Francisco Molina Sánchez
Guillermo Siles Bonilla

Programación Funcional en Java 8

PROGRAMACIÓN FUNCIONAL

- Es un paradigma de programación declarativa que especifica los programas a través de un conjunto de condiciones que describen el problema y detallan su solución.
- Transparencia referencial: el valor devuelto por una función depende exclusivamente de sus entradas, sin efectos secundarios.

NOVEDADES EN JAVA 8

- Interfaces funcionales
- Expresiones Lambda
- Stream API
- Nuevo motor de Javascript Nashorn

INTERFACES FUNCIONALES

- Una interfaz es funcional si declara solamente un método abstracto
- Pueden tener varios métodos predefinidos
- Podemos usar la anotación @FunctionalInterface
- Pueden ser instanciadas por las expresiones lambda

EXPRESIONES LAMBDA

- Código más compacto y limpio
- Inferencia de tipos, es decir,asigna automáticamente un tipo de datos a una función
- Podemos pasar funciones como parámetros
- Métodos anónimos, es decir, métodos sin nombre o clase.
- Introduce la programación funcional en Java 8
- Nacen los Streams API

Francisco Molina Sánchez
Guillermo Siles Bonilla

EXPRESIONES LAMBDA. EJEMPLOS BÁSICOS

Formas básica, extendida y reducida de las expresiones lambda.

EJ.1: INTERFAZ FUNCIONAL Y EXPRESIONES LAMBDA

Se muestra la diferencia entre las clases internas anónimas y las expresiones lambda. También se hace uso de los métodos por defecto dentro de una interfaz funcional.

EJ.2: FILTRAR ARCHIVOS

Filtramos los archivos de un directorio de forma tradicional usando la interfaz FilenameFilter como filtro y expresiones lambda.

INTERFACES FUNCIONALES BÁSICAS

Interfaces y métodos básicos que ya están implementados en el paquete java.util.function.*;

EJ.3: INTERFACES FUNCIONALES BÁSICAS

Ejemplos sencillos con los métodos vistos en el apartado anterior.

- Expresión lambda que toma un argumento de entrada y devuelve un booleano según la condición.
- Expresión lambda que coge 1 argumento de entrada y devuelve su hashcode
- Expresión lambda que "consume" un argumento de entrada y lo imprime
- Ejemplo del uso de forEach de una lista que toma como argumento un Consumer.

EJ.4: FUNCIONES DE ORDEN SUPERIOR

Son las que tienen algún argumento de entrada es una función o devuelven una función.

Ejemplo del uso del paso como parámetro de una función. En este caso, le pasamos a la funcion procesar, la expresión lambda "x -> x.hashCode()" y el mismo String. Ésta expresion lambda dado un String devuelve su hashcode. Procesar ejecuta la expresion lambda teniendo como argumento de entrada el String cadena.

EJ.5: DEVOLVIENDO UNA FUNCIÓN

En este ejemplo mostramos como devolver una expresión lambda.Podemos calcular la nota final de dos tipos de alumno, según la evaluacion continua o solo con examen final.

Ventaja: podemos usar la expresión lambda en diversos contextos.

EJ.6: ASIGNANDO UNA EXPRESIÓN LAMBDA O FUNCIÓN

Usamos el ejemplo anterior para mostrar cómo se puede asignar una función a una variable y su uso.

EJ.7: CURRIFICACIÓN: FUNCIONES PARCIALES

Mostramos como aplicar currificación con expresiones lambda. En este caso, usamos la BiFunction mayorQue imprimiendo una función parcial, y la aplicamos sobre una lista.

EJ.8: COMPOSICIÓN DE FUNCIONES

Mostramos como usar composición de funciones con expresiones lambda. Podemos usar los métodos "compose" (después de) y "andThen" (antes de) para aplicarlo según el orden que deseamos en el que se ejecutan las funciones.

STREAMS

Definimos las principales características de los streams, junto con un primer ejemplo que aclara un poco la sintaxis. Comentamos que funciones son las más usadas, que hacen, y pasamos a ver distintos ejemplos usando streams.

EJ.9: LAZY

Vemos esta característica de los stream, como es su comportamiento cuando los ejecutamos.

EJ.10: TIPOS DE DATOS

Con este ejemplo vemos que para los datos básicos como int, double, long… hay streams especiales (intStream, longStream…) que añaden funciones especiales como la media, el máximo, etc.

EJ. 11: IMPORTANCIA DEL ORDEN

Un ejemplo que ilustra la importancia del orden de las operaciones que realiza el stream. Eligiendo mal este orden tendremos un comportamiento peor del algoritmo.

EJ. 12: REUTILIZACIÓN

Con estos ejemplos vemos como reusar un stream.

EJ. 13: ACCIDENTES

Este apartado es importante ya que vemos los diferentes errores que se suelen cometer cuando se empieza a usar streams en java. Algunos ya los hemos visto antes en otras secciones como la reutilización o la importancia del orden.

EJ. 14: OPERACIONES AVANZADAS

Más que operaciones avanzadas, son otro tipo de operaciones no tan básicas pero que podemos necesitar frecuentemente cuando usemos streams.

EJ. 15: PARALLEL STREAM

Por último vemos un ejemplo sobre un stream ejecutándose en paralelo.