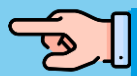
Arreglos y archivos

Secuencia de datos

{desafío} latam_

Implementar una aplicación básica de consola utilizando las buenas prácticas y convenciones para resolver un problema de baja complejidad acorde al lenguaje Java

- Unidad 1: Flujo, ciclos y métodos
- Unidad 2: Arreglos y archivos
- Unidad 3: Programación orientada a objetos
- Unidad 4: Pruebas unitarias y TDD



Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Conocer la secuencia de objetos que admiten varios métodos que se pueden canalizar para producir un resultado deseado.*

¿Qué entendemos por secuencia de objetos?

/* Secuencia de datos */

Características

- Toma datos de las colecciones, las matrices o los canales de E / S.
- Proporcionan el resultado como los métodos canalizados.
- Cada operación intermedia se ejecuta de forma perezosa y, como resultado, devuelve una secuencia, por lo que se pueden canalizar varias operaciones intermedias.
- La operación de terminal marca el final de la secuencia y devuelve el resultado.

Stream

Secuencia de datos

Para operar sobre arreglos, o colecciones de datos en general, tenemos la API Stream, que nos permite realizar operaciones sobre nuestro conjunto de datos.

Mostrar el Stream

Para ver los elementos

```
List<Integer> numeros = Arrays.asList(1,4,8,5,5,10,2);
```

Para ver los elementos

```
numeros.stream().forEach(System.out::println);
```

Map

Trabajar con Map mediante Stream

Map dentro de un stream de Java nos da la posibilidad de convertir un tipo de objeto en otro, o trabajar con el objeto que hemos recibido.

```
numeros.stream().map(n -> n*3).forEach(System.out::println);
```

```
3
12
24
15
15
30
6
```

{desafío} latam_

Map

Trabajar con Map mediante Stream

Cada elemento se multiplicó por 3, y luego se mostró por pantalla, y si vemos que pasó con el arreglo

```
System.out.println(numeros);
```

ningún elemento fue modificado:

```
[1, 4, 8, 5, 5, 10, 2]
```

Creando una lista para almacenar el resultado

```
List<Integer> numeros = Arrays.asList(1,4,8,5,5,10,2);
List<Integer> numerosAumentados = new ArrayList<Integer>();
numeros.stream().map(n -> n*3).forEach(numerosAumentados::add);

System.out.println(numeros);
System.out.println(numerosAumentados);
```

```
[1, 4, 8, 5, 5, 10, 2]
[3, 12, 24, 15, 15, 30, 6]
```

Usando collectors

Permite establecer las reglas de agrupamiento y recolección de los datos del modo que más se ajuste a nuestras necesidades.
Devuelve la salida de las operaciones intermedias; también se puede utilizar para convertir la salida en la estructura de datos deseada.

```
import  java.util.stream.Collectors;
List<Integer> numeros = Arrays.asList(1,4,8,5,5,10,2);
List<Integer> numerosAumentados2 = numeros.stream().map(n->n*n).collect(Collectors.toList());
```

Predicado

Un predicado es una interfaz funcional.
Reciben un argumento y devuelven un valor luego de realizar una operación.

```
List<Integer> numeros = Arrays.asList(1,4,8,5,5,10,2);
List<Integer> numerosAUno = numeros.stream().map(n -> 1).collect(Collectors.toList());
System.out.println(numerosAUno);


// [1, 1, 1, 1, 1, 1, 1]
```

Creando filtros

filter(predicado)

```
List<Integer> numeros = Arrays.asList(1,4,8,5,5,10,2);
List<Integer> numFiltrados = numeros.stream()
        .filter(x -> x>=4)
        .collect(Collectors.toList());
System.out.println(numFiltrados);

// [4, 8, 5, 5, 10]
```

Creando filtros

filter(predicado)

```
// y si queremos el cuadrado de cada uno de ellos

List<Integer> numFiltrados = numeros.stream()
        .filter(x -> x>=4)
        .map(x -> x*x)
        .collect(Collectors.toList());
System.out.println(numFiltrados);


// [16, 64, 25, 25, 100]
```

Otros filtros

```
List<Integer> numerosDistintos = numeros.stream()
            .distinct()
            .collect(Collectors.toList());
System.out.println(numerosDistintos);


//[1, 4, 8, 5, 10, 2]
```

```
List<Integer> numerosLimit = numeros.stream()
            .limit(2)
            .collect(Collectors.toList());
System.out.println(numerosLimit);


//[1,4]
```

Otros filtros

```
List<Integer> numerosLimit = numeros.stream()
            .limit(2)
            .collect(Collectors.toList());
System.out.println(numerosLimit);


//[1,4]
```

```
List<Integer> numerosSkip = numeros.stream()
            .skip(2)
            .collect(Collectors.toList());
System.out.println(numerosSkip);


//[5, 5, 10, 2]
```

Reduciendo los datos

Si queremos obtener, por ejemplo, la suma de todos los elemento del arreglo:

```
int suma = numeros.stream().reduce(0, (a,b) ->a+b);
System.out.println(9suma);
```

reduce(): es un método acumulador, donde se define en valor de inicio y la operación a realizar.

```
int multiplicacion = numeros.stream().reduce(1, (a,b) ->a*b);
System.out.println(multiplicacion);
```

para multiplicar todos los elementos del arreglo

```
int max = numeros.stream().reduce(1, Integer::max);
int min = numeros.stream().reduce(1000,Integer::min);
System.out.println("min: " + min + "\nmax: "+ max);  //10
```

Operando con Streams numéricos

Existen Streams que operan sobre números:

- mapToInt()
- mapToDouble()
- mapToLong()

y tienen métodos como:

- sum()
- average()
- count()
- max()
- min()
- summaryStatistics()

```
int suma = numeros.stream().mapToInt(Integer::intValue).sum();
System.out.println(suma);
```

Ejercicio 1

¡Practiquemos juntos!

Dado un arreglo de enteros

```
List<Integer> numeros = Arrays.asList(1,9,2,10,2,4,7,4,7,1,4);
```

Realizar las siguientes operaciones

- Utilizando reduce, sumar todos los valores del array
- Utilizando reduce, sumar todos los valores no repetidos del array
- Convertir todos los datos al tipo float
- Filtrar todos los elementos menores a 5 sin repetir
- Utilizando **`mapToInt`**, sumar todos los valores del arreglo
- Utilizando **`.count()`** contar todos los elementos menores que 5

Ejercicio 2

¡Practiquemos juntos!

Dado un arreglo con nombres

```
List<String> nombres =
Arrays.asList("Anastasia","Beatriz","Clara","Carla","Marianela","Paula","Pia");
```

- Obtener todos los elementos que excedan los 5 caracteres
- Utilizar map para transformar todos los nombres a minúscula
- Crear un arreglo con todos los nombres que comiencen con P
- Utilizando **`.count`**, contar los elementos que empiecen con 'A', 'B' o 'C'.
- Utilizando **`.map`**, crear un arreglo con la cantidad de letras que tiene cada nombre.

Entonces, ¿Qué es la secuencia de objetos y para qué sirve?

Próxima sesión...

- *Comprender la Clase FileWriter y BufferedWriter para el uso de los métodos en la construcción y escritura de archivos.*
- *Construir Clase FileReader a través de su constructor para lectura de archivos.*
- *Aplicar la Clase BufferedReader y sus métodos para lectura de archivos.*

{desafío}
latam_

Academia de
talentos digitales