UNIDAD 6. PROGRAMACIÓN FUNCIONAL

Índice

Introducción

¿ Qué es la programaciónfuncional?
Es un tipo de paradigma de programación que surgió en los años 30.
Se basa en un lenguaje matemático formal: **las expresiones lambda**.
En este paradigma, la salida de una función depende exclusivamente de sus parámetros de entrada, para producir la salida.
Es más expresivo (necesitamos menos código) y elegante.
Presente en otros lenguajes de programación como (Python, Scala, C#,etc...)

Funciones como entidades de primer nivel

Consideramos **entidades de primer nivel** aquellas que puedo pasar como **parámetros en un método**. Estas eran:

- Variables.
- Literales.
- Objetos. (en sus diferentes formas, colecciones, arrays, etc.)
  Ahora **las funciones** pasan a integrarse en este **primer nivel** lo que nos permite pasarlas como **argumentos** de los **métodos**.

Expresiones Lambda - sintaxis

La sintaxis general de una expresión lambda es

```
(tipo1 param1, tipo2 param2,….) -> { Cuerpo de la expresión lambda };
```

- **Una lista de parámetros** formales, entre paréntesis, separados por comas. Los tipos de los parámetros se pueden omitir si java los puede inferir del contexto. Cuando solo hay un solo parámetro de entrada,

también se pueden omitir los paréntesis.

- **Una flecha**
- **El cuerpo de la función** entre llaves, que puede consistir en una sentencia o un bloque de sentencias. Si es una única sentencia y no devuelve ningún valor, las llaves se pueden omitir. Si es una única sentencia y devuelve un valor, la orden return se puede omitir, ya que java devuelve automáticamente el resultado de la operación.

Lo podemos ver como un método abstracto, sin nombre.
posibles estructuras:

```
() -> expresión
// ejemplo
()->new ArrayList<>()

(parámetros) -> expresión
//ejemplo
(int a, int b)-> a+b

(parámetros) -> { sentencias; }
//ejemplo
(a)->{System.out.println(a);
return true;}
```

Interfaz funcional

Aquella que solo tiene **un solo método abstracto propio**.
Recuerda que no se cuentan ni las static, default, ni aquellas directamente heredadas de Object (equals, etc.)
Las expresiones lambda van asociadas a las interfaces funcionales y solo se pueden usar con ellas. La razón es porque las interfaces funcionales solo implementan un método, y por lo tanto no se hace necesario ni llamarlo por su nombre.

Para implementar **una interfaz funcional con una expresión lambda**, basta escribir la lista de parámetros y cuerpo de la función abstracta separados por una flecha ->
Ejemplo:

```
Comparator<Cliente> compCliNombre=(Cliente c1, Cliente c2)->{return
c1.nombre.CompareTo(c2.nombre);};
// se puede resumir la sentencia en
Comparator<Cliente> compCliNombre=(c1,c2)->{return
c1.nombre.CompareTo(c2.nombre);};
```

 Hoja de ejemplos (Ejemplo3 - listas de objetos ordenadas y Ejemplo7 -expresiones lambda)

Java API Stream

Un **Stream** en Java es una **secuencia de elementos**.

Permite hacer operaciones con un flujo de elementos de forma más sencilla.

Los stream en Java permiten encadenar las operaciones una a continuación de otra, formando **tuberías**, para dar un resultado final, sin necesidad de acceder a los resultados intermedios.

Un stream se recorre solo una vez.

Algunas Formas de definir un Stream

```
Stream.of(...)
Stream.empty()
Arrays.stream(T[])
Collection<T>.stream()
```

Interfaz Stream

Las operaciones realizas con un Stream pueden ser de dos tipos:

Operaciones intermedias

Dan como resultado un nuevo stream, al que se le pueden seguir aplicando nuevas operaciones

- filter(condicion): filtrar datos.
- limit(entero): limitar datos.
- skip(entero): saltar datos.
- map(funcion): aplica funciones a los elementos.
- flatMap(funcion): aplica funciones a un flujo de colecciones de un flujo de elementos combinados de la colección.
- peek(funcion): aplica funciones a un flujo de colecciones de un flujo.
- sorted():ordenar.

Operaciones terminales

Dan un resultado final, numérico o de otro tipo, pero no un Stream.

- forEach(System.out::println). Para visualizar los datos.
- max, min, count, sum... Operacione de agrupación.
- anyMatch(condición). Devuelve verdadero si al menos un elemento satisface la condición.
- Collect(). Nos permite transformar la salida en una colección.
- findFirst(). Devuelve el primero.

La ventaja de crear el Stream es que dispone de muchas más operaciones para procesar sus datos que las colecciones o las tablas(arrays).

Los Stream son objetos que implementan la interfaz . Por tanto, La clase Stream no existe y los objetos Stream no se pueden crear con un constructor, sino llamando a alguna de las funciones implementadas para ello.

Los Stream se han diseñado para trabajar con expresiones lambda.

 Hoja de ejemplos(EjemploStreams)

 Hoja de ejercicios Stream 1 y 2