

Tabla de Contenidos



- [Métodos Java Predicate Interface](#)

- [Conclusiones](#)

- [Otros artículos relacionados:](#)

¿Cuales son los métodos más utilizados de un Java Predicate Interface? . Estamos muy acostumbrados a usar un predicado concreto para filtrar Streams . Muchas veces se nos olvida que el interface soporta varios métodos que aportan flexibilidad. Vamos a echarlos un vistazo . Para ello partiremos de una colección de Personas que queremos recorrer utilizando Streams.

```
package com.arquitecturajava;
```

```
public class Persona {
```

```
    private String nombre;
```

```
    private String apellidos;
```

```
    private int edad;
```

```
    public String getNombre() {
```

```
        return nombre;
```

```
    }
```

```
    public void setNombre(String nombre) {
```

```
        this.nombre = nombre;
```

```
    }
```

```
    public String getApellidos() {
```

```
        return apellidos;
```

```
    }
```

```
    public void setApellidos(String apellidos) {
```

```
        this.apellidos = apellidos;
```

```
    }
```

```

    public int getEdad() {
        return edad;
    }
    public void setEdad(int edad) {
        this.edad = edad;
    }

    public Persona(String nombre, String apellidos, int edad) {
        super();
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.edad = edad;
    }
}

package com.arquitecturajava;

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

public class Principal {

    public static void main(String[] args) {

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

        Persona p1= new Persona ("pepe","perez",20);
        Persona p2= new Persona ("ana","perez",30);
        Persona p3= new Persona ("gema","sanchez",40);
    }
}

```

```

        Persona p4= new Persona ("pedro","gomez",70);

        lista.add(p1);
        lista.add(p2);
        lista.add(p3);
        lista.add(p4);

    }

}

```

Métodos Java Predicate Interface

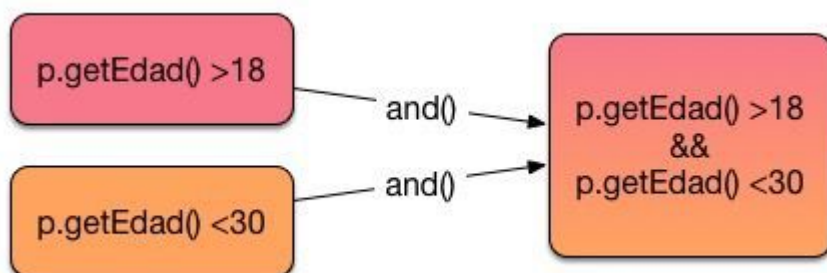
Uno de los métodos más utilizados del interface es `and()` que permite unir dos predicados y que la condición se cumpla si ambos predicados son ciertos.

```

Predicate < Persona > predicado1 = p -> p.getEdad() > 18;
Predicate < Persona > predicado2 = p -> p.getEdad() > 30;
Predicate < Persona > predicado3 = predicado1.and(predicado2);

```

En este caso estamos fusionando el predicado1 y el predicado2:



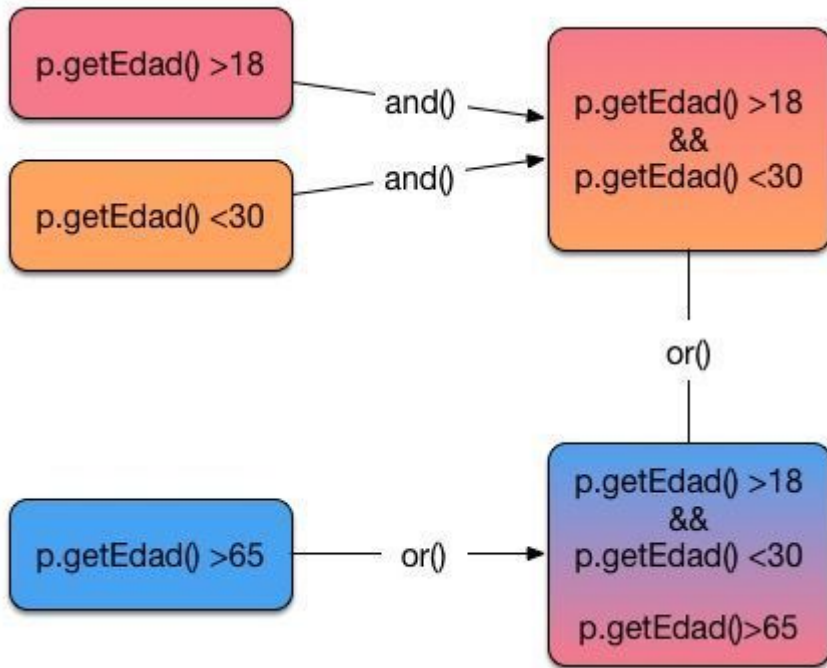
De igual forma podemos usar la clausula `or()` que fusiona los predicados pero hace que se cumpla la condición si uno de los dos es cierto.

```

Predicate < Persona > predicado4 = p -> p.getEdad() > 65;

```

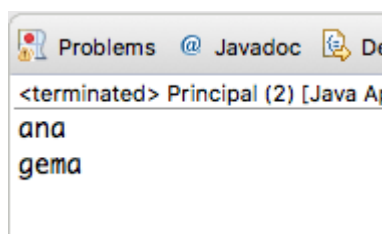
```
Predicate < Persona > predicado5 = predicado4.or(predicado3);
```



Por último podemos modificar el predicado y añadir una negación de tal forma que se cumpla lo contrario:

```
Predicate < Persona > predicado4 = p -> p.getEdad() > 65;  
Predicate < Persona > predicado5 = predicado4.or(predicado3).negate();
```

Es momento de recorrer la lista de Personas e imprimir por pantalla los nombres de las que cumplen las condiciones:



Conclusiones

Predicate es uno de los interfaces que más vamos a utilizar manejando Streams conocerlo a detalle es importante. El soporte de sintaxis fluida los hace muy cómodos.

Otros artículos relacionados:

- [¿Qué es un Java Lambda?](#)
- [Utilizando Java 8 Predicate](#)
- [Java Predicate Not y como usarlo](#)
- [Java Pattern Predicate y filtrados](#)
- [Java 8](#)