Java Streams API

Charles Moloney

Advanced Java Learning Workshops: Week 1

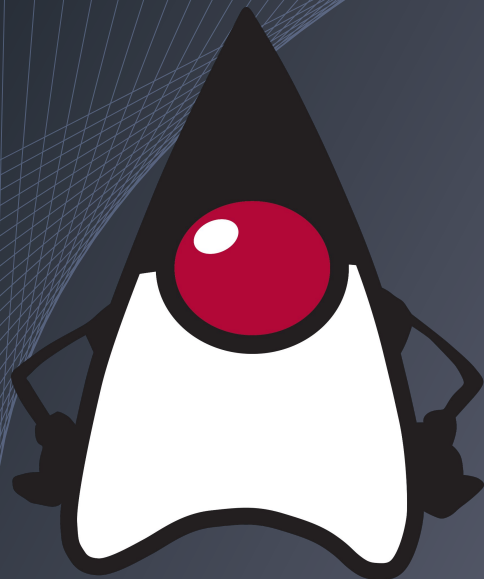
Agenda

1. Background
2. Brain Teaser
3. Introduction to Streams
4. Live Code-Along
5. Questions

Background

Java can do anything!!!

- But it's mostly used for backends
  - Data manipulation is a common task
  - Writing concise, readable, robust code is key
- A lot has changed since the language first came out:
  - Java 5: 2004
    - Generics, Enums
  - Java 8: 2014
    - Lambdas, Streams, and much more
  - Java 11: 2018
    - "var" keyword, better file and String support
  - Java 17: 2021
    - Text blocks, better switch statements, records
  - Java 21: 2023
    - Pattern matching, Sequenced Collections



Brain Teaser

```
String[] teaser = {"Ellenna", "Brendan", "Josh", "Charles", "Emma"};
```

I have an array of Strings. Write a function that returns me a single String separated by dashes where:

1. Strings with less than or equal to 4 characters are ignored
2. The Strings are all uppercase
3. The Strings are sorted in alphabetical order
4. Each String in the array is separated by dashes

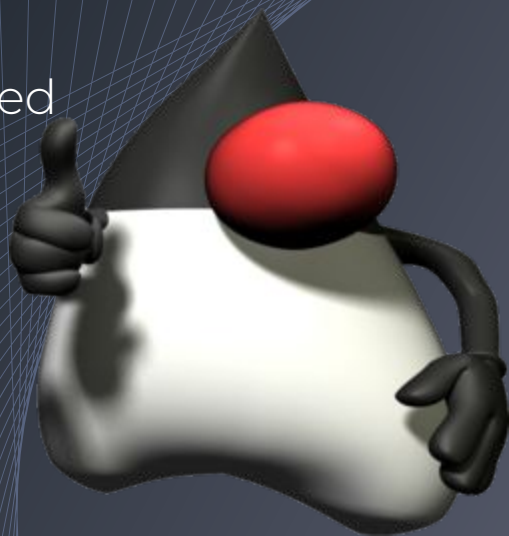
```
"BRENDAN-CHARLES-ELLENNA"
```

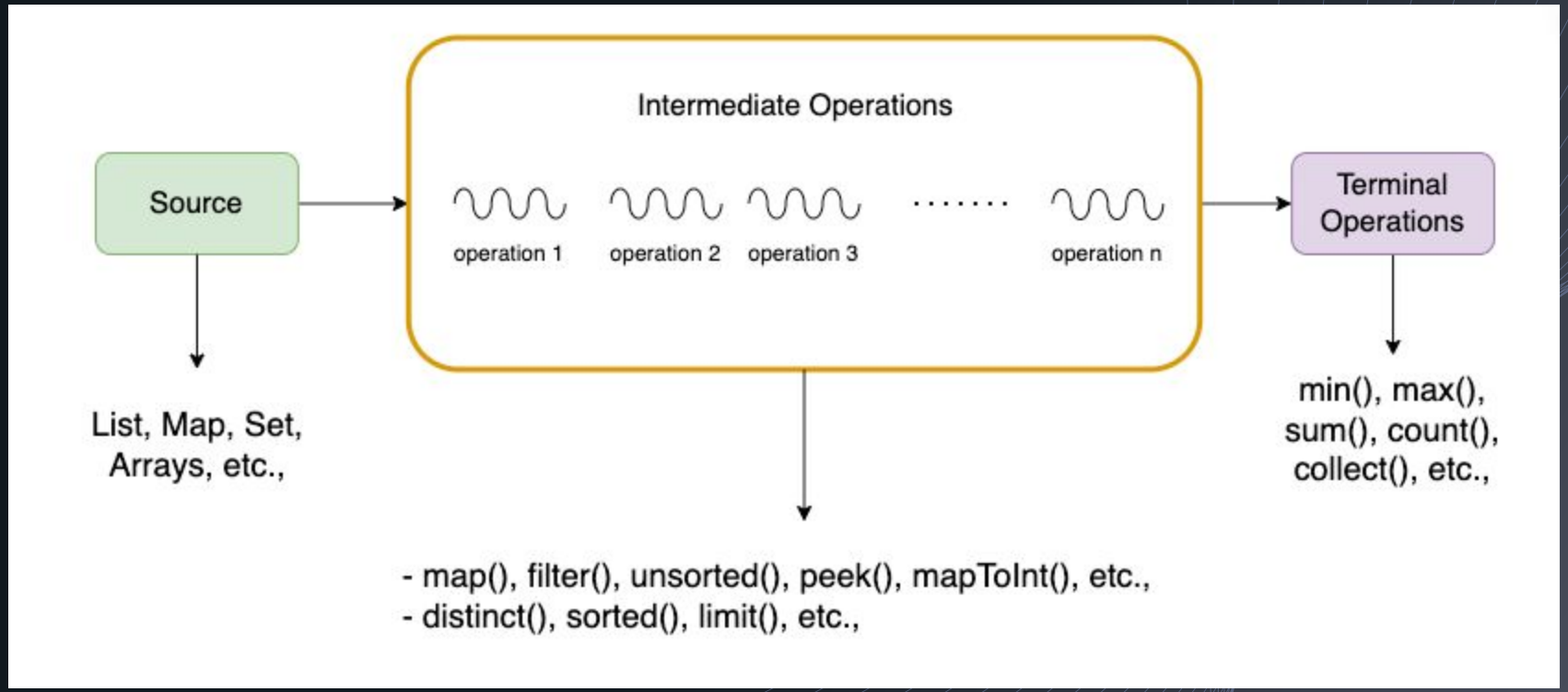
```
return Arrays.stream(teaser)
        .filter(s -> s.length() > 4)
        .map(String::toUpperCase)
        .sorted()
        .collect(Collectors.joining("-"));
```

Introduction to Streams

Streams

- A stream is a sequence of elements
- A stream is **NOT** a collection (List, Set, etc.)
  - Single-use
  - Items are obtained/processed on demand, not stored
- Can be ordered or unordered
- Can be parallel or sequential
- Employ Lazy Execution
  - Operations aren't performed until a value is needed
- Do not affect the original object*
- Special support for Ints, Doubles, and Longs




Intermediate Operations
Source
operation 1
operation 2
operation 3
operation n
Terminal Operations
List, Map, Set, Arrays, etc.,
min(), max(), sum(), count(), collect(), etc.,
- map(), filter(), unsorted(), peek(), mapToInt(), etc.,
- distinct(), sorted(), limit(), etc.,

```
return Arrays.stream(teaser)
        .filter(s -> s.length() > 4)
        .map(String::toUpperCase)
        .sorted()
        .collect(Collectors.joining("-"));
```

```
return Arrays.stream(teaser)
        .filter(s -> s.length() > 4)
        .map(String::toUpperCase)
        .sorted()
        .collect(Collectors.joining("-"));
```

Streams make heavy use of Lambda Expressions

```
s -> s.length() > 4
```

```
String::toUpperCase
```

Source Operations

- .stream()
  - Returns a stream from the specified object
- .parallelStream()
  - Returns a parallelized stream from the specified object
- .of(T... values)
  - Directly creates a stream from any number of values
- .generate(Supplier<T> s)
  - Creates an infinite stream based off some supplier
  - Stream.generate(() -> Math.random())
- .iterate(T seed, UnaryOperator<T> f)
  - Creates an infinite stream based off a seed and function
  - Stream.iterate(2, i -> i * 2);

Intermediate Operations

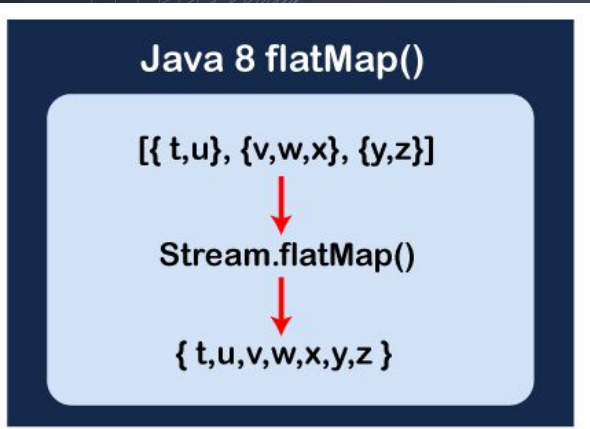
- .map(Function<? super T,? extends R> mapper)
  - Returns a stream consisting of the results of applying the given function to the elements of this stream.
  - Can change the **TYPE** of the stream
  - Stream.of(“a”, “aa”).map(x -> x.length())
    - Example converts from type String to type int
  - Also support for longs, ints, and doubles
    - mapToInt, mapToDouble, mapToLong
- .filter(Predicate<? super T> predicate)
  - Returns a stream consisting of the elements of this stream that match the given predicate.
  - Can change the **NUMBER OF ELEMENTS** of the stream

Intermediate Operations

- .peek(Consumer<? super T> action)
  - Returns a stream consisting of the elements of this stream, additionally performing the provided action on each element as elements are consumed from the resulting stream.
  - Cannot change the type of the stream
- .sorted()
  - Returns a stream consisting of the elements of this stream, sorted according to natural order.
  - Can also pass in a comparator

Intermediate Operations

- .flatMap(Function<? super T,? extends Stream<? extends R>> mapper)
  - Returns a stream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element.
  - Also specific flatmaps for long, double, int
    - flatMapToInt
    - flatMapToLong
    - flatMapToDouble



Intermediate Operations

- .distinct()
  - Returns a stream consisting of the distinct elements (according to Object.equals(Object)) of this stream.
- .limit(long maxSize)
  - Returns a stream consisting of the elements of this stream, truncated to be no longer than maxSize in length.
- .parallel()
  - Returns an equivalent stream that is parallel
- .unordered()
  - Returns an equivalent stream that is unordered
- .sequential()
  - Returns an equivalent stream that is sequential.

Terminal Operations

- .forEach(Consumer<? super T> action)
  - Performs an action for each element of this stream.
  - **Note: Collections have a similar method to directly perform this method**
- .collect(Collector<? super T,A,R> collector)
  - Performs a mutable reduction operation on the elements of this stream using a Collector.
- .collect(Supplier<R> supplier, BiConsumer<R,? super T> accumulator, BiConsumer<R,R> combiner)
  - Performs a mutable reduction operation on the elements of this stream.

Terminal Operations - Collectors

- .joining()
  - Used for strings
  - Can be overloaded with delimiters and fencepost prefix/suffix
- .toSet()
- .toCollection(Supplier<C> collectionFactory)
  - Can specify a specific type of collection
- .toList()
  - No guarantee on what type of list
- .groupingBy(Function<? super T,? extends K> classifier)
  - Creates a Map based on specified groupings
- .mapping(Function<? super T,? extends U> mapper, Collector<? super U,A,R> downstream)
  - Apply a mapping function to each element before accumulation

Terminal Operations

- .findFirst()
  - Returns an Optional describing the first element of this stream, or an empty Optional if the stream is empty.
  - Optional is a wrapper class that handles null safety
- count()
  - Returns the number of elements still in the stream
- min(Comparator<? super T> comparator)
  - Returns the minimum element of this stream according to the provided Comparator.
  - Same operation for max
- reduce(T identity, BinaryOperator<T> accumulator)
  - identity is the starting value and accumulator is the binary operation we repeatedly apply.
  - sum, min, max, average, and concat are all special cases

Code Time!