

Que es un functional Interface?

① Es una interfaz con un solo método abstracto
↓

Se marca abstracto por que en java 8 podemos tener mas cosas que no son abstractas, en java 7, esto seria redundante debido a que en esa version de java solo se aceptan esos metodos y constantes, por que no conforma a eso por que pudi haber ~~el~~ un interfaz sin ~~en~~ un metodo pero no abstracto, en ese caso, no se puede llamar functional interface

②

En los metodo de la clase objeto no cuenta, por ejemplo ~~se~~ equals, clone, etc...

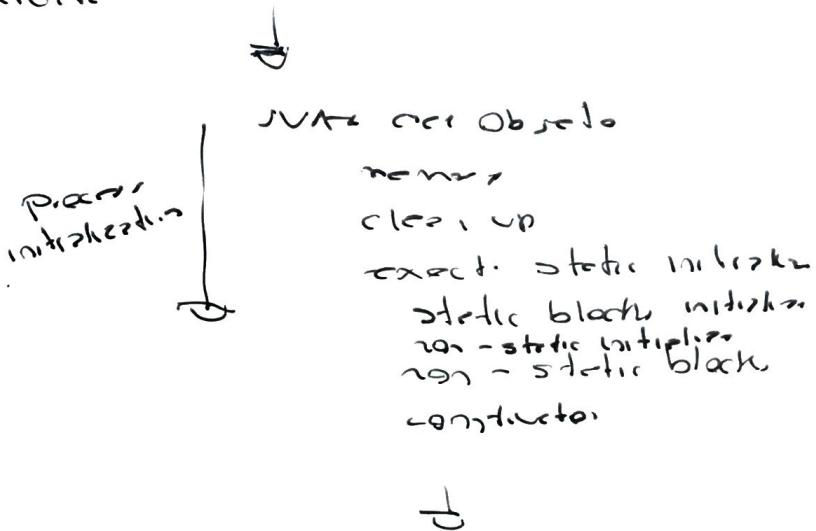
③ Podemos anotar @FunctionalInterface,
o optional, el compilador podra resolver si es funcional o no, si no lo es nos dara un error de compilacion

Un lambda puel ponerse en una variable
Si y podemos pasarlo como parametro
y podemos tambien retornerlo

Una expresión lambda es un objeto

No es si, es un objeto sin una identidad

Es diferente a otra clase anónima y a lambda
en que en clase anónima usamos "new" y
cada vez que la utilizamos tenemos que llamar
por el proceso de inicialización, lambda no, es
más eficiente



↓

No tenemos más
que hace co. lambda
expresión, no tiene

* No debemos de usar la lambda como
un objeto normal, por ejemplo NO
llamar al método `exec`, hacerlo
en todo tipo de método de clase objeto.

Primer cod de se mire al método.
-> Usar en muchos lugares

- Supplier → No tiene ningún objeto y provee un nuevo objeto de tipo T
 $T \text{ get();}$
 Consumer → Recibe un objeto y no regresa nada
 $\text{void accept}(T+);$
 Bi consumer → acepta dos objetos, no regresa nada
 $\text{void accept}(T+, U+)$
 Predicador → Toma un objeto y nos regresa un booleano.
 $\text{boolean test}(T+);$
 Bipredicador → Toma dos objetos y nos regresa un booleano.
 $\text{boolean test}(T+, U+);$
 Function → Toma un objeto y retorna un objeto de otro tipo
 $\checkmark u \quad R \text{ apply }(T+);$
 Bi función → 2 objetos y retorna un objeto.
 Binario Operador → Recibe 2 ob., del mismo tipo y regresa otro de mismo tipo
~~Unary~~ $\xrightarrow{\text{Bi función}}$
~~Unary Operador.~~
 Unary Operator / Funci. → Recibir un objet.
 Viene con otro objeto del mismo, los anteriores no regresan un objeto de mismo tipo,
 entre si

Podemos omitir el tipo de los parámetros
 de lambda, ya que el compilador lo resuelve, ejemplo
 $\text{Comparador } <\text{String}> c = (\text{String } s_1, \text{String } s_2) \rightarrow \dots$
 Se convierte en
 $\text{Comparador } <\text{String}> c = (s_1, s_2) \rightarrow \dots$

Método `reflexo`, (~~se~~ `>hortcut`)

Obtiene los paquetes y paquetes de módulo.
ejecutar por medio de : `comando`.

Comparador <`Integre`> `c = (i1, i2) → Integre.compar(i1, i2)`

Se ve lo

Comparador <`Integre`> `c = Integre::compar`

Otro ejemplo

Consumo <`String`> `c = s → System.out.println(s)`

Se pasa a

Consumo <`String`> `c = System.out::println`

—
Como podemos procesar mas información a

Paralelo con lambda

Dobla la tareas aplicacion , sucede esto.

Hasta utilizando colecciones, (`list, map, etc`),
para que podamos trabajar en las colecciones

Ejemplo

`List<Integre> num = ...`

`list.forEach(System.out::println)`

`list.forEach(System.out::println)`

— \rightarrow Distinct Regla
Podemos por ejemplo encadenar `consumo`

con "andThen", esto no permite

que quede código EJ.

`list<Integre> num2 =` `new Filter(lv1); num2.add(c1)`

`consumo<Integre> c1 = System.out::println;`

`consumo<Integre> c2 = num2::add`

`num2.forEach(c2.andThen(c2)System.out::println)`

Default method.

Debido a que si yo pongo un nuevo metodo en un interface, comprender el contexto y tener que implementar todos mis clientes, no se puede hacer eso, lo que se puede hacer es sobre escribir el metodo default en el interface, esto es que si se pone la implementacion de un metodo default en un interface, se puede hacer lo siguiente:

- Implementacion de igual
- poner metodos estaticos en interfaces

Ej) Implementacion default

```
default void foEach(Consumer<E> consumer) {
    for (E e : this) {
        consumer.accept(e);
    }
}
```

Podemos hacer mas cosas.

Predicado <String> p1 = s -> s.length() < 20;

Predicado <String> p2 = s -> s.length() > 10;

Predicado <String> p3 = p1.and(p2)

Algo con

con metodo

1) Equals

2) equals

3) true

cuando

un metodo

default, no tiene

que escribir implementacion

listas

Map / Filter, Reduce: Diagrama que recorre 1 o
Personas, orden 20 2500 v 22
1 Pasa, toma una lista de personas, digamos
3 Personas, toma que mapea la a entero
v transforma 3 enteros en un lista de enteros

- 2 De la lista de enteros, toma otra lista para
Pred que con meno, elemento
- 3 Reducirlos, por ejemplo a sumar toda veces.
el promedio

→ Que es un stream S-licencia

Es una typical interface, es decir podemos tener
un sistema de personas, de enteros, etc

is like a collection but not

De conflictos con listas o colecciones por que
trabajan sobre la informacion como una lista,
pero su potencia esta en la EFICIENCIA
EN EL PROCESAMIENTO, ya sea stream
pequeña o largo.

- Liberan el potencial de procesamiento multicore
- Todo se hace en un Pipeline, es decir que
solo se hace en un solo punto, (como
en un dominio?).)

Tambien existen

Int Stream }
Long Stream } Primitivo.
Double Stream }

Passo. stream(). forEach(...)

Filtros opcionales p se usar con predicados (ons) parametrizado.

Los predicados and, or no respectan la prioridad default de sever, ser procesados en el orden en los que los pone.

Ej de lo anterior:

$$pt. \gtreqless(R2). \text{or}(P1) \rightarrow (P1 \text{ AND } R2) \text{ on } P$$
$$P1. \text{or}(P2). \text{and}(P3) \rightarrow (P1 \text{ OR } P2) \text{ AND } P3$$

El filtro regresa un stream, con un new instancia

Stream s = filter().

Recordaros, qe un stream no puede ~~almacenar~~ almacenar ninguna informacion, entonces cuando retorna un stream qe es lo qe pasa?

En realidad no regresa nada, solo es un paso intermedio para un resultado final, otra forma de verlo es decir qe es un declaracion de una expresion o un paso mas en medio del procesamiento

* Todo lo qe recorre un stream es llamado "lazy"

Diferencias entre intermedias operaciones y final operaciones

- Si un metodo no regresa un stream, quiere decir que es una operacion intermedia o la diferencia con un final, es que la final dispara el o los operadores contenidos y si terminase con un operador intermedio save poder que sea poniendo otra intermedia o final como se puede ver?

continuacion

List \rightarrow List

List create = new ArrayList

stream. peek (System.out:: println)

- Filter (p1, and (p2))

- peek (emptyList():: add) \rightarrow peek es similar



No resulta en
nada, si quieras
que suceda, se usa así

2 for each para
regresar un stream
por lo cual se
convierte en una
operacion intermedia

Stream. peek (System.out:: println)

- filter (p1, and (p2))

- ForEach (emptyList:: add)

* como forEach no regresa nada, este sera el trabajo para que se ejecute todo lo que contiene el stream.

forEach (Consumo)

peek (Consumo) \rightarrow List

filter (Predicado) \rightarrow List

Map Function

Si tenemos una lista

$$a = [1, 2, 3, 4, 5]$$

$$b = [1, 4, 5]$$

$$c = [3, 5, 10]$$

con map se le transforma con un `map`

$$[[1, 2, 3, 4, 5], [1, 4, 5], [3, 5, 10]]$$

Si lo hacemos flatMap se le sigue el
mismo

$$[1, 2, 3, 4, 5, 1, 4, 5, 3, 5, 10]$$

FlatMap

Function <List<Integer>, Stream<Integer>> FlatMap
 $= l \rightarrow l.stream();$ ↗

4

Entre una lista de enteros

Sale un stream de enteros ↗

Reduce

Optional, puede no regresar nada.

como usar Optionals

• `ifPresent()`

• `orElse()`

• `orElseThrow(MandatoryException.class)`

Relations Aggregation ①

max, min, count

allMatch, noneMatch, anyMatch

findFirst(), findAny,

Relations son operadores terminales

list.stream().reduce

O, → identity

Intercalacion → binary operator

los operadores son necesarios pq que los valores default no pueden ser siempre definidos

* → -P

* NO se puede usar un stream de 1 vez con un finalizar, como por ejemplo
forEach, max, min... etc

* Mutable Relation ②

Collectors Cambiar de una lista a un map

Person.stream().collect(

Collector groupingBy

Person::getAge,

Collector<.>.mapping(

Person::getName,

Collector<.>.joining(",")

))

1/

Con collectors podemos sumar mas resultados.
Por ejemplo si una lista es un map y un
despues podemos verlo en el contexto de ese
map.

— ~~Variables~~ variables clásicas

Con lambda, podemos codigos como parametros

- Es ejecutado "despues"
- Es ejecutado en otro contexto (otra thread)

— Lams

An adicida que nuestro metodo tenga
mas de una linea y tambien retorna un valor,
tenemos que ponerlo entre curly braces y
poner la palabra return, en otra linea, si
es solo una linea, podemos omitir el "return"

— Parámetros en lambda expresión

- Podemos omitir el tipo
- Podemos usar final para que nuestros parametros
no sean modificados por el codigo
- Podemos usar anotaciones

¿ Function 1 c. entrada
2 c. salida ?

Function <Person, Interest> f -> p.greet
Function <Person, Interest> p = Person::greetA

Binary Operator → - Toma 2 objetos > misma
- Regresa 1 diferente > tipo

Method References, no son métodos estáticos
no nos debemos confundir

—
Consumer → - Toma un objeto
- No regresa nada

Al parecer por medio de las flechas sobre
los diferentes objetos, también puede omitir
los tipos, por ejemplo Binary Operator
o Consumer

—
Non static var T, the same of the
generic type

static var for example U as ~~set~~
type

—
Para crear nuevos APIs de código legado
tendremos que hacer uso de default methods,
lambda, static methods

Isaac Sosa

Actions para construir streams

④ List<Person> persons ...

```
Stream<Person> stream = persons.stream();  
// llamando al metodo .stream() de la colección  
// cabe señalar que la info se queda en la colección  
// no en el stream, que no almacenó info
```

⑤ Stream<?>

```
Stream.empty() // un empty stream  
Stream.of("one") // es singleton stream  
Stream.of("one", "two") // with several elements  
Stream.generate(() -> "one"); // es content stream  
Stream.iterate(0, s -> s + "+") // growing stream
```

Third) LocalDate.now().index()

int [↓] Stream random

```
IntStream s = "hello".chars() // stream on the  
letters of a  
string
```

- regex

- Files para leer líneas

Stream.Builder<String> builder = Stream.builder()
add → pude ser omitido

accept → igual que add pero sin
poder omitirlo

• Después de invocar .build si
se queremos añadir después de construir
lanzará una excepción

Method references no soportan argumentos

Predicate<String> p = s → s.length() > 2
?

Argumento
no soportado

Comprobación de tipos.

- Como solo son interfaces funcionales, y solo deben tener un método abstracto, esa simplifica todo y de cara menor, dependiendo de el tipo de operación (Predicate por ejemplo conoce lo siguiente)
 - El tipo de entrada y salida
 - Si los parámetros de entrada y el return son compatibles
 - Lo mismo aplica para los excepciones si existieren.

Existen functional interfaces para tipos primitive, por ejemplo

Int Predicate

Int Function

Int ToDouble Function

Patrón para construir una API

Ver que parámetros recibir

Ver que parámetros regresar

Crear métodos defaults y utilizar la interface

metodo to function interface

- putIfAbsent
 - replace (Key, newValue) → (Key, existingValue, newValue)
 - replaceAll
 - remove + nuevo, es especificando el Key-Value, si esto al ~~value~~ Key pero el value no hace match, no se remueve
 - compute *
 - computeIfAbsent
 - computeIfPresent
- } Regresan un valor

~~versión compute~~

Darle un parámetro como Key
 si ese Key existe una cosa hace y si se
 inicializó el tipo del value o no lo hace
 si como un if-else para ejecutar dicha
 logica cuando se agrega un valor al map

Map<String, Map<String, Person>

Map<String, List<Person>

- Merge

* Records

Comparador: comparar (Person::getAge)

- Remover
Repeticiones

→ Crear nuevas instancias con las que
logre vaciar la lista sin una
instancia de más o falta.

Son inmutables

Asociatividad → se reduce el problema con la multiplicación

$$(1 + 5) + 3 = 9$$

$$1 + (5 + 3) = 9$$

$$(1 \times 5) \times 3 = 15$$

$$1 \times (5 \times 3) = 15$$

Es un nuevo tipo de bucle, si podemos multiplicar los datos y recalcular valores diferentes cada vez que se ejecuta el algoritmo, probablemente las listas no serán coherentes y determinar valor en su reducción

* Asociatividad → el orden de los problemas no altera el producto

No nos mandan un caso de comprobación, pero el resultado es correcto

Como se procesa un dato en paralelo



1 punto caso 1

$$\underbrace{1 + 2}_{3 + 4}$$

$$\underbrace{3 + 4}_{\sqrt{11 + 1}}$$

10 puntos caso 2

$$\underbrace{7 + 4}_{\sqrt{11 + 1}}$$

$$\sqrt{12}$$

* Tienen que ser asociativas.

esto se hace a dif.

casos no se alteran

se altera

19

Reduction steps problems

- Association
- Non-existing identity types, channels for certain types of associations

El primer de los problemas es la operación para que se pueda comparar en paralelo, si no es asociativa la función, no se podrá combinar correctamente en paralelo.

El segundo, cuando las operaciones no tienen identidad si pasamos un valor por defecto como resultado, corrompería el resultado porque el valor de Max

— Optional → es un wrapper como la otra cosa
creada no tiene un identificador.
Quiero decir que es un wrapper que puede estar vacío

— More filter, reader → Streams must interleave
la mejor manera sería hacerlo, tenerlos
3 métodos separados, que tendrían que iterar
toda separadas, almacenarlos, transformarlos
y reducirlos, especificarlos si hablamos
de millones de registros,

— Peak para la iteración, no debe usarlo
en producción, solo se debe usar para preparar
el desarrollo y test

Como reconocemos un operador terminal

- Los operadores intermedios no devuelven un stream
- Los operadores terminales sí devuelven un stream, pueden devolver resultados como PC's o no un stream, o devolver void

Se devuelven vacios de información en un stream Limit, skip

Intermediate operation

skip → salte n numero de elemento

limit → limita el set de elementos a 1 numero especificado

Reducciones simples (Match, find, count, reduce)

Any Match → cualquier cumple el predicado

All Match → todos deben cumplir el predicado

None Match → ninguno debe cumplir el predicado

Todos son operaciones terminales
y tienen mecanismos de corte
circular para no iterar de mas

FindFirst

Find Any

↓

* Devuelve un
Operador

→ Ambas necesitan un predicado
(condición) y a ambos casos,
no puede devolver nada.

1) Si la lista es vacia?

2) Si no está vacia ¿puede
no cumplir el predicado?

General Reduction

- reduce() → with identity
- reduce() → without identity
- reduce() → for parallel

} Necesitar cumplir con
associatividad y operador
de que existan
identities debemos tener
en cuenta, por ejemplo
`max()` → con numeros
positivos y negativos
se ignoraran los
negativos en el
resultado

3er curso

Streams, Collectors and Options