Understand the Java Streams Non-Concurrent Collector API

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 structure & functionality of non-concurrent collectors for sequential streams
• Know the API for non-concurrent collectors

Java Interface:

```java
Collector<T,A,R>

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

The same API is also used for concurrent collectors!
The Non-Concurrent Collector API
The Non-Concurrent Collector API

- The Collector interface defines three generic types

```java
Collector<T,A,R>
```

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

See [www.baeldung.com/java-8-collectors](http://www.baeldung.com/java-8-collectors)
The Non-Concurrent Collector API

- The Collector interface defines three generic types
  - **T** - The type of elements available in the stream
    - e.g., Long, String, SearchResults, etc.

```java
<<Java Interface>>

Collector<T, A, R>

- supplier(): Supplier<A>
- accumulator(): BiConsumer<A, T>
- combiner(): BinaryOperator<A>
- finisher(): Function<A, R>
- characteristics(): Set<Characteristics>
```
The Non-.Concurrent Collector API

• The Collector interface defines three generic types
  - T
  - A – The type of mutable accumulator object to use for collecting elements
    - e.g., List of T (implemented via ArrayList, LinkedList, etc.)

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

- supplier(): Supplier<A>
- accumulator(): BiConsumer<A, T>
- combiner(): BinaryOperator<A>
- finisher(): Function<A, R>
- characteristics(): Set<Characteristics>
The Non-Concurrent Collector API

- The Collector interface defines three generic types
  - T
  - A
  - R – The type of the final result
  - e.g., List of T

The type of R & the type of A may or may not be different!
The Non-Concurrent Collector API

- Five factory methods are defined in the Collector interface

<<Java Interface>>

```java
interface Collector<T, A, R> {
    Supplier<A> supplier();
    BiConsumer<A, T> accumulator();
    BinaryOperator<A> combiner();
    Function<A, R> finisher();
    Set<Characteristics> characteristics();
}
```

Again, this discussion assumes we’re implementing a *non-concurrent* collector
The Non-Concurrent Collector API

- Five factory methods are defined in the Collector interface
  - `characteristics()` – provides a stream with additional information used for internal optimizations

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

- supplier(): Supplier<A>
- accumulator(): BiConsumer<A,T>
- combiner(): BinaryOperator<A>
- finisher(): Function<A,R>
- characteristics(): Set<Characteristics>
```
The Non-Concurrent Collector API

• Five factory 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
The Non-.Concurrent Collector API

- Five factory 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

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

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

A collector may preserve encounter order if it incurs no additional overhead
The Non-Concurrent Collector API

- Five factory 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
Five factory methods are defined in the Collector interface

- **characteristics()** – provides a stream with additional information used for internal optimizations, e.g.
  - UNORDERED
  - IDENTITY_FINISH
- CONCURRENT
  - The accumulator method is called concurrently on the result container

The mutable result container must be synchronized!!
The Non-.Concurrent Collector API

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

We’re focusing on a non-concurrent collector, which doesn’t enable CONCURRENT.
The Non-Concurrent Collector API

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

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

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

See [docs.oracle.com/javase/8/docs/api/java/util/EnumSet.html](http://docs.oracle.com/javase/8/docs/api/java/util/EnumSet.html)
The Non-Concurrent Collector API

- Five factory methods are defined in the Collector interface
  - characteristics()
  - supplier() – returns a supplier that acts as a factory to generate an empty result container
The Non- Concurrent Collector API

- Five factory methods are defined in the Collector interface
  - `characteristics()`
  - `supplier()` – returns a supplier that acts as a factory to generate an empty result container, e.g.
    ```java
    Supplier<List> supplier() {
        return ArrayList::new;
    }
    ```

See [docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#ArrayList](http://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#ArrayList)
Five factory methods are defined in the Collector interface:

- `characteristics()`
- `supplier()`
- `accumulator()` – returns a bi-consumer that adds a new element to an existing result container, e.g.

```java
BiConsumer<List, Integer> accumulator() {
    return List::add;
}
```

A non-concurrent collector needs no synchronization.

See [docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#add](docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#add)
Five factory methods are defined in the Collector interface:

- `characteristics()`
- `supplier()`
- `accumulator()`
- `combiner()` – returns a binary operator that merges two result containers together, e.g.

```java
BinaryOperator<List> combiner() {
    return (one, another) -> {
        one.addAll(another);
        return one;
    };
}
```

This combiner() will not be called for a sequential stream.
The Non-Concurrent Collector API

- Five factory methods are defined in the Collector interface
  - characteristics()
  - supplier()
  - accumulator()
  - combiner()
  - finisher() – returns a function that converts the result container to final result type, e.g.
    - return Function.identity()
The Non-Concurrent Collector API

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

Should be a no-op if `IDENTITY_FINISH` characteristic is set
The Non-Concurrent Collector API

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

Stream
  .generate(() ->
    makeBigFraction
    (new Random(), false))
  .limit(sMAX_FRACTIONS)
  .map(reduceAndMultiplyFraction)
  .collect(FuturesCollector
    .toFuture())
  .thenAccept(
    this::sortAndPrintList);

See Java8/ex19/src/main/java/utils/FuturesCollector.java

finisher() can also be much more interesting!
End of Understand the Java Streams Non-Concurrent Collector API