Applying Basic Java
CompletableFuture Features

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 features in the Java completable futures framework
- Know how to apply these basic features to operate on big fractions

See earlier lesson on "Programming with Java Futures"
Learning Objectives in this Part of the Lesson

• Understand the basic features in the Java completable futures framework
• Know how to apply these basic features to operate on big fractions
• Recognize limitations with these basic features

```java
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 Object, 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 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());

See 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().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().toMixedString());
```

Start computation in a background thread
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 [docs.oracle.com/javase/8/docs/api/java/math/BigInteger.html](https://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

```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
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();

    Convert result to a mixed fraction

... 
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 CompletableFuture Features

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

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 CompletableFutures 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 Applying Basic Java CompletableFuture Features