Java Streams

Richard Warburton
JAVA CHAMPION, AUTHOR AND PROGRAMMER

@richardwarburto www.monotonic.co.uk

For loops and iterators are a low level, error prone construct.

Java 8 Streams

A stream is a way of supporting functional-style operations on Collections. In other words aggregate operations that work on sequences of values.

Overview

Introduce Streams in code

Stream Operations

Collectors

Limitations and General features

Live Coding Streams

Operations on Streams

```
streamOfProducts.filter(product -> product.getWeight() > 20)
```

Filter

Remove elements from the Stream that don’t match a predicate

```
streamOfProducts.map(Product::getName)
```

Map

Transform elements from one value into another

```
products.anyMatch(
  prod -> prod.getWeight() > 20);

products.noneMatch(
  prod -> prod.getWeight() > 20);

products.allMatch(
  prod -> prod.getWeight() > 20);
```

- Match family
- Terminal Operation
- Returns a Boolean
- If any / none / all elements match a Predicate

Skip and limit

```
streamOfProducts
  // Discard next N elements
  .skip(elementsOnPage * pageNumber)
  // Only keep next N elements
  .limit(elementsOnPage)
```

Sorted

```
// Sort Comparable objects with default order

products.map(Product::getName).sorted()


// Sort objects with a specified comparator

Comparator<Product> byName = Comparator.comparing(Product::getName)

products.sorted(byName)
```

```
streamOfShipments.flatMap(shipment -> shipment.getLightVanProducts().stream())
```

FlatMap

Transform elements from one value into zero, one or many values

```
// max (or min) element given a sort order
products.max(Comparator.comparingInt(Product::getWeight))

// Side effecting action for each element
products.forEach(prod -> System.out.println(prod.getName()))

// findFirst (or findAny()) get the element
products.filter(prod -> prod.getName().contains(“Chair”)).findFirst()

// Count number of elements in a stream
products.filter(prod -> prod.getName().contains(“Chair”)).count()
```

```
streamOfProducts.reduce(0, (acc, product) -> acc + product.getWeight())
```

Reduce

Combine elements together using a combining function.

Enter the Collector

Conclusion

Are Streams always better?

Streams	Loops
High Level construct	Low level construct
Optimized framework	Can be faster
General better readability	Readability is subjective
Some corner cases worse	Nicer with checked Exceptions
Java 8 or later	All Java versions

Beyond Streams

Further Streams Learning Material

Advanced Collectors

Parallel Streams

Summary

Streams are a powerful Abstraction in Java 8+

Can replace many for loops and iterators

Rely heavily upon Lambda Expressions and Method References.

Next Up: Collection Factories and Improvements