CURSO Java Herencia
GRATIS
APUNTATE!!

El uso del interface Java Comparator es muy común cuando trabajamos con Java . En muchas ocaciones tenemos que ordenar una lista de elementos y usamos el interface Comparator para hacerlo . Hasta la llegada de Java 8 comparar elementos siempre ha sido bastante engorroso . Vamos a ver un ejemplo clásico para luego evolucionarle con Java 8. Para ello partiremos de una lista de Personas.

```
package com.arquitecturajava.ejemplo1;

public class Persona {

	private String nombre;
	private String apellido1;
	private String apellido2;

	public String getNombre() {
		return nombre;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public String getApellido1() {
		return apellido1;
	}
	public void setApellido1(String apellido1) {
```

```
                this.apellido1 = apellido1;
        }
        public String getApellido2() {
                return apellido2;
        }
        public void setApellido2(String apellido2) {
                this.apellido2 = apellido2;
        }
        public Persona(String nombre, String apellido1, String
apellido2) {
                super();
                this.nombre = nombre;
                this.apellido1 = apellido1;
                this.apellido2 = apellido2;
        }
        @Override
        public String toString() {
                return "Persona [nombre=" + nombre + ", apellido1=" +
apellido1 + ", apellido2=" + apellido2 + "]";
        }

}
```

Java Comparator e Interfaces

Vamos a crearnos una lista de Personas que deseamos ordenar. Esta lista puede ser ordenada de diversas formas. Por ejemplo por nombre , apellido1 o apellido2.

```
package com.arquitecturajava.ejemplo1;

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

```
public class Principal2 {

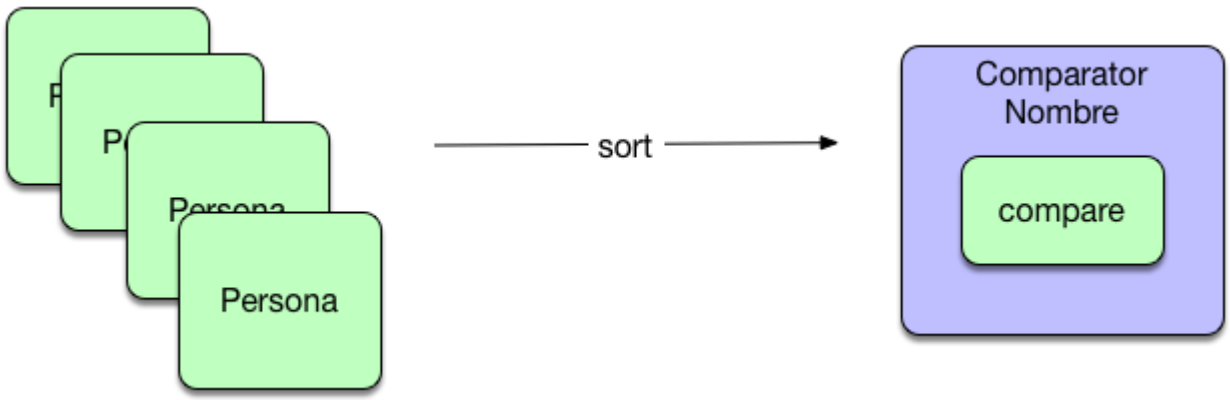
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Persona p1= new Persona("pedro","perez","gomez");
		Persona p2= new Persona("angel","alvarez","zamora");
		Persona p3= new Persona("ana","perez","jimenez");
		Persona p4= new Persona("ana","sainz","jimenez");
		Persona p5= new Persona("maria","alvarez","alvarez");

		List<Persona> lista=Arrays.asList(p1,p2,p3,p4,p5);
		}

}
```

Java Comparator implementación

Para ordenar la lista necesitaremos implementar un Java Comparator que decida de que forma se va a comparar cada elemento.



En nuestro caso la primera comparación que deseamos es la de nombre.

```
package com.arquitecturajava.ejemplo1;
```

```
import java.util.Comparator;

public class ComparadorNombre implements Comparator<Persona> {

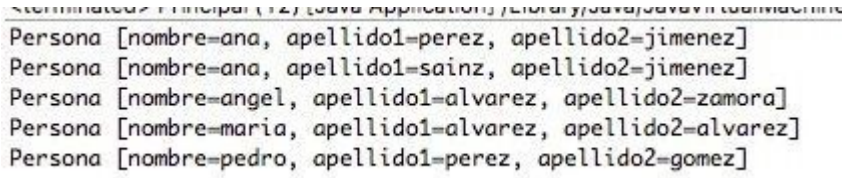
	@Override
	public int compare(Persona p1, Persona p2) {
		// TODO Auto-generated method stub
		return  p1.getNombre().compareTo(p2.getNombre());
	}

}
```

Creado el comparador el siguiente paso es utilizarlo para ordenar la lista para después imprimirla por consola.

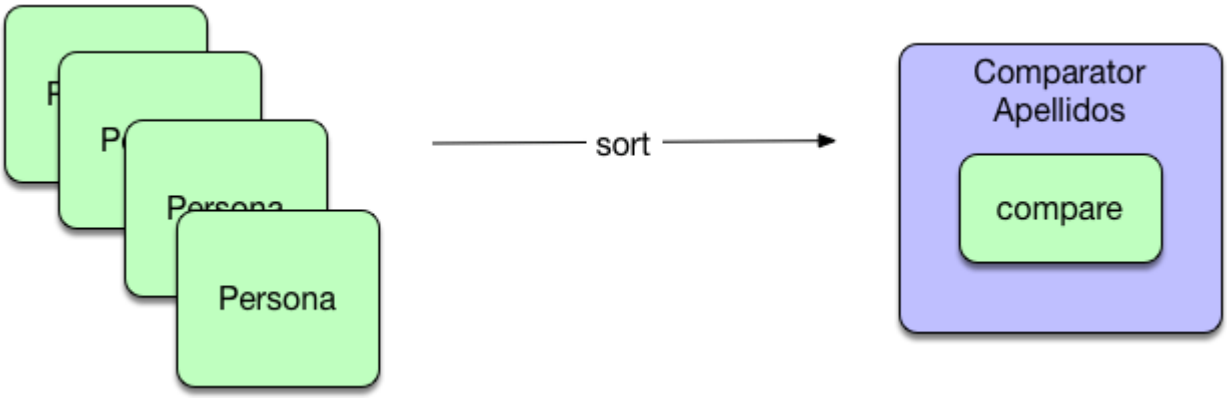
```
lista.sort(new ComparadorNombre());
lista.forEach(System.out::println);
```

La lista se imprime ordenada:



Java Comparator y ordenaciones

Hasta aquí todo funciona perfectamente. Sin embargo habrá situaciones en las cuales queramos ordenar por apellido .



Para ello en Java clásico tendríamos que crear otro Comparador.

```
package com.arquitecturajava.ejemplo1;

import java.util.Comparator;

public class ComparadorApellido implements
Comparator&amp;lt;Persona&amp;gt; {

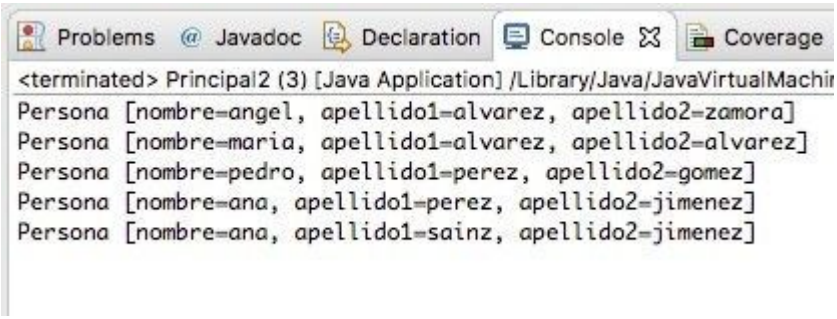
	@Override
	public int compare(Persona p1, Persona p2) {
		// TODO Auto-generated method stub
		return
p1.getApellido1().compareTo(p2.getApellido1());
	}

}
```

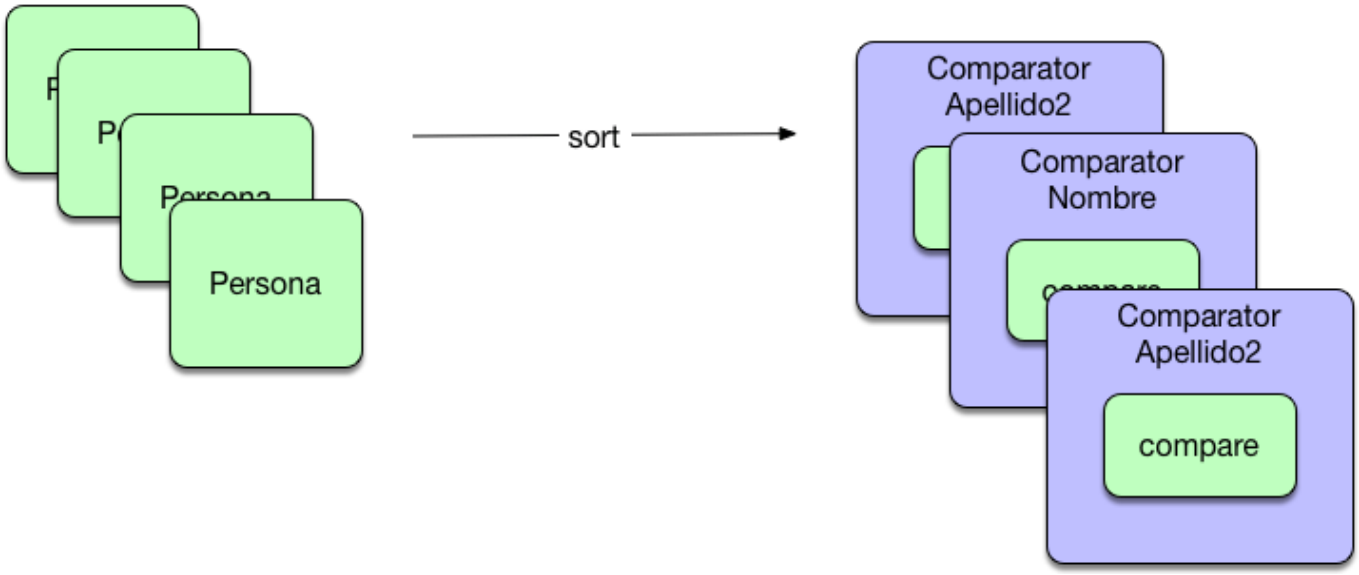
Modificamos el código y volvemos a imprimir:

```
lista.sort(new ComparadorApellido());
lista.forEach(System.out::println);
```

El resultado es :



Todo funciona , ahora bien si queremos realizar una ordenación por otro campo tenemos que construir un nuevo Comparator . Con lo cual rápidamente nos encontraremos con muchos Comparadores.



Java Comparator en Java 8

Esto en muchos casos es un problema ya que existen muchos tipos de comparaciones y combinaciones posibles. Java 8 solventa este problema utilizando expresiones Lambda.

```
package com.arquitecturajava.ejemplo1;

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

```
public class Principal {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Persona p1= new Persona("pedro","perez","gomez");
		Persona p2= new Persona("angel","alvarez","zamora");
		Persona p3= new Persona("ana","perez","jimenez");
		Persona p4= new Persona("ana","sainz","jimenez");
		Persona p5= new Persona("maria","alvarez","alvarez");

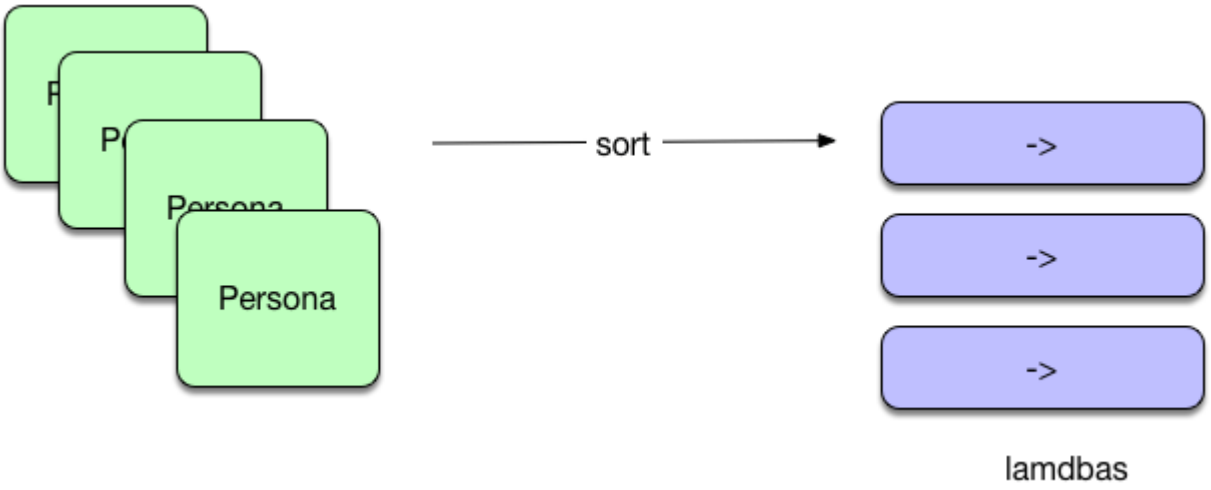
		List<Persona> lista=Arrays.asList(p1,p2,p3,p4,p5);

lista.sort((pa,pb)->pa.getNombre().compareTo(pb.getNombre()));
		lista.forEach(System.out::println);
lista.sort((pa,pb)->pa.getApellido1().compareTo(pb.getApellido1()));
		lista.forEach(System.out::println);

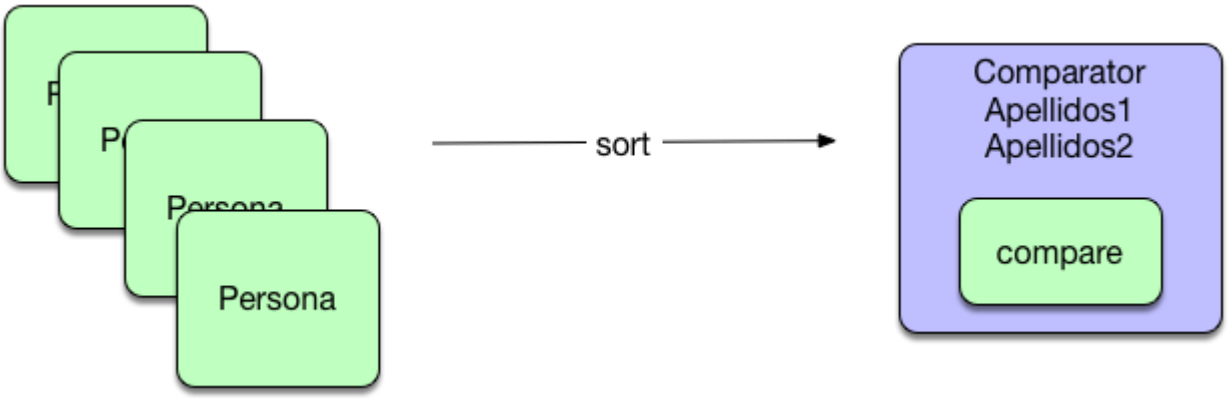
	}

}
```

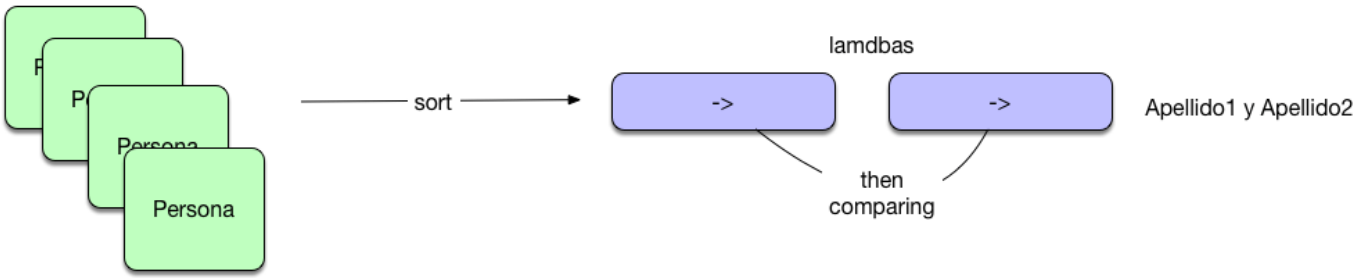
De esta forma se puede conseguir el mismo resultado , pero sin la necesidad de construir clases apoyandonos en Java 8.



No solo eso sino que ademas las expresiones las expresiones lambda a nivel de comparadores permiten su combinación . Por ejemplo imaginémonos que deseamos ordenar la lista primero por apellido1 y luego por apellido2. En Java Clásico no nos quedaría más remedio que implementar una nueva clase Comparator.



Sin embargo en Java 8 podemos encadenar expresiones lamdba a través del método thenComparing del interface Comparator.



Vamos a verlo en código:

```
package com.arquitecturajava.ejemplo1;

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

public class Principal3 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		Persona p1= new Persona("pedro","perez","gomez");
		Persona p2= new Persona("angel","alvarez","zamora");
		Persona p3= new Persona("ana","perez","jimenez");
		Persona p4= new Persona("ana","sainz","jimenez");
		Persona p5= new Persona("maria","alvarez","alvarez");

		List<Persona>lista=Arrays.asList(p1,p2,p3,p4,p5);

		Comparator<Persona> comparadorA=
(pa,pb)->pa.getApellido1().compareTo(pb.getApellido1());
		Comparator<Persona>
comparadorB=comparadorA.thenComparing((pa,pb)->pa.getApellido2().compa
reTo(pb.getApellido2()));
```

```
                lista.sort(comparadorB);

                lista.forEach(System.out::println);

        }

}
```

Podemos ver el resultado en la consola:



Nos ha ordenado primero por el apellido1 y luego por el apellido2. Java 8 aporta muchos puntos de mejora sobre java clásico:

Otros artículos relacionados:

1. Java Optional ifPresent y como utilizarlo
2. Java Lambda reduce y wrappers
3. Java 8 Lambda Expressions (I)

Links externos

1. Java Lambda Oracle