The Java Streams reduce() Terminal Operation (Part 1)

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 common terminal operations, e.g.
  - forEach()
  - collect()
  - reduce()

- Know how reduce() performs an immutable reduction
- Both the two- & three-parameter versions

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

    long sumOfNameLengths =
        matchingCharactersMap
            .values()
            .stream()
            .reduce(0L,
                    Long::sum);
}
```

We showcase reduce() using the Hamlet program

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex12](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex12)
A Stream Terminal Operation That Returns a Primitive
The reduce() terminal operation typically returns a primitive value.

```java
def runCollectReduce1():
    matchingCharactersMap = ...
    sumOfNameLengths = matchingCharactersMap.values().stream().reduce(0L, Long::sum);
```

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#reduce](docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#reduce)
A Stream Terminal Operation That Returns a Primitive

- The reduce() terminal operation typically returns a primitive value

```java
void runCollectReduce1() {
    Map<String, Long> matchingCharactersMap =
        ...
    .collect
        (groupingBy
            (identity(),
             TreeMap::new, summingLong
             (String::length)));}
```

Create a map associating the names of Hamlet characters with their name lengths.

See [github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex12](https://github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex12)
A Stream Terminal Operation That Returns a Primitive

- The reduce() terminal operation typically returns a primitive value

```java
void runCollectReduce1() {
    Map<String, Long> matchingCharactersMap = ...

    long sumOfNameLengths = matchingCharactersMap
        .values()
        .stream()
        .reduce(0L, Long::sum);
}
```

Convert the map’s collection of values into a stream of long values.
A Stream Terminal Operation That Returns a Primitive

- The `reduce()` terminal operation typically returns a primitive value

```java
void runCollectReduce1() {
    Map<String, Long> matchingCharactersMap = ...

    long sumOfNameLengths = matchingCharactersMap.values().stream().reduce(0L, Long::sum);
}
```

**Sum up the lengths of all character names in Hamlet.**
A Stream Terminal Operation That Returns a Primitive

- The reduce() terminal operation typically returns a primitive value

```java
void runCollectReduce1() {
    Map<String, Long>
        matchingCharactersMap = ...

    long sumOfNameLengths =
        matchingCharactersMap
            .values()
            .stream()
            .reduce(0L,
                Long::sum);
```

*0 is the “identity,” i.e., the initial value of the reduction & the default result if there are no elements in the stream.*
void runCollectReduce1() {
    Map<String, Long>
        matchingCharactersMap = ...

    long sumOfNameLengths =
        matchingCharactersMap
            .values()
            .stream()
            .reduce(0L,
                   Long::sum);

This method reference is an "accumulator," which is a stateless function that combines two values into a single (immutable) "reduced" value.

See docs.oracle.com/javase/8/docs/api/java/lang/Long.html#sum
The reduce() terminal operation typically returns a primitive value

```java
void runCollectReduce1() {
    Map<String, Long>
    matchingCharactersMap = ...

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

A lambda expression could also be used here.

See [stackoverflow.com/a/24493905](https://stackoverflow.com/a/24493905)
The Three-Parameter Version of reduce()
The Three-Parameter Version of reduce()

- The three-parameter version of reduce() separates the accumulator from the combiner

```java
void runCollectMapReduce() {
    List<String> characterList = ...

    long sumOfNameLengths =
        characterList
            .parallelStream()
            .reduce(0L,
                (sum, s) ->
                sum + s.length(),
                Long::sum);
```

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)
The Three-Parameter Version of `reduce()`

- The three-parameter version of `reduce()` separates the accumulator from the combiner.
- This variant is primarily used for parallel streams.

```java
void runCollectMapReduce() {
    List<String> characterList = ...

    long sumOfNameLengths = characterList
        .parallelStream()
        .reduce(0L,
               (sum, s) ->
               sum + s.length(),
               Long::sum);
```

See [www.youtube.com/watch?v=oWIWEKNM5Aw](www.youtube.com/watch?v=oWIWEKNM5Aw)
The Three-Parameter Version of reduce()

- The three-parameter version of `reduce()` separates the accumulator from the combiner.
- This variant is primarily used for parallel streams.

```java
void runCollectMapReduce() {
    List<String> characterList = ...

    long sumOfNameLengths =
        characterList
            .parallelStream()
            .reduce(0L,
                    (sum, s) ->
                        sum + s.length(),
                    Long::sum);
}
```

Generate a consistently capitalized & sorted list of names of Hamlet characters starting with the letter 'h'.
The Three-Parameter Version of `reduce()`

- The three-parameter version of `reduce()` separates the accumulator from the combiner.
- This variant is primarily used for parallel streams.

```java
void runCollectMapReduce() {
    List<String> characterList = ...

    long sumOfNameLengths = characterList
                            .parallelStream()
                            .reduce(0L, 
                                    (sum, s) -> sum + s.length(),
                                    Long::sum);
}
```
The Three-Parameter Version of `reduce()`

- The three-parameter version of `reduce()` separates the accumulator from the combiner.
- This variant is primarily used for parallel streams.

```java
void runCollectMapReduce() {
    List<String> characterList = ...

    long sumOfNameLengths = characterList
        .parallelStream()
        .reduce(0L,
               (sum, s) ->
                sum + s.length(),
                Long::sum);
```
The Three-Parameter Version of reduce()

- The three-parameter version of reduce() separates the accumulator from the combiner.
- This variant is primarily used for parallel streams.

```java
void runCollectMapReduce() {
    List<String> characterList = ...

    long sumOfNameLengths = characterList
        .parallelStream()
        .reduce(0L,
               (sum, s) -> sum + s.length(),
               Long::sum);
}
```

This lambda expression is an accumulator BiFunction that performs the "map" operation in the apply phase.

See docs.oracle.com/javase/8/docs/api/java/util/function/BiFunction.html
The Three-Parameter Version of reduce()

- The three-parameter version of reduce() separates the accumulator from the combiner.
- This variant is primarily used for parallel streams.

```java
void runCollectMapReduce() {
    List<String> characterList = ...
    long sumOfNameLengths =
        characterList
            .parallelStream()
            .reduce(0L,
                (sum, s) ->
                sum + s.length(),
                Long::sum);
}
```

This method reference is a combiner BinaryOperator that performs the "reduce" operation in the combine phase.

See [docs.oracle.com/javase/8/docs/api/java/lang/Long.html#sum](https://docs.oracle.com/javase/8/docs/api/java/lang/Long.html#sum)
The Three-Parameter Version of reduce()

- The three-parameter version of reduce() separates the accumulator from the combiner.
- This variant is primarily used for parallel streams.
- It can also be used when the type being streamed is different from the type of the accumulator.

```java
void runCollectMapReduceEx() {
    Map<String, Double> base =
        new HashMap<>(); { ... }
    Map<String, Double> actual =
        new HashMap<>(); { ... }
    Double breakoutFactor = base
        .entrySet()
        .stream()
        .reduce(0.0,
                (sum, entry) -> {
                    ...  
                    return sum + ...;
                },
                Double::sum);
}
```

'sum' is a Double & 'entry' is a Map.Entry<>
Replacing `reduce()` with `sum()`
Replacing reduce() with sum()

- The sum() terminal operation can replace reduce() for streams of primitive values.

```java
class Example {
    void runCollectReduce2() {
        Map<String, Long> matchingCharactersMap = ...

        long sumOfNameLengths = matchingCharactersMap.values().stream().mapToLong(Long::longValue).sum();
    }
}
```

See docs.oracle.com/javase/8/docs/api/java/util/stream/LongStream.html#sum
The sum() terminal operation can replace reduce() for streams of primitive values.

```java
void runCollectReduce2() {
  Map<String, Long>
  matchingCharactersMap = ...

  long sumOfNameLengths =
    matchingCharactersMap
      .values()
      .stream()
      .mapToLong(Long::longValue)
      .sum();
```

Convert the map into a stream of long values.
The `sum()` terminal operation can replace `reduce()` for streams of primitive values.

```java
void runCollectReduce2() {
    Map<String, Long> matchingCharactersMap = ...

    long sumOfNameLengths = matchingCharactersMap.values().stream()
        .mapToLong(Long::longValue)
        .sum();
```

Map the stream of `Long` objects into a stream of long primitives.

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#mapToLong](docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#mapToLong)
The sum() terminal operation can replace reduce() for streams of primitive values.

```java
void runCollectReduce2() {
    Map<String, Long>
        matchingCharactersMap = ...

    long sumOfNameLengths =
        matchingCharactersMap
            .values()
            .stream()
            .mapToLong(Long::longValue)
            .sum();
}
```

Sum the stream of long primitives into a single result.

See [docs.oracle.com/javase/8/docs/api/java/util/stream/LongStream.html#sum](http://docs.oracle.com/javase/8/docs/api/java/util/stream/LongStream.html#sum)
End of the Java Streams reduce() Terminal Operation (Part 1)