# Interfaz Stream

Introducida en Java 8, forma parte del paquete *java.util.stream* y representa una **secuencia de elementos** que permite realizar operaciones funcionales de manera declarativa, eficiente y, a menudo, paralela. No almacena datos, sino que los procesa "*en flujo*", extrayéndolos de una fuente (como colecciones, arrays o I/O) y operando sobre ellos de forma perezosa (lazy evaluation).

**Stream** es ideal para operaciones como filtrado, mapeo, ordenamiento y reducción, promoviendo un estilo de **programación funcional** que evita modificar datos originales (es inmutable por diseño).

## Formas de crear un Stream

Existen varias maneras de crear un Stream dependiendo de la fuente de datos. Aquí las más comunes:

### Desde una colección

Usando el método *stream()* de la interfaz *Collection*.

Ejemplo:

*List<String> lista = Arrays.asList("a", "b", "c");*

*Stream<String> stream = lista.stream();*

### Desde un array

Con *Arrays.stream()*.

Ejemplo:

*String[] array = {"x", "y", "z"};*

*Stream<String> stream = Arrays.stream(array);*

### Desde valores directos

Usando *Stream.of()*.

Ejemplo:

*Stream<String> stream = Stream.of("uno", "dos", "tres");*

### Streams vacíos o generados

* *Stream.empty()* para un Stream vacío.
* *Stream.generate()* o *Stream.iterate()* para generar secuencias.

Ejemplo:

*Stream<Integer> numeros = Stream.iterate(0, n -> n + 1).limit(5);*

### Desde fuentes externas

Como archivos o I/O con *Files.lines()* para leer líneas de un archivo.

Ejemplo:

*Stream<String> lineas = Files.lines(Paths.get("archivo.txt"));*

## Tuberías (Pipelines) en Stream

Un pipeline en Stream es una cadena de operaciones que se aplican secuencialmente a los elementos del flujo. Se dividen en dos tipos de operaciones (tres si hablamos de Java avanzado, las operaciones cortocircuitadas):

### Operaciones intermedias

Transforman un Stream en otro Stream y son **perezosas** (no se ejecutan hasta que se invoca una operación terminal).

Ejemplos:

* *filter(Predicate<T>)*: Filtra elementos según un predicado.

Ejemplo: *stream.filter(s -> s.startsWith("a"))*

* *map(Function<T, R>)*: Transforma elementos.

Ejemplo: *stream.map(String::toUpperCase)*

* *sorted()*: Ordena los elementos.
* *distinct()*: Elimina duplicados.

### Operaciones terminales

Producen un **resultado final** o un **efecto secundario** y cierran el Stream.

Ejemplos:

* *forEach(Consumer<T>)*: Aplica una acción a cada elemento.

Ejemplo: *stream.forEach(System.out::println)*.

* *collect()*: Recolecta elementos en una estructura.

Ejemplo: *List<String> lista = stream.collect(Collectors.toList())*.

* *reduce()*: Reduce el Stream a un solo valor.

Ejemplo: *stream.reduce("", String::concat)*

# Ejemplo general Interfaz Stream

import java.io.IOException;

import java.nio.file.\*;

import java.util.\*;

import java.util.stream.\*;

public class EjemploF1 {

public static void main(String[] args) throws IOException {

// 1. Desde una colección

List<String> pilotos = Arrays.asList("Alonso", "Verstappen", "Hamilton", "Leclerc", "Albon", "Sainz", "Russell", "Alonso");

Stream<String> streamDesdeColeccion = pilotos.stream();

// 2. Desde un array

String[] array = {"Gasly", "Ocon", "Magnussen"};

Stream<String> streamDesdeArray = Arrays.stream(array);

// 3. Desde valores directos

Stream<String> streamDirecto = Stream.of("uno", "dos", "tres");

// 4. Streams vacíos o generados

Stream<String> streamVacio = Stream.empty();

Stream<Integer> streamGenerado = Stream.iterate(0, n -> n + 1).limit(5);

// Tubería

System.out.println("Pilotos que empiezan con 'A':");

pilotos.stream()

.filter(nombre -> nombre.startsWith("A"))

.map(String::toUpperCase)

.distinct()

.sorted()

.forEach(System.out::println);

// collect()

List<String> listaMayusculas = pilotos.stream()

.map(String::toUpperCase)

.collect(Collectors.toList());

System.out.println("Todos en mayúsculas: " + listaMayusculas);

// reduce()

String concatenados = pilotos.stream()

.reduce("", (a, b) -> a + b + " ");

System.out.println("Concatenados: " + concatenados.trim());

}

}