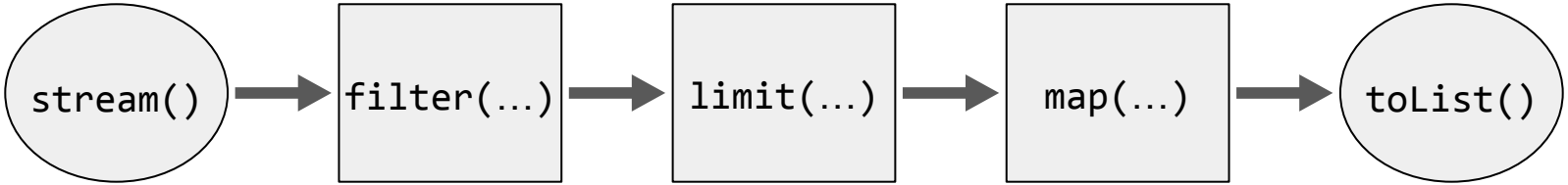
HappyCoders.eu

Stream Gatherers

Schreibe deine eigenen Stream-Operationen!

Sven Woltmann

Stream-Pipeline



Quelle

(source)

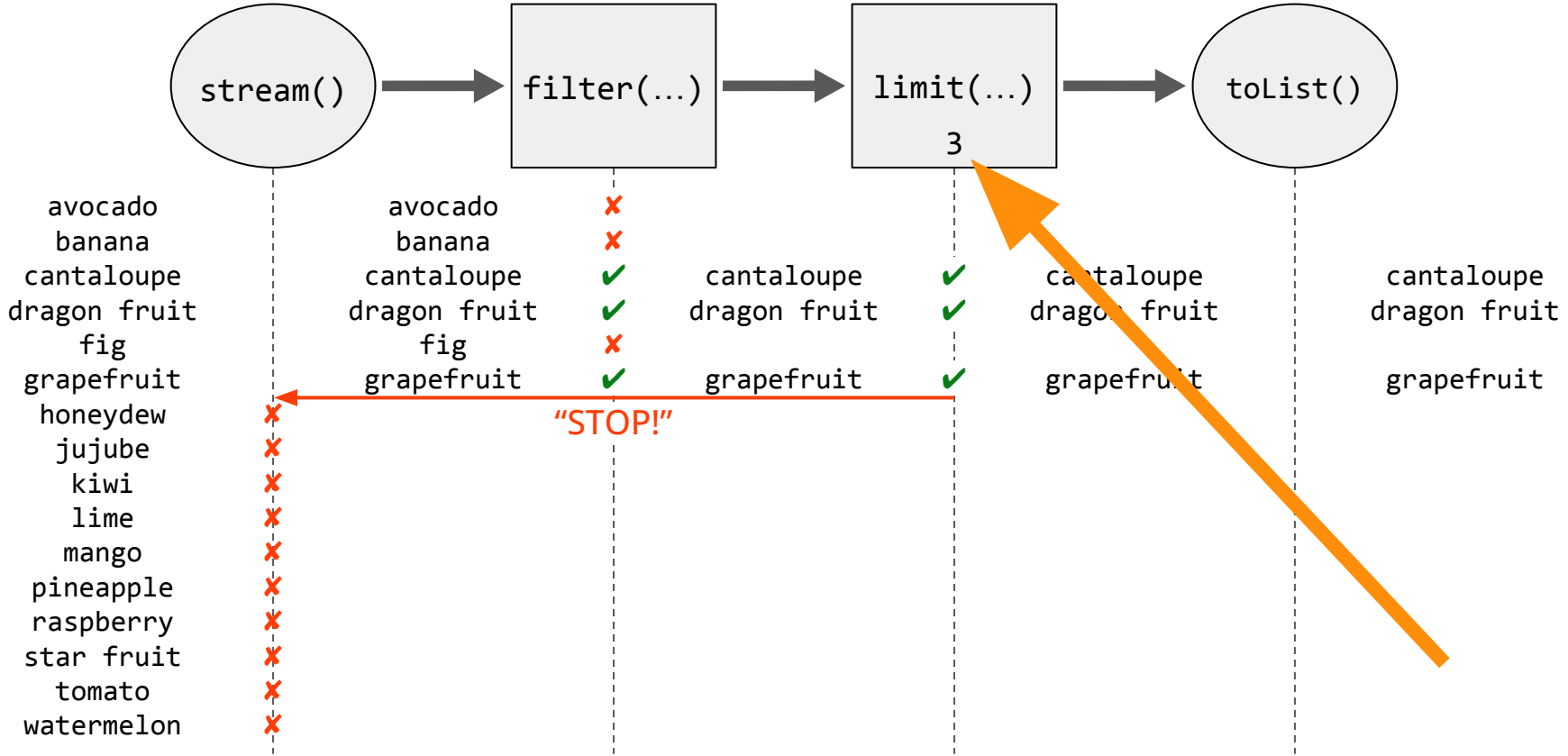
Intermediäre Operationen

(intermediate operations)

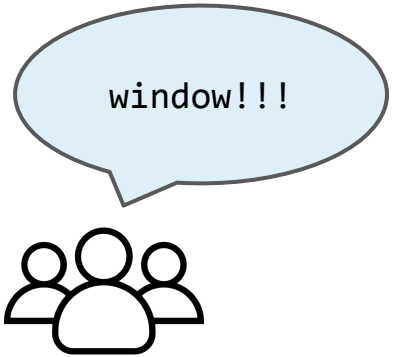
Terminale Operation

(terminal operation)

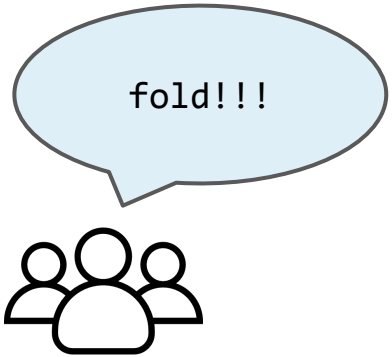
```
List<String> fruits =
    List.of("avocado", "banana", "cantaloupe",
        "dragon fruit", "fig", "grapefruit", "honeydew",
        "jujube", "kiwi", "lime", "mango", "pineapple",
        "raspberry", "star fruit", "tomato", "watermelon");

List<String> result =
   fruits.stream()
       .filter(fruit -> fruit.length() >= 8)
       .limit(3)
       .toList();
```

Intermediäre Operationen des JDK




```
filter(...)
limit(...)

map(...)
flatMap(...)
mapMulti(...)

distinct()
sorted()

skip(...)
takeWhile(...)
dropWhile(...)

peek(...)
```



Stream Gatherers

JEP 461: Stream Gatherers (Preview)

Owner	Viktor Klang
Type	Feature
Scope	SE
Status	Closed / Delivered
Release	22
Component	core-libs / java.util.stream
Discussion	core dash libs dash dev at openjdk dot org
Relates to	JEP 473: Stream Gatherers (Second Preview)
Reviewed by	Alan Bateman, Alex Buckley, Paul Sandoz
Endorsed by	Paul Sandoz
Created	2023/10/11 13:08
Updated	2024/04/17 14:09
Issue	8317955

Summary

Enhance the Stream API to support custom intermediate operations. This will allow stream pipelines to transform data in ways that are not easily achievable with the existing built-in intermediate operations. This is a preview API.

Goals

- Make stream pipelines more flexible and expressive.
- Insofar as possible, allow custom intermediate operations to manipulate streams of infinite size.

Non-Goals

- It is not a goal to change the Java programming language to better facilitate stream processing.
- It is not a goal to special-case the compilation of code that uses the Stream API.

Motivation

Java 8 introduced the first API designed specifically for lambda expressions: the Stream API, `java.util.stream`. A stream is a lazily computed, potentially unbounded sequence of values. The API supports the ability to process a stream either sequentially or in parallel.

A *stream pipeline* consists of three parts: a source of elements, any number of intermediate operations, and a terminal operation. For example:

JEP 473: Stream Gatherers (Second Preview)

Owner	Viktor Klang
Type	Feature
Scope	SE
Status	Closed / Delivered
Release	23
Component	core-libs / java.util.stream
Discussion	core dash libs dash dev at openjdk dot org
Effort	XS
Duration	XS
Relates to	JEP 461: Stream Gatherers (Preview) JEP 485: Stream Gatherers
Reviewed by	Alan Bateman, Paul Sandoz
Endorsed by	Paul Sandoz
Created	2024/03/11 19:39
Updated	2024/09/02 16:05
Issue	8327844

Summary

Enhance the Stream API to support custom intermediate operations. This will allow stream pipelines to transform data in ways that are not easily achievable with the existing built-in intermediate operations. This is a preview API.

History

We proposed Stream Gatherers as a preview feature in JEP 461 and delivered it in JDK 22. We here propose to re-preview the API in JDK 23, without change, in order to gain additional experience and feedback.

Goals

- Make stream pipelines more flexible and expressive.
- Insofar as possible, allow custom intermediate operations to manipulate streams of infinite size.

Non-Goals

- It is not a goal to change the Java programming language to better facilitate stream processing.
- It is not a goal to special-case the compilation of code that uses the Stream API.

JEP 485: Stream Gatherers

Owner	Viktor Klang
Type	Feature
Scope	SE
Status	Candidate
Component	core-libs / java.util.stream
Discussion	core dash libs dash dev at openjdk dot org
Effort	XS
Duration	XS
Relates to	JEP 473: Stream Gatherers (Second Preview)
Reviewed by	Alan Bateman, Paul Sandoz
Created	2024/07/08 15:42
Updated	2024/10/07 21:43
Issue	8335899

Summary

Enhance the Stream API to support custom intermediate operations. This will allow stream pipelines to transform data in ways that are not easily achievable with the existing built-in intermediate operations.

History

Stream Gatherers were proposed as a preview feature by JEP 461 in JDK 22 and re-previewed by JEP 473 in JDK 23. We here propose to finalize the API in JDK 24, without change.

Goals

- Make stream pipelines more flexible and expressive.
- Insofar as possible, allow custom intermediate operations to manipulate streams of infinite size.

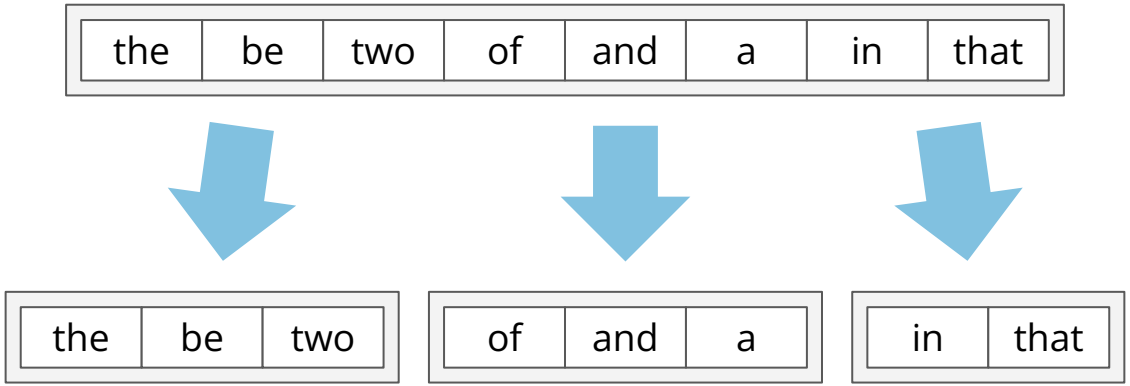
Non-Goals

- It is not a goal to change the Java programming language to better facilitate stream processing.
- It is not a goal to special-case the compilation of code that uses the Stream API.

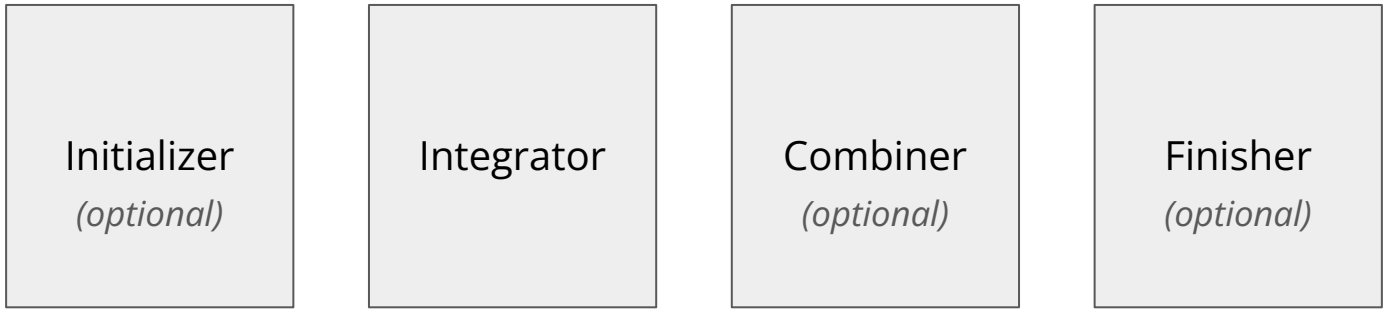
Motivation

Java 8 introduced the first API designed specifically for lambda expressions: the

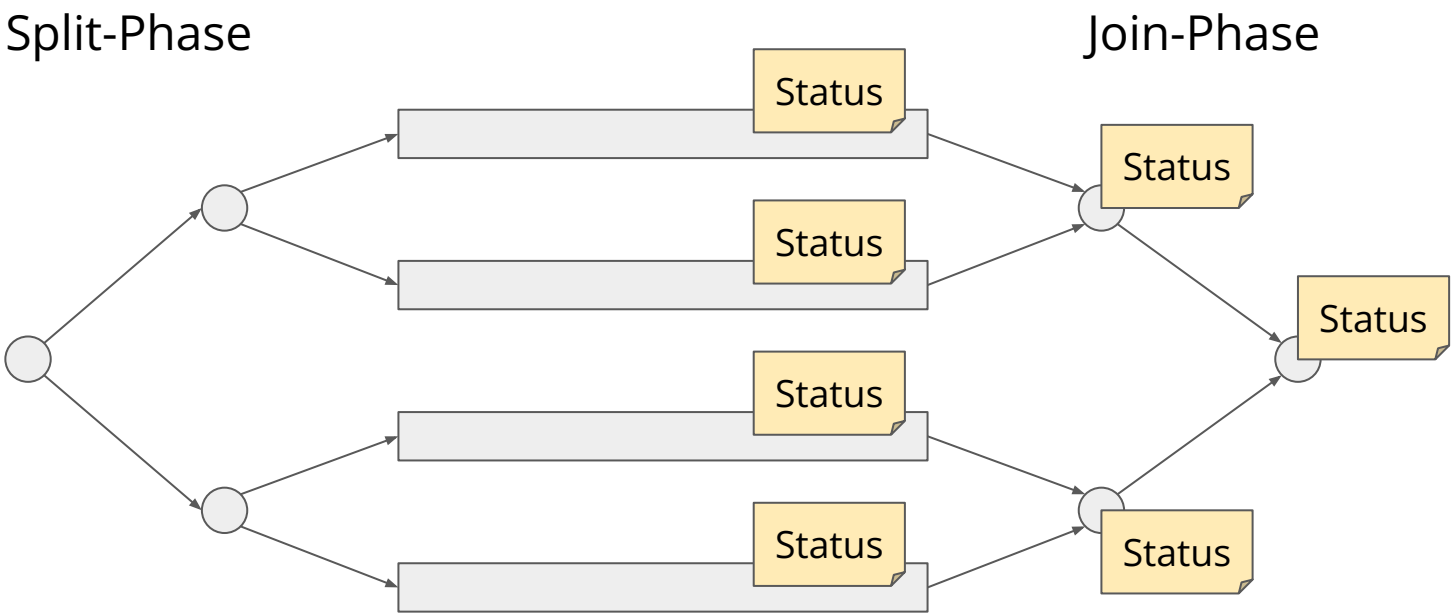
Demo



Komponenten eines Stream Gatherers



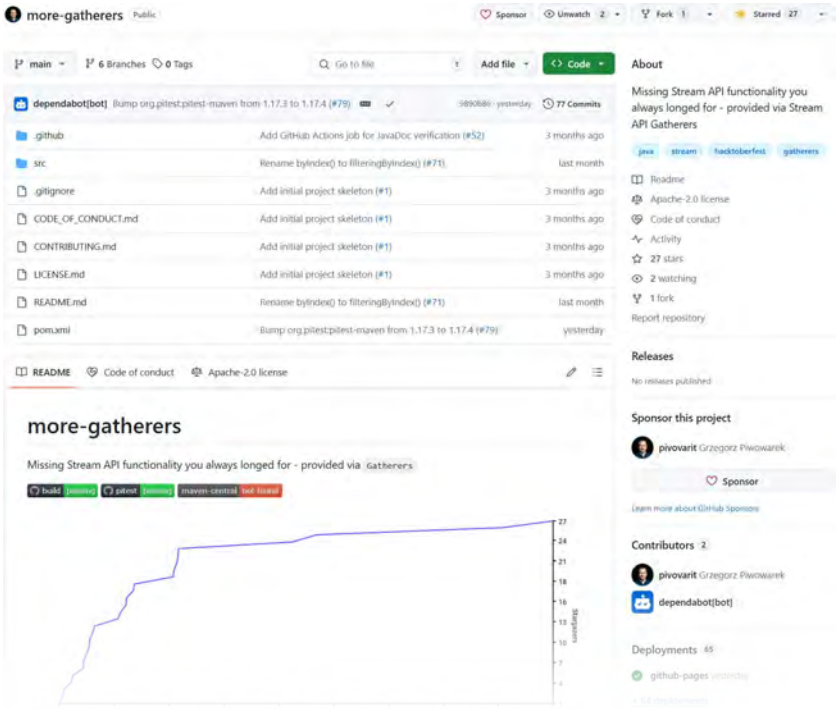
Paralleler Stream

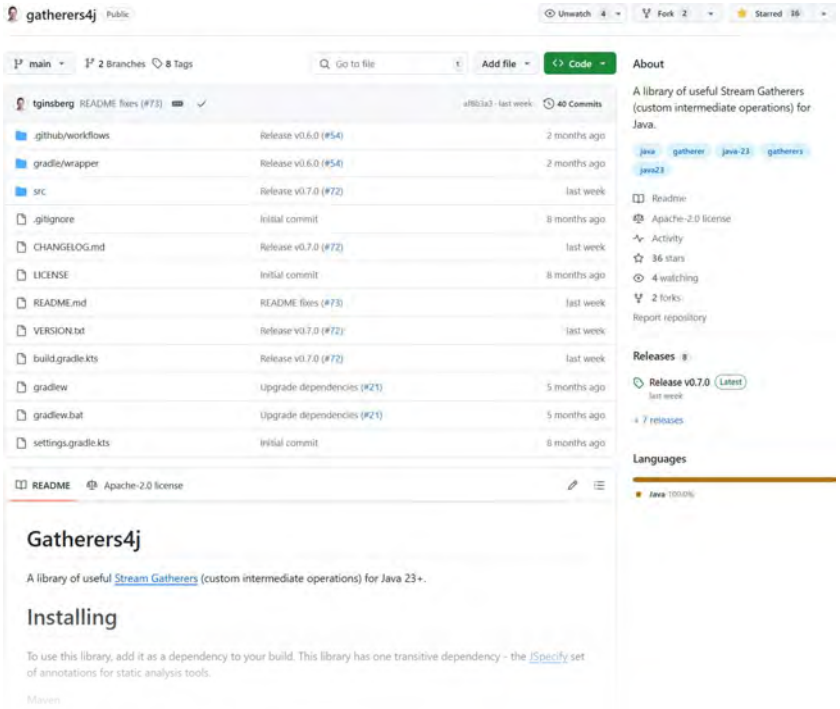


Vordefinierte Stream Gatherer

```
windowFixed(int windowSize)

windowSliding(int windowSize)

fold(Supplier<R> initial,
     BiFunction<? super R, ? super T, ? extends R> folder)

scan(Supplier<R> initial,
     BiFunction<? super R, ? super T, ? extends R> scanner)

mapConcurrent(int maxConcurrency,
              Function<? super T, ? extends R> mapper)
```


3rd Party Stream Gatherers





Performance + Grenzen

- Keine Gatherer für die primitiven Streams `IntStream`, `LongStream` und `DoubleStream`

- Kein Zugriff auf die Charakteristika des Streams,
  keine Indikation, welche Charakteristika beibehalten werden

```
int[] array = IntStream.iterate(1, i -> i + 1)
    .filter(i -> i % 7 == 0)
    .limit(3)
    .toArray();
```


```
List<String> words = . . .

long count = words.stream()
   .map(String::toUpperCase)
   .count();

long count = words.stream()
   .gather(mapping(String::toUpperCase))
   .count();
```

Schneller!

3,7 µs

861529,6 µs

Links

sven@happycoders.eu

https://www.happycoders.eu/

https://linkedin.com/in/sven-woltmann/

https://youtube.com/c/happycoders

https://x.com/svenwoltmann

https://bsky.app/profile/sven.woltmann