Overview of Java 8 Streams (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 the structure & functionality of Java 8 streams

\[ \text{Input } x \]

\[ \text{Aggregate operation (behavior } f) \]

\[ \text{Output } f(x) \]

\[ \text{Aggregate operation (behavior } g) \]

\[ \text{Output } g(f(x)) \]

\[ \text{Aggregate operation (behavior } h) \]

\[ \text{Output } h(g(f(x))) \]
Learning Objectives in this Part of the Lesson

- Understand the structure & functionality of Java 8 streams, e.g.,

- Fundamentals of streams

\[ \text{Input } x \rightarrow \text{Aggregate operation (behavior } f) \rightarrow \text{Output } f(x) \rightarrow \text{Aggregate operation (behavior } g) \rightarrow \text{Output } g(f(x)) \rightarrow \text{Aggregate operation (behavior } h) \rightarrow \text{Output } h(g(f(x))) \]
Learning Objectives in this Part of the Lesson

- Understand the structure & functionality of Java 8 streams, e.g.,
- Fundamentals of streams
- We’ll use an example program to illustrate key concepts

See [github.com/douglascraighschmidt/LiveLessons/tree/master/Java8/ex12](https://github.com/douglascraighschmidt/LiveLessons/tree/master/Java8/ex12)
Overview of Java 8 Streams
Overview of Java 8 Streams

Java 8 streams are a framework in the Java class library that provides several key benefits to programs.

See docs.oracle.com/javase/tutorial/collections/streams
Overview of Java 8 Streams

- Java 8 streams are a framework in the Java class library that provides several key benefits to programs, e.g.
- Manipulate flows of data declaratively via function composition

```
This stream expresses what operations to perform, not how to perform them
```

Overview of Java 8 Streams

- Java 8 streams are a framework in the Java class library that provides several key benefits to programs, e.g.
  - Manipulate flows of data declaratively via function composition
  - Enable transparent parallelization without the need to write any multi-threaded code

The data elements in this stream are automatically mapped to processor cores

See docs.oracle.com/javase/tutorial/collections/stream/parallelism.html
Overview of Java 8 Streams

- A stream is a pipeline of aggregate operations that process a sequence of elements (aka, “values” or “data”)

\[
\begin{align*}
\text{Input } x & \\
\text{Aggregate operation } (\text{behavior } f) & \\
\text{Output } f(x) & \\
\text{Aggregate operation } (\text{behavior } g) & \\
\text{Output } g(f(x)) & \\
\text{Aggregate operation } (\text{behavior } h) &
\end{align*}
\]

See [www.tutorialspoint.com/java8/java8_streams.htm](http://www.tutorialspoint.com/java8/java8_streams.htm)
Overview of Java 8 Streams

• A stream is a pipeline of aggregate operations that process a sequence of elements (aka, “values” or “data”)

An aggregate operation is a higher-order function that applies a “behavior” parameter to every element in a stream

Input $x$

Aggregate operation (behavior $f$)

Output $f(x)$

Aggregate operation (behavior $g$)

Output $g(f(x))$

Aggregate operation (behavior $h$)

See docs.oracle.com/javase/tutorial/collectionsstreams
Overview of Java 8 Streams

- A stream is a pipeline of aggregate operations that process a sequence of elements (aka, “values” or “data”)

Input $x$

Aggregate operation (behavior $f$)

Output $f(x)$

Aggregate operation (behavior $g$)

Output $g(f(x))$

Aggregate operation (behavior $h$)

A stream is conceptually unbounded, though it’s often bounded by practical constraints
A stream is a pipeline of aggregate operations that process a sequence of elements (aka, “values” or “data”)

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](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex12)
Overview of Java 8 Streams

• A stream is created via a factory method

Stream
  .of("horatio",
      "laertes",
      "Hamlet",
      ...

See en.wikipedia.org/wiki/Factory_method_pattern
Overview of Java 8 Streams

- A stream is created via a factory method

Stream
```
of("horatio",
  "laertes",
  "Hamlet",
  ...
)
```

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#of
Overview of Java 8 Streams

• A stream is created via a factory method

```java
List<String> l1 = ...;
List<String> l2 = ...;
List<String> l3 = ...;

Stream
    .of(l1, l2, l3)
    .flatMap(List::stream)
    ...
    .forEach(System.out::println);
```

`of()` is flexible, especially when combined with other aggregate operations

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex12](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex12)
Overview of Java 8 Streams

- A stream is created via a factory method

- There are many other factory methods that create streams
Overview of Java 8 Streams

- An aggregate operation performs a *behavior* on each element in a stream

A behavior is implemented by a lambda expression or method reference.
Overview of Java 8 Streams

- An aggregate operation performs a behavior on each element in a stream

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

Method reference

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex12](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex12)
Overview of Java 8 Streams

- An aggregate operation performs a *behavior* on each element in a stream.
- Ideally, a behavior’s output in a stream depends only on its input arguments.

See [en.wikipedia.org/wiki/Side_effect_(computer_science)](en.wikipedia.org/wiki/Side_effect_(computer_science))
Overview of Java 8 Streams

- An aggregate operation performs a *behavior* on each element in a stream.
- Ideally, a behavior’s output in a stream depends only on its input arguments.

---

```java
String capitalize(String s) {
    if (s.length() == 0)
        return s;
    return s.substring(0, 1).toUpperCase()
       + s.substring(1).toLowerCase();
}
```

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex12](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex12)
Overview of Java 8 Streams

• An aggregate operation performs a behavior on each element in a stream
• Ideally, a behavior’s output in a stream depends only on its input arguments
• Behaviors with side-effects likely incur race conditions in parallel streams

Race conditions arise in software when an application depends on the sequence or timing of threads for it to operate properly

See en.wikipedia.org/wiki/Race_condition#Software
Overview of Java 8 Streams

• An aggregate operation performs a *behavior* on each element in a stream

• Ideally, a behavior’s output in a stream depends only on its input arguments

• Behaviors with side-effects likely incur race conditions in parallel streams

In Java *you* must avoid race conditions, i.e., the compiler & JVM won’t save you..

*Only you can prevent race conditions!*
Overview of Java 8 Streams

- Streams enhance flexibility by forming a “processing pipeline” that chains multiple aggregate operations together

\[
\text{Input } x \rightarrow \text{Aggregate operation (behavior } f) \rightarrow \text{Output } f(x) \rightarrow \text{Aggregate operation (behavior } g) \rightarrow \text{Output } g(f(x)) \rightarrow \text{Aggregate operation (behavior } h) \rightarrow \text{Output } h(g(f(x)))
\]

See [en.wikipedia.org/wiki/Pipeline_(software)](en.wikipedia.org/wiki/Pipeline_(software))
Streams enhance flexibility by forming a “processing pipeline” that chains multiple aggregate operations together.

Each aggregate operation in the pipeline can filter and/or transform the stream.
Overview of Java 8 Streams

- A stream holds no non-transient storage

```
Input x

Aggregate operation (behavior f)

Output f(x)

Aggregate operation (behavior g)

Output g(f(x))

Aggregate operation (behavior h)

Output h(g(f(x)))
```
Overview of Java 8 Streams

• Every stream works very similarly
Overview of Java 8 Streams

- Every stream works very similarly
- Starts with a source of data

```
Stream.of("horatio",
    "laertes",
    "Hamlet",
    ...
)
...```

e.g., a Java array, collection, generator function, or input channel
Overview of Java 8 Streams

- Every stream works very similarly
- Starts with a source of data

```java
List<String> characters =
    Arrays.asList("horatio",
                   "laertes",
                   "Hamlet",
                   ...);

characters.stream()
...
Overview of Java 8 Streams

- Every stream works very similarly
- Starts with a source of data
- Processes data through a pipeline of intermediate operations that each map an input stream to an output stream

Examples of intermediate operations include filter(), map(), & flatMap()

Stream
  .of("horatio", "laertes", "Hamlet", ...)
  .filter(s -> toLowerCase(s.charAt(0)) == 'h')
  .map(this::capitalize)
  .sorted()
...
Every stream works very similarly

- Starts with a source of data
- Processes data through a pipeline of intermediate operations that each map an input stream to an output stream
- Finishes with a terminal operation that yields a non-stream result

```
... .filter(s -> toLowerCase(s.charAt(0)) == 'h') .map(this::capitalize) .sorted() .forEach(System.out::println);
```
Overview of Java 8 Streams

• Every stream works very similarly
  • Starts with a source of data
  • Processes data through a pipeline of intermediate operations that each map an input stream to an output stream
  • Finishes with a terminal operation that yields a non-stream result

```java
... .filter(s -> toLowerCase(s.charAt(0)) == 'h') .map(this::capitalize) .sorted() .forEach(System.out::println);
```

A terminal operation triggers processing of intermediate operations in a stream
Overview of Java 8 Streams

• Every stream works very similarly
  • Starts with a source of data
  • Processes data through a pipeline of intermediate operations that each map an input stream to an output stream
  • Finishes with a terminal operation that yields a non-stream result

Each stream *must* have one (& only one) terminal operation
Overview of Java 8 Streams

- Each aggregate operation in a stream runs its behavior sequentially by default.
Overview of Java 8 Streams

• Each aggregate operation in a stream runs its behavior sequentially by default
  • i.e., one at a time in a single thread

We’ll cover sequential streams first

See docs.oracle.com/javase/tutorial/collectionsStreams
Overview of Java 8 Streams

• A Java 8 parallel stream splits its elements into multiple chunks & uses a common fork-join pool to process the chunks independently.

See docs.oracle.com/javase/tutorial/collections/streams/parallelism.html
End of Overview of Java
8 Streams (Part 1)