USF UNIVERSITY OF SAN FRANCISCO
CHANGE THE WORLD FROM HERE

Stream Pipelines

CS 272 Software Development

Department of Computer Science
https://www.cs.usfca.edu/

Java Stream Pipelines

- Initial stream source
- Zero or more **intermediate** operations
  - Lazily transform stream into another stream
- One **terminal** operation
  - Eagerly triggers the data processing
  - Produces a result or side effect and closes the stream

https://developer.ibm.com/articles/j-java-streams-1-brian-goetz/

Pipeline Anatomy

Data Sources

```
Collection.stream()
Stream.of(…)
BufferedReader.lines()
CharSequence.chars()
IntStream.range(…)
Random.ints(…)
Stream.iterate(…)
Stream.generate(…)
…
```

Intermediate Ops

```
filter(…)
map(…)
flatMap(…)
distinct()
sorted()
limit()
skip()
takeWhile(…)
…
```

Terminal Ops

```
forEach(…)
toArray()
reduce(…)
min(…)
collect(…)
count()
anyMatch(…)
findFirst()
…
```

https://developer.ibm.com/articles/j-java-streams-1-brian-goetz/

Operation Types

Lazy vs eager

Intermediate vs terminal

Stateless vs stateful

Short-circuiting

Non-interfering

With side-effects

Operation Types

Lazy vs eager

Intermediate vs terminal

Stateless vs stateful

Short-circuiting

Non-interfering

With side-effects

Lazy operations

- Executed when needed

Eager operations

- Executed immediately

Operation Types

Lazy vs eager

Intermediate vs terminal

Stateless vs stateful

Short-circuiting

Non-interfering

With side-effects

Intermediate operations

- Always **lazy**
- Return a new stream

Terminal operations

- Usually **eager**
- Returns or has **side effect**
- Closes the stream

Operation Types

Lazy vs eager

Intermediate vs terminal

Stateless vs stateful

Short-circuiting

Non-interfering

With side-effects

Stateless operations

- Don't depend on earlier action
- Ideal for λ-expressions

Stateful operations

- Depends on earlier action
- Bad for parallelism
  - e.g. `distinct()`

Operation Types

Lazy vs eager

Intermediate vs terminal

Stateless vs stateful

Short-circuiting

Non-interfering

With side-effects

Intermediate **short-circuiting** op

- Produces a finite stream from an infinite stream

Terminal **short-circuiting** op

- Terminates in finite time given infinite input

Operation Types

Lazy vs eager

Intermediate vs terminal

Stateless vs stateful

Short-circuiting

Non-interfering

With side-effects

- Ops should be **non-interfering** and **stateless** for parallelism
- **Interference** occurs when the source is unsafely modified during execution

Operation Types

Lazy vs eager

Intermediate vs terminal

Stateless vs stateful

Short-circuiting

Non-interfering

With side-effects

- **Side effects** occur when ops modify state outside scope
  - e.g. λ-expression modifying a list outside its scope
- Terminal operations may have **side effects**
  - Should still be avoided!

Parallelism

- Pipelines without **side-effects** , that are **non-interfering**, **stateless**, and **unordered** may be easily parallelized
  - **Ordered** streams (e.g. from lists) may have non-deterministic results if parallelised
- Involves adding **`parallelStream()`** to pipeline
  - Often easier than multithreading explicitly

https://developer.ibm.com/articles/j-java-streams-1-brian-goetz/

References

Package java.util.stream
https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/util/stream/package-summary.html

The Java Tutorials – Lesson: Aggregate Operations
https://docs.oracle.com/javase/tutorial/collections/streams/index.html

“An introduction to the java.util.stream library” by Brian Goetz
https://developer.ibm.com/articles/j-java-streams-1-brian-goetz/

CHANGE THE WORLD FROM HERE

Professor Sophie Engle
sjengle.cs.usfca.edu