Overview of Basic Java 8
CompletableFuture Features (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 the basic completable futures features
- Know how to apply these basic features to multiply 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 multiply big fractions
• Recognize limitations with these basic features

Class CompletableFuture<T>

java.lang.Object
  java.util.concurrent.CompletableFuture<T>

All Implemented Interfaces:
  CompletionStage<T>, Future<T>

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 Completable Future Features
Applying Basic Completable Future Features

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

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

• We show how to apply basic completetable future features in the context of BigFraction
• Arbitrary-precision fraction, utilizing BigIntegers for numerator & denominator

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8
Applying Basic Completable Future 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 → 4/5
  - 12/24 → 1/2
  - 144/216 → 2/3

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

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8]
Applying Basic Completable Future 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., $\frac{18}{4} \times \frac{2}{3} = 3$

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8
Applying Basic Completable Future 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 arithimetic
- Create a mixed fraction from an improper fraction
  - e.g., 18/4 → 4 1/2

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

- Multiplying big fractions w/a completable future

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

ew 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 Completable Future 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());
```
Applying Basic Completable Future 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());
```

Start computation in a background thread
Applying Basic Completable Future 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());
```

The computation multiplies BigIntegers

See docs.oracle.com/javase/8/docs/api/java/math/BigInteger.html
Applying Basic Completable Future 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 Completable Future 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());
```

Explicitly complete the future
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());

join() blocks until result is computed
Limitations with Basic Completable Futures Features
Limitations with Basic Completable Futures Features

- Basic completable future 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
Limitations with Basic Completable Futures 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 Completable Futures 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(1, SECONDS).toMixedString());
```

Using a timeout to bound the blocking duration is still inefficient & error-prone
Limitations with Basic Completable Futures Features

- We therefore need to leverage the advanced features of completable futures.

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html
End of Overview of Basic Java 8 Completable Future Features (Part 2)