Taller Programación Reactiva. DYAS

Carlos Zuluaga Mora

En este taller, se implementa un caso de uso relacionado con un flujo de órdenes de productos, filtrando y procesando estos datos para simular un escenario típico de validación y reporte de ventas en tiempo real.

La lógica se divide esencialmente en dos secciones: Validación de Pedidos, de acuerdo con cantidad y producto, y Reporte de Ventas.

En la sección de Validación de Pedidos, se crea una lista de órdenes utilizando la clase Order, extraída del repositorio guía del docente

```
//Define lista de productos
List<Order> orders = Arrays.asList(
        new Order( product: "Producto A",  quantity: 5,  price: 50.0),
        new Order( product: "Producto B",  quantity: 1,  price: 30.0),
        new Order( product: "Producto D",  quantity: 1,  price: 50.0),
        new Order( product: "Producto C",  quantity: 3,  price: 20.0)
);
```

Como se mencionó previamente, se establecen dos criterios de validación para las órdenes existentes en la lista:

```
//Estos son los parámetros de validación
int cantidadMinima = 5;
String producto = "Producto A";
```

Posteriormente, se define un flujo de datos mediante la librería Flux, donde se aplica un flatMap para filtrar aquellas órdenes que cumplen con la cantidad especificada.

```
// Procesar flujo de pedidos
Flux<Order> ordersFlux = Flux.fromIterable(orders)
        .flatMap(order -> {
            if (order.getQuantity() > cantidadMinima) {
                return Flux.just(order);
            } else {
                return Flux.empty();
            }
        });
```

En caso de que el flujo esté vacío, se imprime un mensaje en consola:

```
// Validar si el flujo está vacío
ordersFlux
        .map(Order::toString)
        .zipWith(Flux.range( start: 1, orders.size()), (order, index) -> order)
        .switchIfEmpty(Flux.just( data: "Ninguna orden cumple los criterios."))
        .distinct()
        .subscribe(System.out::println);
```

Ya en la sección del reporte de ventas, se mapea la lista de órdenes buscando aquellas relacionadas con el producto de validación, para calcular el precio de toda la orden, guardarlo en una variable 'total' y, mediante el método subscribe(), enviar un mensaje por consola notificando las ventas totales para dicho producto.

```
// Reporte de ventas para el producto indicado
// Se despliega en cualquier caso
Observable.fromIterable(orders) Observable<...>
        .filter(order -> producto.equals(order.getProduct()))
        .map(order -> order.getQuantity() * order.getPrice()) Observable<Double>
        .reduce(Double::sum) Maybe<Double>
        .subscribe(total -> System.out.println("Ventas totales para " + producto + ": $" + total));
```