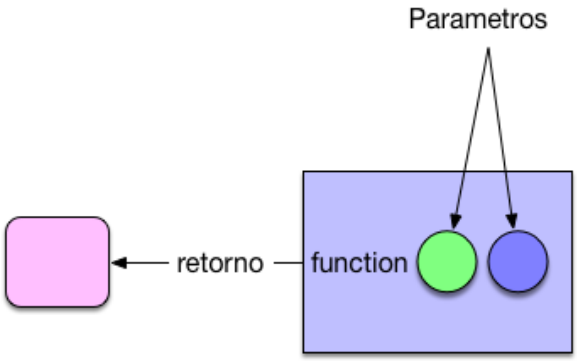
Blog sobre Java EE

Java 8 Functional Interfaces y sus tipos

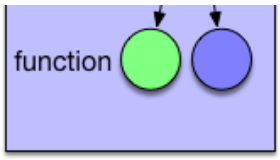
18 abril, 2018 por Cecilio Álvarez Caules — Deja un comentario

Cada día es más importante conocer los tipos de **Java 8 Functional Interfaces**. **¿Cuales son los tipos fundamentales de interfaces funcionales en Java?.** Una función es un bloque de código que recibe varios parámetros y devuelve un resultado.



Java 8 Functional Interfaces y Consumers

Ahora bien existen variaciones sobre el concepto general de función. Por ejemplo podemos disponer **de una función que reciba parámetros** pero que no devuelva nada.

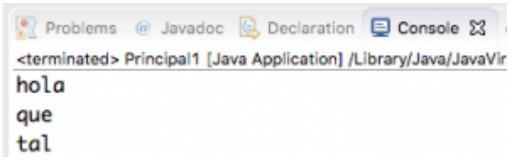


Consumer

Este es el concepto que se conoce como consumidor (consumer) . **Un ejemplo clásico en Java 8 es el uso de un Stream con una sentencia forEach().**

```
1  package com.arquitecturajava;
2
3  import java.util.Arrays;
4  import java.util.List;
5
6  public class Principal1 {
7
8      public static void main(String[] args) {
9
10         Lis<String> lista=Arrays.asList("hola","que","tal");
11         lista.stream().forEach((x)->System.out.println(x));
12
13     }
14
15 }
```

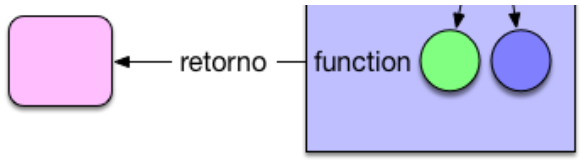
En este caso la función **forEach recibe una función consumidora** que tiene un parámetro x e imprime los valores por la consola no devuelve nada



Es el tipo de interface más sencillo.

Java 8 Functions

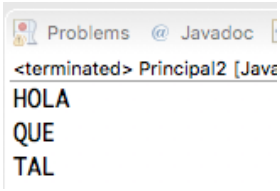
El siguiente tipo de interface son los que habitualmente **se llaman Functions** reciben uno o más parámetros y retornan un resultado:



Este es el caso de uso de un stream cuando utili**za el método map recibe como parámetro un tipo Function:**

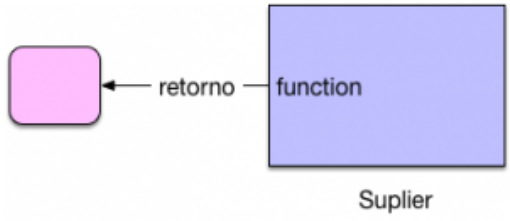
```
1  package com.arquitecturajava;
2
3  import java.util.Arrays;
4  import java.util.List;
5
6  public class Principal2 {
7
8      public static void main(String[] args) {
9
10         List<String> lista=Arrays.asList("hola","que","tal");
11         lista.stream().map((x)->x.toUpperCase()).forEach((x)->System.out.pr
12     }
13 }
```

En este caso nos imprimirá en mayusculas los valores en la consola:



Java 8 Suppliers

Este caso es algo más complejo ya que se trata **de una función que no recibe parámetro alguna y devuelve un resultado**. Nos puede parecer algo absurdo de entrada ya que una función debe recibir algo. Sin embargo hay a veces que necesitamos **generar un nuevo contenido prácticamente desde la nada.**



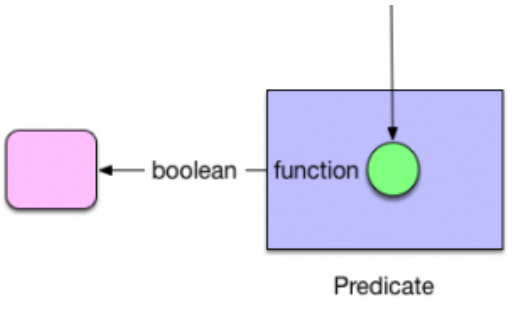
```
import java.util.stream.Stream;

public class Principal3 {

    public static void main(String[] args) {

        List<Persona> lista=Stream.generate(Persona::new)
                .limit(100)
                .peek((p)->p.setNombre("pepe"))
                .collect(Collectors.toList());


        for (Persona p: lista) {

            System.out.println(p.getNombre());
        }

    }
}
```

En este caso el **método generate de un Stream** recibe un Supplier q**ue es una función sin argumentos de entrada y que devuelve un resultado**. En este caso usamos un método de referencia e invocamos al constructor de la clase Persona que es un constructor vacío y devuelve un objeto nuevo :

```
package com.arquitecturajava;

public class Persona {

    private String nombre;

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

}
```

Luego simplemente los recorremos e imprimimos por la consola:

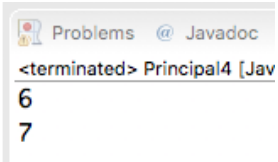
```
pepe
pepe
pepe
pepe
pepe
pepe
pepe
```



Se usan de forma intensiva en operaciones de filtrado.

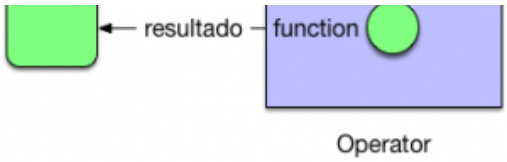
```
1  package com.arquitecturajava;
2
3  import java.util.Arrays;
4  import java.util.List;
5  import java.util.stream.Collectors;
6  import java.util.stream.Stream;
7
8  public class Principal4 {
9      public static void main(String[] args) {
10         List<Integer> lista=Arrays.asList(1,3,4,5,6,7,1,2);
11         lista.stream().filter((x)->x>5).forEach(System.out::println);
12     }
13 }
```

El resultado lo vemos en la consola:



Java 8 Operators

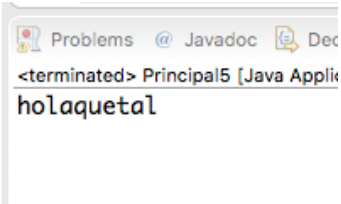
El uso de Java 8 Operators es otro caso especifico de los Function interfaces . Se trata de interfaces funcionales que **reciben un tipo de parámetro y devuelven un resultado que es del mismo tipo.**



Un ejemplo típico usando streams es el resultado que genera el método reduce ya que se encarga de agrupar un conjunto de elementos del mismo tipo:

```
1  package com.arquitecturajava;
2
3  import java.util.Arrays;
4  import java.util.List;
5  import java.util.Optional;
6
7  public class Principal5 {
8
9      public static void main(String[] args) {
10
11         List<String> lista=Arrays.asList("hola","que","tal");
12         Optional<String> resultado=lista.stream().reduce(String::concat);
13         if(resultado.isPresent()) {
14
15             System.out.println(resultado.get());
16         }
17     }
18
19 }
```

El método reduce de los Java 8 Functional Interfaces usa una función operator .Nos apoyamos en un método de referencia concat que recibe un string y devuelve un string agrupado . El resultado será un string optional que aparece en la consola con todo el texto fusionado.



Acabamos de ver un pequeño resumen de los diferentes tipos de **java 8 functional interfaces** y la tipología que soportan.

Otros artículos relacionados:

1. Java Lambda reduce y wrappers

PDF

Archivada en: Java 8

Leave a Reply

Be the First to Comment!

Start the discussion

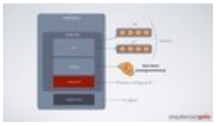
Subscribe

BUSCAR

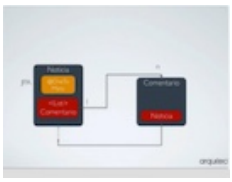
Buscar en este sitio ...

Java JDBC

Servlets

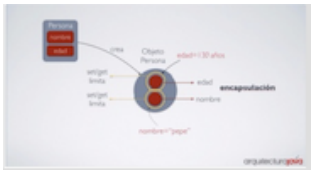


Introduccion JPA



Mis Cursos de Java

Programación Orientada a Objeto en Java



Java APIS Core

Java Web

Pack Java Core

POPULAR

- Spring REST Client con RestTemplates
- Nuevo Curso:Arquitectura Java Sólida con Spring 4.3 y Anotaciones
- Angular 5 Hello World y su funcionamiento
- PostMan App con Spring REST
- Arquitecturas REST y sus niveles
- Java 9 Modules y el concepto de modularidad
- Java Interfaces y el concepto de simplicidad
- El concepto de Java Annotations y su funcionamiento
- Java 8 Lambda Syntax ,simplificando nuestro código
- Spring REST Test utilizando Rest Assured

CONTACTO

contacto@arquitecturajava.com

LO MAS LEIDO

- ¿Qué es Spring Boot?
- Java 8 Functional Interfaces y sus tipos
- Java Constructores this() y super()
- Usando Java Session en aplicaciones web
- 10 Características que me gustan de TypeScript
- Java Iterator vs ForEach
- Introducción a Servicios REST

REST JSON y Java

Arquitecturas REST y sus niveles

Usando el patron factory

Ejemplo de JPA , Introducción (I)

Uso de Java Generics (I)

Mis Libros

¿Qué es un Microservicio?

Comparando java == vs equals

Spring MVC Configuración (I)