Java Streams: Intermediate 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
- Intermediate operations

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

- Understand the structure & functionality of stream aggregate operations
- Intermediate 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 stream aggregate operations
- Intermediate operations

Let's start at the very beginning...

We continue to showcase the SimpleSearchStream program
Learning Objectives in this Part of the Lesson

• Understand the structure & functionality of stream aggregate operations
• Intermediate operations

Intermediate operations are “lazy” & run only after terminal operator is reached.

See www.logicbig.com/tutorials/core-java-tutorial/java-util-stream/lazy-evaluation
Overview of the map() Intermediate Operation
Overview of the map() Intermediate Operation

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

\[ \text{Input } x \]

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

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

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

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

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

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#map
Overview of the map() Intermediate Operation

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

The # of output stream elements must match the # of input stream elements.

Input $x$

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

Output $f(x)$

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

Output $g(f(x))$

$R \text{ collect(}\text{Collector}<\ldots>	ext{ collector})$
Overview of the map() Intermediate Operation

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

\[
\text{Input } x \\
\text{Stream } \text{map} (\text{Function}<\ldots> \text{mapper}) \\
\text{Output } f(x) \\
\text{Stream } \text{filter} (\text{Predicate}<\ldots> \text{pred}) \\
\text{Output } g(f(x)) \\
R \text{ collect} (\text{Collector}<\ldots> \text{collector})
\]

Naturally, a mapper may throw an exception, which could terminate map()
Overview of the map() Intermediate Operation

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

For each word to find, determine the indices (if any) where the word matches the input string.
Overview of the map() Intermediate Operation

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

\[
\text{List} <\text{String}> \\
\text{Stream} <\text{String}> \\
\text{Stream} <\text{SearchResults}>
\]

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

\[
\text{stream()}
\]

\[
\text{map(this::searchForWord)}
\]

map() may transform the type of elements it processes
Overview of the map() Intermediate Operation

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

```java
List<SearchResults> results = 
    wordsToFind
    .stream()
    .map(this::searchForWord)
    .filter(not (SearchResults::isEmpty))
    .collect(toList());
```

Note “fluent” programming style with cascading method calls.

See [en.wikipedia.org/wiki/Fluent_interface](http://en.wikipedia.org/wiki/Fluent_interface)
Overview of the filter()
Intermediate Operation
Overview of the filter() Intermediate Operation

- Tests a predicate against each element of input stream & returns an output stream containing only elements that match the predicate

\[ \text{Input } x \]

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

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

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

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

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

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

- Tests a predicate against each element of input stream & returns an output stream containing only elements that match the predicate

\[
\text{Input } x \quad \rightarrow \quad \text{Stream } \text{map}(\text{Function}<\ldots> \ \text{mapper}) \quad \rightarrow \quad \text{Output } f(x) \quad \rightarrow \quad \text{Stream } \text{filter}(\text{Predicate}<\ldots> \ \text{pred}) \quad \rightarrow \quad \text{Output } g(f(x)) \quad \rightarrow \quad R \ \text{collect}(\text{Collector}<\ldots> \ \text{collector})
\]

The # of output stream elements may be less than the # of input stream elements.
Overview of the filter() Intermediate Operation

- Tests a predicate against each element of input stream & returns an output stream containing only elements that match the predicate

List
<String>

Stream
<String>

Stream
<SearchResults>

Stream
<SearchResults>

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

stream()

map(this::searchForWord)

filter(not(SearchResults::isEmpty))

Filter out empty SearchResults.
Overview of the filter() Intermediate Operation

- Tests a predicate against each element of input stream & returns an output stream containing only elements that match the predicate

```
Stream<SearchResults>
```

```
List<String>
```

```
Stream<String>
```

```
stream()
```

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

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

Search Words

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

```
filter() can't change the type or value of elements it processes
```
Overview of the filter() Intermediate Operation

- Tests a predicate against each element of input stream & returns an output stream containing only elements that match the predicate.

```java
List<SearchResults> results =
    wordsToFind
    .stream()
    .map(this::searchForWord)
    .filter(not
        (SearchResults::isEmpty))
    .collect(toList());
```

Again, note the fluent interface style.

See [en.wikipedia.org/wiki/Fluent_interface](en.wikipedia.org/wiki/Fluent_interface)
Overview of the dropWhile() Intermediate Operation
Overview of the dropWhile() Intermediate Operation

- Overview of the dropWhile() intermediate operation (introduced in Java 9)

If this stream is unordered, return a stream consisting of the remaining elements of this stream after dropping a subset of elements that match the given predicate.

stream()

collect(groupingBy(...))

entrySet().stream()

dropWhile(e -> notEqual(e, word))

forEach(this::printResult)

See docs.oracle.com/javase/9/docs/api/java/util/stream/Stream.html#dropWhile
If this stream is ordered, return a stream consisting of the remaining elements of this stream after dropping the longest prefix of elements that match the given predicate.

See docs.oracle.com/javase/9/docs/api/java/util/stream/Stream.html#dropWhile
Overview of the `dropWhile()` Intermediate Operation

- Overview of the `dropWhile()` intermediate operation (introduced in Java 9)

`dropWhile()` is a “stateful” operation that is costly on ordered parallel streams since threads must cooperate to find the longest contiguous sequence of matching elements in encounter order.

See blog.indrek.io/articles/whats-new-in-java-9-streams
Overview of the dropWhile() Intermediate Operation

Overview of the dropWhile() intermediate operation (introduced in Java 9)

The # of output stream elements may be less than the # of input stream elements.

However, the semantics of dropWhile() differ from the semantics of filter().
Overview of the dropWhile() Intermediate Operation

- Overview of the dropWhile() intermediate operation (introduced in Java 9)

```
List<SearchResults> -> Stream<SearchResults> -> Map<String, List<SearchResults>>
Stream<Entry<String, List<SearchResults>>> -> dropWhile(e -> notEqual(e, word))
Void
```

*dropWhile() also can’t change the type or values of elements it processes*
Overview of the dropWhile() Intermediate Operation

- Overview of the dropWhile() intermediate operation (introduced in Java 9)

```java
listOfResults
.stream()
.collect(groupingBy
    (SearchResults::getWord,
     LinkedHashMap::new,
     toList()))
.entrySet()
.stream()
.dropWhile(e -> notEqual(e, word))
.forEach(e -> printResult
    (e.getKey(),
     e.getValue()));
```
Overview of the dropWhile() Intermediate Operation

- Overview of the dropWhile() intermediate operation (introduced in Java 9)

```java
listOfResults
  .stream()
  .collect
    (groupingBy
      (SearchResults::getWord,
       LinkedHashMap::new,
       toList()))
  .entrySet()
  .stream()
  .dropWhile(e -> notEqual(e, word))
  .forEach(e -> printResult
    (e.getKey(),
     e.getValue()));
```

Diagram:
- Convert list of search results into a stream
- stream()
- collect(groupingBy(...))
- entrySet().stream()
- dropWhile(e -> notEqual(e, word))
- forEach(this::printResult)
Overview of the dropWhile() Intermediate Operation

- Overview of the dropWhile() intermediate operation (introduced in Java 9)

```java
listOfResults
    .stream()
    .collect(
        groupingBy
            (SearchResults::getWord,
             LinkedHashMap::new,
             toList())
    )
    .entrySet()
    .stream()
    .dropWhile(e -> notEqual(e, word))
    .forEach(e -> printResult(e.getKey(), e.getValue()));
```

Collect stream into a map with words as key

Stream

Collect (groupingBy(...))

EntrySet().stream()

DropWhile(e -> notEqual(e, word))

ForEach(this::printResult)
Overview of the dropWhile() Intermediate Operation

- Overview of the dropWhile() intermediate operation (introduced in Java 9)

```java
listOfResults
  .stream()
  .collect
    (groupingBy
      (SearchResults::getWord,
        LinkedHashMap::new,
        toList()))
  .entrySet()
  .stream()
  .dropWhile(e -> notEqual(e, word))
  .forEach(e -> printResult
    (e.getKey(),
    e.getValue()));
```

Convert map into a stream of entries
Overview of the `dropWhile()` Intermediate Operation

- Overview of the `dropWhile()` intermediate operation (introduced in Java 9)

```java
listOfResults
  .stream()
  .collect
    (groupingBy
      (SearchResults::getWord,
       LinkedHashMap::new,
       toList()))
  .entrySet()
  .stream()
  .dropWhile(e -> notEqual(e, word))
  .forEach(e -> printResult
    (e.getKey(),
     e.getValue()));
```

**Ignore entries until there's a match**

`notEqual()` is defined as `return !e.getKey().equals(word)`
Overview of the dropWhile() Intermediate Operation

- Overview of the dropWhile() intermediate operation (introduced in Java 9)

```java
listOfResults
  .stream()
  .collect
    (groupingBy
      (SearchResults::getWord,
       LinkedHashMap::new,
       toList()))
  .entrySet()
  .stream()
  .dropWhile(e -> notEqual(e, word))
  .forEach(e -> printResult
            (e.getKey(),
             e.getValue()));
```

Print results starting at the match
End of Java Streams: Intermediate Operations