Overview of Java 8 Streams (Part 2)

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, e.g.,
  - Fundamentals of streams
  - Common stream aggregate operations

Both intermediate & terminal operations

Input $x$

Intermediate operation (Behavior $f$)

Output $f(x)$

Intermediate operation (Behavior $g$)

Output $g(f(x))$

Terminal operation (Behavior $h$)
Learning Objectives in this Part of the Lesson

- Understand the structure & functionality of Java 8 streams, e.g.,
  - Fundamentals of streams
  - Common stream aggregate operations
    - These operations apply to both sequential & parallel streams

![Diagram of stream operations]

Input $x$ → $f(x)$ → $g(f(x))$ → Terminal operation (Behavior $h$)
Learning Objectives in this Part of the Lesson

- Understand the structure & functionality of Java 8 streams, e.g.,
  - Fundamentals of streams
  - Common stream aggregate operations
  - These operations apply to both sequential & parallel streams

Being a good streams programmer makes you a better parallel streams programmer
Learning Objectives in this Part of the Lesson

- Understand the structure & functionality of Java 8 streams, e.g.,
  - Fundamentals of streams
  - Common stream aggregate operations
  - These operations apply to both sequential & parallel streams

We’ll use a simple sequential stream example to explain common Java 8 aggregate operations

See [github.com/douglascraigschmidt/LiveLessons/tree/master/SimpleSearchStream](https://github.com/douglascraigschmidt/LiveLessons/tree/master/SimpleSearchStream)
Overview of SimpleSearch
Stream Example
Overview of SimpleSearchStream Example

- This example finds words in an input string

**Input String to Search**

Let's start at the very beginning...

**Search Words**

"do", "re", "mi", "fa", "so", "la", "ti", "do"

- stream()
- map(this::searchForWord)
- filter(not(SearchResults::isEmpty))
- collect(toList())

We'll use this example to explain Java 8 aggregate operations throughout this part of the lesson

See [github.com/douglascraigschmidt/LiveLessons/tree/master/SimpleSearchStream](https://github.com/douglascraigschmidt/LiveLessons/tree/master/SimpleSearchStream)
Overview of SimpleSearchStream Example

- This example finds words in an input string

**Input String to Search**

Let's start at the very beginning...

Starting SimpleSearchStream

Word "Re" matched at index: 131, 141, 151, 202, 212, 222, 979, 1025, 1219, 1259, 1278, 1300, 1351, 1370, 1835, 1875, 1899, 1939, 2266, 2295


Word "La" matched at index: 234, 417, 658, 886, 991, 1207, 1247, 1269, 1291, 1339, 1361, 1742, 1847, 1863, 1909, 1949, 2161, 2254, 2276, 2283...

Ending SimpleSearchStream

**Search Words**

"do", "re", "mi", "fa", "so", "la", "ti", "do"

**Operation**

`stream()`

`map(this::searchForWord)`

`filter(not(SearchResults::isEmpty))`

`collect(toList())`

45,000+ phrases

"do", "re", "mi", "fa", "so", "la", "ti", "do"

See [github.com/douglascraigschmidt/LiveLessons/tree/master/SimpleSearchStream](https://github.com/douglascraigschmidt/LiveLessons/tree/master/SimpleSearchStream)
Overview of SimpleSearchStream Example

• This example finds words in an input string

Input a list of words to find

List
<List>

Search Words
"do", "re", "mi", "fa", "so", "la", "ti", "do"

stream()
Overview of SimpleSearchStream Example

- This example finds words in an input string

Convert collection to a (sequential) stream
Overview of SimpleSearchStream Example

- This example finds words in an input string

Output a stream of words to find

List
<String>

Stream
<String>

Search Words
"do", "re", "mi", "fa", "so", "la", "ti", "do"

stream()
Overview of SimpleSearchStream Example

- This example finds words in an input string

Input a stream of words to find

List
<String>

Stream
<String>

Search Words
"do", "re", "mi", "fa", "so", "la", "ti", "do"

map(this::searchForWord)

stream()
Overview of SimpleSearchStream Example

- This example finds words in an input string

```
List <String>
Stream <String>
```

Search Words
"do", "re", "mi", "fa", "so", "la", "ti", "do"

```
stream()
map(this::searchForWord)
```

Search for the word in the input string
Overview of SimpleSearchStream Example

- This example finds words in an input string

Output a stream of search results

List

<String>

Stream

<String>

Stream

<SearchResults>

Search Words

"do", "re", "mi", "fa", "so", "la", "ti", "do"

map(this::searchForWord)

stream()

SearchResults stores # of times a word appeared in the input string
Overview of SimpleSearchStream Example

- This example finds words in an input string

```
stream()
.map(this::searchForWord)
.filter(not(SearchResults::isEmpty))
```

- **Input a stream of search results**

- **Search Words**
  - "do", "re", "mi", "fa", "so", "la", "ti", "do"

- **List** `<String>`
  - ...<br />

- **Stream** `<String>`
  - ...<br />

- **Stream** `<SearchResults>`
  - ...
Overview of SimpleSearchStream Example

- This example finds words in an input string

List
<String>

Stream
<String>

Stream
<SearchResults>

Remove empty search results from the stream

Search Words
"do", "re", "mi", "fa", "so", "la", "ti", "do"

stream()

map(this::searchForWord)

filter(not(SearchResults::isEmpty))
Overview of SimpleSearchStream Example

- This example finds words in an input string

Output a stream of non-empty search results

List

Stream

Stream

Stream

Search Words

"do", "re", "mi", "fa", "so", "la", "ti", "do"

stream()

map(this::searchForWord)

filter(not(SearchResults::isEmpty))
Overview of SimpleSearchStream Example

- This example finds words in an input string

**Input a stream of non-empty search results**

Stream $\langle \text{SearchResults} \rangle$ → Stream $\langle \text{String} \rangle$ → Stream $\langle \text{SearchResults} \rangle$ → ... → Stream $\langle \text{String} \rangle$ → Stream $\langle \text{SearchResults} \rangle$ → ... → Stream $\langle \text{SearchResults} \rangle$

**Search Words**

"do", "re", "mi", "fa", "so", "la", "ti", "do"

1. stream()
2. map(this::searchForWord)
3. filter(not(SearchResults::isEmpty))
4. collect(toList())

45,000+ phrases "do", "re", "mi", "fa", "so", "la", "ti", "do"
Overview of SimpleSearchStream Example

- This example finds words in an input string

List

```
Stream <String>
```

```
Stream <SearchResults>
```

```
Stream <SearchResults>
```

Search Words

```
"do", "re", "mi", "fa", "so", "la", "ti", "do"
```

Stream

```
stream()
```

```
map(this::searchForWord)
```

```
filter(not(SearchResults::isEmpty))
```

```
collect(toList())
```

Trigger intermediate operation processing
Overview of SimpleSearchStream Example

- This example finds words in an input string

```
List<String>
Stream<String>
Stream<SearchResults>
Stream<SearchResults>
List<SearchResults>
```

Search Words

```
"do", "re", "mi", "fa", "so", "la", "ti", "do"
```

Return a list of search results

```
stream()
map(this::searchForWord)
filter(not(SearchResults::isEmpty))
collect(toList())
```
The "physical" processing of a stream differs from the "logical" model, i.e., each element is "pulled" from the source through each aggregate operation.

Overview of SimpleSearchStream Example

```
Stream<SearchResults> stream()
    .map(this::searchForWord)
    .filter(not(SearchResults::isEmpty))
    .collect(toList())
```

List<SearchResults> 45,000+ phrases
"do", "re", "mi", "fa", "so", "la", "ti", "do"

See www.ibm.com/developerworks/library/j-java-streams-3-brian-goetz
Overview of Common Stream Aggregate Operations
Overview of Common 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 $h(g(f(x)))$

See [en.wikipedia.org/wiki/Higher-order_function](en.wikipedia.org/wiki/Higher-order_function)
Overview of Common 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.

See en.wikipedia.org/wiki/Higher-order_function
Overview of Common Stream Aggregate Operations

- Aggregate operations focus on “what” (declarative), not “how” (imperative)

- 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 Common Stream Aggregate Operations

- There are two types of aggregate operations

1. Input $x$
2. Intermediate operation (Behavior $f$)
3. Output $f(x)$
4. Intermediate operation (Behavior $g$)
5. Output $g(f(x))$
6. Terminal operation (Behavior $h$)
There are two types of aggregate operations

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

---

**Overview of Common Stream Aggregate Operations**

- **Input x**
- **Intermediate operation (Behavior f)**
- **Output f(x)**
- **Intermediate operation (Behavior g)**
- **Output g(f(x))**
- **Terminal operation (Behavior h)**
There are two types of aggregate operations:

- **Intermediate operations**
- **Terminal operations**

Terminal operations:
- Trigger intermediate operations & produce a non-stream result
- e.g., forEach(), reduce(), collect(), etc.
Overview of Common Stream Aggregate Operations

- Overview of the `map()` intermediate operation

```
Input x
Stream `map(Function<...> mapper)`
Output f(x)
```

Applies the mapper function to every element of the input stream & returns an output stream consisting of the results

```
Output g(f(x))
R `collect(Collector<...> collector)`
```

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#map
Overview of Common Stream Aggregate Operations

- Overview of the map() intermediate operation

\[
\text{Stream } \text{map}(\text{Function}<...> \text{mapper})
\]

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

\[
\text{Stream filter}(\text{Predicate}<...> \text{pred})
\]

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

\[
R \text{ collect}(\text{Collector}<...> \text{collector})
\]

The # of output stream elements matches the # of input stream elements
Overview of Common Stream Aggregate Operations

• Overview of the map() intermediate operation

For each word to find determine the indices (if any) where the word matches the input string.
Overview of Common Stream Aggregate Operations

- Overview of the map() intermediate operation

```
Stream<SearchResults>
```

Search Words

```
"do", "re", "mi", "fa", "so", "la", "ti", "do"
```

Stream

```
stream()
```

```
map(this::searchForWord)
```

List

```
<String>
```

```
…
```

```
…
```

map() may transform the type of elements it processes
Overview of Common Stream Aggregate Operations

- Overview of the map() intermediate operation

```java
List<SearchResults> results = wordsToFind.stream()
    .map(this::searchForWord)
    .filter(not (SearchResults::isEmpty))
    .collect(toList());
```
Overview of Common Stream Aggregate Operations

- Overview of the `filter()` intermediate operation

Tests the given predicate against each element of the input stream & returns an output stream consisting only of the elements that match the predicate.

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#filter
Overview of the filter() intermediate operation

Tests the given predicate against each element of the input stream & returns an output stream consisting only of the elements that match the predicate.

Input $x$

Output $f(x)$

Output $g(f(x))$

$R$ collect($Collector<...> collector$)

The # of output stream elements may be less than the # of input stream elements.
Overview of Common Stream Aggregate Operations

- Overview of the filter() intermediate operation

The filter() aggregate operation *can’t* change the type of elements it processes.
Overview of Common Stream Aggregate Operations

- Overview of the filter() intermediate operation

```java
List<SearchResults> results = wordsToFind.stream()
    .map(this::searchForWord)
    .filter(not (SearchResults::isEmpty))
    .collect(toList());
```
Overview of Common Stream Aggregate Operations

- Intermediate operations are “lazy” & don’t run until a terminal operator is reached

Input $x$

Stream map($\text{Function}<\ldots> \text{mapper}$)

Output $f(x)$

Stream filter($\text{Predicate}<\ldots> \text{pred}$)

Output $g(f(x))$

$R \text{ collect}($Collector$<\ldots> \text{collector}$)

Output $h(g(f(x)))$

Overview of Common Stream Aggregate Operations

- A terminal operation triggers intermediate operation processing

\[ \text{Input } x \]

\[ \text{Stream map}(\text{Function}\langle \ldots \rangle \text{ mapper}) \]

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

\[ \text{Stream filter}(\text{Predicate}\langle \ldots \rangle \text{ pred}) \]

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

\[ R \text{ collect}(\text{Collector}\langle \ldots \rangle \text{ collector}) \]

\[ \text{Output } h(g(f(x))) \]
Overview of Common Stream Aggregate Operations

• Overview of the collect() terminal operation

This terminal operation uses a collector to perform a reduction on the elements of its input stream & returns the results of the reduction

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#collect
Overview of Common Stream Aggregate Operations

- Overview of the `collect()` terminal operation

![Diagram]

**Search Words**

```
"do", "re", "mi", "fa", 
"so", "la", "ti", "do"
```

**Stream**

```
Stream <String>
```

**List**

```
List <String>
```

**Stream**

```
Stream <SearchResults>
```

**Stream**

```
Stream <SearchResults>
```

**List**

```
List <SearchResults>
```

**Triggers intermediate operation processing**

```
collect(toList())
```
List<SearchResults> results = 
wordsToFind
.stream()
.map(this::searchForWord)
.filter(not
   (SearchResults::isEmpty))
.collect(toList());
Overview of Common Stream Aggregate Operations

- An aggregate operation *may* process all elements in a stream, e.g.
- `map()` processes all of the elements in its input stream

```
Input x

Stream map(Function<…> mapper)

Output f(x)

Stream limit(long maxSize)

Output g(f(x))

Optional findFirst()
```
Overview of Common Stream Aggregate Operations

- An aggregate operation *may* process all elements in a stream, e.g.
  - `map()` processes all of the elements in its input stream
  - `limit()` & `findFirst()` are "short-circuit" operations that halt further processing after their condition is reached

See www.logicbig.com/tutorials/core-java-tutorial/java-util-stream/short-circuiting
End of Overview of Java 8 Streams (Part 2)