Tabla de Contenidos

Java Stream groupby . Cuando trabajamos con listas de objetos en Java, muchas veces necesitamos agruparlos de alguna manera, por ejemplo, por algún atributo común. Con la llegada de los Streams en Java 8, esta tarea se volvió mucho más simple y elegante. Hoy te voy a enseñar cómo puedes agrupar una lista de personas por su nombre usando Streams y la función Collectors.groupingBy().

La Clase Persona

Primero, definimos una clase sencilla Persona que va a tener los atributos: nombre, apellidos y edad. Así de simple:


```
public class Persona {
    private String nombre;
```

```
    private String apellidos;
    private int edad;

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

    public String getNombre() {
        return nombre;
    }

    public String getApellidos() {
        return apellidos;
    }

    public int getEdad() {
        return edad;
    }

    @Override
    public String toString() {
        return "Persona{" +
                "nombre='" + nombre + '\'' +
                ", apellidos='" + apellidos + '\'' +
                ", edad=" + edad +
                '}';
    }
}
```

La Lista de Personas

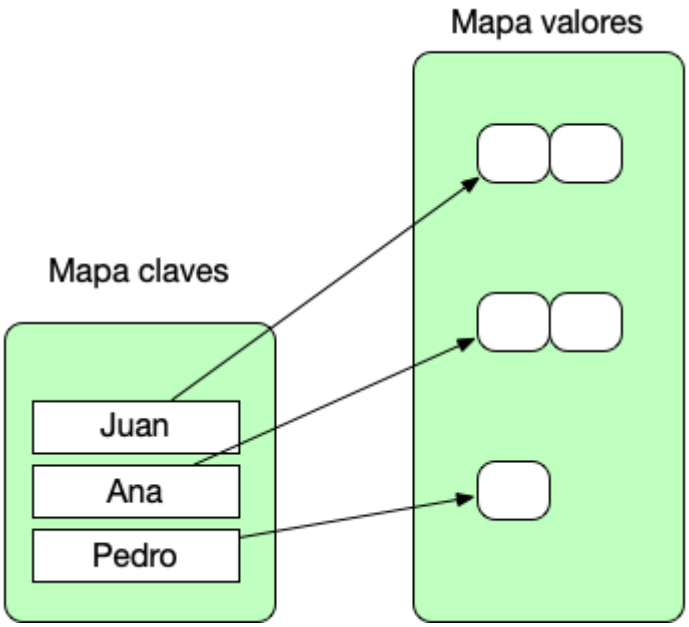
Ahora vamos a crear una lista con varias personas y veremos cómo podemos agruparlas por el atributo `nombre` usando la funcionalidad de java stream groupby (groupingBy)

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

public class Main {
    public static void main(String[] args) {
        List<Persona> personas = Arrays.asList(
                new Persona("Juan", "Pérez", 30),
                new Persona("Ana", "García", 25),
                new Persona("Juan", "López", 28),
                new Persona("Ana", "Martínez", 32),
                new Persona("Pedro", "Rodríguez", 40)
        );
        // Agrupamos por nombre
        Map<String, List<Persona>> personasAgrupadasPorNombre =
personas.stream()
                .collect(Collectors.groupingBy(Persona::getNombre));
        // Imprimimos el resultado
        personasAgrupadasPorNombre.forEach((nombre, listaPersonas) ->
{
            System.out.println("Nombre: " + nombre);
            listaPersonas.forEach(System.out::println);
        });
    }
}
```

Java Stream GroupBy y funcionamiento

1. Creamos una lista de personas: Cada una tiene un nombre, apellidos y edad.
2. Streams y groupingBy(): Con la ayuda de los Streams, convertimos la lista en un flujo de datos y luego usamos Collectors.groupingBy() para agrupar las personas según su `nombre`.
3. Imprimimos el resultado: Con forEach(), recorremos el `Map` que nos devuelve la agrupación e imprimimos cada grupo de personas por su nombre. Demonos cuenta de que el mapa contiene la clave de nombre y por cada valor una lista.



¿Y qué obtenemos como salida?

Si ejecutas este código, verás algo como esto:

```
Nombre: Pedro
Persona{nombre='Pedro', apellidos='Rodríguez', edad=40}
Nombre: Juan
Persona{nombre='Juan', apellidos='Pérez', edad=30}
Persona{nombre='Juan', apellidos='López', edad=28}
Nombre: Ana
Persona{nombre='Ana', apellidos='García', edad=25}
```

```
Persona{nombre='Ana', apellidos='Martínez', edad=32}
```

Por qué usar Streams?

- Menos código, más claro: Los Streams te permiten hacer muchas operaciones comunes (como filtrar, mapear y agrupar) con muy pocas líneas de código.
- Fácil de leer: Al tener un enfoque más declarativo, el código se vuelve más fácil de seguir y entender.
- Rendimiento: Los Streams tienen soporte para procesamiento paralelo, lo que significa que pueden mejorar el rendimiento cuando trabajamos con grandes cantidades de datos.

Filtrando antes de agrupar

¿Y si solo queremos agrupar a las personas que tienen más de 30 años? Es tan sencillo como agregar un filter() antes de agrupar:

```
Map<String, List<Persona>> personasMayoresDe30 = personas.stream()
        .filter(p -> p.getEdad() > 30)
        .collect(Collectors.groupingBy(Persona::getNombre));
```

Con esto, solo estaríamos agrupando a las personas mayores de 30 años.

Java Stream GroupBy Conclusiónes

Agrupar objetos en Java se vuelve muy sencillo con la combinación de Streams y Collectors.groupingBy(). En este ejemplo vimos cómo agrupar una lista de personas por el atributo `nombre`, pero esta técnica es aplicable a cualquier otro atributo o tipo de agrupación que necesites.Espero que esta explicación te haya sido útil. ¡Anímate a probarlo en tus propios proyectos y verás lo práctico que es usar este método

- Utilizando atributos con JSF HTML5
- JDBC Batch y rendimiento
- JUnit Test Suite , TDD y organización
- Spring @Service , usando el patrón Servicio

- JSF y Atributos Rendered