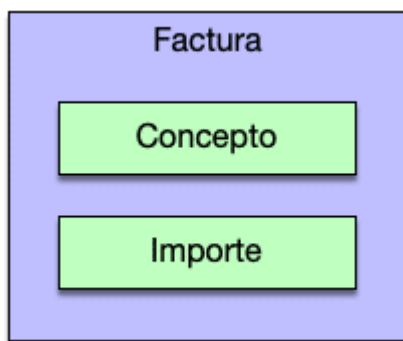


## Tabla de Contenidos



- [Java Stream y flujos de trabajo](#)
- [Java Stream y su comportamiento](#)
- [Streams y Rendimiento](#)
- [Otros artículos relacionados:](#)

En muchas ocasiones me realizan preguntas sobre el concepto de Java Stream ya que a veces es difícil entender su funcionamiento y la relación que tienen con un ArrayList clásico. Vamos a verlo a detalle a partir de del concepto de Factura que contiene concepto e importe :



Veamos un ejemplo en código del uso de un Stream:

**CURSO JAVA 8  
GRATIS  
APUNTATE!!**

```
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());

    }

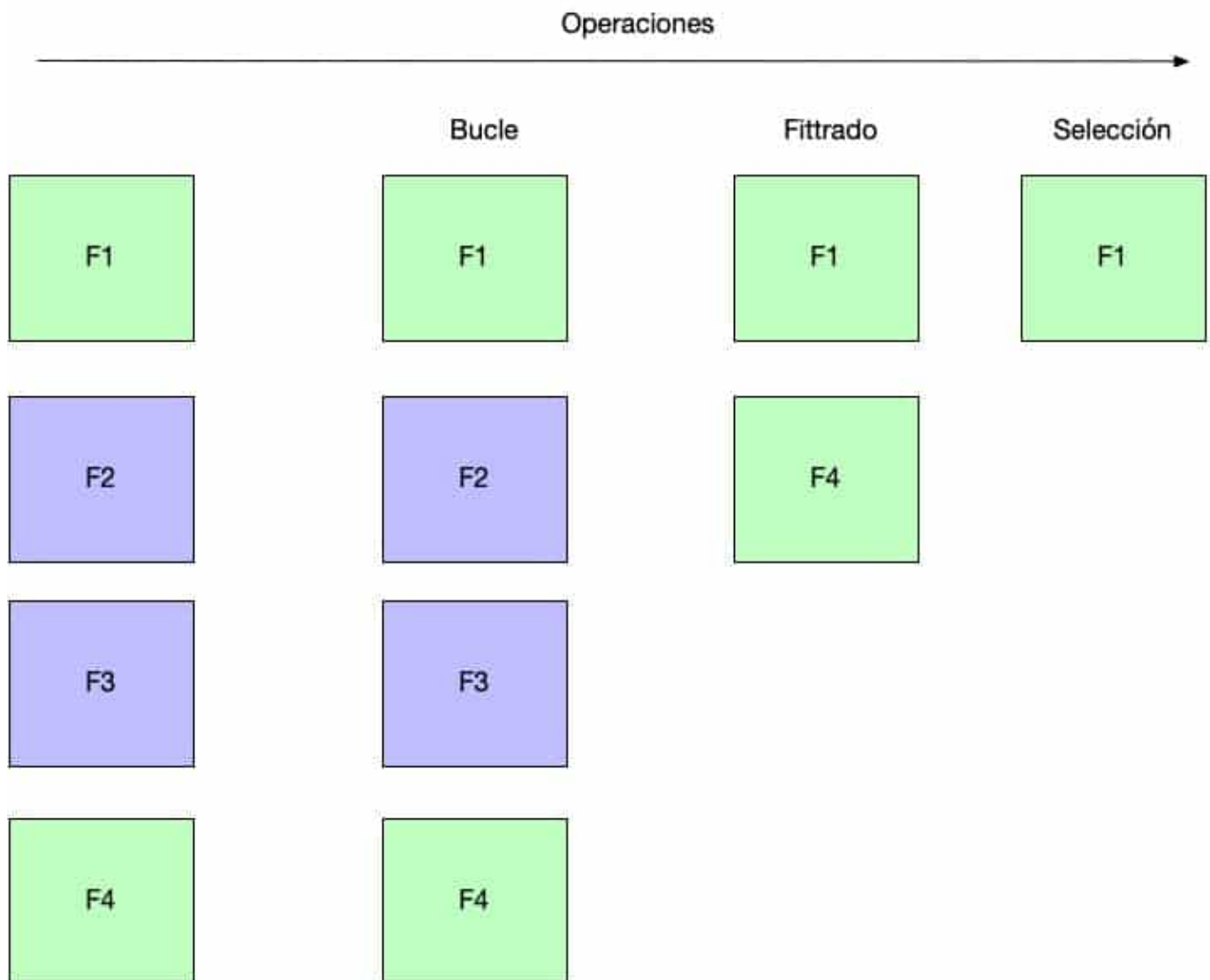
}
```

El ejemplo es relativamente sencillo . Creamos un Stream a partir de una lista de Facturas y usamos la operación de filter dentro del Java Stream para quedarnos con aquellas facturas

que cumplen la condición en este caso solo una de ellas cumple :



La mayoría de los desarrolladores pensarán que recorreremos la lista completa de facturas que luego se realiza la operación de filtrado para finalmente obtener un único elemento.



## Java Stream y flujos 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 .

**TODOS LOS CURSOS  
PROFESIONALES  
25\$/MES  
APUNTATE!!**

```
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()>300;
}

};

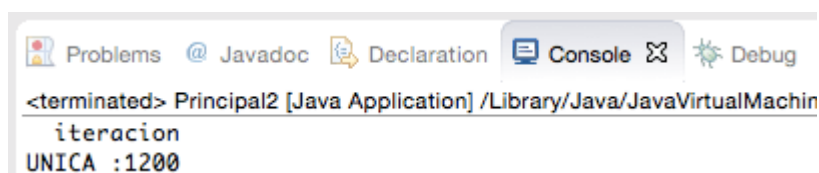
Factura facturaFiltro=
lista.stream()
.filter(predicado).findFirst().get();

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

}

}
```

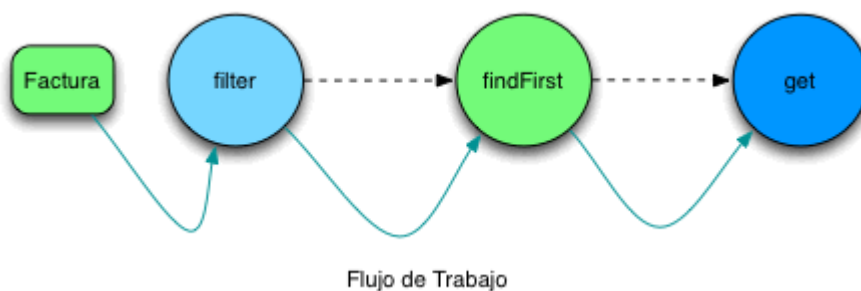
En este caso el predicado ejecuta la misma condición pero ademas imprime la iteración por pantalla .Deberíamos recibir varias iteraciones ya que tenemos 4 elementos . Sin embargo el resultado es muy muy curioso ya que solo se imprime una iteración por la consola.



¿Qué paso con el resto de iteraciones? ¿Cómo funciona un Java Stream?

## Java Stream y su comportamiento

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.



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.

## Streams y Rendimiento

Esto nos permitirá una mejora en el rendimiento ya que no es necesario recorrer la lista completa.

**CURSO SPRING BOOT  
GRATIS  
APUNTATE!!**

## Otros artículos relacionados:

- [Programación funcional y Streams](#)
- [Java 8 default Methods](#)
- [Java 8 Lambda](#)
- [Curso Java 8 Stream](#)