INTRODUCCIÓN A LAS EXPRESIONES LAMBDA

PROGRAMACIÓN FUNCIONAL

- Diferente de la programación imperativa
- Lenguaje matemático formal
- En este paradigma, la salida de una función depende exclusivamente de sus parámetros de entrada.
- Más expresivo (con menos código) y elegante.
- Presente en otros lenguajes de programación desde hace tiempo.

FUNCIONES: ENTIDADES DE PRIMER NIVEL

- Hasta ahora las entidades de primer nivel eran variables, literales u objetos.
- Si las funciones pasan a ser entidades de primer nivel, podrán utilizarse como los anteriores.
- Por ejemplo, pasar una función como argumento de un método.

EXPRESIÓN LAMBDA

- Básicamente es un método abstracto, una función sin nombre.
- Su estructura es

```
() -> expresión

(parámetros) -> expresión

(parámetros) -> { sentencias; }
```

EJEMPLOS DE EXPRESIONES LAMBDA

```
() -> new ArrayList<>()

(int a, int b) -> a+b

(a)-> {
     System.out.println(a);
     return true;
 }
```

INTERFAZ FUNCIONAL

- Se trata de un interfaz con un solo método (que no sea static ni default) (sin contar equals, …).
- En lugar de implementar una clase, o una clase anónima, podemos utilizar una expresión lambda.

https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html

COLLECIONES y forEach

- Nuevo bucle para recorrer colecciones
- Recibe como parámetro un objeto que instancie una interfaz funcional, `Consumer<T>`.
- Perfecto para usar expresiones lambda.

ACCESO A MÉTODOS

- Se puede usar el operador ::
- Abrevia aun más la expresión

String::valueOf	x-> String.valueOf(x)
Object::toString	x -> x.toString()
x::toString	() -> x.toString()
ArrayList::new	() -> new ArrayList
System.out::println	x -> System.out.println(x)

API STREAM

- Nos permite trabajar con una colección con si se tratara de un flujo de información.
- Permite realizar fácilmente operaciones de filtrado, transformación, ordenación , agrupación y presentación de información.