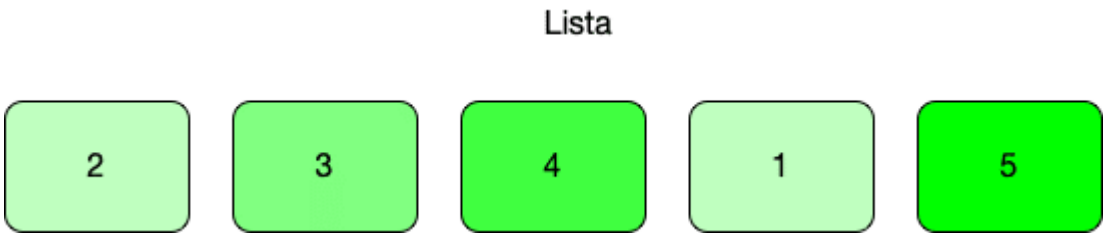
El concepto de Java Stream Sorted es bastante común cuando trabajamos en Java ya que existen muchas situaciones en la que tenemos una lista de elementos en Java y necesitamos ordenarla de forma sencilla.



Para ello podemos apoyarnos en el método sorted de los Java Streams . Eso sí recordemos que para ordenar cualquier conjunto de objetos de una clase necesitamos en un primer lugar o como punto de partida implementar el interface Comparable. Vamos a ver una serie de casos a través del concepto de Persona.

```
package com.arquitecturajava;

public class Persona implements Comparable {

	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;
	}

	@Override
	public String toString() {
		return "Persona [nombre=" + nombre + ", apellidos=" +
apellidos + ", edad=" + edad + "]";
	}

	@Override
	public int compareTo(Object o) {
		Persona otra = (Persona) o;
		return this.getNombre().compareTo(otra.getNombre());
```

```
        }

}
```

Java Stream Sorted

Como podemos ver hemos implementado el método compareTo para comparar de forma automática dos Personas por nombre . Ademas hemos sobreescrito el método toString para facilitar su impresión en la consola y ver las operaciones de una forma más directa. Vamos a construir una lista de Personas y a través del método sorted y la implementación del interface Comparable imprimirla por la consola.

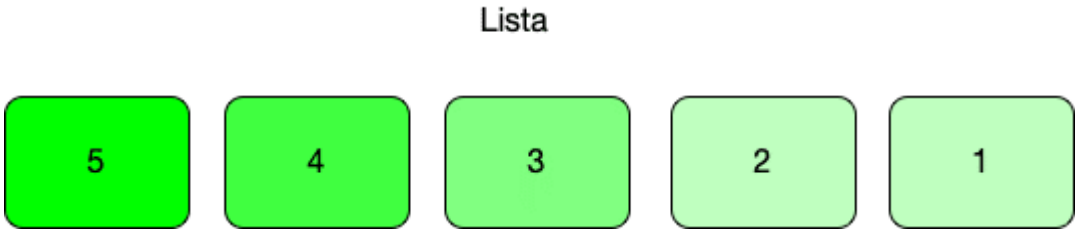
```
package com.arquitecturajava;

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

public class Principal {

        public static void main(String[] args) {
                Persona p1= new Persona("juan","blanco",30);
                Persona p2= new Persona("juan","alvarez",25);
                Persona p3= new Persona("ana","sanchez",40);
                Persona p4= new Persona("ana","simancas",30);
                Persona p5= new Persona("ana","simancas",25);
                Persona p6= new Persona("amaya","sierra",35);
                Persona p7= new Persona("jose","perez",60);
                List<Persona> lista=
Arrays.asList(p1,p2,p3,p4,p5,p6,p7);
               lista.stream().sorted().forEach(System.out::println);
        }
```

}

Si ejecutamos el código el resultado será:

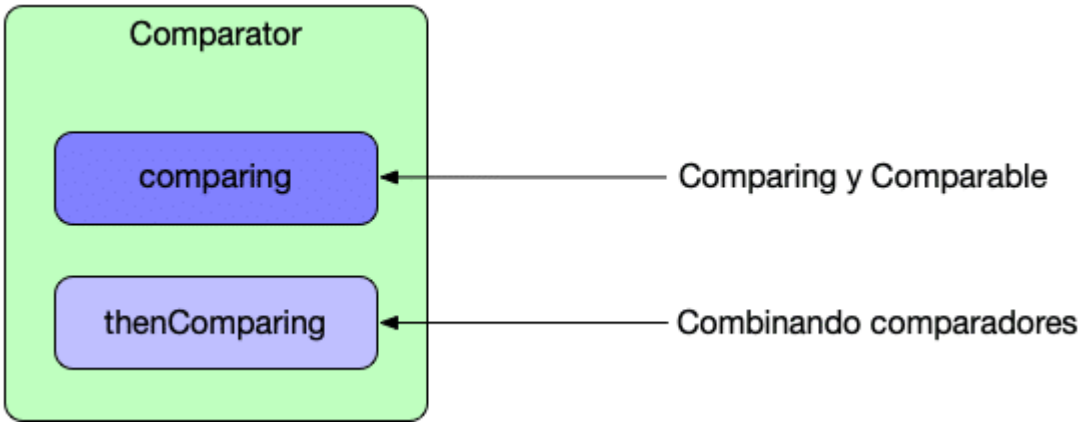
```
Persona [nombre=amaya, apellidos=sierra, edad=35]
Persona [nombre=ana, apellidos=sanchez, edad=40]
Persona [nombre=ana, apellidos=simancas, edad=30]
Persona [nombre=ana, apellidos=simancas, edad=25]
Persona [nombre=jose, apellidos=perez, edad=60]
Persona [nombre=juan, apellidos=blanco, edad=30]
Persona [nombre=juan, apellidos=alvarez, edad=25]
```

Como podemos ver la lista aparece ordenada por nombre que es lo que deseábamos



Java Comparator y comparing

De la misma manera que podemos usar el interface Comparable podemos hacer uso de un helper method del interface Comparator (comparing) para comparar apoyándonos en cualquiera de las propiedades de la clase Persona.



Por ejemplo por el apellido:

```
lista.stream().sorted(Comparator.comparing(Persona::getApellidos)).forEach(System.out::println);
```

El resultado será una lista en los cuales los elementos aparecen ordenados por apellidos:

```
Persona [nombre=juan, apellidos=blanco, edad=30]
Persona [nombre=juan, apellidos=alvarez, edad=25]
Persona [nombre=ana, apellidos=sanchez, edad=40]
Persona [nombre=ana, apellidos=simancas, edad=30]
Persona [nombre=ana, apellidos=simancas, edad=25]
Persona [nombre=amaya, apellidos=sierra, edad=35]
Persona [nombre=jose, apellidos=perez, edad=60]
```

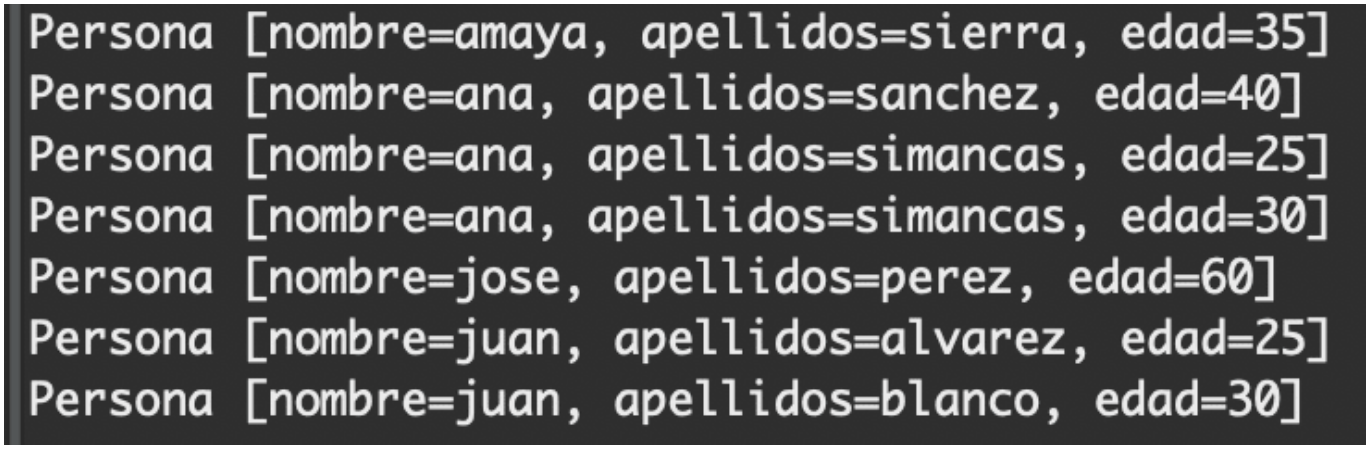
Java Stream Sorted (Comparator thenComparing)

No solo eso es posible sino que ademas podemos solicitar a través del metodo thenComparing una comparación que se aplique a varios campos.

```
        Comparator<Persona> comparadorMultiple=
Comparator.comparing(Persona::getNombre).thenComparing(Comparator.comparing(Persona::getApellidos)).thenComparing(Comparator.comparing(Persona::getEdad));
```

```
lista.stream().sorted(comparadorMultiple).forEach(System.out::println)
;
```

El resultado lo podemos ver en la consola con todos los campos ordenados:



Java Comparator y Comparable

Acabamos de ordenar una lista de Personas de varias maneras apoyandonos en los interfaces Comparable y Comparator.

Otros artículos relacionados

- Java Stream
- Java Stream Filter
- Java Stream Map
- Java Stream Reduce
- Java Stream Oracle