Advanced Java CompletableFuture
Features: Applying Factory Methods

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 initiate async computations
  - Applying factory methods
  - Multiply BigFraction objects concurrently & asynchronously
Learning Objectives in this Part of the Lesson

• Understand advanced features of completable futures, e.g.
  • Factory methods initiate async computations
    • Applying factory methods
      • Multiply BigFraction objects concurrently & asynchronously
    • Evaluate pros & cons of factory methods
Applying Completable Future Factory Methods
Applying CompletableFuture Factory Methods

- Use `supplyAsync()` to multiply BigFraction objects

```java
String f1 = "62675744/15668936";
String f2 = "609136/913704";
CompletableFuture<BigFraction> future = CompletableFuture
    .supplyAsync(() -> {
        BigFraction bf1 = 
            new BigFraction(f1);
        BigFraction bf2 = 
            new BigFraction(f2);

        return bf1.multiply(bf2);
    });

... 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 CompletableFuture Factory Methods

- Use supplyAsync() to multiply BigFraction objects

```java
String f1 = "62675744/15668936";
String f2 = "609136/913704";
CompletableFuture<BigFraction> future = CompletableFuture.
supplyAsync()
    .orElseGet(() -> {
        BigFraction bf1 =
            new BigFraction(f1);
        BigFraction bf2 =
            new BigFraction(f2);
        return bf1.multiply(bf2);
    });
...
System.out.println(future.join().toMixedString());
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#supplyAsync
Applying CompletableFuture Factory Methods

- Use supplyAsync() to multiply BigFraction objects

```java
String f1 = "62675744/15668936";
String f2 = "609136/913704";
CompletableFuture<BigFraction> future = CompletableFuture.supplyAsync(() -> {
    BigFraction bf1 = new BigFraction(f1);
    BigFraction bf2 = new BigFraction(f2);

    return bf1.multiply(bf2);
});

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

Define a supplier lambda that multiplies two BigFractions.
Applying CompletableFuture Factory Methods

- Use `supplyAsync()` to multiply `BigFraction` objects

```java
String f1 = "62675744/15668936";
String f2 = "609136/913704";
CompletableFuture<BigFraction> future = CompletableFuture.supplyAsync(() -> {
    BigFraction bf1 = new BigFraction(f1);
    BigFraction bf2 = new BigFraction(f2);
    return bf1.multiply(bf2);
});

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

These computations run concurrently
Apply CompletableFuture Factory Methods

- Use supplyAsync() to multiply BigFraction objects

```java
String f1 = "62675744/15668936";
String f2 = "609136/913704";
CompletableFuture<BigFraction> future = 
    CompletableFuture.supplyAsync(() -> {
        BigFraction bf1 = 
            new BigFraction(f1); 
        BigFraction bf2 = 
            new BigFraction(f2); 

        return bf1.multiply(bf2); 
    });

System.out.println(future.join().toMixedString());
```
Evaluating Completable Future Factory Methods
Evaluating CompletableFuture Factory Methods

- Pros of using CompletableFuture.supplyAsync()
Evaluating CompletableFuture Factory Methods

• Pros of using CompletableFuture.supplyAsync()
  • No need to explicitly complete the future since supplyAsync() returns one

```java
completableFuture<BigFraction> future = CompletableFuture.supplyAsync(() -> {
    BigFraction bf1 = new BigFraction(f1);
    BigFraction bf2 = new BigFraction(f2);
    return bf1.multiply(bf2);
});
...
System.out.println(future.join().toMixedString());
```
Evaluating CompletableFuture Factory Methods

- Pros of using CompletableFuture.supplyAsync()
  - No need to explicitly complete the future since supplyAsync() returns one
  - Avoids the explicit creation/use of threads

```java
CompletableFuture<BigFraction> future = CompletableFuture
  .supplyAsync(() -> {
      BigFraction bf1 = new BigFraction(f1);
      BigFraction bf2 = new BigFraction(f2);
      return bf1.multiply(bf2);
  });
...
System.out.println(future.join().toMixedString());
```

The supplier lambda runs in the Java common fork-join pool
Evaluating CompletableFuture Factory Methods

- Cons of using CompletableFuture.supplyAsync()
Evaluating CompletableFuture Factory Methods

- Cons of using CompletableFuture.supplyAsync()
- We still must fix the problem with calling join()

```java
CompletableFuture<BigFraction> future = CompletableFuture
    .supplyAsync(() -> {
        BigFraction bf1 = new BigFraction(f1);
        BigFraction bf2 = new BigFraction(f2);

        return bf1.multiply(bf2);
    });

... System.out.println(future.join().toMixedString());
```
Evaluating CompletableFuture Factory Methods

- Cons of using CompletableFuture.supplyAsync()
  - We still must fix the problem with calling join()

```java
CompletableFuture<BigFraction> future = CompletableFuture.supplyAsync(() -> {
    BigFraction bf1 = new BigFraction(f1);
    BigFraction bf2 = new BigFraction(f2);
    return bf1.multiply(bf2);
});
...
System.out.println(future.join().toMixedString());
```

Addressing this issue motivates the advanced Java CompletableFuture features!
End of Advanced Java
CompletableFuture Features:
Applying Factory Methods