Las opciones de Java Stream Filter son muy amplias pero para poder trabajar con ellas de forma cómoda hay que entender el funcionamiento del interface Predicate a detalle. Vamos a construir un ejemplo con una lista de libros que nos permite entender mejor como funcionan los filtros. Para ello vamos a partir de los siguientes objetos.

```
package com.arquitecturajava;

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

public class Principal {

    public static void main(String[] args) {

        Libro l = new Libro("El señor de los anillos", "fantasia",
1100);
        Libro l2 = new Libro("El Juego de Ender", "ciencia ficcion",
500);
        Libro l3 = new Libro("La fundacion", "ciencia ficcion", 500);
        Libro l4 = new Libro("Los pilares de la tierra", "historica",
1200);

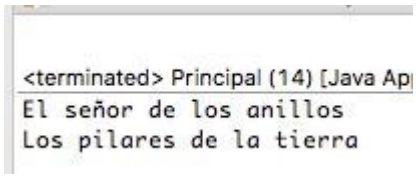
        List < Libro > lista = Arrays.asList(l, l2, l3, l4);

        lista.stream()
            .filter(libro - > libro.getPaginas() > 1000)
            .map(libro - > libro.getTitulo())
            .forEach(System.out::println);

    }
```

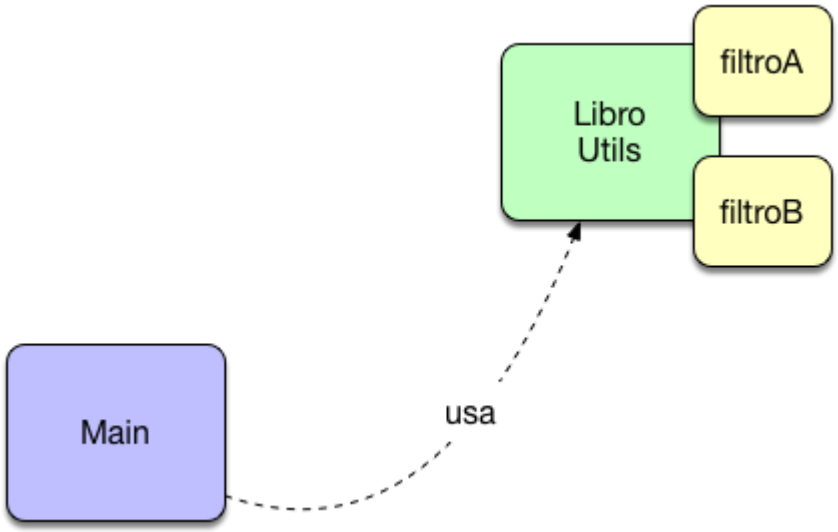
```
}
```

Tenemos cuatro libros y acabamos de usar un filtro sencillo que nos selecciona los libros que pasan de 1000 páginas utilizando un Predicate con una expresión lambda. El resultado se imprime en la consola.



Java Stream Filter

El filtro funciona de forma correcta , pero no siempre queremos filtran usando una expresión lambda de una forma directa ya que no tiene una gran capacidad de reutilización. En muchos casos suele ser mejor tener alguna clase de apoyo que defina una serie de expresiones lambda y predicados que nos ayuden y se reutilicen.



Vamos a ver un par de ejemplos.

```
package com.arquitecturajava;

import java.util.function.Predicate;
```

```
public class LibroUtils {

	public static Predicate<Libro> filtroCategoria(String
categoria) {

		return (Libro l) -> {
			return l.getCategoria().equals(categoria);
		};
	}

}
```

De esta forma podemos podemos reutilizar la expresión , y filtrar por la categoría que deseemos:

```
lista
.stream()
.filter(LibroUtils.filtroCategoria("ciencia ficcion"))
.map(libro - > libro.getTitulo())
.forEach(System.out::println);
```

Sin embargo lo más habitual en muchos casos es generar expresiones funcionales complejas y luego ligarlas a través de métodos de referencia . Por ejemplo algo como la siguiente expresion.

```
package com.arquitecturajava;

import java.util.function.Predicate;

public class LibroUtils {
```

```
    public static Predicate < Libro > filtroCategoria(String
categoria) {

        return (Libro l) - > {
            return l.getCategoria().equals(categoria);
        };
    }

    public static boolean buenosLibros(Libro libro) {

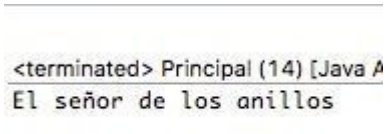
        Predicate < Libro > p1 = (Libro l) - >
l.getCategoria().equals("ciencia ficcion");
        Predicate < Libro > p2 = (Libro l) - >
l.getCategoria().equals("fantasia");
        Predicate < Libro > p3 = (Libro l) - > l.getPaginas() > 1000;
        Predicate < Libro > ptotal = p1.or(p2).and(p3);

        return ptotal.test(libro);
    }
}
```

De esta forma nosotros podemos invocar este tipo de filtro y reutilizarlo.

```
lista
    .stream()
    .filter(LibroUtils::buenosLibros)
    .map(Libro::getTitulo)
    .forEach(System.out::println);
```

El resultado es:



Acabamos de usar Java Stream Filter y sus opciones.

Otros artículos relacionados:

1. Java Predicate Interface y sus métodos
2. Java Stream forEach y colecciones
3. Java 8 Lambda Expressions (I)
4. Oracle Java Lambda