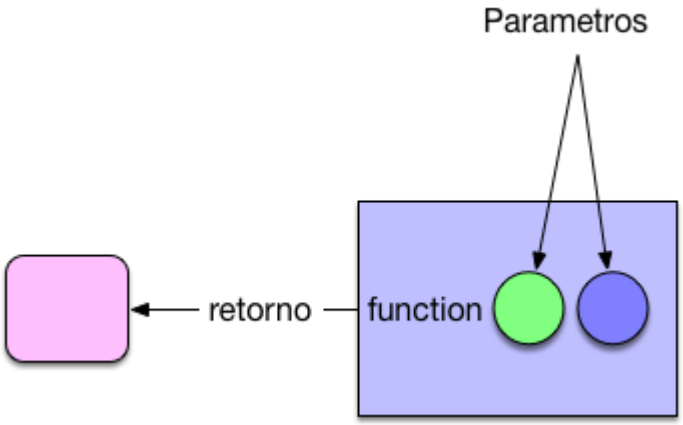
Tabla de Contenidos

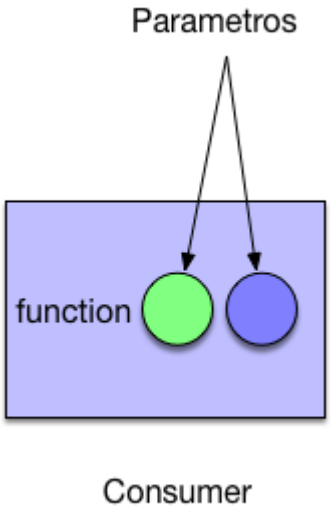
- Java 8 Functional Interfaces y Consumers
- Java 8 Functions
- Java 8 Suppliers
- Java 8 Predicates
- Java 8 Operators
- Otros artículos relacionados:

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.



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().

```
package com.arquitecturajava;

import java.util.Arrays;
import java.util.List;

public class Principal1 {

    public static void main(String[] args) {

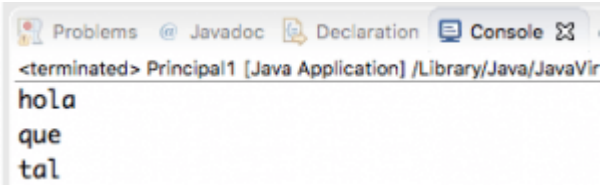
        Lis < String > lista = Arrays.asList("hola", "que", "tal");
        lista.stream().forEach((x) - > System.out.println(x));

    }

}
```

En este caso la función forEach recibe una función consumidora que tiene un parámetro x e

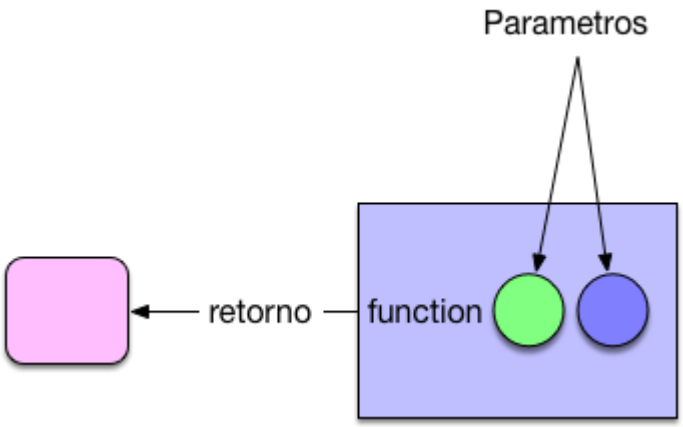
imprime los valores por la consola no devuelve nada



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 utiliza el método map recibe como parámetro un tipo Function:

```
package com.arquitecturajava;

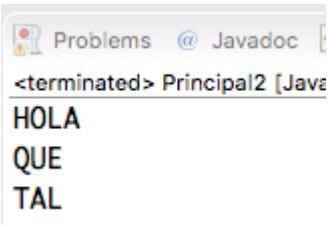
import java.util.Arrays;
import java.util.List;

public class Principal2 {
```

```
    public static void main(String[] args) {

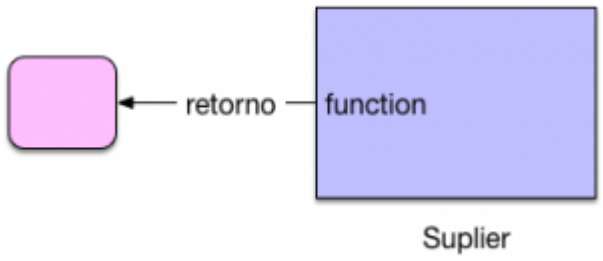
        List < String > lista = Arrays.asList("hola", "que", "tal");
        lista.stream().map((x) - > x.toUpperCase()).forEach((x) - >
System.out.println(x));
    }
}
```

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.



Veamos un ejemplo en código:

```
package com.arquitecturajava;

import java.util.List;
```

```
import java.util.stream.Collectors;
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 que 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;
    }

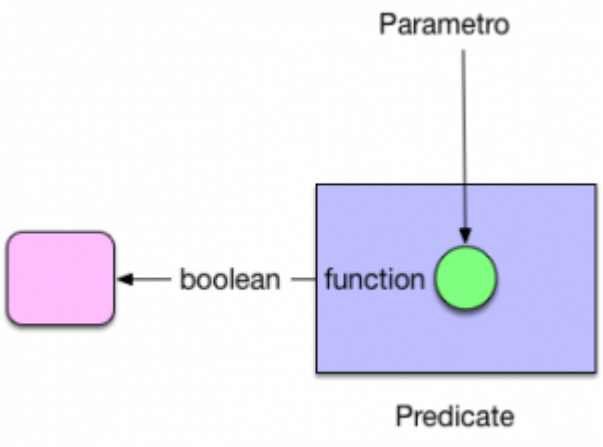
}
```

Luego simplemente los recorremos e imprimimos por la consola:

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

Java 8 Predicates

Los interfaces de tipo predicado son una especialización de los Functions ya que reciben un parámeto y devuelven un valor booleano.



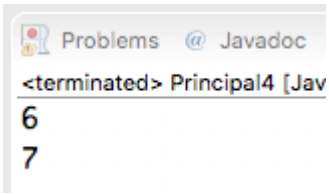
Se usan de forma intensiva en operaciones de filtrado.

```
package com.arquitecturajava;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

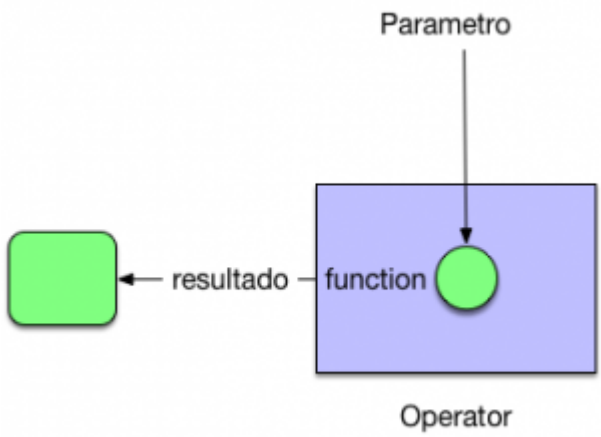
public class Principal4 {
    public static void main(String[] args) {
        List < Integer > lista = Arrays.asList(1, 3, 4, 5, 6, 7, 1,
2);
        lista.stream().filter((x) - > 5).forEach(System.out::println);
    }
}
```

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:

```
package com.arquitecturajava;

import java.util.Arrays;
import java.util.List;
import java.util.Optional;

public class Principal5 {

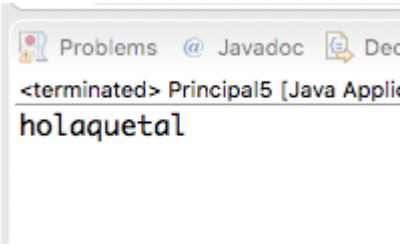
	public static void main(String[] args) {

		List<String> lista=Arrays.asList("hola","que","tal");
		Optional<String>
resultado=lista.stream().reduce(String::concat);
		if(resultado.isPresent()) {

			System.out.println(resultado.get());
		}
	}
```

```
}
```

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
2. Java 8 Lambda Expressions (I)
3. Java Stream File y manejo de ficheros
4. Java Stream Filter y Predicates

Artículos externos

1. Java Streams