Instituto Tecnológico Superior de Jerez

Jerez de García Salinas, Zacatecas.

Fecha: 20/03/2020

Alumno: Adán Ruiz Villalobos

Núm. Control: 16070137

Correo: adnruiz1@gmail.com

Ing. Sistemas Computacionales

Materia: Programación Lógica & Funcional.

Semestre: 8°

Actividad: Mapa Conceptual

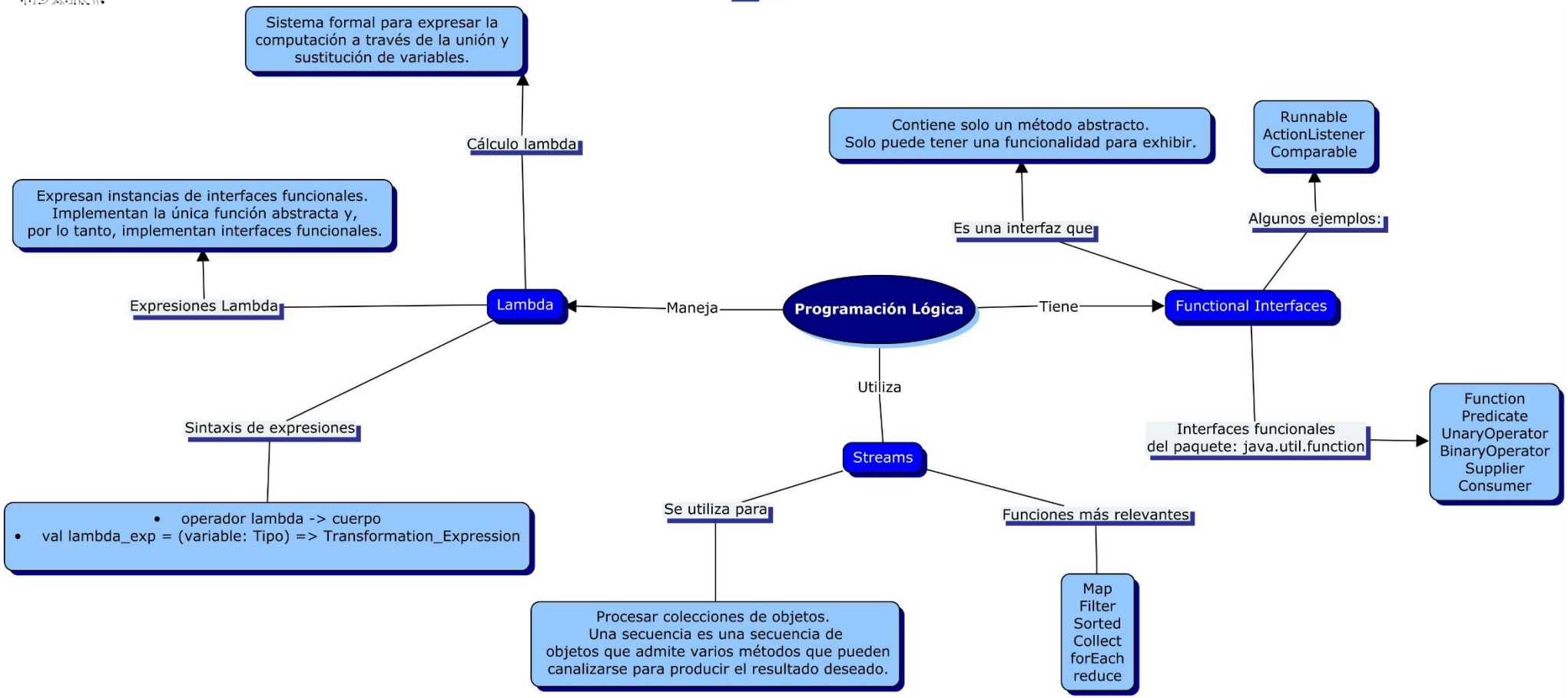
Docente: I.S.C. Salvador Acevedo Sandoval

Lambda Expressions. Java 8 (Paradigma funcional).

1. **Matemáticamente, que es cálculo lambda**
   Es un sistema formal en lógica matemática y ciencias de la computación para expresar la computación a través de la unión y sustitución de variables. Puede considerarse el lenguaje de programación universal más pequeño, cualquier función computable puede evaluarse en el contexto del cálculo λ y la evaluación de programas en el lenguaje consiste en una sola regla de transformación: sustitución de variables.
2. **Que son las 'functional interfaces' en Java**
   Es una interfaz que contiene solo un método abstracto. Solo puede tener una funcionalidad para exhibir. Puede tener cualquier cantidad de métodos predeterminados: **Runnable, ActionListener, Comparable** son algunos de los ejemplos de interfaces funcionales.
3. **Cuales son las 6 interfaces funcionales del paquete java.util.function**
   a. **Function:** (java.util.function.Function) representa una función (método) que toma un solo parámetro y devuelve un solo valor.
   b. **Predicate:** (java.util.function.Predicate) representa una función simple que toma un solo valor como parámetro y devuelve verdadero o falso.
   c. **UnaryOperator:** representa una operación que toma un solo parámetro y devuelve un parámetro del mismo tipo. Se puede usar para representar una operación que toma un objeto específico como parámetro, modifica ese objeto y lo devuelve nuevamente, posiblemente como parte de una cadena de procesamiento de flujo funcional.
   d. **BinaryOperator:** representa una operación que toma dos parámetros y devuelve un solo valor. Ambos parámetros y el tipo de retorno deben ser del mismo tipo.
   e. **Supplier:** representa una función que proporciona un valor de algunos tipos.
   f. **Consumer:** representa una función que consume un valor sin devolver ningún valor. Podria estar imprimiendo un valor, o escribiéndolo en un archivi, o en la red, etc. Imprime el valor pasado como parámetro System.out
   (Jenkov, 2019)
4. **Que son las expresiones lambda**
   Basicamente expresan instancias de interfaces funcionales. Implementan la única función abstracta y, por lo tanto, implementan interfaces funcionales.
   La expresión Lambda se refiere a una expresión que usa una función anónima en lugar de variable o valor. Son más convenientes cuando tenemos una función simple para usar en un solo lugar. Estas expresiones son más rápidas y más expresivas que definir una función completa.
   (MohammadKhalid, s.f.)
5. **Sintaxis de las expresiones lambda**
   - operador lambda -> cuerpo
   - val lambda_exp = (variable: Tipo) => Transformation_Expression
6. **Que son los streams y para que sirven**
   Se utiliza para procesar colecciones de objetos. Una secuencia es una secuencia de objetos que admite varios métodos que pueden canalizarse para producir el resultado deseado.
7. **Funciones más relevantes de la clase Stream**

- **Map:** El método de mapa se usa para devolver una secuencia que consiste en los resultados de aplicar la función dada a los elementos de esta secuencia.
    - *List number = Arrays.asList(2,3,4,5);*
    - *List square = number.stream().map(x->x*x).collect(Collectors.toList());*
- **Filter:** el método de filtro se utiliza para seleccionar elementos según el predicado pasado como argumento.
    - *List names = Arrays.asList("Reflection","Collection","Stream");*
    - *List result = names.stream().filter(s->s.startsWith("S")).collect(Collectors.toList());*
- **Sorted:** el método ordenado se utiliza para ordenar la secuencia.
    - *List names = Arrays.asList("Reflection","Collection","Stream");*
    - *List result = names.stream().sorted().collect(Collectors.toList());*
- **Collect:** el método collect se utiliza para devolver el resultado de las operaciones intermedias realizadas en la secuencia.
    - *List number = Arrays.asList(2,3,4,5,3);*
    - *Set square = number.stream().map(x->x*x).collect(Collectors.toSet());*
- **forEach:** el método forEach se usa para recorrer cada elemento de la secuencia.
    - *List number = Arrays.asList(2,3,4,5);*
    - *number.stream().map(x->x*x).forEach(y->System.out.println(y));*
- **reduce:** el método de reducción se utiliza para reducir los elementos de una secuencia a un solo valor.
    - *List number = Arrays.asList(2,3,4,5);*
    - *int even = number.stream().filter(x->x%2==0).reduce(0,(ans,i)-> ans+i);*

(Khem Raj Meena, FredrikKemling, s.f.)



Bibliografía

Jenkov, J. (22 de 09 de 2019). Obtenido de http://tutorials.jenkov.com/java-functional-programming/functional-interfaces.html

Khem Raj Meena, FredrikKemling. (s.f.). *www.geeksforgeeks.org*. Obtenido de https://www.geeksforgeeks.org/stream-in-java/

MohammadKhalid. (s.f.). *www.geeksforgeeks.org*. Obtenido de https://www.geeksforgeeks.org/lambda-expression-in-scala/