El concepto de Java Predicate es muy común en el mundo de la programación funcional. Un predicado es un interface funcional que sirve para filtrar una coleccion de elementos que en muchos casos se encuentra a nivel de un flujo de Stream

Vamos a ver un ejemplo muy sencillo con una lista de Facturas.

```
package com.arquitecturajava.ejemplo1;

import java.util.Objects;

public class Factura  {

    private int numero;
    private String concepto;
    private double importe;



    public int getNumero() {
        return numero;
    }

    public void setNumero(int numero) {
        this.numero = numero;
    }

    public String getConcepto() {
        return concepto;
    }

    public void setConcepto(String concepto) {
        this.concepto = concepto;
    }

    public double getImporte() {
```

```
    return importe;
}

public void setImporte(double importe) {
    this.importe = importe;
}

public Factura(int numero, String concepto, double importe) {
    this.numero = numero;
    this.concepto = concepto;
    this.importe = importe;
}



@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof Factura)) return false;
    Factura factura = (Factura) o;
    return numero == factura.numero;
}

@Override
public int hashCode() {
    return Objects.hash(numero);
}



@Override
```

```
    public String toString() {
        return "Factura{" +
                "numero=" + numero +
                ", concepto='" + concepto + '\'' +
                ", importe=" + importe +
                '}';
    }
}
```

Esta Factura contiene la propiedad importe y podemos definir una lista de Facturas de la cual queremos realizar un filtrado por importe. En este caso sería tan sencillo como :

```
package com.arquitecturajava.ejemplo1;

import com.arquitecturajava.Factura;

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

public class Principal {

    public static void main (String[] args) {


        List<Factura> lista= new
ArrayList<com.arquitecturajava.Factura>();
        lista.add(new Factura(1, "ordenador", 200));
        lista.add(new Factura(2, "tablet", 300));
        lista.add(new Factura(3, "auricular", 50));
        lista.add(new Factura(4, "portatil", 500));
```

```
        lista.add(new Factura(5, "mi ordenador", 200));
        lista.add(new Factura(6, "mesa", 150));

        Predicate<Factura> p1 = (f) -> f.getImporte() >= 200;

        List<Factura>
nuevaLista=lista.stream().filter(p1).collect(Collectors.toList());
        nuevaLista.forEach((f)->System.out.println(f.getConcepto()));

    }
}
```

Usamos un Java Predicate y realizamos un filtrado por importe mayor o igual a 200 . El resultado se imprime por la consola en donde vemos que solo se incluyen las facturas que cumplen la condición.

```
ordenador
tablet
portatil
mi ordenador
```

Combinando Predicados

Los Predicados se pueden combinar y al combinarlos construiremos filtros más complejos .A continuación se muestra un ejemplo de clausula and.

```
  Predicate<Factura> p1 = (f) -> f.getImporte() >= 200;
        Predicate<Factura> p2 = (f) -> f.getImporte() <= 300;

        Predicate<Factura> pfinal= p1.and(p2);
        List<Factura>
```

```
nuevaLista=lista.stream().filter(pfinal).collect(Collectors.toList());
        nuevaLista.forEach((f)->System.out.println(f.getConcepto()));
```

En ese caso el filtrado será más intenso y que hemos añadido una condición adicional:

```
ordenador
tablet
mi ordenador
```

De la misma forma que tenemos and y or también disponemos de negate que se puede combinar con los anteriores.

```
        Predicate<Factura> p1 = (f) -> f.getImporte() >= 200;
        Predicate<Factura> pfinal= p1.negate();
        List<Factura>
nuevaLista=lista.stream().filter(pfinal).collect(Collectors.toList());
        nuevaLista.forEach((f)->System.out.println(f.getConcepto()));
```

El resultado incluye ahora únicamente al auricular y la mesa

```
auricular
mesa
```

Java Predicate Complejidad y Flexibilidad (Premium)

[ihc-hide-content ihc_mb_type=”show” ihc_mb_who=”4″ ihc_mb_template=”1″]

Hay situaciones en las que no nos es suficiente con diseñar un conjunto de 2 predicados y combinarlos de formas diferentes sino que necesitamos una mayor flexibilidad. ¿Cómo podemos abordar esta situación?. Para ello deberemos abordar una operación de reducción que soporte una lista de predicados cualquiera.

```
package com.arquitecturajava.ejemplo1;

import com.arquitecturajava.Factura;

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

public class Principal3 {

    public static void main (String[] args) {


        List<Factura> lista= new ArrayList<Factura>();
        lista.add(new Factura(1, "ordenador", 200));
        lista.add(new Factura(2, "tablet", 300));
        lista.add(new Factura(3, "auricular", 50));
        lista.add(new Factura(4, "portatil", 500));
        lista.add(new Factura(5, "mi ordenador", 200));
        lista.add(new Factura(6, "mesa", 150));

        List<String> conceptos= new ArrayList<String>();
        conceptos.add("ordenador");
        conceptos.add("tablet");
        conceptos.add("mesa");

        List<Predicate<Factura>> predicados = new
ArrayList<Predicate<Factura>>();
        for (String concepto: conceptos) {
            predicados.add(f->f.getConcepto().contains(concepto));
```

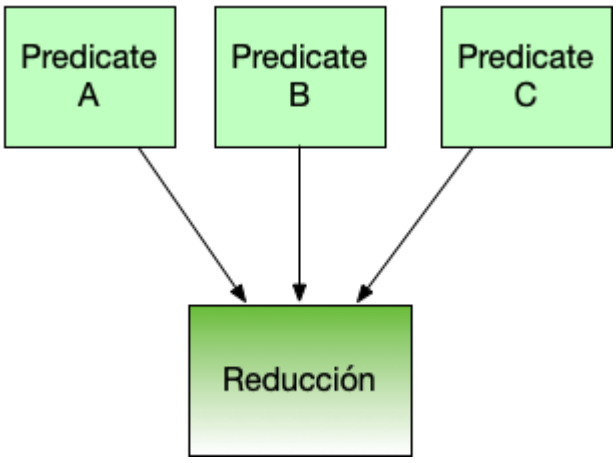
```
        }

        List<Factura> listaFinal=
lista.stream().filter(predicados.stream().reduce(x->false,Predicate::o
r)).collect(Collectors.toList());

listaFinal.stream().forEach((f)->System.out.println(f.getConcepto()));
    }
}
```

De esta manera conseguiremos obtener una lista con los ordenadores , tablets y mesas de forma flexible:

```
ordenador
tablet
mi ordenador
mesa
```



Otros artículos relacionados

- Java Stream if else y Optionals
- Java Array to Stream y sus opciones
- Java 8 Functional Interfaces y sus tipos
- ¿Qué es un Java Lambda?