

The Java Streams reduce() Terminal Operation (Part 2)

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 what `reduce()` does

- Recognize interesting variants of `collect()` & `reduce()`

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

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

*We showcase `reduce()`
using the Hamlet program*

Interesting Variants of collect() & reduce()

Interesting Variants of collect() & reduce()

- Thus far we've used reduce() to return a primitive value

```
void runCollectReduce() {  
    Map<String, Long>  
        matchingCharactersMap =  
        ...  
  
    long sumOfNameLengths =  
        matchingCharactersMap  
            .values()  
            .stream()  
            .reduce(0L,  
                    Long::sum) ;  
}
```

See earlier lesson on "The Java Stream reduce() Terminal Operation (Part 1)"

Interesting Variants of collect() & reduce()

- However, collect() can also be used to return a primitive value

```
void runCollectReduce3() {  
    Map<String, Long>  
        matchingCharactersMap =  
        ...  
  
    long sumOfNameLengths =  
        matchingCharactersMap  
            .values()  
            .stream()  
            .collect  
                (summingLong  
                    (Long::longValue));  
}
```

See earlier lesson on "*Java Streams: the collect() Terminal Operation*"

Interesting Variants of collect() & reduce()

- However, collect() can also be used to return a primitive value

```
void runCollectReduce3() {  
    Map<String, Long>  
        matchingCharactersMap =  
        ...  
  
    long sumOfNameLengths =  
        matchingCharactersMap  
            .values()  
            .stream()  
            .collect  
                (summingLong  
                 (Long::longValue));  
}
```

Trigger the stream processing.

Interesting Variants of collect() & reduce()

- However, collect() can also be used to return a primitive value

```
void runCollectReduce3() {  
    Map<String, Long>  
        matchingCharactersMap =  
        ...  
  
    long sumOfNameLengths =  
        matchingCharactersMap  
            .values()  
            .stream()  
            .collect  
                (summingLong  
                 (Long::longValue));  
}
```

Return a collector that produces the sum of a long-value function applied to input elements.

Interesting Variants of collect() & reduce()

- Likewise, reduce() can be used to return a non-primitive value

```
void streamReduceConcat
    (boolean parallel) {
    ...
    Stream<String> stringStream =
        allWords.stream();

    ...

    String words = stringStream
        .reduce("",
            (x, y) -> x + y);
```


Interesting Variants of collect() & reduce()

- Likewise, reduce() can be used to return a non-primitive value

```
void streamReduceConcat
    (boolean parallel) {
    ...
    Stream<String> stringStream =
        allStrings.stream();
    ...
    String words = stringStream
        .reduce("",
            (x, y) -> x + y);
```

*Create a stream of all the
String objects in a List*

Interesting Variants of collect() & reduce()

- Likewise, reduce() can be used to return a non-primitive value

```
void streamReduceConcat  
    (boolean parallel) {  
    ...  
    Stream<String> stringStream =  
        allWords.stream();  
  
    ...  
  
    String words = stringStream  
        .reduce("",  
            (x, y) -> x + y);
```

reduce() creates an immutable String object containing all concatenated words in a stream

Interesting Variants of collect() & reduce()

- Likewise, reduce() can be used to return a non-primitive value

```
void streamReduceConcat
    (boolean parallel) {
    ...
    Stream<String> stringStream =
        allWords.stream();

    ...

    String words = stringStream
        .reduce("",
            (x, y) -> x + y);
```

However, this code is inefficient due to string concatenation overhead!!

See upcoming lesson on "Java Parallel Stream Internals: Combining Results (Part 2)"

Interesting Variants of collect() & reduce()

- The joining() Collector can be used to alleviate the overhead of Java String concatenation

```
void streamCollectJoining  
    (boolean parallel) {  
    ...  
    Stream<String> stringStream =  
        allWords.stream();  
  
    ...  
  
    String words = stringStream  
        .collect(Collectors  
            .joining());
```

*Efficiently concatenate
Java String objects*



See upcoming lesson on "Java Parallel Stream Internals: Combining Results (Part 2)"

Interesting Variants of collect() & reduce()

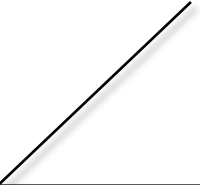
- reduce() can also return a stream!

```
generateOuterStream  
    (Options.instance().iterations())
```

```
.map(ex35::innerStream)
```

```
.reduce(Stream::concat)  
.orElse(Stream.empty())
```

```
.collect(Collectors.toList());
```



*reduce() returns a stream that
is then processed via collect()!*

End of the Java Streams reduce() Terminal Operation (Part 2)