Interfaces Funcionales y el Tipo Stream

Fundamentos de Programación
Departamento de Lenguajes y Sistemas Informáticos

2023/2024

2023/24

Interfases Funcionales y el tipo Stream

Índice

- Expresiones *Lamba*
- Expresiones *Referencia a Métodos*
- *Interfaces funcionales*
  - Comparator<T>
  - Function<T,R>
  - Predicate<T>
  - Consumer<T>
  - Suplier<T>
- Tipo *Stream*<T>

Tipo Stream

2023/24

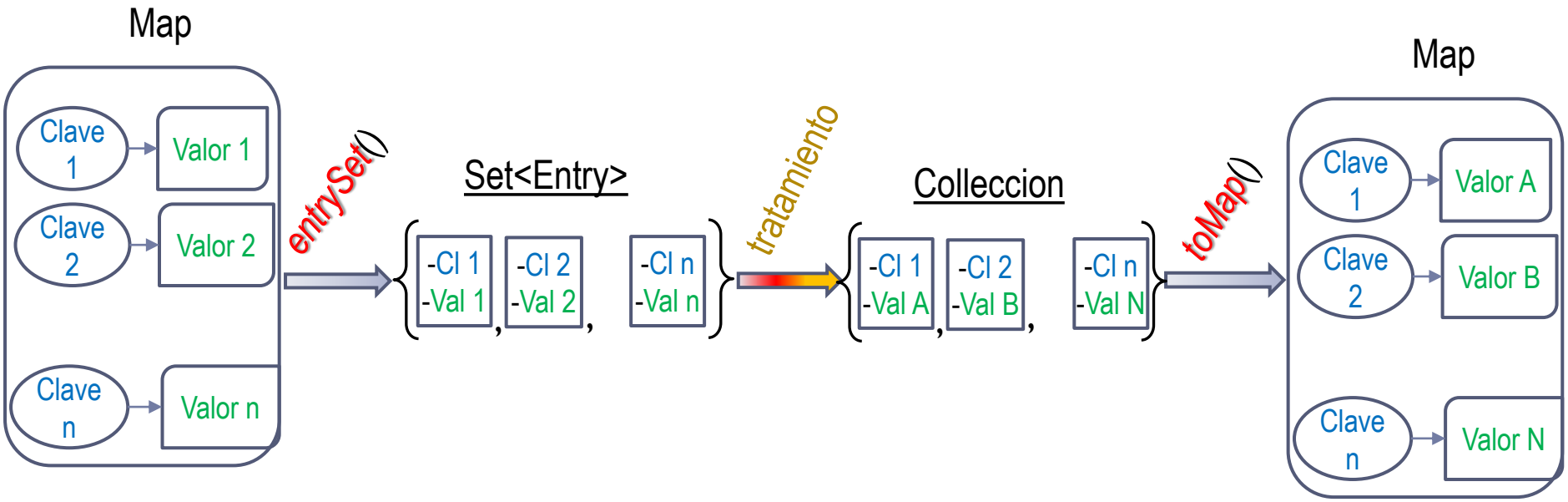
Métodos de Collectors:

Collectors.toList()
Collectors.toSet():
Collectors.toCollection(Supplier<C> constructor)
Collectors *groupingBy*(*con 1, 2 o 3 parámetros*)
Collectors.counting()
*Collectors.collectingAndThen(*método 1, método2*)*
Collectors.mapping(*Function*<T,R>, Collection)
Collectors.flatMapping(*Function*<T,R>, Collection)
Collectors.summingInt(propiedad numérica)
Collectors.summingLong(propiedad numérica)
Collectors.summingDouble(propiedad numérica)
Collectors.averagingInt(propiedad numérica)
Collectors.averagingLong(propiedad numérica)
Collectors.averagingDouble(propiedad numérica)
Collectors.maxBy(*Comparator*):
Collectors.minBy(*Comparator*):
Collectors.toMap(*Function*<T,K>, *Function*<T,V>)

Tipo Stream

El método Collectors. *toMap* (*Function*<T,K>, *Function*<T,V>):

- Es un método que permite, *construir un mapa a partir de un objeto con dos propiedades cualquieras*, previamente "emparejadas". No admite claves repetidas, por lo que la propiedad que se elija como clave no puede repetirse. Resulta muy útil para obtener un **Map** desde una colección de objetos **Entry**.



Tipo Stream

El método Collectors. *toMap* ():

Ejemplo. Dado un número entero "n" devuelve un mapa con los destinos y el número de vuelos, que tengan más de n vuelos a dichos destinos.

1) Realizamos un mapa Map<String,Long> que cuente cuantos vuelos hay a cada destino:

Map<String,Long>→
```
sv.collect(Collectors.groupingBy
        (Vuelo::destino, Collectors.counting()))
```

2) Convertimos el Map en una colección (Set) de objetos *Entry* con pares (destino-nº vuelos) y se filtran los pares cuyo valor es mayor que *n*. Es decir que tengan más de "*n*" vuelos:

```
entrySet().stream().filter(par->par.getValue()>n)
```

Tipo Stream

3) Por último, generamos un mapa a partir de los pares filtrados.

```
collect(Collectors.toMap(par->par.getKey(),
                         par->par.getValue()))
```

4) ¡El ejercicio completo es!:

```
Map<String,Long> →sv.collect(Collectors.groupingBy(
                    Vuelo::destino,Collectors.counting()))
.entrySet().stream().filter(par->par.getValue()>n))
.collect(Collectors.toMap(par->par.getKey(),
                          par->par.getValue()));
```

Volvemos a tener un Map

Tipo Stream

2023/24

¿Cómo sería el ejercicio si en lugar de pedir un Map<String,Long>, se pidiese un Map<String,Integer>?

```
sv.collect(Collectors.groupingBy(Vuelo::destino,
        Collectors.collectingAndThen(Collectors.counting(),
                                     cont->cont.intValue())))
    .entrySet().stream().filter(par->par.getValue()>n)
    .collect(Collectors.toMap(par->par.getKey(),par->par.getValue()));
```

Tipo Stream

El método Collectors. *groupingBy* ()

Ejecutando nuestras propias funciones

En ocasiones, la obtención de los valores del mapa que hay que devolver no tiene un método de *Collectors* apropiado para aplicar en el *groupingBy*.

En este caso, podemos sustituir la llamada a un método de *Collectors*, por la llamada a *otro método privado hecho por nosotros*.

Por ejemplo: Se define la mediana de una serie de números ordenados como aquel que deja tantos valores a la "derecha" como a la "izquierda". Así, si tenemos los valores 12,15,28,31,40,42,47 (7 valores), la mediana es el *31*. Si fuesen pares la mediana es el promedio de los dos centrales: Si 12,15,28,31,40,42,47,50 (8 valores) la mediana es (31+40)/2=*35.5*

Tipo Stream

Ejemplo:

Calcular un Map<String,Double> que a cada destino le haga corresponde la mediana del número de pasajeros de los vuelos a dichos destino.

1) Se calcula, inicialmente un mapa que obtenga como valores las listas a las que hay que calcularles las medianas (lista de número de pasajeros)

```
sv.collect(Collectors.groupingBy
     (Vuelo::destino,
      Collectors.mapping(Vuelo::númeroPasajeros,
                         Collectors.toList()))
```

2) Ahora tratamos las listas generadas por el mapping, con un *método* realizado por nosotros, por lo que ese ***mapping*** habrá que ponerlo como 1er parámetro de un ***collectingAndThen***, el 2º parámetro será una expresión lambda que invoque a *nuestro método*.

Tipo Stream

es decir:

```
sv.collect(Collectors.groupingBy
(Vuelo::destino,
Collectors.collectingAndThen(
Collectors.mapping(Vuelo::númeroPasajeros,
                   Collectors.toList()),
                   lista->calculaMediana(lista))));
```

3) Ya sólo queda hacer el método privado *calculaMediana* que reciba una lista de Integer y devuelva un Double (debe ser Double por si el número de elementos es par y se calcula un promedio, que probablemente tendrá decimales)

Tipo Stream

2023/24

```
private Double calculaMediana(List<Integer> lista) {
   List<Integer> aux=lista.stream()
          .sorted(Comparator.naturalOrder())
          .collect(Collectors.toList());
   Double res=null;
   Integer n=aux.size();   //número de elementos de la lista
   if (n%2==0)             //Si n es par hay que calcular el promedio
     res=(aux.get((n/2)-1)+aux.get(n/2))/2.0; //promedio
   else
     res=(double)aux.get((n-1)/2); //el central es Integer → casting a Double
   return res;
   }
```

Ejercicio Aeropuerto

- Realice los ejercicios del ***EnunciadoAeropuerto10***

  Apartado 23