Understand Java Streams Intermediate Operations map() & mapToInt()
Learning Objectives in this Part of the Lesson

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

Input String to Search

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

Let's start at the very beginning..

We continue to showcase the SimpleSearchStream program

See github.com/douglascraigschmidt/LiveLessons/tree/master/SimpleSearchStream
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.

Learning Objectives in this Part of the Lesson

- Understand the structure & functionality of stream aggregate operations
- Intermediate operations
  - `map()` & `mapToInt()`

These are both stateless, run-to-completion operations
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

```
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#map
• Applies a mapper function to every element of the input stream & returns an output stream consisting of the results

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

The # of output stream elements must match the # of input stream elements.
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
- A mapper may throw an exception, which could terminate map()

Input $x$

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

Output $f(x)$

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

Output $g(f(x))$

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

See dzone.com/articles/exception-handling-in-java-streams
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
- A mapper may throw an exception, which could terminate map()
- A short-circuit terminal operation also causes the map() operation to only process a subset of the input stream

See dzone.com/articles/collectors-part-1-%E2%80%93-reductions
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
- A mapper may throw an exception, which could terminate map()
- A short-circuit terminal operation also causes the map() operation to only process a subset of the input stream

\[
\text{Input } x
\]

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

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

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

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

\[
\text{Optional<T> findFirst()}
\]

These caveats apply to all “run-to-completion” intermediate operations!
For each word to find, determine the indices (if any) where the word matches the input string.
Overview of the map() Intermediate Operation

- Example of applying map() & a mapper function in the SimpleSearchStream program

List
<String>

Stream
<String>

Stream
<SearchResults>

map(this::searchForWord)

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

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

- Example of applying map() & a mapper function in the SimpleSearchStream program

```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](en.wikipedia.org/wiki/Fluent_interface)
Overview of the mapToInt() Intermediate Operation
Overview of the `mapToInt()` Intermediate Operation

- Returns an `IntStream` consisting of the results of applying the given mapper function to all elements of the input stream.

```
Input x

IntStream mapToInt(TToIntFunction<...> mapper)

Output f(x)

max()

Output g(f(x))

orElse(0)
```

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#mapToInt](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#mapToInt)
Overview of the `mapToInt()` Intermediate Operation

- Returns an `IntStream` consisting of the results of applying the given mapper function to all elements of the input stream

```
Input x

IntStream mapToInt(TToIntFunction<...> mapper)

Output f(x)

max()

Output g(f(x))

orElse(0)
```

*IntStream is a specialization of Stream for the int primitive.*

See [docs.oracle.com/javase/8/docs/api/java/util/stream/IntStream.html](docs.oracle.com/javase/8/docs/api/java/util/stream/IntStream.html)
Overview of the mapToInt() Intermediate Operation

- Returns an IntStream consisting of the results of applying the given mapper function to all elements of the input stream.

```
Input x

IntStream mapToInt(TToIntFunction<…> mapper)

Output f(x)

max()

Output g(f(x))

orElse(0)
```

The # of output stream elements must match the # of input stream elements.
Overview of the `mapToInt()` Intermediate Operation

- Example of applying `mapToInt()` & a mapper function in the SimpleSearchStream program

Transform the stream of results into a stream of primitive int indices.

```
List <Result>
Stream <Result>
IntStream <int>
OptionalInt
```

```
resultsList
128|138|148|199|209|219|503

stream()
mapToInt(Result::getIndex)
max()
orElse(0)
```
Overview of the `mapToInt()` Intermediate Operation

- Example of applying `mapToInt()` & a mapper function in the `SimpleSearchStream` program

List `<Result>`

Stream `<Result>`

IntStream `<int>`

OptionalInt

mapToInt() transforms the type of elements it processes into primitive ints
Example of applying mapToInt() & a mapper function in the SimpleSearchStream program

```java
int computeMax(List<SearchResults.Result> resultsList) {
    return resultsList.stream()
        .mapToInt(SearchResults.Result::getIndex)
        .max()
        .orElse(0);
}
```

Overview of the mapToInt() Intermediate Operation

Note “fluent” programming style with cascading method calls.

See [en.wikipedia.org/wiki/Fluent_interface](https://en.wikipedia.org/wiki/Fluent_interface)
End of Understand Java Streams Intermediate Operations
map() & mapToInt()