

# How to Implement Custom Non-Concurrent Collectors

**Douglas C. Schmidt**

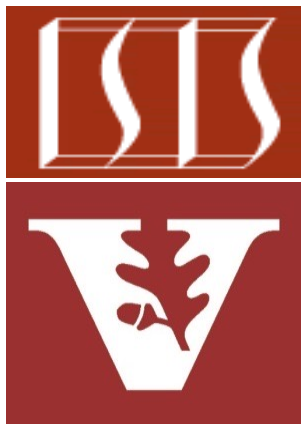
**[d.schmidt@vanderbilt.edu](mailto:d.schmidt@vanderbilt.edu)**

**[www.dre.vanderbilt.edu/~schmidt](http://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) {  
                ...  
            } ...  
}
```

# 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



```
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
```

---

# 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()),
```



```
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
```

*SearchResults's custom  
collector formats itself*

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

See [SimpleSearchStream/src/main/java/search/SearchResults.java](https://github.com/pschuch/simply/blob/master/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*

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

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



# Implementing Custom Non-Concurrent Collectors (Part 1)

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

```
public String toString() {
```

```
    ...
```

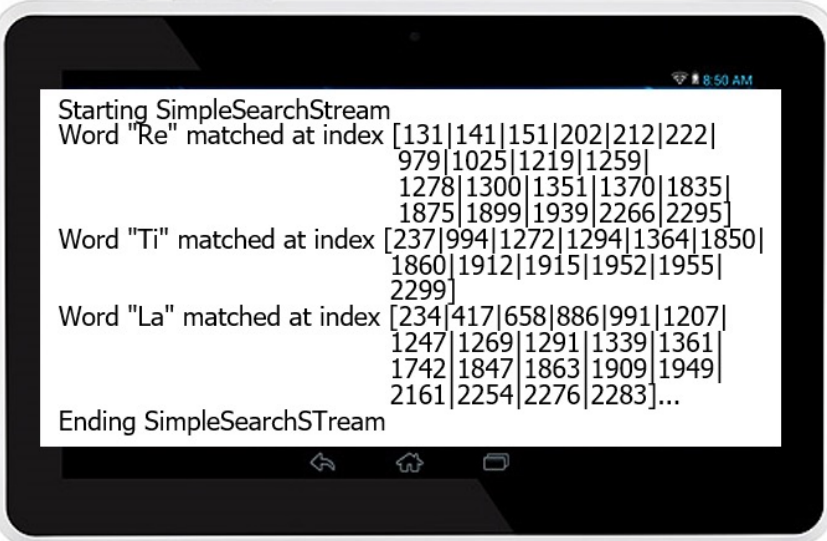
```
    mList.stream()
```

```
        .collect(Collector.of(() -> new StringJoiner("|"),
```

*This lambda supplier creates the mutable result container*

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

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



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

See [docs.oracle.com/javase/8/docs/api/java/util/StringJoiner.html](https://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

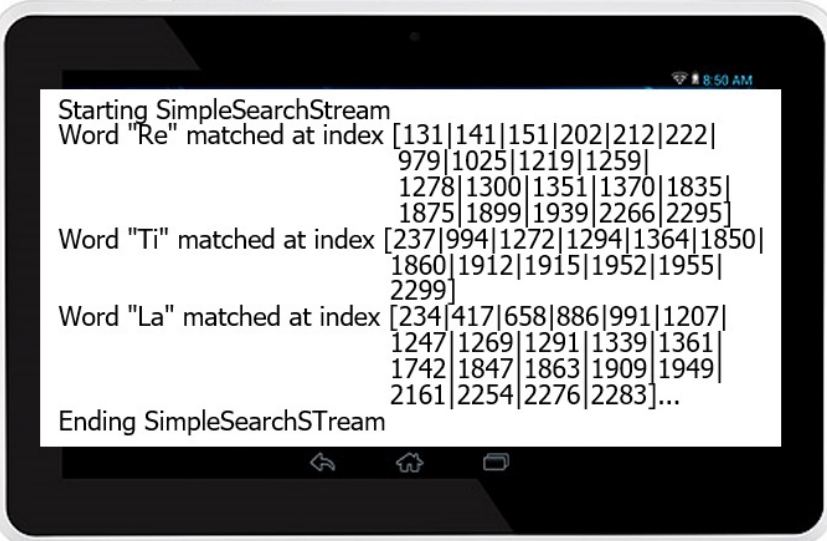
```
public String toString() {
```

```
    ...
```

```
    mList.stream()
```

```
        .collect(Collector.of(() -> new StringJoiner("|"),
```

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



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

*Combine two StringJoiners*

```
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

```
public String toString() {
```

```
    ...
```


```
    mList.stream()
```

```
        .collect(Collector.of(() -> new StringJoiner("|"),
```

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

*This finisher converts a  
StringJoiner to a String*

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



```
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
```

# Implementing Custom Non-Concurrent Collectors (Part 1)

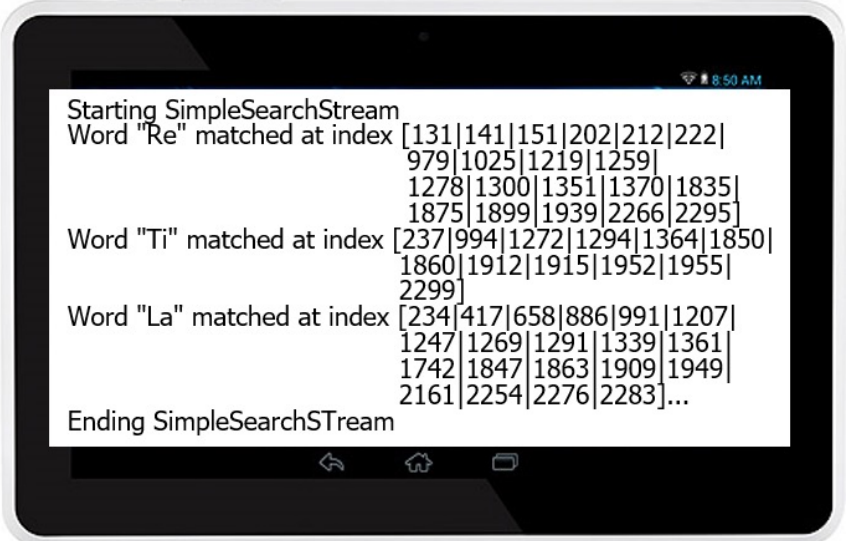
- The `SearchResults.toString()` method uses `Collector.of()` to format results

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

*Only four params are passed to `of()` since `Characteristics...` is an optional parameter!*

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

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



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

---

# 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,

         (r1, sr) -> r1.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,
```

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

```
        (r1, sr) -> r1.addAll(sr.getResultList()),
```

```
        StreamUtils::concat);
```

```
}
```

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,
         (r1, sr) -> r1.addAll(sr.getResultList()),
         StreamUtils::concat);
}
```

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

# 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,
    (r1, sr) -> r1.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,
         (r1, sr) -> r1.addAll(sr.getResultList()),
         StreamUtils::concat);
}
```

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

**StreamUtils::concat**;

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

See [SimpleSearchStream/src/main/java/Utils/StreamUtils.java](https://github.com/wordsearcher/wordsearcher/blob/master/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()



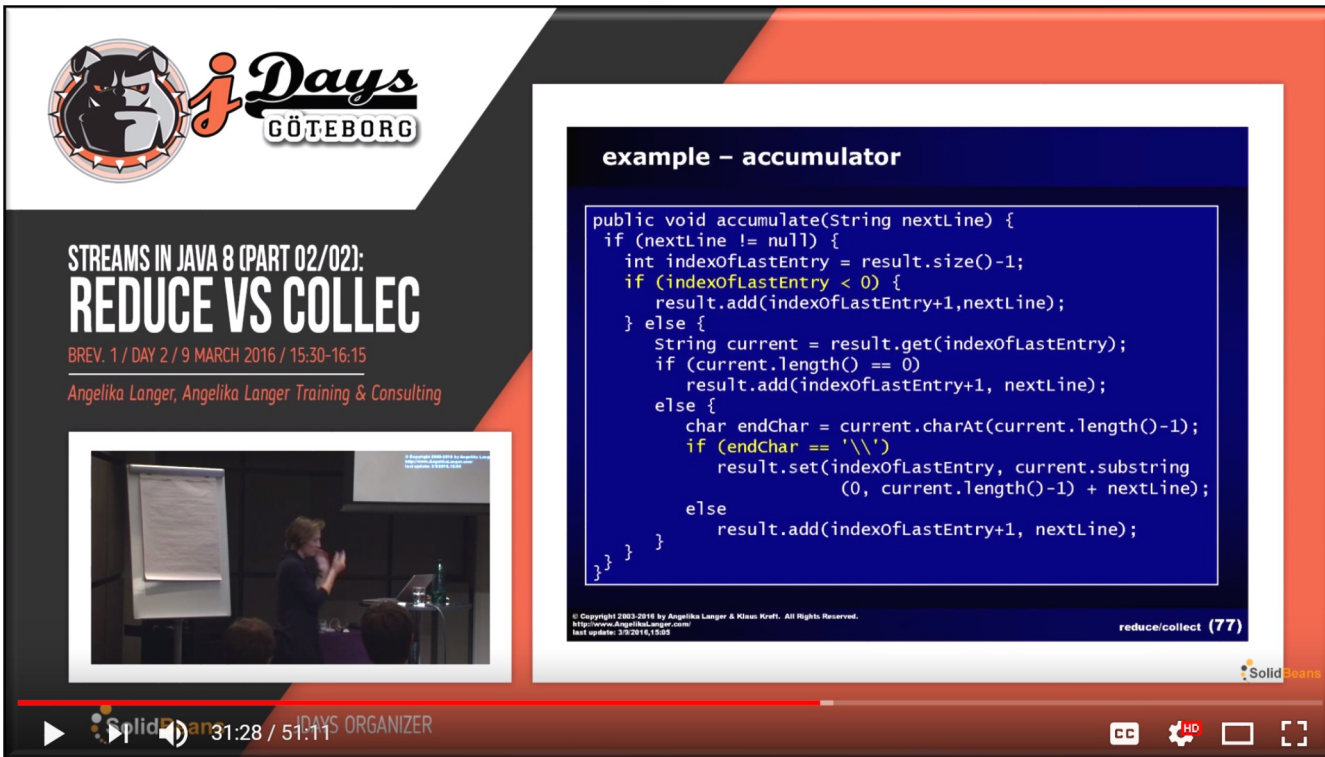
```
<<Java Interface>>  
Collector<T,A,R>  
  
● supplier():Supplier<A>  
● accumulator():BiConsumer<A,T>  
● combiner():BinaryOperator<A>  
● finisher():Function<A,R>  
● characteristics():Set<Characteristics>
```


```
<<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





**STREAMS IN JAVA 8 (PART 02/02):  
REDUCE VS COLLEC**

BREV. 1 / DAY 2 / 9 MARCH 2016 / 15:30-16:15

Angelika Langer, Angelika Langer Training & Consulting

```
example – accumulator

public void accumulate(String nextLine) {
    if (nextLine != null) {
        int indexOfLastEntry = result.size()-1;
        if (indexOfLastEntry < 0) {
            result.add(indexOfLastEntry+1,nextLine);
        } else {
            String current = result.get(indexOfLastEntry);
            if (current.length() == 0)
                result.add(indexOfLastEntry+1, nextLine);
            else {
                char endChar = current.charAt(current.length()-1);
                if (endChar == '\\')
                    result.set(indexOfLastEntry, current.substring
                        (0, current.length()-1) + nextLine);
                else
                    result.add(indexOfLastEntry+1, nextLine);
            }
        }
    }
}
```

© Copyright 2002-2016 by Angelika Langer & Klaus Krott. All Rights Reserved.  
http://www.AngelikaLanger.com  
last update: 3/9/2016, 13:05

reduce/collect (77)

SolidBeans

31:28 / 51:11 ORGANIZER

See [www.youtube.com/watch?v=H7VbRz9aj7c](http://www.youtube.com/watch?v=H7VbRz9aj7c)

---

# End of How to Implement Custom Non- Concurrent Collectors