INSTITUTO TECNOLÓCIO SUPERIOR DE JEREZ

INGENIERÍA EN SISTEMAS COMPUTACIONALES

PROGRAMACIÓN LÓGICA Y FUNCIONAL

8° SEMESTRE

I.S.C. SALVADOR ACEVEDO SANDOVAL

“MAPA CONCEPTUAL: INTERFACES FUNCIONALES”

ALBAR DE LA TORRE GARCÍA

No. Control: 16070122

Correo: albar00@hotmail.com

JEREZ ZACATECAS

20 DE MARZO DEL 2020

Matemáticamente, que es el cálculo lambda

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 es universal porque cualquier función computable puede ser expresada y evaluada a través de él. Por lo tanto, es equivalente a las máquinas de Turing. Sin embargo, el cálculo lambda no hace énfasis en el uso de reglas de transformación y no considera las máquinas reales que pueden implementarlo. Se trata de una propuesta más cercana al software que al hardware.

Que son las “Functional Interfaces” en Java

Una interfaz funcional es una interfaz que contiene solo un método abstracto. Solo pueden tener una funcionalidad para exhibir. Desde Java 8 en adelante, las expresiones lambda se pueden usar para representar la instancia de una interfaz funcional. Una interfaz funcional puede tener cualquier cantidad de métodos predeterminados. Runnable, ActionListener, Comparable son algunos de los ejemplos de interfaces funcionales.

Antes de Java 8, teníamos que crear objetos anónimos de clase interna o implementar estas interfaces.

Cuales son las 6 interfaces funcionales del paquete java.util.function

Function <T, R>

Representa una función que acepta un argumento y produce un resultado.

Predicate<T>

Representa un predicado (función con valor booleano) de un argumento.

UnaryOperator<T>

Representa una operación en un solo operando que produce un resultado del mismo tipo que su operando.

BinaryOperator<T>

Representa una operación sobre dos operandos del mismo tipo, produciendo un resultado del mismo tipo que los operandos.

Supplier<T>

Representa a un proveedor de resultados.

Consumer<T>

Representa una operación que acepta un único argumento de entrada y no devuelve ningún resultado.

Que son las expresiones lambda

una expresión lambda, también denominada función lambda, función literal o función anónima, es una subrutina definida que no está enlazada a un identificador. Las expresiones lambda a menudo son argumentos que se pasan a funciones de orden superior, o se usan para construir el resultado de una función de orden superior que necesita devolver una función. Si la función solo se usa una vez o un número limitado de veces, una expresión lambda puede ser sintácticamente más simple que usar una función con nombre.

Sintaxis de las expresiones lambda

```
// with no parameter
() -> System.out.println("Hello, world.")

// Con un parámetro (este ejemplo es una función de identidad).
a -> a

// Con una expresión
(a, b) -> a + b

// Con información de tipo explícita
(long id, String name) -> "id: " + id + ", name:" + name

// Con un bloque de código
(a, b) -> { return a + b; }

// Con múltiples afirmaciones en el cuerpo lambda. Necesita un bloque de
código.
// Este ejemplo también incluye dos expresiones lambda anidadas (la
primera también es una clausura).
(id, defaultPrice) -> {
  Optional<Product> product = productList.stream().filter(p -> p.getId()
== id).findFirst();
  return product.map(p -> p.getPrice()).orElse(defaultPrice);
}
```

Que son los streams y para qué sirven

Un Streams es un medio utilizado para leer datos de una fuente y para escribir datos en un destino. Tanto la fuente como el destino pueden ser archivos, sockets, memoria, cadena de caracteres, y también procesos.

Los Streams se caracterizan por se unidireccionales, es decir que un Stream se utilizara solo para leer, solo para escribir, pero no ambas acciones al mismo tiempo.

Para utilizar una Stream, el programa a realizar deberá construir el Stream relacionándolo directamente con una fuente o con un destino, dependiendo si se necesita leer o escribir información.

La acción de leer información de una fuente es conocida también como input, y la acción de escribir información es un destino es conocida como output. Dentro de Java, todas las clases utilizadas tanto para el input como para el output están incluidas en el paquete Java.io

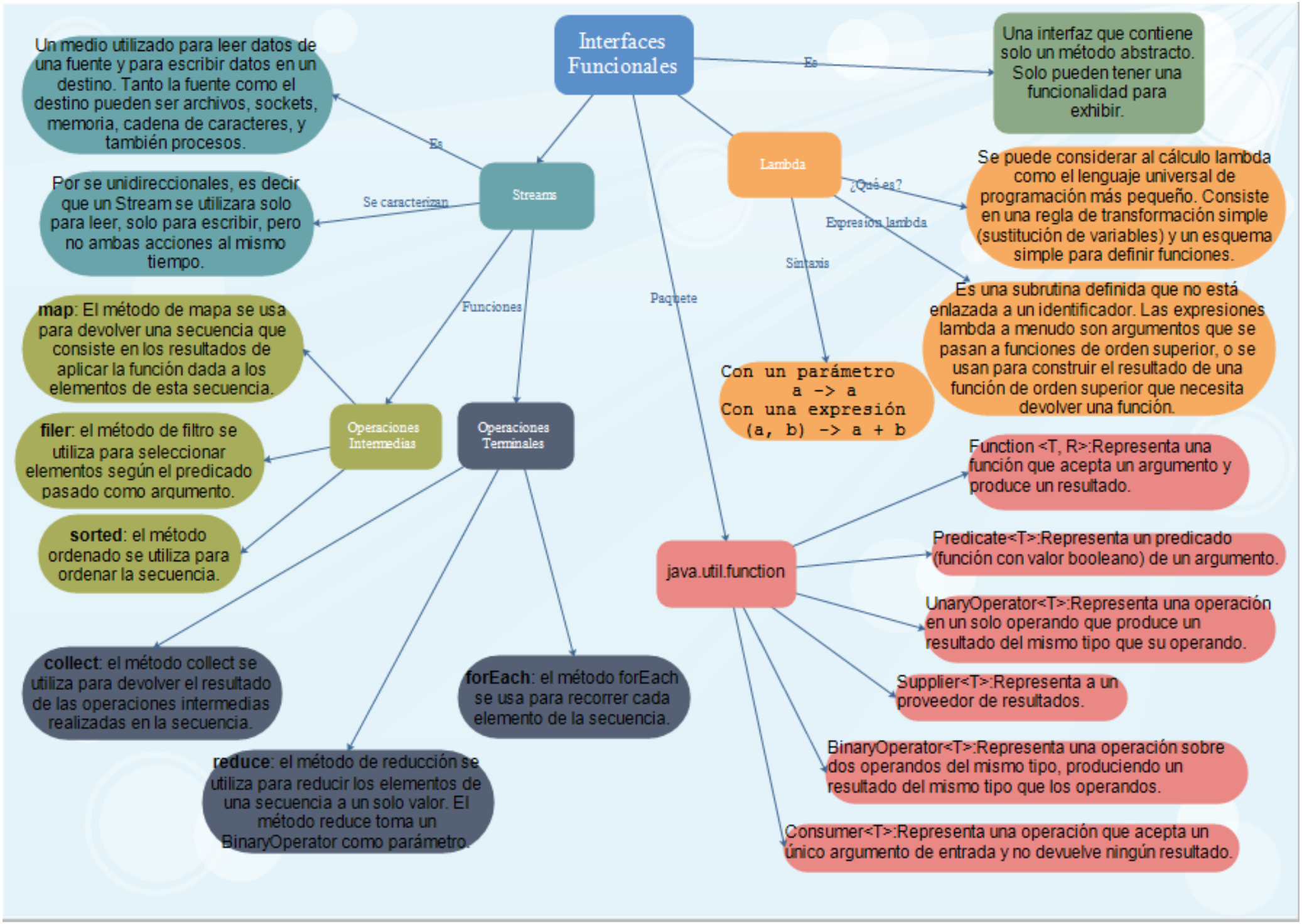
Funciones más relevantes de la clase Stream

Operaciones intermedias:

- **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.
- **filer**: el método de filtro se utiliza para seleccionar elementos según el predicado pasado como argumento.
- **sorted**: el método ordenado se utiliza para ordenar la secuencia.

Operaciones terminales:

- **collect**: el método collect se utiliza para devolver el resultado de las operaciones intermedias realizadas en la secuencia.
- **forEach**: el método forEach se usa para recorrer cada elemento de la secuencia.
- **reduce**: el método de reducción se utiliza para reducir los elementos de una secuencia a un solo valor. El método reduce toma un BinaryOperator como parámetro.


Interfaces Funcionales
Es
Una interfaz que contiene solo un método abstracto. Solo pueden tener una funcionalidad para exhibir.
Un medio utilizado para leer datos de una fuente y para escribir datos en un destino. Tanto la fuente como el destino pueden ser archivos, sockets, memoria, cadena de caracteres, y también procesos.
Es
Streams
Se caracterizan
Por se unidireccionales, es decir que un Stream se utilizara solo para leer, solo para escribir, pero no ambas acciones al mismo tiempo.
Funciones
Operaciones Intermedias
Operaciones Terminales
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.
filer: el método de filtro se utiliza para seleccionar elementos según el predicado pasado como argumento.
sorted: el método ordenado se utiliza para ordenar la secuencia.
collect: el método collect se utiliza para devolver el resultado de las operaciones intermedias realizadas en la secuencia.
forEach: el método forEach se usa para recorrer cada elemento de la secuencia.
reduce: el método de reducción se utiliza para reducir los elementos de una secuencia a un solo valor. El método reduce toma un BinaryOperator como parámetro.
Paquete
java.util.function
Function <T, R>:Representa una función que acepta un argumento y produce un resultado.
Predicate<T>:Representa un predicado (función con valor booleano) de un argumento.
UnaryOperator<T>:Representa una operación en un solo operando que produce un resultado del mismo tipo que su operando.
Supplier<T>:Representa a un proveedor de resultados.
BinaryOperator<T>:Representa una operación sobre dos operandos del mismo tipo, produciendo un resultado del mismo tipo que los operandos.
Consumer<T>:Representa una operación que acepta un único argumento de entrada y no devuelve ningún resultado.
Lambda
¿Qué es?
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.
Expresión lambda
Es una subrutina definida que no está enlazada a un identificador. Las expresiones lambda a menudo son argumentos que se pasan a funciones de orden superior, o se usan para construir el resultado de una función de orden superior que necesita devolver una función.
Sintaxis
Con un parámetro
a -> a
Con una expresión
(a, b) -> a + b