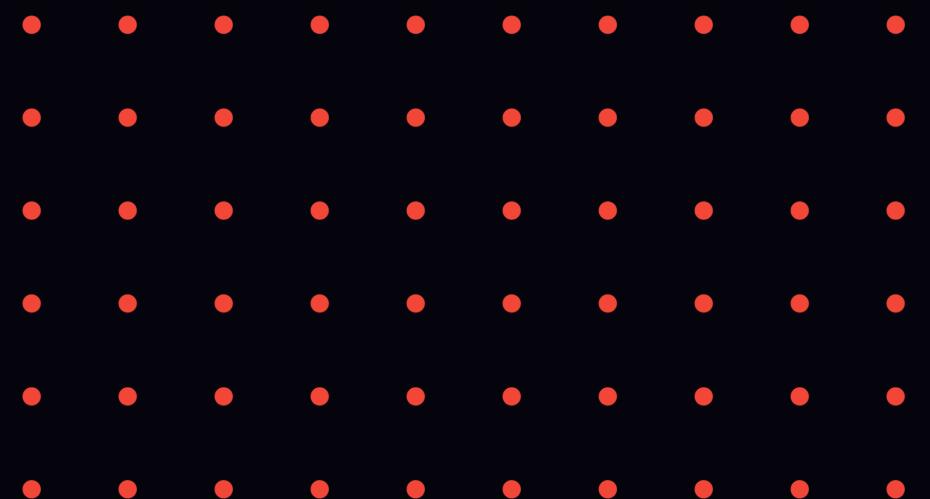




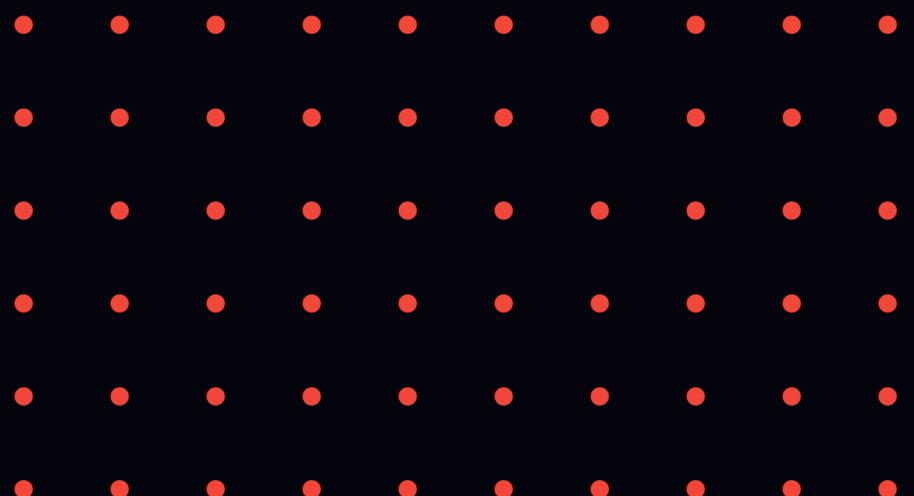
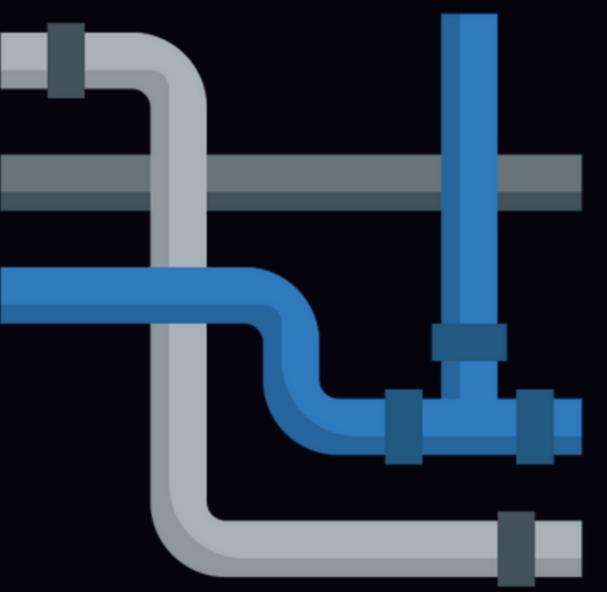
Stream

Por: Gabriel Chaldú



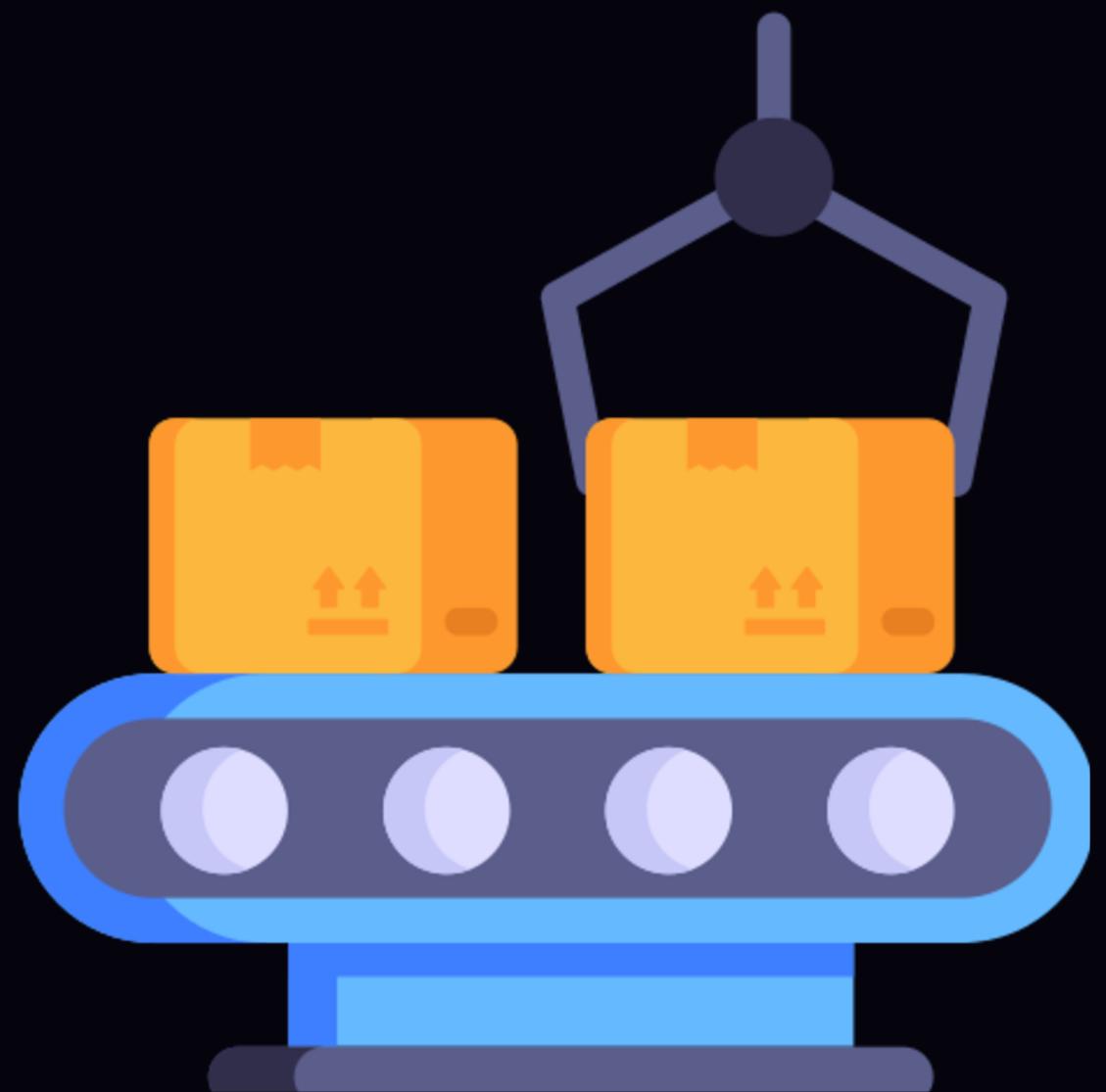
¿Qué es un Stream?

Es una herramienta que permite **procesar** una colección de datos y **declarativa, sin modificar la fuente original.**



Cinta Transportadora

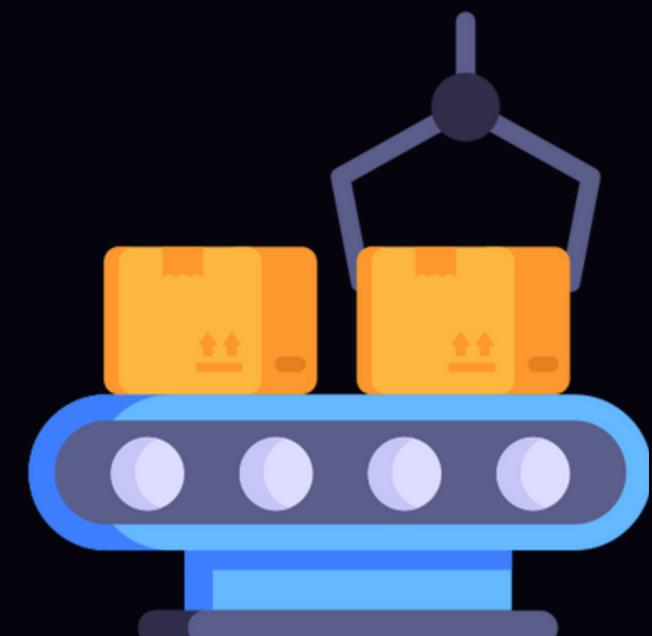
1. **Filtrado**: se retiran los productos defectuosos.
2. **Transformación**: se etiquetan los productos restantes.
3. **Contar**: se cuentan los productos finales antes de empaquetarlos.



Así funciona un Stream en Java:

1. **Los productos** representan los **datos** en la colección.
2. **La cinta** *procesa los datos* sin almacenarlos.
3. **Las estaciones** son las *operaciones intermedias* (filter(), map(), etc.).
4. **El empaquetado** es una *operación terminal* (collect(), count(), etc.).

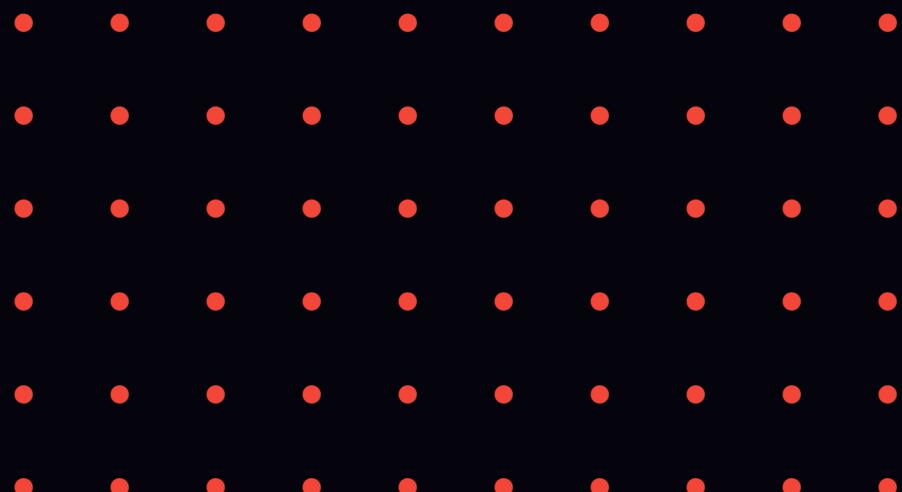
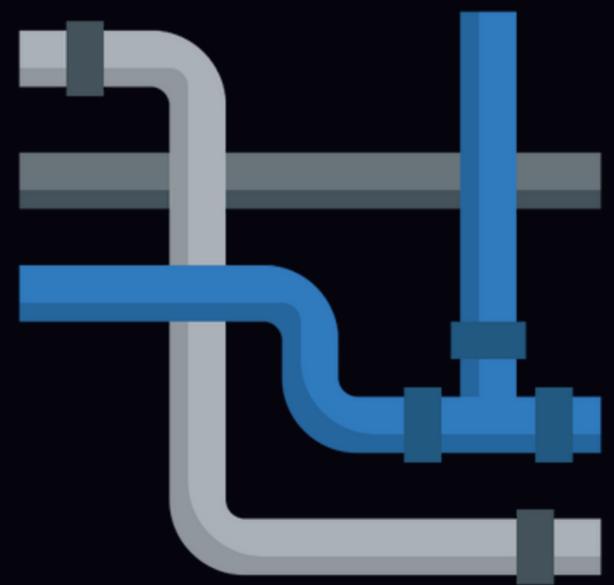
Un Stream no altera los datos originales.



Ejemplo básico

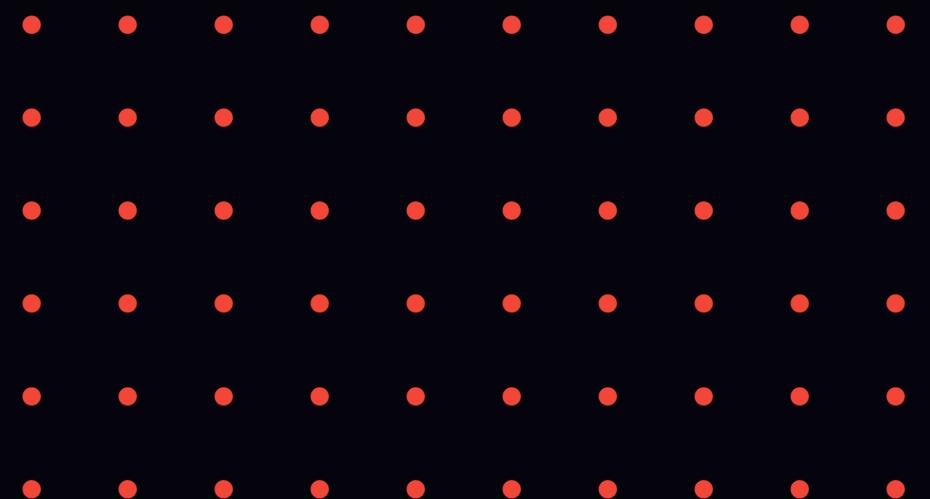
- Objetivo: imprimir una lista de nombres usando un Stream.

```
public static void main(String[] args) {  
    List<String> names = Arrays.asList("Carolina", "Fernando", "Ana");  
  
    names.stream() // Creamos el Stream a partir de la lista  
        .forEach(System.out::println); // Operación final que imprime cada elemento  
}
```



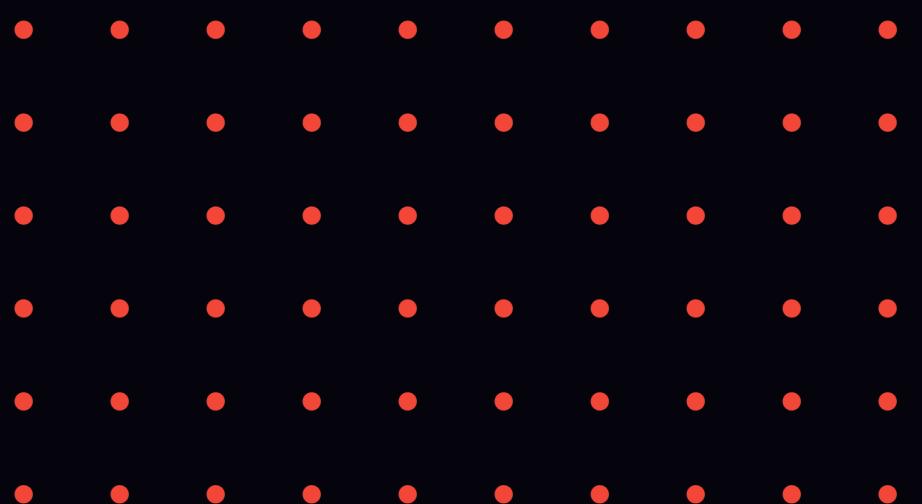
Los Stream:

- Son consumibles.
- No reutilizables por defecto.



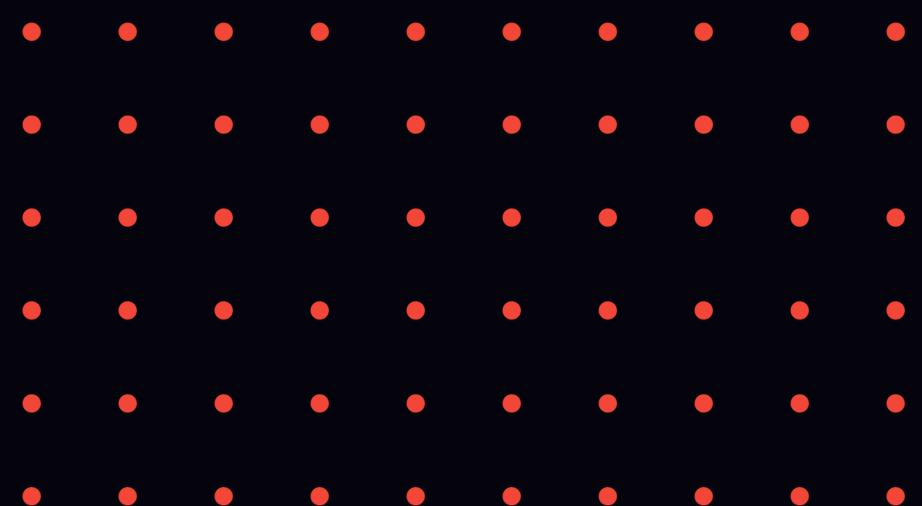
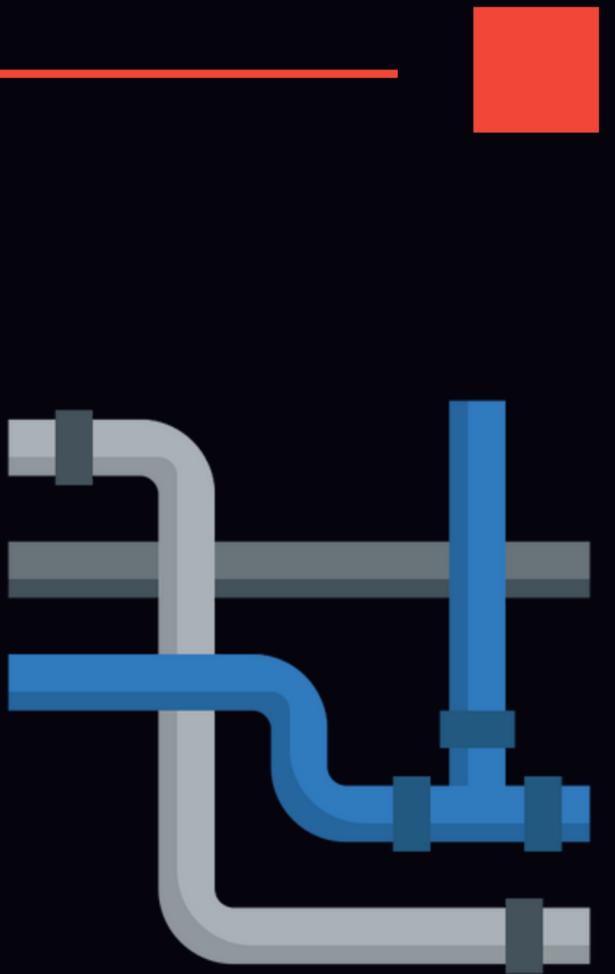
Los Stream:

```
Stream<String> names =  
    Stream.of("Gabriel", "Mauro", "María", "Ana", "Beto");  
  
names.forEach(name -> System.out.println(name));  
  
List<String> list = names.toList();  
  
System.out.println(list)
```



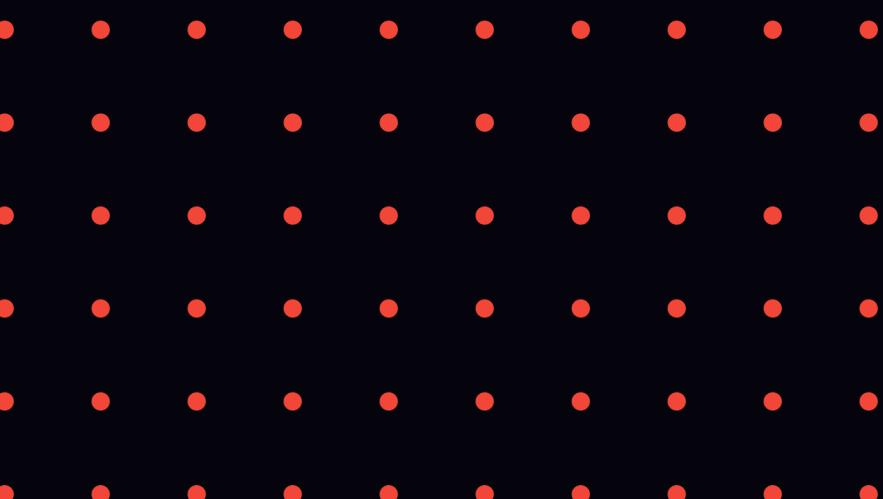
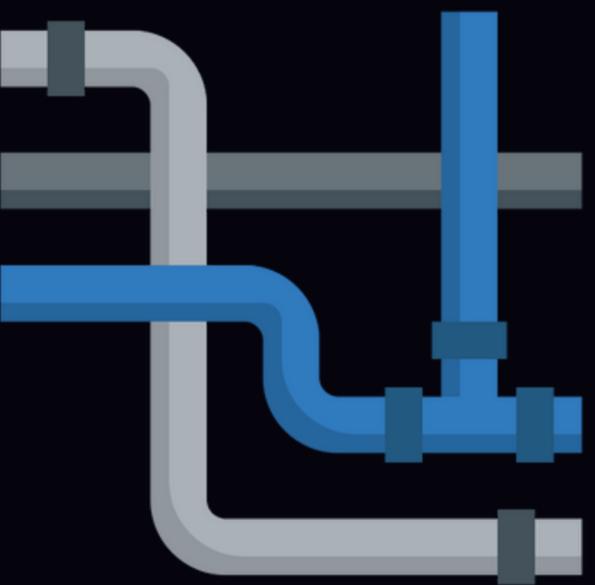
Operaciones intermedias

- Transforman un Stream en otro Stream.
- **Son perezosas (lazy)**: no se ejecutan hasta que una operación terminal las obliga.



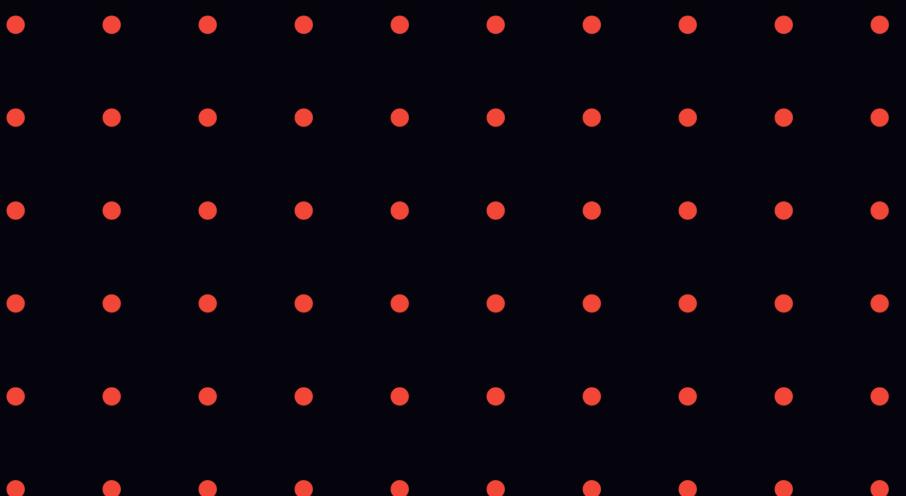
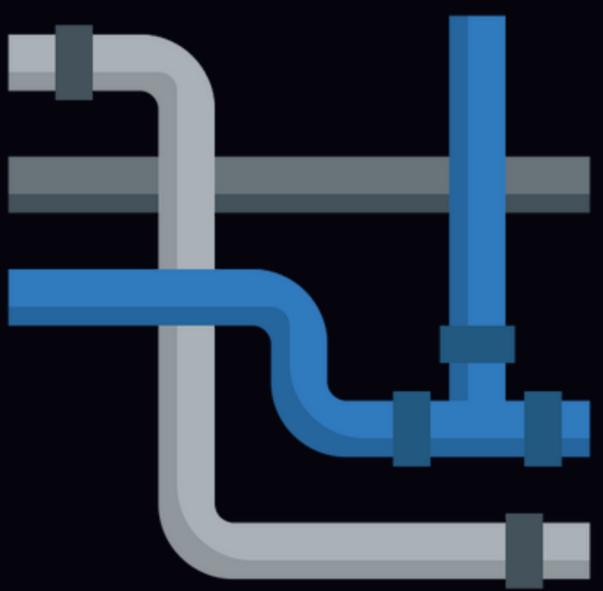
Ejemplo lazy

```
public class LazyStreamExample {  
    public static void main(String[] args) {  
        List<String> names = Arrays.asList("Gabriel", "Fernando", "Maria", "Marta");  
  
        Stream<String> filteredStream = names.stream()  
            .filter(name -> {  
                System.out.println("Filtrando: " + name);  
                return name.startsWith("M");  
            });  
  
        System.out.println("Nada se ha impreso aún"); // Nada pasa todavía  
  
        filteredStream.forEach(System.out::println); // Ahora sí se ejecuta el filtro y se  
        // imprimen los nombres  
    }  
}
```



Operaciones terminales

- Son aquellas que **consumen el Stream**.
- ***Devuelven un resultado concreto*** (lista, número, impresión en consola, etc.).
- Un Stream solo puede tener ***UNA operación terminal.***



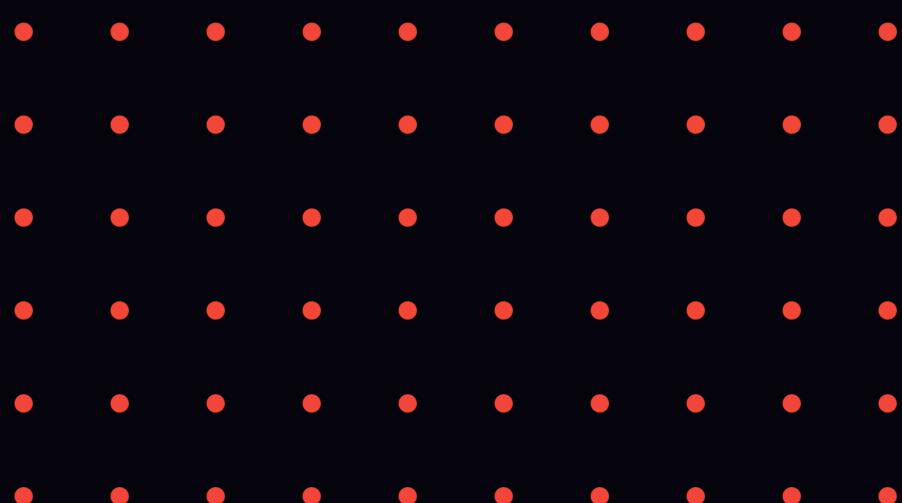
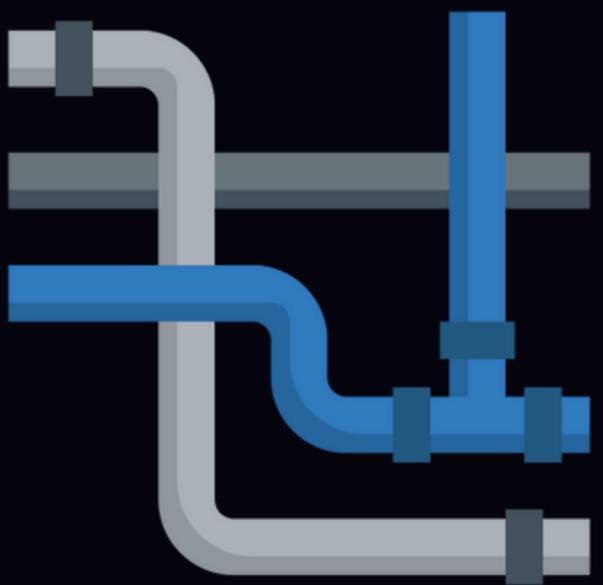
Ejemplo count()

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

public class TerminalStreamExample {
    public static void main(String[] args) {
        List<String> names = Arrays.asList("Gabriel", "Fernando", "Maria", "Marta");

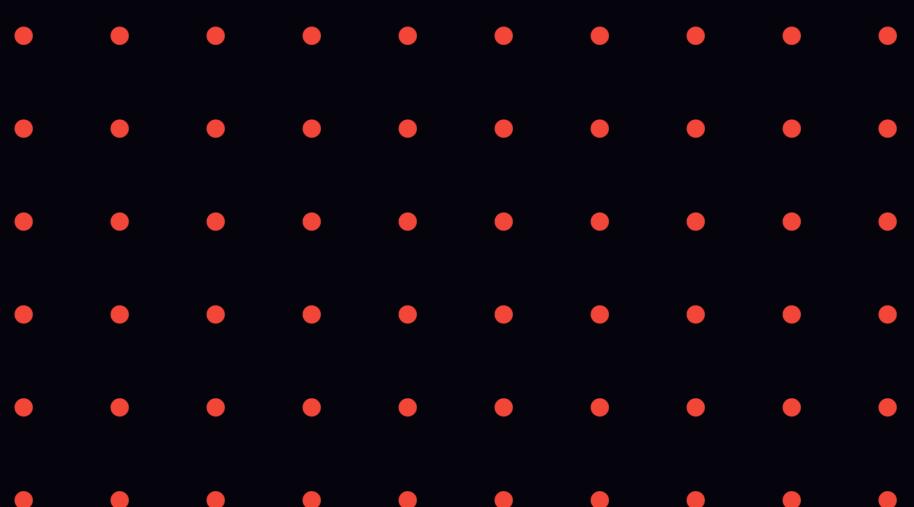
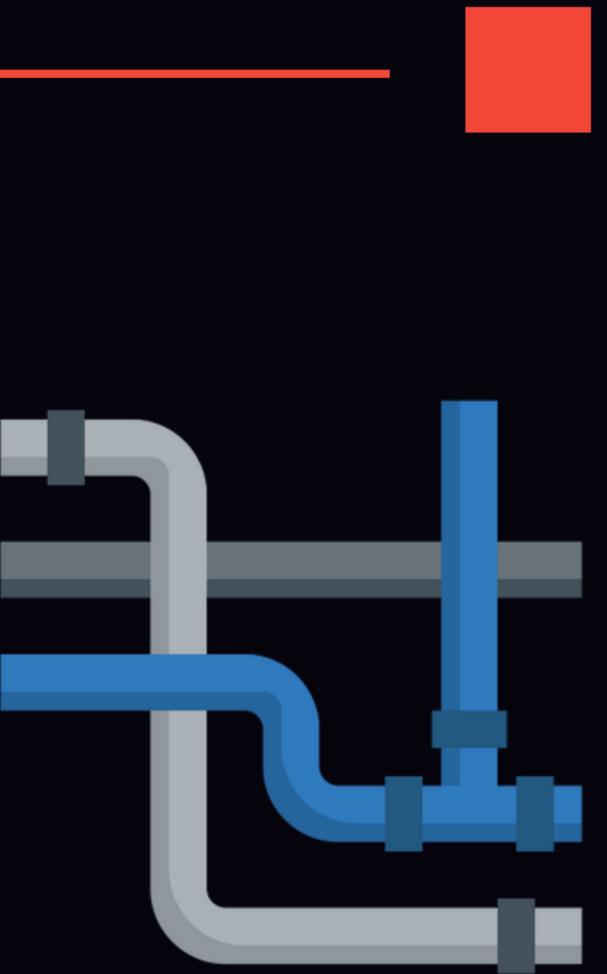
        long count = names.stream()
            .filter(name -> name.startsWith("M")) // Intermedia
            .count(); // Terminal

        System.out.println("Cantidad de nombres que empiezan con M: " + count);
    }
}
```



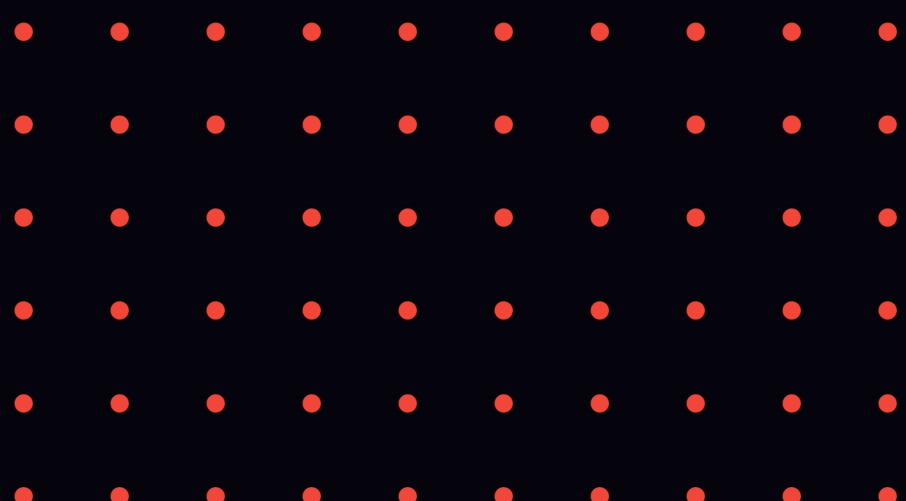
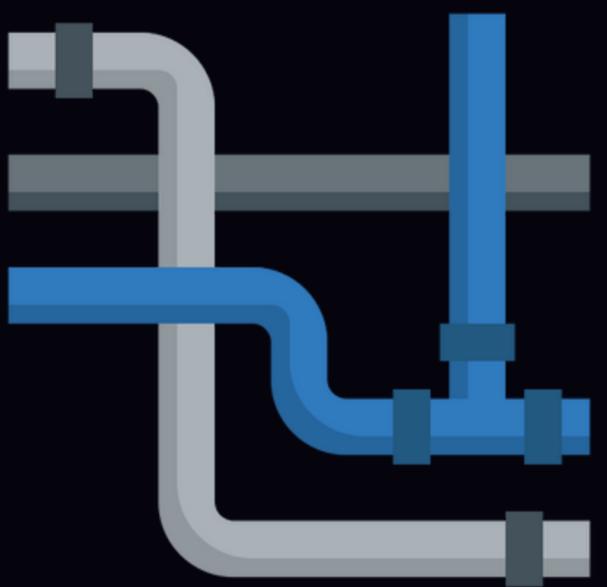
¿Cómo diferenciar una OI de una OT?

- Si devuelve un Stream, es intermedia.
- Si devuelve otro tipo de dato, es terminal
(porque el Stream termina ahí).



¿Cuando utilizar OI o OT?

Situación	Operación recomendada
Necesitas transformar datos.	<code>map()</code>
Debes recolectar los datos en una lista.	<code>filter()</code>
Solo necesitas recorrer e imprimir.	<code>Collect()</code>





**¡A seguir
mejorando!**

