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

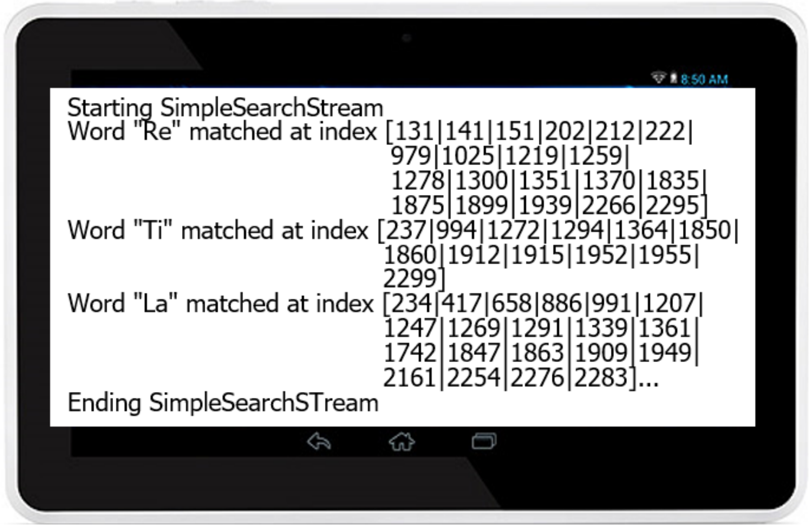
- Understand the structure & functionality of non-concurrent collectors for sequential streams
- Know the API for non-concurrent collectors
- Recognize how to apply pre-defined non-concurrent collectors
- Be able to implement custom non-concurrent collectors

```
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



```
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
```

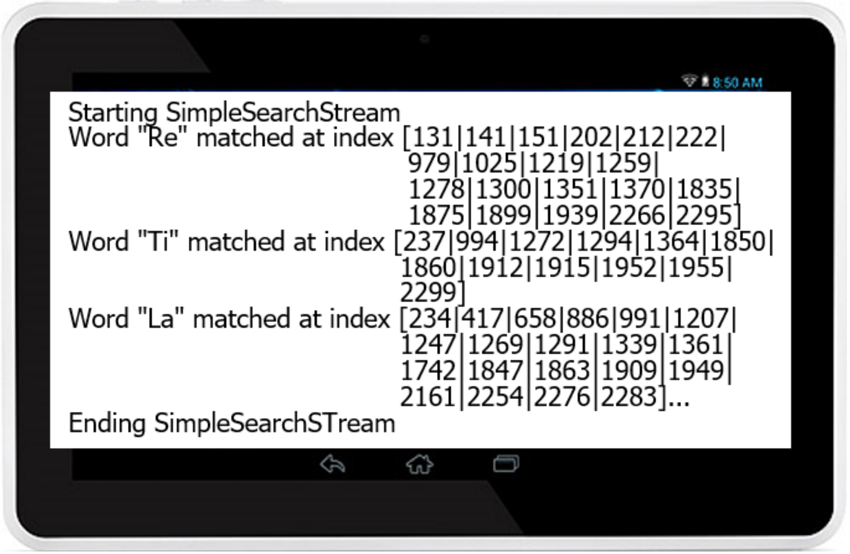
See github.com/douglascraigschmidt/LiveLessons/tree/master/SimpleSearchStream

Implementing Custom Non-Concurrent Collectors (Part 1)

Implementing Custom Non-Concurrent Collectors (Part 1)

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

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



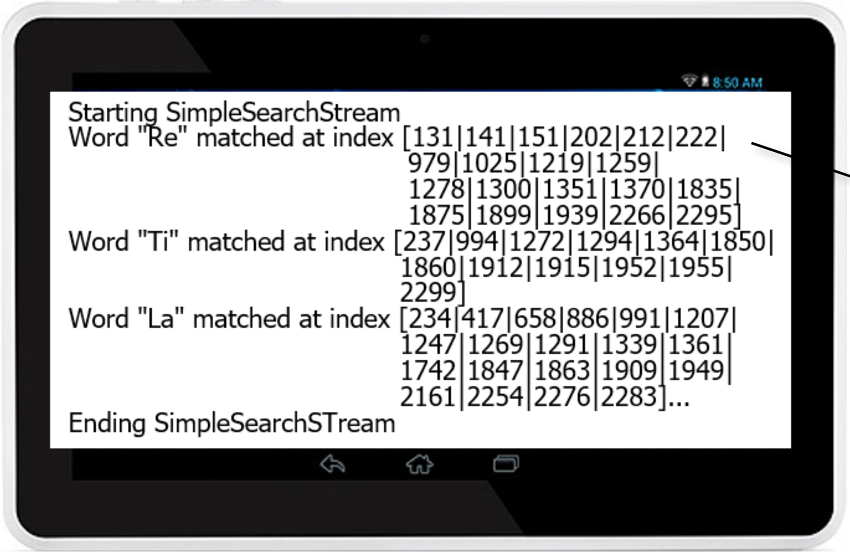
```
Starting SimpleSearchStream
Word "Re" matched at index [131|141|151|202|212|222|
                            979|1025|1219|1259|
                            1278|1300|1351|1370|1835|
                            1875|1899|1939|2266|2295]
Word "Ti" matched at index [237|994|1272|1294|1364|1850|
                            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/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)); ...
```



```
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

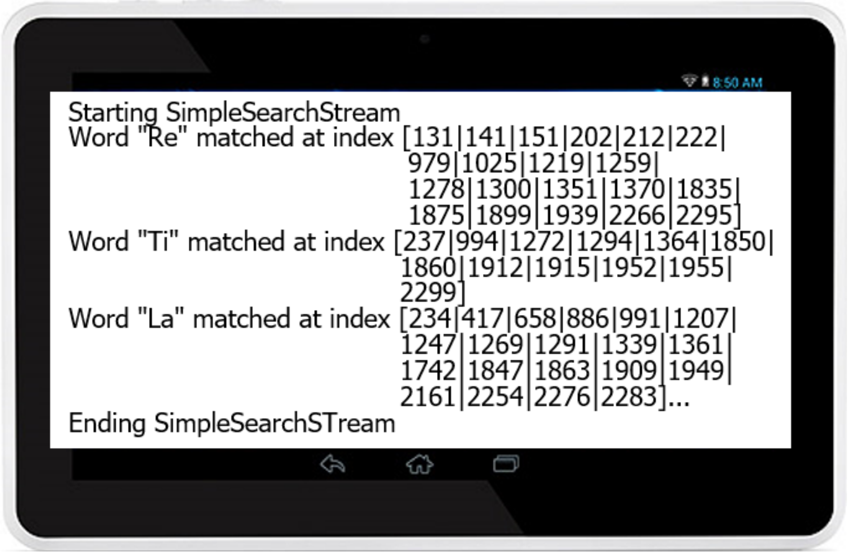
See SimpleSearchStream/src/main/java/search/SearchResults.java

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



```
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
```

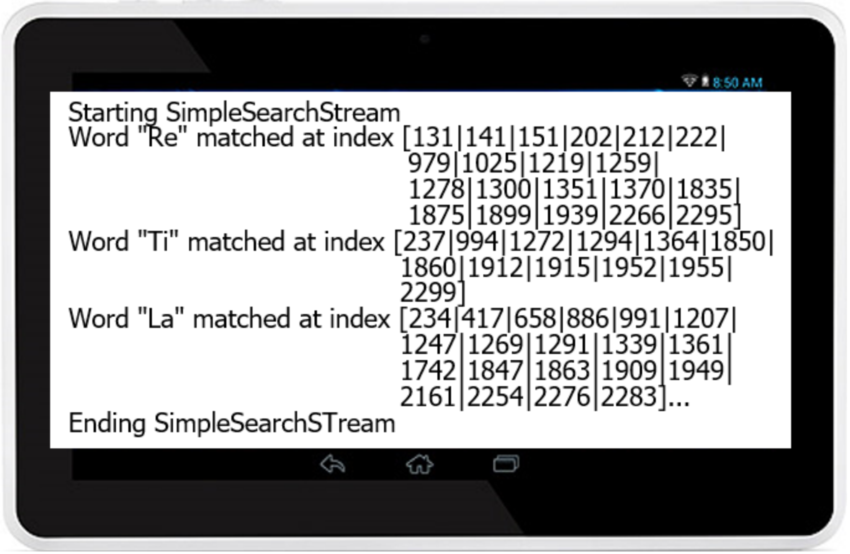
See docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html#of

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



```
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

Implementing Custom Non-Concurrent Collectors (Part 1)

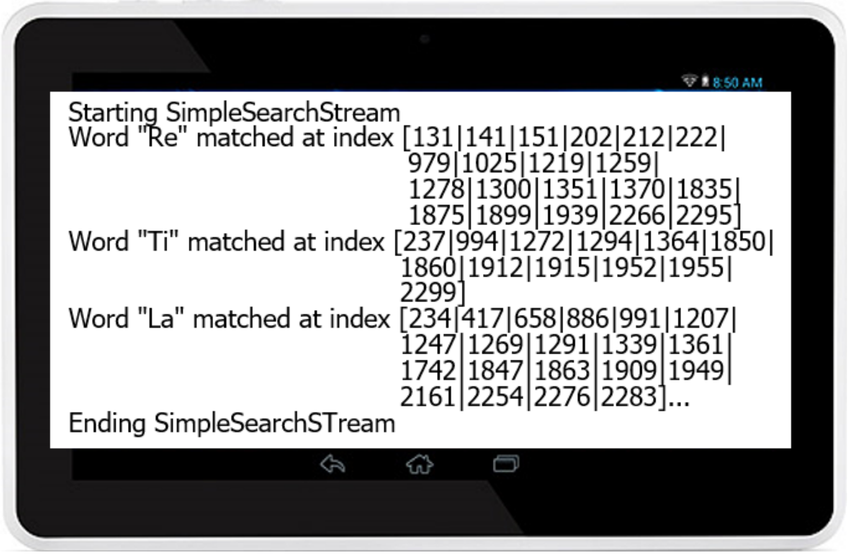
- The SearchResults.toString() method uses Collector.of() to format results

```
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 joiner

```
                              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
```

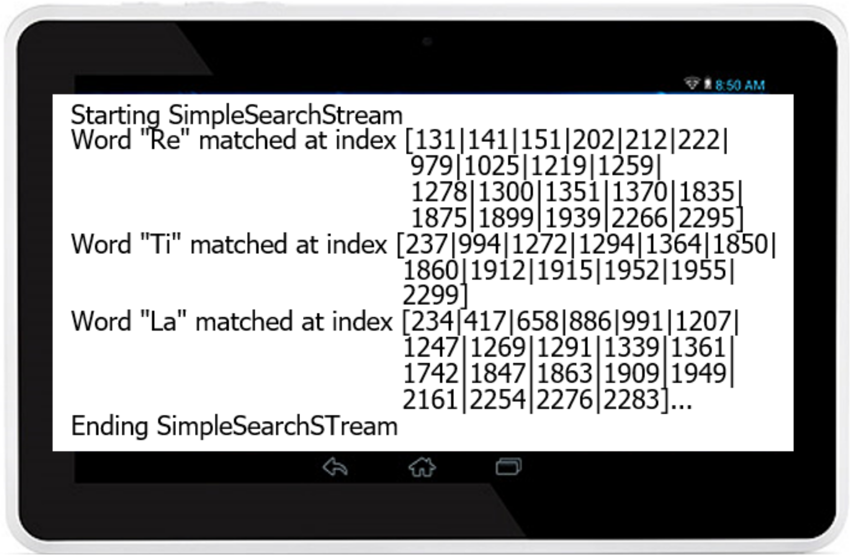
`(j, r)` is equivalent to `(StringJoiner j, SearchResults.Result r)`

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 string joiners



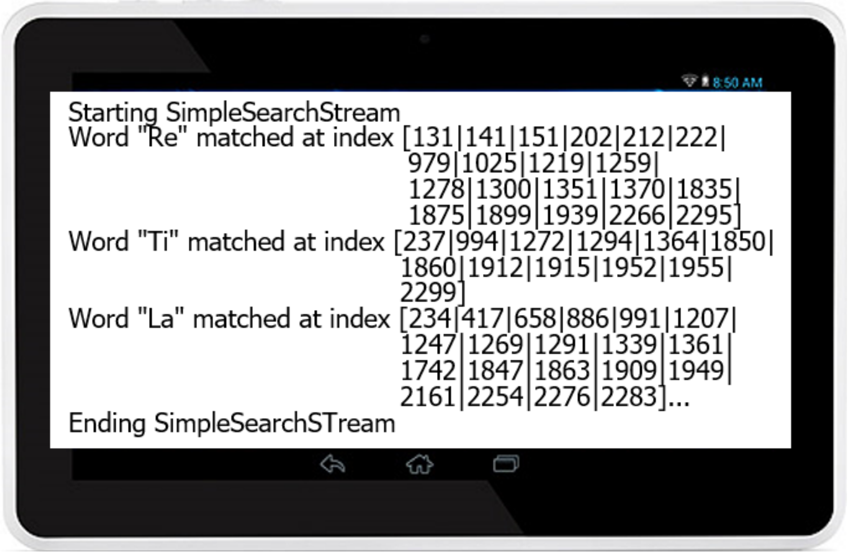
```
Starting SimpleSearchStream
Word "Re" matched at index [131|141|151|202|212|222|
                            979|1025|1219|1259|
                            1278|1300|1351|1370|1835|
                            1875|1899|1939|2266|2295]
Word "Ti" matched at index [237|994|1272|1294|1364|1850|
                           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
```

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)); ...
```



```
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
```

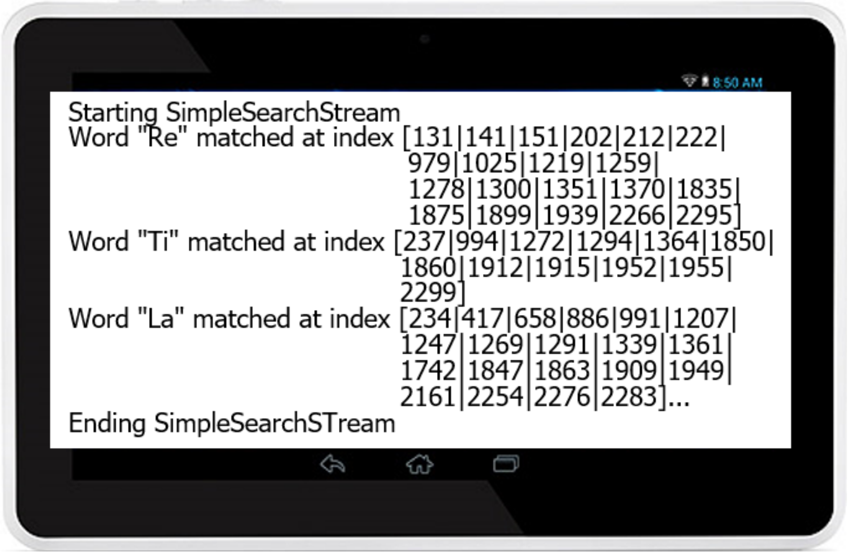
This finisher converts a string joiner to a string

Implementing Custom Non-Concurrent Collectors (Part 1)

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

```
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!



```
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,



           (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()

```
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()

```
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;
             });
}
```

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()),

              (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()

```
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()

```
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)

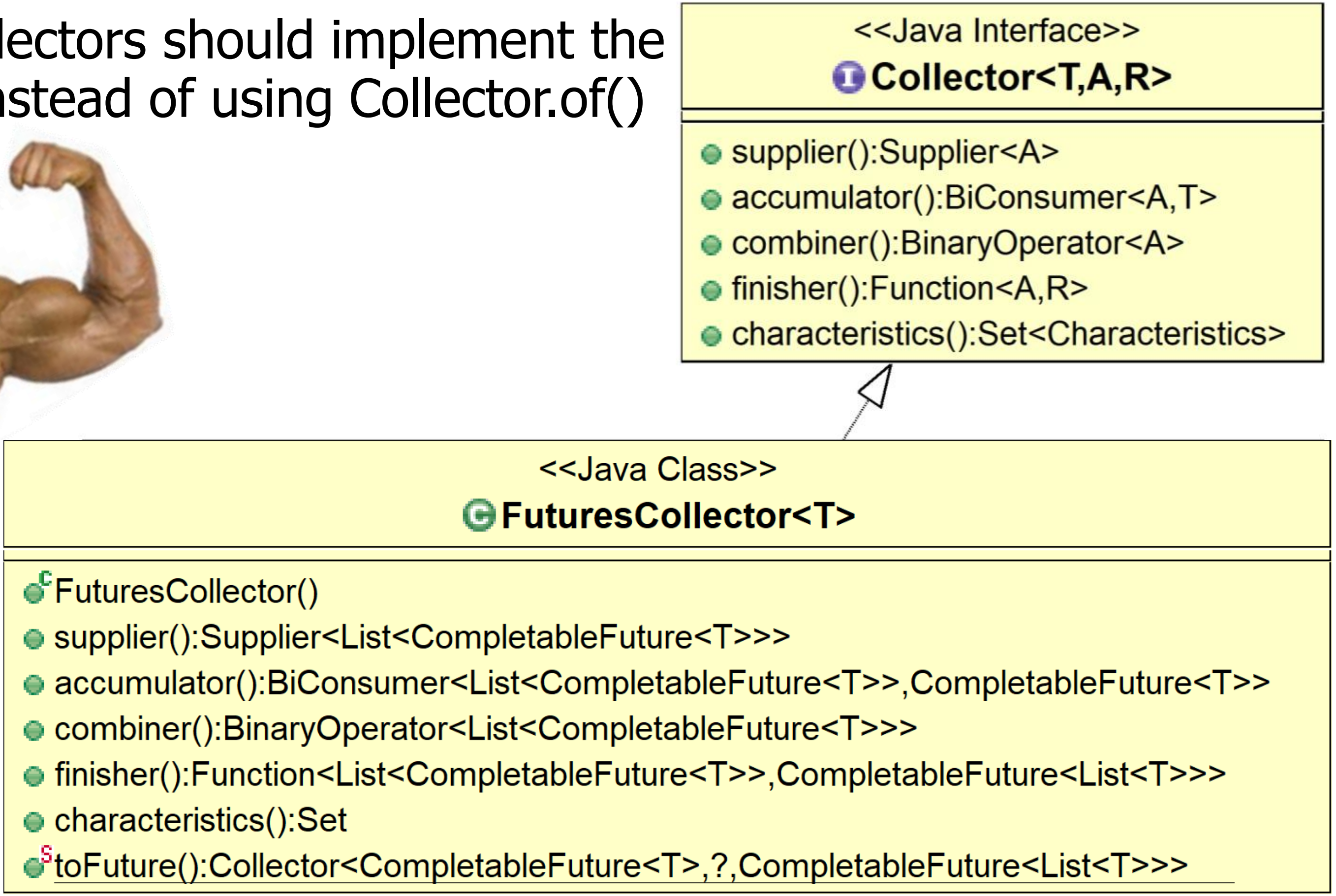
- The WordSearcher.toDownstreamCollector also uses Collector.of()

```
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()

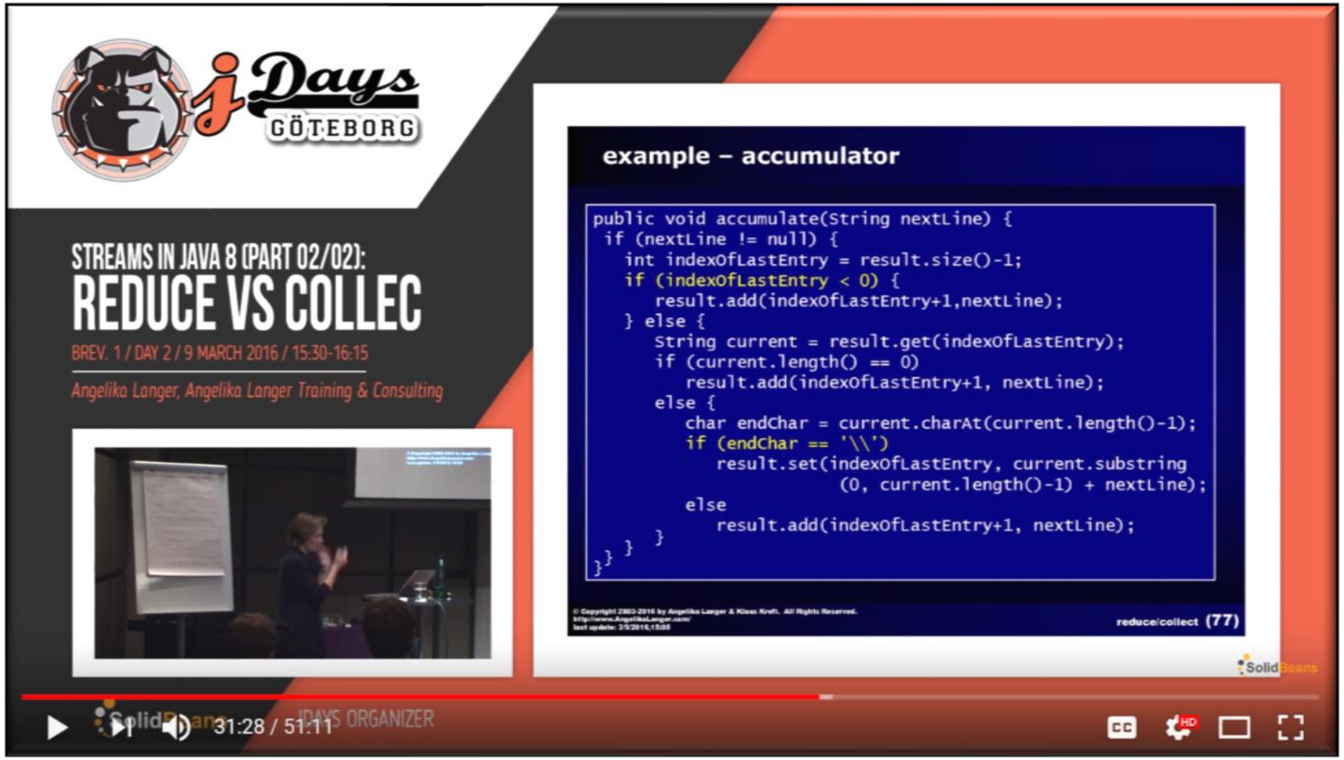


<<Java Interface>>
Collector<T,A,R>
supplier():Supplier<A>
accumulator():BiConsumer<A,T>
combiner():BinaryOperator<A>
finisher():Function<A,R>
characteristics():Set<Characteristics>
<<Java Class>>
FuturesCollector<T>
FuturesCollector()
supplier():Supplier<List<CompletableFuture<T>>>
accumulator():BiConsumer<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


jDays
GÖTEBORG
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
SolidBeans
JDAYS ORGANIZER
31:28 / 51:11

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);
      }
   }
 }
}
```

reduce/collect (77)

See www.youtube.com/watch?v=H7VbRz9aj7c

End of Java Streams: Implementing Custom Non-Concurrent Collectors