

Java Parallel Streams Internals: Non-Concurrent & Concurrent Collectors (Part 2)

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt



Professor of Computer Science

**Institute for Software
Integrated Systems**

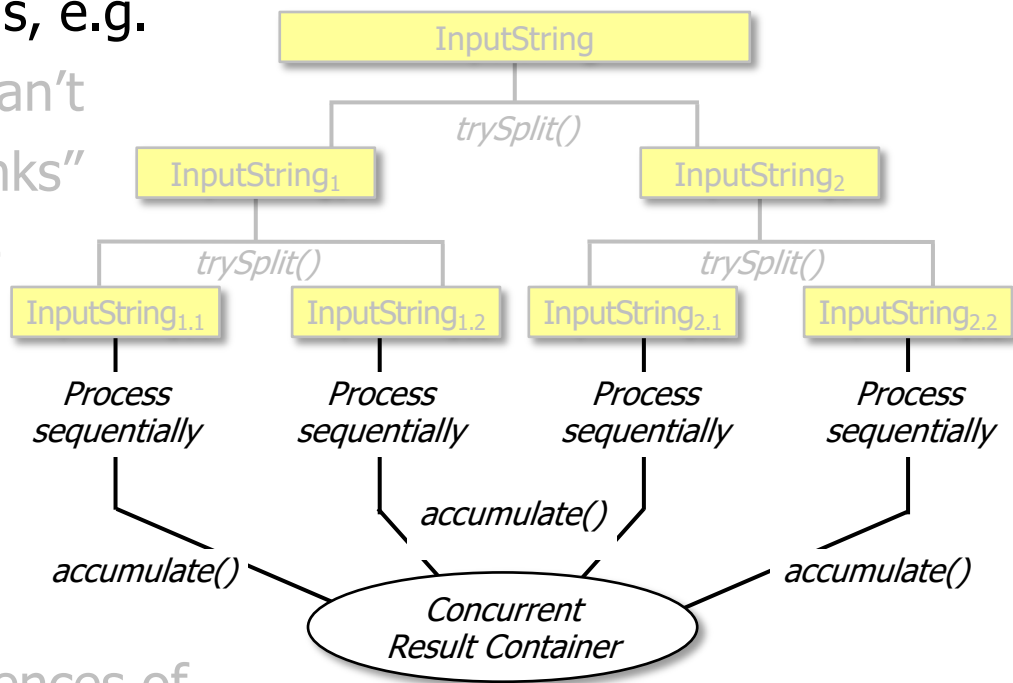
**Vanderbilt University
Nashville, Tennessee, USA**



Learning Objectives in this Part of the Lesson

- Understand parallel stream internals, e.g.

- Know what can change & what can't
- Partition a data source into "chunks"
- Process chunks in parallel via the common fork-join pool
- Configure the Java parallel stream common fork-join pool
- Perform a reduction to combine partial results into a single result
- Recognize key behaviors & differences of non-concurrent & concurrent collectors
- Be aware of non-concurrent & concurrent collector APIs



Non-Concurrent & Concurrent Collector APIs

Non-Concurrent & Concurrent Collector APIs

- The Collector interface defines three generic types



	<code>Collector<T, A, R></code>
<code>(m)</code> <code>accumulator()</code>	<code>BiConsumer<A, T></code>
<code>(m)</code> <code>characteristics()</code>	<code>Set<Characteristics></code>
<code>(m)</code> <code>combiner()</code>	<code>BinaryOperator<A></code>
<code>(m)</code> <code>finisher()</code>	<code>Function<A, R></code>
<code>(m)</code> <code>supplier()</code>	<code>Supplier<A></code>

See www.baeldung.com/java-8-collectors

Non-Concurrent & Concurrent Collector APIs

- The Collector interface defines three generic types
 - **T** – The type of objects available in the stream
 - e.g., Integer, String, Double, SearchResults, etc.

I		Collector< T , A, R>
(m)	accumulator()	BiConsumer<A, T>
(m)	characteristics()	Set<Characteristics>
(m)	combiner()	BinaryOperator<A>
(m)	finisher()	Function<A, R>
(m)	supplier()	Supplier<A>

Non-Concurrent & Concurrent Collector APIs

- The Collector interface defines three generic types
 - **T**
 - **A** – The type of a mutable result container for accumulation
 - e.g., List of T, Set of T, ConcurrentHashMap, KeySetView, etc.

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

Non-Concurrent & Concurrent Collector APIs

- The Collector interface defines three generic types
 - **T**
 - **A** – The type of a mutable result container for accumulation
 - e.g., List of T, Set of T, ConcurrentHashMap, KeySetView, etc.
 - Lists can be implemented by ArrayList, LinkedList, etc.

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

Non-Concurrent & Concurrent Collector APIs












- The Collector interface defines three generic types
 - **T**
 - **A**
 - **R** – The type of a final result
 - e.g., a List of T, KeyViewSet in ConcurrentHashMap, a CompletableFuture to List of T, etc.

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

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface

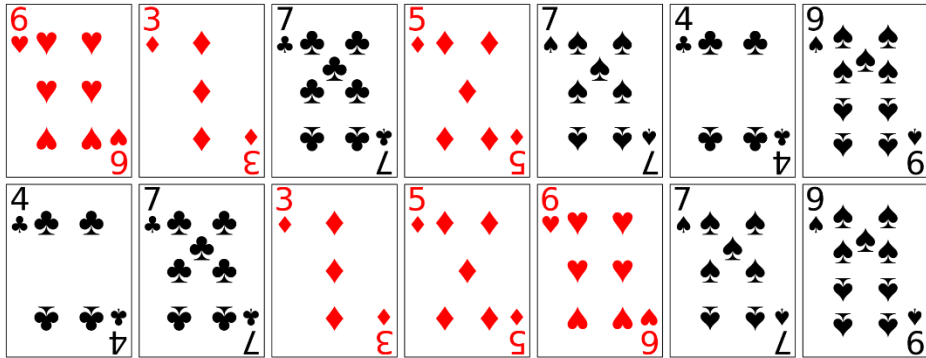


I  Collector<T, A, R>		
 	accumulator()	BiConsumer<A, T>
 	characteristics()	Set<Characteristics>
 	combiner()	BinaryOperator<A>
 	finisher()	Function<A, R>
 	supplier()	Supplier<A>

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
- **characteristics()** – provides a stream with additional information used for internal optimizations, e.g.
 - UNORDERED
 - The collector need not preserve the encounter order

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



A concurrent collector *should* be UNORDERED, but a non-concurrent collector *can* be ORDERED

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
- **characteristics()** – provides a stream with additional information used for internal optimizations, e.g.
 - UNORDERED
 - IDENTITY_FINISH
 - The finisher() is the identity function so it can be a no-op
 - e.g., finisher() just returns null

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



Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
- **characteristics()** – provides a stream with additional information used for internal optimizations, e.g.
 - UNORDERED
 - IDENTITY_FINISH
- CONCURRENT
 - accumulator() is called concurrently on result container

I		Collector<T, A, R>
(m)	accumulator()	BiConsumer<A, T>
(m)	characteristics()	Set<Characteristics>
(m)	combiner()	BinaryOperator<A>
(m)	finisher()	Function<A, R>
(m)	supplier()	Supplier<A>

The mutable result container must be synchronized!!



A concurrent collector *should* be CONCURRENT, but a non-concurrent collector should *not* be!

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
- **characteristics()** – provides a stream with additional information used for internal optimizations, e.g.
 - UNORDERED
 - IDENTITY_FINISH
 - CONCURRENT
 - accumulator() is called concurrently on result container
 - The combiner() method is a no-op

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



Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
- **characteristics()** – provides a stream with additional information used for internal optimizations, e.g.
 - UNORDERED
 - IDENTITY_FINISH
 - CONCURRENT
 - accumulator() is called concurrently on result container
 - The combiner() method is a no-op
 - A non-concurrent collector can be used with either sequential or parallel streams

	I	Collector<T, A, R>
(m)		accumulator() BiConsumer<A, T>
(m)		characteristics() Set<Characteristics>
(m)		combiner() BinaryOperator<A>
(m)		finisher() Function<A, R>
(m)		supplier() Supplier<A>



Internally, the streams framework decides how to ensure correct behavior

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
- **characteristics()** – provides a stream with additional information used for internal optimizations, e.g.

Any/all characteristics can be set using EnumSet.of()












```
Set<Characteristics> characteristics() {  
    return Collections.unmodifiableSet  
        (EnumSet.of(Collector.Characteristics.CONCURRENT,  
                  Collector.Characteristics.UNORDERED));  
}
```

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

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












Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
- **supplier()** – returns a Supplier that acts as a factory to generate an empty result container

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

Non-Concurrent & Concurrent Collector APIs

















- Five methods are defined in the Collector interface
- **supplier()** – returns a Supplier that acts as a factory to generate an empty result container, e.g.
 - `return ArrayList::new`

I  Collector<T, A, R>		
 	<code>accumulator()</code>	<code>BiConsumer<A, T></code>
 	<code>characteristics()</code>	<code>Set<Characteristics></code>
 	<code>combiner()</code>	<code>BinaryOperator<A></code>
 	<code>finisher()</code>	<code>Function<A, R></code>
 	<code>supplier()</code>	<code>Supplier<A></code>

A non-concurrent collector provides a result container for each thread in a parallel stream

Non-Concurrent & Concurrent Collector APIs












- Five methods are defined in the Collector interface
- **supplier()** – returns a Supplier that acts as a factory to generate an empty result container, e.g.
 - `return ArrayList::new`
 - `return ConcurrentHashMap::newKeySet`

		<code>ConcurrentSetCollector<E, S></code>	
		<code>mSetSupplier</code>	<code>Supplier<S></code>
		<code>accumulator()</code>	<code>BiConsumer<Set<E>, E></code>
		<code>characteristics()</code>	<code>Set<Characteristics></code>
		<code>combiner()</code>	<code>BinaryOperator<Set<E>></code>
		<code>finisher()</code>	<code>Function<Set<E>, S></code>
		<code>supplier()</code>	<code>Supplier<Set<E>></code>
		<code>toSet(Supplier<S>)</code>	<code>Collector<T, ?, S></code>

A concurrent collector has one result container shared by all threads in a parallel stream

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
 - `supplier()`
 - **`accumulator()`** – returns a Bi-Consumer that adds a new element to an existing result container

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

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
 - `supplier()`
 - **`accumulator()`** – returns a Bi-Consumer that adds a new element to an existing result container, e.g.
 - `return List::add`

Collector<T, A, R>		
(m)	<code>accumulator()</code>	BiConsumer<A, T>
(m)	<code>characteristics()</code>	Set<Characteristics>
(m)	<code>combiner()</code>	BinaryOperator<A>
(m)	<code>finisher()</code>	Function<A, R>
(m)	<code>supplier()</code>	Supplier<A>









A non-concurrent collector needs no synchronization



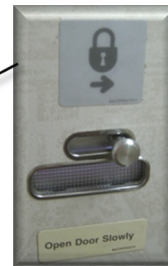
See docs.oracle.com/javase/8/docs/api/java/util/List.html#add

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
 - `supplier()`
 - **`accumulator()`** – returns a Bi-Consumer that adds a new element to an existing result container, e.g.
 - `return List::add`
 - `return ConcurrentHashMap.KeySetView::add`












	<code>ConcurrentSetCollector<E, S></code>	
	<code>mSetSupplier</code>	<code>Supplier<S></code>
	<code>accumulator()</code>	<code>BiConsumer<Set<E>, E></code>
	<code>characteristics()</code>	<code>Set<Characteristics></code>
	<code>combiner()</code>	<code>BinaryOperator<Set<E>></code>
	<code>finisher()</code>	<code>Function<Set<E>, S></code>
	<code>supplier()</code>	<code>Supplier<Set<E>></code>
	<code>toSet(Supplier<S>)</code>	<code>Collector<T, ?, S></code>

A concurrent collector's result container must be synchronized



Non-Concurrent & Concurrent Collector APIs












- Five methods are defined in the Collector interface
 - `supplier()`
 - `accumulator()`
 - **`combiner()`** – returns a Binary Operator that merges two result containers together

I  Collector<T, A, R>		
 	<code>accumulator()</code>	BiConsumer<A, T>
 	<code>characteristics()</code>	Set<Characteristics>
 	<code>combiner()</code>	BinaryOperator<A>
 	<code>finisher()</code>	Function<A, R>
 	<code>supplier()</code>	Supplier<A>

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
 - `supplier()`
 - `accumulator()`
 - **`combiner()`** – returns a Binary Operator that merges two result containers together, e.g.

- ```
return (one, another) -> {
 one.addAll(another);
 return one;
}
```

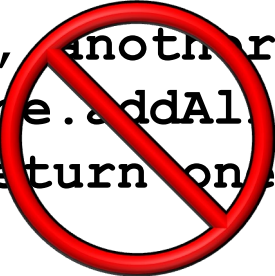
| I  Collector<T, A, R>                                                               |                                                     |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------|
|   | <code>accumulator()</code> BiConsumer<A, T>         |
|   | <code>characteristics()</code> Set<Characteristics> |
|   | <b><code>combiner()</code></b> BinaryOperator<A>    |
|   | <code>finisher()</code> Function<A, R>              |
|   | <code>supplier()</code> Supplier<A>                 |

A `combiner()` is only used for a non-concurrent collector

# Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - `supplier()`
  - `accumulator()`
  - **`combiner()`** – returns a Binary Operator that merges two result containers together, e.g.

```
• return (one, another) -> {
 one.addAll(another);
 return one;
}
```














| Collector<T, A, R> |                                |                      |
|--------------------|--------------------------------|----------------------|
| (m)                | <code>accumulator()</code>     | BiConsumer<A, T>     |
| (m)                | <code>characteristics()</code> | Set<Characteristics> |
| (m)                | <b><code>combiner()</code></b> | BinaryOperator<A>    |
| (m)                | <code>finisher()</code>        | Function<A, R>       |
| (m)                | <code>supplier()</code>        | Supplier<A>          |

The `combiner()` method is not called when CONCURRENT is set














# Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - `supplier()`
  - `accumulator()`
  - `combiner()`
  - **`finisher()`** – returns a Function that converts the result container to final result type

| I  Collector<T, A, R>                                                               |                                |                      |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------|----------------------|
|   | <code>accumulator()</code>     | BiConsumer<A, T>     |
|   | <code>characteristics()</code> | Set<Characteristics> |
|   | <code>combiner()</code>        | BinaryOperator<A>    |
|   | <b><code>finisher()</code></b> | Function<A, R>       |
|   | <code>supplier()</code>        | Supplier<A>          |

# Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - `supplier()`
  - `accumulator()`
  - `combiner()`
  - **`finisher()`** – returns a Function that converts the result container to final result type, e.g.
    - `Function.identity()`

| I  Collector<T, A, R>                                                               |                                                     |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------|
|   | <code>accumulator()</code> BiConsumer<A, T>         |
|   | <code>characteristics()</code> Set<Characteristics> |
|   | <code>combiner()</code> BinaryOperator<A>           |
|   | <b><code>finisher()</code></b> Function<A, R>       |
|   | <code>supplier()</code> Supplier<A>                 |

# Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - **supplier()**
  - **accumulator()**
  - **combiner()**
  - **finisher()** – returns a Function that converts the result container to final result type, e.g.
    - `Function.identity()`
    - **return null**

*Should be a no-op if IDENTITY\_FINISH characteristic is set*















| Collector<T, A, R> |                   |                      |
|--------------------|-------------------|----------------------|
| (m)                | accumulator()     | BiConsumer<A, T>     |
| (m)                | characteristics() | Set<Characteristics> |
| (m)                | combiner()        | BinaryOperator<A>    |
| (m)                | <b>finisher()</b> | Function<A, R>       |
| (m)                | supplier()        | Supplier<A>          |



# Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - `supplier()`
  - `accumulator()`
  - `combiner()`
  - **`finisher()`** – returns a Function that converts the result container to final result type, e.g.
    - `Function.identity()`
    - `return null`

*finisher() can also be more interesting!*

















|                                                                                     |                                                                                     |                                                 |                                                 |
|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|-------------------------------------------------|-------------------------------------------------|
|  |  | <code>ConcurrentSetCollector&lt;E, S&gt;</code> |                                                 |
|   |  | <code>mSetSupplier</code>                       | <code>Supplier&lt;S&gt;</code>                  |
|   |  | <code>accumulator()</code>                      | <code>BiConsumer&lt;Set&lt;E&gt;, E&gt;</code>  |
|   |  | <code>characteristics()</code>                  | <code>Set&lt;Characteristics&gt;</code>         |
|   |  | <code>combiner()</code>                         | <code>BinaryOperator&lt;Set&lt;E&gt;&gt;</code> |
|   |  | <b><code>finisher()</code></b>                  | <code>Function&lt;Set&lt;E&gt;, S&gt;</code>    |
|   |  | <code>supplier()</code>                         | <code>Supplier&lt;Set&lt;E&gt;&gt;</code>       |

```
return set -> {
 S ns = mSetSupplier.get();
 if (ns instanceof ConcurrentHashMap
 .KeySetView)
 return (S) set;
 else { ns.addAll(set); return ns; }
};
```

See [Java8/ex36/src/main/java/utils/ConcurrentSetCollector.java](#)

# Non-Concurrent & Concurrent Collector APIs

- We also define a factory method that creates a new instance of the ConcurrentSetCollector class

|                                                                                     |                                                                                     |                                      |                        |
|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|--------------------------------------|------------------------|
|  |  | <b>ConcurrentSetCollector</b> <E, S> |                        |
|   |  | <b>mSetSupplier</b>                  | Supplier<S>            |
|   |  | <b>accumulator()</b>                 | BiConsumer<Set<E>, E>  |
|   |  | <b>characteristics()</b>             | Set<Characteristics>   |
|   |  | <b>combiner()</b>                    | BinaryOperator<Set<E>> |
|   |  | <b>finisher()</b>                    | Function<Set<E>, S>    |
|   |  | <b>supplier()</b>                    | Supplier<Set<E>>       |
|   |  | <b>toSet(Supplier&lt;S&gt;)</b>      | Collector<T, ?, S>     |

See [Java8/ex36/src/main/java/Utils/ConcurrentSetCollector.java](https://github.com/openjdk/jdk8/blob/master/src/main/java/Utils/ConcurrentSetCollector.java)

---

# End of Java Parallel Streams Internals: Non-Concurrent & Concurrent Collectors (Part 2)