Overview of Advanced Java 8
CompletableFuture Features (Part 3)

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 that initiate async functionality
- Completion stage methods used to chain together actions that perform async result processing & composition
- Apply completion stage methods to BigFractions

See [github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex8](https://github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex8)
Learning Objectives in this Part of the Lesson

- Understand advanced features of completable futures, e.g.
  - Factory methods that initiate async functionality
- Completion stage methods used to chain together actions that perform async result processing & composition
- Apply completion stage methods to BigFractions
- Know how to handle runtime exceptions
Applying Completable Future Completion Stage Methods
Applying CompletableFuture Completion Stage Methods

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

```java
static void testFractionMultiplications() {
    ...  
    Stream.generate(() -> makeBigFraction(new Random(), false))
        .limit(sMAX_FRACTIONS)
        .map(reduceAndMultiplyFractions)
        .collect(FuturesCollector.toFuture())
        .thenAccept(printSortedList);
}
```

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

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

```java
static void testFractionMultiplications() {
    Function<BigFraction, CompletableFuture<BigFraction>>
    reduceAndMultiplyFraction = unreducedFrac ->
        CompletableFuture.supplyAsync(() -> BigFraction.reduce(unreducedFrac))
            .thenCompose(reducedFrac -> CompletableFuture.supplyAsync(() -> reducedFrac.multiply(sBigFraction)));

    //...
```

Lambda that asynchronously reduces/multiplies a big fraction

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

```java
static void testFractionMultiplications() {
    Function<BigFraction, CompletableFuture<BigFraction>>
    reduceAndMultiplyFraction = unreducedFrac ->
        CompletableFuture.supplyAsync(() -> BigFraction.reduce(unreducedFrac))
            .thenCompose(reducedFrac -> CompletableFuture.supplyAsync(() ->
                reducedFrac.multiply(sBigFraction)));
...
```

Asynchronously reduce a big fraction
We show key completion stage methods via the testFractionMultiplications() method that multiplies big fractions using a stream of CompletableFutures.

```java
static void testFractionMultiplications() {
    Function<BigFraction, CompletableFuture<BigFraction>>
    reduceAndMultiplyFraction = unreducedFrac ->
        CompletableFuture
            .supplyAsync(() -> BigFraction.reduce(unreducedFrac))
            .thenCompose(reducedFrac -> CompletableFuture
                .supplyAsync(() -> reducedFrac.multiply(sBigFraction));
```
Applying Completable Future Completion Stage Methods

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

```java
static void testFractionMultiplications() {
    Function<BigFraction, CompletableFuture<BigFraction>>
        reduceAndMultiplyFraction = unreducedFrac -> CompletableFuture
            .supplyAsync(() -> BigFraction.reduce(unreducedFrac))
            .thenCompose(reducedFrac -> CompletableFuture
                .supplyAsync(() -> reducedFrac.multiply(sBigFraction)));
    ...
```

Asynchronously multiply big fractions
Applying CompletableFuture Completion Stage Methods

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

```java
static void testFractionMultiplications() {
    ...
    Consumer<List<BigFraction>> printSortedList = list -> {
        CompletableFuture<List<BigFraction>> quickSortF = CompletableFuture.supplyAsync(() -> quickSort(list));
        CompletableFuture<List<BigFraction>> mergeSortF = CompletableFuture.supplyAsync(() -> mergeSort(list));
        quickSortF.acceptEither(mergeSortF, results -> results.forEach(frac -> display(frac.toMixedString())));
    }
    ...
```

Sorts & prints a list of reduced fractions
We show key completion stage methods via the testFractionMultiplications() method that multiplies big fractions using a stream of CompletableFutures.

```java
static void testFractionMultiplications()
{
    Consumer<List<BigFraction>> printSortedList = list -> {
        CompletableFuture<List<BigFraction>> quickSortF = CompletableFuture.supplyAsync(() -> quickSort(list));
        CompletableFuture<List<BigFraction>> mergeSortF = CompletableFuture.supplyAsync(() -> mergeSort(list));
        quickSortF.acceptEither(mergeSortF, results -> results.forEach(frac -> display(frac.toMixedString())));
    };
    ...
    Asynchronously apply quick sort & merge sort!
}```
We show key completion stage methods via the testFractionMultiplications() method that multiplies big fractions using a stream of CompletableFutures.

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

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

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

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

```java
static void testFractionMultiplications() {
    ...
    Stream.generate(() -> makeBigFraction(new Random(), false))
        .limit(sMAX_FRACTIONS)
        .map(reduceAndMultiplyFraction)
        .collect(FuturesCollector.toFuture())
        .thenAccept(printSortedList);
}
```
Applying Completable Future Completion Stage Methods

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

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

Reduce & multiply these fractions asynchronously.
Applying Completable Future Completion Stage Methods

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

```java
static void testFractionMultiplications() {
    // ...
    Stream.generate(() -> makeBigFraction(new Random(), false))
        .limit(sMAX_FRACTIONS)
        .map(reduceAndMultiplyFraction)
        .collect(FuturesCollector.toFuture())
        .thenAccept(printSortedList);
}
```

Return a future to a list of big fractions being reduced & multiplied asynchronously
Applying Completable Future Completion Stage Methods

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

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

Sort & print results when all async computations complete.
Handling Runtime Exceptions in Completion Stages
Handling Runtime Exceptions in Completion Stages

- Completion stage methods handle runtime exceptions
### Handling Runtime Exceptions in Completion Stages

- **Completion stage methods handle runtime exceptions**

<table>
<thead>
<tr>
<th>Methods</th>
<th>Params</th>
<th>Returns</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>when Complete (Async)</td>
<td>Bi Consumer</td>
<td>Completable Future with result of earlier stage or throws exception</td>
<td>Handle outcome of a stage, whether a result value or an exception</td>
</tr>
<tr>
<td>handle (Async)</td>
<td>Bi Function</td>
<td>Completable Future with result of BiFunction</td>
<td>Handle outcome of a stage &amp; return new value</td>
</tr>
<tr>
<td>exceptionally Function</td>
<td>Completable Future &lt;Void&gt;</td>
<td>When exception occurs, replace exception with result value</td>
<td></td>
</tr>
</tbody>
</table>

See [community.oracle.com/docs/DOC-995305](community.oracle.com/docs/DOC-995305)
Handling Runtime Exceptions in Completion Stages

- Using the handle() method to handle exceptional or normal completions

```java
CompletableFuture.supplyAsync(() -> BigFraction.valueOf(100, denominator))
    .handle((fraction, ex) -> {
        if (fraction == null)
            return BigFraction.ZERO;
        else
            return fraction.multiply(sBigReducedFraction);
    })
    .thenAccept(fraction -> System.out.println(fraction.toMixedString()));
```

Handle outcome of previous stage
Using the handle() method to handle exceptional or normal completions

```java
CompletableFuture.supplyAsync(() ->
    BigFraction.valueOf(100, denominator))
    .handle((fraction, ex) -> {
      if (fraction == null)
        return BigFraction.ZERO;
      else
        return fraction.multiply(sBigReducedFraction);
    })
    .thenAccept(fraction ->
        System.out.println(fraction.toMixedString()));
```

The exception path
Handling Runtime Exceptions in Completion Stages

- Using the `handle()` method to handle exceptional or normal completions

```java
CompletableFuture.supplyAsync(() ->
    BigFraction.valueOf(100, denominator))
    .handle((fraction, ex) -> {
        if (fraction == null)
            return BigFraction.ZERO;
        else
            return fraction.multiply(sBigReducedFraction);
    })
    .thenAccept(fraction ->
    System.out.println(fraction.toMixedString()));
```

The “normal” path
Handling Runtime Exceptions in Completion Stages

- Using the `exceptionally()` method to handle exceptional or normal completions

```java
CompletableFuture.supplyAsync(() ->
    BigFraction.valueOf(100, denominator))
    .thenApply(fraction ->
        fraction.multiply(sBigReducedFraction))
    .exceptionally(ex -> BigFraction.ZERO)
    .thenAccept(fraction ->
        System.out.println(fraction.toMixedString()));
```

**Convert an exception to a 0 result**
End of Overview of Advanced Java 8 Completable Future Features (Part 3)