How to Implement 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

```
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/Collector.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

Implementing Custom Non-Concurrent Collectors (Part 1)

Implementing Custom Non-Concurrent Collectors (Part 1)

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

```
public String toString() {
   ...
   mList.stream()
        .collect(Collector.of(() -> new StringJoiner("|"),

                              (j, r) -> j.add(r.toString()),

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

SearchResults's custom collector formats itself

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

```
public String toString() {
   ...
   mList.stream()
        .collect(Collector.of(() -> new StringJoiner("|"),

                 (j, r) -> j.add(r.toString()),

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

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

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

```
public String toString() {
   ...
   mList.stream()
        .collect(Collector.of(() -> new StringJoiner("|"),

                 (j, r) -> j.add(r.toString()),

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

This lambda supplier creates the mutable result container

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

```
public String toString() {
   ...
   mList.stream()
        .collect(Collector.of(() -> new StringJoiner("|"),
```


```
(j, r) -> j.add(r.toString()),
```

This lambda BiConsumer adds a new String to the StringJoiner

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

`(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

```
public String toString() {
   ...
   mList.stream()
        .collect(Collector.of(() -> new StringJoiner("|"),

                 (j, r) -> j.add(r.toString()),

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

Combine two StringJoiners

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

```
public String toString() {
   ...
   mList.stream()
        .collect(Collector.of(() -> new StringJoiner("|"),

                 (j, r) -> j.add(r.toString()),

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

This finisher converts a StringJoiner to a String

Implementing Custom Non-Concurrent Collectors (Part 1)

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

```
public String toString() {
   ...
   mList.stream()
        .collect(Collector.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)

Implementing Custom Non-Concurrent Collectors (Part 2)

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

```
static Collector<SearchResults, List<SearchResults.Result>,
                 List<SearchResults.Result>>
                                          toDownstreamCollector() {
  return Collector.of
           (ArrayList::new,

           (rl, sr) -> rl.addAll(sr.getResultList()),

            StreamUtils::concat);
}
```

See earlier lesson on "*Java Streams: Visualizing WordSearcher.printResults()*"

Implementing Custom Non-Concurrent Collectors (Part 2)

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

```
static Collector<SearchResults, List<SearchResults.Result>,
                List<SearchResults.Result>>
                                          toDownstreamCollector() {
  return Collector.of
           (ArrayList::new,

           (rl, sr) -> rl.addAll(sr.getResultList()),

            StreamUtils::concat);
}
```

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()

```
static Collector<SearchResults, List<SearchResults.Result>,
                List<SearchResults.Result>>
                                        toDownstreamCollector() {
  return Collector.of
           (ArrayList::new,

           (rl, sr) -> rl.addAll(sr.getResultList()),

            StreamUtils::concat);
}
```

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

See 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()

```
static Collector<SearchResults, List<SearchResults.Result>,
                List<SearchResults.Result>>
                                        toDownstreamCollector() {
  return Collector.of
           (ArrayList::new,

           (rl, sr) -> rl.addAll(sr.getResultList()),

            StreamUtils::concat);
}
```

Make a mutable results list container from an array list

Implementing Custom Non-Concurrent Collectors (Part 2)

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

```
static Collector<SearchResults, List<SearchResults.Result>,
                List<SearchResults.Result>>
                                        toDownstreamCollector() {
  return Collector.of
           (ArrayList::new,

           (rl, sr) -> rl.addAll(sr.getResultList()),

            StreamUtils::concat);
}
```

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()

```
static Collector<SearchResults, List<SearchResults.Result>,
                 List<SearchResults.Result>>
                                        toDownstreamCollector() {
  return Collector.of
            (ArrayList::new,

            (rl, sr) -> rl.addAll(sr.getResultList()),

             StreamUtils::concat);
}
```

Merge two results lists into one results list (only used for parallel streams)

```
static <T> List<T> concat(List<T> l1, List<T> l2)
{ l1.addAll(l2); return l1; }
```

See SimpleSearchStream/src/main/java/utils/StreamUtils.java

Implementing Custom Non-Concurrent Collectors (Part 2)

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

```
static Collector<SearchResults, List<SearchResults.Result>,
                List<SearchResults.Result>>
                                        toDownstreamCollector() {
  return Collector.of
           (ArrayList::new,

           (rl, sr) -> rl.addAll(sr.getResultList()),

           StreamUtils::concat);
}
```

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()

<<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>>

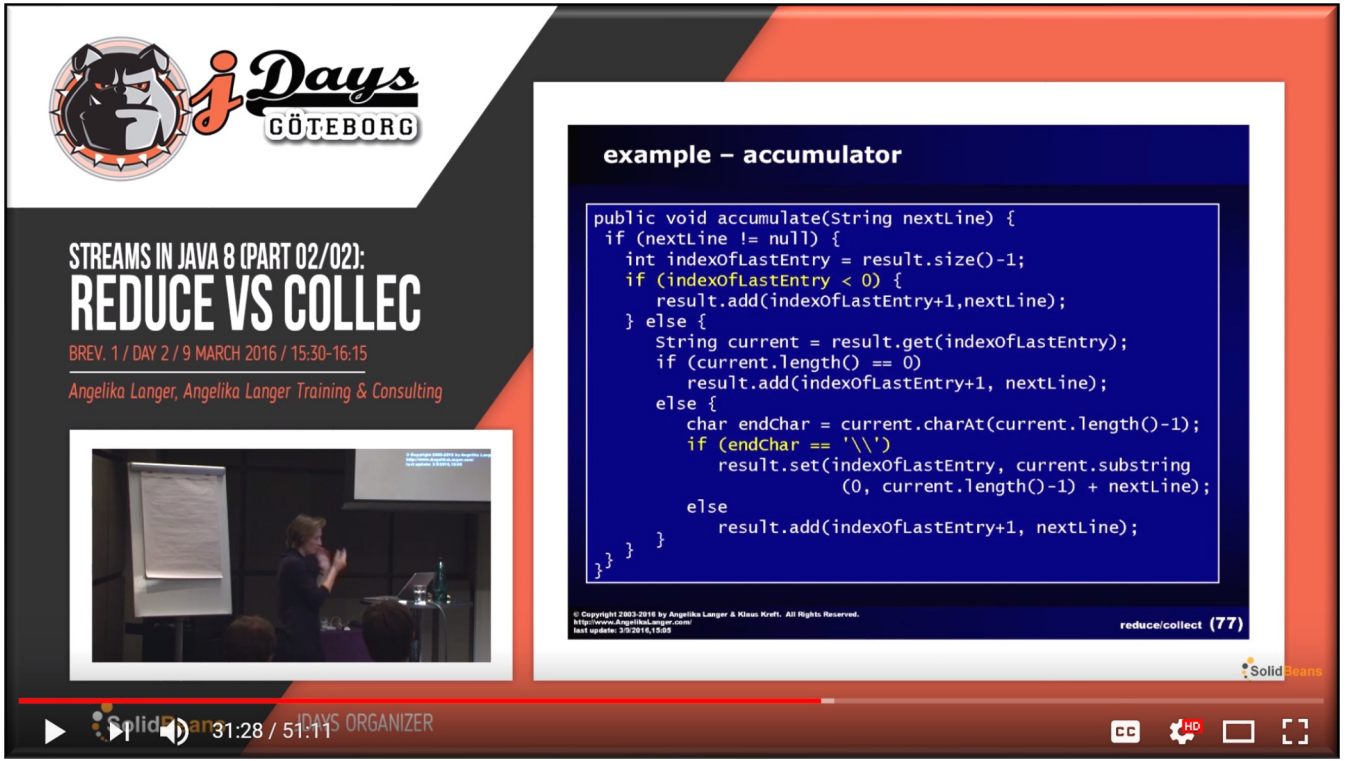
FuturesCollector<T>

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

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

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 How to Implement Custom Non-Concurrent Collectors