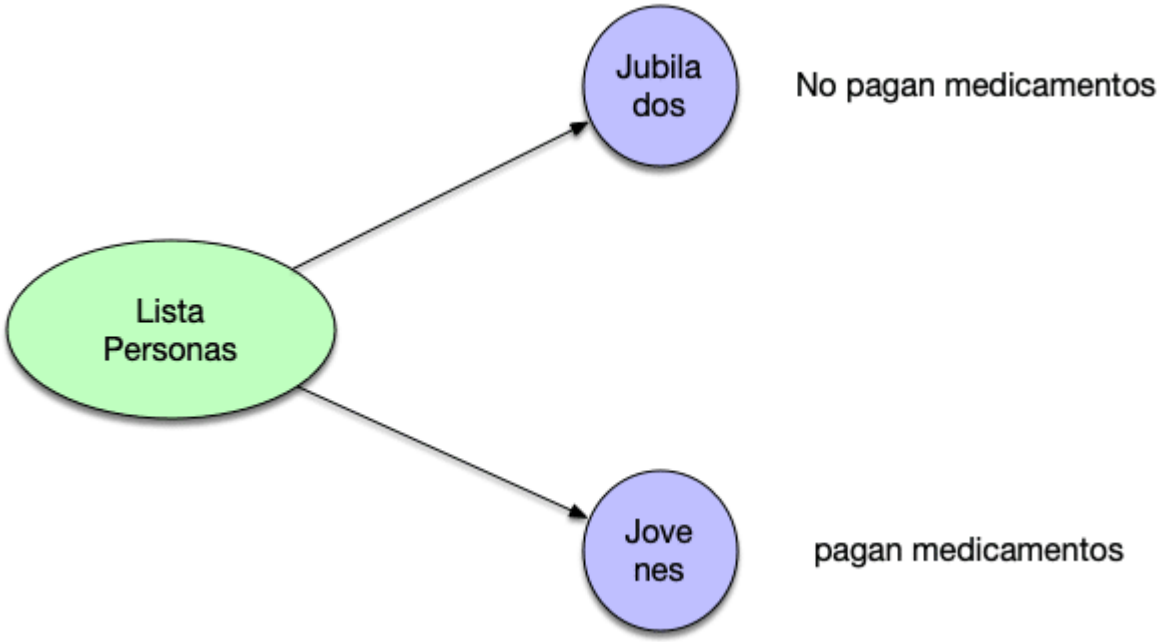
Tabla de Contenidos

- Persona y Objetos de negocio
- Java Stream PartitionBy y Mapas
- Otros artículos relacionados

El concepto de Java Stream Partition es un concepto bastante útil en nuestro día a día cuando nos encontramos manejando listas de objetos. El manejo de estas es muy habitual y entre las operaciones comunes se encuentra el quedarnos con una lista de objetos A o una lista de objetos dependiendo de algún tipo de condición . Por ejemplo podemos tener una lista de Personas y dependiendo si están jubilados o no imprimir por pantalla que deben pagar los medicamentos o son gratuitos para ellos. Se trata de una lógica muy sencilla en la cual tendremos que generar a partir de una lista inicial dos listas una con las personas que están jubiladas y otras con las que no lo están.



Persona y Objetos de negocio

Lo primero que tendremos que hacer es crear la clase Persona que en este caso contendrá un método que nos permitirá decidir si esa persona esta en edad de Jubilación o no.

```
package com.arquitecturajava;

public class Persona {

	private String nombre;
	private int edad;
	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) {
		super();
		this.nombre = nombre;
		this.edad = edad;
	}
	public boolean estaJubilado() {
		return this.getEdad()>65;
	}
}
```

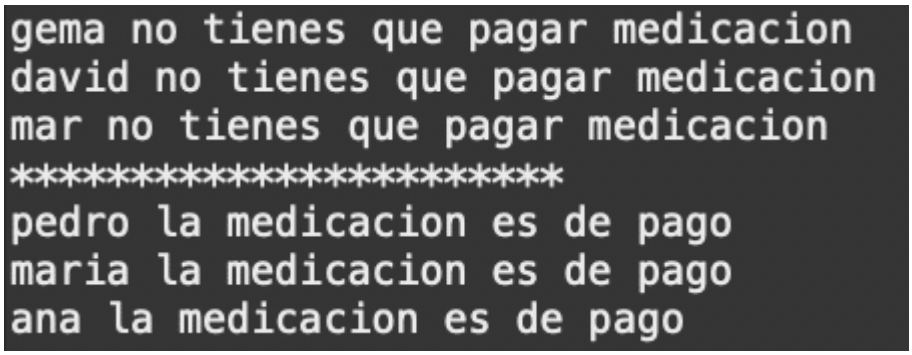
Para poder dividir la lista en dos sublistas de una forma sencilla , podríamos construir un programa similar a este :

```
package com.arquitecturajava;
```

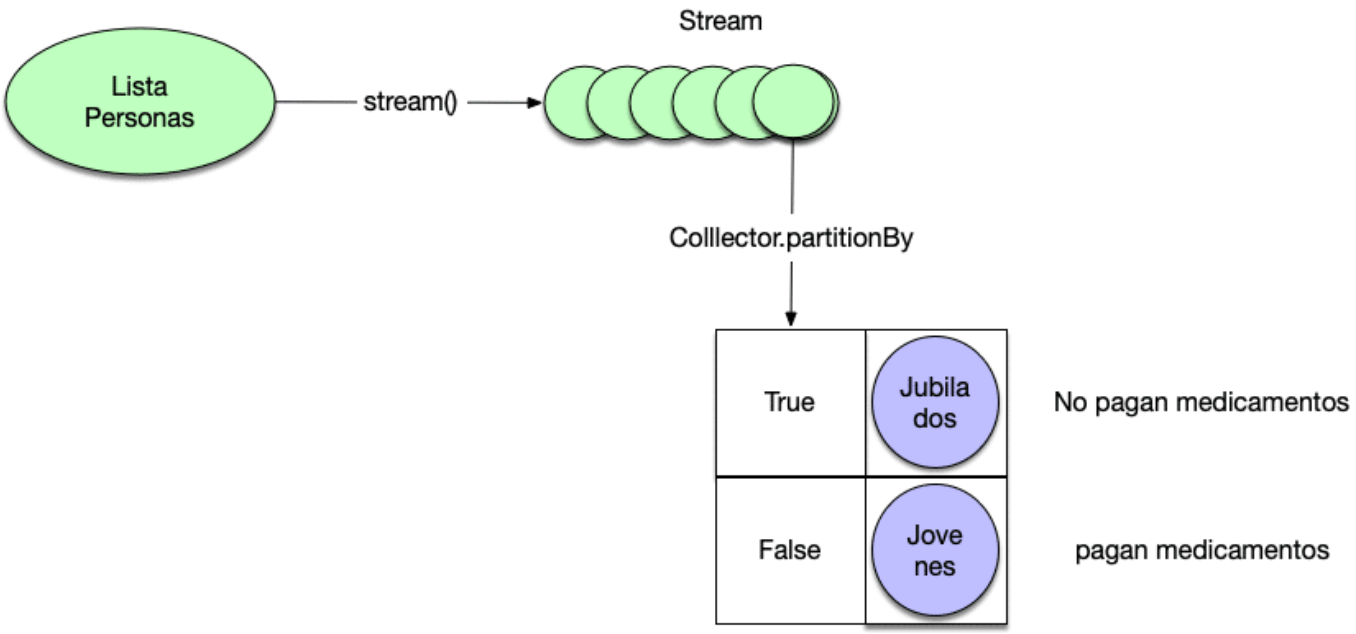
```
import java.util.ArrayList;
import java.util.List;

public class Principal {

	public static void main(String[] args) {

		List<Persona> lista = new ArrayList<Persona>();
		lista.add(new Persona("pedro", 20));
		lista.add(new Persona("maria", 50));
		lista.add(new Persona("gema", 70));
		lista.add(new Persona("ana", 15));
		lista.add(new Persona("david", 75));
		lista.add(new Persona("mar", 70));

		List<Persona> jubilados = new ArrayList<Persona>();
		List<Persona> jovenes = new ArrayList<Persona>();

		for (Persona p : lista) {

			if (p.estaJubilado()) {
				jubilados.add(p);
			} else {
				jovenes.add(p);
			}
		}

		for (Persona p : jubilados) {

			System.out.println(p.getNombre() + " no tienes
que pagar medicacion");
```

```
                }

                System.out.println("**********************");
                for (Persona p : jovenes) {

                        System.out.println(p.getNombre() + " la
medicacion es de pago");
                }
        }

}
```

Si ejecutamos el programa en la consola obtendremos dos bloques de mensajes sencillos.

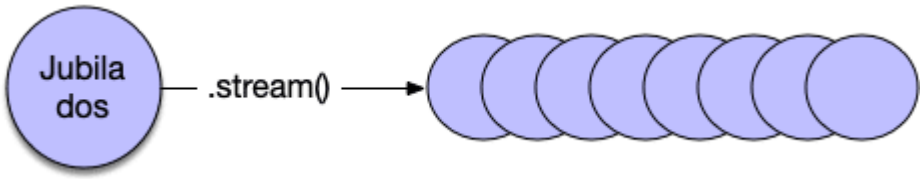


Todo funciona correctamente . ¿Existe alguna forma más directa y compacta de hacerlo?. La respuesta es sí y para ello deberemos apoyarnos en Java Stream Partition, un método de la clase Collectors de Java que nos permite realizar una partición rápida de un Stream de Datos en dos listas que se almacenarán en un Mapa.



Java Stream PartitionBy y Mapas

Una vez que disponemos de los dos mapas con las listas independencias es cuestión de trabajar con las listas en formato Stream e imprimir la información por la consola.



Vamos a ver el código que ayudará a aclarar las cosas y entender la simplicación que supone:

```
package com.arquitecturajava;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

public class Principal {
```

```
        public static void main(String[] args) {

                List<Persona> lista = new ArrayList<Persona>();
                lista.add(new Persona("pedro", 20));
                lista.add(new Persona("maria", 50));
                lista.add(new Persona("gema", 70));
                lista.add(new Persona("ana", 15));
                lista.add(new Persona("david", 75));
                lista.add(new Persona("mar", 70));

                Map<Boolean,List<Persona>>
mapa=lista.stream().collect(Collectors.partitioningBy(Persona::estaJub
ilado));
mapa.get(true).stream().map(Persona::getNombre).forEach(n->System.out.
println(n+" no tienes que pagar medicacion"));
                System.out.println("**********************");
mapa.get(false).stream().map(Persona::getNombre).forEach(n->System.out
.println(n+" la medicacion es de pago"));
        }

}
```

De esta forma el resultado es el mismo que anteriormente pero el código es mucho más compacto.

Otros artículos relacionados

- ¿Que es un Java Stream?
- Java Stream forEach y colecciones
- Java Stream peek funcionamiento y logging
- Curso Java 8 Streams