Overview of Advanced Java 8
CompletableFuture Features (Part 4)

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 that initiate async functionality
  - Completion stage methods used to chain together actions that perform async result processing & composition
- Arbitrary-arity methods that process futures in bulk

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html
Arbitrary-Arity Methods
Process Futures in Bulk
**Arbitrary-Arity Methods Process Futures in Bulk**

- Arbitrary-arity methods return futures that are triggered after completion of some/all futures.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Params</th>
<th>Returns</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>allOf</td>
<td>Varags</td>
<td>Completable Future&lt;Void&gt;</td>
<td>Return a future that completes when all futures in params complete</td>
</tr>
<tr>
<td>anyOf</td>
<td>Varargs</td>
<td>Completable Future&lt;Void&gt;</td>
<td>Return a future that completes when any future in params complete</td>
</tr>
</tbody>
</table>
Arbitrary-Arity Methods Process Futures in Bulk

- Arbitrary-arity methods return futures that are triggered after completion of some/all futures.
- The returned future can be used to wait for any or all of $N$ completable futures in an array to complete.

```java
CompletableFuture.allOf(CompletableFuture[] futures)
CompletableFuture.anyOf(CompletableFuture[] futures)
```
Arbitrary-Arity Methods Process Futures in Bulk

- Arbitrary-arity methods return futures that are triggered after completion of some/all futures.
- The returned future can be used to wait for any or all of \( N \) completable futures in an array to complete.

These "arbitrary-arity" methods are hard to program without using wrappers.
Arbitrary-Arity Methods Process Futures in Bulk

- Arbitrary-arity methods return futures that are triggered after completion of some/all futures.
- The returned future can be used to wait for any or all of $N$ completable futures in an array to complete.

We focus on `allOf()`, which is like `thenCombine()` on steroids!

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#allOf](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#allOf)
**Arbitrary-Arity Methods Process Futures in Bulk**

- FuturesCollector returns a completable future to a list of big fractions that are being reduced and multiplied asynchronously

```java
static void testFractionMultiplications1() {
    ...
    Stream.generate(() -> makeBigFraction(new Random(), false))
        .limit(sMAX_FRACTIONS)
        .map(reduceAndMultiplyFractions)
        .collect(FuturesCollector.toFuture())
        .thenAccept(this::sortAndPrintList);
}
```

`collect()` converts a stream of completable futures into a single completable future.

See [github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex8](https://github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex8)
Arbitrary-Arity Methods Process Futures in Bulk

- FuturesCollector provides a wrapper for allOf()

```java
FuturesCollector()  
supplier(): Supplier<List<CompletableFuture<T>>>  
accumulator(): BiConsumer<List<CompletableFuture<T>>, CompletableFuture<T>>  
combiner(): BinaryOperator<CompletableFuture<T>>  
finisher(): Function<List<CompletableFuture<T>>, CompletableFuture<List<T>>>  
characteristics(): Set  
toFuture(): Collector<CompletableFuture<T>, ?, CompletableFuture<List<T>>>
```

See [Java8/ex8/utils/FuturesCollector.java](Java8/ex8/utils/FuturesCollector.java)
FuturesCollector provides a wrapper for allOf()

- Converts a *stream* of completable futures into a *single* completable future that’s triggered when *all* futures in the stream complete

**Java Interface**

```java
interface Collector<T, A, R> {
    Supplier<A> supplier();
    BiConsumer<A, T> accumulator();
    BinaryOperator<T> combiner();
    Function<T, R> finisher();
    Set<Characteristics> characteristics();
}
```

**Java Class**

```java
public class FuturesCollector<T> {
    FuturesCollector()
    Supplier<List<CompletableFuture<T>>> supplier();
    BiConsumer<List<CompletableFuture<T>>, CompletableFuture<T>> accumulator();
    BinaryOperator<CompletableFuture<T>> combiner();
    Function<List<CompletableFuture<T>>, CompletableFuture<List<T>>> finisher();
    Set<Characteristics> characteristics();
    CompletableFuture<List<T>> toFuture();
}
```

FuturesCollector is a non-concurrent collector (supports parallel & sequential streams)
Arbitrary-Arity Methods Process Futures in Bulk

- FuturesCollector provides a wrapper for allOf()
  - Converts a stream of completable futures into a single completable future that’s triggered when all futures in the stream complete
- Implements the Collector interface that accumulates input elements into a mutable result container

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html](docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html)
Arbitrary-Arity Methods Process Futures in Bulk

- FuturesCollector provides a wrapper for allOf()

```java
public interface Collector<T, A, R> {
    supplier(): Supplier<A>
    accumulator(): BiConsumer<List<CompletableFuture<T>>, CompletableFuture<T>>
    combiner(): BinaryOperator<List<CompletableFuture<T>>>
    finisher(): Function<List<CompletableFuture<T>>, CompletableFuture<List<T>>>
    characteristics(): Set<Characteristics>
}
```

```java
public class FuturesCollector<T> {
    FuturesCollector()
    supplier(): Supplier<List<CompletableFuture<T>>>
    accumulator(): BiConsumer<List<CompletableFuture<T>>, CompletableFuture<T>>
    combiner(): BinaryOperator<List<CompletableFuture<T>>>
    finisher(): Function<List<CompletableFuture<T>>, CompletableFuture<List<T>>>
    characteristics(): Set
    toFuture(): Collector<CompletableFuture<T>, ?, CompletableFuture<List<T>>>
}
```

FuturesCollector provides a powerful wrapper for some complex code!!!
• **FuturesCollector** provides a wrapper for `allOf()`

```java
public class FuturesCollector<T>
    implements Collector<CompletableFuture<T>,
    List<CompletableFuture<T>>,
    CompletableFuture<List<T>>> {
    ...
```

*Implements a custom collector*

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html](http://docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html)
FuturesCollector provides a wrapper for allOf()

```java
public class FuturesCollector<T>
    implements Collector<CompletableFuture<T>,
                        List<CompletableFuture<T>>,
                        CompletableFuture<List<T>>> {
...
• FuturesCollector provides a wrapper for allOf()

```java
public class FuturesCollector<T>
    implements Collector<CompletableFuture<T>,
    List<CompletableFuture<T>>,
    CompletableFuture<List<T>>> {
...
```

The mutable accumulation type of the accumulator() method
Arbitrary-Arity Methods Process Futures in Bulk

• FuturesCollector provides a wrapper for allOf()

```java
public class FuturesCollector<T>
    implements Collector<CompletableFuture<T>,
                    List<CompletableFuture<T>>,
                    CompletableFuture<List<T>>> {
```

... 

*The result type of the finisher() method, i.e., the final output of the collector*
Arbitrary-Arity Methods Process Futures in Bulk

- FuturesCollector provides a wrapper for allOf()

```java
class FuturesCollector<T> implements Collector<CompletableFuture<T>,
    List<CompletableFuture<T>>,
    CompletableFuture<List<T>>> {

    public Supplier<List<CompletableFuture<T>>> supplier() {
        return ArrayList::new;
    }

    public BiConsumer<List<CompletableFuture<T>>,
        CompletableFuture<List<T>>> accumulator() {
        return List::add;
    }

    ...
}
```

*This factory method returns a supplier used by the Java 8 streams collector framework to create a new mutable array list container*
• FuturesCollector provides a wrapper for allOf()

```java
public class FuturesCollector<T>
    implements Collector<CompletableFuture<T>,
                        List<CompletableFuture<T>>,
                        CompletableFuture<List<T>>> {
    public Supplier<List<CompletableFuture<T>>> supplier() {
        return ArrayList::new;
    }

    public BiConsumer<List<CompletableFuture<T>>,
                      CompletableFuture<T>> accumulator()
    { return List::add; }

    ...
}
```

*This factory method returns a bi-consumer used by the Java 8 streams collector framework to add a new completable future into the mutable array list container*

*This method is only ever called in a single thread (so no locks are needed)*
Arbitrary-Arity Methods Process Futures in Bulk

• FuturesCollector provides a wrapper for allOf()

```java
class FuturesCollector<T> {
  ...
  public BinaryOperator<List<CompletableFuture<T>>> combiner() {
    return (List<CompletableFuture<T>> one, List<CompletableFuture<T>> another) -> {
      one.addAll(another);
      return one;
    };
  }
}
```

This factory method returns a binary operator that merges two partial array list results into a single array list (only relevant for parallel streams)

This method is only ever called in a single thread (so no locks are needed)
Arbitrary-Arity Methods Process Futures in Bulk

• FuturesCollector provides a wrapper for allOf()

```java
public class FuturesCollector<T>

    public Function<List<CompletableFuture<T>>, CompletableFuture<List<T>>> finisher() {
        return futures -> CompletableFuture.allOf(futures.toArray(new CompletableFuture[0])).
            thenApply(v -> futures.stream().map(CompletableFuture::join).
            collect(toList()));
    }

This factory method returns a function used by the Java 8 streams collector framework to transform the array list accumulation type to the completable future result type
```

• FuturesCollector provides a wrapper for allOf()
FuturesCollector provides a wrapper for allOf()

```java
public class FuturesCollector<T>
{
    ...

    public Function<List<CompletableFuture<T>>, CompletableFuture<List<T>>> finisher() {
        return futures -> CompletableFuture.allOf(futures.toArray(new CompletableFuture[0]))
            .thenApply(v -> futures.stream()
                .map(CompletableFuture::join)
                .collect(toList()));
    }
}
```

Convert list of futures to array of futures & pass to allOf() to obtain a future that will complete when all futures complete
Arbitrary-Arity Methods Process Futures in Bulk

- FuturesCollector provides a wrapper for `allOf()`

```java
public class FuturesCollector<T>
    ...

    public Function<List<CompletableFuture<T>>, CompletableFuture<List<T>>> finisher() {
        return futures -> CompletableFuture.allOf(futures.toArray(new CompletableFuture[0])).
            thenApply(v -> futures.stream().map(CompletableFuture::join).collect(toList()));
    }
```

*When all futures have completed get a single future to a list of joined elements of type T*
Arbitrary-Arity Methods Process Futures in Bulk

- FuturesCollector provides a wrapper for allOf()

```java
public class FuturesCollector<T> {
    ...
    public Function<List<CompletableFuture<T>>, CompletableFuture<List<T>>> finisher() {
        CompletableFuture<List<T>> finisher()
        return futures -> CompletableFuture.allOf(futures.toArray(new CompletableFuture[0]))
            .thenApply(v -> futures.stream()
                .map(CompletableFuture::join)
                .collect(toList()));
    }
    ...
```
Arbitrary-Arity Methods Process Futures in Bulk

- FuturesCollector provides a wrapper for allOf()

```java
public class FuturesCollector<T> {
    ...
    
    public Function<List<CompletableFuture<T>>, CompletableFuture<List<T>>> finisher() {
        return futures -> CompletableFuture.allOf(futures.toArray(new CompletableFuture[0]))
            .thenApply(v -> futures.stream()
            .map(CompletableFuture::join)
            .collect(toList()));
    }
    ...
}
```

*This call to join() will never block!*
Arbitrary-Arity Methods Process Futures in Bulk

- FuturesCollector provides a wrapper for allOf()

```java
public class FuturesCollector<T> {
    ...
    public Function<List<CompletableFuture<T>>, CompletableFuture<List<T>>> finisher() {
        return futures -> CompletableFuture.allOf(futures.toArray(new CompletableFuture[0]))
            .thenApply(v -> futures.stream()
                .map(CompletableFuture::join)
                .collect(toList()));
    }
    ...
}
```

Return a future to a list of elements of T
• FuturesCollector is used to return a completable future to a list of big fractions that are being reduced and multiplied asynchronously

```java
static void testFractionMultiplications1() {
    ...
    Stream.generate(() -> makeBigFraction(new Random(), false))
        .limit(sMAX_FRACTIONS)
        .map(reduceAndMultiplyFraction)
        .collect(FuturesCollector.toFuture())
        .thenAccept(this::sortAndPrintList);
}
```

`thenAccept()` is called only when the future returned from `collect()` completes
FuturesCollector provides a wrapper for `allOf()`

```java
public class FuturesCollector<T> {
    ... 
    public Set characteristics() {
        return Collections.singleton(Characteristics.UNORDERED);
    }
}
```

```java
public static <T> Collector<CompletableFuture<T>, ?, CompletableFuture<List<T>>>
    toFuture() {
    return new FuturesCollector<>();
}
```

FuturesCollector is thus a *non-concurrent* collector
Arbitrary-Arity Methods Process Futures in Bulk

- FuturesCollector provides a wrapper for allOf()

```java
class FuturesCollector<T> {
    ...
    public Set characteristics() {
        return Collections.singleton( Characteristics.UNORDERED );
    }
}
```

Static factory method creates a new FuturesCollector

```java
public static <T> Collector< CompletableFuture<T>, ?, CompletableFuture<List<T>>> toFuture() {
    return new FuturesCollector<>();
}
```
End of Overview of Advanced Java 8 Completable Future Features (Part 4)