¿ Java Lambda vs reference method cual es mejor? . Cuando empezamos a trabajar con programación funcional rapidamente nos encontramos con situaciones en las que comparamos las expresiones lamdba con los métodos de referencia. ¿Cual es mejor mejor opción? . Vamos a construir un ejemplo que nos ayude a entenderlo de forma más sencilla . Para ello partiremos de la clase Persona:



Vamos a verlo en código:

```
package com.arquitecturajava.ejemplo1;

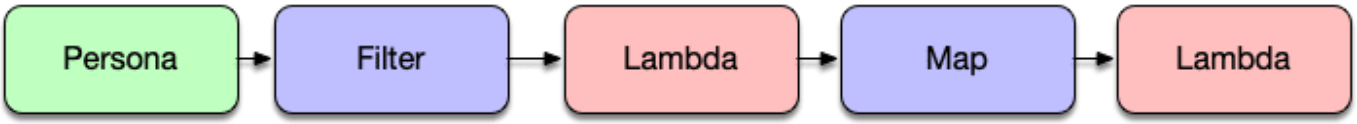
public class Persona {

	private String nombre;
	private int edad;
	private double salario;
	public double getSalario() {
		return salario;
	}
	public void setSalario(double salario) {
		this.salario = salario;
	}
	public String getNombre() {
```

```
                return nombre;
        }
        public void setNombre(String nombre) {
                this.nombre = nombre;
        }
        public int getEdad() {
                return edad;
        }
        public void setEdad(int edad) {
                this.edad = edad;
        }

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

Esta clase dispone de la propiedad edad y la propiedad salario . Nosotros queremos construir un programa que nos permita calcular el salario final de las Personas que ya están jubiladas.



Se trata de un código relativamente rápido de construir a nivel de expresiones lambda.

```
package com.arquitecturajava.ejemplo1;

import java.util.Arrays;
```

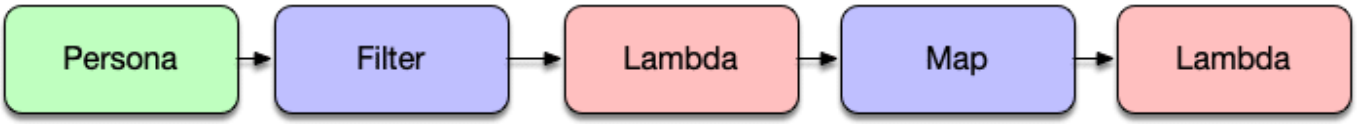
```
import java.util.List;
import java.util.stream.Collectors;

public class Principal {

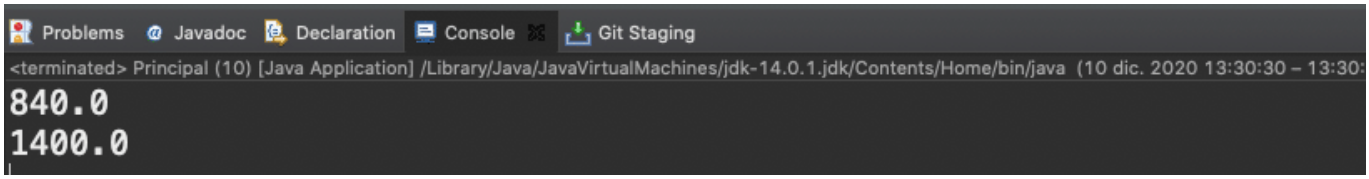
	public static void main(String[] args) {
		Persona p1= new Persona("juan",50,1000);
		Persona p2= new Persona("ana",70,1200);
		Persona p3= new Persona("miguel",50,900);
		Persona p4= new Persona("maria",80,2000);
		List<Persona> lista= Arrays.asList(p1,p2,p3,p4);
		List<Double> salariosJubilados=
				lista.stream()
				.filter(p->p.getEdad()>67)
				.map(p->p.getSalario()*0.70)
				.collect(Collectors.toList());
		salariosJubilados.forEach(System.out::println);
	}

}
```

En este caso filtramos la lista en un primer lugar para quedarnos con los jubilados para luego realizar una operación de transformación (map) que nos permita obtener el 70% del salario de la persona.

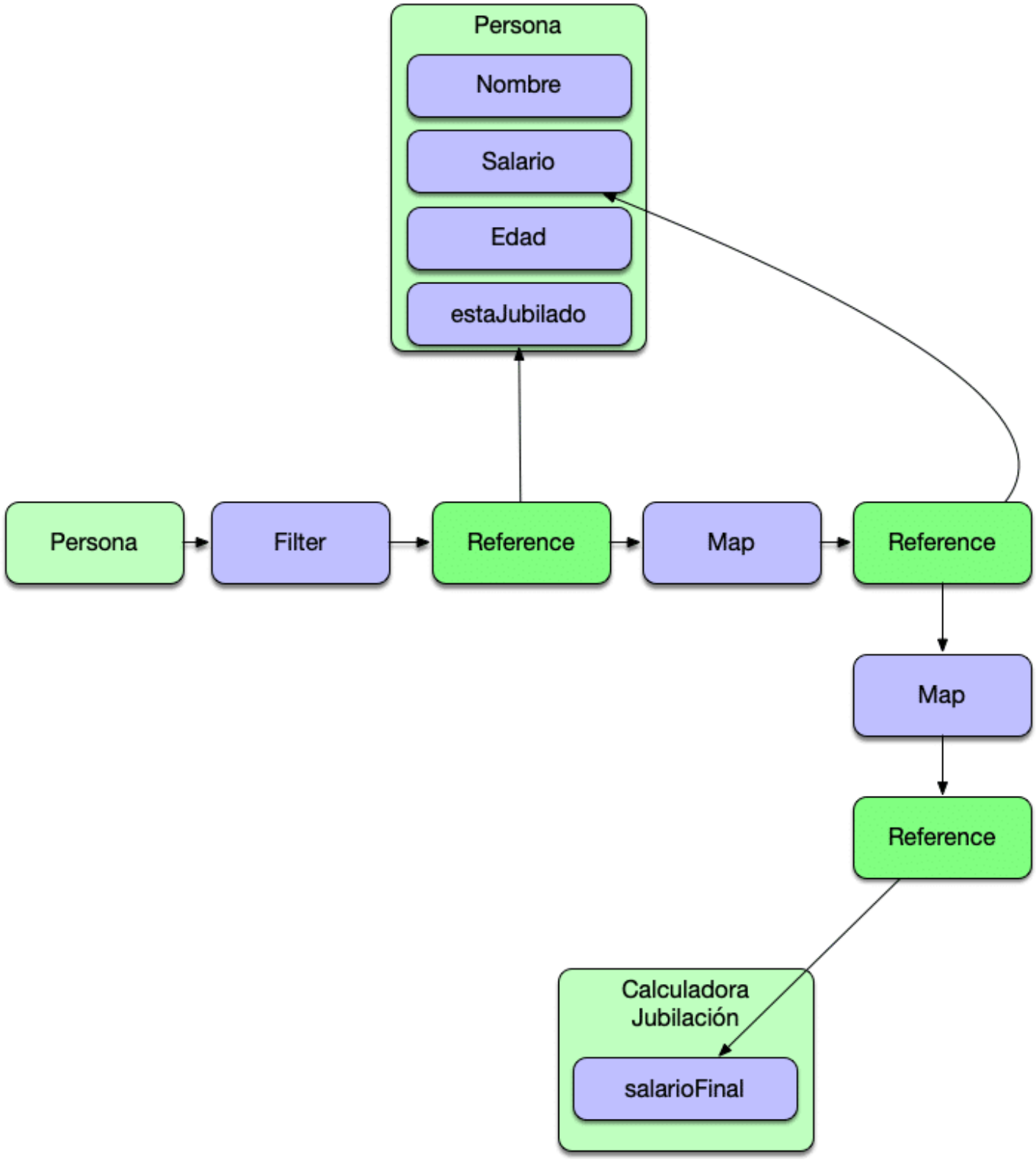


Con eso simplemente nos quedamos con una lista y la recorremos.



Java Lambda vs Reference Method

Hasta aquí todo parece perfecto , pero la realidad es que muchas de las operaciones que estamos usando son muy muy standard y se podrían reutilizar en otras áreas del programa estamos abriéndonos a la repetición de código en otras zonas. Por lo tanto puede ser interesante reestructurar el código para usar métodos de referencia.




```
package com.arquitecturajava.ejemplo2;
```

```
public class Persona {

	private String nombre;
	private int edad;
	private double salario;
	public double getSalario() {
		return salario;
	}
	public void setSalario(double salario) {
		this.salario = salario;
	}
	public String getNombre() {
		return nombre;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public int getEdad() {
		return edad;
	}
	public void setEdad(int edad) {
		this.edad = edad;
	}

	public Persona(String nombre, int edad, double salario) {
		super();
		this.nombre = nombre;
		this.edad = edad;
		this.salario = salario;
	}
	public boolean estaJubilado() {
```

```
                return this.getEdad()>67);
        }
}
```

La clase Persona contendrá el método que nos dice si una Persona esta jubilada y la clase CalculadoraJubilación nos indica que dinero debe recibir una vez jubilada.

```
package com.arquitecturajava.ejemplo2;

public class CalculadoraJubilacion {

        public static  double salarioFinal(double salario) {
                return salario*0.70;
        }
}
```

Una vez realizadas estas modificaciones podemos crear la nueva versión del programa.

```
package com.arquitecturajava.ejemplo2;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class Principal {

        public static void main(String[] args) {
                Persona p1= new Persona("juan",50,1000);
                Persona p2= new Persona("ana",70,1200);
                Persona p3= new Persona("miguel",50,900);
                Persona p4= new Persona("maria",80,2000);
                List<Persona> lista= Arrays.asList(p1,p2,p3,p4);
```

```
                List<Double> salariosJubilados=lista.stream()
                                    .filter(Persona::estaJubilado)
                                    .map(Persona::getSalario)
.map(CalculadoraJubilacion::salarioFinal)
                                    .collect(Collectors.toList());
                salariosJubilados.forEach(System.out::println);
        }

}
```

En ella nos apoyamos en el método estaJubilado, getSalario y salarioFinal para realizar la misma operación . El código queda algo más amplio pero hemos ubicado las responsabilidades en las clases correctas e incrementado el nivel de reutilización. La mayor parte de las veces a la hora de tomar la decisión entre Java Lambdas vs reference methods es mejor usar estos últimos.

Otros artículos relacionados

- Java Stream Sorted y Comparators
- Java Stream Reduce , eliminando bucles
- Java Collector Join Streams y Strings
- Curso Java 8 Stream y Lambdas