Advanced Java Completable Future Features: Applying Completion Stage Methods (Part 2)

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

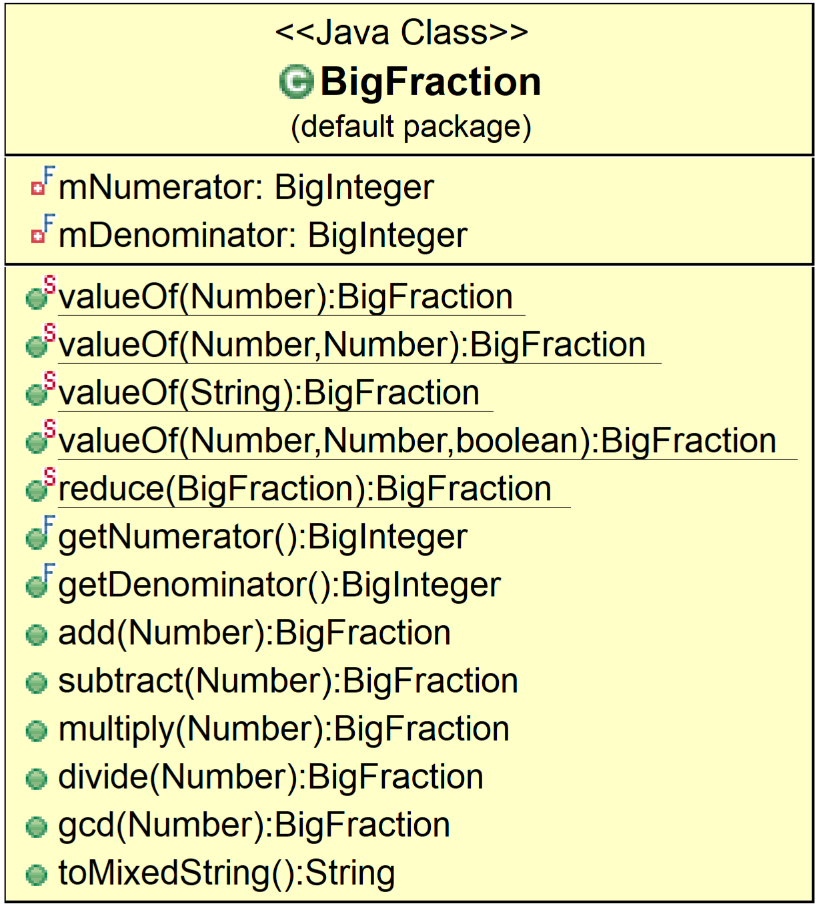
Professor of Computer Science

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()



<<Java Class>>
BigFraction
(default package)

mNumerator: BigInteger
mDenominator: BigInteger

valueOf(Number):BigFraction
valueOf(Number,Number):BigFraction
valueOf(String):BigFraction
valueOf(Number,Number,boolean):BigFraction
reduce(BigFraction):BigFraction
getNumerator():BigInteger
getDenominator():BigInteger
add(Number):BigFraction
subtract(Number):BigFraction
multiply(Number):BigFraction
divide(Number):BigFraction
gcd(Number):BigFraction
toMixedString():String

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

Applying Completable Future Completion Stage Methods

Applying Completable Future Completion Stage Methods

- Show completion stage methods via the testFractionMultiplications1() method that multiplies BigFraction objects using a CompletableFuture stream

```
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 BigFraction objects being reduced & multiplied asynchronously

See lesson on "*Advanced Java Completable Future Features: Implementing FuturesCollector*"

Applying Completable Future Completion Stage Methods

- Show completion stage methods via the testFractionMultiplications1() method that multiplies BigFraction objects using a CompletableFuture stream

```
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

- Show completion stage methods via the testFractionMultiplications1() method that multiplies BigFraction objects using a CompletableFuture stream

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

Sort & print a List of reduced/multiplied BigFraction objects

```
  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

- Show completion stage methods via the testFractionMultiplications1() method that multiplies BigFraction objects using a CompletableFuture stream

```
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

- Show completion stage methods via the testFractionMultiplications1() method that multiplies BigFraction objects using a CompletableFuture stream

```
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()));
};  ...
```

Apply whichever result finishes first..

Applying Completable Future Completion Stage Methods

- Show completion stage methods via the testFractionMultiplications1() method that multiplies BigFraction objects using a CompletableFuture stream

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


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

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

If future is already completed the action runs in the thread that registered the action

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

Applying Completable Future Completion Stage Methods

- Show completion stage methods via the testFractionMultiplications1() method that multiplies BigFraction objects using a CompletableFuture stream

```
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

- Show completion stage methods via the testFractionMultiplications1() method that multiplies BigFraction objects using a CompletableFuture stream

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


   CompletableFuture<List<BigFraction>> quickSortF =
     CompletableFuture.supplyAsync(() -> qui[illegible]kSort([illegible]));

   CompletableFuture<List<BigFraction>>
     CompletableFuture.supplyAsync(() -> [illegible]


   quickSortF.acceptEither(mergeSortF, sortedList ->
       sortedList.forEach(frac -> display(fra[illegible]toMix[illegible]tring()));
}; ...
```

acceptEither() does *not* cancel the second future after the first one completes

End of Advanced Java CompletableFuture Features: Applying Completion Stage Methods (Part 2)