Java Streams: Implementing Custom 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 the API for non-concurrent collectors
- Recognize how to apply pre-defined non-concurrent collectors
- Be able to implement custom non-concurrent collectors

```java
interface Collector<T, A, R>{
...
    static<T, R>
    Collector<T, R, R> of
    (Supplier<R> supplier,
     BiConsumer<R, T> accumulator,
     BinaryOperator<R> combiner,
     Function<A,R> finisher,
     Characteristics... chars) {
        ...
    }
}
```

See docs.oracle.com/javase/8/docs/api/java/util/stream/Colector.html#of
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
• Recognize how to apply pre-defined non-concurrent collectors
• Be able to implement custom non-concurrent collectors

• e.g., we analyze several implementations of non-concurrent collectors from the SimpleSearchStream program

See [github.com/douglascraigschmidt/LiveLessons/tree/master/SimpleSearchStream](https://github.com/douglascraigschmidt/LiveLessons/tree/master/SimpleSearchStream)
Implementing Custom Non-Concurrent Collectors (Part 1)
Implementing Custom Non-.Concurrent Collectors (Part 1)

- `Collector.of()` can implement custom collectors that have pithy lambdas

```java
public String toString() {
    ...
    mList.stream()
        .collect(Collectors.of(() -> new StringJoiner("|")
            (j, r) -> j.add(r.toString()),
            StringJoiner::merge,
            StringJoiner::toString)); ...
```

Starting SimpleSearchStream
Word "Re" matched at index [131|141|151|202|212|222|1279|1025|1219|1259|1278|1300|1351|1370|1835|1875|1899|1939|2266|2295]
Word "Ti" matched at index [237|994|1272|1294|1364|1850|1860|1912|1915|1952|1955|2299]
Word "La" matched at index [234|417|658|886|991|1207|1247|1269|1291|1339|1361|1742|1847|1863|1909|1949|2161|2254|2276|2283]...
```

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html#of](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html#of)
Implementing Custom Non-Concurrent Collectors (Part 1)

- The `SearchResults.toString()` method uses `Collector.of()` to format results

```java
public String toString() {
    ...
    mList.stream()
        .collect(Collectors.<Collector.of() -> new StringJoiner("|"),
                         (j, r) -> j.add(r.toString()),
                         StringJoiner::merge,
                         StringJoiner::toString)); ...
```

See `SimpleSearchStream/src/main/java/search/SearchResults.java`
Implementing Custom Non-Concurrent Collectors (Part 1)

• The SearchResults.toString() method uses Collector.of() to format results

```java
public String toString() {
    ...  
    mList.stream()
        .collect(Collectors.of(() -> (Collector<>(j, r) -> j.add(r.toString())))); ...
```

Factory method creates a new collector via the five-param of() method version

Starting SimpleSearchStream
Word "Re" matched at index [131|141|151|202|212|222]
979|1025|1219|1259|
1278|1300|1351|1370|1835|
1875|1899|1939|2266|2295|

Word "Ti" matched at index [237|994|1272|1294|1364|1850|
1860|1912|1915|1952|1955|
2299]

Word "La" matched at index [234|417|658|886|991|1207|
1247|1269|1291|1339|1361|
1742|1847|1863|1909|1949|
2161|2254|2276|2283]...

Ending SimpleSearchStream

StringJoiner::merge,
StringJoiner::toString)); ...

See docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html#of
Implementing Custom Non-Concurrent Collectors (Part 1)

- The SearchResults.toString() method uses Collector.of() to format results

```java
public String toString() {
    ... 
    mList.stream()
        .collect(Collectors.<Collector.of(() -> new StringJoiner("|")
            , (j, r) -> j.add(r.toString()),
            StringJoiner::merge,
            StringJoiner::toString)); ...
```

See docs.oracle.com/javase/8/docs/api/java/util/StringJoiner.html
Implementing Custom Non-Concurrent Collectors (Part 1)

- The SearchResults.toString() method uses Collector.of() to format results

```java
public String toString() {
    ... 
    mList.stream()
        .collect(Collectors.of(() -> new StringJoiner(" | "), 
                        (j, r) -> j.add(r.toString()),
                        StringJoiner::merge,
                        StringJoiner::toString)); ... 
```

This lambda biconsumer adds a new string to the joiner

(j, r) is equivalent to (StringJoiner j, SearchResults.Result r)
Implementing Custom Non-Concurrent Collectors (Part 1)

- The SearchResults.toString() method uses Collector.of() to format results

```java
public String toString() {
    ... 
    mList.stream()
        .collect(Collector.of(() -> new StringJoiner("|"),
                          (j, r) -> j.add(r.toString()),
                          StringJoiner::merge,
                          StringJoiner::toString)); ...
```

This combiner is only used for parallel streams.
Implementing Custom Non-Concurrent Collectors (Part 1)

- The SearchResults.toString() method uses Collector.of() to format results

```java
public String toString() {
    ...
    mList.stream()
        .collect(Collectors.<Collector.of() -> new StringJoiner("|"),
                         (j, r) -> j.add(r.toString()),
                         StringJoiner::merge,
                         StringJoiner::toString)); ...
```

This finisher converts a string joiner to a string
Implementing Custom Non-Concurrent Collectors (Part 1)

- The SearchResults.toString() method uses Collector.of() to format results

```java
public String toString() {
    ...
    mList.stream()
        .collect(Collectors.of(() -> new StringJoiner("|"),
            (j, r) -> j.add(r.toString()),
            StringJoiner::merge,
            StringJoiner::toString)); ...
```

Only four params are passed to of() since Characteristics... is an optional parameter!
Implementing Custom Non-Concurrent Collectors (Part 2)
The WordSearcher.toDownstreamCollector also uses Collector.of()

```java
static Collector<SearchResults, List<SearchResults.Result>,
    List<SearchResults.Result>>
    toDownstreamCollector() {
    return Collector.of
        (ArrayList::new,
         (rl, sr) -> rl.addAll(sr.getResultList()),
         (left, right) -> { 
             left.addAll(right);
             return left;
         });
}
```

See earlier lesson on "Java Streams: Visualizing WordSearcher.printSuffixSlice()"
Implementing Custom Non-Concurrent Collectors (Part 2)

- The WordSearcher.toDownstreamCollector also uses Collector.of()

```java
static Collector<SearchResults, List<SearchResults.Result>, List<SearchResults.Result>>
toDownstreamCollector() {
    return Collector.of
        (ArrayList::new,
         (rl, sr) -> rl.addAll(sr.getResultList()),
         (left, right) -> {
            left.addAll(right);
            return left;
        });
}
```

This factory method creates a downstream collector that merges results lists together.

See SimpleSearchStream/src/main/java/search/WordSearcher.java
Implementing Custom Non-Concurrent Collectors (Part 2)

• The WordSearcher.toDownstreamCollector also uses Collector.of()

```java
static Collector<SearchResults, List<SearchResults.Result>,
        List<SearchResults.Result>>
toDownstreamCollector() {
    return Collector.of
        (ArrayList::new,
         (rl, sr) -> rl.addAll(sr.getResultList()),
         (left, right) -> {
             left.addAll(right);
             return left;
         });
}
```

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html#of](http://docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html#of)
Implementing Custom Non-Concurrent Collectors (Part 2)

- The WordSearcher.toDownstreamCollector also uses Collector.of()

```java
static Collector<SearchResults, List<SearchResults.Result>, List<SearchResults.Result>>
toDownstreamCollector() {
    return Collector.of
        (ArrayList::new,
         (rl, sr) -> rl.addAll(sr.getResultList()),
         (left, right) -> {
             left.addAll(right);
             return left;
         });
}
```

Make a mutable results list container from an array list
Implementing Custom Non-Concurrent Collectors (Part 2)

- The `WordSearcher.toDownstreamCollector` also uses `Collector.of()`

```java
static Collector<SearchResults, List<SearchResults.Result>,
    List<SearchResults.Result>>
toDownstreamCollector() {
    return Collector.of
        (ArrayList::new,
         (rl, sr) -> rl.addAll(sr.getResultList()),
         (left, right) -> {
             left.addAll(right);
             return left;
         });
}
```

Accumulate all result objects from a `SearchResults` object into the results list.
Implementing Custom Non-Concurrent Collectors (Part 2)

- The WordSearcher.toDownstreamCollector also uses Collector.of()

```java
static Collector<SearchResults, List<SearchResults.Result>,
    List<SearchResults.Result>>
    toDownstreamCollector() {

    return Collector.of
    (ArrayList::new,
     (rl, sr) -> rl.addAll(sr.getResultList()),
     (left, right) -> {
         left.addAll(right);
         return left;
     }));
```

Merge two results lists into a single results list

This combiner is only used for parallel streams
Implementing Custom Non-Concurrent Collectors (Part 2)

• The WordSearcher.toDownstreamCollector also uses Collector.of()

```java
static Collector<SearchResults, List<SearchResults.Result>,
    List<SearchResults.Result>>
    toDownstreamCollector() {
    return Collector.of
    (ArrayList::new,
    (rl, sr) -> rl.addAll(sr.getResultList()),
    (left, right) -> {
    left.addAll(right);
    return left;
    });
}
```

Only three params are passed to of() since Characteristics... is an optional parameter!
Implementing Custom Non-Concurrent Collectors (Part 2)

- Complex custom collectors should implement the Collector interface instead of using Collector.of()

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

```
public class FuturesCollector<T> {
    public FuturesCollector()
    public supplier(): Supplier<List<CompletableFuture<T>>>
    public accumulator(): BiConsumer<List<CompletableFuture<T>>, CompletableFuture<T>>
    public combiner(): BinaryOperator<CompletableFuture<T>>
    public finisher(): Function<CompletableFuture<T>, CompletableFuture<List<T>>>
    public characteristics(): Set
    public toFuture(): Collector<CompletableFuture<T>, ?, CompletableFuture<List<T>>>
```
Implementing Custom Non-Concurrent Collectors (Part 2)

- More information on implementing custom collectors is available online

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