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.

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
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*

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
The forEach() & forEachOrdered() 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 howtodoinjava.com/java8/foreach-method-example
Stream Terminal Operations That Return No Value

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

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

```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/douglasraigschmidt/LiveLessons/tree/master/Java8/ex12](https://github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex12)
Stream Terminal Operations That Return No Value

- The forEach() & forEachOrdered() 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);
    ...
}
```

Create & process a stream consisting of characters from the play “Hamlet”
The `forEach()` & `forEachOrdered()` 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 [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)
Comparing forEach() with forEachOrdered()
Comparing forEach() with forEachOrdered()

- Use forEachOrdered() if presenting the results in “encounter order” is important.

```java
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

void runForEach() {
  ...

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

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

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

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

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

forEachOrdered() is only really relevant for parallel streams.

See [docs.oracle.com/javase/8/docs/api/java/util/stream/BaseStream.html#parallel](http://docs.oracle.com/javase/8/docs/api/java/util/stream/BaseStream.html#parallel)
Common Traps & Pitfalls with These Terminal Operations
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.
However, using `forEach()` to assign local objects it tricky!

```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);
    ...
```

*i.e., programmers must remember to initialize the results object!*
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);
    ...
• However, using `forEach()` to assign local objects it tricky!

    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);
        ...
    }

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

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentLinkedQueue.html
It’s usually much better to apply the collect() terminal operation in these cases!

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

*collect() handles all the allocation & concurrent processing seamlessly*
End of the Java Streams
forEach() & forEachOrdered()
Terminal Operation