1. Java8 Stream 활용기

1. 들어가며

첫번째 연구노트 주제로 Jdk 1.8 에서 도입된 Stream 활용 경험을 선택하였습니다.

Java8이 2014년 출시되고 이미 7년이 지났지만, 선택한 이유는 이번 프로젝트에서 각 지점별 통계를 구하는 과정에서 많은 도움을 받았기 때문입니다.

이미 인터넷 블로그나 웹사이트에 많은 자료가 있지만, 이번에 정리한 주제는 "**실제 프로젝트를 진행하면서 stream을 어떻게 활용했는가?**" 입니다.

2. 객체 클래스(VO) 사이의 연산

2개의 객체클래스가 있다고 하겠습니다. 하나는 사용자가 사용한 포인트정보클래스 객체이고, 다른 하나는 사용자가 주문한 주문정보객체 클래스입니다. 각각의 클래스는 공통적으로 해당 정보가 발생한 날짜에 대한 필드를 가지고 있습니다. 이 날짜를 기준으로 그룹핑하여 두 클래스를 하나의 클래스로 나타내어 프론트로 전달하려고 합니다.

우선, stream을 사용하지 않는 방법으로 코드를 작성하면 아래와 같은 형태가 될 것입니다.

```
PointHistory pointHistory = PointHistory.builder()
                                        .sumPoint(2000)
                                        .yy("20211209")
                                        .build();
OrderHistory orderHistory = OrderHistory.builder()
                                        .sumPayFeeSum(3000)
                                        .yy("20211209")
                                        .build();

if(pointHistory.getYy().equals(orderHistory.getYy())) {
    SalesSummary sales = SalesSummary.builder()
                                     .usePoint(pointHistory.getSumPoint())
                                     .sumSales(orderHistory.getSumPayFeeSum())
                                     .build();
}
```

하나의 데이터라면 이런 방법도 나쁘지 않고 간단해 보입니다. 하지만 각 객체가 List 형태로 이루어져있고, 객체의 필드 중 날짜를 의미하는 yy가 서로 일치하지 않는다면 굉장히 많은 조건문과 중복제거 등 개발자가 해야 하는 일은 많아지고 코드는 복잡해질 것 입니다.

달리 접근하여 우선 날짜별로 서로의 데이터를 그룹핑한 다음 그룹핑한 데이터를 Map 형태의 자료로 표현하고 Map을 데이터 클래스에 매핑시키는 방법도 있지만, 이 방법 또한 쉽지는 않고 코드가 굉장히 길어지게 될 것 입니다. 또한 그 과정에서 결과를 뽑아내기 위한 불필요한 임시클래스나 변수들이 많이 사용되어 코드의 가독성과 효율성을 떨어뜨리게 될 것입니다.

이럴 때 stream을 활용하면 객체클래스들을 날짜별로 그룹핑하여 합산하는 일을 손쉽게 할 수 있습니다.

3. Stream을 활용한 통계

Stream은 객체지향 언어로 표현되는 Java에서 함수형 프로그래밍을 활용할 수 있는 API입니다. 데이터를 추상화하고, 처리하는데 자주 사용되는 함수들이 정의되어 있고, 3가지 단계를 거칩니다.

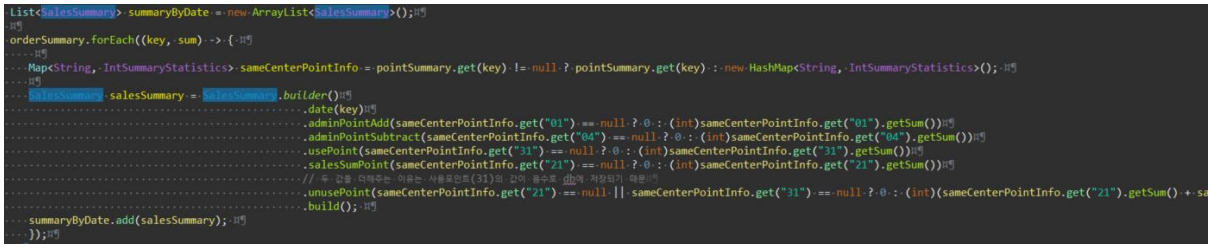
첫 번째로, stream을 생성한 다음,

두 번째로 데이터를 가공하는 중개 연산을 거치고,

마지막으로 결과를 만드는 최종 연산으로 이루어집니다.

Stream은 복잡한 연산을 간결하게 표현하고 활용하는 API이므로, stream 연산을 하더라도 원본의 데이터를 변경하지 않습니다. 또한 일회용으로 한 번 사용하면 재사용이 불가능합니다. Stream이 또 필요한 경우에는 다시 생성해 주어야 합니다. 그렇기 때문에, 메서드 체이닝형태로 연산을 수행합니다. 또한 내부에서 반복적으로 처리되는 작업을 코드 속에 숨겨 for나 while같은 반복문이 겉으로 드러나지 않도록 합니다.

이와 같은 stream의 특징을 활용하여 공통적인 필드를 가지는 서로 다른 두 객체클래스를 하나로 묶어서 표현할 수 있습니다.



위 코드를 보면, 포인트 리스트와 주문 리스트 두개의 DTO list를 기간 검색 옵션에 따라 월별 혹은 일별로 묶어 준 후 동일한 날짜의 데이터를 새로운 DTO List에 필드값으로 넣어 반환하는 코드입니다. 2개의 stream에서 사용된 collect 함수는 stream API에서 지원하는 최종연산 중 하나로 stream연산의 결과값을 수집하여 반환하는 역할을 합니다. collect안의 groupingBy 함수를 통해서 날짜를 통해 데이터를 그룹핑하고, summarizingInt함수는 stream을 반복하며 얻은 값들의 최대값, 최소값, 평균 값, 합계등을 정수형으로 연산하는 함수입니다.

이를 통해 얻은 결과값들을 Map 형태로 반환받습니다. 이 후에는 공통적으로 가지는 키값, 여기서는 동일한 날짜가 되겠습니다. 동일한 키값을 가진, 하나로 합쳐진 새로운 DTO의 필드로 설정해주기만 하면 됩니다. 마치 JSON 처럼요!

객체지향 프로그래밍은 굉장히 멋지고 좋은 프로그래밍 패러다임이지만, 때로는 아쉬운 부분도 있습니다. 그런 부분을 해결하는 데에 있어서 stream API는 아주 좋은 선택이 될 수도 있습니다.