Java Streams: Avoiding Common Programming Mistakes

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

• Know how to apply sequential streams to the SearchStreamGang program
• Recognize how a Spliterator is used in SearchWithSequentialStreams
• Understand the pros & cons of the SearchWithSequentialStreams class
• Learn how to avoid common streams programming mistakes

See blog.jooq.org/2014/06/13/java-8-friday-10-subtle-mistakes-when-using-the-streams-api
Learning Objectives in this Part of the Lesson

- Know how to apply sequential streams to the SearchStreamGang program
- Recognize how a Spliterator is used in SearchWithSequentialStreams
- Understand the pros & cons of the SearchWithSequentialStreams class
- Learn how to avoid common streams programming mistakes

We discuss several examples in this lesson, including SearchWithSequentialStreams.

See `streamgangs/SearchWithSequentialStreams.java`
Avoiding Common Streams Programming Mistakes
Avoiding Common Streams Programming Mistakes

• Don’t forget the terminal operation!

```java
List<CharSequence> input =
    getInput();

Stream<List<SearchResults>> input
    .stream()
    .map(this::processInput);
```

This is an all-to-common beginner mistake..

See `streamgangs/SearchWithSequentialStreams.java`
Avoiding Common Streams Programming Mistakes

- Only traverse a stream once

```
Aggregate operation (behavior f)

Output f(x)
```

```
Aggregate operation (behavior g)

Output g(f(x))
```

```
Aggregate operation (behavior h)

Output h(g(f(x)))
```
Avoiding Common Streams Programming Mistakes

• Only traverse a stream once

List<CharSequence> input =
   getInput();

Stream<List<SearchResults>> s = input
   .stream()
   .map(this::processInput);

s.forEach(System.out::println);

Duplicate calls are invalid!
 Avoiding Common Streams Programming Mistakes

- Only traverse a stream once

```java
List<CharSequence> input = getInput();
Stream<List<SearchResults>> s = input.stream()
    .map(this::processInput);

s.forEach(System.out::println);
s.forEach(System.out::println);
```

Throws `java.lang.IllegalStateException`

See [docs.oracle.com/javase/8/docs/api/java/lang/IllegalStateException.html](docs.oracle.com/javase/8/docs/api/java/lang/IllegalStateException.html)
Avoiding Common Streams Programming Mistakes

- Only traverse a stream once

To traverse a stream again you need to get a new stream from the data source

Aggregate operation (behavior f)

Output $f(x)$

Aggregate operation (behavior $g$)

Output $g(f(x))$

Aggregate operation (behavior $h$)

Output $h(g(f(x)))$
Avoiding Common Streams Programming Mistakes

• Don’t modify the backing collection of a stream

```java
List<Integer> list = IntStream
    .range(0, 10)
    .boxed()
    .collect(toList());

list
    .stream()
    .peek(list::remove)
    .forEach(System.out::println);
```

See [github.com/douglasraignschmidt/LiveLessons/tree/master/Java8/ex11](https://github.com/douglasraignschmidt/LiveLessons/tree/master/Java8/ex11)
Avoiding Common Streams Programming Mistakes

- Don’t modify the backing collection of a stream

```java
List<Integer> list = IntStream
    .range(0, 10)
    .boxed()
    .collect(toList());
```

Create a list of ten integers in range 0..9

```java
list
    .stream()
    .peek(list::remove)
    .forEach(System.out::println);
```
Avoiding Common Streams Programming Mistakes

- Don’t modify the backing collection of a stream

```java
List<Integer> list = IntStream
    .range(0, 10)
    .boxed()
    .collect(toList());

list
    .stream()
    .peek(list::remove)
    .forEach(System.out::println);
```

If a non-concurrent collection is modified while it’s being operated on the results will be chao & insanity!!

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#peek](docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#peek)
Avoiding Common Streams Programming Mistakes

- Don’t modify the backing collection of a stream

```java
List<Integer> list = IntStream
    .range(0, 10)
    .boxed()
    .collect(toList());

list
    .stream()
    .peek(list::remove)
    .forEach(System.out::println);
```

Modifying a list while it’s been iterated/spliterated through will yield weird results!

See [docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html#NonInterference](docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html#NonInterference)
Avoiding Common Streams Programming Mistakes

- Remember that a stream holds no non-transient storage

```
Aggregate operation (behavior f)

Output f(x)

Aggregate operation (behavior g)

Output g(f(x))

Aggregate operation (behavior h)

Output h(g(f(x)))
```
Avoiding Common Streams Programming Mistakes

- Remember that a stream holds no non-transient storage

Apps are responsible for persisting any data that must be preserved

See dzone.com/articles/database-crud-operations-in-java-8-streams
End of Java Streams: Avoiding Common Programming Mistakes