Java Lambda

UTN – Facultad Regional Paraná

Titular: Ernesto Zapata Icart

Aux: Mariano Carpio

Programación funcional

- Estilo de programación que percibe el mundo como la evaluación de funciones matemáticas.
- Las funciones pueden tomar funciones como parámetros y devolver una función como resultado.
- Preferencia sobre la técnica de recursión sobre bucles for-while.
- Nos permite escribir código legible, más declarativo, programas más concisos que la programación imperativa.
- Nos permite enfocarnos más en el problema que en el código
- Facilita el paralelismo.

El cálculo Lambda λ

- El cálculo lambda es un sistema formal diseñado para investigar la definición de función, la noción de aplicación de funciones y la recursión.
- Fue introducido por **Alonzo Church** y **Stephen Kleene** en la década de 1930.
- Puede ser usado para definir de manera limpia y precisa qué es una "función computable".
- Se puede considerar al cálculo lambda como el lenguaje universal de programación más pequeño. Consiste en una regla de transformación simple (sustitución de variables) y un esquema simple para definir funciones.
- El cálculo lambda tiene una gran influencia sobre los lenguajes funcionales, como Lisp, ML y Haskell. Recientemente adoptado en Java 8.

El cálculo Lambda λ

- El concepto central del cálculo Lambda es el de *“expresión”*. Un *“nombre”* o *“variable”* es un identificador. Una expresión se define recursivamente como:
- `<expresión> := <nombre> | <función> | <aplicación>`
- `<función>  := λ <nombre>.<expresión>`
- `<aplicación>:= <expresión><expresión>`
- El orden de ejecución, por convención, está dado de izquierda a derecha. Si tenemos la siguiente expresión $E_1.E_2.E_3…E_n$, el orden de evaluación de las expresiones será: (… $((E_1.E_2).E_3)…E_n$).

El cálculo Lambda λ

- La expresión Lambda

$$\boldsymbol{\lambda}\ \boldsymbol{x}\ .\ (+\ \boldsymbol{x}\ 1)\ 2$$

representa la aplicación del parámetro 2 a la función $\boldsymbol{\lambda}\ x\ .\ (+\ x\ 1)$ con el parámetro formal $\boldsymbol{x}$ y el cuerpo de la función $(+\ x\ 1)$. Notese que la función $\boldsymbol{\lambda}\ x\ .\ (+\ x\ 1)$ no tiene nombre, y es denominada Función anónima.

- En Java 8, representamos esta definición de función con la expresión Lambda: **x -> x + 1**

Expresiones Lambda en Java 8

- Una expresión Lambda en Java 8 es básicamente un método sin la declaración, usualmente escrita de la siguiente forma:
  - **(** parámetros **) -> {** cuerpo **}**
  - `(int x, int y) -> { return x + y; }`
  - `x -> x * x`
  - `() -> x`
- Una expresión lambda puede tener cero o más parámetros separados por comas y sus tipos pueden ser explícitamente declarados o inferidos del contexto.
- Los parámetros no son necesarios alrededor de un simple parámetro.
- () es usado para denotar la no existencia de parámetros.
- El cuerpo puede contener cero o más sentencias.
- {} no son necesarias cuando tengo una sola sentencia.

Beneficios de expresiones Lambda en Java 8

- Expresiones Lambda son el nuevo agregado más importante en Java 8.
- El desafío mas grande fue introducir expresiones Lambdas sin la necesidad de recompilación de binarios existentes.
- Permite desarrollar un estilo de programación funcional.
- Permite escribir código más compacto y liviano.
- Facilita la programación paralela.
- Permite desarrollar componentes más genéricos, flexibles y reusables.
- Permite pasar datos como así también comportamiento a funciones.

Java 8 Lambdas

- Sintaxis de expresiones Lambda en Java 8.
- Interfaces funcionales.
- Captura de variables.
- Referencia de métodos.
- Métodos Default.

Sintaxis de expresiones Lambda

```
List<Integer> intSeq = Arrays.asList(1,2,3);
intSeq.forEach(x -> System.out.println(x));
```

- x -> System.out.println(x) es una expresión lambda que define una función anónima con un parámetro x del tipo *Integer*.

Sintaxis de expresiones Lambda

```
List<Integer> intSeq = Arrays.asList(1,2,3);
intSeq.forEach(x -> {
   x += 2;
   System.out.println(x);
});
```

- {} son necesarias para contener el cuerpo de la función con más de una sentencia en una expresión lambda.

Sintaxis de expresiones Lambda

```
List<Integer> intSeq = Arrays.asList(1,2,3);
intSeq.forEach((Integer x) -> {
    int y = x * 2;
    System.out.println(y);
});
```

- En expresiones lambda, como en cualquier método ordinario, puede definir variables locales dentro del cuerpo de la expresión lambda.
- También se puede especificar el tipo de parámetro.

Interfaces funcionales

- Decisión de diseño: Los lambdas de Java 8 se asignan a interfaces funcionales.
- Una interfaz funcional es una interfaz Java con exactamente un método no-default.

```
public interface Consumer<T> {
   void accept(T t);
}
```

- El paquete del API estándar de Java `java.util.function` define muchas nuevas interfaces funcionales.
- https://www.tutorialspoint.com/java8/java8_functional_interfaces.htm

Captura de variables

- Las expresiones lambda pueden interactuar con variables definidas fuera del cuerpo de la expresión lambda.
- El uso de estas variables se denomina: captura de variables.

```
public class Example {
  public static void main(String[] args) {
    List<Integer> intSeq = Arrays.asList(1,2,3);
    int var = 10;
    intSeq.forEach(x -> System.out.println(x + var));
  }
}
```

- Las variables usadas dentro del cuerpo de una expresión lambda debe ser **final** o **efectivamente final**.

Referencia de métodos

- Referencias a métodos pueden ser usadas para pasar una función existente en el lugar donde se esperaba una expresión lambda.
- La referencia a un método se logra usando el operador **::**. Se pueden referenciar métodos estáticos, de instancias y constructores.

Method Reference Type	Syntax	Example
static	ClassName::StaticMethodName	String::valueOf
constructor	ClassName::new	ArrayList::new
specific object instance	objectReference::MethodName	x::toString
arbitrary object of a given type	ClassName::InstanceMethodName	Object::toString

Referencia de métodos

- Se pude reescribir expresiones lambda

```
intSeq.forEach(x -> System.out.println(x));
```

- por otra expresión más concisa usando referencia a un método

```
intSeq.forEach(System.out::println);
```

Métodos Default

- Java 8 introduce un nuevo concepto de implementación de método por defecto en interfaces.
- Esto permite mantener compatibilidad hacia atrás, de modo tal que viejas interfaces pueden ser usadas para reforzar la capacidad de las expresiones lambda.
- Ej: las interfaces **List** y **Collections** no tienen el método “forEach”. Pensar en agregar este método haría que el framework Collection se tenga que reescribir. Al introducir el concepto de implementación por defecto para el método forEach, permite que las clases que implementan la interfaz tengan la libertad de sobreescribir el método o usar el comportamiento por defecto.

Métodos Default

- Sintaxis

```
public interface vehicle {
   default void print(){
      System.out.println("I am a vehicle!");
   }
}
```

- https://www.tutorialspoint.com/java8/java8_default_methods.htm