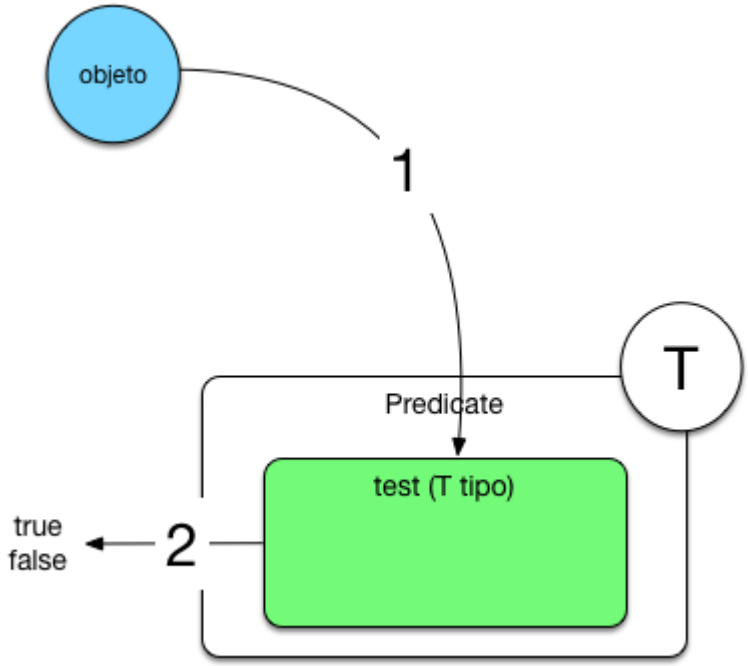
Crear un Java Predicate , es una de las operaciones que más realizaremos cuando trabajemos con expresiones Lambda y Streams en Java 8. ¿Qué es un Predicado? . Un Predicado es un interface funcional que define una condición que un objeto determinado debe cumplir . ¿Por ejemplo es una Persona mayor de 20 años?.



El Predicado dispone de un único método denominado test que recibe el objeto y comprueba si cumple la condición. Vamos a construir en ejemplo apoyándonos en la clase Persona.

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

    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;
    }
}
```

Lista de Personas

Vamos a construir una lista de Personas para más adelante aplicar un Predicado sobre ella.

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

public class Principal {

    public static void main(String[] args) {

        List<Persona> lista = new ArrayList<>();
        Persona p1 = new Persona("pepe", "perez", 20);
        Persona p2 = new Persona("angel", "sanchez", 30);
        Persona p3 = new Persona("pepe", "blanco", 40);
        lista.add(p1);
        lista.add(p2);
        lista.add(p3);

        // Resto del código...

    }
}
```

Finalmente convertiremos la lista en un Stream de datos y la recorreremos:

```
lista.stream().forEach(new Consumer<Persona>() {
    @Override
    public void accept(Persona p) {
        System.out.println(p.getNombre());
    }
});
```

El resultado será :

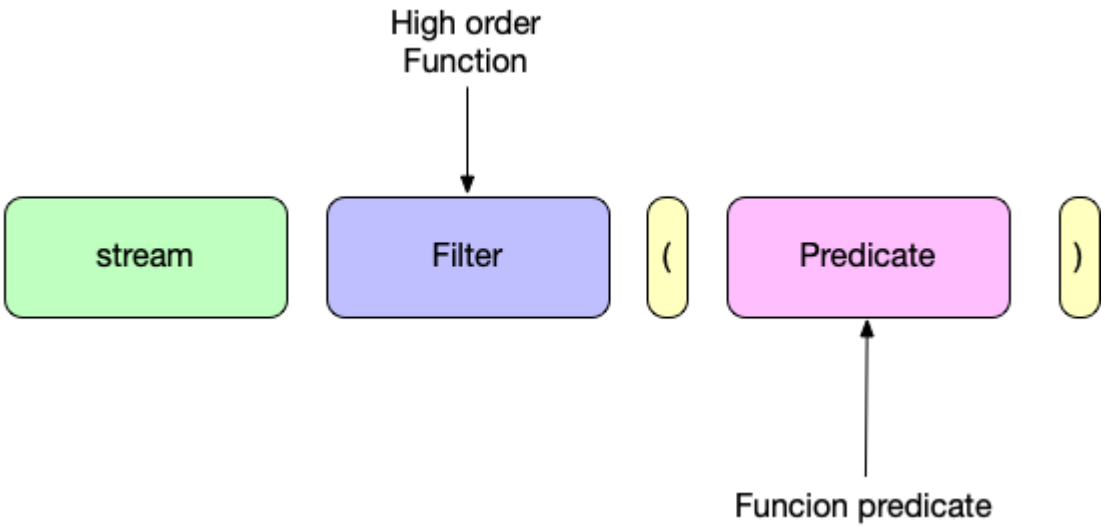
```
pepe
angel
pepe
```

Java Predicate al rescate

Hemos recorrido la lista , vamos a crear un Predicado que se encargue de definir una condición que permita filtrar la lista para nuestro Stream.

```
Predicate<Persona> predicadoNombre = new Predicate<Persona>() {
    @Override
    public boolean test(Persona p) {
        return p.getNombre().equals("pepe");
    }
};
```

En este caso acabamos de crear un Predicado que solo cumplirán las personas que se llamen “pepe”. Usamos la función filter a nivel de Streams esta función es una función de orden superior que recibe como parametro otra función.



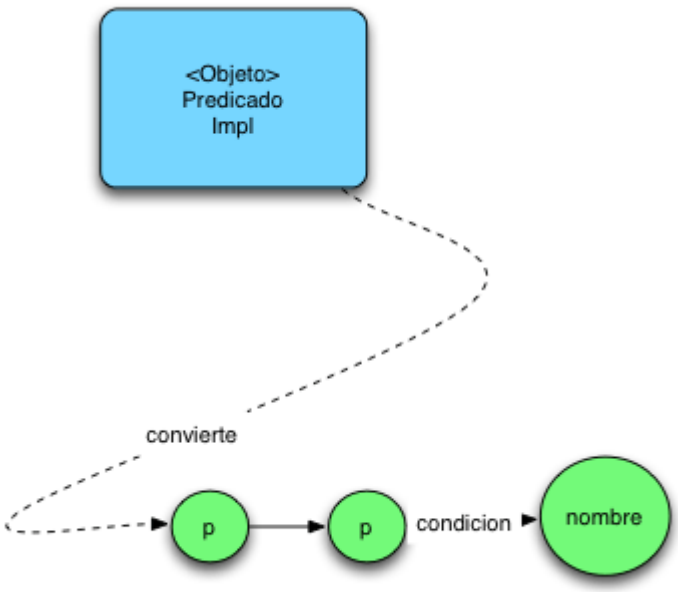
En este caso la función es un Predicado y obligar a que se filtre en base a la condición que el Predicado define.

```
lista.stream().filter(predicadoNombre).forEach(p ->
System.out.println(p.getApellidos()));
```

El resultado en pantalla será:

```
***************
perez
blanco
***************
```

El Stream se ha apoyado en el Predicado para realizar el filtrado y quedarnos con las Personas que se llaman “pepe” . Recordemos que los interfaces funcionales se pueden ver como expresiones Lambda y su gestión es más sencilla.



Es momento de convertir nuestro Java 8 Predicate a una expresión lambda para ganar claridad:

```
lista.stream()
    .filter(p -> p.getNombre().equals("pepe"))
    .forEach(p -> System.out.println(p.getApellidos()));
```

El resultado será idéntico:

```
**********
perez
blanco
**********
```

Acabamos de usar predicados apoyandonos en Java Lambdas de una forma natural.

Otros artículos relacionados:

- Introducción a Spring Data y JPA
- Java Predicate Interface y sus métodos
- Java Override y encapsulación
- Java Predicate Not y como usarlo