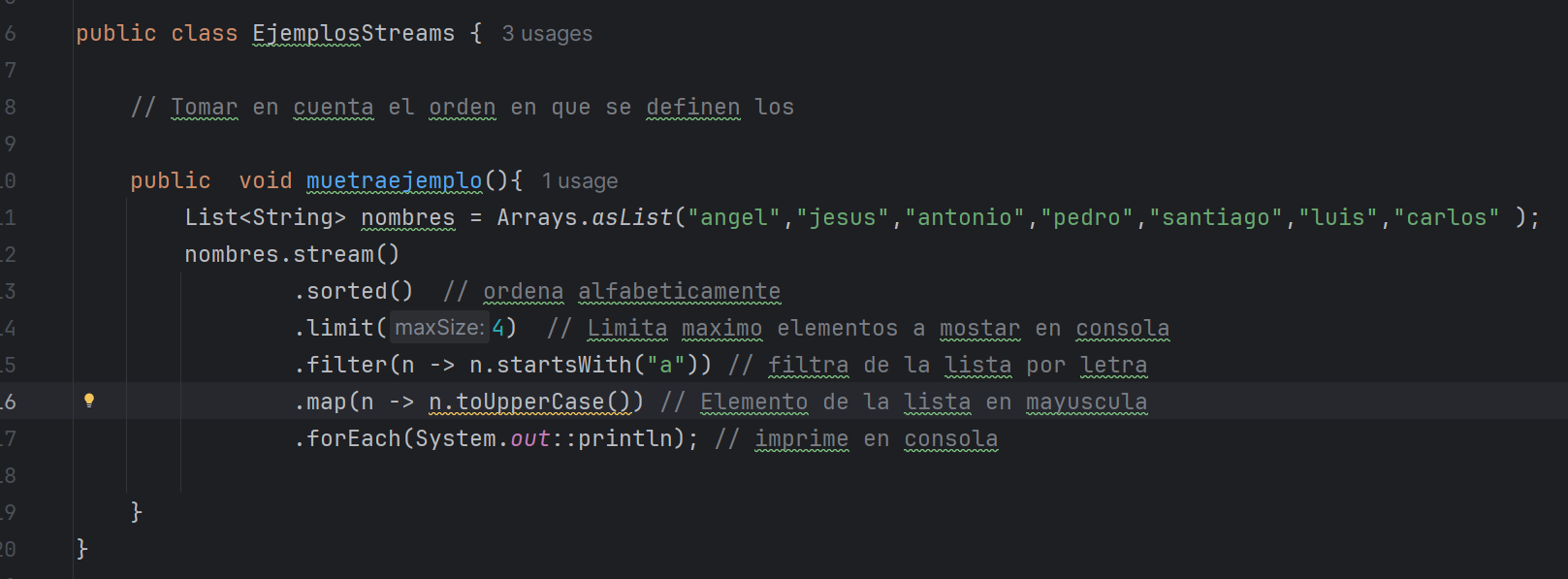


Api alura



Uso de streams

Los streams son una nueva forma de trabajar con colecciones de datos en Java, introducidas a partir de Java 8. Permiten realizar operaciones de manera más eficiente y concisa, utilizando un enfoque funcional.

Un stream es una secuencia de elementos que puede procesarse en paralelo o en serie. Puede crearse a partir de una colección, un array, un archivo, entre otros. A partir de ahí, podemos realizar diversas operaciones en ese stream, como filtrar, mapear, ordenar, entre otras.

Las operaciones intermedias son aquellas que pueden aplicarse en un stream y retornan un nuevo stream como resultado. Estas operaciones no se ejecutan inmediatamente, sino solo cuando se llama a una operación final.

Veamos algunos ejemplos de operaciones intermedias:

Filter: permite filtrar los elementos del stream basándose en una condición. Por ejemplo, podemos filtrar una lista de números para devolver solo los números pares.

List<**Integer**> numeros = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

List<**Integer**> numerosPares = numeros.stream()

.**filter**(n -> n % 2 == 0)

.**collect**(Collectors.toList());

System.out.println(numerosPares); // Salida: [2, 4, 6, 8, 10]

COPIA EL CÓDIGO

Map: permite transformar cada elemento del stream en otro tipo de dato. Por ejemplo, podemos transformar una lista de cadenas en una lista de sus respectivas longitudes.

List<String> palabras = Arrays.asList("Java", "Stream", "Operaciones", "Intermedias");

List<Integer> tamaños = palabras.stream()

.**map**(s -> s.length())

.collect(Collectors.toList());

System.out.println(tamaños); // Salida: [4, 6, 11, 17]

COPIA EL CÓDIGO

Las operaciones finales son aquellas que cierran el stream y devuelven un resultado concreto. Algunas operaciones finales comunes son forEach, collect y count.

Veamos algunos ejemplos de operaciones finales:

ForEach: permite ejecutar una acción en cada elemento de la stream. Por ejemplo, podemos imprimir cada elemento de la lista.

java

List<**String**> nombres = Arrays.**asList**("Juan", "Maria", "Pedro", "Ana");

nombres.**stream**()

.**forEach**(nombre -> System.out.**println**("Hola, " + nombre + "!"));

COPIA EL CÓDIGO

// Salida:

// Hola, Juan!

// Hola, Maria!

// Hola, Pedro!

// Hola, Ana!

Collect: permite recopilar los elementos de la stream en una colección o en otro tipo de dato. Por ejemplo, podemos recopilar los números pares en un conjunto.

List<**Integer**> numeros = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

**Set**<**Integer**> numerosPares = numeros.stream()

.**filter**(n -> n % 2 == 0)

.**collect**(Collectors.toSet());

System.out.println(numerosPares); // Salida: [2, 4, 6, 8, 10]

COPIA EL CÓDIGO

En resumen, los streams y sus operaciones intermedias y finales son una forma poderosa de manipular colecciones de datos en Java de manera más eficiente y concisa. Permiten escribir código más legible y expresivo, facilitando el procesamiento y la transformación de datos.

Además de las operaciones intermedias y finales mencionadas, hay muchas otras disponibles, como distinct (que elimina elementos duplicados), limit (que limita el número de elementos de la stream), skip (que salta los primeros elementos del stream), reduce (que combina los elementos de la stream en un único resultado) y muchas más.

Los streams en Java son una herramienta poderosa para la manipulación de colecciones de datos. Las operaciones intermedias permiten filtrar, mapear y ordenar los elementos del stream, mientras que las operaciones finales cierran el stream y devuelven un resultado concreto. Combinando estas operaciones, podemos realizar transformaciones y acciones de manera concisa y eficiente.