

# Streams

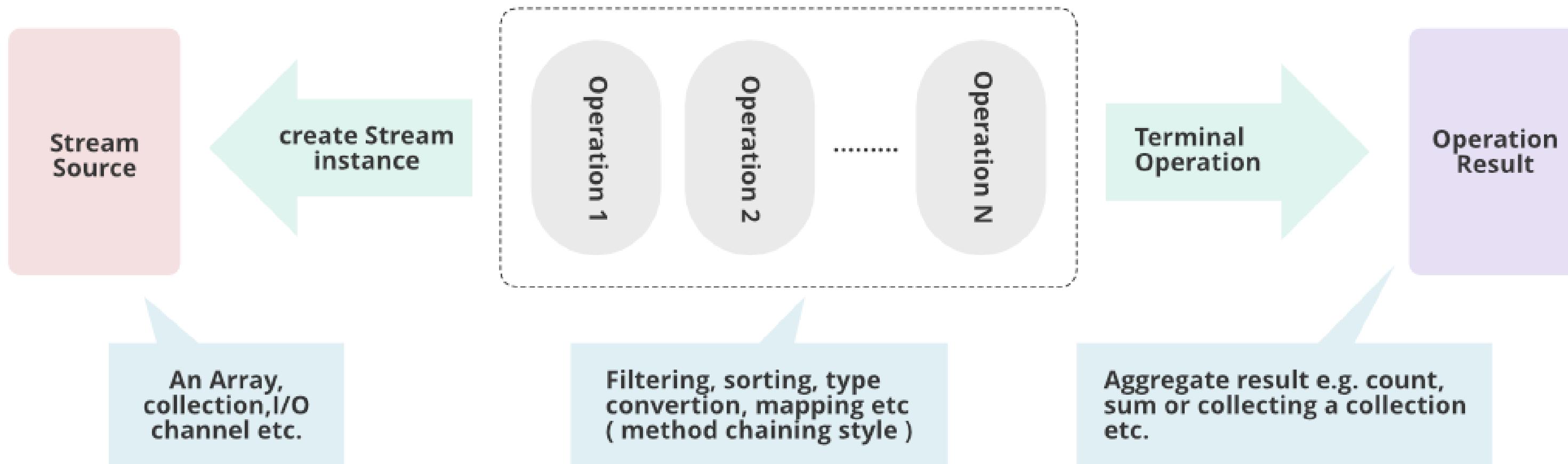
Danna Valeria Morales Aguilar

## ¿QUÉ ES UN STREAM?

Es una secuencia de elementos que se procesan de manera funcional y en paralelo utilizando operaciones de alto nivel. Se usan para trabajar con colecciones de datos de manera mas elegante y eficiente, permitiendo realizar operaciones de filtrado, transformación, agrupación y reducción de datos de manera declarativa.

# Java Streams

## Intermediate Operations



# CARACTERÍSTICAS

- Declaratividad

Permiten escribir código especificando que deseas hacer con los datos en lugar de cómo hacerlo.

- Operaciones de alto nivel

Dan una variedad de operaciones de alto nivel como filter, map, reduce, collect, etc.

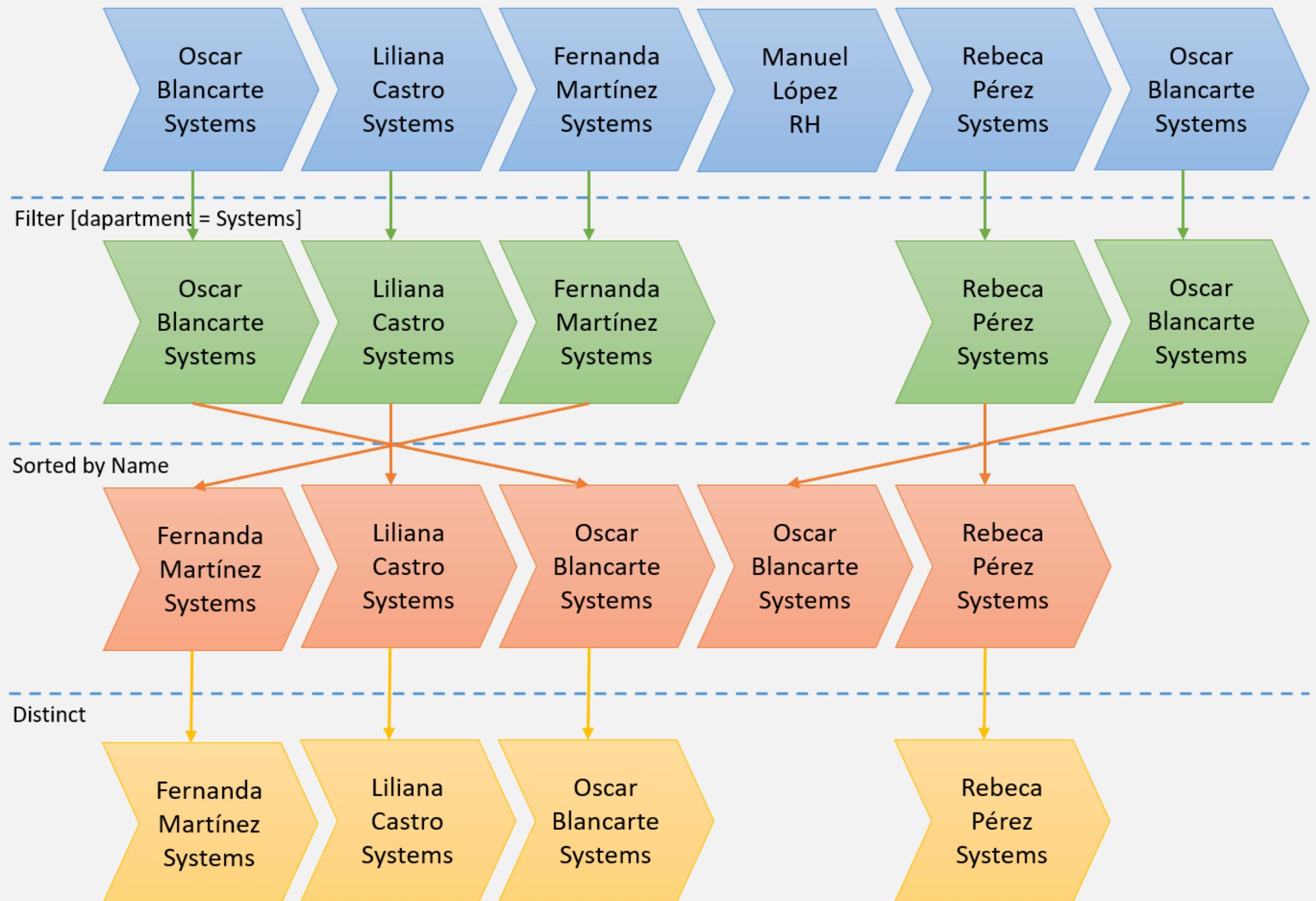
- Lazy evaluation

Usan una evaluación perezosa, o sea se ejecutan solo cuando se necesita el resultado final, optimiza el rendimiento.

- Paralelismo

- No modifican los datos originales.

## Java Streams



EJEMPLO

```
public class Empleado {  
  
    private int id;  
    private String nombre;  
    private String departamento;  
  
    public Empleado(int id, String nombre,  
                    String departamento) {  
        this.id = id;  
        this.nombre = nombre;  
        this.departamento = departamento;  
    }  
  
    public int getId() {  
        return id;  
    }  
  
    public void setId(int id) {  
        this.id = id;  
    }  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
}
```

Se define una clase Empleado como campos de id, nombre y departamento, así como sus métodos getters, setters, equals y hashCode.

```
@Override  
public boolean equals(Object o) {  
    if (this == o)  
        return true;  
    if (o == null || getClass() != o.getClass())  
        return false;  
    Empleado empleado = (Empleado) o;  
    return id == empleado.id;  
}  
  
@Override  
public int hashCode() {  
    return Objects.hash(id);  
}
```

```
public class Main {  
  
    public static void main(String[] args) {  
        List<Empleado> listaEmpleados = new ArrayList<>();  
  
        listaEmpleados.add(new Empleado(1, "Oscar Blancarte", "Systems"));  
        listaEmpleados.add(new Empleado(2, "Liliana Castro", "Systems"));  
        listaEmpleados.add(new Empleado(3, "Fernanda Martinez", "Systems"));  
        listaEmpleados.add(new Empleado(4, "Manuel Lopez", "RH"));  
        listaEmpleados.add(new Empleado(5, "Rebeca Perez", "Systems"));  
        listaEmpleados.add(new Empleado(1, "Oscar Blancarte", "Systems"));
```

Se crea una lista llamada listaEmpleados que contiene objetos de la clase Empleado. Estos objetos representan a diferentes empleados con sus respectivos datos.

```
List<Empleado> empleadosSystemsOrdenadosSinDuplicados = listaEmpleados.stream()
    .filter(empleado -> empleado.getDepartamento().equals("Systems"))
    .sorted(Comparator.comparing(Empleado::getNombre))
    .distinct()
    .collect(Collectors.toList());
```

Se utiliza `listaEmpleados.stream()` para convertir la lista en un Stream y luego `filter()` para seleccionar solo los empleados cuyo departamento es "Systems".

Sorted se usa para ordenar los empleados filtrados por su nombre en orden ascendente.

Se utiliza `distinct()` para eliminar empleados duplicados. Aquí es donde entra el método `equals()` y `hashCode()` de la clase Empleado. En este caso, dos empleados son considerados duplicados si tienen el mismo ID.

Y `collect` se usa para recopilar los resultados de las operaciones en una nueva lista.