Understand Java Streams

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 the structure & functionality of stream aggregate operations

  Input $x$

  **Intermediate operation (Behavior $f$)**

  Output $f(x)$

  **Intermediate operation (Behavior $g$)**

  Output $g(f(x))$

  **Terminal operation (Behavior $h$)**
Overview of Stream Aggregate Operations
Overview of Stream Aggregate Operations

• An aggregate operation is a higher-order function that applies a “behavior” on elements in a stream

A “higher order function” is a function that is passed a function as a param

See [en.wikipedia.org/wiki/Higher-order_function](http://en.wikipedia.org/wiki/Higher-order_function)
Overview of Stream Aggregate Operations

- An aggregate operation is a higher-order function that applies a “behavior” on elements in a stream.

```
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)))
```

The behavior can be a lambda or method reference to a function, predicate, consumer, supplier, etc.

Overview of Stream Aggregate Operations

- Aggregate operations form a declarative pipeline that emphasizes the “what” & deemphasizes the “how”

See blog.jooq.org/2015/09/17/comparing-imperative-and-functional-algorithms-in-java-8
Overview of Stream Aggregate Operations

• There are two types of aggregate operations
There are two types of aggregate operations

- **Intermediate operations**
  - Process elements in their input stream & yield an output stream
  - e.g., filter(), map(), flatMap(), takeWhile(), dropWhile(), etc.

There are two types of aggregate operations:

- **Intermediate operations**
  - Process elements in their input stream & yield an output stream
  - e.g., filter(), map(), flatMap(), takeWhile(), dropWhile(), etc.

- Terminal operation (Behavior h)

```java
long HamletCharacters = Stream
    .of("horatio", "laertes", "Hamlet", ...)
    .count();
```
Overview of Stream Aggregate Operations

- There are two types of aggregate operations
  - Intermediate operations
    - Process elements in their input stream & yield an output stream
      - e.g., filter(), map(), flatMap(), takeWhile(), dropWhile(), etc.

*The semantics of count() are now weird.*

```java
long HamletCharacters = Stream.of("horatio", "laertes", "Hamlet", ...)
  .peek(System.out::print)
  .count();
```

Overview of Stream Aggregate Operations

- There are two types of aggregate operations
  - **Intermediate operations**
    - Process elements in their input stream & yield an output stream
    - e.g., filter(), map(), flatMap(), takeWhile(), dropWhile(), etc.

```java
long HamletCharacters = Stream
  .of("horatio", "laertes", "Hamlet", ...)
  .count();
```

Newer versions of Java optimize streams containing no intermediate operations.

Input $x$
Overview of Stream Aggregate Operations

- There are two types of aggregate operations
  - **Intermediate operations**
    - Process elements in their input stream & yield an output stream
    - Intermediate operations can be further classified via several dimensions

<table>
<thead>
<tr>
<th></th>
<th>Run-to-completion</th>
<th>Short-circuiting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Stateful</strong></td>
<td>distinct(), skip(), sorted()</td>
<td>limit(), takeWhile(), dropWhile(), etc.</td>
</tr>
<tr>
<td><strong>Stateless</strong></td>
<td>filter(), map(), flatMap(), etc.</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Overview of Stream Aggregate Operations

- There are two types of aggregate operations

  - **Intermediate operations**
    - Process elements in their input stream & yield an output stream
    - Intermediate operations can be further classified via several dimensions, e.g.
      - **Stateful**
        - Store info from a prior invocation for use in a future invocation

<table>
<thead>
<tr>
<th></th>
<th>Run-to-completion</th>
<th>Short-Circuiting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Stateful</strong></td>
<td>distinct(), skip(), sorted()</td>
<td>limit(), takeWhile(), dropWhile(), etc.</td>
</tr>
<tr>
<td><strong>Stateless</strong></td>
<td>filter(), map(), flatMap(), etc.</td>
<td>N/A</td>
</tr>
</tbody>
</table>

See [stuartmarks.wordpress.com/2015/01/09/writing-stateful-stream-operations](stuartmarks.wordpress.com/2015/01/09/writing-stateful-stream-operations)
There are two types of aggregate operations:

**Intermediate operations**
- Process elements in their input stream & yield an output stream
- Intermediate operations can be further classified via several dimensions, e.g.
  - **Stateful**
  - **Stateless**
    - Do not store info from any prior invocations

<table>
<thead>
<tr>
<th></th>
<th>Run-to-completion</th>
<th>Short-Circuiting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Stateful</strong></td>
<td>distinct(), skip(), sorted()</td>
<td>limit(), takeWhile(), dropWhile(), etc.</td>
</tr>
<tr>
<td><strong>Stateless</strong></td>
<td>filter(), map(), flatMap(), etc.</td>
<td>N/A</td>
</tr>
</tbody>
</table>

See [javapapers.com/java/java-stream-api](http://javapapers.com/java/java-stream-api)
There are two types of aggregate operations

- **Intermediate operations**
  - Process elements in their input stream & yield an output stream
  - Intermediate operations can be further classified via several dimensions, e.g.
    - **Stateful**
    - **Stateless**
      - Do not store info from any prior invocations

<table>
<thead>
<tr>
<th></th>
<th>Run-to-completion</th>
<th>Short-Circuiting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Stateful</strong></td>
<td>distinct(), skip(), sorted()</td>
<td>limit(), takeWhile(), dropWhile(), etc.</td>
</tr>
<tr>
<td><strong>Stateless</strong></td>
<td>filter(), map(), flatMap(), etc.</td>
<td>N/A</td>
</tr>
</tbody>
</table>

*Stateless operations often require significantly fewer processing & memory resources than stateful operations!*

See [automationrhapsody.com/java-8-features-stream-api-explained](http://automationrhapsody.com/java-8-features-stream-api-explained)
Overview of Stream Aggregate Operations

- There are two types of aggregate operations

- **Intermediate operations**
  - Process elements in their input stream & yield an output stream
  - Intermediate operations can be further classified via several dimensions, e.g.
    - **Stateful**
    - **Stateless**
    - **Run-to-completion**
      - Process all elements in the input stream

<table>
<thead>
<tr>
<th></th>
<th>Run-to-completion</th>
<th>Short-Circuiting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Stateful</strong></td>
<td>distinct(), skip(), sorted()</td>
<td>limit(), takeWhile(), dropWhile(), etc.</td>
</tr>
<tr>
<td><strong>Stateless</strong></td>
<td>filter(), map(), flatMap(), etc.</td>
<td>N/A</td>
</tr>
</tbody>
</table>

See [en.wikipedia.org/wiki/Run_to_completion_scheduling](en.wikipedia.org/wiki/Run_to_completion_scheduling)
Overview of Stream Aggregate Operations

- There are two types of aggregate operations
- **Intermediate operations**
  - Process elements in their input stream & yield an output stream
  - Intermediate operations can be further classified via several dimensions, e.g.
    - **Stateful**
    - **Stateless**
    - Run-to-completion
    - Short-circuiting
      - Make stream operate on a reduced size

<table>
<thead>
<tr>
<th></th>
<th>Run-to-completion</th>
<th>Short-Circuiting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stateful</td>
<td>distinct(), skip(), sorted()</td>
<td>limit(), takeWhile(), dropWhile(), etc.</td>
</tr>
<tr>
<td>Stateless</td>
<td>filter(), map(), flatMap(), etc.</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Overview of Stream Aggregate Operations

- There are two types of aggregate operations
  - Intermediate operations
  - Terminal operations
    - Trigger intermediate operations & produce a non-stream result
    - e.g., forEach(), reduce(), collect(), findAny(), etc.

A stream must have one (and only one) terminal operation

See www.leveluplunch.com/java/examples/stream-terminal-operations-example
Overview of Stream Aggregate Operations

- There are two types of aggregate operations
  - **Intermediate operations**
  - **Terminal operations**
    - Trigger intermediate operations & produce a non-stream result
    - Terminal operations can also be classified via several dimensions

<table>
<thead>
<tr>
<th>Operation Type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run-to-completion</td>
<td>reduce(), collect(), forEach(), etc.</td>
</tr>
<tr>
<td>Short-circuiting</td>
<td>allMatch(), anyMatch(), findAny(), findFirst(), noneMatch()</td>
</tr>
</tbody>
</table>
There are two types of aggregate operations:

- **Intermediate operations**

- **Terminal operations**
  - Trigger intermediate operations & produce a non-stream result.
  - Terminal operations can also be classified via several dimensions, e.g.
    - Run-to-completion
      - Terminate only after processing all elements in the stream.

### Overview of Stream Aggregate Operations

<table>
<thead>
<tr>
<th>Operation Type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run-to-completion</td>
<td><code>reduce()</code>, <code>collect()</code>, <code>forEach()</code> etc.</td>
</tr>
<tr>
<td>Short-circuiting</td>
<td><code>allMatch()</code>, <code>anyMatch()</code>, <code>findAny()</code>, <code>findFirst()</code>, <code>noneMatch()</code></td>
</tr>
</tbody>
</table>
There are two types of aggregate operations:

- **Intermediate operations**
  - Trigger intermediate operations & produce a non-stream result

- **Terminal operations**
  - Terminal operations can also be classified via several dimensions, e.g.
    - Run-to-completion
    - Short-circuiting
      - May cause a stream to terminate before processing all values

### Overview of Stream Aggregate Operations

<table>
<thead>
<tr>
<th>Operation Type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run-to-completion</td>
<td>reduce(), collect(), forEach(), etc.</td>
</tr>
<tr>
<td>Short-circuiting</td>
<td>allMatch(), anyMatch(), findAny(), findFirst(), noneMatch()</td>
</tr>
</tbody>
</table>
End of Understand Java Streams Aggregate Operations