Advanced Java CompletableFuture Features: Introducing 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

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html
Factory Methods Initiate Async Computations
• Four factory methods initiate async computations

See en.wikipedia.org/wiki/Factory_method_pattern
Factory Methods Initiate Async Computations

- Four factory methods initiate async computations
- These computations may or may not return a value

```java
CompletableFuture.supplyAsync(Supplier<T> supplier)
CompletableFuture.supplyAsync(Supplier<T>, Executor executor)
CompletableFuture.runAsync(Runnable task)
CompletableFuture.runAsync(Runnable task, Executor executor)
```
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>supply Async</td>
<td>Supplier</td>
<td>Completable Future with result of Supplier</td>
<td>Asynchronously run supplier in common fork/join pool</td>
</tr>
<tr>
<td>supply Async</td>
<td>Supplier, Executor</td>
<td>Completable Future with result of Supplier</td>
<td>Asynchronously run supplier in given executor pool</td>
</tr>
</tbody>
</table>
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

```
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 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

```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

Params are passed as "effectively final" objects to the supplier lambda
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(() -> {
        BigFraction bf1 =
            new BigFraction(f1);
        BigFraction bf2 =
            new BigFraction(f2);

        return bf1.multiply(bf2);
    });
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><code>runAsync</code></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><code>runAsync</code></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](https://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

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
- Returns no value

```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
  - Returns no value

```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).toString());
});
```

Any output must therefore come from “side-effects”
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

supplyAsync() is more commonly used than runAsync() in practice
Four factory methods initiate async computations

- These computations may or may not return a value

Async functionality runs in a thread pool

Help make programs more *elastic* by leveraging a pool of worker threads
Four factory methods initiate async computations
- These computations may or may not return a value
- Async functionality runs in a thread pool

By default, the common fork-join pool is used

See dzone.com/articles/common-fork-join-pool-and-streams
Four factory methods initiate async computations:
- These computations may or may not return a value.
- Async functionality runs in a thread pool.

However, a pre- or user-defined thread pool can also be given.
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
  - In contrast, Java parallel streams are designed for use with the common fork-join pool

See lesson on “Java Parallel Stream Internals: Parallel Processing via the Common Fork-Join Pool”
End of Advanced Java
CompletableFuture Features:
Introducing Factory Methods