



Curso de Java Standard



Ing. Octavio Robleto



octavio.robleto@gmail.com



<https://octaviorobleto.com>

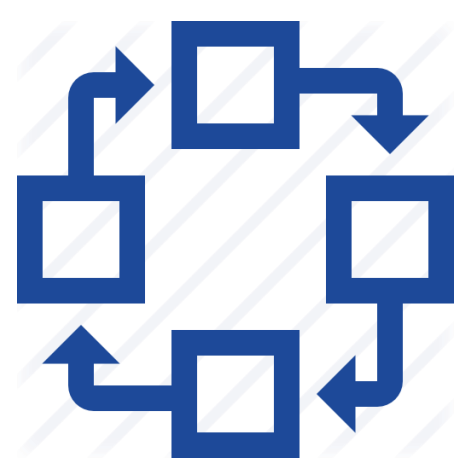


Introducción

Es una interfaz que nos provee clases para procesar datos de manera funcional como flujo de datos, No es otra colección.

También nos provee de una serie de métodos de orden superior (funciones que reciben una o mas funciones y retornan otra función o un objeto).

Estas funciones nos ayudan a transformar, filtrar y reducir una colección dada.

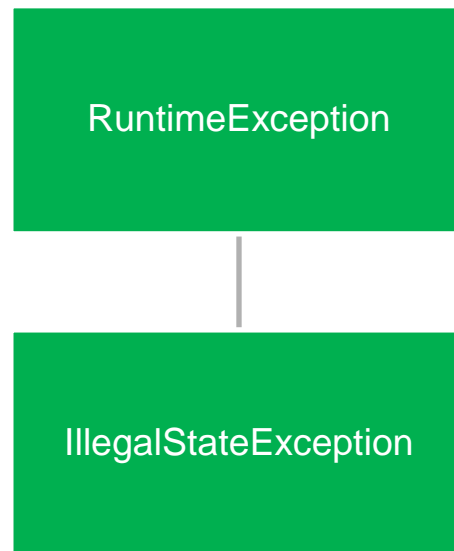


Excepciones

La excepción **RuntimeException** es la super clase encargada de lanzar un error en tiempo de ejecución de nuestra aplicación.

La excepción **IllegalStateException** se encargara de lanzar un error cuando tratamos de acceder a un objeto en un momento inapropiado, puede ser por uso o por otras causas.

Un **Stream** debe operarse (invocando una operación de secuencia intermedia o terminal) solo una vez



Stream

La interfaz **BaseStream** provee un flujo de objetos que permite el procesamiento paralelo y secuencial de dicho flujo.

La interfaz **Stream** provee los métodos necesarios para hacer operaciones, aunque suele ser confundida con una colección sabemos que tienen objetivos muy diferentes, las colecciones proporcionan una manera eficaz de acceso y modificación de datos y los flujos de operaciones.



```
graph TD; A[BaseStream<T,Stream<T>>] --- B[Stream<T>];
```

BaseStream<T,Stream<T>>

Stream<T>

Métodos Stream


Tipo	Método	Descripción
boolean	<code>allMatch(Predicate<? super T> predicate)</code>	Devuelve true si todos los elementos coinciden con el predicado proporcionado.
boolean	<code>anyMatch(Predicate<? super T> predicate)</code>	Devuelve true si algún elemento coincide con el predicado proporcionado.
<code><R,A> R</code>	<code>collect(Collector<? super T,A,R> collector)</code>	Realiza una operación de reducción en los elementos usando un Collector.
long	<code>count()</code>	Devuelve el recuento de elementos.
<code>Stream<T></code>	<code>distinct()</code>	Devuelve un flujo que consta de los distintos elementos (<code>Object.equals(Object)</code>)
<code>static <T> Stream<T></code>	<code>empty()</code>	Devuelve true si esta vacío el flujo.
<code>Stream<T></code>	<code>filter(Predicate<? super T> predicate)</code>	Devuelve un flujo que consta de los elementos que coinciden con el predicado dado.
void	<code>forEach(Consumer<? super T> action)</code>	Realiza una acción para cada elemento de este flujo.
<code><R> Stream<R></code>	<code>map(Function<? super T,? extends R> mapper)</code>	Devuelve un flujo que consta de los resultados de aplicar la función dada a los elemento.
<code>static <T> Stream<T></code>	<code>of(T... values)</code>	Devuelve un flujo ordenada secuencial cuyos elementos son los valores especificados.
<code>T</code>	<code>reduce(T identity, BinaryOperator<T> accumulator)</code>	Realiza una reducción en los elementos, utilizando el valor de identidad proporcionado y una función de acumulación asociativa , y devuelve el valor reducido.

Function

Es una función que toma los parámetros, los procesa o los opera.

Ejemplo Tradicional: a cada elemento de la colección se le multiplica por dos y se agrega a una nueva colección.

```
List<Integer> numerosOriginal = new ArrayList<>();
numerosOriginal.add(2);
numerosOriginal.add(3);
numerosOriginal.add(4);
List<Integer> numerosNuevos = new ArrayList<>();
System.out.println(numerosOriginal);
for (Integer numero; numerosOriginal) {
    numerosNuevos.add(numero * 2);
}
System.out.println(numerosNuevos);
```




Numeros Original: [2, 3, 4]
Numeros Nuevos: [4, 6, 8]

Predicate

Es una función que toma los parámetros y evalúa si cumple con la condición indicada.

Ejemplo Tradicional: se evalúa cada elemento de la colección y si es par se agrega a una nueva colección.

```
List<Integer> numerosOriginal = new ArrayList<>();
numerosOriginal.add(2);
numerosOriginal.add(3);
numerosOriginal.add(4);
List<Integer> numerosNuevos = new ArrayList<>();
System.out.println("Numeros Original: " + numerosOriginal);
for (Integer numero : numerosOriginal) {
    if (numero%2==0) {
        numerosNuevos.add(numero);
    }
}
System.out.println("Numeros Nuevos: " + numerosNuevos);
```



Numeros Original: [2, 3, 4]
Numeros Nuevos: [2, 4]


Consumer

Es una función que toma los argumentos y no retorna nada, simplemente procesa.

Ejemplo Tradicional: se envía cada elemento como argumento al método **System.out.println()**.

```
List<Integer> numerosOriginal = new ArrayList<>();
numerosOriginal.add(2);
numerosOriginal.add(3);
numerosOriginal.add(4);

System.out.println("Numeros Original: " + numerosOriginal);
for (Integer numero : numerosOriginal) {
    System.out.println(numero);
}
```



```
Numeros Original: [2, 3, 4]
2
3
4
```


Conceptos

- ✓ Las colecciones poseen el método **stream()** que retorna el un flujo con el contenido de la colección.
- ✓ En Java 8 las **Function, Predicate y Consumer** se realizaran con expresiones Lambdas.
 - ((Parametros) -> Function)
 - ((Parametros) -> Predicate)
 - ((Parametros) -> Consumer)

Adicionalmente debemos tener en cuenta que **no** podemos realizar mas de una operación a la vez ya que lanzaría una excepción de tipo **IllegalStateException**.

- ✓ Los métodos filter(), map() son operaciones intermedias.
- ✓ Los métodos count() y sum() son operaciones de terminal.



forEach

Es un método que se usa para recorrer cada uno de los elementos y espera como argumento un Consumer, los Stream y Colecciones poseen este método de recorrido.

```
List<String> nombres = Arrays.asList("Octavio", "Sabrina", "Sebastian", "Ariel", "Nahuel");

System.out.println("Iteracion Tradicional:");

for (String nombre : nombres) {
    System.out.println(nombre);
}

System.out.println("Iteracion Funcional:");
nombres.forEach((e) -> System.out.println(e));
```

Map

Es un método que se usa para transformar un objeto en otro aplicando una función y retornara un nuevo flujo.

```
List<String> nombres = Arrays.asList("Octavio", "Sabrina", "Sebastian", "Ariel", "Nahuel");

System.out.println("Nombres en Mayuscula Funcional:");
List<String> nombresMayusculasFuncional = nombres.stream().map((e) -> e.toUpperCase())
    .collect(Collectors.toList());

System.out.println("Iteracion Funcional:");
nombresMayusculasFuncional.forEach((e) -> System.out.println(e));
```



Filter

Este método filtra elementos en función de un predicado que se haya enviado.

```
List<String> nombres = Arrays.asList("Octavio", "Sabrina", "Sebastian", "Ariel", "Nahuel");

System.out.println("Nombres que comiencen con la letra S Funcional:");
List<String> nombresComienzanConSFuncional = nombres.stream().filter((e) -> e.startsWith("S"))
    .collect(Collectors.toList());

System.out.println("Iteracion Funcional:");
nombresComienzanConSFuncional.forEach((e) -> System.out.println(e));
```



Reduce

Este método recibe un función de acumulación y retorna un solo valor.

```
List<String> nombres = Arrays.asList("Octavio", "Sabrina", "Sebastian", "Ariel", "Nahuel");  
  
System.out.println("Convertir lista a cadena Funcional:");  
String cadenaNombresFuncional = nombres.stream().reduce("", (a, b) -> a + " " + b);  
  
System.out.println("Cadena de Nombres Funcional: " + cadenaNombresFuncional);
```

