Advanced Java CompletableFuture Features: Factory Method Internals

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
  - Internals of factory methods
    - i.e., how `supplyAsync()` runs a supplier lambda param asynchronously & concurrently

Diagram:
- CompletionStage methods
- Exception methods
- Arbitrary-arity methods
- Basic methods
- Factory methods
Mapping `supplyAsync()` to the Common Fork-Join Pool
Mapping supplyAsync() to the Common Fork-Join Pool

- supplyAsync() arranges to run the supplier lambda param concurrently & asynchronously in a thread residing in the Java common fork-join pool

```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
Mapping supplyAsync() to the Common Fork-Join Pool

- supplyAsync() arranges to run the supplier lambda param concurrently & asynchronously in a thread residing in the Java common fork-join pool

```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
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());

Instead, it return a future that’s completed by a worker thread running in common fork-join pool

See dzone.com/articles/be-aware-of-forkjoinpoolcommonpool
supplyAsync() arranges to run the supplier lambda param concurrently & asynchronously in a thread residing in the Java common fork-join pool

```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());
```

- `supplyAsync()`'s param is a supplier lambda that multiplies two BigFraction objects
String \texttt{f1}("62675744/15668936"); String \texttt{f2}("609136/913704");

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

    return \texttt{bf1.multiply(bf2);});

System.out.println(\texttt{future.join().toMixedString()});

Although Supplier.get() takes no params, effectively final values can be passed to this supplier lambda.
Internals of CompletableFuture Factory Methods

- `supplyAsync()` arranges to run the supplier lambda param concurrently & asynchronously in a thread residing in the Java common fork-join pool

```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)
Internals of Completable Future Factory Methods
Internals of CompletableFuture Factory Methods

- supplyAsync() is implemented by leveraging a message-passing framework that feeds tasks to the Java common fork-join pool

```java
CompletableFuture<U> supplyAsync(Supplier<U> supplier) {
    CompletableFuture<U> f =
        new CompletableFuture<U>();
    execAsync(ForkJoinPool.commonPool(),
              new AsyncSupply<U>(supplier, f));
    return f;
}
```

See classes/java/util/concurrent/CompletableFuture.java
Internals of Completable Future Factory Methods

- `supplyAsync()` is implemented by leveraging a message-passing framework that feeds tasks to the Java common fork-join pool

```java
public static <U> CompletableFuture<U> supplyAsync(Supplier<U> supplier) {
    CompletableFuture<U> f =
        new CompletableFuture<U>();
    execAsync(ForkJoinPool.commonPool(),
        new AsyncSupply<U>(supplier, f));
    return f;
}
```

The supplier parameter is bound to the lambda passed to `supplyAsync()`.
Internals of Completable Future Factory Methods

- supplyAsync() is implemented by leveraging a message-passing framework that feeds tasks to the Java common fork-join pool

```java
<U> CompletableFuture<U> supplyAsync(Supplier<U> supplier) {
    CompletableFuture<U> f =
        new CompletableFuture<U>();

    execAsync(ForkJoinPool.commonPool(),
              new AsyncSupply<U>(supplier, f));

    return f;
}
```

An “incomplete” future is created here

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#CompletableFuture](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#CompletableFuture)
Internals of Completable Future Factory Methods

- `supplyAsync()` is implemented by leveraging a message-passing framework that feeds tasks to the Java common fork-join pool

```java
<U> CompletableFuture<U> supplyAsync(Supplier<U> supplier) {
    CompletableFuture<U> f =
        new CompletableFuture<U>();

    execAsync(ForkJoinPool.commonPool(),
              new AsyncSupply<U>(supplier, f));

    return f;
}
```

The supplier & incomplete future are encapsulated in an AsyncSupply message
Internals of Completable Future Factory Methods

- supplyAsync() is implemented by leveraging a message-passing framework that feeds tasks to the Java common fork-join pool

```java
<U> CompletableFuture<U> supplyAsync(Supplier<U> supplier) {
   CompletableFuture<U> f = new CompletableFuture<U>();
   execAsync(ForkJoinPool.commonPool(),
             new AsyncSupply<U>(supplier, f));

   return f;
}
```

This message is enqueued for async execution in common fork-join pool.

This design is one example of “message passing” a la Reactive programming!
Internals of Completable Future Factory Methods

- `supplyAsync()` is implemented by leveraging a message-passing framework that feeds tasks to the Java common fork-join pool

```java
<U> CompletableFuture<U> supplyAsync(Supplier<U> supplier) {
    ... 
    CompletableFuture<U> f =
        new CompletableFuture<U>();

    execAsync(ForkJoinPool.commonPool(),
              new AsyncSupply<U>(supplier, f));

    return f;
}
```

The incomplete future is returned to the caller for subsequent use (e.g., with completion stage methods)
AsyncSupply is a nested class that executes the supplier lambda param in a thread residing in the Java common fork-join pool.

```java
static final class AsyncSupply<U> extends Async {
    final Supplier<U> fn;
    final CompletableFuture<U> dst;

    AsyncSupply(Supplier<U> fn, CompletableFuture<T> dst) {
        this.fn = fn;
        this.dst = dst;
    }

    public final boolean exec() {
        ...
        U u = fn.get();
        ...
        d.internalComplete(u, ex);
        ...
    }
}
```

See `classes/java/util/concurrent/CompletableFuture.java`
Internals of Completable Future Factory Methods

- AsyncSupply is a nested class that executes the supplier lambda param in a thread residing in the Java common fork-join pool.

```java
static final class AsyncSupply<U> extends Async {
    final Supplier<U> fn;
    final CompletableFuture<U> dst;

    AsyncSupply(Supplier<U> fn, CompletableFuture<T> dst) {
        this.fn = fn;
        this.dst = dst;
    }

    public final boolean exec() {
        ...
        U u = fn.get();
        ...
        dst.internalComplete(u, ex);
        ...
    }
}
```

Async extends ForkJoinTask & Runnable so it can be executed.

See classes/java/util/concurrent/CompletableFuture.java
Internals of Completable Future Factory Methods

- AsyncSupply is a nested class that executes the supplier lambda param in a thread residing in the Java common fork-join pool

```java
static final class AsyncSupply<U> extends Async {
    final Supplier<U> fn;
    final CompletableFuture<U> dst;

    AsyncSupply(Supplier<U> fn, CompletableFuture<T> dst) {
        this.fn = fn;
        this.dst = dst;
    }

    public final boolean exec() {
        ...
        U u = fn.get();
        ...
        dst.internalComplete(u, ex);
        ...
    }
}
```

AsyncSupply stores the original supplier lambda passed into supplyAsync()
Internals of Completable Future Factory Methods

- AsyncSupply is a nested class that executes the supplier lambda param in a thread residing in the Java common fork-join pool

```java
static final class AsyncSupply<U> extends Async {
    final Supplier<U> fn;
    final CompletableFuture<U> dst;

    AsyncSupply(Supplier<U> fn, CompletableFuture<T> dst) {
        this.fn = fn;
        this.dst = dst;
    }

    public final boolean exec() {
        ...
        U u = fn.get();
        ...
        d.internalComplete(u, ex);
        ...
    }
}
```

A worker thread then runs the supplier lambda asynchronously & stores the result
Internals of Completable Future Factory Methods

- AsyncSupply is a nested class that executes the supplier lambda param in a thread residing in the Java common fork-join pool

```java
static final class AsyncSupply<U> extends Async {
    final Supplier<U> fn;
    final CompletableFuture<U> dst;

    AsyncSupply(Supplier<U> fn, CompletableFuture<T> dst) {
        this.fn = fn; this.dst = dst;
    }

    public final boolean exec() {
        ...
        U u = fn.get();
        ...
        d.internalComplete(u, ex);
        ...
    }
}
```

get() can use the ForkJoinPool Managed Blocker mechanism to auto-scale the common pool size for blocking operations

See earlier lesson on “The Java Fork-Join Pool: the ManagedBlocker Interface”
Internals of Completable Future Factory Methods

AsyncSupply is a nested class that executes the supplier lambda param in a thread residing in the Java common fork-join pool.

```java
static final class AsyncSupply<U> extends Async {
    final Supplier<U> fn;
    final CompletableFuture<U> dst;

    AsyncSupply(Supplier<U> fn, CompletableFuture<T> dst) {
        this.fn = fn; this.dst = dst;
    }

    public final boolean exec() {
        ...
        U u = fn.get();
        ...
        dst.internalComplete(u, ex);
        ...
    }
}
```

Triggers completion of the future using the encoding of the given arguments.
End of Advanced Java
CompletableFuture Features:
Factory Method Internals