

A veces usar Java Stream peek , es clave para entender cual es el funcionamiento de un flujo de Stream a nivel de Java 8 . Recordemos que los Streams nos permiten realizar operaciones complejas sobre un conjunto de datos . En ocasiones si el flujo de operaciones es elaborado cuesta entender realmente lo que esta ocurriendo. Vamos a construir un ejemplo basado en un array de cadenas y su manejo con streams, veamos el código:

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

public class Principal {

    public static void main(String[] args) {

        List<String> lista= Arrays.asList("hola","que" ,"tal",
"estas","tu");

        lista.stream()
            .filter((cadena)->cadena.length(>3)
            .map((cadena)->cadena.toUpperCase()).
            .forEach(System.out::println);

    }

}
```

En este caso realizamos una operación de filtrado , y una transformación de strings para luego finalmente imprimir el resultado en la consola.



Nos hemos quedado con las cadenas que tienen una longitud mayor de 3 caracteres y las hemos pasado a mayúsculas. El ejemplo es muy sencillo , y no hay mucho que objetar.

# Utilizando Java Stream peek

Sin embargo en otras situaciones nos puede generar dudas el tipo de operación que se ha ejecutado. ¿Qué podemos hacer en esos casos? . Podemos utilizar java Stream peek y realizar un logging de cada paso que realizamos con el flujo de Stream ,vamos a implementarlo:

```
package com.arquitecturajava.ejemplo003;

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",
"estas","tu");
        lista.stream()
            .peek((cadena)-> {
                System.out.println("***inicio***");
                System.out.println(cadena);
                System.out.println("****fin inicio****");

            }).filter((cadena)->cadena.length(>3)
            .peek((cadena)-> {
                System.out.println("-----filtro-----");
                System.out.println(cadena);
                System.out.println("-----fin filtro-----");

            })
    }
```

```

        .map((cadena) -> cadena.toUpperCase()).
        peek((cadena) -> {
            System.out.println(">>>>>mayusculas>>>>>");
            System.out.println(cadena);
            System.out.println(">>>>>fin
mayusculas>>>>>");

        }).forEach(System.out::println);

    }

}

```

El resultado de usar el método peek es bastante aclaratorio:

```

***inicio***
hola
****fin inicio****
-----filtro-----
hola
-----fin filtro-----
>>>>>mayusculas>>>>>
HOLA
>>>>>fin mayusculas>>>>>
HOLA
***inicio***
que
****fin inicio****
***inicio***
tal
****fin inicio****
***inicio***

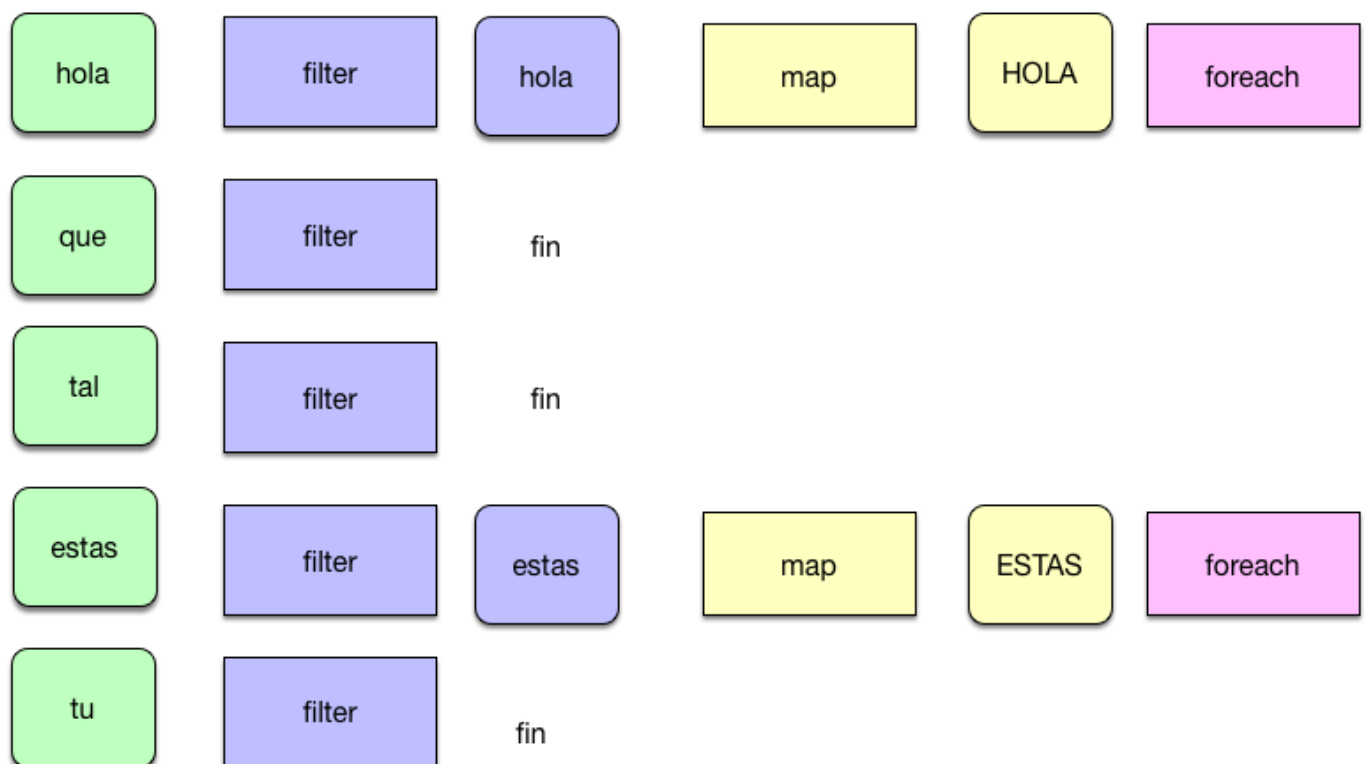
```

```

estas
****fin inicio****
-----filtro-----
estas
-----fin filtro-----
>>>>>mayusculas>>>>>
ESTAS
>>>>>>fin mayusculas>>>>>
ESTAS
***inicio****
tu
****fin inicio****

```

Es aquí donde podemos ver con claridad como funciona un stream, cada elemento realiza las operaciones oportunas de forma independiente:



java stream peek

El uso de Java Stream peek nos ayudará siempre a clarificar el manejo de los Streams, algo que necesitaremos en muchas situaciones.

## Otros artículos relacionados:

1. [Java 8 FlatMap y Streams](#)
2. [El concepto de Java infinite Stream](#)
3. [Programación Funcional, Java 8 Streams](#)
4. [Oracle Java Streams](#)