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

```
Aggregate operation (Function f)
Input x
Output f(x)
```

```
Aggregate operation (Function g)
Output g(f(x))
```

```
Terminal operation (Function 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](image-url)
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

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

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())

See 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|1879|1925|1979|1999|1989|2161|2254|2276|2283]...

Word "Ti" matched at index [237|994|1272|1294|1364|1850|1860|1912|1915|1952|1955|2299]

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

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

```
List <String>
```

**Input a list of words to find**

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

```
stream()
```
Overview of SimpleSearchStream Example

- This example finds words in an input string

List
<String>

```
|   |   |   |   |   | ... |   |
```

Convert collection to a (sequential) stream

Search Words

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

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

List
<String>

Stream
<String>

Input a stream of words to find

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

```
stream()
```

```
map (this::searchForWord)
```
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 each 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()
Overview of SimpleSearchStream Example

- This example finds words in an input string

Input a stream of search results

List
<String>

Stream
<String>

Stream
/SearchResults>

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

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

Stream
<SearchResults>

List
<String>

Stream
<String>

stream() → map (this::searchForWord) → filter (not(SearchResults::isEmpty))

Remove empty search results from the stream
Overview of SimpleSearchStream Example

- This example finds words in an input string

Output a stream of non-empty search results

```
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))
```

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

- This example finds words in an input string

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

```
List <String>
        |  |
        v  v
Stream <String>  stream()
        |  |
        v  v
Stream <SearchResults>  map (this::searchForWord)
        |  |
        v  v
Stream <SearchResults>  filter (not(SearchResults::isEmpty))
        |  |
        v  v
Stream <SearchResults>  collect (toList())
```

Search Words
```
"do", "re", "mi", "fa", "so", "la", "ti", "do"
```
Overview of SimpleSearchStream Example

• This example finds words in an input string

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

Search Words

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

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

Trigger intermediate operation processing
Overview of SimpleSearchStream Example

- This example finds words in an input string

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

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
The “physical” processing of a stream is different from the “logical” model we discussed here.

**Overview of SimpleSearchStream Example**

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

- **Stream**
  - `<String>`
  - `<SearchResults>`

- **List**
  - `<String>`
  - `<SearchResults>`

Overview of Common Stream Aggregate Operations
Overview of Common Stream Aggregate Operations

- An aggregate operation processes all elements in a stream

An aggregate operation is a higher-order function that applies a “behavior” (e.g., function or predicate) parameter to every element in a stream

Input x

Aggregate operation (Function f)

Output f(x)

Aggregate operation (Function g)

Output g(f(x))

Aggregate operation (Function h)

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

• An aggregate operation processes all elements in a stream

The focus is on the “what” (declarative), *not* the “how” (imperative)
Overview of Common Stream Aggregate Operations

- An aggregate operation processes all elements in a stream

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

Common aggregate operations include \text{map}(), \text{filter}(), & \text{collect}()
Overview of Common Stream Aggregate Operations

- An aggregate operation processes all elements in a stream

```
Stream map(Function<...> mapper)
```

Applies the given function to the elements of the input stream & returns an output stream consisting of the results.

```
Stream filter(Predicate<...> pred)
```

Output $f(x)$

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

Output $g(f(x))$

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#map
An aggregate operation processes all elements in a stream.

- **Overview of Common Stream Aggregate Operations**

  - **Streammap** (Function<...> mapper)
  - **Streamfilter** (Predicate<...> pred)
  - **collect** (Collector<...> collector)

- Applies the given function to the elements of the input stream & returns an output stream consisting of the results.

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

- An aggregate operation processes all elements in a stream

For each word to find determine the indices (if any) where the word matches the input string

The map() aggregate operation may change the type of the elements it processes
Overview of Common Stream Aggregate Operations

- An aggregate operation processes all elements in a stream

The map() aggregate operation *may* change the type of elements it processes
Overview of Common Stream Aggregate Operations

- An aggregate operation processes all elements in a stream

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

- An aggregate operation processes all elements in a stream

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

```
Stream map(Function<…> mapper)
```

Input x

```
Stream filter(Predicate<…> pred)
```

Output f(x)

```
R collect(Collectors<…> collector)
```

Output g(f(x))

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#filter
An aggregate operation processes all elements in a stream.

Overview of Common Stream Aggregate Operations

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

- Stream map(Function<...> mapper)
- Stream filter(Predicate<...> pred)
- R collect(Collector<...> collector)

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

- An aggregate operation processes all elements in a stream

List
<String>

Stream
<String>

Stream
<SearchResults>

Stream
<SearchResults>

The filter() aggregate operation can’t change the type of elements it processes

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

stream()

map (this::searchForWord)

filter (not(SearchResults::isEmpty))

Filter out empty SearchResults

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

- An aggregate operation processes all elements in a stream

```
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))
```
Overview of Common Stream Aggregate Operations

• An aggregate operation processes all elements in a stream

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

- An aggregate operation processes all elements in a stream

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

- An aggregate operation processes all elements in a stream

```
Input x
Stream map(Function<…> mapper)
Output f(x)
Stream filter(Predicate<…> pred)
Output g(f(x))
R collect(Collector<…> collector)
Output h(g(f(x)))
```

Intermediate operations are “lazy” & don’t run until a terminal operator is reached
Overview of Common Stream Aggregate Operations

- An aggregate operation processes all elements in a stream

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

Input x

- Stream map(Function<…> mapper)

  Output f(x)

- Stream filter(Predicate<…> pred)

  Output g(f(x))

- R collect(Collectors<…> collector)

  Output h(g(f(x)))

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

- An aggregate operation processes all elements in a stream

- Trigger intermediate operation processing & also create a list of SearchResults

List
<String>

Stream
<String>

Stream
<SearchResults>

Stream
<SearchResults>

List
<SearchResults>

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

stream()

map (this::searchForWord)

filter (not(SearchResults::isEmpty))

collect (toList())
Overview of Common Stream Aggregate Operations

- An aggregate operation processes all elements in a stream

Trigger intermediate operation processing & also create a list of SearchResults

List<SearchResults> results =
    wordsToFind
    .stream()
    .map(this::searchForWord)
    .filter(not (SearchResults::isEmpty))
    .collect(toList());
End of Overview of Java 8 Streams (Part 2)