

Understand Java Streams Spliterators

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 “Splittable iterators” (Spliterators)

Interface **Spliterator**<T>

Type Parameters:

T - the type of elements returned by this Spliterator

All Known Subinterfaces:

`Spliterator.OfDouble`, `Spliterator.OfInt`, `Spliterator.OfLong`,
`Spliterator.OfPrimitive`<T,T_CONS,T_SPLITER>

All Known Implementing Classes:

`Spliterators.AbstractDoubleSpliterator`,
`Spliterators.AbstractIntSpliterator`,
`Spliterators.AbstractLongSpliterator`,
`Spliterators.AbstractSpliterator`

```
public interface Spliterator<T>
```

An object for traversing and partitioning elements of a source. The source of elements covered by a Spliterator could be, for example, an array, a `Collection`, an IO channel, or a generator function.

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

Overview of the Java Splititerator

Overview of the Java Spliterator

- A Spliterator is a new type of "splittable iterator" in Java 8

Interface Spliterator<T>

Type Parameters:

T - the type of elements returned by this Spliterator

All Known Subinterfaces:

`Spliterator.OfDouble`, `Spliterator.OfInt`, `Spliterator.OfLong`,
`Spliterator.OfPrimitive<T,T_CONS,T_SPLITR>`

All Known Implementing Classes:

`Spliterators.AbstractDoubleSpliterator`,
`Spliterators.AbstractIntSpliterator`,
`Spliterators.AbstractLongSpliterator`,
`Spliterators.AbstractSpliterator`

```
public interface Spliterator<T>
```

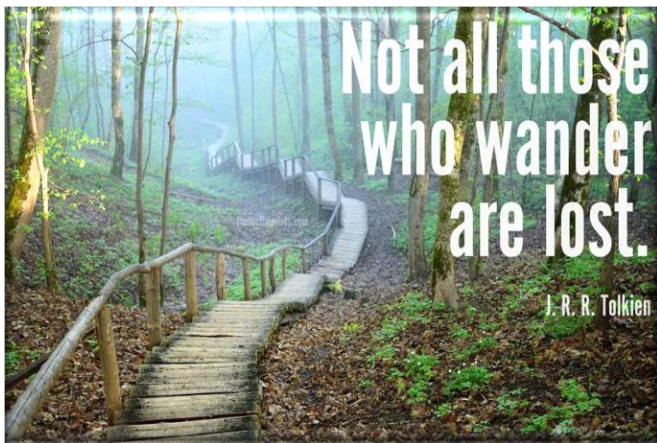
An object for traversing and partitioning elements of a source. The source of elements covered by a Spliterator could be, for example, an array, a `Collection`, an IO channel, or a generator function.

A Spliterator may traverse elements individually (`tryAdvance()`) or sequentially in bulk (`forEachRemaining()`).

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

Overview of the Java Splitterator

- A Splitterator is a new type of "splittable iterator" in Java 8
- *Iterator* – It can be used to traverse elements of a source



```
List<String> quote = List.of  
    ("This ", "above ", "all- ",  
     "to ", "thine ", "own ",  
     "self ", "be ", "true", "\n",  
     ...);
```

```
for (Splitterator<String> s =  
     quote.splitterator();  
     s.tryAdvance(System.out::print)  
     != false;  
    )  
    continue;
```

Overview of the Java Splitterator

- A Splitterator is a new type of "splittable iterator" in Java 8
- *Iterator* – It can be used to traverse elements of a source
 - e.g., a collection, array, etc.

```
List<String> quote = List.of  
    ("This ", "above ", "all- ",  
     "to ", "thine ", "own ",  
     "self ", "be ", "true", "\n",  
     ...);  
  
for (Splitterator<String> s =  
     quote.splitterator();  
     s.tryAdvance(System.out::print)  
     != false;  
     )  
    continue;
```

Overview of the Java Splitterator

- A Splitterator is a new type of "splittable iterator" in Java 8
- *Iterator* – It can be used to traverse elements of a source
 - e.g., a collection, array, etc.

This source is an array/list of strings

```
List<String> quote = List.of  
    ("This ", "above ", "all- ",  
     "to ", "thine ", "own ",  
     "self ", "be ", "true", "\n",  
     ...);  
  
for (Splitterator<String> s =  
     quote.splitterator();  
     s.tryAdvance(System.out::print)  
     != false;  
     )  
    continue;
```

Overview of the Java Splitterator

- A Splitterator is a new type of "splittable iterator" in Java 8
- *Iterator* – It can be used to traverse elements of a source
 - e.g., a collection, array, etc.

```
List<String> quote = List.of  
    ("This ", "above ", "all- ",  
     "to ", "thine ", "own ",  
     "self ", "be ", "true", "\n",  
     ...);  
  
for (Splitterator<String> s =  
     quote.splitterator();  
     s.tryAdvance(System.out::print)  
     != false;  
     )  
    continue;
```

*Create a splitterator for
the entire array/list*

Overview of the Java Splitterator

- A Splitterator is a new type of "splittable iterator" in Java 8
- *Iterator* – It can be used to traverse elements of a source
 - e.g., a collection, array, etc.

```
List<String> quote = List.of  
    ("This ", "above ", "all- ",  
     "to ", "thine ", "own ",  
     "self ", "be ", "true", "\n",  
     ...);  
  
for (Splitterator<String> s =  
     quote.splitterator();  
     s.tryAdvance(System.out::print)  
     != false;  
     )  
    continue;
```

*tryAdvance() combines
the hasNext() & next()
methods of Iterator*

Overview of the Java Splitterator

- A Splitterator is a new type of "splittable iterator" in Java 8
- *Iterator* – It can be used to traverse elements of a source
 - e.g., a collection, array, etc.

```
boolean tryAdvance(Consumer  
    <? super T> action) {  
    if (noMoreElementsRemain)  
        return false;  
    else { ...  
        action.accept  
            (nextElement);  
        return true;  
    }  
}
```

```
List<String> quote = List.of  
    ("This ", "above ", "all- ",  
     "to ", "thine ", "own ",  
     "self ", "be ", "true", "\n",  
     ...);  
  
for (Splitterator<String> s =  
    quote.splitterator();  
    s.tryAdvance(System.out::print)  
    != false;  
    )  
    continue;
```

Overview of the Java Splitterator

- A Splitterator is a new type of "splittable iterator" in Java 8
- *Iterator* – It can be used to traverse elements of a source
 - e.g., a collection, array, etc.

```
boolean tryAdvance(Consumer  
    <? super T> action) {  
    if (noMoreElementsRemain)  
        return false;  
    else { ...  
        action.accept  
            (nextElement);  
        return true;  
    }  
}
```

```
List<String> quote = List.of  
    ("This ", "above ", "all- ",  
     "to ", "thine ", "own ",  
     "self ", "be ", "true", "\n",  
     ...);
```

```
for (Splitterator<String> s =  
    quote.splitterator();  
    s.tryAdvance(System.out::print)  
    != false;  
    )  
    continue;
```

Overview of the Java Splitterator

- A Splitterator is a new type of "splittable iterator" in Java 8
- *Iterator* – It can be used to traverse elements of a source
 - e.g., a collection, array, etc.

```
boolean tryAdvance(Consumer  
    <? super T> action) {  
    if (noMoreElementsRemain)  
        return false;  
    else { ...  
        action.accept  
            (nextElement);  
        return true;  
    }  
}
```

```
List<String> quote = List.of  
    ("This ", "above ", "all- ",  
     "to ", "thine ", "own ",  
     "self ", "be ", "true", "\n",  
     ...);
```

```
for (Splitterator<String> s =  
    quote.splitterator();  
    s.tryAdvance(System.out::print)  
    != false;  
    )  
    continue;
```

Overview of the Java Splitterator

- A Splitterator is a new type of "splittable iterator" in Java 8
- *Iterator* – It can be used to traverse elements of a source
 - e.g., a collection, array, etc.

```
List<String> quote = List.of  
    ("This ", "above ", "all- ",  
     "to ", "thine ", "own ",  
     "self ", "be ", "true", "\n",  
     ...);  
  
for (Splitterator<String> s =  
     quote.splitterator();  
     s.tryAdvance(System.out::print)  
     != false;  
    )  
    continue;
```

*Print value of each
string in the quote*

Overview of the Java Splitter

- A Splitter is a new type of "splittable iterator" in Java 8
 - *Iterator* – It can be used to traverse elements of a source
 - *Split* – It can also partition all elements of a source



```
List<String> quote = List.of  
    ("This ", "above ", "all- ",  
     "to ", "thine ", "own ",  
     "self ", "be ", "true", "\n",  
     ...);
```

```
Splitter<String> secondHalf =  
    quote.splitter();  
Splitter<String> firstHalf =  
    secondHalf.trySplit();
```

```
firstHalf.forEachRemaining  
    (System.out::print);  
secondHalf.forEachRemaining  
    (System.out::print);
```

Overview of the Java Splitter

- A Splitter is a new type of "splittable iterator" in Java 8
 - *Iterator* – It can be used to traverse elements of a source
 - *Split* – It can also partition all elements of a source

Create a splitter for the entire array/list

```
List<String> quote = List.of  
    ("This ", "above ", "all- ",  
     "to ", "thine ", "own ",  
     "self ", "be ", "true", "\n",  
     ...);  
  
Splitter<String> secondHalf =  
    quote.splitter();  
Splitter<String> firstHalf =  
    secondHalf.trySplit();  
  
firstHalf.forEachRemaining  
    (System.out::print);  
secondHalf.forEachRemaining  
    (System.out::print);
```

Overview of the Java Splitter

- A Splitter is a new type of "splittable iterator" in Java 8
 - *Iterator* – It can be used to traverse elements of a source
 - *Split* – It can also partition all elements of a source

```
List<String> quote = List.of  
    ("This ", "above ", "all- ",  
     "to ", "thine ", "own ",  
     "self ", "be ", "true", "\n",  
     ...);  
  
Splitter<String> secondHalf =  
    quote.splitter();  
Splitter<String> firstHalf =  
    secondHalf.trySplit();  
  
firstHalf.forEachRemaining  
    (System.out::print);  
secondHalf.forEachRemaining  
    (System.out::print);
```

trySplit() returns a splitter covering elements that will no longer be covered by the invoking splitter

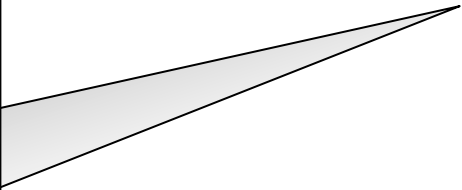
Overview of the Java Splitter

- A Splitter is a new type of "splittable iterator" in Java 8
 - *Iterator* – It can be used to traverse elements of a source
 - *Split* – It can also partition all elements of a source

```
Splitter<T> trySplit() {  
    if (input <= minimum size)  
        return null  
    else {  
        split input in 2 chunks  
        update "right chunk"  
        return splitter  
            for "left chunk"  
    }  
}
```

```
List<String> quote = List.of  
    ("This ", "above ", "all- ",  
     "to ", "thine ", "own ",  
     "self ", "be ", "true", "\n",  
     ...);
```

```
Splitter<String> secondHalf =  
    quote.splitter();  
Splitter<String> firstHalf =  
    secondHalf.trySplit();
```



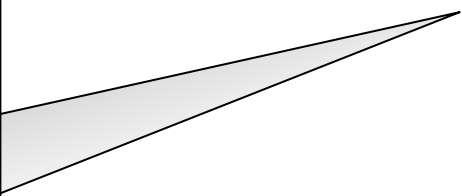
Overview of the Java Splitter

- A Splitter is a new type of "splittable iterator" in Java 8
 - *Iterator* – It can be used to traverse elements of a source
 - *Split* – It can also partition all elements of a source

```
Splitter<T> trySplit() {  
    if (input <= minimum size)  
        return null  
    else {  
        split input in 2 chunks  
        update "right chunk"  
        return splitter  
        for "left chunk"  
    }  
}
```

```
List<String> quote = List.of  
    ("This ", "above ", "all- ",  
     "to ", "thine ", "own ",  
     "self ", "be ", "true", "\n",  
     ...);
```

```
Splitter<String> secondHalf =  
    quote.splitter();  
Splitter<String> firstHalf =  
    secondHalf.trySplit();
```



trySplit() calls itself recursively until all chunks are <= to the minimize size

Overview of the Java Splitterator

- A Splitterator is a new type of "splittable iterator" in Java 8
 - *Iterator* – It can be used to traverse elements of a source
 - *Split* – It can also partition all elements of a source

```
Splitterator<T> trySplit() {  
    if (input <= minimum size)  
        return null  
    else {  
        split input in 2 chunks  
        update "right chunk"  
        return splitterator  
            for "left chunk"  
    }  
}
```

```
List<String> quote = List.of  
    ("This ", "above ", "all- ",  
     "to ", "thine ", "own ",  
     "self ", "be ", "true", "\n",  
     ...);
```

```
Splitterator<String> secondHalf =  
    quote.splitterator();  
Splitterator<String> firstHalf =  
    secondHalf.trySplit();
```



Ideally, a splitterator efficiently splits the original input source in half!

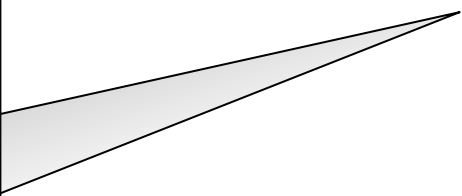
Overview of the Java Splitter

- A Splitter is a new type of "splittable iterator" in Java 8
 - *Iterator* – It can be used to traverse elements of a source
 - *Split* – It can also partition all elements of a source

```
Splitter<T> trySplit() {  
    if (input <= minimum size)  
        return null  
    else {  
        split input in 2 chunks  
        update "right chunk"  
        return splitter  
        for "left chunk"  
    }  
}
```

```
List<String> quote = List.of  
    ("This ", "above ", "all- ",  
     "to ", "thine ", "own ",  
     "self ", "be ", "true", "\n",  
     ...);
```

```
Splitter<String> secondHalf =  
    quote.splitter();  
Splitter<String> firstHalf =  
    secondHalf.trySplit();
```



The "right chunk" is defined by updating the state of `this` splitter object

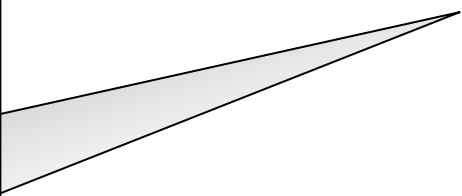
Overview of the Java Splitter

- A Splitter is a new type of "splittable iterator" in Java 8
 - *Iterator* – It can be used to traverse elements of a source
 - *Split* – It can also partition all elements of a source

```
Splitter<T> trySplit() {  
    if (input <= minimum size)  
        return null  
    else {  
        split input in 2 chunks  
        update "right chunk"  
        return splitter  
        for "left chunk"  
    }  
}
```

```
List<String> quote = List.of  
    ("This ", "above ", "all- ",  
     "to ", "thine ", "own ",  
     "self ", "be ", "true", "\n",  
     ...);
```

```
Splitter<String> secondHalf =  
    quote.splitter();  
Splitter<String> firstHalf =  
    secondHalf.trySplit();
```



The "left chunk" is defined by creating/returning a new splitter object

Overview of the Java Splitterator

- A Splitterator is a new type of "splittable iterator" in Java 8
 - *Iterator* – It can be used to traverse elements of a source
 - *Split* – It can also partition all elements of a source

Performs the action for each element in the splitterator

```
List<String> quote = List.of  
    ("This ", "above ", "all- ",  
     "to ", "thine ", "own ",  
     "self ", "be ", "true", "\n",  
     ...);  
  
Splitterator<String> secondHalf =  
    quote.splitterator();  
Splitterator<String> firstHalf =  
    secondHalf.trySplit();  
  
firstHalf.forEachRemaining  
    (System.out::print);  
secondHalf.forEachRemaining  
    (System.out::print);
```

Overview of the Java Splitter

- A Splitter is a new type of "splittable iterator" in Java 8
 - *Iterator* – It can be used to traverse elements of a source
 - *Split* – It can also partition all elements of a source

```
List<String> quote = List.of  
    ("This ", "above ", "all- ",  
     "to ", "thine ", "own ",  
     "self ", "be ", "true", "\n",  
     ...);
```

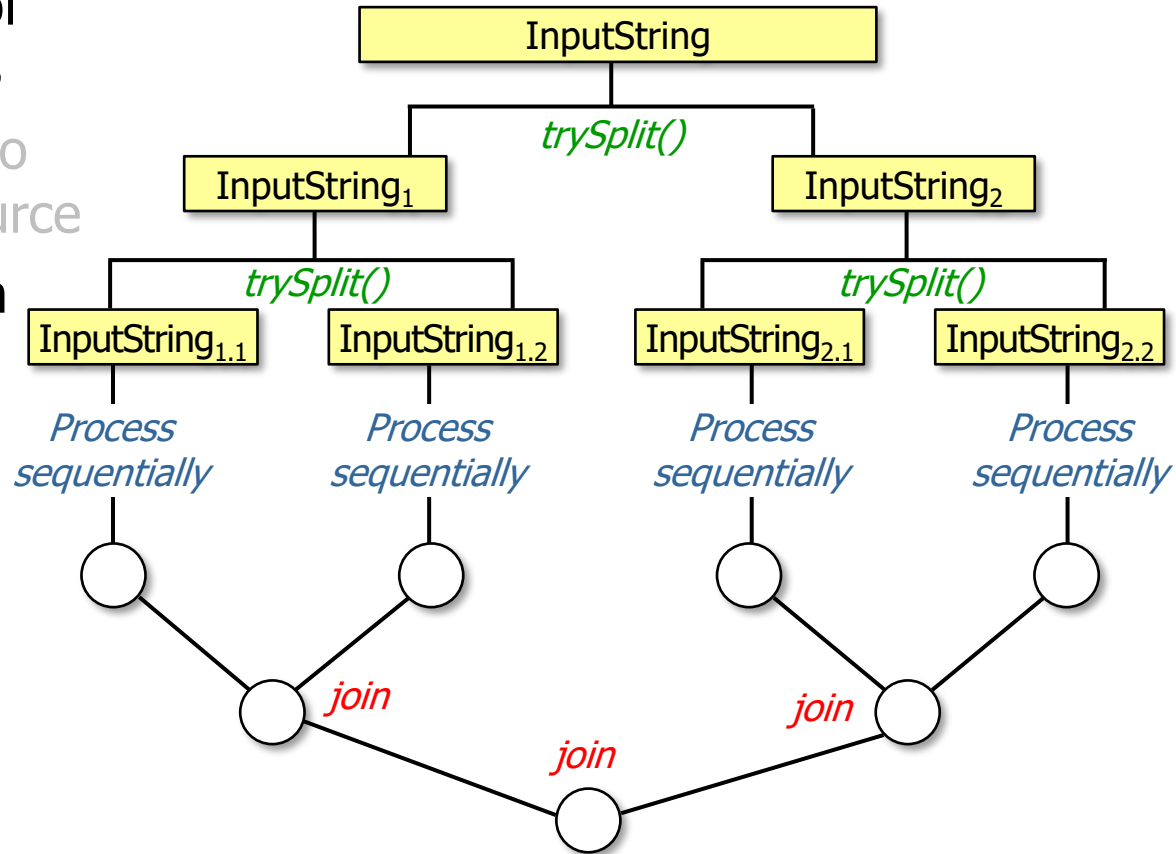
```
Splitter<String> secondHalf =  
    quote.splitter();  
Splitter<String> firstHalf =  
    secondHalf.trySplit();
```

*Print value of each
string in the quote*

```
firstHalf.forEachRemaining  
    (System.out::print);  
secondHalf.forEachRemaining  
    (System.out::print);
```

Overview of the Java Splitter

- A Splitter is a new type of "splittable iterator" in Java 8
 - *Iterator* – It can be used to traverse elements of a source
 - *Split* – It can also partition all elements of a source
 - Mostly used with Java parallel streams



Overview of the Java Splitter

- A Splitter is a new type of "splittable iterator" in Java 8
- *Iterator* – It can be used to traverse elements of a source
- *Split* – It can also partition all elements of a source



Interface `Splitter<T>`

Type Parameters:

`T` - the type of elements returned by this Splitter

All Known Subinterfaces:

`Splitter.OfDouble`, `Splitter.OfInt`, `Splitter.OfLong`,
`Splitter.OfPrimitive<T,T_CONS,T_SPLITR>`

All Known Implementing Classes:

`Splitters.AbstractDoubleSplitter`,
`Splitters.AbstractIntSplitter`,
`Splitters.AbstractLongSplitter`,
`Splitters.AbstractSplitter`

```
public interface Splitter<T>
```

An object for traversing and partitioning elements of a source. The source of elements covered by a Splitter could be, for example, an array, a `Collection`, an IO channel, or a generator function.

A Splitter may traverse elements individually (`tryAdvance()`) or sequentially in bulk (`forEachRemaining()`).

We focus on traversal now & on partitioning later when covering parallel streams

Overview of the Java Spliterator

- The `StreamSupport.stream()` factory method creates a new sequential or parallel stream from a spliterator

stream

```
public static <T> Stream<T> stream(Spliterator<T> spliterator,  
                                   boolean parallel)
```

Creates a new sequential or parallel `Stream` from a `Spliterator`.

The spliterator is only traversed, split, or queried for estimated size after the terminal operation of the stream pipeline commences.

It is strongly recommended the spliterator report a characteristic of `IMMUTABLE` or `CONCURRENT`, or be late-binding. Otherwise, `stream(java.util.function.Supplier, int, boolean)` should be used to reduce the scope of potential interference with the source. See [Non-Interference](#) for more details.

Type Parameters:

`T` - the type of stream elements

Parameters:

`spliterator` - a `Spliterator` describing the stream elements

`parallel` - if true then the returned stream is a parallel stream; if false the returned stream is a sequential stream.

Returns:

a new sequential or parallel `Stream`

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

Overview of the Java Splitterator

- The `StreamSupport.stream()` factory method creates a new sequential or parallel stream from a splitterator
- e.g., the `Collection` interface defines two default methods using this capability

```
public interface Collection<E>
    extends Iterable<E> {

    ...

    default Stream<E> stream() {
        return StreamSupport
            .stream(splitterator(),
                false);
    }

    default Stream<E>
        parallelStream() {
        return StreamSupport
            .stream(splitterator(),
                true);
    }
}
```

See jdk8/jdk8/jdk/file/tip/src/share/classes/java/util/Collection.java

Overview of the Java Splitterator

- The `StreamSupport.stream()` factory method creates a new sequential or parallel stream from a splitterator
- e.g., the `Collection` interface defines two default methods using this capability

```
public interface Collection<E>
    extends Iterable<E> {
    ...
    default Stream<E> stream() {
        return StreamSupport
            .stream(spliterator(),
                false);
    }

    default Stream<E>
        parallelStream() {
            return StreamSupport
                .stream(spliterator(),
                    true);
        }
    }
```

The 'false' parameter creates a sequential stream, whereas 'true' creates a parallel stream

End of Understand Java Streams Spliterators