2.Java Reflection

연구 목적

- 클래스 내 존재하는 getter 메소드를 동적으로 호출하기 위함

활용

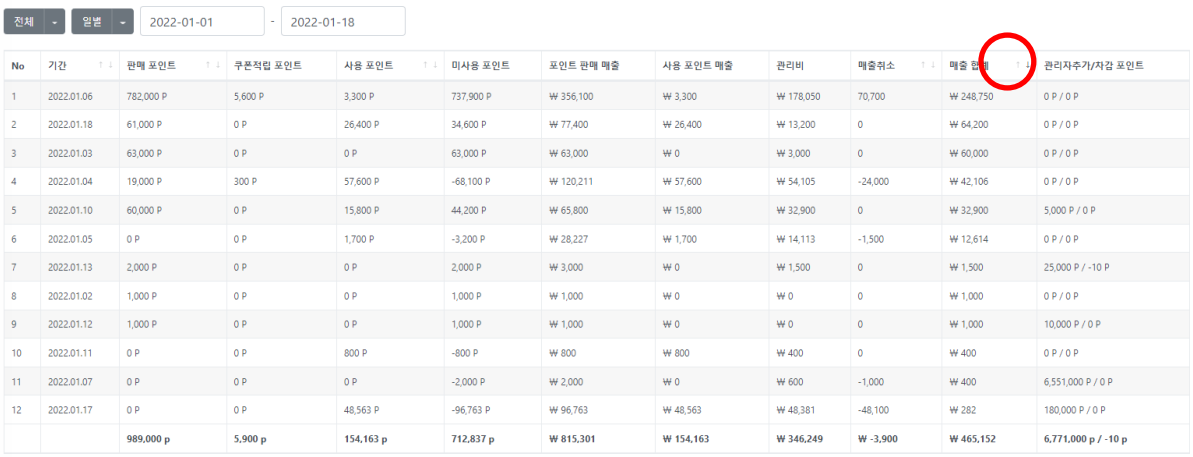
- DB로부터 전달받은 데이터를 담은 VO를 List형태로 만들고 List를 VO의 필드별로 정렬하기 위해 활용

```
@Setter
@Getter
public class SalesSummary implements Comparable<SalesSummary>{

    private int salesSumPoint;
    private int accCouponPoint;
    private int usePoint;
    private int unusePoint;
    private int sumSalesMoney;
    private int usePointMoney;
    private int cancelSales;
    private int totalSales;
    private int adminPointAdd;
    private int adminPointSubtract;
    private int manageFee;
    private int cancelOrderFee;
    private double valueOfSupply;
    private double vat;
    private String date;
    private String sortingKey;
```

Figure 1 정렬할 VO

- 화면 영역에서 버튼을 통해 올림차 순, 내림차 순으로 정렬하는 기능을 구현하는 과정에서 각 VO의 필드의 값을 얻어오기 위해 getMethod를 호출할 필요가 있음



전체 일별 2022-01-01 - 2022-01-18

No	기간	판매 포인트	쿠폰적립 포인트	사용 포인트	미사용 포인트	포인트 판매 매출	사용 포인트 매출	관리비	매출취소	매출 합계	관리자추가/차감 포인트
1	2022.01.06	782,000 P	5,600 P	3,300 P	737,900 P	₩ 356,100	₩ 3,300	₩ 178,050	70,700	₩ 248,750	0 P / 0 P
2	2022.01.18	61,000 P	0 P	26,400 P	34,600 P	₩ 77,400	₩ 26,400	₩ 13,200	0	₩ 64,200	0 P / 0 P
3	2022.01.03	63,000 P	0 P	0 P	63,000 P	₩ 63,000	₩ 0	₩ 3,000	0	₩ 60,000	0 P / 0 P
4	2022.01.04	19,000 P	300 P	57,600 P	-68,100 P	₩ 120,211	₩ 57,600	₩ 54,105	-24,000	₩ 42,106	0 P / 0 P
5	2022.01.10	60,000 P	0 P	15,800 P	44,200 P	₩ 65,800	₩ 15,800	₩ 32,900	0	₩ 32,900	5,000 P / 0 P
6	2022.01.05	0 P	0 P	1,700 P	-3,200 P	₩ 28,227	₩ 1,700	₩ 14,113	-1,500	₩ 12,614	0 P / 0 P
7	2022.01.13	2,000 P	0 P	0 P	2,000 P	₩ 3,000	₩ 0	₩ 1,500	0	₩ 1,500	25,000 P / -10 P
8	2022.01.02	1,000 P	0 P	0 P	1,000 P	₩ 1,000	₩ 0	₩ 0	0	₩ 1,000	0 P / 0 P
9	2022.01.12	1,000 P	0 P	0 P	1,000 P	₩ 1,000	₩ 0	₩ 0	0	₩ 1,000	10,000 P / 0 P
10	2022.01.11	0 P	0 P	800 P	-800 P	₩ 800	₩ 800	₩ 400	0	₩ 400	0 P / 0 P
11	2022.01.07	0 P	0 P	0 P	-2,000 P	₩ 2,000	₩ 0	₩ 600	-1,000	₩ 400	6,551,000 P / 0 P
12	2022.01.17	0 P	0 P	48,563 P	-96,763 P	₩ 96,763	₩ 48,563	₩ 48,381	-48,100	₩ 282	180,000 P / 0 P
		989,000 p	**5,900 p**	**154,163 p**	**712,837 p**	**₩ 815,301**	**₩ 154,163**	**₩ 346,249**	**₩ -3,900**	**₩ 465,152**	**6,771,000 p / -10 p**

Figure 2 구현화면

- Figure 2와 같이 사용자가 컬럼별로 버튼을 클릭함에 따라 정렬이 되어야 함 -> java runtime 중에 동적으로 getMethod가 호출되어야 함
- Switch 문으로 구현할 경우, VO의 필드가 많아질 경우, 케이스가 많아져 코드를 유지보수하기 힘들어지고 가독성이 떨어짐 -> Reflection의 필요성
-

```java
@Override
public int compareTo(SalesSummary o) {
    // 정렬 조건 없을 때 기간순 정렬
    if(getSortingKey().equals("") || getSortingKey().equals("RegDt")) {

        SimpleDateFormat format = new SimpleDateFormat( pattern: "yyyyMMdd");

        if( getDate().length() == 2) {
            format = new SimpleDateFormat( pattern: "MM");
        }

        try {
            Date date1 = format.parse(getDate());
            Date date2 = format.parse(o.getDate());

            return date1.compareTo(date2);

        } catch (ParseException e) {
            e.printStackTrace();
        }
    }
    else {
        Class thisClass = this.getClass();
        String methodName = "get" + getSortingKey();
        Method m = null;
        Method m2 = null;

        try {
            m = thisClass.getMethod(methodName, ...parameterTypes: null);
            m2 = o.getClass().getMethod(methodName, ...parameterTypes: null);
        } catch (NoSuchMethodException e) {
            e.printStackTrace();
        }

        try {
            int result = Integer.compare((Integer) m.invoke( obj: this), (Integer) m2.invoke(o));
            return result;
```

Figure 3 VO 비교를 위한 compareTo 함수

- VO 내부에 Compareable 인터페이스의 compareTo함수를 오버라이딩하여 클래스의 정보를 받아와서 전달받은 정렬 컬럼의 키값을 기준으로 compareTo 함수 재정의
- VO 의 자료형은 String과 Int, double이 존재하고, 그 중 date의 경우 일반적인 수의 비교와 달리 날짜비교가 필요하므로 Date로 형변환한 후 비교해야 하므로 if문으로 분기를 나누어 처리.
- 자기 자신의 클래스 정보를 얻어와서 동적으로 getMethod를 호출하도록 구현

- reflection으로 얻어온 값을 비교하여 compareTo 함수 구현

```
Collections.sort(summaryByDate, Collections.reverseOrder());

// 컬럼과 정렬방향(asc,desc)에 따라 정렬
if (sortingDir.equals("asc")) {
    Collections.sort(summaryByDate);
} else if (sortingDir.equals("desc")) {
    Collections.sort(summaryByDate, Collections.reverseOrder());
}
```

Figure 4 올림차순과 내림차순에 따라 분기 처리

- Collection의 sort함수를 통해 정렬 방법(올림차,내림차)에 따라 VO 객체 필드의 값들을 비교하여 정렬이 가능하도록 구현

결론

- 로직의 분기가 많은 경우 reflection을 통해 동적으로 값을 얻어오게 되면 이후 정렬할 컬럼이 늘어나더라도 VO의 필드만 추가하면 동적으로 필드의 값을 얻어오도록 동작할 것이므로 개발자가 신경쓸 여지가 적음.
- VO 객체 내부에 핵심로직이 존재하므로 객체지향적인 코드 작성 가능