Understand 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
Applying Completable Future Factory Methods
Applying CompletableFuture Factory Methods

• Using supplyAsync() to multiply big fractions

```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

- Using `supplyAsync()` to multiply big fractions

```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 [docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#supplyAsync](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#supplyAsync)
Applying CompletableFuture Factory Methods

- Using `supplyAsync()` to multiply big fractions

```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**
• Using `supplyAsync()` to multiply big fractions

```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
Using `supplyAsync()` to multiply big fractions

```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());
```

**Diagram:**
- `supplyAsync`: Blocks until the result is complete.
- `join`: Blocks until the result is complete.
Applying CompletableFuture Factory Methods

- Using `supplyAsync()` to multiply big fractions

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

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

Calling `CompletableFuture.supplyAsync()` avoids the use of threads in this example!
There’s no need to explicitly complete the future since `supplyAsync()` returns one
• Using supplyAsync() to multiply big fractions

```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());
```

However, we still must fix the problem with calling join() explicitly.
End of Understand Advanced Java CompletableFuture Features: Applying Factory Methods