Overview of Java 8 Streams (Part 4)

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 Java 8 streams, e.g.,
  - Fundamentals of streams
  - Common stream aggregate operations
  - “Splittable iterators” (Spliterators)
  - Terminating a stream
Terminating a Stream
Terminating a Stream

- Every stream finishes with a terminal operation that yields a non-stream result

Stream
  .of("horatio",
    "laertes",
    "Hamlet", ...)
  .filter(s -> toLowerCase(s.charAt(0)) == 'h')
  .map(this::capitalize)
  .sorted()
  .forEach(System.out::println);

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex12
Every stream finishes with a terminal operation that yields a non-stream result, e.g.

- No value at all
  - i.e., only side-effects!
Terminating a Stream

• Every stream finishes with a terminal operation that yields a non-stream result, e.g.
  • No value at all
  • The result of a reduction operation

See docs.oracle.com/javase/tutorial/collections/streams/reduction.html
Terminating a Stream

- Several terminal operations return no value at all

```java
void runForEach() {
    ...

    Stream
        .of("horatio", "laertes", "Hamlet", ...)
        .filter(s -> toLowerCase(s.charAt(0)) == 'h')
        .map(this::capitalize)
        .sorted()
        .forEach(System.out::println);

    ...
}
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex12
Several terminal operations return no value at all

```java
void runForEach() {
    ... 
    Stream.of("horatio", "laertes", "Hamlet", ...)
        .filter(s -> toLowerCase(s.charAt(0)) == 'h')
        .map(this::capitalize)
        .sorted()
        .forEach(System.out::println);
    ... 
}
```

Perform the designated action on each element of this stream

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#forEach](http://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#forEach)
Several terminal operations return no value at all.

```java
void runForEach() {
    List<String> results =
        new ArrayList<>();

    Stream.of("horatio", "laertes", "Hamlet", ...)
        .filter(s -> toLowerCase(s.charAt(0)) == 'h')
        .map(this::capitalize)
        .sorted()
        .forEach(results::add);

    ...
}
```

The lambda passed to `forEach()` is allowed to have side-effects.

Several terminal operations return no value at all.

```java
void runForEach() {
    List<String> results =
        new ArrayList<>();

    Stream.of("horatio", "laertes", "Hamlet", ...)
        .parallel()
        .filter(s -> s.toLowerCase().charAt(0) == 'h')
        .map(this::capitalize)
        .sorted()
        .forEach(results::add);
    ...
}
```

Avoid using `forEach()` with side-effects in a parallel stream!!!

See [docs.oracle.com/javase/tutorial/collectionsSTREAMS/parallelism.html#side_effects](docs.oracle.com/javase/tutorial/collectionsSTREAMS/parallelism.html#side_effects)
• Other terminal operations return the result of a reduction operation

```java
void runCollectTo*() {
    List<String> characters = Arrays.asList("horatio", "laertes", "Hamlet", ...);

    ...<String> results = characters
        .stream()
        .filter(s -> toLowerCase(...) == 'h')
        .map(this::capitalize)
        .sorted()
        .collect(...); ...
}
```

See `docs.oracle.com/javase/tutorial/collections/streams/reduction.html`
Terminating a Stream

- Other terminal operations return the result of a reduction operation, e.g.
  - a collection

```java
void runCollectTo*() {
    List<String> characters = Arrays.asList("horatio", "laertes", "Hamlet", ...);

    ...<String> results = characters
        .stream()
        .filter(s -> toLowerCase(...) == 'h')
        .map(this::capitalize)
        .sorted()
        .collect(...); ...
```

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#collect
Terminating a Stream

- Other terminal operations return the result of a reduction operation, e.g.
  - a collection

```java
void runCollectTo*() {
    List<String> characters = Arrays.asList("horatio", "laertes", "Hamlet", ...);

    ...<String> results = characters
        .stream()
        .filter(s ->
            toLowerCase(...) == 'h')
        .map(this::capitalize)
        .sorted()
        .collect(...); ...
```

A collector performs reduction operations, e.g., summarizing elements according to various criteria, accumulating elements into various types of collections, etc.

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html](docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html)
Terminating a Stream

- Other terminal operations return the result of a reduction operation, e.g.
  - a collection

```java
void runCollectToList() {
    List<String> characters = Arrays.asList("horatio", "laertes", "Hamlet, ...");
    List<String> results = characters
        .stream()
        .filter(s -> toLowerCase(...) == 'h')
        .map(this::capitalize)
        .sorted()
        .collect(toList()); ...
    // Collect the results into a ArrayList

See docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html#toList
Terminating a Stream

- Other terminal operations return the result of a reduction operation, e.g.
  - a collection

```java
void runCollectToSet() {
    List<String> characters = Arrays.asList("horatio", "laertes", "Hamlet", ...);
    Set<String> results = characters
        .stream()
        .filter(s -> toLowerCase(...) == 'h')
        .map(this::capitalize)
        .collect(toSet());
    ...}
```

*Collect the results into a HashSet, which has no duplicate entries*

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html#toSet](docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html#toSet)
Terminating a Stream

Other terminal operations return the result of a reduction operation, e.g.

- a collection

```java
void runCollectToMap() {
    List<String> characters = Arrays.asList("horatio", "laertes", "Hamlet", ...);
    Map<String, Integer> results =
        characters.stream()
            .filter(s ->
                toLowerCase(...) == 'h')
            .map(this::capitalize)
            .collect(toMap(identity(), String::length, Integer::sum));
    ...}
```

Collect results into a HashMap, along with the length of (merged duplicate) entries

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html#toMap](http://docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html#toMap)
Terminating a Stream

- Other terminal operations return the result of a reduction operation, e.g.
  - a collection

```java
void runCollectGroupingBy() {
    List<String> characters = Arrays.asList("horatio",
                                          "laertes",
                                          "Hamlet", ...);
    Map<String, Long> results = ...
                              .collect
                              (groupingBy
                               (identity(),
                                TreeMap::new,
                                summingLong
                                (String::length)));
    ...
}
```

Collect the results into a TreeMap by grouping elements according to name (key) & name length (value)

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html#groupingBy](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html#groupingBy)
Terminating a Stream

- Other terminal operations return the result of a reduction operation, e.g.
  - a collection

```java
void runCollectGroupingBy() {
    List<String> characters = Arrays.asList("horatio", "laertes", "Hamlet", ...);
    Map<String, Long> results = ...
        .collect(
            groupingBy
                (identity(),
                 TreeMap::new,
                 summingLong
                    (String::length)));
    ...
}
```

`groupingBy()` partitions a stream via a "classifier" function, e.g., by the `identity()` function that always returns its input argument.

See [docs.oracle.com/javase/8/docs/api/java/util/function/Function.html#identity](https://docs.oracle.com/javase/8/docs/api/java/util/function/Function.html#identity)
• Other terminal operations return the result of a reduction operation, e.g.
  • a collection

```
void runCollectGroupingBy() {
    List<String> characters =
        Arrays.asList("horatio",
                      "laertes",
                      "Hamlet", ...);
    Map<String, Long> results =
        ...
        .collect
            (groupingBy
                (identity(),
                 TreeMap::new,
                 summingLong
                 (String::length)));
    ...
}
```
Terminating a Stream

- Other terminal operations return the result of a reduction operation, e.g.
  - a collection

```java
void runCollectGroupingBy() {
    List<String> characters = Arrays.asList("horatio", "laertes", "Hamlet", ...);
    Map<String, Long> results = ...
        .collect
            (groupingBy
                (identity(),
                 TreeMap::new,
                 summingLong
                     (String::length)));
    ...
}
```

A “downstream collector” defines a collector applied by the Java runtime to the results of an earlier collector.

See docs.oracle.com/javase/tutorial/collections/stream/reduction.html
Terminating a Stream

- Other terminal operations return the result of a reduction operation, e.g.
  - a collection

```java
void runCollectReduce() {
    Map<String, Long> matchingCharactersMap = Pattern.compile("","").splitAsStream("horatio,Hamlet,...").collect(
        groupingBy
        (identity(),
            TreeMap::new,
            summingLong
            (String::length)));
}
```

See [docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html#splitAsStream](http://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html#splitAsStream)
 Terminating a Stream

- Other terminal operations return the result of a reduction operation, e.g.
- a collection

```java
void runCollectReduce() {
    Map<String, Long>
        matchingCharactersMap =
        Pattern.compile(",,"
            .splitAsStream
                ("horatio,Hamlet,...")
                .collect
                ((groupingBy
                    (identity(),
                    TreeMap::new,
                    summingLong
                        (String::length))));
```

Collect the results into a TreeMap by grouping elements according to name (key) & name length (value)

See docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html#groupingBy
Terminating a Stream

- Other terminal operations return the result of a reduction operation, e.g.
  - a collection
  - a primitive value

```java
void runCollectReduce() {
    Map<String, Long>
    matchingCharactersMap = ...

    long countOfNameLengths =
    matchingCharactersMap
    .values()
    .stream()
    .reduce(0L,
           (x, y) -> x + y);
}
```

*Sum up the lengths of all character names in Hamlet*

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#reduce](http://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#reduce)
Other terminal operations return the result of a reduction operation, e.g.

- a collection
- a primitive value

0 is the "identity," i.e., the initial value of the reduction & the default result if there are no elements in the stream

```java
void runCollectReduce() {
    Map<String, Long>
        matchingCharactersMap = ...

    long countOfNameLengths =
        matchingCharactersMap
            .values()
            .stream()
            .reduce(0L,
                (x, y) -> x + y);
}```
void runCollectReduce() {
    Map<String, Long> matchingCharactersMap =
        ...

    long countOfNameLengths =
        matchingCharactersMap.values().stream().reduce(0L, 
            (x, y) -> x + y);

This lambda is the “accumulator,” which is a stateless function that combines two values into a single (immutable) “reduced” value
Terminating a Stream

- Other terminal operations return the result of a reduction operation, e.g.
  - a collection
  - a primitive value

```java
void runCollectReduce() {
    Map<String, Long>
        matchingCharactersMap = ...

    long countOfNameLengths =
        matchingCharactersMap
            .values()
            .parallelStream()
            .reduce(0L,
                    (x, y) -> x + y,
                    (x, y) -> x + y);
```

There's a 3 parameter "map/reduce" version of reduce() that's used in parallel streams

See [www.youtube.com/watch?v=oWIWEKNM5Aw](www.youtube.com/watch?v=oWIWEKNM5Aw)
Terminating a Stream

- Other terminal operations return the result of a reduction operation, e.g.
  - a collection
  - a primitive value

```java
void runCollectReduce() {
    Map<String, Long> matchingCharactersMap = ... 
    long countOfNameLengths = 
        matchingCharactersMap
            .values()
            .stream()
            .sum()
}
```

There's a `sum()` method that simplifies the use of `reduce()`

See [docs.oracle.com/javase/8/docs/api/java/util/stream/LongStream.html#sum](docs.oracle.com/javase/8/docs/api/java/util/stream/LongStream.html#sum)
Terminating a Stream

• Other terminal operations return the result of a reduction operation, e.g.
  • a collection
  • a primitive value

```
parallelStream()

Input x

Intermediate operation (behavior f)

Output f(x)

Intermediate operation (behavior g)

Output g(f(x))

Terminal operation (behavior h)
```

collect() & reduce() terminal operations work seamlessly with parallel streams
End of Overview of Java 8 Streams (Part 4)