Understand Java Stream Internals: Execution

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

- Understand stream internals, e.g.
  - Know what can change & what can’t
  - Recognize how a Java stream is constructed
- Be aware of how a Java stream is executed
Java Stream Execution
Java Stream Execution

- When terminal operation runs the streams framework picks an execution plan.
Java Stream Execution

- When terminal operation runs the streams framework picks an execution plan.
- The plan is based on properties of the source & aggregate operations.

\[
\begin{align*}
\text{Input } x & \quad \text{Stream map(Function\langle\ldots\rangle \ mapper)} \\
\text{Output } f(x) & \quad \text{Stream filter(Predicate\langle\ldots\rangle \ pred)} \\
\text{Output } g(f(x)) & \quad \text{Stream sorted()} \\
\text{Output } h(g(f(x))) & \quad R \text{ collect(Collector\langle\ldots\rangle \ collector)}
\end{align*}
\]
Java Stream Execution

- When terminal operation runs the streams framework picks an execution plan
- The plan is based on properties of the source & aggregate operations
- Intermediate operations are divided into two categories

```
When terminal operation runs the streams framework picks an execution plan
```

```
• The plan is based on properties of the source & aggregate operations
```

```
• Intermediate operations are divided into two categories
```

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

```
```

```
```

```
THING 1
THING 2
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```
Java Stream Execution

- When terminal operation runs the streams framework picks an execution plan.
- The plan is based on properties of the source & aggregate operations.
- Intermediate operations are divided into two categories:
  - Stateless
    - e.g., filter(), map(), flatMap(), etc.
  - Stateful

A pipeline with only stateless operations runs in one pass (even if it’s parallel).
When terminal operation runs the streams framework picks an execution plan.

- The plan is based on properties of the source & aggregate operations.

- Intermediate operations are divided into two categories:
  - Stateless
  - Stateful
    - e.g., `sorted()`, `limit()`, `distinct()`, `dropWhile()`, etc.

A pipeline with stateful operations is divided into sections & runs in multiple passes.
Java Stream Execution

- When terminal operation runs the streams framework picks an execution plan
- The plan is based on properties of the source & aggregate operations
- Intermediate operations are divided into two categories
- Terminal operations are also divided into two categories

```
Stream map(Function<...> mapper)
Stream filter(Predicate<...> pred)
Stream sorted()
R collect(Collectors<...> collector)
```
Java Stream Execution

- When terminal operation runs the streams framework picks an execution plan
  - The plan is based on properties of the source & aggregate operations
  - Intermediate operations are divided into two categories
  - Terminal operations are also divided into two categories
    - Run-to-completion
      - e.g., reduce(), collect(), forEach(), etc.

These terminal operation process data in bulk using Spliterator.forEachRemaining()
When terminal operation runs the streams framework picks an execution plan.

- The plan is based on properties of the source & aggregate operations.
- Intermediate operations are divided into two categories.
- Terminal operations are also divided into two categories:
  - Run-to-completion
  - Short-circuiting (e.g., anyMatch(), findFirst(), etc.)

Java Stream Execution

These terminal operation process data one element at a time using tryAdvance().
End of Understand Java
Stream Internals: Execution