Understanding Java Streams

Common Aggregate 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 Java streams structure & functionality, e.g.
  - Fundamentals of streams
  - Three streams phases
  - Operations that create a stream
  - Aggregate operations in a stream

![Diagram showing stream source, aggregate operations, and output]

- Stream source
  - Input $x$
  - Aggregate operation (behavior $f$)
    - Output $f(x)$
  - Aggregate operation (behavior $g$)
    - Output $g(f(x))$
  - Aggregate operation (behavior $h$)
Java Streams
Aggregate Operations
Java Streams Aggregate Operations

- An aggregate operation performs a *behavior* on elements in a stream

A behavior is implemented by a lambda expression or method reference corresponding to a functional interface

See blog.indrek.io/articles/java-8-behavior-parameterization
An aggregate operation performs a behavior on elements in a stream.

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

- An aggregate operation performs a *behavior* on elements in a stream
- Some aggregate operations perform behaviors on all elements in a stream
Java Streams Aggregate Operations

• An aggregate operation performs a *behavior* on elements in a stream
  • Some aggregate operations perform behaviors on all elements in a stream
  • Other aggregate operations perform behaviors on some elements in a stream
Java Streams Aggregate Operations

- Aggregate operations can be composed to form a pipeline of processing phases

See [en.wikipedia.org/wiki/Pipeline_(software)](en.wikipedia.org/wiki/Pipeline_(software))
Java Streams Aggregate Operations

- Aggregate operations can be composed to form a pipeline of processing phases

The output of one aggregate operation can be input into the next one in the stream.
Java Streams Aggregate Operations

- Aggregate operations can be composed to form a pipeline of processing phases.

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

Java streams supports pipelining of aggregate operations via “fluent interfaces”.

See [en.wikipedia.org/wiki/Fluent_interface](en.wikipedia.org/wiki/Fluent_interface)
Aggregate operations can be composed to form a pipeline of processing phases.

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

A factory method that creates a stream from an array of elements.

See upcoming lessons on “Stream Creation Operations”
Java Streams Aggregate Operations

- Aggregate operations can be composed to form a pipeline of processing phases

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

An aggregate operation that returns a stream containing only elements matching the predicate

See upcoming lessons on “Stream Intermediate Operations”
Java Streams Aggregate Operations

- Aggregate operations can be composed to form a pipeline of processing phases

An aggregate operation that returns a stream consisting of results of applying a function to elements of this stream

See upcoming lessons on “Stream Intermediate Operations"
Java Streams Aggregate Operations

- Aggregate operations can be composed to form a pipeline of processing phases.

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

An aggregate operation that returns a stream consisting of results sorted in the natural order.

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#sorted](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#sorted)
Java Streams Aggregate Operations

- Aggregate operations can be composed to form a pipeline of processing phases

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

An aggregate operation that performs an action on each element of the stream

See upcoming lessons on “Stream Terminal Operations"
Java Streams Aggregate Operations

- Java streams iterate internally (& invisibly) between aggregate operations

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

Internal iteration enhances opportunities for transparent optimization & incurs fewer accidental complexities

In contrast, collections are iterated explicitly using loops and/or iterators.

```java
List<String> l = new LinkedList<>(List.of("horatio", "laertes", "Hamlet", ...));

for (int i = 0; i < l.size();)
    if (toLowerCase(l.get(i).charAt(0)) != 'h')
        l.remove(i);
    else {
        l.set(i, capitalize(l.get(i))); i++;
    }

Collections.sort(l);

for (String s : l) System.out.println(s);
```

Explicit control constructs yield more opportunities for accidental complexities & are hard to optimize.

See upcoming lessons on “External vs. Internal Iterators in Java”
End of Understanding Java Streams Common Aggregate Operations