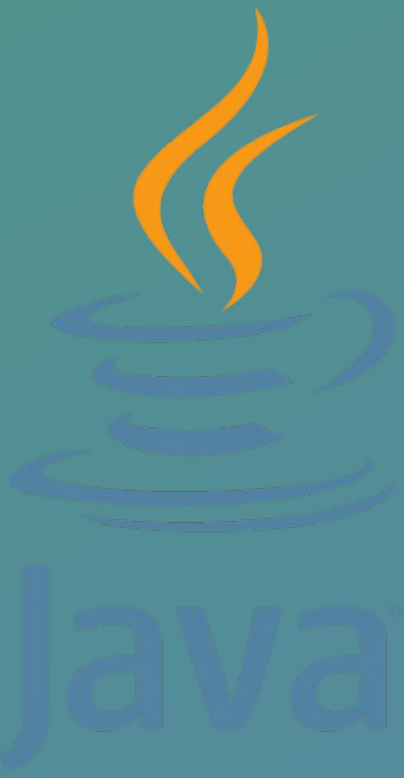

Java – Introducción a la programación funcional

Streams y expresiones Lambda en Java

¿Que es la programación funcional?

- Paradigma de programación:
  - **Imperativo:**
    - Damos ordenes
    - Usado habitualmente
  - **Declarativo**
    - Declaramos que queremos
    - **Programación funcional**
      - Importa que se esta haciendo y no “como”

¿Donde esta disponible?

- Multitud de lenguajes
  - Javascript, Python, PHP
  - **Java a partir de “Java 8”**
- Ejemplos de la presentación disponibles en
  - https://github.com/sergarb1/JavaFuncional

Ejemplo Imperativo

```
List<Integer> numeros = Arrays.asList(18, 6, 4, 15, 55, 78, 12, 9, 8);

int contador = 0;
for(int numero : numeros) {
    if(numero > 10) {
        contador ++;
    }
}
System.out.println(contador);
```

Ejemplo Funcional

```
List<Integer> numeros = Arrays.asList(18, 6, 4, 15, 55, 78, 12, 9, 8);

long contador = numeros.stream().filter(n->n > 10).count();
System.out.println(contador);
```

Expresiones Lambda (1)

- **Expresiones Lambda** λ
  - Compuesta por dos elementos, separados por una flecha →
    - Izquierda flecha parámetros
    - Derecha flecha expresión
  - Es un función anónima
- **Ejemplo**

```
n -> n > 10
```

Expresiones Lambda (2)

- **Parte izquierda de la flecha**
  - Parámetros
  - Pueden tener 0, 1 o varios.
  - **Ejemplo**: `n -> n > 10`
    - 1 parámetro (n)

Expresiones Lambda (3)

- **Parte derecha de la flecha →**
  - Expresión Lambda a ejecutar.
  - Devuelve lo que devuelve la operación.
  - Permite ejecutar código
    - Ejemplo, hacer un **System.out.println**
  - **Ejemplo:**
    - Procesa n > 10 `n -> n > 10`
    - Devuelve true o false por cada elemento

Expresiones Lambda en ejemplo

```
List<Integer> numeros = Arrays.asList(18, 6, 4, 15, 55, 78, 12, 9, 8);

long contador = numeros.stream().filter(n->n > 10).count();
System.out.println(contador);
```

- **filter(num → num > 10)**
- **A la izquierda de →**
  - Son los parámetros (en este caso 1, llamado **n**)
- **A la derecha de →**
  - Operación a realizar
  - Se ejecutara tantas veces como elementos hay.

Streams (1)

- **Streams**
  - Un conjunto de funciones que se ejecutan de forma anidada
  - No es una estructura de datos, pero puede modificar datos
  - Se inicia creando el flujo con “stream”
    - Ejemplo: `numeros.stream()`

Streams (2)

- **Funcionamiento**
  - Flujo de funciones, donde el resultado de una es la entrada de la siguiente.
  - Existen operaciones **intermedias** y **terminales**.
    - Intermedias: generan streams donde se pueden aplicar nuevas funciones.
    - Terminales: procesan un resultado sobre un stream

Streams (3)

- **Operaciones intermedias con Streams**
  - filter : filtra elementos
  - sorted: ordena elementos
  - map: mapea una operación a cada elemento
- **Otras operaciones intermedias**
  - flatmap, distinct, peek, limit, skip

Streams (4)

- **Operaciones terminales con Streams**
  - toArray
  - forEach
  - count
  - min / max
- **Otras operaciones terminales**
  - Collect, forEachOrdered, reduce, anyMatch, allMatch

Streams en el ejemplo

```
List<Integer> numeros = Arrays.asList(18, 6, 4, 15, 55, 78, 12, 9, 8);

long contador = numeros.stream().filter(n->n > 10).count();
System.out.println(contador);
```

- **numeros.stream()**
  - Genera el stream de la lista
- **filter(n → n > 10)**
  - Filtra dejando solo elementos mayores que 10
- **count()**
  - Filtra dejando un entero con el número de elementos

Enlaces Interesantes

- **Enlaces interesantes**
  - https://www.north-47.com/knowledge-base/java-8-streams/
  - https://www.arquitecturajava.com/programacion-funcional-java-8-streams/
- **Cheat Sheets**
  - https://www.jrebel.com/blog/java-streams-cheat-sheet
  - https://programming.guide/java/lambda-cheat-sheet.html