

The Java CompletableFuture ImageStream Gang Case Study: StreamOfFuturesCollector

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 the design of the Java completable future version of ImageStreamGang
- Know how to apply completable futures to ImageStreamGang, e.g.
 - Factory methods
 - Completion stage methods
 - Arbitrary-arity methods
 - Wrap the allOf() method to work with the Java Streams framework

<<Java Class>>	
G CompletableFuture<T>	
•	CompletableFuture()
•	cancel(boolean):boolean
•	isCancelled():boolean
•	isDone():boolean
•	get()
•	get(long,TimeUnit)
•	join()
•	complete(T):boolean
•	^S supplyAsync(Supplier<U>):CompletableFuture<U>
•	^S supplyAsync(Supplier<U>,Executor):CompletableFuture<U>
•	^S runAsync(Runnable):CompletableFuture<Void>
•	^S runAsync(Runnable,Executor):CompletableFuture<Void>
•	^S completedFuture(U):CompletableFuture<U>
•	thenApply(Function<?>):CompletableFuture<U>
•	thenAccept(Consumer<? super T>):CompletableFuture<Void>
•	thenCombine(CompletionStage<? extends U>,BiFunction<?>):CompletableFuture<V>
•	thenCompose(Function<?>):CompletableFuture<U>
•	whenComplete(BiConsumer<?>):CompletableFuture<T>
•	^S allOf(CompletableFuture[]<?>):CompletableFuture<Void>
•	^S anyOf(CompletableFuture[]<?>):CompletableFuture<Object>

Implementing the StreamOf FuturesCollector Class

Implementing the StreamOfFuturesCollector Class

- StreamOfFuturesCollector 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>>

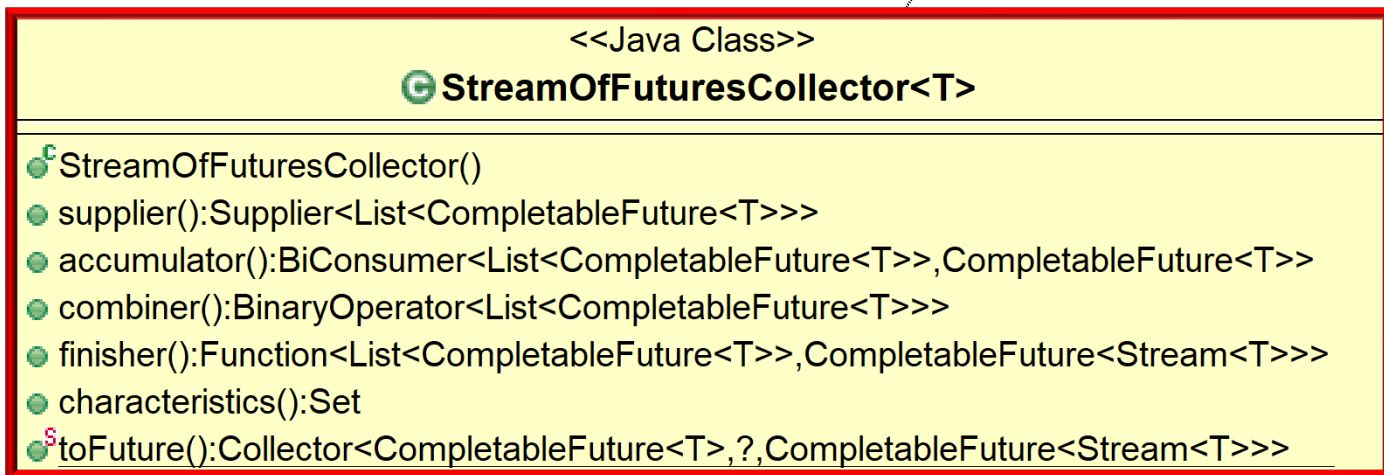
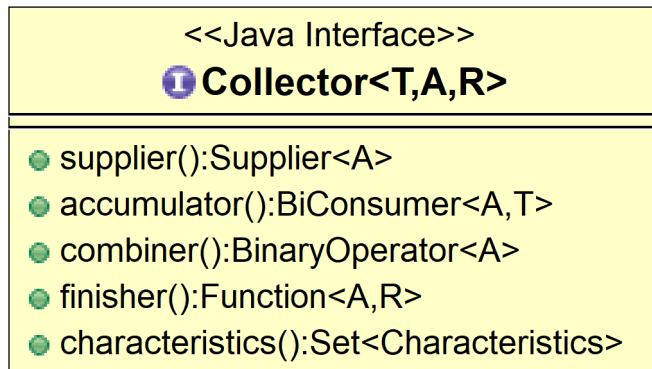
StreamOfFuturesCollector<T>

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

See livelessons/utis/StreamOfFuturesCollector.java

Implementing the StreamOfFuturesCollector Class

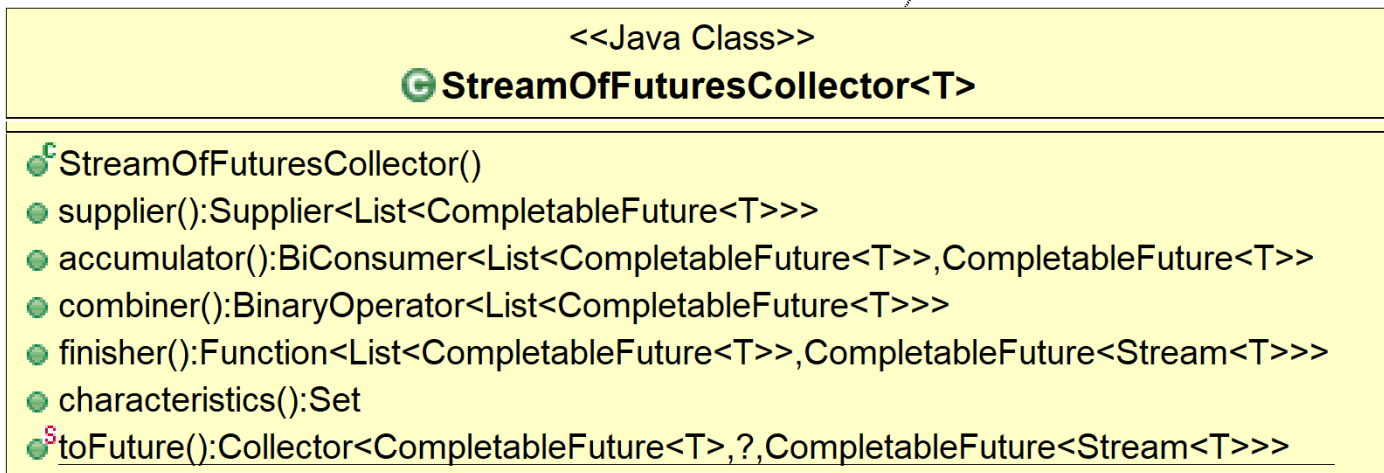
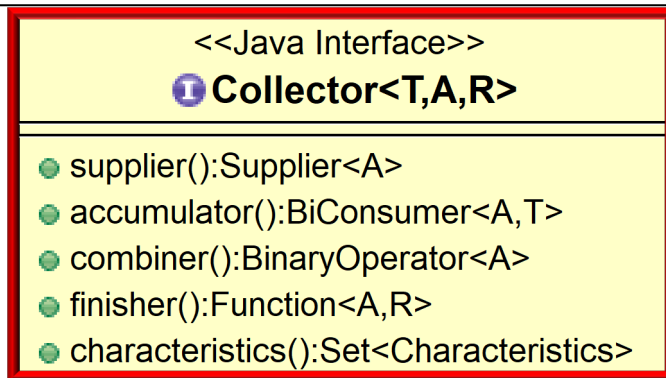
- StreamOfFuturesCollector wraps allOf() to work with the Java Streams framework seamlessly
- Converts a *stream* of CompletableFuture objects to a *single* CompletableFuture that trigger when *all* futures complete



StreamOfFuturesCollector is a non-concurrent collector (supports parallel & sequential streams)

Implementing the StreamOfFuturesCollector Class

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



See docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html

Implementing the StreamOfFuturesCollector Class

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



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

<<Java Class>>	
G StreamOfFuturesCollector<T>	
●	StreamOfFuturesCollector()
●	supplier():Supplier<List<CompletableFuture<T>>>
●	accumulator():BiConsumer<List<CompletableFuture<T>>,CompletableFuture<T>>
●	combiner():BinaryOperator<List<CompletableFuture<T>>>
●	finisher():Function<List<CompletableFuture<T>>,CompletableFuture<Stream<T>>>
●	characteristics():Set
●	toFuture():Collector<CompletableFuture<T>=?,CompletableFuture<Stream<T>>>

A collector accumulates input stream elements into a mutable result container

Implementing the StreamOfFuturesCollector Class

- StreamOfFuturesCollector provides a powerful wrapper for some 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>>

StreamOfFuturesCollector<T>

- StreamOfFuturesCollector()
- supplier():Supplier<List<CompletableFuture<T>>>
- accumulator():BiConsumer<List<CompletableFuture<T>>,CompletableFuture<T>>
- combiner():BinaryOperator<List<CompletableFuture<T>>>
- finisher():Function<List<CompletableFuture<T>>,CompletableFuture<Stream<T>>>
- characteristics():Set
- ^stoFuture():Collector<CompletableFuture<T>=?,CompletableFuture<Stream<T>>>

Implementing the StreamOfFuturesCollector Class

- StreamOfFuturesCollector implements all methods in the Collector interface

```
public class StreamOfFuturesCollector<T>  
    implements Collector<CompletableFuture<T>,  
                        List<CompletableFuture<T>>,  
                        CompletableFuture<Stream<T>>>> {  
    ...  
}
```

Implements a custom collector

Implementing the StreamOfFuturesCollector Class

- StreamOfFuturesCollector implements all methods in the Collector interface

```
public class StreamOfFuturesCollector<T>  
    implements Collector<CompletableFuture<T>,  
                        List<CompletableFuture<T>>,  
                        CompletableFuture<Stream<T>>>> {  
    ...
```

The type of input elements in the stream

Implementing the StreamOfFuturesCollector Class

- StreamOfFuturesCollector implements all methods in the Collector interface

```
public class StreamOfFuturesCollector<T>  
    implements Collector<CompletableFuture<T>,  
                        List<CompletableFuture<T>>,  
                        CompletableFuture<Stream<T>>>> {  
    ...
```

The mutable result container type

Implementing the StreamOfFuturesCollector Class

- StreamOfFuturesCollector implements all methods in the Collector interface

```
public class StreamOfFuturesCollector<T>
    implements Collector<CompletableFuture<T>,
                        List<CompletableFuture<T>>,
                        CompletableFuture<Stream<T>>>> {
    ...
```



The result type of final output of the collector

Implementing the StreamOfFuturesCollector Class

- StreamOfFuturesCollector implements all methods in the Collector interface

```
public class StreamOfFuturesCollector<T>  
    implements Collector<CompletableFuture<T>,  
                        List<CompletableFuture<T>>,  
                        CompletableFuture<Stream<T>>>> {  
  
    ...
```

The Stream<T> parameter differs from the List<T> parameter applied by the previous FuturesCollector

Implementing the StreamOfFuturesCollector Class

- StreamOfFuturesCollector implements all methods in the Collector interface

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

This factory method returns a supplier used by the Java streams collector framework to create a new mutable ArrayList container

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

Implementing the StreamOfFuturesCollector Class

- StreamOfFuturesCollector implements all methods in the Collector interface

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

This mutable result container stores a list of completable futures of type T

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

Implementing the StreamOfFuturesCollector Class

- StreamOfFuturesCollector implements all methods in the Collector interface

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

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

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

This method is only ever called in a single thread (so no locks are needed)

Implementing the StreamOfFuturesCollector Class

- StreamOfFuturesCollector implements all methods in the Collector interface

```
public class StreamOfFuturesCollector<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 ArrayList results into a single ArrayList (only relevant for parallel streams)

This method is only ever called in a single thread (so no locks are needed)

Implementing the StreamOfFuturesCollector Class

- StreamOfFuturesCollector implements all methods in the Collector interface

```
public class StreamOfFuturesCollector<T>
```

```
...
```

```
public Function<List<CompletableFuture<T>>,  
                CompletableFuture<Stream<T>>> finisher() {  
    return futures -> CompletableFuture  
        .allOf(futures.toArray(new CompletableFuture[0]))
```

This factory method returns a function used by the Java streams collector framework to transform the ArrayList mutable result container to the completable future result type

```
        .thenApply(v -> futures.stream()  
                    .map(CompletableFuture::join));  
    }  
    ...
```

Implementing the StreamOfFuturesCollector Class

- StreamOfFuturesCollector implements all methods in the Collector interface

```
public class StreamOfFuturesCollector<T>
```

```
...
```

```
public Function<List<CompletableFuture<T>>,  
                CompletableFuture<Stream<T>>> finisher() {  
    return futures -> CompletableFuture  
        .allOf(futures.toArray(new CompletableFuture[0]))
```

Reference to the mutable result container, which is an ArrayList

```
        .thenApply(v -> futures.stream()  
                    .map(CompletableFuture::join));  
    }  
    ...
```

Implementing the StreamOfFuturesCollector Class

- StreamOfFuturesCollector implements all methods in the Collector interface

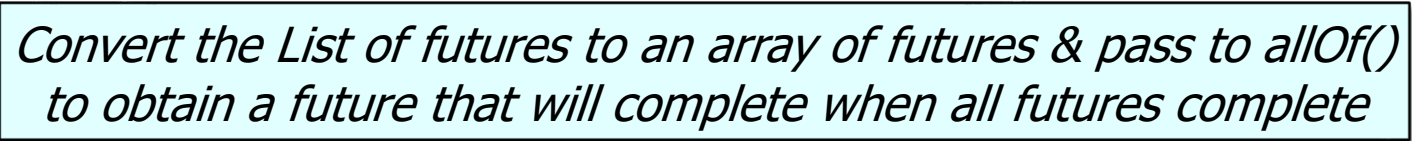
```
public class StreamOfFuturesCollector<T>
```

```
...
```

```
public Function<List<CompletableFuture<T>>,  
                CompletableFuture<Stream<T>>> finisher() {
```

```
    return futures -> CompletableFuture
```

```
        .allOf(futures.toArray(new CompletableFuture[0]))
```



*Convert the List of futures to an array of futures & pass to allOf()
to obtain a future that will complete when all futures complete*

```
        .thenApply(v -> futures.stream()
```

```
                .map(CompletableFuture::join));
```

```
}
```

```
...
```

Implementing the StreamOfFuturesCollector Class

- StreamOfFuturesCollector implements all methods in the Collector interface

```
public class StreamOfFuturesCollector<T>
```

```
...
```

```
public Function<List<CompletableFuture<T>>,  
               CompletableFuture<Stream<T>>> finisher() {  
    return futures -> CompletableFuture  
        .allOf(futures.toArray(new CompletableFuture[0]))
```

When all futures have completed get a single future to a stream of joined elements of type T

```
    .thenApply(v -> futures.stream()  
               .map(CompletableFuture::join));  
}  
...
```

Implementing the StreamOfFuturesCollector Class

- StreamOfFuturesCollector implements all methods in the Collector interface

```
public class StreamOfFuturesCollector<T>
```

```
...
```

```
public Function<List<CompletableFuture<T>>,
               CompletableFuture<Stream<T>>> finisher() {
    return futures -> CompletableFuture
        .allOf(futures.toArray(new CompletableFuture[0]))
```

Convert the ArrayList of futures into a stream of futures

```
        .thenApply(v -> futures.stream()
                    .map(CompletableFuture::join));
    }
    ...
```

Implementing the StreamOfFuturesCollector Class

- StreamOfFuturesCollector implements all methods in the Collector interface

```
public class StreamOfFuturesCollector<T>
```

```
...
```

```
public Function<List<CompletableFuture<T>>,  
                CompletableFuture<Stream<T>>> finisher() {  
    return futures -> CompletableFuture  
        .allOf(futures.toArray(new CompletableFuture[0]))
```

This call to join() will never block!

```
        .thenApply(v -> futures.stream()  
                    .map(CompletableFuture::join));  
    }  
    ...
```

Implementing the StreamOfFuturesCollector Class

- StreamOfFuturesCollector implements all methods in the Collector interface

```
public class StreamOfFuturesCollector<T>
```

```
...
```

```
public Function<List<CompletableFuture<T>>,  
                CompletableFuture<Stream<T>>> finisher() {  
    return futures -> CompletableFuture  
        .allOf(futures.toArray(new CompletableFuture[0]))
```

Return future to stream of elements of T since no terminal operation after map()

```
    .thenApply(v -> futures.stream()  
                .map(CompletableFuture::join));  
}
```

```
...
```

Implementing the StreamOfFuturesCollector Class

- StreamOfFuturesCollector implements all methods in the Collector interface

```
public class StreamOfFuturesCollector<T>
```

```
...
```

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

*Returns a set indicating the characteristics
of the StreamOfFutureCollector class*

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

StreamOfFuturesCollector is thus a *non-concurrent* collector

Implementing the StreamOfFuturesCollector Class

- StreamOfFuturesCollector implements all methods in the Collector interface

```
public class StreamOfFuturesCollector<T>
```

```
...
```

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

This static factory method creates a new StreamOfFuturesCollector

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

Implementing the StreamOfFuturesCollector Class

- toFuture() returns a future to a stream of futures to images being downloaded, filtered, & stored asynchronously

```
void processStream() {  
    List<URL> urls = getInput();  
  
    CompletableFuture<Stream<Image>>  
        resultsFuture = urls  
            .stream()  
            .map(this::checkUrlCachedAsync)  
            .map(this::downloadImageAsync)  
            .flatMap(this::applyFiltersAsync)  
            .collect(toFuture())  
            .thenApply(stream ->  
                log(stream.flatMap  
                    (Optional::stream),  
                    urls.size()))  
            .join();  
}
```

Provides a single means to await completion of a set of futures before continuing with the program

Implementing the StreamOfFuturesCollector Class

- toFuture() returns a future to a stream of futures to images being downloaded, filtered, & stored asynchronously

```
void processStream() {  
    List<URL> urls = getInput();  
  
    CompletableFuture<Stream<Image>>  
        resultsFuture = urls  
            .stream()  
            .map(this::checkUrlCachedAsync)  
            .map(this::downloadImageAsync)  
            .flatMap(this::applyFiltersAsync)  
            .collect(toFuture())  
            .thenApply(stream ->  
                log(stream.flatMap  
                    (Optional::stream),  
                    urls.size()))  
}
```

thenApply() is called only after the future returned from collect() completes

End of the Java Completable Future ImageStreamGang Case Study: StreamOfFuturesCollector