METRICA

CONSULTING

Streams

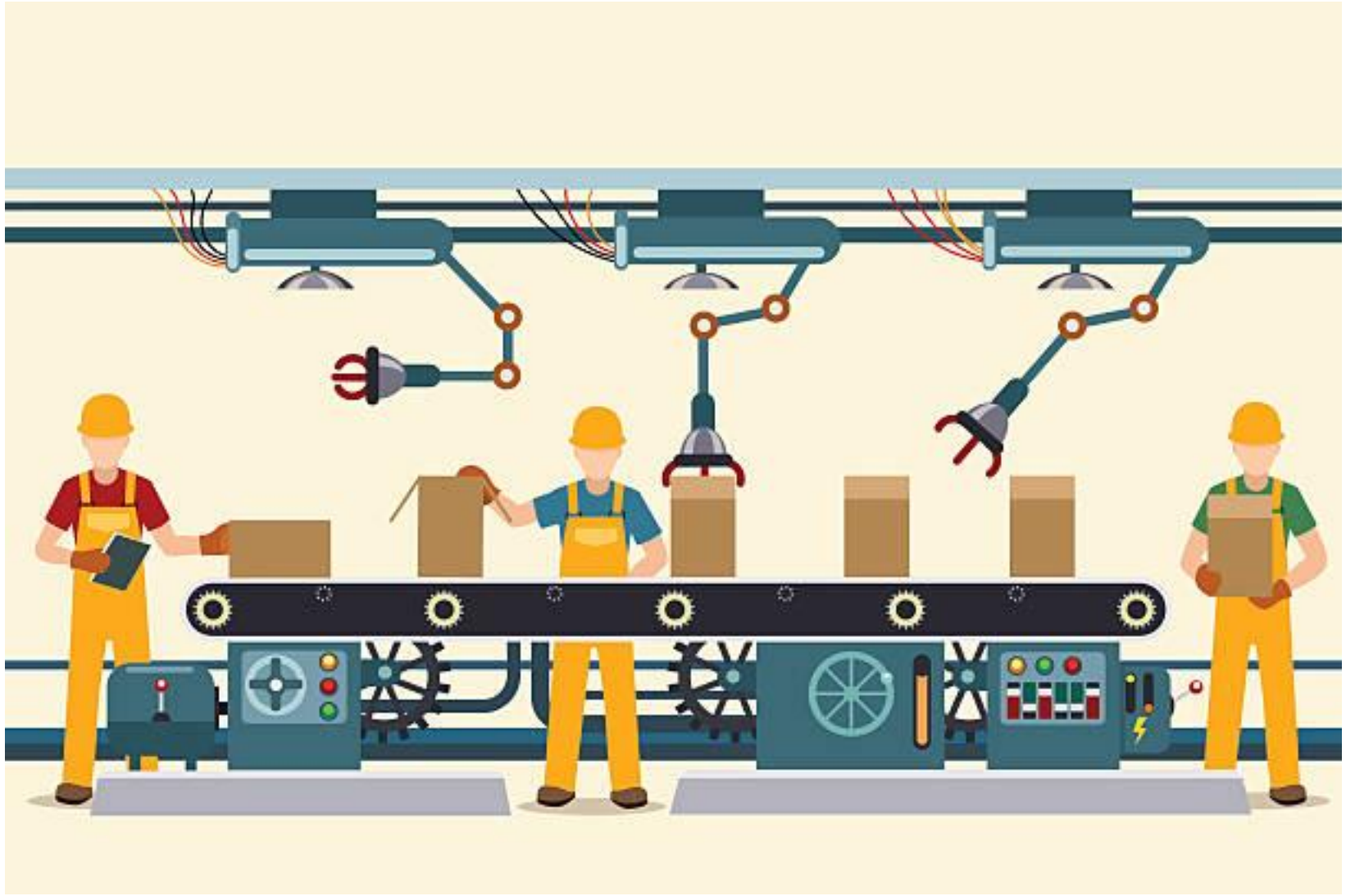
package java.util.stream

Definición

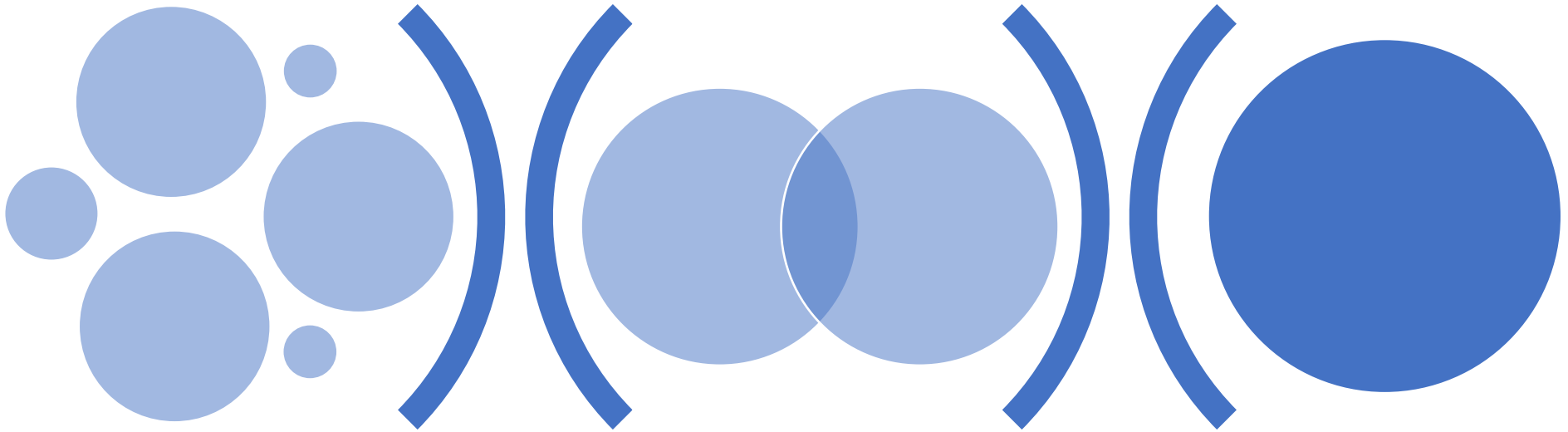
01

> “A sequence of elements from a source that support data processing operations”
>
> Raoul-Gabriel Urma

- Flujo de datos
- Indefinido
- Opera secuencialmente
- Operaciones anidadas
- Evaluación perezosa



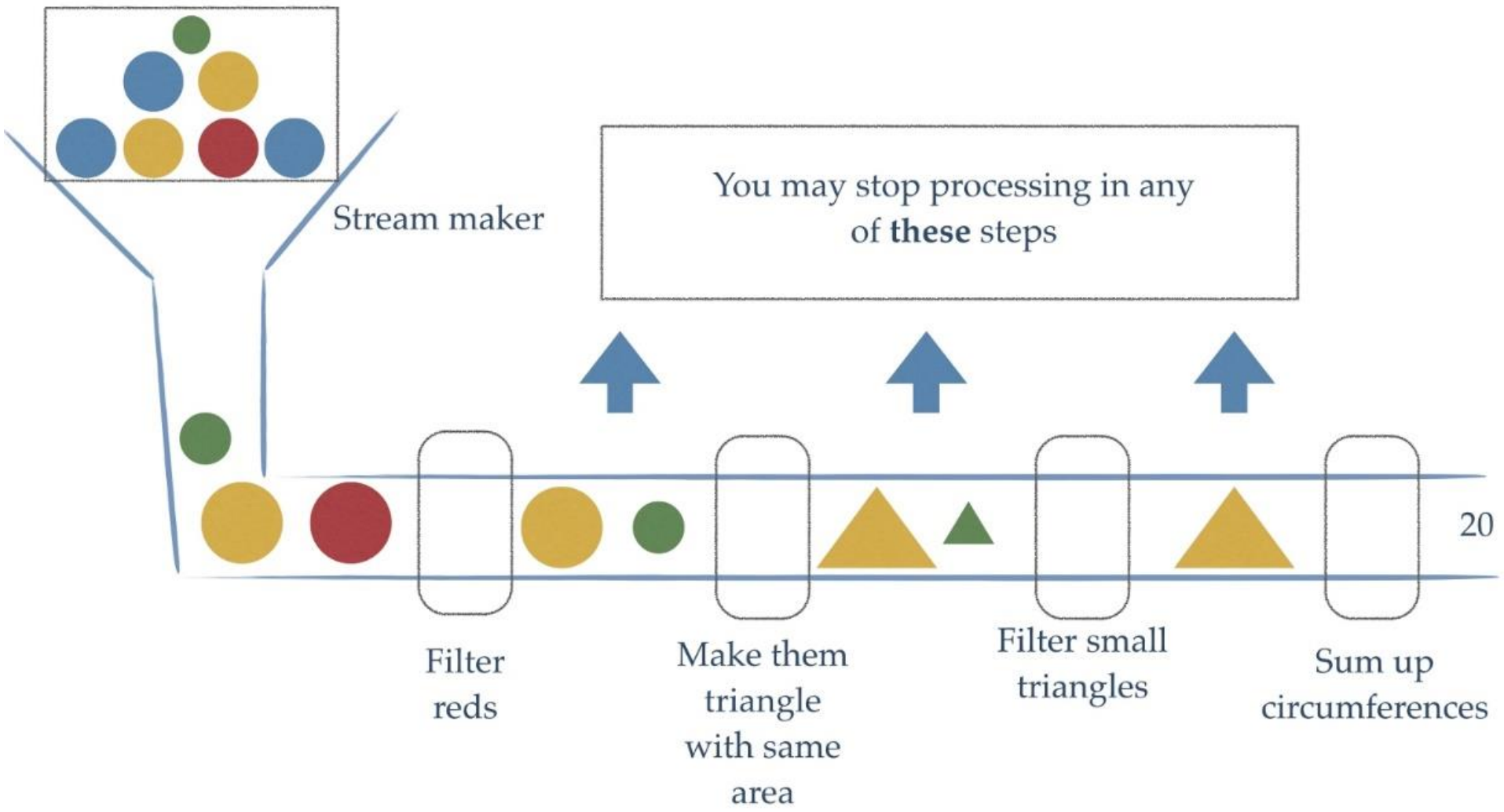
02 Anidación de operaciones



Stream

Funciones intermedias

Función final


Stream maker
You may stop processing in any of **these** steps
20
Filter reds
Make them triangle with same area
Filter small triangles
Sum up circumferences

03 Origen de un Stream

- De una función intermedia
- De la unión de dos Stream
- Desde un Collection
- Desde un Array
- Iterador o generador
- Flujo de I/O

Motivación

04

"A *sequence* of elements supporting sequential and parallel aggregate operations"

https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.htm

no almacenadas	secuencias indefinidas	realizan iteraciones internas
procesadas en paralelo	soportan programación funcional	evaluación perezosa
no modifican datos origen	no pueden ser reutilizadas	EFICIENTES

05

Ejemplos

```
Stream<Long> longGenerator = Stream.iterate(1L, n -> n+1);
```

```
Stream<Double> numbeeeeeeeeeeeeeeeers = Stream.generate(Math::random);
```

```
List<Integer> lista = List.of(1,2,3,4);
int sumPar = lista.parallelStream()
                  .filter(n -> n %2 ==0)
                  .map(n -> n*n)
                  .reduce(0, Integer::sum);
```

Tipos de intermedias 06

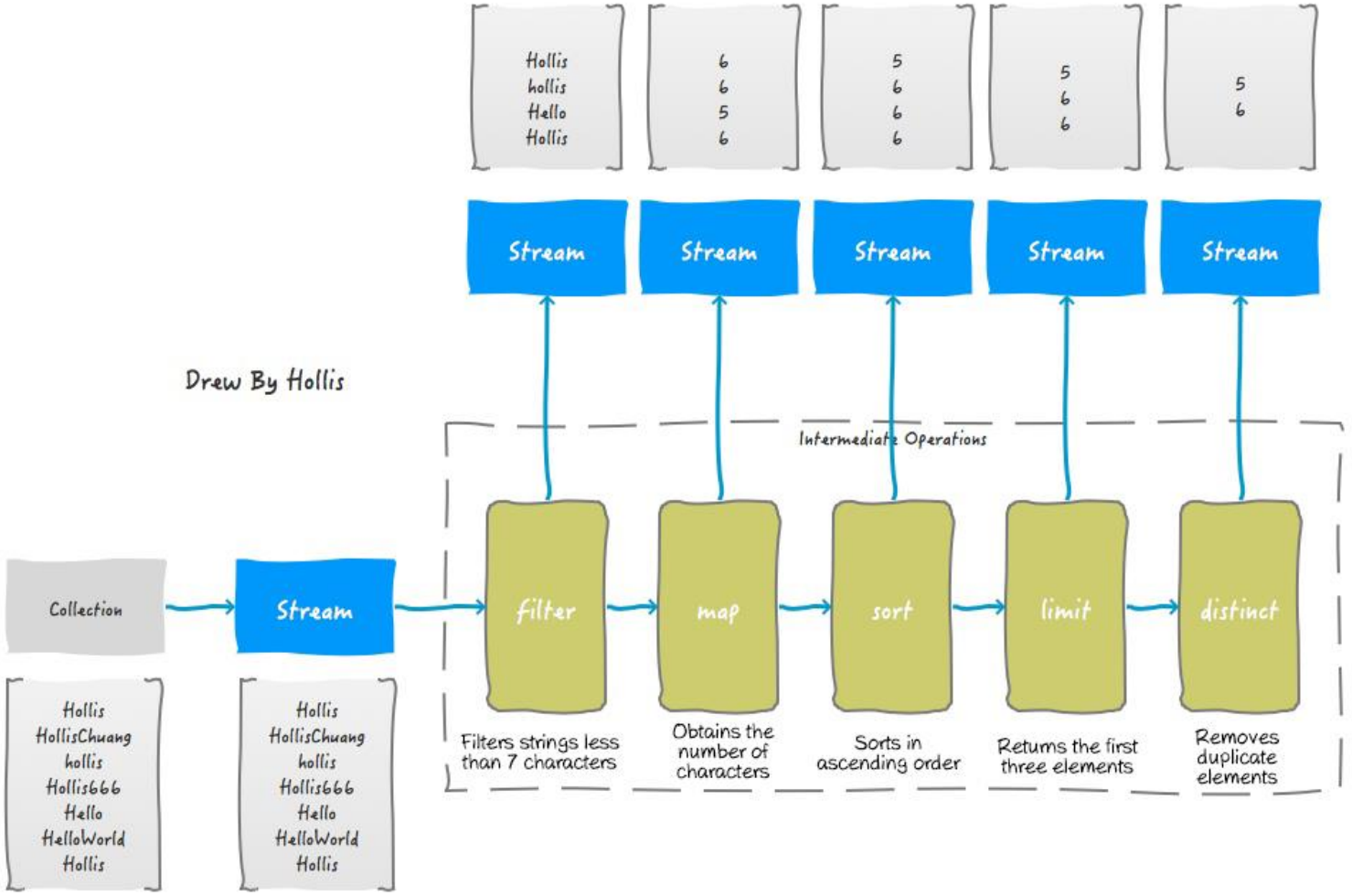
Operaciones intermedias

sin estado

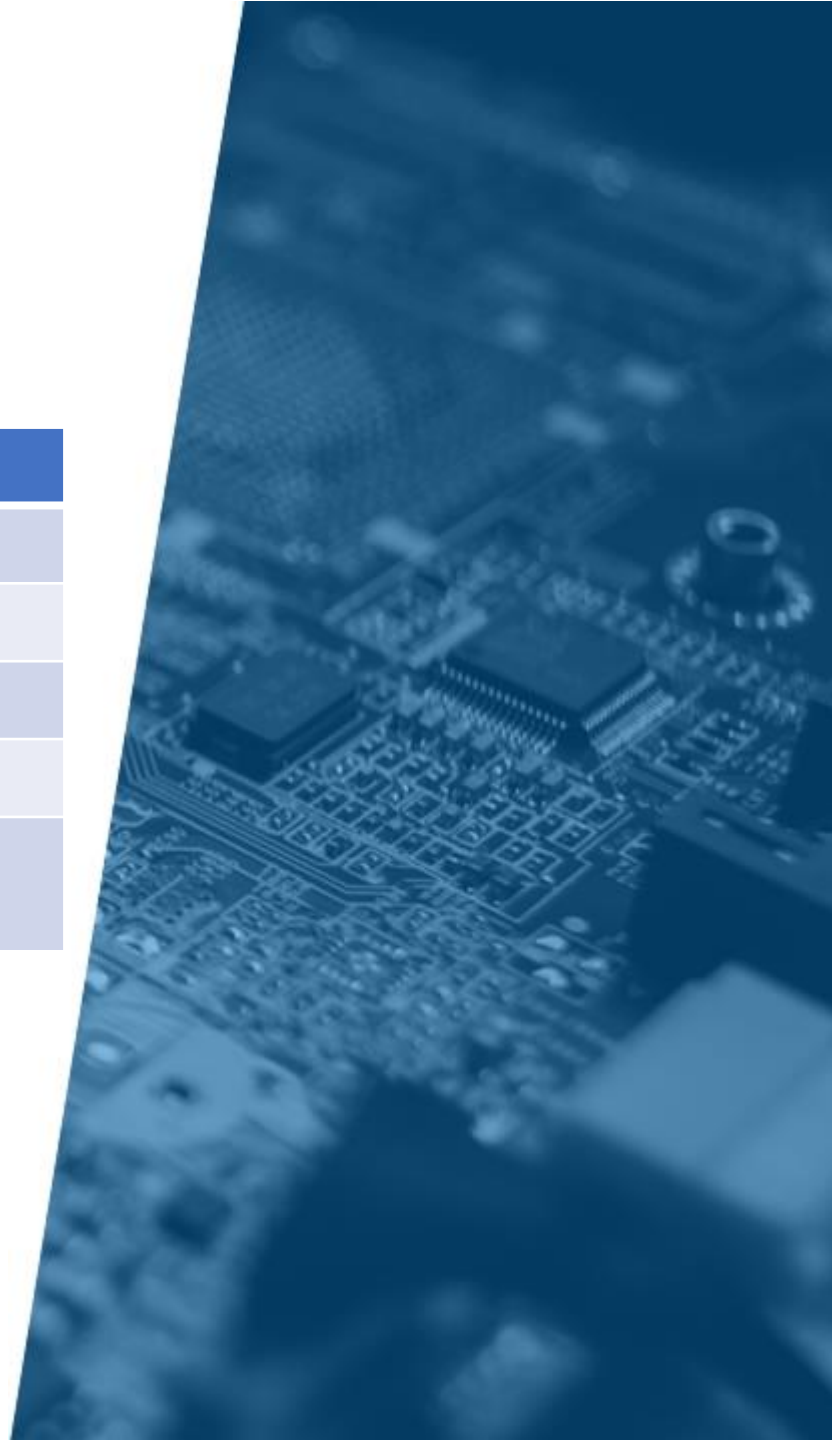
- map
- filter

completas

- sort
- distinct


Hollis
hollis
Hello
Hollis
6
6
5
6
5
6
6
6
5
6
6
5
6
Stream
Stream
Stream
Stream
Stream
Drew By Hollis
Intermediate Operations
Collection
Stream
filter
map
sort
limit
distinct
Hollis
HollisChuang
hollis
Hollis666
Hello
HelloWorld
Hollis
Hollis
HollisChuang
hollis
Hollis666
Hello
HelloWorld
Hollis
Filters strings less than 7 characters
Obtains the number of characters
Sorts in ascending order
Returns the first three elements
Removes duplicate elements

Operator	Objetivo	Input
filter	filtrar elementos según un criterio	Predicate<T>
map	transformar elementos	Fuction<T, E>
limit	limita el número de elementos	int
sorted	procesa elementos de forma ordenados	Comparator
distinct	elimina elementos duplicados según el criterio de equals	



Tipos de finales

07

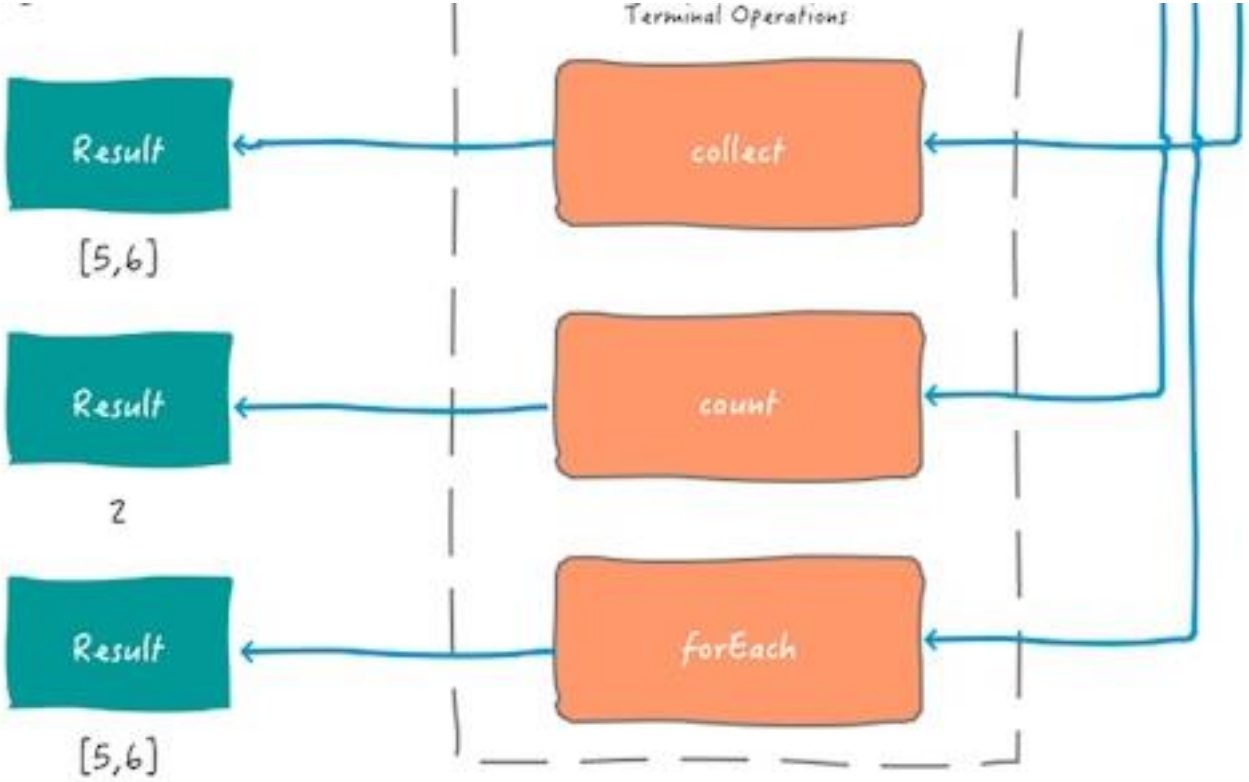
Operaciones finales

terminales

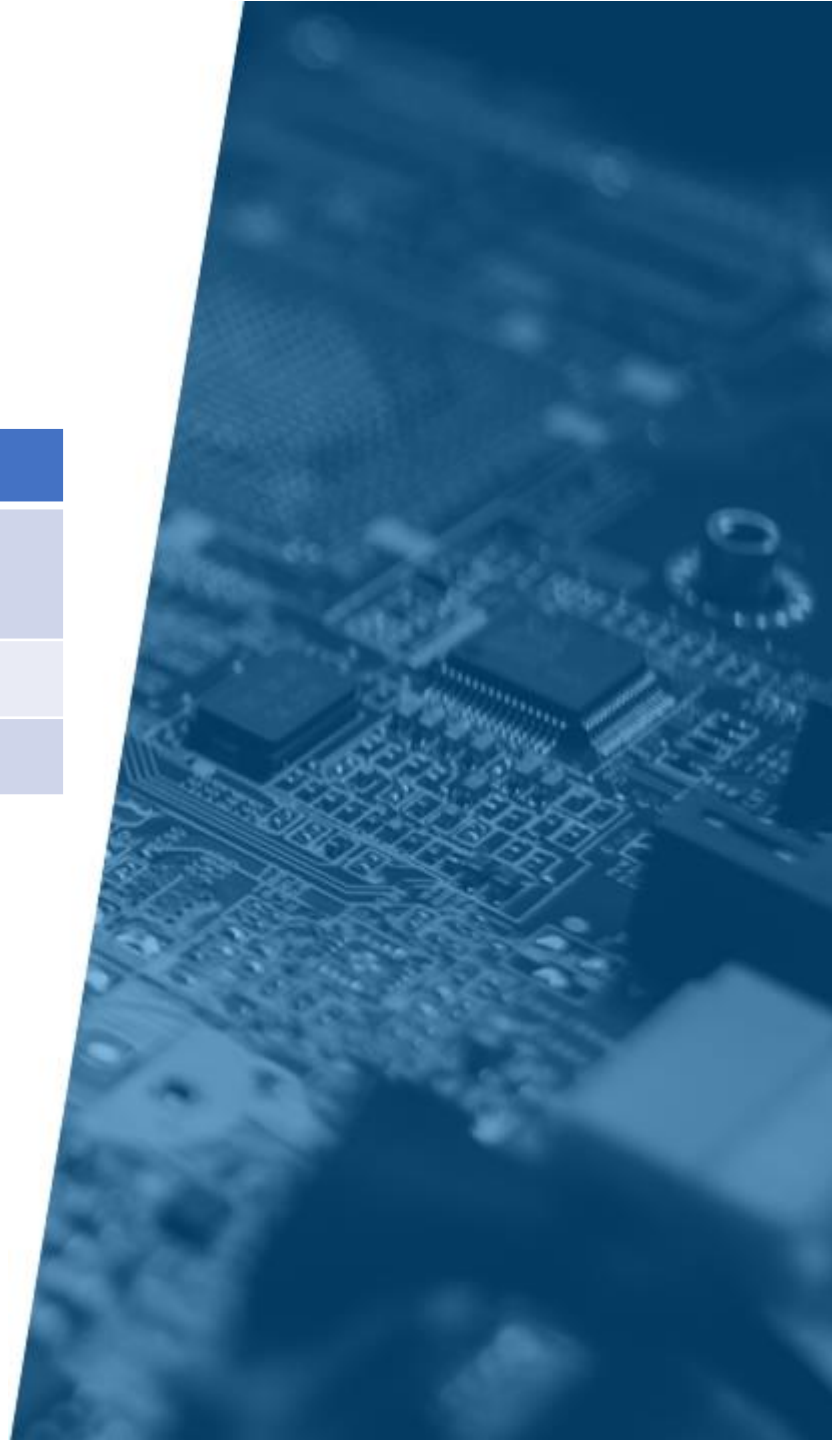
- min
- max
- count
- forEach

cortocircuitadas

- anyMatch
- allMatch
- noneMatch


Terminal Operations
Result
[5,6]
collect
Result
2
count
Result
[5,6]
forEach

Operator	Objetivo	Input
forEach	por cada elemento se realiza una operación	Consumer<T>
count	cuenta los elementos actuales	
collect	agrega los elementos en una estructura	



Ejemplo

```
var unknow = Stream.of("Ken","Ellen","Li")
                   .sorted()
                   .filter(s -> s.length()>10)
                   .findFirst();
```

08 Ejemplo

¿cuál es el primo que ocupa la posición 1000?

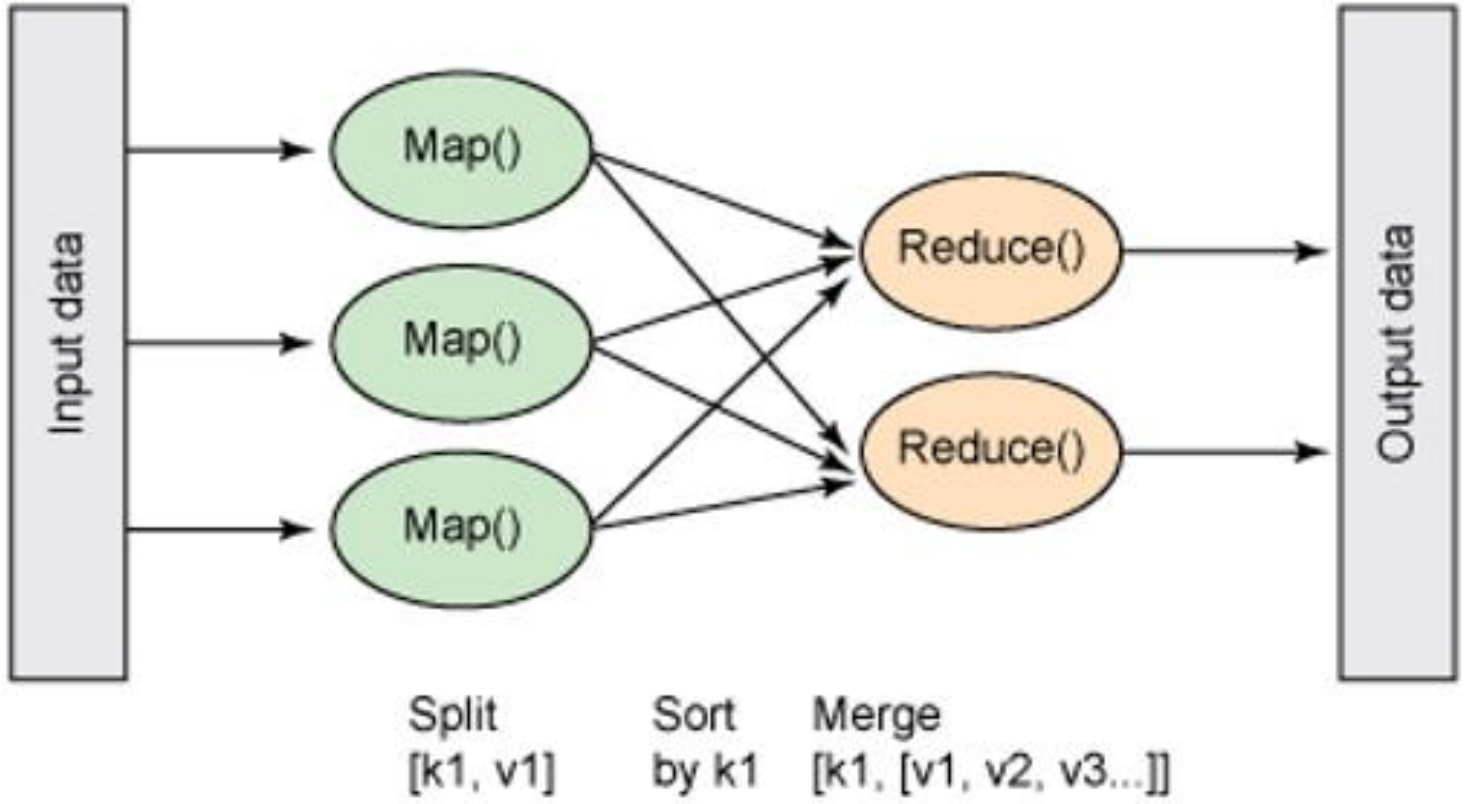
- iterate
- filter
- skip

```
Long target = Stream.iterate(1L, n -> n+1)
                    .filter(Main::esPrimo)
                    .skip(1000)
                    .findFirst()
                    .get();
```

08 ¿Qué devuelve?

```
Long target = Stream.iterate(1L, n -> n+1)
                    .sorted()
                    .filter(Main::esPrimo)
                    .skip(1000)
                    .findFirst()
                    .get();
```

Map - Reduce


Input data
Map()
Map()
Map()
Reduce()
Reduce()
Output data
Split
[k1, v1]
Sort
by k1
Merge
[k1, [v1, v2, v3...]]

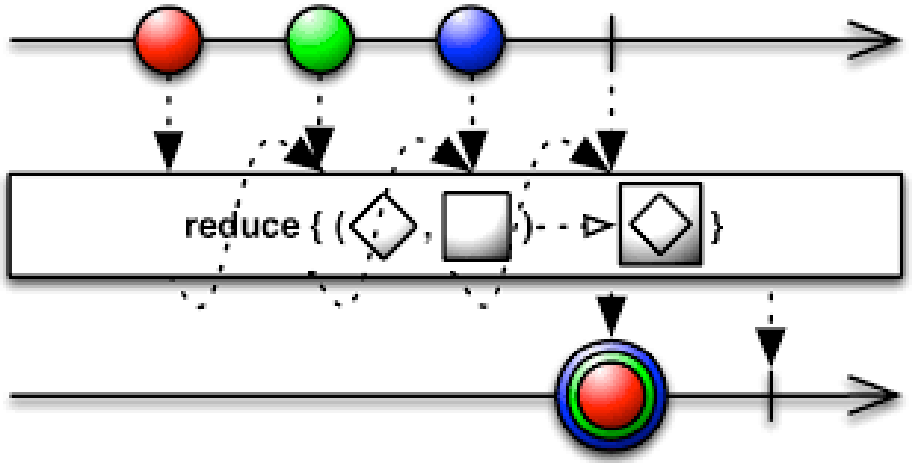
10

Stream<T>::map

`<R> Stream<R>`	`map(Function<? super T,? extends R> mapper)`
`DoubleStream`	`mapToDouble(ToDoubleFunction<? super T> mapper)`
`IntStream`	`mapToInt(ToIntFunction<? super T> mapper)`
`LongStream`	`mapToLong(ToLongFunction<? super T> mapper)`

11

Stream<T>::Reduce



11

Stream<T>::Reduce

`Optional<T>`	`reduce(BinaryOperator<T> accumulator)`
`T`	`reduce(T identity, BinaryOperator<T> accumulator)`
`<U> U`	`reduce(U identity, BiFunction<U,? super T,U> accumulator, BinaryOperator<U> combiner)`

12 Ejemplo

Calcular la suma de los 100 primeros números

- iterate
- limit
- reduce
- longValue

```
Long sum = Stream.iterate(1L, a -> a+1)
                 .limit(100)
                 .reduce(0L, Long::sum)
                 .longValue();
```

12 Ejemplo

Calcular la suma de las 3 primeros raíces de naturales

- iterate
- limit
- reduce

```
Double sum = Stream.iterate(1L, a -> a+1)
          .limit(3)
          .reduce(0.0, (a, b)-> a+ Math.sqrt(b), Double::sum)
          .doubleValue();
```

13

Stream<T>::collect

<R,A> R	**collect**(**Collector**<? super T,A,R> collector)	Performs a mutable reduction operation on the elements of this stream using a Collector.

```
.collect(Collectors.toList());
.collect(Collectors.toMap(Person::getName, Person::getGender));
```

<R> R	**collect**(**Supplier**<R> supplier, **BiConsumer**<R,? super T> accumulator, **BiConsumer**<R,R> combiner)	Performs a mutable reduction operation on the elements of this stream.

```
.collect(ArrayList::new, ArrayList::add, ArrayList::addAll);
```

https://docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html

Speed Limit

Vowel Count

Sum of Odd Cubed Numbers

Meeting