Lambda

# Propósito

Este tutorial presenta las nuevas expresiones lambda, incluidas en Java Platafom Edition (Java SE 8).

# Introducción

Las expresiones Lambda son una nueva e importante característica incluida en Java SE 8. Proveen un claro y conciso camino para representar una interface de un método usando una expresión. Estas expresiones también mejoran las librerías de Colecciones haciendo fácil iterarlas, filtrarlas y extraer datos. In adición, nuevas características de concurrencia mejoran el rendimiento en ambientes multi-núcleo.

Este OBE (Oracle by example) provee una introducción a las expresiones lambda. Una introducción a funciones anónimas internas seguida de una discusión de interfaces funcionales y la nueva sintaxis lambda. Entonces, ejemplos de patrones comunes antes y después de lambda se muestran.

La siguiente sección revisa una búsqueda común y cómo el código de java puede ser mejorado con la inclusión de expresiones lambda. En adición algunas interfaces funcionales comunes Predicate y Function provistas en el paquete java.util.function son mostradas en acción.

Este OBE termina con una revisión de cómo las colecciones java han sido actualizadas con expresiones Lambda.

# Clases Anónimas internas

En Java, las clases anónimas internas proveen un camino para implementar clases que ocurren solo una vez en una aplicación. Por ejemplo, en una aplicación estándar de Swing o JavaFX un número de manejadores de eventos son requeridos para los eventos de mouse y teclado. En lugar de escribir eventos separados puedes escribir algo como esto:

[Ejemplo.txt](src/java/oca2/lambda/Ejemplo.txt)

De otra manera, una clase separada que implemente la interface ActionListener es requerida para cada evento. Creando la clase en el lugar donde es necesitada, el código es un poco más fácil de leer. El código no es elegante, porque requieres de más código para definir un método.

# Interfaces funcionales

El código que define la interface ActionListener se observa algo como esto:

[ActionListener.java](src/java/oca2/lambda/ActionListener.java)

Esta interface tiene un solo método. Con Java SE 8, una interface que sigue este patrón es conocida como interface funcional; antes se conocía como Single Abstract Method type (SAM).

Usar interfaces funcionales con clases internas anónimas es un patrón común en java. En adición a las clases EventListener, interfaces como Runnable y Comparator son usadas de manera similar. Por lo tanto, las interfaces funcionales son apalancadas usando expresiones lambda.

# Sintaxis de expresiones Lambda

Una expresión lambda es compuesta por tres partes:

|  |  |  |
| --- | --- | --- |
| Lista de argumentos | Token flecha | Cuerpo |
| (int x, int y) | -> | x + y |

El cuerpo puede ser tanto una expresión simple o un bloque de sentencias. En la forma de expresión, el cuerpo es simplemente evaluado y retornado. En la forma de bloque, el cuerpo es evaluado como un método y una sentencia return retorna el control al llamador del método anónimo. Las palabras clave como break y continue son ilegales en el nivel más alto, pero son permitidas en los loops. Si el cuerpo produce un resultado, toda la ruta de control debe retornar algo o lanzar una excepción.

Veamos estos ejemplos:

* (int x, int y) -> x + y
* () -> 42
* (String s) -> {System.out.println(s);}

La primera expresión toma dos argumentos, llamados “x” y “y” y usa la expresión a retornar “x + y”. La segunda expresión no toma argumentos y usa la expresión para retornar un entero 42. La tercera expresión toma una String y usa la forma de bloque para imprimir la String a la consola y no retorna nada.

Con la sintaxis básica cubierta vamos a ver unos ejemplos.

# Ejemplos Lambda

Aquí hay algunos casos comunes usando los ejemplos previos.

Lambda Runnable: RunnableTest.java