Understand 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](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
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>Supplier</td>
<td>Completable Future with result of Supplier</td>
<td>Asynchronously run supplier in common fork/join pool</td>
</tr>
<tr>
<td><code>supplyAsync</code></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>

See [docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html](https://docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html)
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()
   .supplyAsync(() -> {
      BigFraction bf1 = new BigFraction(f1);
      BigFraction bf2 = new BigFraction(f2);

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

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8](https://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);
});
```

Params are passed as "effectively final" objects to the supplier lambda

See javarevisited.blogspot.com/2015/03/what-is-effectively-final-variable-of.html
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 [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8](https://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

<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)
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

```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](https://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

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());
});

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
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

By default, the common fork-join pool is used

See dzone.com/articles/common-fork-join-pool-and-streams
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

However, a pre- or user-defined thread pool can also be given

```java
CompletableFuture()
cancel(boolean): boolean
isCancelled(): boolean
isDone(): boolean
get()
get(long, TimeUnit)
join()
complete(T): boolean
supplyAsync(Supplier<T>, Executor): CompletableFuture<T>
runAsync(Runnable, Executor): CompletableFuture<Void>
```
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 Understand Advanced Java
CompletableFuture Features:
Introducing Factory Methods