Advanced Java CompletableFuture Features: Implementing FuturesCollector

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
  - Completion stage methods chain together actions to perform async result processing & composition
  - Arbitrary-arity methods that process futures in bulk
  - Provide a wrapper for the allOf() method

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)
Implementing the FuturesCollector Class
Implementing the FuturesCollector Class

- 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/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8)
Implementing the FuturesCollector Class

- FuturesCollector provides a wrapper for allOf()

```java
<<Java Interface>>
Collector<T,A,R>
- supplier(): Supplier<A>
- accumulator(): BiConsumer<A,T>
- combiner(): BinaryOperator<A>
- finisher(): Function<A,R>
- characteristics(): Set<Characteristics>
```

```java
<<Java 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>>>
```

See [Java8/ex8/utils/FuturesCollector.java](Java8/ex8/utils/FuturesCollector.java)
Implementing the FuturesCollector Class

- 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

```
FuturesCollector

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
isFuture(): CompletableFuture<T>
```
Implementing the FuturesCollector Class

- 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/Collections.html
Implementing the FuturesCollector Class

• FuturesCollector provides a wrapper for allOf()

```
<Java Interface>>
Collector<T, A, R>
 supplier(): Supplier<A>
 accumulator(): BiConsumer<A, T>
 combiner(): BinaryOperator<A>
 finisher(): Function<A, R>
 characteristics(): Set<Characteristics>
```

```
<Java 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!!!
Implementing the FuturesCollector Class

- 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`
Implementing the FuturesCollector Class

- FuturesCollector provides a wrapper for allOf()

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

The type of input elements in the stream
Implementing the FuturesCollector Class

- FuturesCollector provides a wrapper for allOf()

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

The mutable result container type
```
Implementing the FuturesCollector Class

• 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 final output of the collector
```
Implementing the FuturesCollector Class

- 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 supplier used by the Java streams collector framework to create a new mutable array list container.
Implementing the FuturesCollector Class

• 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<T>> accumulator() {
        return List::add;
    }

    ...
```

This mutable result container stores a list of completable futures of type T
Implementing the FuturesCollector Class

- 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<T>> accumulator() {
        return List::add;
    }

    //...}
```

This factory method returns a bi-consumer used by the Java 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)
Implementing the FuturesCollector Class

- FuturesCollector provides a wrapper for allOf()

```java
public class FuturesCollector<T> {
    ...
    public <T> 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)
Implementing the FuturesCollector Class

- 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 streams collector framework to transform the array list multiple result container to the completable future result type.
Implementing the FuturesCollector Class

• FuturesCollector provides a wrapper for allOf()

```java
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()));
    }
    ...
}```

Reference to the mutable result contain, which is an ArrayList.
Implementing the FuturesCollector Class

- 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 the list of futures to an array of futures & pass to allOf() to obtain a future that will complete when all futures complete.
Implementing the FuturesCollector Class

- 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.
Implementing the FuturesCollector Class

- 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 the array list of futures into a stream of futures
Implementing the FuturesCollector Class

- 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()));
    }
    ...
```
Implementing the FuturesCollector Class

- 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
Implementing the FuturesCollector Class

- 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.

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8)
Implementing the FuturesCollector Class

- FuturesCollector provides a wrapper for allOf()

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

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

*FuturesCollector is thus a non-concurrent collector*
Implementing the FuturesCollector Class

• FuturesCollector provides a wrapper for allOf()

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

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

CompletableFuture Features:
Implementing FuturesCollector