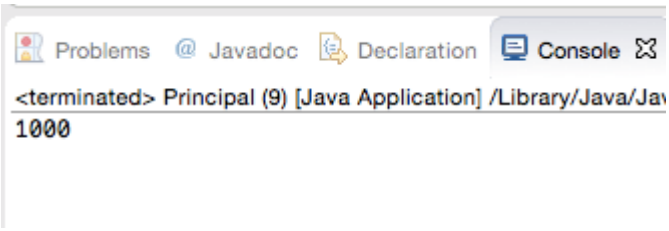
En muchas ocasiones me realizan preguntas sobre Java 8 Stream ya que a veces es difícil entender como funcionan y que relación tienen con la gestión de listas habituales. Vamos a partir de un ejemplo para explicar cual es su funcionamiento.

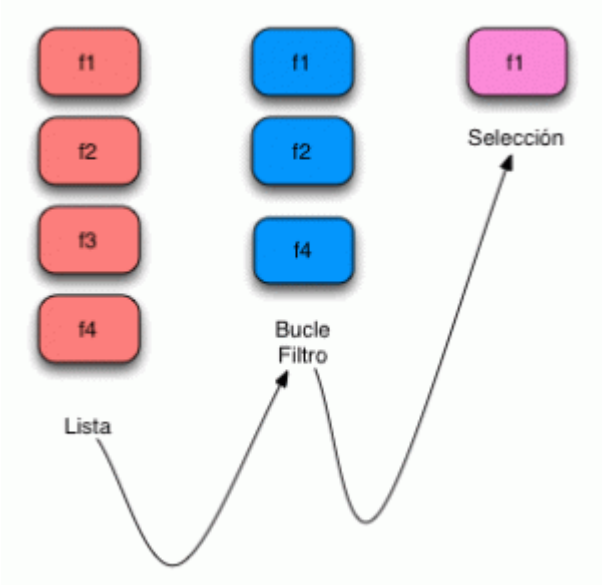
```
package com.arquitecturajava;

import java.util.ArrayList;
import java.util.List;

public class Principal {

 public static void main(String[] args) {


 Factura f= new Factura("ordenador",1000);
 Factura f2= new Factura("movil",300);
 Factura f3= new Factura("impresora",200);
 Factura f4= new Factura("imac",1500);

 List<Factura> lista= new ArrayList<Factura>();

 lista.add(f);
 lista.add(f2);
 lista.add(f3);
 lista.add(f4);

 Factura facturaFiltro=
 lista.stream()
```

```
 .filter(elemento->elemento.getImporte()>300).findFirst().get();

 System.out.println(facturaFiltro.getImporte());


 }

}
```

En este caso utilizamos la función filter y seleccionamos todas las facturas que tengan más de 300 euros. Una vez hecho nos quedamos con la primera e imprimimos por pantalla su valor.



La mayoría de la gente entendería que el proceso es el siguiente:



Java 8 Stream flujo de trabajo

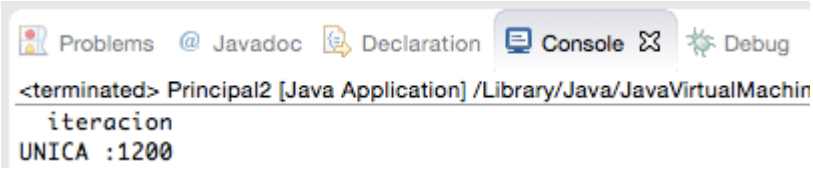
Aunque este parece el comportamiento correcto, realmente no lo es y para comprobarlo basta con cambiar el código y añadir nosotros nuestro propio predicado con la condición:

```
package com.arquitecturajava;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Predicate;

public class Principal2 {
```

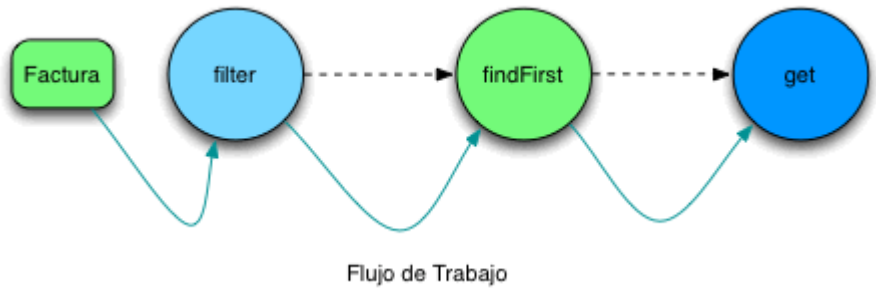
```
public static void main(String[] args) {


Factura f= new Factura("ordenador",1200);
Factura f2= new Factura("movil",300);
Factura f3= new Factura("impresora",200);
Factura f4= new Factura("imac",1500);
List<Factura> lista= new ArrayList<Factura>();

lista.add(f);
lista.add(f2);
lista.add(f3);
lista.add(f4);

Predicate<Factura> predicado= new Predicate<Factura>() {

@Override
public boolean test(Factura f) {

System.out.println(" iteracion ");
return f.getImporte()>200;
}


};

Factura facturaFiltro=
lista.stream()
.filter(predicado).findFirst().get();

System.out.println("UNICA :"+facturaFiltro.getImporte());
```

```
 }

}
```

En este caso el predicado ejecuta la misma condición pero ademas imprime la iteración por pantalla:



Sorprendentemente sólo se imprimirá una única iteración cuando realmente esperábamos 4 una por cada factura cuando el filtro las recorre. Esta es parte difícil de entender del concepto de Stream. Los Streams diseñan un flujo de trabajo que se ejecuta de forma unitaria item a item.



Flujo de Trabajo

En este caso es evidente que el flujo de trabajo es: busca el primer elemento cuyo importe supere los 300 euros y retórnalo. Una vez encontrado no es necesario recorrer el resto de la lista , de hecho es un error hacerlo . Por lo tanto los streams simplemente ejecutan el flujo de trabajo para el primer elemento y como este cumple los requisitos el flujo termina , así de sencillo. Esto nos permitirá una mejora en el rendimiento ya que no es necesario recorrer

la lista completa. Si quisieramos recorrer la lista entera deberíamos utilizar forEach.

```
 lista.stream()
.filter(predicado).forEach(factura->System.out.println("VARIAS:"+factu
ra.getImporte()));
```

Otros artículos relacionados: Programación funcional y Streams , Java 8 default Methods , Java 8 Lambda