Advanced Java Completable Future Features: Designing 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 arbitrary-arity methods

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html
Learning Objectives in this Part of the Lesson

- Understand how arbitrary-arity methods process CompletableFuture objects in bulk
- Recognize the limitations with arbitrary-arity methods
- Know how to address these limitations by wrapping the allOf() method to work seamlessly with the Java Streams framework

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html
Motivating the FuturesCollector Class
Motivating the FuturesCollector Class

- The arbitrary-arity methods are hard to program directly.

See en.wikipedia.org/wiki/Gordian_Knot
Motivating the FuturesCollector Class

- The arbitrary-arity methods are hard to program directly, e.g.
  - They’re passed an array of CompletableFuture objects
  - This typically requires converting from a List form to an array form

See en.wikipedia.org/wiki/Gordian_Knot
Motivating the FuturesCollector Class

- The arbitrary-arity methods are hard to program directly, e.g.
  - They’re passed an array of CompletableFuture objects
- They are not strongly-typed
  - They return a CompletableFuture to Void & Object types
• It’s therefore useful to create wrappers that encapsulate programmers from allOf() & anyOf() complexities!!
Motivating the FuturesCollector Class

- FuturesCollector is a wrapper that returns one CompletableFuture to a List of BigFraction objects being reduced & 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 lesson on "Advanced Java Completable Future Features: Applying Completion Stage Methods"
Motivating the FuturesCollector Class

- FuturesCollector is a wrapper that returns one CompletableFuture to a List of BigFraction objects being reduced & 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 lesson on "Advanced Java Completable Future Features: Applying Completion Stage Methods"
Designing the FuturesCollector Class
Designing the FuturesCollector Class

- FuturesCollector wraps allOf() to work with the Java Streams framework seamlessly

<<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>

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

See Java8/ex8/utils/FuturesCollector.java
Designing the FuturesCollector Class

- FuturesCollector wraps `allOf()` to work with the Java Streams framework seamlessly
- Converts a *stream* of CompletableFuture objects to a *single* CompletableFuture that triggers when *all* futures complete

```
<<Java Interface>>
Collector<T,A,R>

- supplier():Supplier<A>
- accumulator():BiConsumer<A,T>
- combiner():BinaryOperator<A>
- finisher():Function<T,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 is a non-concurrent collector (supports parallel & sequential streams)
Designing the FuturesCollector Class

- FuturesCollector wraps allOf() to work with the Java Streams framework seamlessly.
  - Converts a stream of CompletableFuture objects to a single CompletableFuture that triggers when all futures complete.
- Implements the Collector interface.

See docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html
Designing the FuturesCollector Class

- FuturesCollector wraps allOf() to work with the Java streams framework
  - Converts a *stream* of CompletableFuture objects to a *single* CompletableFuture that triggers when *all* futures complete
- Implements the Collector interface

A collector accumulates input stream elements into a mutable result container
Designing the FuturesCollector Class

• FuturesCollector provides a powerful wrapper for some otherwise complex code!

<<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>>>

See next part on "Advanced Java Completable Future Features: Implementing the FuturesCollector Class"
End of Advanced Java
CompletableFuture Features:
Designing the FuturesCollector Class