Java Streams: Overview & Benefits

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 the structure & functionality of Java streams

Diagram:

- **Stream source**
  - Input $x$
  - Aggregate operation (behavior $f$)
  - Output $f(x)$
  - Aggregate operation (behavior $g$)
  - Output $g(f(x))$
  - Aggregate operation (behavior $h$)
Learning Objectives in this Part of the Lesson

• Understand the structure & functionality of Java streams, e.g.,
• Fundamentals of streams

Stream source

Input x

Aggregate operation (behavior f)

Output f(x)

Aggregate operation (behavior g)

Output g(f(x))

Aggregate operation (behavior h)
Learning Objectives in this Part of the Lesson

- Understand the structure & functionality of Java streams, e.g.,
  - Fundamentals of streams
  - Benefits of streams
Overview of Java Streams
Overview of Java Streams

- Java streams are a framework first introduced into the Java class library in Java 8

See docs.oracle.com/javase/tutorial/collections/streams

What's New in JDK 8

Java Platform, Standard Edition 8 is a major feature release. This document summarizes features and enhancements in Java SE 8 and in JDK 8, Oracle’s implementation of Java SE 8. Click the component name for a more detailed description of the enhancements for that component.

- Java Programming Language
  - Lambda Expressions, a new language feature, has been introduced in this release. They enable you to treat functionality as a method argument, or code as data. Lambda expressions let you express instances of single-method interfaces (referred to as functional interfaces) more compactly.
  - Method references provide easy-to-read lambda expressions for methods that already have a name.
  - Default methods enable new functionality to be added to the interfaces of libraries and ensure binary compatibility with code written for older versions of those interfaces.
  - Repeating Annotations provide the ability to apply the same annotation type more than once to the same declaration or type use.
  - Type Annotations provide the ability to apply an annotation anywhere a type is used, not just on a declaration. Used with a pluggable type system, this feature enables improved type checking of your code.
  - Improved type inference.
  - Method parameter reflection.

- Collections
  - Classes in the new java.util.stream package provide a Stream API to support functional-style operations on streams of elements. The Stream API is integrated into the Collections API, which enables bulk operations on collections, such as sequential or parallel map-reduce transformations.
  - Performance Improvement for HashMaps with Key Collisions

See docs.oracle.com/javase/tutorial/collections/streams
Overview of Java Streams

- Java streams are a framework first introduced into the Java class library in Java 8
- They have continued to evolve a bit in later versions of Java

Overview of Java Streams

A stream is a pipeline of aggregate operations that process a sequence of elements (aka, “values” or “data”)

Input $x$

$\text{Aggregate operation (behavior } f\text{)}$

Output $f(x)$

$\text{Aggregate operation (behavior } g\text{)}$

Output $g(f(x))$

$\text{Aggregate operation (behavior } h\text{)}$

See docs.oracle.com/javase/tutorial/collections/streams
Overview of Java Streams

- A stream is a pipeline of aggregate operations that process a sequence of elements (aka, “values” or “data”)

An aggregate operation is a higher-order function that applies a “behavior” param to every element in a stream.

See en.wikipedia.org/wiki/Higher-order_function
Overview of Java Streams

• A stream is a pipeline of aggregate operations that process a sequence of elements (aka, “values” or “data”)

Behavioral parameterization simplifies coping with changing requirements.

See blog.indrek.io/articles/java-8-behavior-parameterization
Overview of Java Streams

- A stream is a pipeline of aggregate operations that process a sequence of elements (aka, “values” or “data”)

\[
\text{Input } x \\
\xrightarrow{\text{Aggregate operation (behavior } f\text{)}} \\
\text{Output } f(x) \\
\xrightarrow{\text{Aggregate operation (behavior } g\text{)}} \\
\text{Output } g(f(x)) \\
\xrightarrow{\text{Aggregate operation (behavior } h\text{)}} \\
\]

A stream is conceptually unbounded, though it’s often bounded by practical constraints.
Overview of Java Streams

- We use this stream as a case study example throughout this introduction.

Stream
```java
.of("Ophelia","horatio", "laertes","Gertrude", "Hamlet","fortinbras", ...)
.filter(s -> toLowerCase(s.charAt(0)) == 'h')
.map(this::capitalize)
.sorted()
.forEach(System.out::println);
```

Print each character in Hamlet that starts with ‘H’ or ‘h’ in consistently capitalized & sorted order.

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex12
Benefits of Java Streams
• Java streams provide several key benefits to programs & programmers

```java
stream()
.filter(not(this::urlCached))
.map(this::downloadImage)
.flatMap(this::applyFilters)
.collect(toList())
```
Benefits of Java Streams

- Java streams provide several key benefits to programs & programmers

List of URLs to Download

List of Filters to Apply

Benefits of Java Streams

- Java streams provide several key benefits to programs & programmers, e.g.
  - **Concise & readable**
    - Declarative paradigm focuses on *what* functions to perform, not *how* to perform them

See [en.wikipedia.org/wiki/Declarative_programming](en.wikipedia.org/wiki/Declarative_programming)
Java streams provide several key benefits to programs & programmers, e.g.

- **Concise & readable**
  - Declarative paradigm focuses on *what* functions to perform, not *how* to perform them

```java
stream()
  filter(not(this::urlCached))
  map(this::downloadImage)
  flatMap(this::applyFilters)
  collect(toList())
```

See [docs.oracle.com/javase/tutorial/java/nutsandbolts/flow.html](docs.oracle.com/javase/tutorial/java/nutsandbolts/flow.html)
Java streams provide several key benefits to programs & programmers, e.g.

- Concise & readable

- Flexible & composable
  - Functions are automatically composed together

```
stream()
    .filter(not(this::urlCached))
    .map(this::downloadImage)
    .flatMap(this::applyFilters)
    .collect(toList())
```
Java streams provide several key benefits to programs & programmers, e.g.

- **Concise & readable**
- **Flexible & composable**
  - Functions are automatically composed together

**e.g., the output from filter() is passed as the input to map() etc.**
Benefits of Java Streams

- Java streams provide several key benefits to programs & programmers, e.g.
  - Concise & readable
  - Flexible & composable

- Scalable
  - Parallelize performance without the need to write any multi-threaded code

See [docs.oracle.com/javase/tutorial/collections/streams/parallelism.html](docs.oracle.com/javase/tutorial/collections/streams/parallelism.html)
Benefits of Java Streams

- Java streams provide several key benefits to programs & programmers, e.g.
  - Concise & readable
  - Flexible & composable
  - Scalable
    - Parallelize performance without the need to write any multi-threaded code

A pool of worker threads is used to process behaviors in parallel

See dzone.com/articles/common-fork-join-pool-and-streams
Benefits of Java Streams

- Java streams provide several key benefits to programs & programmers, e.g.
  - Concise & readable
  - Flexible & composable
  - Scalable
    - Parallelize performance without the need to write any multi-threaded code

Data mapped automatically to underlying processor cores

See gee.cs.oswego.edu/dl/papers/fj.pdf
End of Java Streams: Overview & Benefits