Learning Objectives in this Part of the Lesson

- Understand the basic completable futures features
- Know how to apply these basic features to operate on big fractions

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

- Understand the basic completable futures features
- Know how to apply these basic features to operate on big fractions
- Recognize limitations with these basic features

```java
public class CompletableFuture<T>
    extends Object
    implements Future<T>, CompletionStage<T>
```

A *Future* that may be explicitly completed (setting its value and status), and may be used as a *CompletionStage*, supporting dependent functions and actions that trigger upon its completion.

When two or more threads attempt to complete, `completeExceptionally`, or cancel a `CompletableFuture`, only one of them succeeds.

In addition to these and related methods for directly manipulating status and results, `CompletableFuture` implements interface `CompletionStage` with the following policies:
Applying Basic CompletableFuture Features
Applying Basic CompletableFuture Features

- We show how to apply basic completable future features in the context of BigFraction.

<table>
<thead>
<tr>
<th>Java Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigFraction</td>
</tr>
</tbody>
</table>

- mNumerator: BigInteger
- mDenominator: BigInteger

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigFraction()</td>
</tr>
<tr>
<td>static BigFraction of(Number)</td>
</tr>
<tr>
<td>static BigFraction of(Number, Number)</td>
</tr>
<tr>
<td>static BigFraction of(String)</td>
</tr>
<tr>
<td>static BigFraction of(Number, Number, boolean)</td>
</tr>
<tr>
<td>reduce(BigFraction): BigFraction</td>
</tr>
<tr>
<td>getNumerator(): BigInteger</td>
</tr>
<tr>
<td>getDenominator(): BigInteger</td>
</tr>
<tr>
<td>add(Number): BigFraction</td>
</tr>
<tr>
<td>subtract(Number): BigFraction</td>
</tr>
<tr>
<td>multiply(Number): BigFraction</td>
</tr>
<tr>
<td>divide(Number): BigFraction</td>
</tr>
<tr>
<td>gcd(Number): BigFraction</td>
</tr>
<tr>
<td>toMixedString(): String</td>
</tr>
</tbody>
</table>

See [LiveLessons/blob/master/Java8/ex8/src/utils/BigFraction.java](LiveLessons/blob/master/Java8/ex8/src/utils/BigFraction.java)
Applying Basic CompletableFuture Features

- We show how to apply basic completable future features in the context of BigFraction.

- Arbitrary-precision fraction, utilizing BigIntegers for numerator & denominator.

See docs.oracle.com/javase/8/docs/api/java/math/BigInteger.html
Applying Basic CompletableFuture Features

- We show how to apply basic completable future features in the context of BigFraction.
  - Arbitrary-precision fraction, utilizing BigIntegers for numerator & denominator.
  - Factory methods for creating “reduced” fractions, e.g.
    - $44/55 \rightarrow 4/5$
    - $12/24 \rightarrow 1/2$
    - $144/216 \rightarrow 2/3$
Applying Basic CompletableFuture Features

- We show how to apply basic completable future features in the context of BigFraction
  - Arbitrary-precision fraction, utilizing BigIntegers for numerator & denominator
  - Factory methods for creating “reduced” fractions
  - Factory methods for creating “non-reduced” fractions (& then reducing them)
    - e.g., 12/24 (→ 1/2)
Applying Basic CompletableFuture Features

- We show how to apply basic completable future features in the context of BigFraction
  - Arbitrary-precision fraction, utilizing BigIntegers for numerator & denominator
  - Factory methods for creating “reduced” fractions
  - Factory methods for creating “non-reduced” fractions (& then reducing them)
  - Arbitrary-precision fraction arithmetic
    - e.g., \( \frac{18}{4} \times \frac{2}{3} = 3 \)
Applying Basic CompletableFuture Features

- We show how to apply basic completable future features in the context of BigFraction
- Arbitrary-precision fraction, utilizing BigIntegers for numerator & denominator
- Factory methods for creating “reduced” fractions
- Factory methods for creating “non-reduced” fractions (& then reducing them)
- Arbitrary-precision fraction arithmetic
- Create a mixed fraction from an improper fraction
  - e.g., 18/4 → 4 1/2

See www.mathsisfun.com/improper-fractions.html
Applying Basic CompletableFuture Features

- Multiplying big fractions w/a completable future

```java
CompletableFuture<BigFraction> future = new CompletableFuture<>();

new Thread () -> {
    BigFraction bf1 =
        new BigFraction("62675744/15668936");
    BigFraction bf2 =
        new BigFraction("609136/913704");

    future.complete(bf1.multiply(bf2));
}.start();

...

System.out.println(future.join().toMixedString());
```

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8)
Applying Basic CompletableFuture Features

- Multiplying big fractions w/a completable future

```java
CompletableFuture<BigFraction> future = new CompletableFuture<>();

new Thread () -> {
    BigFraction bf1 = new BigFraction("62675744/15668936");
    BigFraction bf2 = new BigFraction("609136/913704");

    future.complete(bf1.multiply(bf2));
}.start();

... System.out.println(future.join().toString());
```

![Diagram](image)
Applying Basic CompletableFuture Features

- Multiplying big fractions w/a completable future

```
CompletableFuture<BigFraction> future = new CompletableFuture<>();

new Thread () -> {
    BigFraction bf1 = 
        new BigFraction("62675744/15668936");
    BigFraction bf2 = 
        new BigFraction("609136/913704");

    future.complete(bf1.multiply(bf2));
}).start();

... 
System.out.println(future.join().toMixedString());
```
Applying Basic CompletableFuture Features

- Multiplying big fractions w/a completable future

```java
CompletableFuture<BigFraction> future = new CompletableFuture<>();

new Thread () -> {
    BigFraction bf1 = new BigFraction("62675744/15668936");
    BigFraction bf2 = new BigFraction("609136/913704");

    future.complete(bf1.multiply(bf2));
}).start();
```

System.out.println(future.join().toString());

The computation multiplies BigFractions (via BigIntegers)

See docs.oracle.com/javase/8/docs/api/java/math/BigInteger.html
Applying Basic CompletableFuture Features

- Multiplying big fractions w/a completable future

```java
CompletableFuture<BigFraction> future = new CompletableFuture<>();

new Thread () -> {
    BigFraction bf1 =
        new BigFraction("62675744/15668936");
    BigFraction bf2 =
        new BigFraction("609136/913704");
    future.complete(bf1.multiply(bf2));
}).start();
```

These computations run concurrently

```java
System.out.println(future.join().toMixedString());
```
Applying Basic CompletableFuture Features

- Multiplying big fractions w/a completable future

```java
CompletableFuture<BigFraction> future = new CompletableFuture<>();

new Thread () {
    BigFraction bf1 = new BigFraction("62675744/15668936");
    BigFraction bf2 = new BigFraction("609136/913704");

    future.complete(bf1.multiply(bf2));
}.start();
```

... Explicitly complete the future w/result

```java
System.out.println(future.join().toMixedString());
```
Applying Basic CompletableFuture Features

- Multiplying big fractions w/a completable future

CompletableFuture<BigFraction> future = new CompletableFuture<>();

new Thread () -> {
    BigFraction bf1 =
        new BigFraction("62675744/15668936");
    BigFraction bf2 =
        new BigFraction("609136/913704");

    future.complete(bf1.multiply(bf2));
}.start();

future.join().toMixedString();

join() blocks until result is computed

...
Applying Basic CompletableFuture Features

- Multiplying big fractions w/a completable future

```java
CompletableFuture<BigFraction> future = new CompletableFuture<>();

new Thread () -> {
    BigFraction bf1 =
        new BigFraction("62675744/15668936");
    BigFraction bf2 =
        new BigFraction("609136/913704");

    future.complete(bf1.multiply(bf2));
}.start();
```

Convert result to a mixed fraction

```java
System.out.println(future.join().toMixedString());
```

See [www.mathsisfun.com/mixed-fractions.html](http://www.mathsisfun.com/mixed-fractions.html)
Limitations with Basic CompletableFuture Features
Limitations with Basic CompletableFuture Features

- Basic CompletableFuture features have similar limitations as futures
  - **Cannot** be chained fluently to handle async results
  - **Cannot** be triggered reactively
  - **Cannot** be treated efficiently as a `collection` of futures

See earlier lesson on "Evaluating the Pros & Cons of Java Futures"
Limitations with Basic CompletableFutures Features

- e.g., `join()` blocks until the future is completed.

```java
CompletableFuture<BigFraction> future = new CompletableFuture<>();

new Thread () -> {
    BigFraction bf1 =
        new BigFraction("62675744/15668936");
    BigFraction bf2 =
        new BigFraction("609136/913704");

    future.complete(bf1.multiply(bf2));
}.start();

... System.out.println(future.join().toMixedString());
```

This blocking call underutilizes cores & increases overhead.
Limitations with Basic CompletableFutures Features

- ..using timed get() is also problematic..

```java
CompletableFuture<BigFraction> future = new CompletableFuture<>();

new Thread(() -> {
    BigFraction bf1 = new BigFraction("62675744/15668936");
    BigFraction bf2 = new BigFraction("609136/913704");

    future.complete(bf1.multiply(bf2));
}).start();

System.out.println(future.get(1, SECONDS).toMixedString());
```

...Using a timeout to bound the blocking duration is inefficient & error-prone...
Limitations with Basic CompletableFuture Features

- We therefore need to leverage the advanced features of completable futures.
End of Applying Basic Java CompletableFuture Features