Overview of Advanced Java 8
CompletableFuture Features (Part 1)

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

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:

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html)
Learning Objectives in this Part of the Lesson

• Understand advanced features of completable futures, e.g.
• Factory methods that initiate async computations
Factory Methods Initiate Async Computations
Four factory methods initiate async computations

See en.wikipedia.org/wiki/Factory_method_pattern
Four factory methods initiate async computations.
These computations may or may not return a value.
Factory Methods Initiate Async Computations

- Four factory methods initiate async computations
- These computations may or may not return a value
- `supplyAsync()` allows two-way calls via a supplier

<table>
<thead>
<tr>
<th>Methods</th>
<th>Params</th>
<th>Returns</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>supplyAsync</code></td>
<td><code>Supplier</code></td>
<td><code>Completable Future with result of Supplier</code></td>
<td>Asynchronously run supplier in common fork/join pool</td>
</tr>
<tr>
<td><code>supplyAsync</code></td>
<td><code>Supplier, Executor</code></td>
<td><code>Completable Future with result of Supplier</code></td>
<td>Asynchronously run supplier in given executor pool</td>
</tr>
</tbody>
</table>

See [docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html](http://docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html)
Four factory methods initiate async computations

These computations may or may not return a value

supplyAsync() allows two-way calls via a supplier

Can be passed params & returns a value

```
String f1 = "62675744/15668936";
String f2 = "609136/913704";

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

            return bf1.multiply(bf2);
        });
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8
Factory Methods Initiate Async Computations

- Four factory methods initiate async computations
- These computations may or may not return a value
- `supplyAsync()` allows two-way calls via a supplier
- Can be passed params & returns a value

```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);
});
```

See javarevisited.blogspot.com/2015/03/what-is-effectively-final-variable-of.html
Four factory methods initiate async computations
- These computations may or may not return a value
  - supplyAsync() allows two-way calls via a supplier
  - runAsync() enables one-way calls via a runnable

<table>
<thead>
<tr>
<th>Methods</th>
<th>Params</th>
<th>Returns</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>run Async</td>
<td>Runnable</td>
<td>Completable Future with result of Void</td>
<td>Asynchronously run runnable in common fork/join pool</td>
</tr>
<tr>
<td>run Async</td>
<td>Runnable, Executor</td>
<td>Completable Future with result of Void</td>
<td>Asynchronously run runnable in given executor pool</td>
</tr>
</tbody>
</table>

See [docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html](docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html)
Four factory methods initiate async computations

- These computations may or may not return a value
- supplyAsync() allows two-way calls via a supplier
- runAsync() enables one-way calls via a runnable
- Can be passed params, but returns no values

```java
String f1 = "62675744/15668936";
String f2 = "609136/913704";

CompletableFuture<Void> future = CompletableFuture.runAsync(() -> {
    BigFraction bf1 = new BigFraction(f1);
    BigFraction bf2 = new BigFraction(f2);

    System.out.println(bf1.multiply(bf2).toMixedString());
});
```

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8]
Factory Methods Initiate Async Computations

- Four factory methods initiate async computations
  
- These computations may or may not return a value
  
  - supplyAsync() allows two-way calls via a supplier
  
  - runAsync() enables one-way calls via a runnable
  
  - Can be passed params, but returns no values

```java
String f1 = "62675744/15668936";
String f2 = "609136/913704";

CompletableFuture<Void> future = CompletableFuture.runAsync(() -> {
    BigFraction bf1 = new BigFraction(f1);
    BigFraction bf2 = new BigFraction(f2);

    System.out.println(bf1.multiply(bf2).toMixedString());
});
```

"Void" is not a value!
Factory Methods Initiate Async Computations

- Four factory methods initiate async computations
- These computations may or may not return a value
  - supplyAsync() allows two-way calls via a supplier
  - runAsync() enables one-way calls via a runnable
- Can be passed params, but returns no values

Any output must therefore come from "side-effects"

```java
String f1 = "62675744/15668936";
String f2 = "609136/913704";

CompletableFuture<Void> future = CompletableFuture.runAsync(() -> {
    BigFraction bf1 = new BigFraction(f1);
    BigFraction bf2 = new BigFraction(f2);
    System.out.println(bf1.multiply(bf2).toMixedString());
});
```
Factory Methods Initiate Async Computations

- Four factory methods initiate async computations
  - These computations may or may not return a value
- Async functionality runs in a thread pool
Factory Methods Initiate Async Computations

- Four factory methods initiate async computations
  - These computations may or may not return a value
- Async functionality runs in a thread pool

This thread pool defaults to common fork-join pool, but can be given explicitly.
Applying Completable Future 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
• 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());
```
• 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
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());
```
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());
```

Calling `CompletableFuture.supplyAsync()` avoids the use of threads in this example!
There's no need to explicitly complete the future since `supplyAsync()` returns one.
Applying Completable Future 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());
```

However, we still have the problem with having to call `join()` explicitly.
End of Overview of Advanced Java 8 Completable Future Features (Part 1)