Advanced Java Completable Future Features: Implementing the FuturesCollector Class

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 how arbitrary-arity methods process Completable Future objects in bulk
- Recognize the limitations with these methods
- Know how to address these limitations by wrapping the allOf() method to work seamlessly with Java’s Streams framework
- Learn how to implement the FuturesCollector class
  - Integrates the arbitrary-arity method allOf() into the Java Streams collector framework & returns a CompletableFuture to a List of objects that are being processed asynchronously & concurrently

See Java8/ex8/src/main/java/utils/FuturesCollector.java
Implementing the FuturesCollector Class
Implementing the FuturesCollector Class

- FuturesCollector implements all methods in the Collector interface

```java
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)
Implementing the FuturesCollector Class

- FuturesCollector implements all methods in the Collector interface

```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 implements all methods in the Collector interface

```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 implements all methods in the Collector interface

```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 implements all methods in the Collector interface

```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 implements all methods in the Collector interface

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

• FuturesCollector implements all methods in the Collector interface

```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 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 implements all methods in the Collector interface

```java
public 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)
Implementing the FuturesCollector Class

- FuturesCollector implements all methods in the Collector interface

```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).toList());
    }
    ...
}
```

This factory method returns a function used by the Java streams collector framework to transform the array list mutable result container to the completable future result type.
Implementing the FuturesCollector Class

- FuturesCollector implements all methods in the Collector interface

```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).toList());
    }

    ...
}
```

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

• FuturesCollector implements all methods in the Collector interface

```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).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 implements all methods in the Collector interface

```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)
        .toList());
  }

  ...
```

*FuturesCollector implements all methods in the Collector interface.*

*When all futures have completed get a single future to a list of joined elements of type T.*
Implementing the FuturesCollector Class

- FuturesCollector implements all methods in the Collector interface

```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)
                .toList());
    }
    ...
}
```

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

- FuturesCollector implements all methods in the Collector interface

```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)
                                 .toList());
    }
    ...
}
```

*This call to join() will never block!*
FuturesCollector implements all methods in the Collector interface

```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).toList());
    }
    ...
```

Return a future to a list of elements of T
Implementing the FuturesCollector Class

- FuturesCollector implements all methods in the Collector interface

```java
public class FuturesCollector<T> {
    ...
    public Set<Characteristics> 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 implements all methods in the Collector interface

```java
public class FuturesCollector<T>

    ...

    public Set<Characteristics> characteristics() {
        return Collections.singleton(Characteristics.UNORDERED);
    }

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

```
Implementing the FuturesCollector Class

- FuturesCollector is used to return a completable future to a list of big fractions that are being reduced & multiplied asynchronously

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

`toFuture()` returns an instance of the FuturesCollector for use with collect()

See lesson on "Advanced Java Completable Future Features: Applying Completion Stage Methods"
Implementing the FuturesCollector Class

- FuturesCollector is used to return a completable future to a list of big fractions that are being reduced & 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 after the future returned from collect() completes*
End of Advanced Java Completable Future Features: Implementing the FuturesCollector Class