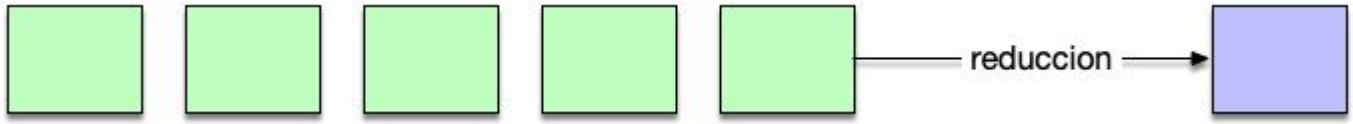
Tabla de Contenidos

- Java Clásico
- Java Stream Reduce
  - Simplificación de sintaxis
- Otros artículos relacionados

El concepto de Java Stream Reduce , es uno de los más importantes a nivel de programación funcional ya que cubre las operaciones de Reducción que nos permiten convertir una lista de elementos X en un resultado Y único . Esto en un principio nos puede parecer un poco curioso pero es bastante útil ya que muchas operaciones comienzan con n elementos y terminan en un único resultado un ejemplo podría ser la suma de n elementos.



Java Clásico

Vamos a ver un ejemplo sencillo supongamos que disponemos de una lista de gastos y queremos calcular su total. Normalmente en una situación clásica deberíamos utilizar un bucle for .

```
package com.arquitecturajava;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import java.util.stream.Stream;

public class Principal4 {

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

```
    List<Integer> gastos= new ArrayList<Integer>();
    int total=0;
    gastos.add(100);
    gastos.add(200);
    gastos.add(300);
    for(int gasto : gastos) {
      total+=gasto;
    }
    System.out.println(total);

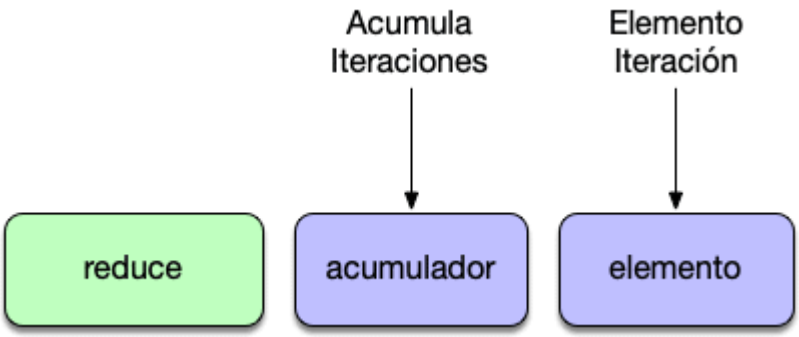
  }
}
```

el resultado lo podemos ver salir por la consola:

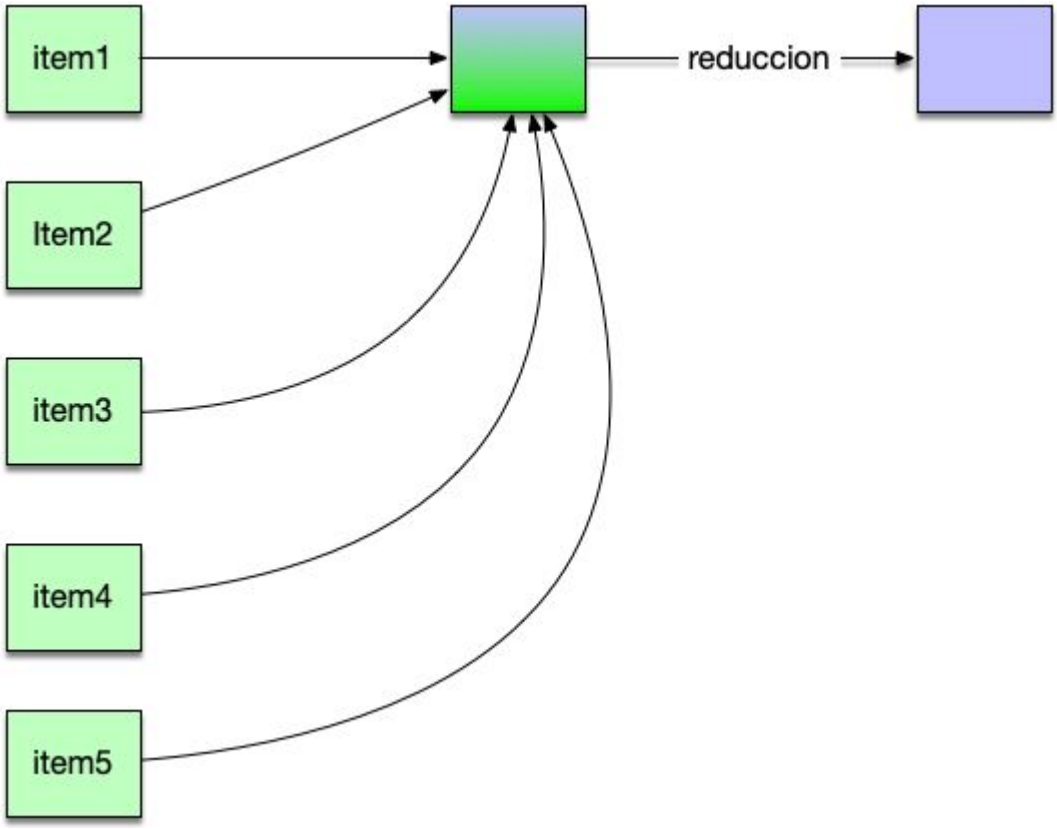
600

Java Stream Reduce

Gracias a la programación funcional podemos realizar estas operaciones de una forma muy diferente nos podemos apoyar en el método reduce que recibe 2 parámetros un acumulador como primero y un elemento como segundo .



De esta forma realiza una funcionalidad de “reducción” convirtiendo una lista de elementos en un único resultado.



Vamos a verlo en código :

```
package com.arquitecturajava;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import java.util.stream.Stream;

public class Principal3 {

  public static void main(String[] args) {
    List<Integer> gastos= new ArrayList<Integer>();
    gastos.add(100);
    gastos.add(200);
```

```
    gastos.add(300);
    gastos.stream().reduce((acumulador,numero)-> {
      return acumulador+numero;
    }).ifPresent(System.out::println);

}
```

Simplificación de sintaxis

Esto nos permite realizar la misma operación y obtener el mismo resultado pero realizando un ejemplo de programación funcional. Todavía lo podemos simplificar más ya que podríamos delegar en la clase Integer y en su reference method de sum que realiza la misma operación.

```
gastos.stream().reduce(Integer::sum).ifPresent(System.out::println);
```

De igual manera que usamos el método de reducción para sumar números podemos usarlo también por ejemplo para combinar cadenas.

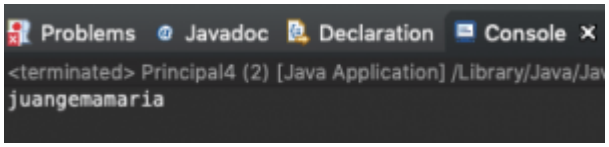
```
package com.arquitecturajava;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import java.util.stream.Stream;

public class Principal4 {

  public static void main(String[] args) {
    List<String> nombres= new ArrayList<String>();
    nombres.add("juan");
    nombres.add("gema");
```

```
    nombres.add("maria");
nombres.stream().reduce(String::concat).ifPresent(System.out::println)
;
  }
}
```



El uso de Java Stream Reduce nos puede ayudar en muchas situaciones a eliminar bucles y abordar situaciones de programación funcional que de otra forma serían complejas.

Otros artículos relacionados

- JavaScript Reduce function y eliminación de bucles
- Programación Funcional, Java 8 Streams
- JavaScript reduce y su flexibilidad
- Java Lambda reduce y wrappers