The Java Streams forEach() & forEachOrdered() Terminal Operations

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() & forEachOrdered()
    - These two "run-to-completion" terminal operations return no value at all & only have side-effects

Java Stream : forEachOrdered() vs forEach()

By Arvind Rai, June 13, 2020

On this page we will provide differences between `Stream.forEachOrdered()` and `Stream.forEach()` methods. Both methods perform an action as `Consumer`. The difference between `forEachOrdered()` and `forEach()` methods is that `forEachOrdered()` will always perform given action in encounter order of elements in stream whereas `forEach()` method is non-deterministic. In parallel stream `forEach()` method may not necessarily respect the order whereas `forEachOrdered()` will always respect the order. In sequential stream both methods respect the order. So we should use `forEachOrdered()` method, if we want action to be perform in encounter order in every case whether the stream is sequential or parallel. If the stream is sequential, we can use any method to respect order. But if stream can be parallel too, then we should use `forEachOrdered()` method to respect the order.

See www.concretepage.com/java/java-8/java-stream-foreachordered-vs-foreach

Learning Objectives in this Part of the Lesson

- Understand common terminal operations, e.g.
  - forEach() & forEachOrdered()
    - These two "run-to-completion" terminal operations return no value at all & only have side-effects

```
void runForEach() {
  ...

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

We showcase these two terminal operations using the Hamlet program

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex12

Learning Objectives in this Part of the Lesson

- Understand common terminal operations, e.g.
  - forEach() & forEachOrdered()
    - These two "run-to-completion" terminal operations return no value at all & only have side-effects
    - We also discuss common traps & pitfalls with these operations

Stream Terminal Operations That Return No Value

Stream Terminal Operations That Return No Value

- The forEach() & forEachOrdered() terminal operations return no value at all

```
void runForEach() {
  ...


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

See howtodoinjava.com/java8/foreach-method-example

Stream Terminal Operations That Return No Value

- The forEach() & forEachOrdered() terminal operations return no value at all
  - i.e., they only have side-effects!


```
void runForEach() {
  ...

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

Stream Terminal Operations That Return No Value

- The forEach() & forEachOrdered() terminal operations return no value at all
  - i.e., they only have side-effects!

Several variants of forEach() are showcased in this example.

```
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

Stream Terminal Operations That Return No Value

- The forEach() & forEachOrdered() terminal operations return no value at all
  - i.e., they only have side-effects!

Create & process a stream consisting of characters from the play "Hamlet"

```
void runForEach() {
  ...

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

Stream Terminal Operations That Return No Value

- The forEach() & forEachOrdered() terminal operations return no value at all
  - i.e., they only have side-effects!

```
void runForEach() {
  ...

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

Performs the designated action on each element of this stream

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#forEach

Comparing forEach() with forEachOrdered()

Comparing forEach() with forEachOrdered()

- Use forEachOrdered() if presenting the results in "encounter order" is important

```
void runForEach() {
   ...

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

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#forEachOrdered

Comparing forEach() with forEachOrdered()

- Use forEachOrdered() if presenting the results in "encounter order" is important

"Encounter order" is simply the order in which a Stream encounters data

```
void runForEach() {
   ...

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

See www.baeldung.com/java-stream-ordering

Comparing forEach() with forEachOrdered()

- Use forEachOrdered() if presenting the results in "encounter order" is important
  - forEach() does not preserve encounter order, whereas forEachOrdered() does

forEachOrdered() is only really relevant for parallel streams..

```
void runForEach() {
   ...

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

See docs.oracle.com/javase/8/docs/api/java/util/stream/BaseStream.html#parallel

Common Traps & Pitfalls with These Terminal Operations

Common Traps & Pitfalls with These Terminal Operations

- forEach() can be used to assign values to local objects

```
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);
...
```

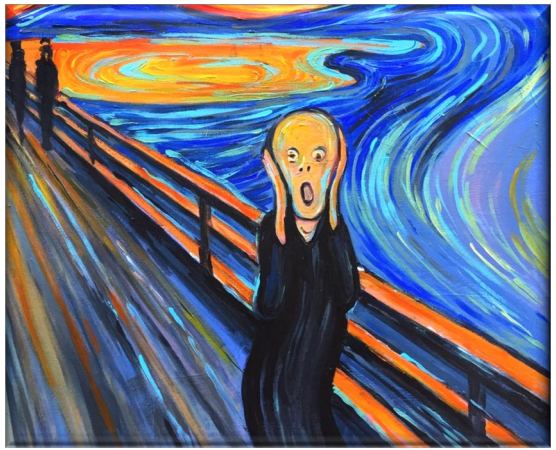
This code "works" since a method reference passed to forEach() can have side-effects by design..

See www.logicbig.com/tutorials/core-java-tutorial/java-util-stream/side-effects

Common Traps & Pitfalls with These Terminal Operations

- However, using forEach() to assign local objects it tricky!

i.e., programmers must remember to initialize the results object!



```
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);
  ...
```

Common Traps & Pitfalls with These Terminal Operations

- However, using forEach() to assign local objects it tricky!

Likewise, using forEach() with side-effects in a parallel stream can incur nasty race conditions!!

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

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

See docs.oracle.com/javase/tutorial/collections/streams/parallelism.html#side_effects

Common Traps & Pitfalls with These Terminal Operations

- However, using forEach() to assign local objects it tricky!

ConcurrentLinkedQueue could be used here, but it's still error-prone & inefficient

```
void runForEach() {
  Queue<String> results = new
    ConcurrentLinkedQueue<>();

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

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentLinkedQueue.html

Common Traps & Pitfalls with These Terminal Operations

- It's usually much better to apply the collect() terminal operation in these cases!

collect() handles all the allocation & concurrent processing seamlessly

```
void runForEach() {
  List<String> results = Stream
    .of("horatio", "laertes",
        "Hamlet", ...)
    .parallel()
    .filter(s -> toLowerCase
           (s.charAt(0)) == 'h')
    .map(this::capitalize)
    .sorted()
    .collect(toList());
...
```

See upcoming lesson on "*The Java Streams collect() Terminal Operation*"

End of the Java Streams forEach() & forEachOrdered() Terminal Operation