Java Streams Intermediate

Operations filter() & flatMap()

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
  - map() & mapToInt()
  - filter() & flatMap()

These are both stateless, run-to-completion operations
Learning Objectives in this Part of the Lesson

- Understand the structure & functionality of stream aggregate operations
- Intermediate operations
  - `map()` & `mapToInt()`
  - `filter()` & `flatMap()`
- We also discuss a curious limitation with `flatMap()` that makes it ineffective for parallel streams

See [stackoverflow.com/questions/45038120/parallel-flatmap-always-sequential/66386078](stackoverflow.com/questions/45038120/parallel-flatmap-always-sequential/66386078)
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

```
Input x

Stream map(Function<…> mapper)

Output f(x)

Stream filter(Predicate<…> pred)

Output g(f(x))

R collect(Collector<…> collector)
```

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#filter](https://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.

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

Input x

Stream map(Function<…> mapper)

Output f(x)

Stream filter(Predicate<…> pred)

Output g(f(x))

R collect(Collectors<…> collector)
Overview of the filter() Intermediate Operation

- Example of applying filter() & a predicate in the SimpleSearchStream program

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

Example of applying filter() & a predicate in the SimpleSearchStream program

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

- Example of applying filter() & a predicate in the SimpleSearchStream program

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

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

Again, note the fluent interface style.

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

- Returns a stream that replaces each stream element w/contents of a mapped stream produced by applying the provided mapping function to each element.

This definition sounds like map() at first glance, but there are important differences!

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

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

- Returns a stream that replaces each stream element with contents of a mapped stream produced by applying the provided mapping function to each element.

The # of output stream elements may differ from the # of input stream elements.

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

Output $f(x)$
Output $g(f(x))$
Output $\ldots(g(f(x)))$
Overview of the flatMap() Intermediate Operation

- Returns a stream that replaces each stream element with contents of a mapped stream produced by applying the provided mapping function to each element.

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

Output \( f(x) \)

Output \( g(f(x)) \)

Output \( …(g(f(x))) \)

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

“Flatten” an array of lists of strings into a stream of strings.
Overview of the flatMap() Intermediate Operation

- Returns a stream that replaces each stream element with contents of a mapped stream produced by applying the provided mapping function to each element.

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

Output \( f(x) \)

Output \( g(f(x)) \)

Output \( \ldots(g(f(x))) \)

forEach(System.out::println)

flatMap() *may* transform the type of elements it processes.
List<String> l1 = ...;
List<String> l2 = ...;
...
List<String> ln = ...;

Stream
    .of(l1, l2, l3, ..., ln)
    .flatMap(List::stream)
    .filter(s -> s.charAt(0).toLowerCase() == 'h')
    .forEach(System.out::println);

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex12](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex12)

Overview of the flatMap() Intermediate Operation

- Returns a stream that replaces each stream element with contents of a mapped stream produced by applying the provided mapping function to each element.
A Limitation with flatMap()
A Limitation with flatMap()

- A limitation with the flatMap() implementation forces sequential processing

```java
<R> Stream<R> flatMap
    (Function<? super P_OUT,
        ? extends Stream<? extends R>>
mapper) {

    ...

    public void accept(P_OUT u) {
        try(Stream<? extends R> result
            = mapper.apply(u)) {
            if (result != null) {
                if (...) {
                    result
                        .sequential()
                        .forEach(downstream);
                ...
            }
        }
    }
```

This code always runs sequentially for “inner streams” that use flatMap()

See stackoverflow.com/questions/45038120/parallel-flatmap-always-sequential/66386078
A Limitation with flatMap()

- A limitation with the flatMap() implementation forces sequential processing

```java
List<Integer> list = IntStream.rangeClosed(1, outerCount)
    .boxed()
    .parallel()
    .flatMap(innerCount -> IntStream.rangeClosed(1, innerCount)
        .boxed()
        .parallel())
    .collect(toList());
```

Due to a limitation with flatMap() this inner stream will always run sequentially, even though it is explicitly designated as .parallel()
A Limitation with flatMap()

- A simple workaround is to use replace flatMap() with map() & reduce(Stream::concat)

```java
List<Integer> list = IntStream
    .rangeClosed(1, outerCount)
    .boxed()
    .parallel()
    .map(innerCount -> IntStream
        .rangeClosed(1, innerCount)
        .boxed()
        .parallel())
    .reduce(Stream::concat)
    .orElse(Stream.empty())
    .collect(toList());
```

This inner stream now runs in parallel, as intended

See github.com/douglas craigschmidt/LiveLessons/tree/master/Java8/ex35
End of Java Streams
Intermediate Operations
filter() & flatMap()