El uso de Java Stream for loop nos puede parecer un poco extraño en un principio ya que estamos acostumbrados a utilizar bucles for y nos parece lo más natural y más sencillo del mundo.

```
package com.arquitecturajava;

public class Principal {

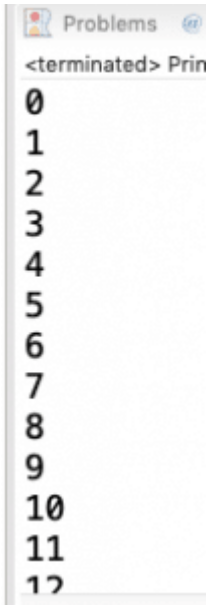
  public static void main(String[] args) {

    for (int i=0;i<100;i++) {
      System.out.println(i);
    }

  }

}
```

El resultado es :



Esta misma operación se puede realizar con Java Streams.

```
package com.arquitecturajava;

import java.util.stream.IntStream;

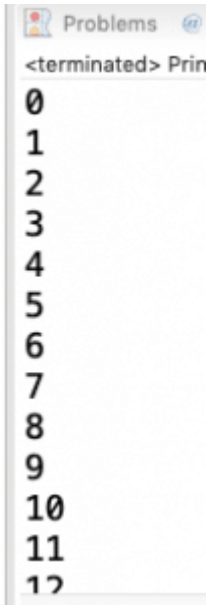
public class Principal2 {

  public static void main(String[] args) {
    IntStream miStream= IntStream.range(0, 100);
    miStream.forEach(System.out::println);

  }

}
```

El resultado es el mismo



Java Stream for loop

El código es algo más compacto y hace uso de un reference method de Java 8 . Aún así lo podemos simplificar un poquito más y dejarlo en :

```
package com.arquitecturajava;

import java.util.stream.IntStream;

public class Principal3 {

  public static void main(String[] args) {
    IntStream.range(0, 100).forEach(System.out::println);

  }

}
```

Lo cual es mucho más cómodo . Esta claro que esto no nos terminará de convencer a la hora de hacer un mayor uso de la programación funcional. Ahora bien sus ventajas en muchos casos son más que evidentes. Imaginemonos que tenemos que sumar los cuadrados de los primeros 100 números enteros. Se trata de una operación muy sencilla que podemos realizar con un bucle for :

```
package com.arquitecturajava;

public class Principal4 {

  public static void main(String[] args) {
    int total=0;
    for (int i=0;i<100;i++) {
      total+=Math.pow(i, 2);
```

```
    }
    System.out.println(total);

  }
```

Este código nos imprime el total por la consola :

328350.0

Podemos hacerlo de igual forma utilizando Streams y el código quedará un poco más claro y organizado apoyándose en programación funcional.

```
package com.arquitecturajava;

import java.util.stream.IntStream;

public class Principal5 {

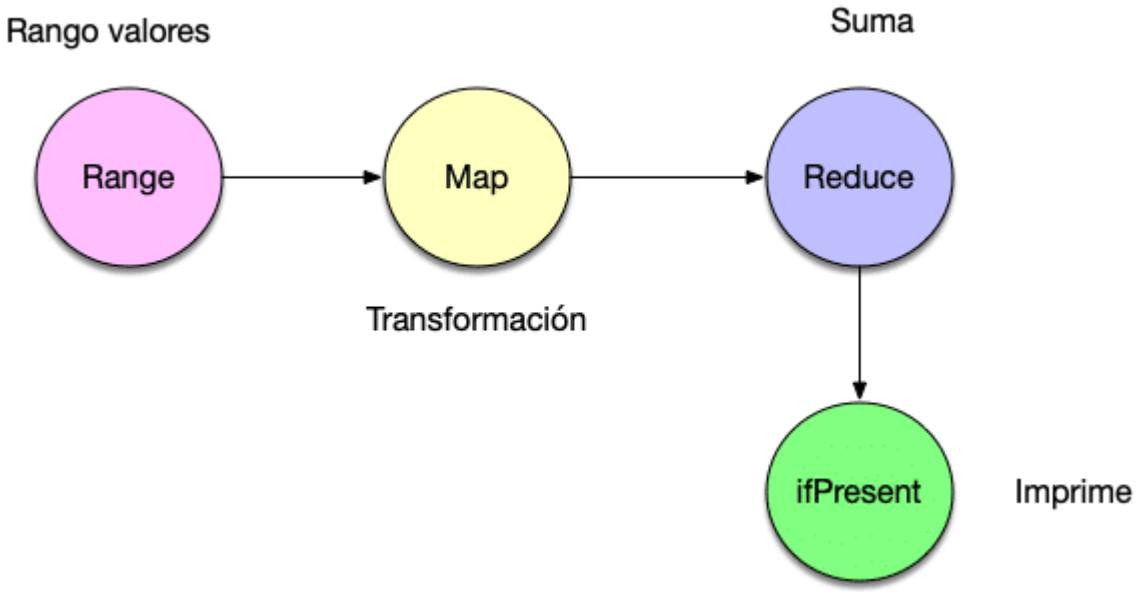
  public static void main(String[] args) {

    IntStream
    .range(0, 100)
    .map(x -> x*x)
    .reduce(Integer::sum)
    .ifPresent(System.out::println);

  }

}
```

El resultado es el mismo



Acabamos de generar un Java Stream For Loop :

328350.0

En este caso hemos usado:

1. range : Para delimitar los topes del bucle
2. map : Para elevar el numero al cuadrado
3. reduce: Para sumar todos los elementos usando un método de referencia. Podríamos haber usado sum directamente
4. ifPresent: Para imprimirlo en la consola si el resultado es valido

Otros artículos relacionados:

1. Java Stream File y manejo de ficheros
2. Java Stream map y estadísticas
3. Java Stream Reduce , eliminando bucles
4. Java Stream Filter
5. Introducción a Streams