Understand 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

- CompletionStage methods
- Factory methods
- Exception methods
- Arbitrary-arity methods
- Basic methods
Internals of Completable
Future Factory Methods
Internals of CompletableFuture Factory Methods

- This supplyAsync() method arranges to run the supplier lambda param in a thread residing in the 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
Internals of CompletableFuture Factory Methods

• This supplyAsync() method arranges to run the supplier lambda param in a thread residing in the 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() does not create a new thread!

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#supplyAsync
Internals of CompletableFuture Factory Methods

• This supplyAsync() method arranges to run the supplier lambda param in a thread residing in the 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());
```

Instead, it return a future that’s completed by a worker thread running in common fork-join pool
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());
Internals of CompletableFuture Factory Methods

- This supplyAsync() method arranges to run the supplier lambda param in a thread residing in the common fork-join pool

```
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());
```

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

See javarevisited.blogspot.com/2015/03/what-is-effectively-final-variable-of.html
Internals of CompletableFuture Factory Methods

• This supplyAsync() method arranges to run the supplier lambda param in a thread residing in the common fork-join pool

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

CompletableFuture<BigFraction> future = CompletableFuture.of(() -> {
    BigFraction bf1 = new BigFraction(f1);
    BigFraction bf2 = new BigFraction(f2);
    return bf1.multiply(bf2);});

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

The worker thread calls the Supplier. get() method to obtain this supplier lambda & perform the computation

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#supplyAsync
Internals of Completable Future Factory Methods

- This `supplyAsync()` method arranges to run the supplier lambda param in a thread residing in the 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;
}
```

See classes/java/util/concurrent/CompletableFuture.java
The supplier parameter is bound to the lambda passed to `supplyAsync()` method arranges to run the supplier lambda param in a thread residing in the common fork-join pool

```
<U> CompletableFuture<U> supplyAsync(Supplier<U> supplier) {
    CompletableFuture<U> f =
        new CompletableFuture<U>();
    execAsync(ForkJoinPool.commonPool(),
        new AsyncSupply<U>(supplier, f));
    return f;
}
...
Internals of Completable Future Factory Methods

• This supplyAsync() method arranges to run the supplier lambda param in a thread residing in the 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 is encapsulated in an AsyncSupply message.
Internals of Completable Future Factory Methods

• This `supplyAsync()` method arranges to run the supplier lambda param in a thread residing in the 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

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

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

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

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

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 common fork-join pool.

```java
static final class AsyncSupply<U> extends Async {
  final Supplier<U> fn;
  
  AsyncSupply(Supplier<U> fn, ...) { this.fn = fn; ... }

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

Async extends ForkJoinTask & Runnable so it can be executed in a thread pool.

See classes/java/util/concurrent/CompletableFuture.java
AsyncSupply is a nested class that executes the supplier lambda param in a thread residing in the common fork-join pool

... static final class AsyncSupply<U> extends Async {
    final Supplier<U> fn;

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

    public final boolean exec() {
        ...
        U u = fn.get();
        ...
    }
}

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

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

... static final class AsyncSupply<U> extends Async {
    final Supplier<U> fn;

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

    public final boolean exec() {
        ... 
        U u = fn.get();
        ... 
    }
}

A worker thread in the pool then runs the supplier lambda asynchronously
AsyncSupply is a nested class that executes the supplier lambda param in a thread residing in the common fork-join pool

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

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

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

This `get()` method could use ForkJoinPool ManagedBlocker mechanism to auto-scale the pool size for blocking operations.

See earlier lesson on “The Java Fork-Join Pool: the ManagedBlocker Interface”
End of Understand Advanced Java

CompletableFuture Features:
Factory Method Internals