

Java Streams: Introducing Non-Concurrent Collectors

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 non-concurrent collectors for sequential streams

Interface `Collector<T,A,R>`

Type Parameters:

T - the type of input elements to the reduction operation

A - the mutable accumulation type of the reduction operation (often hidden as an implementation detail)

R - the result type of the reduction operation

```
public interface Collector<T,A,R>
```

A mutable reduction operation that accumulates input elements into a mutable result container, optionally transforming the accumulated result into a final representation after all input elements have been processed. Reduction operations can be performed either sequentially or in parallel.

Examples of mutable reduction operations include: accumulating elements into a `Collection`; concatenating strings using a `StringBuilder`; computing summary information about elements such as sum, min, max, or average; computing "pivot table" summaries such as "maximum valued transaction by seller", etc. The class `Collectors` provides implementations of many common mutable reductions.

A `Collector` is specified by four functions that work together to accumulate entries into a mutable result container, and optionally perform a final transform on the result. They are:

See docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html

Overview of Non-Concurrent Collectors

Overview of Non-Concurrent Collectors

- A collector is used to terminate a stream

```
void runCollectToList() {  
    List<String> characters = Arrays  
        .asList("horatio", "laertes",  
            "Hamlet, ...");
```

```
    List<String> results =  
        characters  
            .stream()  
            .filter(s ->  
                toLowerCase(...) == 'h')  
            .map(this::capitalize)  
            .sorted()  
            .collect(toList()); ...
```

Collect the results into a ArrayList

Overview of Non-Concurrent Collectors

- A collector is used to terminate a stream
- Collector defines an interface whose implementations can accumulate stream elements into mutable result containers

Interface `Collector<T,A,R>`

Type Parameters:

`T` - the type of input elements to the reduction operation

`A` - the mutable accumulation type of the reduction operation (often hidden as an implementation detail)

`R` - the result type of the reduction operation

```
public interface Collector<T,A,R>
```

A mutable reduction operation that accumulates input elements into a mutable result container, optionally transforming the accumulated result into a final representation after all input elements have been processed. Reduction operations can be performed either sequentially or in parallel.

Examples of mutable reduction operations include: accumulating elements into a `Collection`; concatenating strings using a `StringBuilder`; computing summary information about elements such as sum, min, max, or average; computing "pivot table" summaries such as "maximum valued transaction by seller", etc. The class `Collectors` provides implementations of many common mutable reductions.

A `Collector` is specified by four functions that work together to accumulate entries into a mutable result container, and optionally perform a final transform on the result. They are:

See docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html

Overview of Non-Concurrent Collectors

- Collector implementations can either be non-concurrent or concurrent based on their characteristics

Enum Collector.Characteristics

```
java.lang.Object  
    java.lang.Enum<Collector.Characteristics>  
        java.util.stream.Collector.Characteristics
```

All Implemented Interfaces:

Serializable, Comparable<Collector.Characteristics>

Enclosing Interface:

Collector<T,A,R>

```
public static enum Collector.Characteristics  
    extends Enum<Collector.Characteristics>
```

Characteristics indicating properties of a Collector, which can be used to optimize reduction implementations.

Enum Constant Summary

Enum Constants

Enum Constant and Description

CONCURRENT

Indicates that this collector is *concurrent*, meaning that the result container can support the accumulator function being called concurrently with the same result container from multiple threads.

IDENTITY_FINISH

Indicates that the finisher function is the identity function and can be elided.

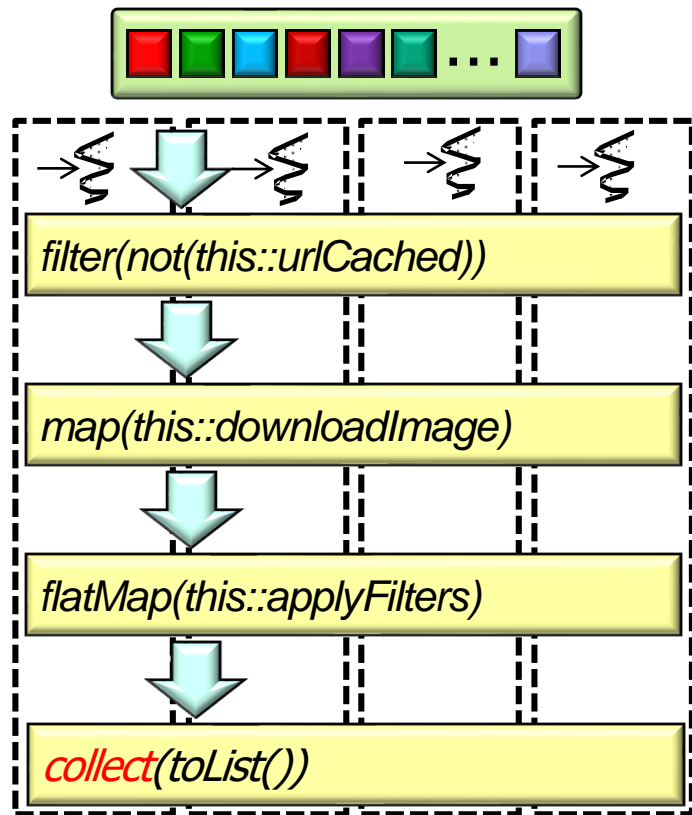
UNORDERED

Indicates that the collection operation does not commit to preserving the encounter order of input elements.

See docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.Characteristics.html

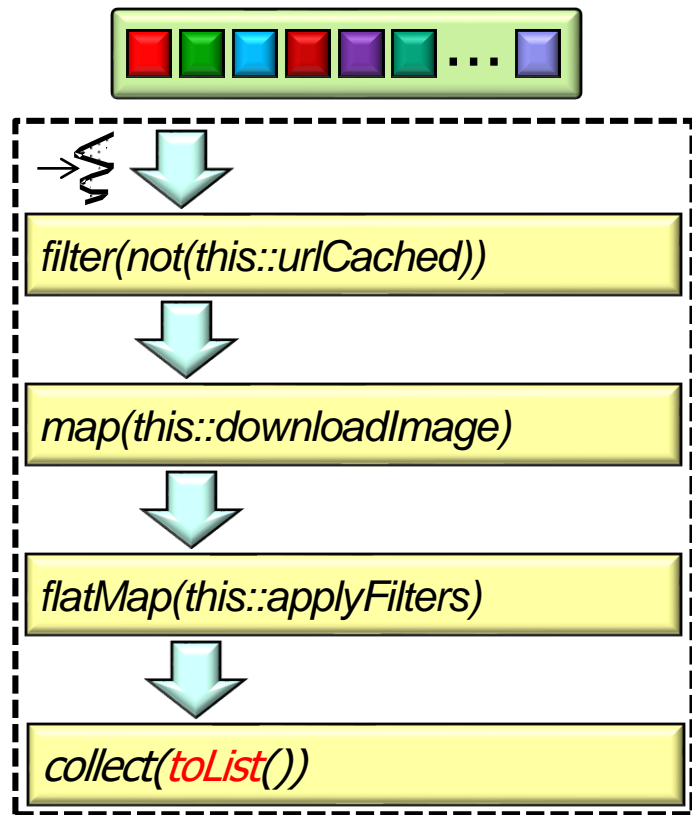
Overview of Non-Concurrent Collectors

- Collector implementations can either be non-concurrent or concurrent based on their characteristics
- This distinction is only relevant for *parallel* streams



Overview of Non-Concurrent Collectors

- Collector implementations can either be non-concurrent or concurrent based on their characteristics
 - This distinction is only relevant for *parallel* streams
- Our focus here is on non-concurrent collectors for sequential streams



Non-concurrent & concurrent collectors for parallel streams are covered later

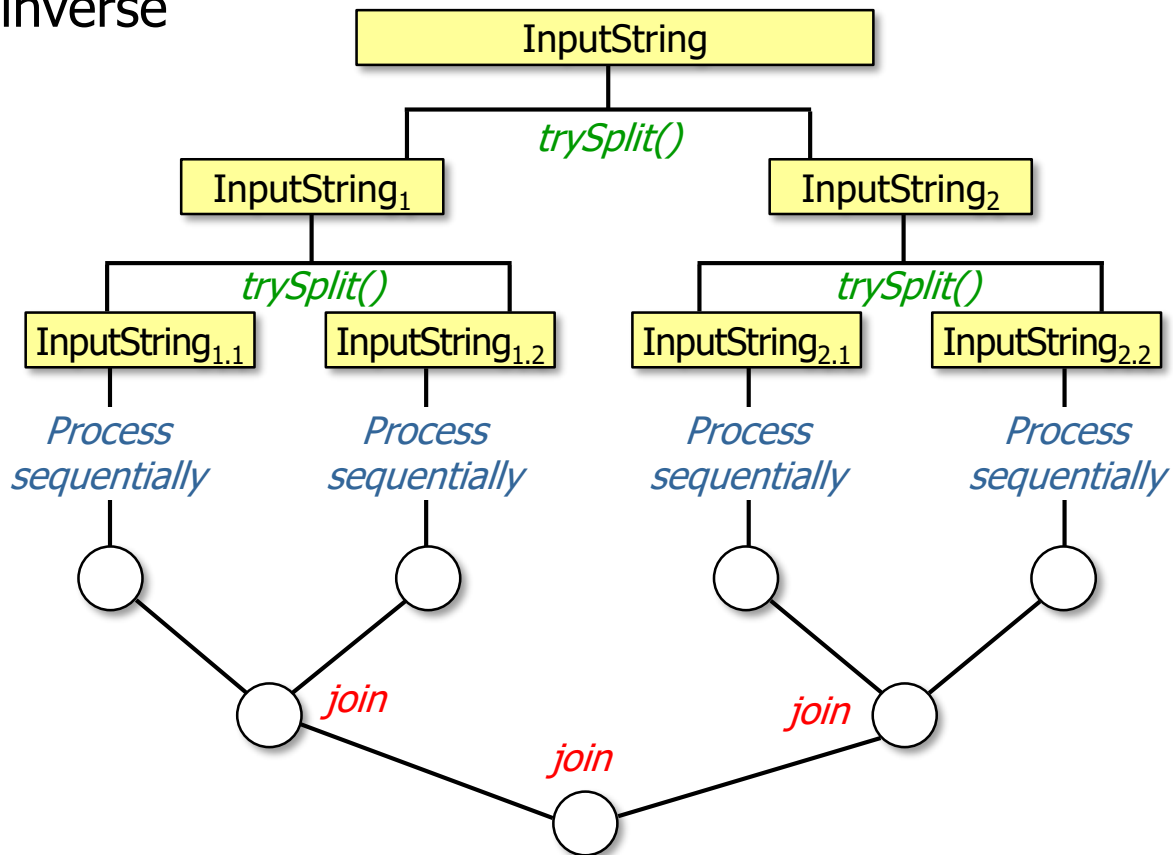
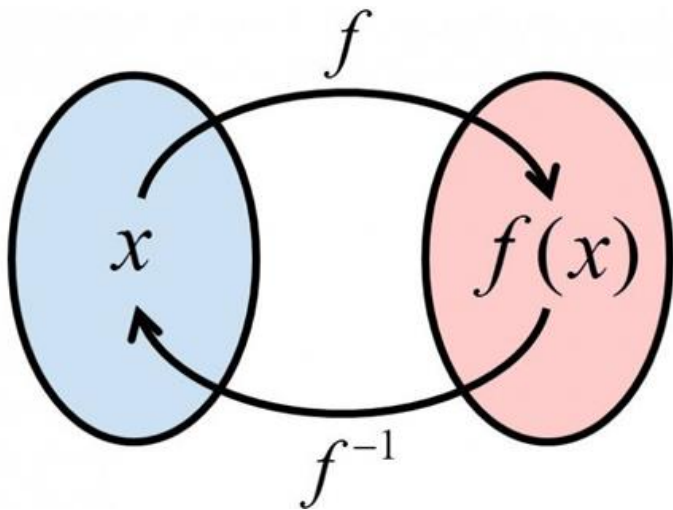
Overview of Non-Concurrent Collectors

- A non-concurrent collector for a sequential stream simply accumulates elements into a mutable result container



Overview of Non-Concurrent Collectors

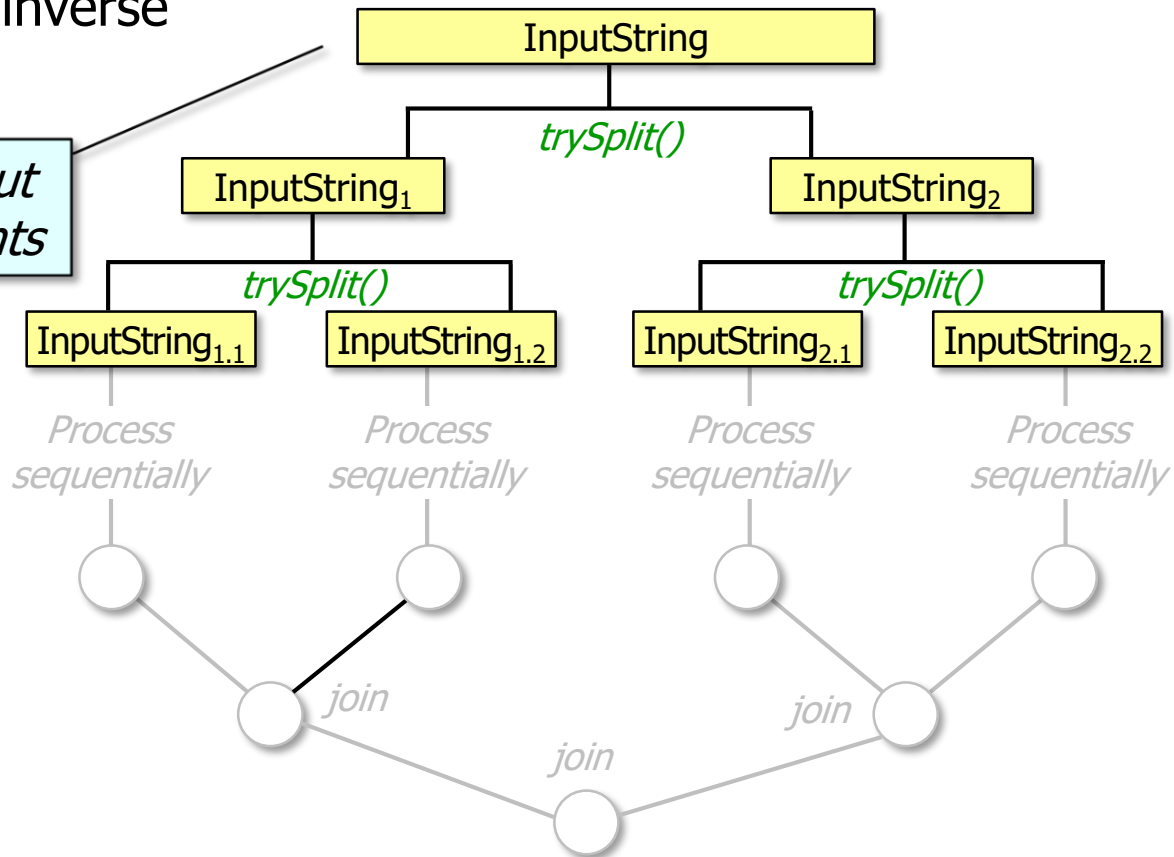
- A collector is essentially the inverse of a spliterator



Overview of Non-Concurrent Collectors

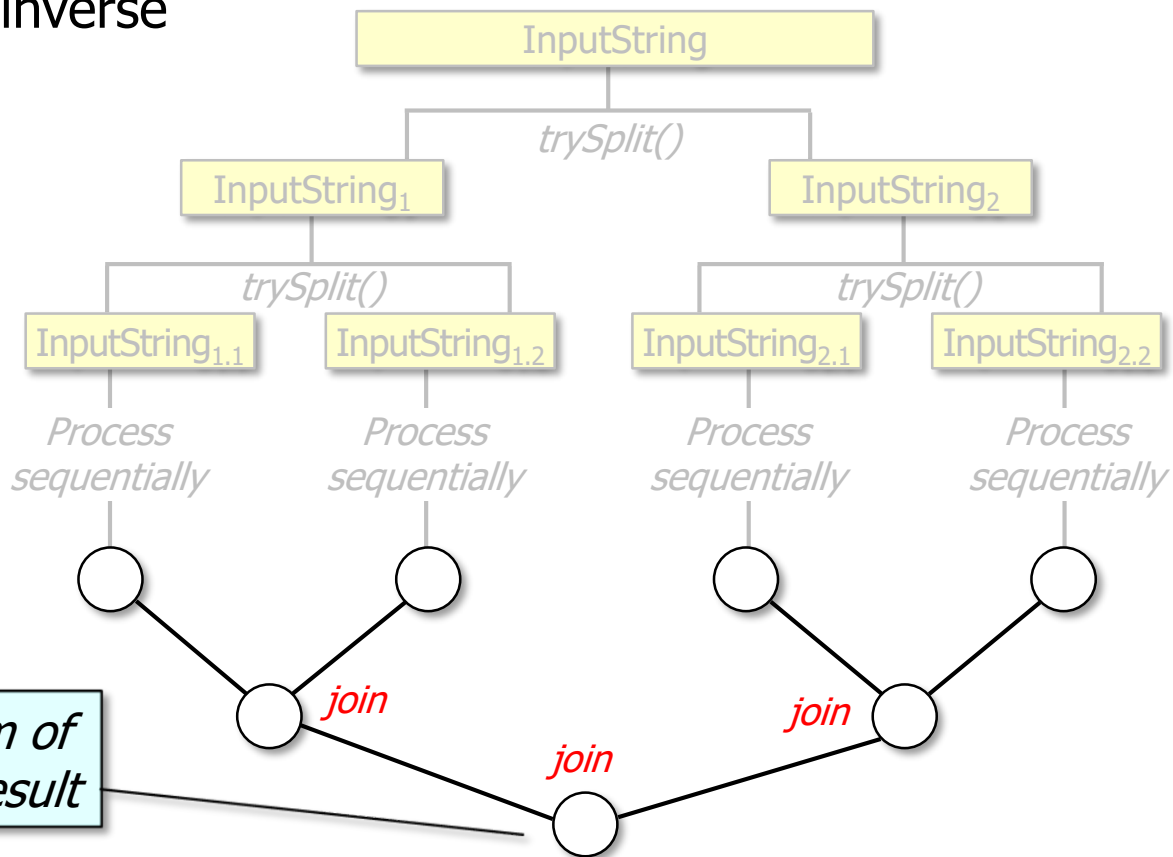
- A collector is essentially the inverse of a spliterator

A spliterator splits a single input source into a stream of elements



Overview of Non-Concurrent Collectors

- A collector is essentially the inverse of a spliterator



A collector combines a stream of elements back into a single result

End of Java Streams: Introducing Non- Concurrent Collectors