Advanced Java Completable Future Features: Applying Completion Stage Methods (Part 1)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

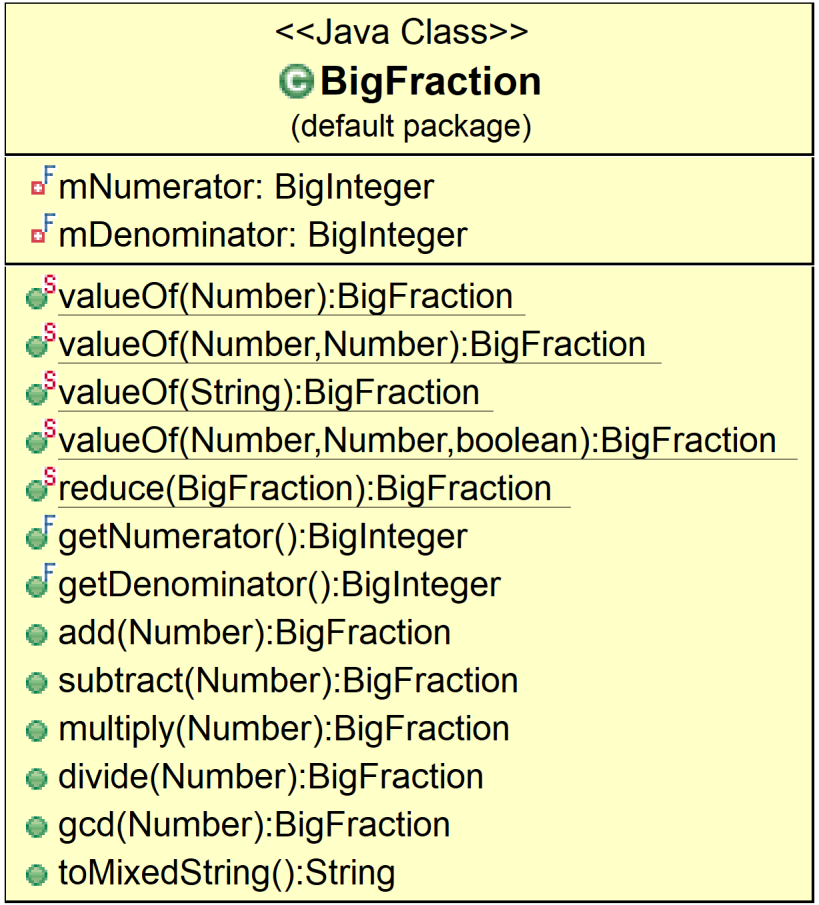
Professor of Computer Science

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()



See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8

Applying Completable Future Completion Stage Methods

Applying Completable Future Completion Stage Methods

- We show completion stage methods via testFractionMultiplications1(), which multiplies BigFraction objects using a stream of CompletableFuture objects

```
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

Applying Completable Future Completion Stage Methods

- We show completion stage methods via testFractionMultiplications1(), which multiplies BigFraction objects using a stream of CompletableFuture objects

```
static void testFractionMultiplications1() {
   ...
   Stream.generate(() -> makeBigFraction(new Random(), false))

         .limit(sMAX_FRACTIONS)

         .map(reduceAndMultiplyFraction)

         .collect(FuturesCollector.toFuture())

         .thenAccept(ex8::sortAndPrintList);
}
```

Generate a bounded # of large, random, & un-reduced BigFaction objects

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#generate

Applying Completable Future Completion Stage Methods

- We show completion stage methods via testFractionMultiplications1(), which multiplies BigFraction objects using a stream of CompletableFuture objects

```
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 BigFraction

Applying Completable Future Completion Stage Methods

- We show completion stage methods via testFractionMultiplications1(), which multiplies BigFraction objects using a stream of CompletableFuture objects

```
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 completion stage methods via testFractionMultiplications1(), which multiplies BigFraction objects using a stream of CompletableFuture objects

```
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 completion stage methods via testFractionMultiplications1(), which multiplies BigFraction objects using a stream of CompletableFuture objects

```
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 completion stage methods via testFractionMultiplications1(), which multiplies BigFraction objects using a stream of CompletableFuture objects

```
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 completion stage methods via testFractionMultiplications1(), which multiplies BigFraction objects using a stream of CompletableFuture objects

```
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 Fraction objects asynchronously

Applying Completable Future Completion Stage Methods

- We show completion stage methods via testFractionMultiplications1(), which multiplies BigFraction objects using a stream of CompletableFuture objects

```
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 BigFraction objects

Applying Completable Future Completion Stage Methods

- We show completion stage methods via testFractionMultiplications1(), which multiplies BigFraction objects using a stream of CompletableFuture objects

```
static void testFractionMultiplications1() {
  Function<BigFraction, CompletableFuture<BigFraction>>
    reduceAndMultiplyFraction = unreducedFrac ->
      CompletableFuture

        .supplyAsync(() -> BigFraction.reduce(unreducedFrac))

        .thenCompose(reducedFrac -> CompletableFuture
                     .supplyAsync(() -> reducedFrac
                                  .multiply(sBigFraction)));

  ...
```

Asynchronously reduce a BigFraction

Applying Completable Future Completion Stage Methods

- We show completion stage methods via testFractionMultiplications1(), which multiplies BigFraction objects using a stream of CompletableFuture objects

```
static void testFractionMultiplications1() {
  Function<BigFraction, CompletableFuture<BigFraction>>
    reduceAndMultiplyFraction = unreducedFrac ->
      CompletableFuture


        .supplyAsync(() -> BigFraction.reduce(unreducedFrac))

        .thenCompose(reducedFrac -> CompletableFuture
                     .supplyAsync(() -> reducedFrac
                                  .multiply(sBigFraction)));
  ...
```

Asynchronously multiply Big Fraction objects

Applying Completable Future Completion Stage Methods

- We show completion stage methods via testFractionMultiplications1(), which multiplies BigFraction objects using a stream of CompletableFuture objects

```
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 completion stage methods via testFractionMultiplications1(), which multiplies BigFraction objects using a stream of CompletableFuture objects

```
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 completion stage methods via testFractionMultiplications1(), which multiplies BigFraction objects using a stream of CompletableFuture objects

```
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 BigFraction objects

Applying Completable Future Completion Stage Methods

- We show completion stage methods via testFractionMultiplications1(), which multiplies BigFraction objects using a stream of CompletableFuture objects

```
static void testFractionMultiplications1() {
   ...
   Stream.generate(() -> makeBigFraction(new Random(), false))

         .limit(sMAX_FRACTIONS)

         .map(reduceAndMultiplyFraction)

         .collect(FuturesCollector.toFuture())

         .thenAccept(ex8::sortAndPrintList);
}
```

Part 2 of this lesson focuses on other CompletableFuture & Stream methods

End of Advanced Java CompletableFuture Features: Applying Completion Stage Methods (Part 1)