Java – Introducción a la programación funcional

Expresiones lambda y streams

¿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 en como

¿Donde esta disponible?

- Multitud de lenguajes
  - Javascript y frameworks modernos (jQuery, Vue)
  - Python
  - PHP
  - **Java a partir de Java 8**
- Ejemplos de la presentación disponibles en
  - https://github.com/sergarb1/JavaFuncional

Ejemplo Imperativo

```java
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
  - Compuesta por dos elementos, separados por una flecha →
  - Es un función anónima
- Parte izquierda de la flecha →
  - Parámetros de entrada
  - Pueden ser varios

Expresiones Lambda (2)

- Parte derecha de la flecha →
  - Expresión Lambda.
  - Devuelve lo que devuelve la operación.
    - Comportamiento depende de donde se use (filtrar, ordenar)
  - Permite ejecutar código (ejemplo, hacer un System.out.println)

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 num)
- 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

Streams (2)

- Funcionamiento
  - Se aplica una función a un Stream.
  - La siguiente función anidada se aplica al flujo modificado por la función anterior.
  - Este proceso se repite para todas las funciones.

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(num → num > 10)
  - Filtra dejando solo elementos mayores que 10
- count()
  - Filtra dejando un entero con el número de elementos

Otro ejemplo Imperativo

```
ArrayList<Persona> milista= new ArrayList<Persona>();
milista.add(new Persona("Mariano"));
milista.add(new Persona("Sergi"));
milista.add(new Persona("Laura"));
milista.add(new Persona("Miguel"));
Collections.sort(milista,new Comparator<Persona>() {
    public int compare(Persona p1,Persona p2) {
        return p1.getNombre().compareTo(p2.getNombre());
    }
});
for (Persona p: milista) {
    if(!p.getNombre().startsWith("M"))
       System.out.println(p.getNombre());
}
```

Otro ejemplo funcional

```
ArrayList<Persona> milista= new ArrayList<Persona>();
milista.add(new Persona("Mariano"));
milista.add(new Persona("Sergi"));
milista.add(new Persona("Laura"));
milista.add(new Persona("Miguel"));
Collections.sort(milista,
    (Persona p1,Persona p2)-> p1.getNombre().compareTo(p2.getNombre())
);
milista.stream().filter(p -> !p.getNombre().startsWith("M"))
.forEach(p -> System.out.println(p.getNombre()));
```

Explicación ejemplo (1)

```
Collections.sort(milista,
    (Persona p1,Persona p2)-> p1.getNombre().compareTo(p2.getNombre())
);
```

- Expresión lambda
  - A la izquierda de →
    - 2 parámetros tipo Persona p1 y p2
  - A la derecha de →
    - Expresión. Obtiene atributo nombre y los compara mediante compareTo

Explicación ejemplo (2)

```
milista.stream().filter(p -> !p.getNombre().startsWith("M"))
.forEach(p -> System.out.println(p.getNombre()));
```

- Filter: filtra elementos según expresión Lambda
- Expresión Lambda
  - A la izquierda de → 1 parámetro p
  - A la derecha de → solo elementos que no empiecen por M

Explicación ejemplo (3)

```
milista.stream().filter(p -> !p.getNombre().startsWith("M"))
.forEach(p -> System.out.println(p.getNombre()));
```

- ForEach: se aplica a cada elemento
  - En Este caso se aplica a los elementos que quedan tras aplicar filter()
- Expresión Lambda
  - A la izquierda de → 1 parámetro p
  - A la derecha de → Acción a ejecutar, imprimimos el nombre del parámetro

Enlaces Interesantes (1)

- Enlaces Streams y Lambda
  - https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html
  - https://www.oracle.com/technetwork/es/articles/java/procesamiento-streams-java-se-8-2763402-esa.html
- Cheat Sheets
  - https://www.jrebel.com/blog/java-streams-cheat-sheet
  - https://programming.guide/java/lambda-cheat-sheet.html

Enlaces Interesantes (2)

Web "Arquitectura Java"

- https://www.arquitecturajava.com/java-stream-filter-y-predicates/
- https://www.arquitecturajava.com/programacion-funcional-java-8-streams/
- https://www.arquitecturajava.com/java-8-lambda-y-foreach-ii/
- https://www.arquitecturajava.com/programacion-funcional-java-8-streams/