Java Streams: Implementing Non-.Concurrent Collectors

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 how to implement a non-concurrent collector for sequential streams

<<Java Interface>>

Collector<T,A,R>

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

- The Collector interface defines three generic types

```java
<<Java Interface>>

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)
Implementing a Non-Concurrent Collector

- 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>
Implementing a Non-Concurrent Collector

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

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

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

<<Java Interface>>

Collector<T,A,R>

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

- Five factory methods are defined in the Collector interface

```
<<Java Interface>>

Collector<T,A,R>

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

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

- Five factory methods are defined in the Collector interface

  - `characteristics()` – provides a stream with additional information used for internal optimizations

```
<<Java Interface>>

Collector<T,A,R>

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

- 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
Implementing a Non-Concurrent Collector

- 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

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

- 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

<<Java Interface>>

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

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

- 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!!
Implementing a Non-Concurrent Collector

- 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.
Implementing a Non-Concurrent Collector

- 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](docs.oracle.com/javase/8/docs/api/java/util/EnumSet.html)
Implementing a Non-Concurrent Collector

- 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

```java
<<Java Interface>>

Collector<T,A,R>

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

- 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;
    }
    ```
Implementing a Non-Concurrent Collector

• 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
Implementing a Non-Concurrent Collector

- 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..
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()`
Implementing a Non-Concurrent Collector

- 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
Implementing a Non-Concurrent Collector

- 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);
```

finisher() can also be much more interesting!

See Java8/ex19/src/main/java/utils/FuturesCollector.java
Applying Non-Concurrent Collectors

- More information on implementing custom collectors is available online

See www.youtube.com/watch?v=H7VbRz9aj7c
End of Java Streams: Implementing Non-Concurrent Collectors