Understand Java Streams Aggregate Operations

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

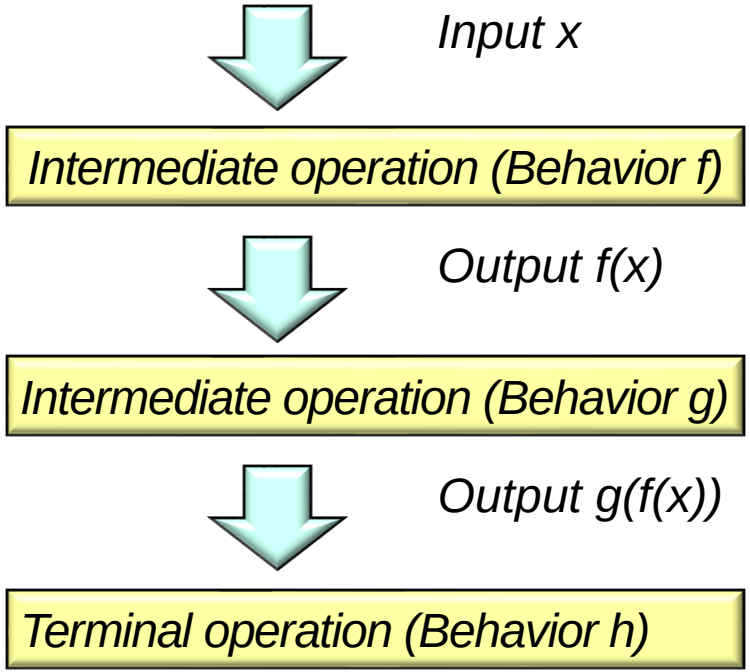
Institute for Software Integrated Systems

Vanderbilt University

Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand the structure & functionality of stream aggregate operations

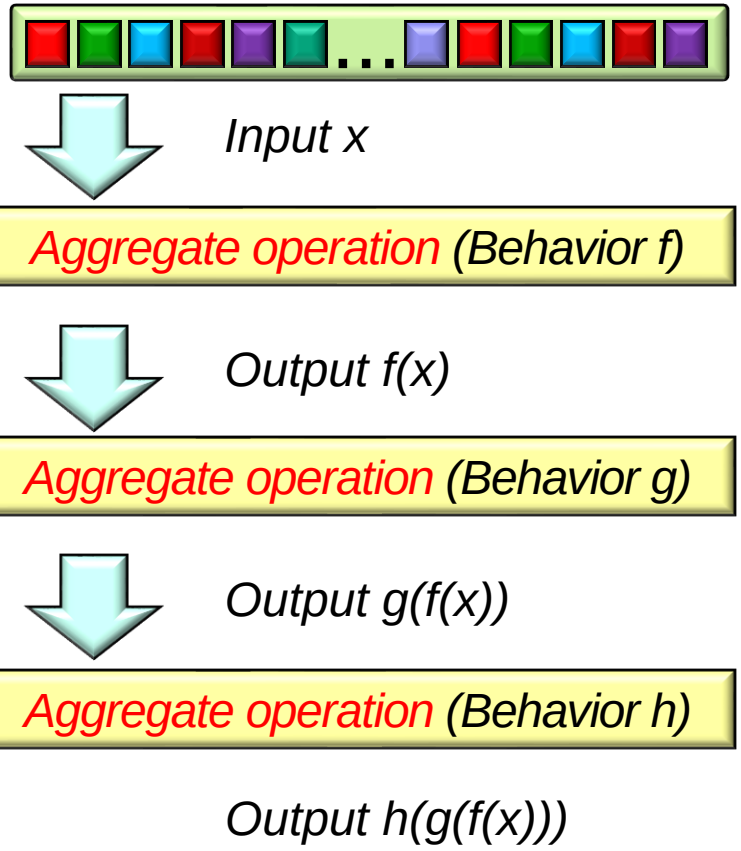


Overview of Stream Aggregate Operations

Overview of Stream Aggregate Operations

- An aggregate operation is a higher-order function that applies a "behavior" on elements in a stream

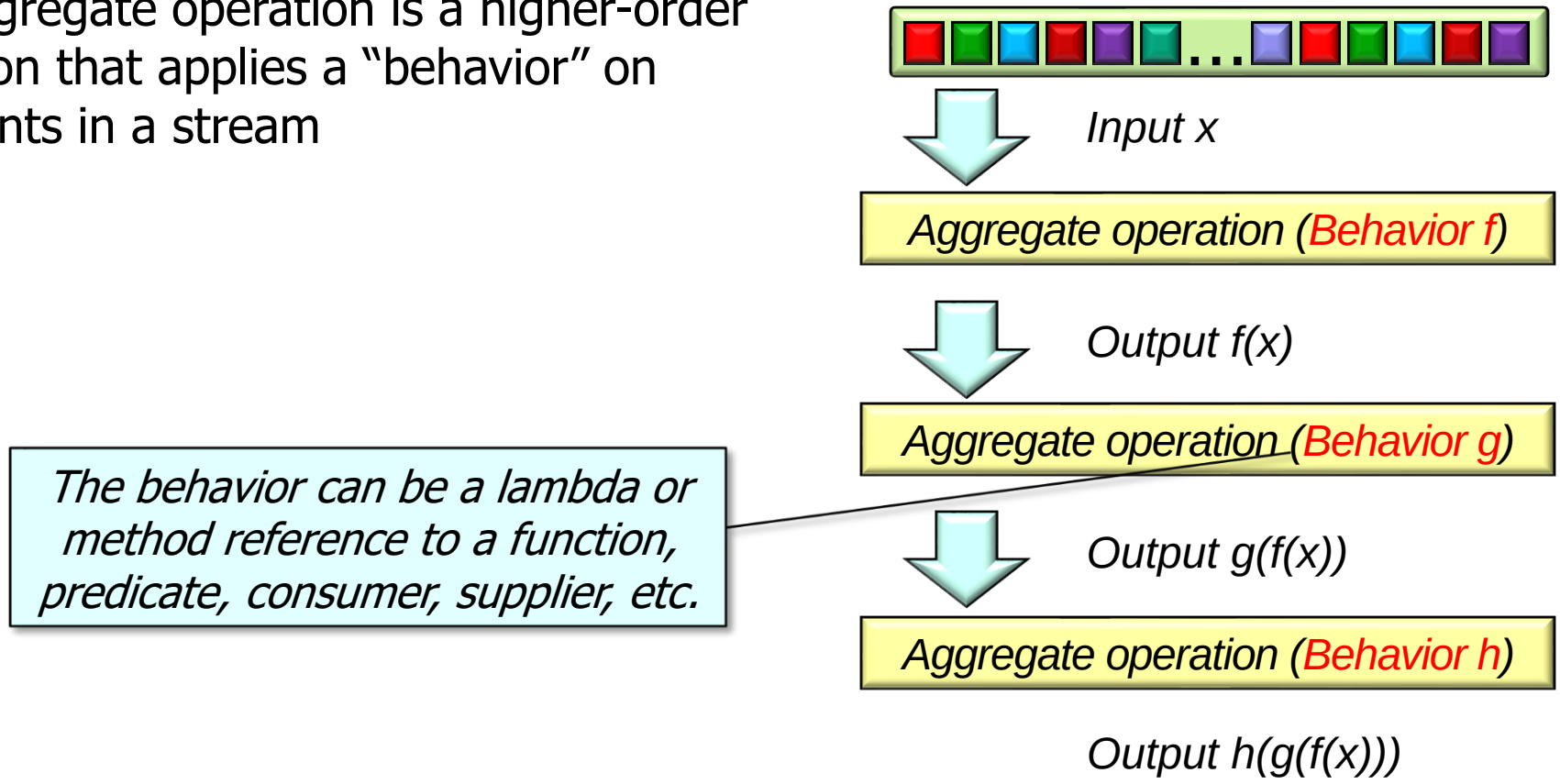
A "higher order function" is a function that is passed a function as a param



See en.wikipedia.org/wiki/Higher-order_function

Overview of Stream Aggregate Operations

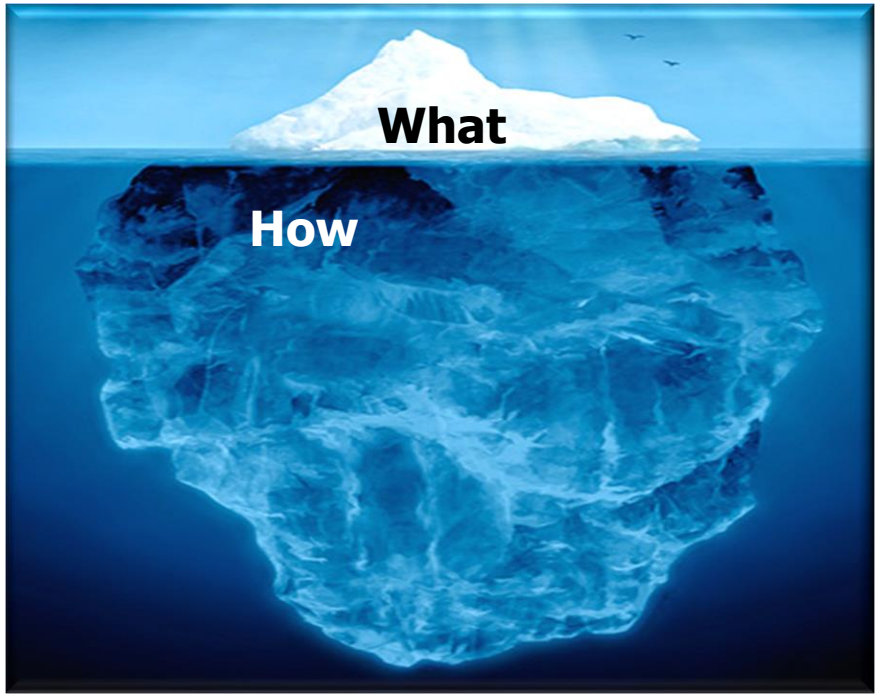
- An aggregate operation is a higher-order function that applies a "behavior" on elements in a stream

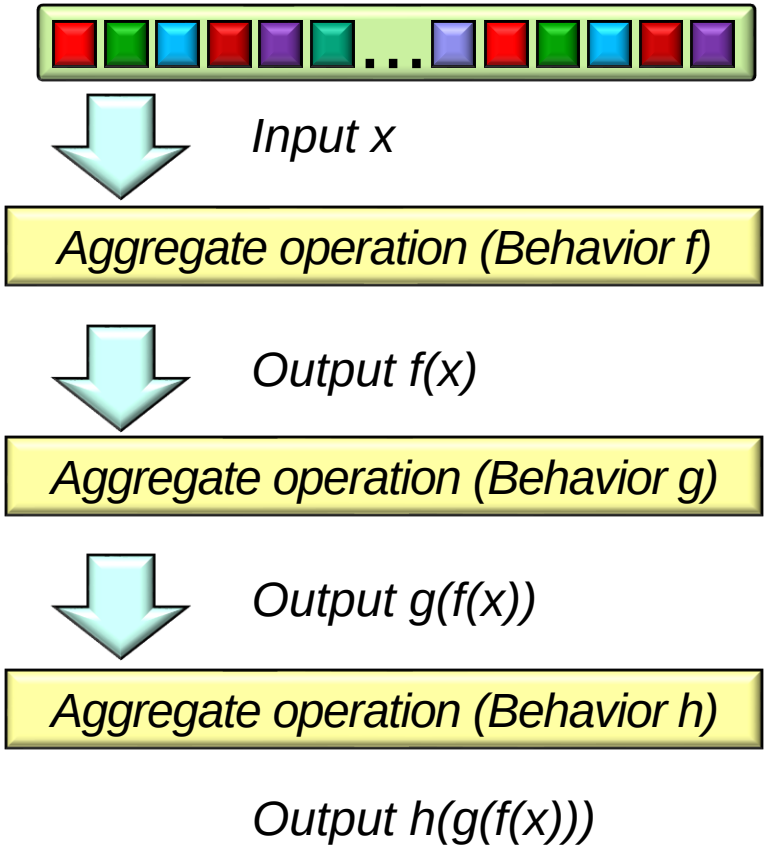


See www.drdobbs.com/jvm/lambda-expressions-in-java-8/240166764

Overview of Stream Aggregate Operations

- Aggregate operations form a declarative pipeline that emphasizes the "what" & deemphasizes the "how"

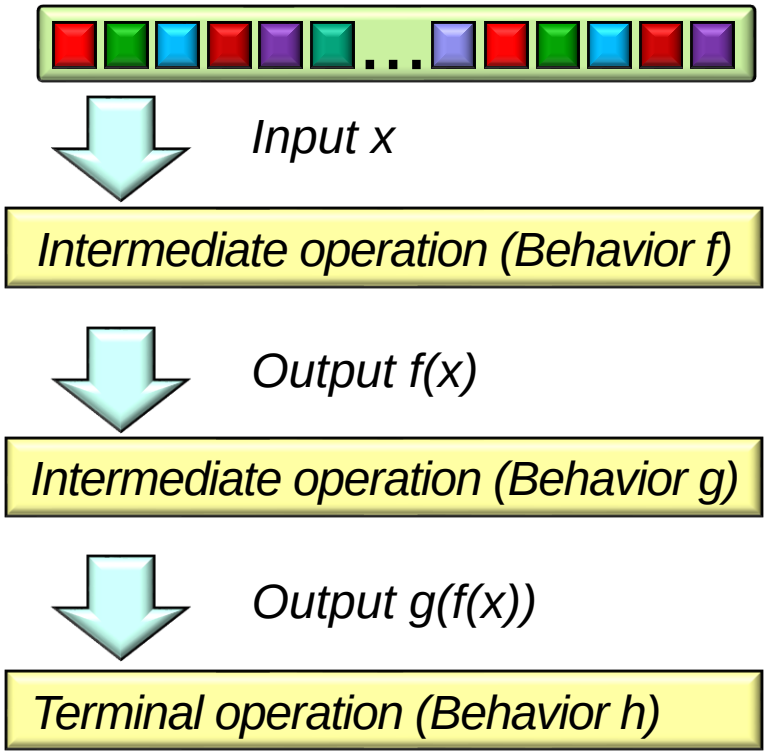




See blog.jooq.org/2015/09/17/comparing-imperative-and-functional-algorithms-in-java-8

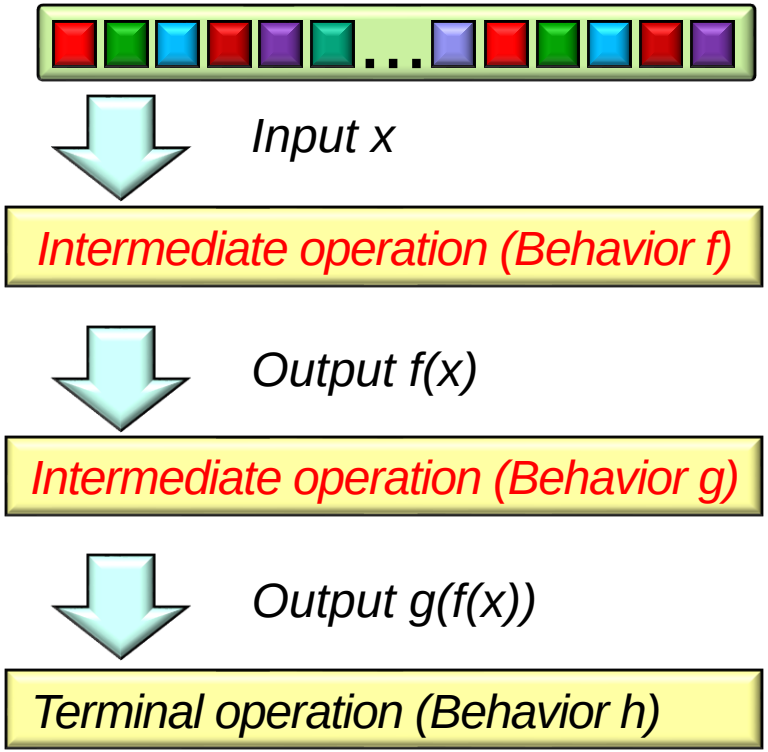
Overview of Stream Aggregate Operations

- There are two types of aggregate operations



Overview of Stream Aggregate Operations

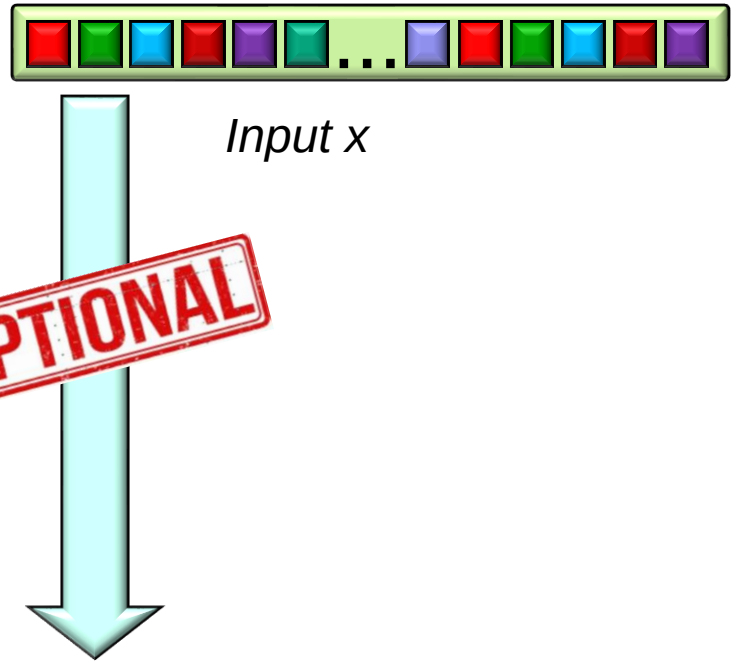
- There are two types of aggregate operations
  - **Intermediate operations**
    - Process elements in their input stream & yield an output stream
      - e.g., filter(), map(), flatMap(), takeWhile(), dropWhile(), etc.



See geekylearner.com/java-stream-intermediate-operations-learn-by-examples

Overview of Stream Aggregate Operations

- There are two types of aggregate operations
  - **Intermediate operations**
    - Process elements in their input stream & yield an output stream
      - e.g., filter(), map(), flatMap(), takeWhile(), dropWhile(), etc.



Intermediate operations are optional.

```
long HamletCharacters = Stream
  .of("horatio", "laertes",
      "Hamlet", ...)
  .count();
```

Overview of Stream Aggregate Operations

- There are two types of aggregate operations
  - **Intermediate operations**
    - Process elements in their input stream & yield an output stream
      - e.g., filter(), map(), flatMap(), takeWhile(), dropWhile(), etc.

Input x

The semantics of count() are now weird..

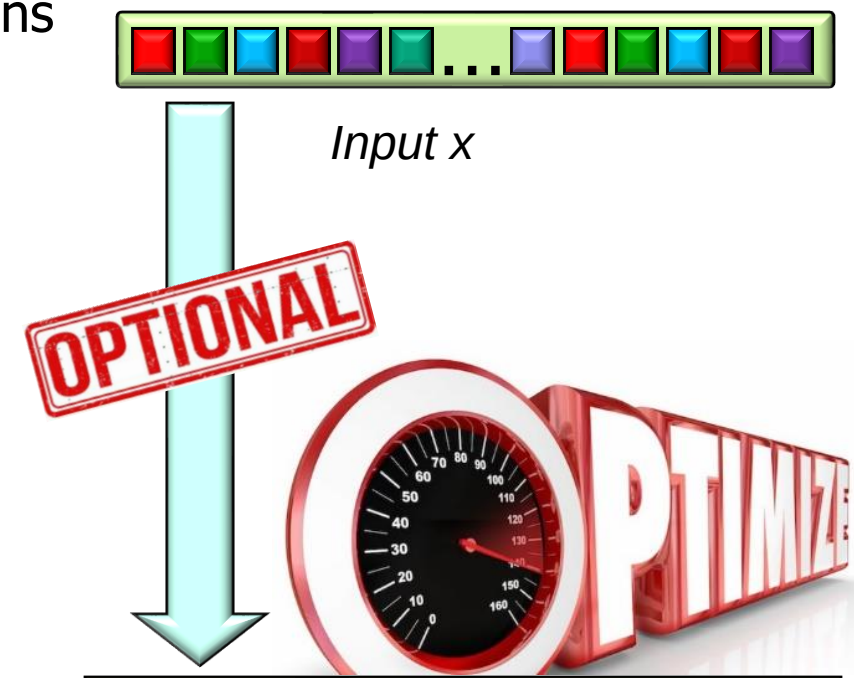
```
long HamletCharacters = Stream
   .of("horatio", "laertes",
       "Hamlet", ...)
   .peek(System.out::print)
   .count();
```

Terminal operation (Behavior h)

See mkyong.com/java8/java-8-stream-the-peek-is-not-working-with-count

Overview of Stream Aggregate Operations

- There are two types of aggregate operations
  - **Intermediate operations**
    - Process elements in their input stream & yield an output stream
      - e.g., filter(), map(), flatMap(), takeWhile(), dropWhile(), etc.




```
long HamletCharacters = Stream
   .of("horatio", "laertes",
       "Hamlet", ...)
   .count();
```

Terminal operation (Behavior h)

Newer versions of Java optimize streams containing no intermediate operations

Overview of Stream Aggregate Operations

- There are two types of aggregate operations
  - **Intermediate operations**
    - Process elements in their input stream & yield an output stream
    - Intermediate operations can be further classified via several dimensions

	Run-to-completion	Shorct-Circuiting
Stateful	distinct(), skip(), sorted()	limit(), takeWhile(), dropWhile(), etc.
Stateless	filter(), map(), flatMap(), etc.	N/A

Overview of Stream Aggregate Operations

- There are two types of aggregate operations
  - **Intermediate operations**
    - Process elements in their input stream & yield an output stream
    - Intermediate operations can be further classified via several dimensions, e.g.
      - Stateful
        - Store info from a prior invocation for use in a future invocation

	Run-to-completion	Short-Circuiting
Stateful	distinct(), skip(), sorted()	limit(), takeWhile(), dropWhile(), etc.
Stateless	filter(), map(), flatMap(), etc.	N/A

See stuartmarks.wordpress.com/2015/01/09/writing-stateful-stream-operations

Overview of Stream Aggregate Operations

- There are two types of aggregate operations
  - **Intermediate operations**
    - Process elements in their input stream & yield an output stream
    - Intermediate operations can be further classified via several dimensions, e.g.
      - Stateful
      - Stateless
        - Do not store info from any prior invocations

	Run-to-completion	Short-Circuiting
Stateful	distinct(), skip(), sorted()	limit(), takeWhile(), dropWhile(), etc.
Stateless	filter(), map(), flatMap(), etc.	N/A

See javapapers.com/java/java-stream-api

Overview of Stream Aggregate Operations

- There are two types of aggregate operations
  - **Intermediate operations**
    - Process elements in their input stream & yield an output stream
    - Intermediate operations can be further classified via several dimensions, e.g.
      - Stateful
      - Stateless
        - Do not store info from any prior invocations

	Run-to-completion	Short-Circuiting
Stateful	distinct(), skip(), sorted()	limit(), takeWhile(), dropWhile(), etc.
Stateless	filter(), map(), flatMap(), etc.	N/A

Stateless operations often require significantly fewer processing & memory resources than stateful operations!

See automationrhapsody.com/java-8-features-stream-api-explained

Overview of Stream Aggregate Operations

- There are two types of aggregate operations
  - **Intermediate operations**
    - Process elements in their input stream & yield an output stream
    - Intermediate operations can be further classified via several dimensions, e.g.
      - Stateful
      - Stateless
      - Run-to-completion
        - Process all elements in the input stream

	Run-to-completion	Short-Circuiting
Stateful	distinct(), skip(), sorted()	limit(), takeWhile(), dropWhile(), etc.
Stateless	filter(), map(), flatMap(), etc.	N/A

See en.wikipedia.org/wiki/Run_to_completion_scheduling

Overview of Stream Aggregate Operations

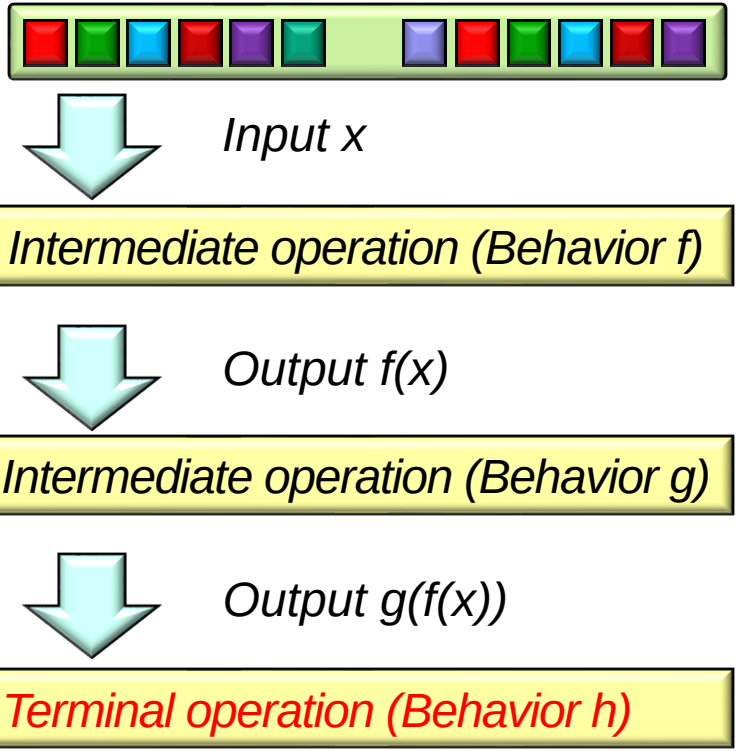
- There are two types of aggregate operations
  - **Intermediate operations**
    - Process elements in their input stream & yield an output stream
    - Intermediate operations can be further classified via several dimensions, e.g.
      - Stateful
      - Stateless
      - Run-to-completion
      - Short-circuiting
        - Make stream operate on a reduced size

	Run-to-completion	Short-Circuiting
Stateful	distinct(), skip(), sorted()	limit(), takeWhile(), dropWhile(), etc.
Stateless	filter(), map(), flatMap(), etc.	N/A

See www.logicbig.com/tutorials/core-java-tutorial/java-util-stream/short-circuiting.html

Overview of Stream Aggregate Operations

- There are two types of aggregate operations
  - Intermediate operations
  - **Terminal operations**
    - Trigger intermediate operations & produce a non-stream result
      - e.g., forEach(), reduce(), collect(), findAny(), etc.



A stream must have one (& only one) terminal operation

See www.leveluplunch.com/java/examples/stream-terminal-operations-example

Overview of Stream Aggregate Operations

- There are two types of aggregate operations
  - **Intermediate operations**
  - **Terminal operations**
    - Trigger intermediate operations & produce a non-stream result
    - Terminal operations can also be classified via several dimensions

Operation Type	Examples
Run-to-completion	reduce(), collect(), forEach(), etc.
Short-circuiting	allMatch(), anyMatch(), findAny(), findFirst(), noneMatch()

Overview of Stream Aggregate Operations

- There are two types of aggregate operations
  - **Intermediate operations**
  - **Terminal operations**
    - Trigger intermediate operations & produce a non-stream result
    - Terminal operations can also be classified via several dimensions, e.g.
      - Run-to-completion
        - Terminate only after processing all elements in the stream

Operation Type	Examples
Run-to-completion	reduce(), collect(), forEach(), etc.
Short-circuiting	allMatch(), anyMatch(), findAny(), findFirst(), noneMatch()

Overview of Stream Aggregate Operations

- There are two types of aggregate operations
  - **Intermediate operations**
  - **Terminal operations**
    - Trigger intermediate operations & produce a non-stream result
    - Terminal operations can also be classified via several dimensions, e.g.
      - Run-to-completion
      - Short-circuiting
        - May cause a stream to terminate before processing all values

Operation Type	Examples
Run-to-completion	reduce(), collect(), forEach(), etc.
Short-circuiting	allMatch(), anyMatch(), findAny(), findFirst(), noneMatch()

End of Understand Java Streams Aggregate Operations