applying Completable Future Completion Stage Methods
Applying CompletableFuture 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(reduceAndMultiplyFractions)
        .collect(FuturesCollector.toFuture())
        .thenAccept(ex8::sortAndPrintList);
}
```

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8)
Applying CompletableFuture Future Completion Stage Methods

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

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

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#generate
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
BigFraction makeBigFraction(Random random, boolean reduced) {
    BigInteger numerator = new BigInteger(150000, random);
    BigInteger denominator = numerator.divide(BigInteger.valueOf(random.nextInt(10) + 1));

    return BigFraction.valueOf(numerator, denominator, reduced);
}
```

Factory method that creates a large & random big fraction
Applying CompletableFuture Future Completion Stage Methods

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

```java
BigFraction makeBigFraction(Random random, boolean reduced) {
    BigInteger numerator =
        new BigInteger(150000, random);

    BigInteger denominator =
        numerator.divide(BigInteger.valueOf(random.nextInt(10) + 1));

    return BigFraction.valueOf(numerator,
                               denominator,
                               reduced);
}
```

A random # generator & a flag indicating whether to reduce the BigFraction
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
BigFraction makeBigFraction(Random random, boolean reduced) {
    BigInteger numerator =
        new BigInteger(150000, random);

    BigInteger denominator =
        numerator.divide(BigInteger.valueOf(random.nextInt(10) + 1));

    return BigFraction.valueOf(numerator, denominator, reduced);
}
```

Make a random numerator uniformly distributed over range 0 to \(2^{150000} - 1\)

See `docs.oracle.com/javase/8/docs/api/java/math/BigInteger.html#BigInteger`
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
BigFraction makeBigFraction(Random random, boolean reduced) {
    BigInteger numerator =
        new BigInteger(150000, random);

    BigInteger denominator =
        numerator.divide(BigInteger
                       .valueOf(random.nextInt(10) + 1));

    return BigFraction.valueOf(numerator,
                                denominator,
                                reduced);
}
```

Make a denominator by dividing the numerator by random # between 1 & 10.
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
BigFraction makeBigFraction(Random random, boolean reduced) {
    BigInteger numerator =
        new BigInteger(150000, random);

    BigInteger denominator =
        numerator.divide(BigInteger
            .valueOf(random.nextInt(10) + 1));

    return BigFraction.valueOf(numerator,
        denominator,
        reduced);
}
```

Return a BigFraction w/the numerator & denominator
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 testFractionMultiplications1() {
    ...
    Stream.generate(() -> makeBigFraction(new Random(), false))
        .limit(sMAX_FRACTIONS)
        .map(reduceAndMultiplyFraction)
        .collect(FuturesCollector.toFuture())
        .thenAccept(ex8::sortAndPrintList);
}
```

Reduce & multiply all these big fractions asynchronously
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 testFractionMultiplications1() {
    Function<BigFraction, CompletableFuture<BigFraction>>
    reduceAndMultiplyFraction = unreducedFrac ->
        CompletableFuture.supplyAsync(() ->
            BigFraction.reduce(unreducedFrac))
        .thenCompose(reducedFrac ->
            CompletableFuture.supplyAsync(() ->
                reducedFrac
                .multiply(sBigFraction)));

```

Lambda function that asynchronously reduces & multiplies big fractions
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 testFractionMultiplications1() {
    Function<BigFraction, CompletableFuture<BigFraction>>
        reduceAndMultiplyFraction = unreducedFrac ->
            CompletableFuture
                .supplyAsync(() -> BigFraction.reduce(unreducedFrac))
                .thenCompose(reducedFrac -> CompletableFuture
                    .supplyAsync(() -> reducedFrac.multiply(sBigFraction)))
                .thenCompose(reducedFrac -> CompletableFuture
                    .supplyAsync(() -> reducedFrac.multiply(sBigFraction)));
    ...
```

Asynchronously reduce a big fraction.
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 testFractionMultiplications1() {
    Function<BigFraction, CompletableFuture<BigFraction>>
        reduceAndMultiplyFraction = unreducedFrac ->
        CompletableFuture.supplyAsync(() -> BigFraction.reduce(unreducedFrac))
            .thenCompose(reducedFrac -> CompletableFuture.supplyAsync(() -> reducedFrac.multiply(sBigFraction)));

    ... Asynchronously multiply big fractions ...
}```
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 testFractionMultiplications1() {
    Function<BigFraction, CompletableFuture<BigFraction>>
    reduceAndMultiplyFraction = unreducedFrac ->
    CompletableFuture.supplyAsync(() -> BigFraction.reduce(unreducedFrac)).
    thenCompose(reducedFrac -> CompletableFuture.supplyAsync(() -> reducedFrac.multiply(sBigFraction)));

    ...  
```

thenCompose() acts like flatMap() to ensure one level of CompletableFuture nesting.
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 testFractionMultiplications2() {
    Function<BigFraction, CompletableFuture<BigFraction>>
        reduceAndMultiplyFraction = unreducedFrac ->
        CompletableFuture.supplyAsync(() ->
            BigFraction.reduce(unreducedFrac))
            .thenApplyAsync(reducedFrac ->
                reducedFrac.multiply(sBigFraction));
```

`thenApplyAsync()` is an alternative means to avoid calling `supplyAsync()` again

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

Outputs a stream of completable futures to async operations on big fractions
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 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”
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 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
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())));
}
```

Sort & print a list of reduced & multiplied big fractions
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())));
}
```

Asynchronously apply quick sort & merge sort!
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())));
}
```

*Select whichever result finishes first.*
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*
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