

객체 지향 프로그래밍 팀플 보고서

<편의점 POS>

12조

20205602 박정현
20186756 박현진
20206147 서은서
20183611 최수아
20213865 황인영

1. 실제 편의점 POS를 코드로 구현한다?

편의점 POS의 기저에 놓인 핵심은, "손님과 편의점의 상호작용"을 가능하게 하는 것이다. 손님의 요구를 편의점에 전달하고, 편의점의 동작을 손님에게 전달하는 효과적인 방법에 맞춰 프로젝트를 진행하였다.

실제 편의점 POS의 기능을 분석하여 "판매 / 환불 / 재고관리 / 회계관리" 크게 4가지 기능을 수행하는 POS로 재구성 하였다. 해당 기능들은 편의점 내부의 "상품"을 통해 서로 소통한다. 상품의 변동이 연쇄적으로 각 기능의 변동을 이끌어내고, 각각의 기능들은 또 다시 서로 영향을 주고 받으며 편의점을 구성한다.

매우 복잡하게 얹혀있는 각 기능의 상호작용을 효율적으로 관리할 수 있도록 설계의 방향성을 잡았고 설계 과정에서 고려한 주요 쟁점은 다음과 같다.

- 1) OOP적 관점을 토대로 각 클래스를 어떻게 분업하는가?
- 2) 상품을 어떻게 추상화해야 하는가?
- 3) 상품을 어떻게 저장하여 관리하는가?
- 4) 핵심 기능이라 할 수 있는 판매 상황에서, 손님에게 어떤 식으로 여러 개의 구매 상품 목록을 받아오는가?

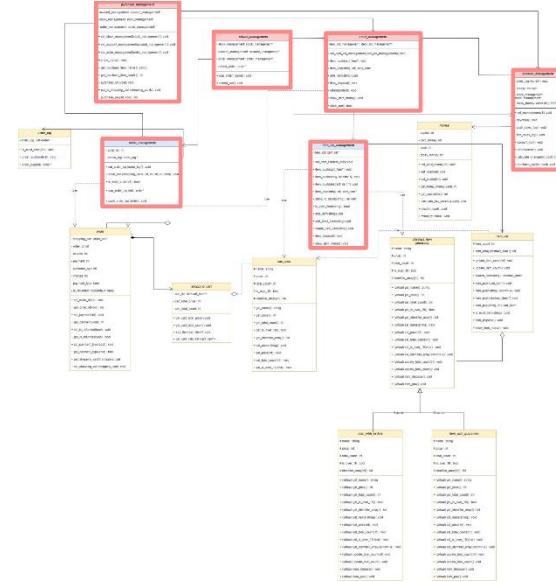
다음 단락은, 이 질문들에 대한 답을 중심으로 POS 설계 구조와 과정을 설명한다.

2. 설계 소개

1) OOP 적 관점을 토대로 각 클래스를 어떻게 분업하는가?

구매 (Purchase) / 주문(order) / 환불 (Refund) / 회계 (Account) / 재고 (Stock) / 상품 (Item) 6가지에 대해 management를 두어 각각을 담당하여 관리하도록 하고, 상호 작용에 있어서도 이 5가지 management를 통해서만 정보를 주고 받도록 설계하였다. 이 방식을 택함으로써 다음과 같은 이점을 얻을 수 있다.

- **Encapsulation:** 위의 변경이 빈번한 5가지에 대해서는 각각의 management들만이 접근이 가능하게 하여, 임의로 변경하지 못하는 부분 뒤에 구체적인 기능을 숨길 수 있도록 하였다. 이를 통해 서로 주고 받는 명령을 줄일 수 있고, 클래스 간의 복잡도를 줄일 수 있게 된다. 이는 이를 통해 encapsulation을 구현할 수 있게 되었다.
- **Single Responsibility:** 각각을 management 클래스로 나눔으로써, 한 클래스가 하나의 책임을 가질 수 있다. 예를 들어 purchase_management를 통해서는 손님과의 상호 작용의 역할만을 수행할 수 있다. 이후 재고의 변동은 stock_management가, 자본의 변동은 account_management만이 관여 할 수 있다. 즉 구매로 일어나는 POS의 변동을 각 management들이 독립적으로 맡아서 수행할 수 있게 되는 것이다. 업무의 수행은 담당 management를 중심으로 이루어지며, 다른 class들은 정보를 넘기기만 하면 된다.



2) 상품을 어떻게 추상화해야 하는가?

모든 상품이 동일한 정보 구조를 가지고 있지 않으므로, 모든 상품의 공통점을 토대로 추상화하여 큰 틀로 두고 이를 상속 받는 하위 클래스를 두는 구조를 이용하기로 했다. 우리 설계에서는, 재고 관리를 하는 상황에서 유통 기한의 유무가 상품을 다루는 데에 있어 크게 좌우될 것을 고려하였다.

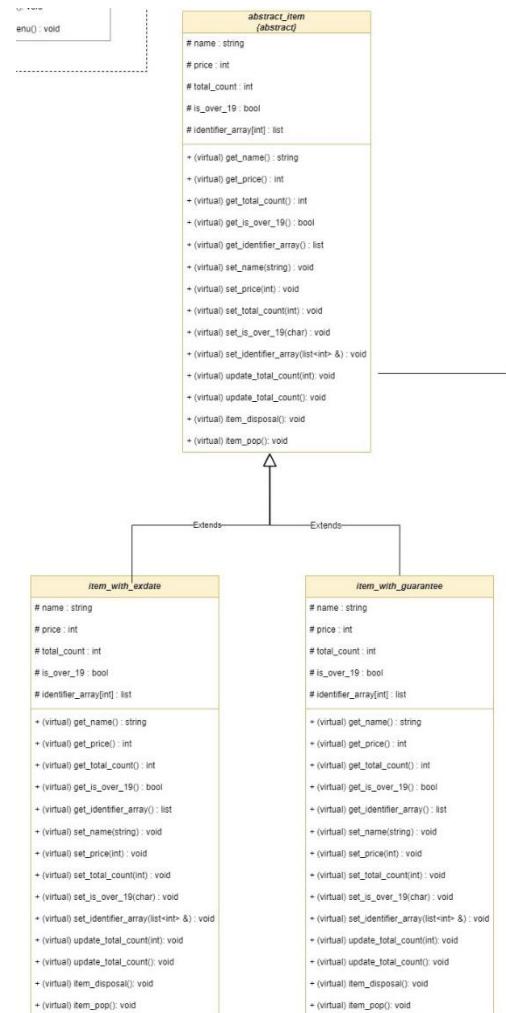
오른쪽 그림과 같이 item 전체를 가리키는 추상클래스인 abstract_item를 상위에 두고, 유통기한이 있는 상품 (item_with_exdate)과 없는 상품

(item_with_guarantee)이 item 추상클래스를 상속받도록 하여, 개념적인 상속이 가능하도록 하였다. 이때

Item_with_guarantee는, 유통기한 대신 품질 보증 기간을 두어, 이 기간 이후로는 품질이 떨어진다고 가정하였다. 이를 통해 얻게 된 장점 중 하나로는 다음과 같다.

- **Open-closed principle** : Abstract_item을 상속받는 subclass들은 type에 종속 받지 않게 되어, 다른 class에 해당함에도 동일한 행동을 이끌어낼 수 있게 된다. 예를 들어

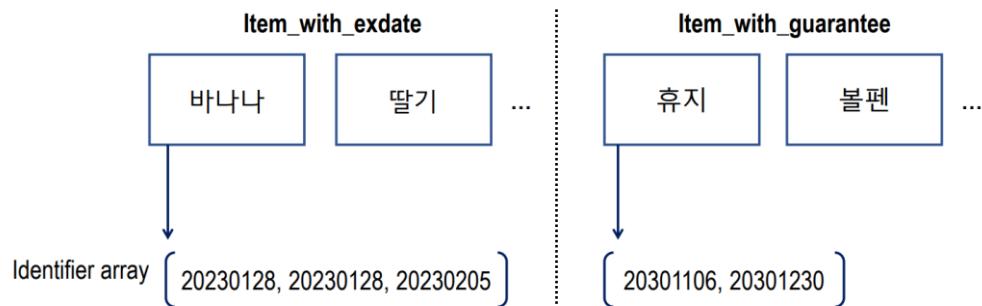
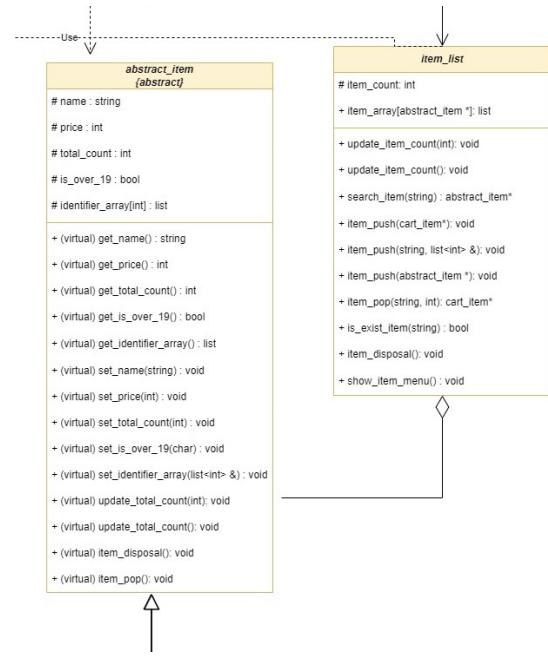
유통기한이 있는 상품/없는 상품 외의 다른 정보 구조를 가진 item을 추가하고 싶은 경우, 추상 클래스인 abstract_item을 상속받게 한다면 코드상의 큰 수정 없이 item_list를 통해 관리될 수 있다. 즉, 확장, 추가에 개방적인 구조를 가지게 된다.



3) 상품을 어떻게 관리하고 저장하는가?

3-1) 상품의 관리

만들어 낸 item_with_exdate / item_with_guarantee 클래스로 찍어낸 각각의 객체들은 하나의 상품목이 되며, 이 객체에는 멤버 변수로 identifier_array가 존재한다. 이 array에는 유통기한 및 품질 보증 기한이 담기고, 담긴 유통기한 및 품질 보증 기한 정보 원소 하나는 실제 상품 하나를 표상하게 된다. 이를 모식도로 표현하면 아래와 같다.



- ▲ 바나나, 딸기는 item_with_exdate로 찍어낸 객체이고, 휴지, 소주는 item_with_guarantee로 찍어낸 객체이다. Identifier_array는 그 객체 안에 있는 멤버 변수를 의미한다.

위의 예시에서는, 바나나 객체 내의 identifier_array에는 20230128, 20230128, 20230205 세 개의 날짜가 저장되어 있다. 이는 바나나라는 품목은 유통 기한이 2023년 1월 28일인 것 2개, 2023년 2월 5일인 것 1개, 총 3개가 재고로 저장되어 있다는 것을 의미한다. 마찬가지로, 휴지 객체 내에 있는 identifier_array를 보면 보증기한이 2030년 11월 06일인 휴지 1개, 2030년 12월 30일까지인 휴지가 1개씩, 총 2개가 존재하는 것을 알 수 있다.

정리하자면, 클래스로 찍어낸 객체는 품목이 되고, 각 상품의 종류 내에 identifier_array는 각 멤버변수 안에 있는 배열의 원소 하나하나는 각각 실제 상품을 의

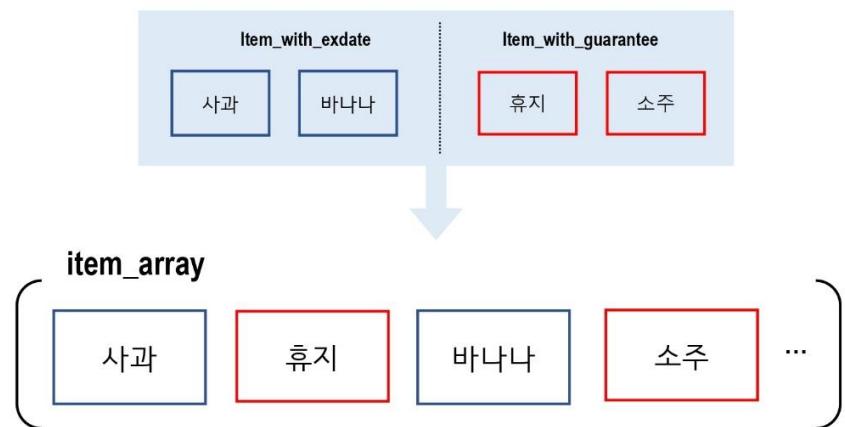
미하게 된다.

이러한 방식을 선택함으로써 우리 설계가 취한 주요 이점은 다음과 같다.

- **효율적인 관리** : 유통기한과 보증기한이 지난 상품을 폐기하는 상황, 상품을 입고 하는 상황, 재고의 현황을 확인해야 하는 상황 등에서 모든 상품을 검색하지 않고 상품 항목에 따라 조회할 수 있다. 따라서 시간을 단축할 수 있으며 체계적이고 효율적인 관리가 가능해진다.

3-2) 상품의 저장

상품의 저장은 검색에 있어서 용이하게 하기 위해, Item_with_exdate 객체와 item_with_guarantee 객체를 구분 없이 한 컨테이너에 두기로 했다. 이는 2-1)에서와 같이, item_with_exdate와 item_with_guarantee가 그 상위의 추상 클래스인 abstract_item을 공통으로 상속받는 클래스여서 가능해졌다.



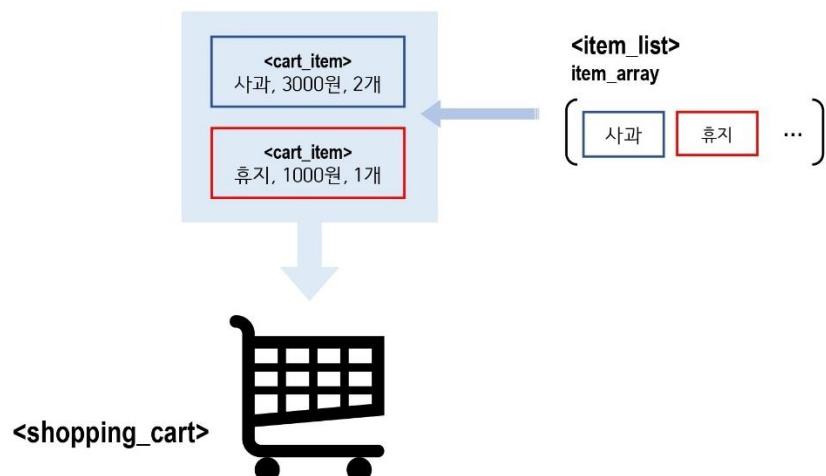
이때 단순히 Array 형식을 사용하기보다 STL 컨테이너인 List를 사용한 이유는 사이즈가 고정적이면서도 메모리와 성능의 최적화할 수 있기 때문이다. 편의점과 같은 작은 상점은 재고를 보관할 수 있는 공간이 한정되어 있고 상품의 총 종류와 그 개수가 크게 가변적이지 않기 때문에 List를 사용하는 게 더 적절할 것이라 판단하였다.

또, 품절된 상품이 재입고되는 경우가 빈번한 편의점 특성을 고려하여, 한 상품 항목이 품절되면, 해당 상품 항목의 객체를 Delete 하는 것이 아니라 그 객체의 멤버 변수 중 total_count (해당 상품 항목에서 유통기한/품질 보증 기한으로 구분되는 물건 개수의 합계)을 0으로 두는 것으로 설정하였다.

4) 핵심 기능이라 할 수 있는 판매 상황에서, 손님에게서 어떤 방식으로 여러 개의 구매 상품 목록을 받아오는가?

이는 shopping_cart (장바구니)와 cart_item () 클래스를 따로 분리해 둠으로써 해결하였다.

한 번에 상품명과 개수를 입력 받아 상품 정보를 받아온다. 이 상품에 대한 정보의 값을 다른 클래스에 넘겨주고, item_list 내부에 있는 item_array 정보를 수정한 후 상품의 정보를 바탕으로 한 cart_item 자료형 객체를 받아온다. 이 cart_item은 shopping_cart에 담긴다. 이 과정을 반복함으로써, 한 shopping_cart 객체에 여러 개의 cart_item, 즉 손님이 선택한 여러 종류의 상품이 담긴다. 따라서 상품 항목의 개수 수정이 일어나는 것은 shopping_cart에 담을 때마다 일어난다. 이 shopping_cart를 통해 총 금액을 구하고 결제에 이용될 수 있게 하여, 한 손님이 여러 개의 상품을 한번에 구매할 수 있도록 한다.



이러한 설계로 다음과 같은 이점을 얻을 수 있다.

- **의존성 낮춤** : item 객체를 직접 shopping_cart에 담는 것이 아닌, cart_item 자료형으로 값을 복사해서 사용하는 이유는 item 클래스에 정의되어 있는 모든 멤버 함수들을 구매 과정에서 사용하게 되면, item 클래스에 대한 의존성이 높아지기 때문이다. 이를 해소하기 위해 cart_item을 두어 필요한 정보만을 복사해 오도록 하였다. 또, item 객체는 특정 management를 통해서만 접근이 가능하게 하여 캡슐화가 더 안정적으로 지켜질 수 있도록 하였다.

3. 기능 소개

다음으로 편의점에서 일어날 수 있는 상황을 가정하고, 그 상황을 바탕으로 POS의 기능과 클래스들의 상호 작용이 일어나는 순서를 설명하겠다. 상황은 다음과 같이 크게 두 분류로 나눌 수 있다.

1. 손님과 POS의 상호작용: 손님의 편의점에 대한 용건과 관련된 상황 (구매, 환불)
2. 편의점과 POS의 상호작용: 전체적인 편의점 관리와 관련된 상황 (재고 관리, 회계 관리)

후술할 내용은 각 코드의 기능에 관한 구체적인 설명이며, 각 구현은 주석에 상세히 설명되어 있다.

1-1. 손님이 상품을 구매하는 상황

- 1) purchase_mangement를 통해 먼저 손님에게서 5가지의 정보를 받아온다.: `age` (나이), `payment` (지불 수단, 현금 또는 카드), `coupon` (할인되는 만큼의 가격), `shopping cart` (item 클래스에 있는 멤버 변수를 토대로 구매할 상품의 정보를 cart_item 자료형으로 받아와 shopping_cart에 List로 담고, 이를 통해 `total_order_price`를 계산함), `paid_money` (손님이 지불한 금액)

※ shopping_cart에 cart_item을 담는 과정: For 문을 통해 손님으로부터 (상품 정보, 개수)를 여러 번 입력 받고, 입력 받을 때마다 그 정보를 stock_management에 넘겨 준다. stock_management는 item_list_management에 접근하여 유통 기한/품질 보증 기한이 가장 적게 남은 item_array의 원소를 삭제하고, purchase_management에게 cart_item을 반환한다. 받아온 cart_item은 shopping_cart에 담긴다.

!!! 기능 제한 : shopping_cart에 담긴 물건은 수정될 수 없다.

- 2) payment_type, 즉 지불 수단이 현금이면 calculate_change()를 통해 change (거스름 돈)를 계산한다.

- 3) 구매로 인해 (1) 구매 기록의 추가, (2) 재고의 감소와 (3) POS기 회계 상의 변경이 일어난다.

3-1) 구매 기록의 추가

purchase_management 클래스에서 cart_item이 담긴 shopping_cart 객체와 손님이 입력한 정보를 order_management 클래스에 보낸다. order_management 클래스에서는 받은 값들을 토대로 order 클래스의 멤버 변수 값을 채워 넣어 order를 생성하고, 생성된 order를 order_log의 log_list에 추가한다. log_list에는 이러한 order 객체들이 들어 있어, 이전 구매 정보에 대한 내용들을 저장할 수 있다. 저장한 구매 기록을 통해 영수증 출력, 환불과 같은 기능을 수행할 수 있다.

3-2) 재고의 변경

1-1-1)에서 설명한 바와 같이, 상품 정보를 입력 받을 때마다 purchase_management는 stock_management에 정보를 넘겨주고, stock_management에서는 item_list_management에 접근하여 유통 기한/품질 보증 기한이 가장 적게 남은 item_array의 원소를 삭제한 다음, purchase_management에 cart_item을 반환한다. 이 cart_item은 입력 받을 때마다 하나의 shopping_cart 객체에 반복하여 담긴다.

3-3) POS기 회계 상의 변경

purchase_pay()함수를 통해 account_management에 접근하여 결제 방식에 따라 매출을 증가시킨다.

1-2. 손님이 영수증, 물건을 들고 환불을 요청하러 온 상황

여기서 영수증은, order의 멤버 변수 값들로 구성되어 있다.

- 1) 손님이 준 영수증으로부터, refund_management 클래스를 통해 '영수증'에 해당하는 order의 order_id (영수증 번호)를 읽어온다.
- 2) order_management에 읽어온 영수증 번호를 넘겨준다.

- 3) order_management 클래스는 pop_order_log()를 통해 order_log 클래스에 접근한다. 이를 통해 log_array에 있는 order의 order_id 값을 검색하고, 해당 order를 지운다.
- 4) Search 하여 찾은 order 객체를 order_management 클래스의 pop_order_log()를 통해 refund_purchase()로 넘기고, 해당 order 기록을 삭제한다.
- 5) 이 넘겨진 정보를 바탕으로 (1) 재고의 변경, (2) POS기 회계 상의 변경이 있다.

5-1) 재고의 변경

refund_management 클래스에서 stock_management로 해당 order를 넘겨준다. stock_management는 order가 가지고 있는 shopping_cart의 cart_item 객체 정보를 item_list_management에 전달하여, item_push()를 실행한다. 이를 통해 item_list에 접근하여 identifier_array에 item의 유통기한 / 품질 보증 기한을 추가하고, 해당 item 객체의 count를 증가시킨다.

5-2) POS기 회계 상의 변경

refund_management에서 account_management에 접근하고, account_management에서 다시 money에 접근하여 해당하는 환불가 만큼의 금액을 사용되었던 결제 방식에서 감소시킨다. (card_money : 카드 / cash : 현금)

!!! 기능 제한 : 부분 환불은 불가하므로, 이를 원한다면 전체 환불 후 재결제 하는 방식을 이용해야 한다.

2-1. 점원이 포스기를 통해 재고를 추가하는 상황

- 1) 추가할 상품의 이름을 입력 받고, add_item(string)을 통해 item_list_management 클래스에 접근하여 search(string)를 실행한다.
→ 상품이 item_list 내부에 존재하는 객체이면 add_exist_item(string, abstract_item&)을 실행하여 기존에 존재하는 객체로부터 정보를 얻어와, 유통 기한/상품 보증 기한을 입력하여 추가한다.

- 상품이 item_list 내부에 존재하지 않는 객체이면 create_new_item(string name)을 실행하여 item 정보를 입력하고 객체를 새로 생성한다.

2-2 알바생이 마감 정산을 하는 상황

- 1) 마감 정산의 경우, account_management 클래스의 account_run()을 통해 dayclose()를 실행한다. 이 함수를 통해 (1)회계 상의 변화 (2) sales_log에의 추가 (3)유통 기한 지난 상품 폐기와 같은 변동이 있다.

1-1) 회계 상의 변화

account_management 클래스에서 money 클래스에 접근하여, card_money와 cash를 더해 하루 매출액을 계산하고, 이를 capital에 더하여 자본금 정보를 업데이트 한다. sales_log에 하루 매출을 날짜와 추가한 후, 초기화를 진행한다. 카드 매출의 경우, money class의 set_card_money(int)를 이용해 카드 매출을 0원으로 초기화 한다. 현금 매출의 경우, money class의 set_cash(int)를 이용해 현금 매출을 영업 준비금인 10만 원으로 초기화 한다. 영업 준비금인 10만 원은 자본에서 빼 와서 설정하는 것이므로, money class의 get_capital()에서 10만 원인 ready_money을 제한다. 이 결과 값 set_capital(int)에 넣어 자본금을 업데이트 한다.

1-2) sales_log의 추가

오늘 날짜와 함께 card_money와 cash를 더해 계산된 하루 매출액을 push_sales_log()를 통해 sales_log에 추가한다. sales_log는 Map Container로, 날짜를 Key, 하루 매출액을 Value로 갖는다. Map을 선택한 이유는 sales_log에 추가하는 건 하루에 한 번뿐이므로 중복되는 Key가 없고, 날짜를 통해 검색하기 용이하기 때문이다.

1-3) 유통기한 지난 상품 폐기

stock_management class의 item_disposal()을 이용하여, 유통 기한이 지난 상품들을 제거한다.

4) 결론 및 느낀 점

편의점 POS의 기능 분석에서 시작하여 설계 및 개발까지 완료하였다. 추상화, 캡슐화, 상속, 다형성, 오버로딩, UML, 디자인패턴, STL 등 수업시간에 배운 OOP의 핵심적인 기능 및 컨셉을 최대한 녹여내고자 노력했고 만족스러운 결과물을 얻을 수 있었다.

조금 더 충분한 시간이 있었다면 에러 처리 및 섬세한 기능까지 구현을 할 수 있었을 것이라 생각이 되어 약간의 아쉬움이 남는다. 하지만 해당 프로젝트를 진행하며 큰 규모의 프로젝트를 어떻게 하면 효율적으로 진행하고 협업할 수 있는가에 대해 직접 느끼고 체험할 수 있는 좋은 기회가 되었다. 특히 협업 과정에서 서로의 의견을 공유하고 자신의 생각을 이해시키는 커뮤니케이션의 능력이 매우 중요하며 생각보다 쉽지 않다는 것을 몸소 체험할 수 있었다.

팀원들간의 시간적 조율이 이루어 진다면, 해당 프로젝트를 발전시켜 더 완성도 높은 편의점 POS기를 만들어보고자 한다.

POS 시스템 구현 결과

현재 코드의 개선이 필요한 점

현재 만든 Main 함수 및 각 Management에서 실행하는 Run 함수는 사용자가 잘못된 입력을 하지 않는다는 전제를 하고 만들었다. 따라서 별도의 오류 처리가 되어 있지 않으며, 입력이 잘못되었을 경우 시스템이 강제 종료될 가능성이 있다. 현재는 사용자로부터 입력을 받을 때, 옆에 입력 예시를 적어 놓아 오류의 가능성을 최소화할 수 있도록 하였다.

[MAIN]

```
POS 시스템을 시작합니다.  
오늘의 날짜는 20221207 입니다.  
무엇을 하시겠습니까?  
===== POS 목록 =====  
1. 재고 관리 2. 구매  
3. 환불 4. 회계  
5. POS 시스템 종료(모든 정보 초기화)  
=====  
입력 (입력 예시: 1) : 1
```

POS 시스템을 테스트하기 위해서는 [MAIN]에서 1. 재고 관리를 가장 먼저 실행해야 한다. 실행 초기의 POS에는 판매할 상품이 아무것도 입고되지 않았기 때문이다. 따라서 재고 관리를 실행해 재고를 추가한 후 상황에 따라 2. 구매, 3. 환불, 4. 회계를 실행한다.

[재고 관리]

```
재고를 추가하시겠습니까? (Y / N) y  
===== 재고 추가 시작 =====  
추가할 상품의 이름을 입력해주세요. (입력 예시: banana, 더이상 추가할 상품이 없을 경우 0입력) 우유  
우유는(는) 19세 미만 판매금지 상품입니다? (Y / N) n  
우유의 가격을 입력하세요. (입력 예시: 2000) 1000  
우유의 개수를 입력하세요. (입력 예시: 3) 5  
우유의 고유 정보는 무엇입니까? (1. 유통 기한 2. 보증 기한) (입력 예시: 1) 1  
우유 5개의 유통 기한을 입력하세요. (유통 기한이 같을 경우에도 각각 입력)  
(입력 예시: 20221219 20221219 20221224)(입력 후 Enter)  
20221213 20221213 20221213 20221213 20221213  
상품이 추가되었습니다.  
===== 재고 추가 완료 =====  
  
===== 재고 추가 시작 =====  
추가할 상품의 이름을 입력해주세요. (입력 예시: banana, 더이상 추가할 상품이 없을 경우 0입력) 김  
김은(는) 19세 미만 판매금지 상품입니다? (Y / N) n  
김의 가격을 입력하세요. (입력 예시: 2000) 500  
김의 개수를 입력하세요. (입력 예시: 3) 3  
김의 고유 정보는 무엇입니까? (1. 유통 기한 2. 보증 기한) (입력 예시: 1) 1  
김 3개의 유통 기한을 입력하세요. (유통 기한이 같을 경우에도 각각 입력)  
(입력 예시: 20221219 20221219 20221224)(입력 후 Enter)  
20230430 20230430 20230430  
상품이 추가되었습니다.  
===== 재고 추가 완료 =====  
  
===== 재고 추가 시작 =====  
추가할 상품의 이름을 입력해주세요. (입력 예시: banana, 더이상 추가할 상품이 없을 경우 0입력) 0  
재고를 추가하시겠습니까? (Y / N) n  
재고를 추가하지 않습니다.  
===== 재고 추가 취소 =====
```

- 상품의 이름을 문자로 입력한다.
- Y 또는 N으로 답한다. (대소문자는 상관 없다.)
- 가격과 상품의 개수는 숫자이다.
- 고유 정보는 (1) 유통 기한이 있는 상품인지 (2) 품질 보증 기한이 있는 상품인지 묻는 항목이다. 1 또는 2로 답한다.
- 유통 기한 혹은 보증 기한을 입력한 상품의 개수만큼 적어준다. 상품과 상품은 띄어쓰기로 구분된다.
- 재고 관리를 종료하고 싶을 때는 “추가할 상품의 이름을 입력해주세요. (입력 예시: banana, 더 이상 추가할 상품이 없을 경우 0 입력)”에서 0을 입력한다.
- 0 입력 후 “재고를 추가하시겠습니까? (Y / N)”에서 N을 입력하면 종료된 후 [MAIN]으로 돌아 간다.

[구매]

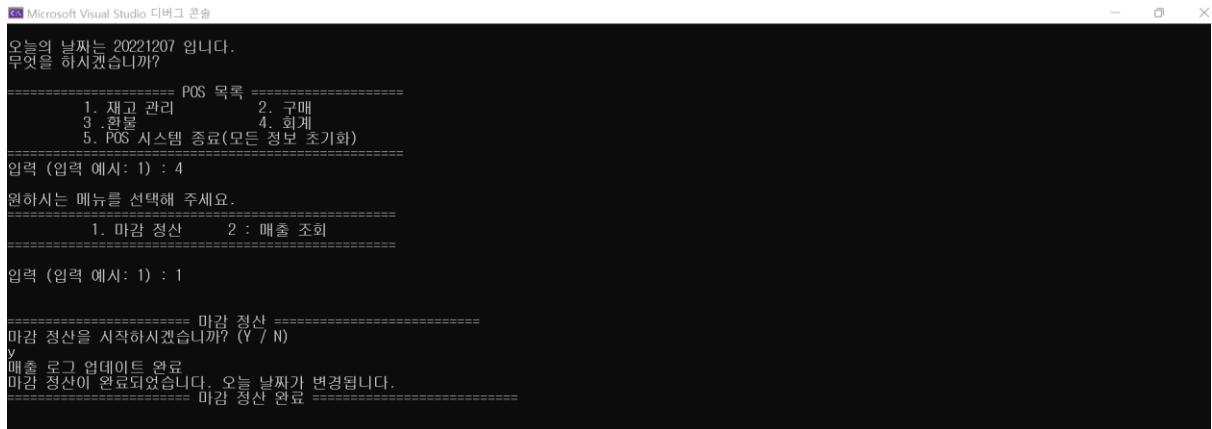
POS의 재고관리를 통해 직원이 상품을 입고하면 손님은 구매가 가능해진다. [MAIN]에서 2를 입력한다.

```
무엇을 하시겠습니까?  
===== POS 목록 =====  
1. 재고 관리 2. 구매  
3. 환불 4. 회계  
5. POS 시스템 종료(모든 정보 초기화)  
입력 (입력 예시: 1) : 2  
  
===== 상품 목록 =====  
1. 우유 가격: 1000 2. 껌 가격: 500  
3. 우유 수량: 5 4. 껌 수량: 3  
  
===== 구매 시작 =====  
손님의 나이를 입력해주세요: (입력 예시 : 20) : 20  
위의 상품 목록을 참고하여 구매할 상품을 하나씩 입력해주세요.  
(입력 예시: banana, 없다면 0입력)(한 번에 한 종류의 상품만 입력하세요.) : 우유  
구매할 상품의 개수를 입력하세요. (입력 예시: 1) : 1  
쇼핑카트에 상품이 담겼습니다.  
  
위의 상품 목록을 참고하여 구매할 상품을 하나씩 입력해주세요.  
(입력 예시: banana, 없다면 0입력)(한 번에 한 종류의 상품만 입력하세요.) : 껌  
장바구니에 구매한 상품이 모두 추가되었습니다.  
결제를 시작합니다.  
1. 쿠폰이 있다면 할인기를 입력하세요 (입력 예시 : 500, 없다면 0입력) : 0  
총 1500원입니다.  
결제 방식을 선택해주세요 (0: 카드 1: 현금) : 0  
1500원 결제 완료.  
===== 구매 완료 =====
```

- 현재 편의점에 있는 상품의 목록과 가격, 개수가 출력된다.
- 나이를 입력한다. 나이는 반드시 숫자이다.
- “위의 상품 목록을 참고하여 구매할 상품을 하나씩 입력해주세요. (입력 예시: banana, 없다면 0입력)(한 번에 한 종류의 상품만 입력하세요.) :” 에 구매하고자 하는 품목을 입력한다. 목록에 없는 품목은 입력하지 않는다.
- “구매할 상품의 개수를 입력하세요. (입력 예시: 1) :” 에 구매하고자 하는 개수를 입력한다. 이는 1 이상의 숫자이다.
- 쇼핑 카트에 상품이 담기며, 위의 작업을 반복하여 상품을 추가할 수 있다.
- “위의 상품 목록을 참고하여 구매할 상품을 하나씩 입력해주세요.”에서 0을 입력하면 카트에 상품을 담는 작업이 종료된 후 결제가 시작된다.
- 쿠폰을 입력한다. 반드시 숫자이다. 없다면 0을 입력한다.
- 결제 방식은 반드시 0 (카드) 또는 1 (현금)을 입력한다.
- 결제가 완료되면 종료된 후 [MAIN]으로 돌아 간다.

[회계]

회계는 마감 정산 및 원하는 날짜의 매출 기록 열람이 가능하다. 상품을 입고하고 손님에게 판매 했으니 하루 매출을 정산한다. [MAIN]에서 4를 입력한다.



```
Microsoft Visual Studio 디버그 콘솔
오늘의 날짜는 20221207 입니다.
무엇을 하시겠습니까?

===== POS 목록 =====
1. 재고 관리 2. 구매
3. 환불 4. 회계
5. POS 시스템 종료(모든 정보 초기화)

입력 (입력 예시: 1) : 4

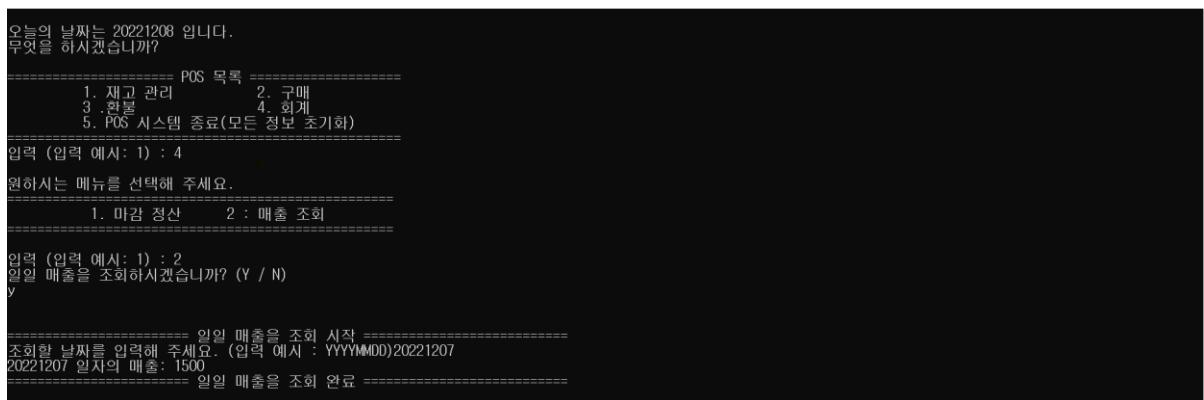
원하시는 메뉴를 선택해 주세요.

===== 1. 마감 정산 2 : 매출 조회 =====

입력 (입력 예시: 1) : 1

===== 마감 정산 =====
마감 정산을 시작하시겠습니까? (Y / N)
y
매출 로그 업데이트 완료
마감 정산이 완료되었습니다. 오늘 날짜가 변경됩니다.
===== 마감 정산 완료 =====
```

- 마감 정산 후 매출이 한 개 이상 기록되어야 매출 조회가 가능하다. 따라서 1. 마감 정산 을 실행해야 한다. 1을 입력한다.
- “마감 정산을 시작하시겠습니까? (Y / N)”에 Y를 입력하면 마감 정산이 시작되고 완료된다.
- 마감 정산이 완료되는 순간 자동으로 날짜가 다음 날로 변경되고, 자동으로 종료된 후 [MAIN]으로 돌아 간다.



```
오늘의 날짜는 20221208 입니다.
무엇을 하시겠습니까?

===== POS 목록 =====
1. 재고 관리 2. 구매
3. 환불 4. 회계
5. POS 시스템 종료(모든 정보 초기화)

입력 (입력 예시: 1) : 4

원하시는 메뉴를 선택해 주세요.

===== 1. 마감 정산 2 : 매출 조회 =====

입력 (입력 예시: 1) : 2
일일 매출을 조회하시겠습니까? (Y / N)
y

===== 일일 매출을 조회 시작 =====
조회할 날짜를 입력해 주세요. (입력 예시 : YYYYMMDD) 20221207
20221207 일자의 매출: 1500
===== 일일 매출을 조회 완료 =====
```

- MAIN에서 4를 선택한 후, 2를 입력한다.
- 일일 매출 조회를 할 수 있다. Y를 입력 후 20221207을 입력하자 해당 일자의 매출이 화면에 출력된다.

[환불]

이전에 상품을 구매했던 손님이 영수증을 가지고 환불을 하려 오면 [MAIN]에서 3을 입력한다.

- “환불을 진행하시겠습니까? (부분 환불은 가능하지 않습니다.) (Y / N)”에서 Y를 입력하면 환불이 시작된다.
- 손님은 우리 편의점의 0번째 구매 손님이었고, 영수증 번호는 0이다. 따라서 영수증 번호 0을 입력한다.
- 환불이 완료되면 종료된 후 [MAIN]으로 돌아 간다.

[POS 시스템 종료]

[MAIN]에서 5를 입력하면 POS 시스템이 종료된다.

```
오늘의 날짜는 20221208 입니다.  
무엇을 하시겠습니까?  
===== POS 목록 =====  
1. 재고 관리 2. 구매  
3. 환불 4. 회계  
5. POS 시스템 종료(모든 정보 초기화)  
=====  
입력 (입력 예시: 1) : 3  
환불을 진행하시겠습니까? (부분 환불은 가능하지 않습니다.) (Y / N)y  
===== 환불 시작 =====  
환불 영수증 번호를 입력해 주세요. (입력 예시: 0)0  
내역 조회에 성공했습니다. 환불을 진행합니다.  
환불을 성공했습니다.  
===== 환불 종료 =====  
  
오늘의 날짜는 20221208 입니다.  
무엇을 하시겠습니까?  
===== POS 목록 =====  
1. 재고 관리 2. 구매  
3. 환불 4. 회계  
5. POS 시스템 종료(모든 정보 초기화)  
=====  
입력 (입력 예시: 1) : 5  
POS 시스템을 종료합니다.  
C:\Users\wity\Downloads\pos_team12_final (1)\x64\Debug\pos_team12_final.exe(프로세스 4596개)이(가) 종료되었습니다(코드: 0개).  
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.  
이 창을 닫으려면 아무 키나 누르세요...
```

****이후 내용은 실제 코드 구현 내용입니다.**

<헤더파일>

```
1  #pragma once
2  #include <iostream>
3  #include <string>
4  #include <list>
5  using namespace std;
6
7  namespace pos {
8  public class abstract_item {
9      protected:
10         // Protected property
11         string name;
12         int price;
13         int total_count;
14         bool is_over_19;
15         list<int> identifier_array;
16
17     public:
18         // Constructor & Destructor
19         abstract_item();
20         virtual ~abstract_item();
21
22         // Public pure virtual method
23         virtual string get_name() = 0;
24         virtual int get_price() = 0;
25         virtual int get_total_count() = 0;
26         virtual bool get_is_over_19() = 0;
27         virtual list<int> get_identifier_array() = 0;
28
29         virtual void set_name(std::string) = 0;
30         virtual void set_price(int) = 0;
31         virtual void set_total_count(int) = 0;
32         virtual void set_is_over_19(char) = 0;
33         virtual void set_identifier_array(list<int>&) = 0;
34
35         virtual void update_total_count() = 0;
36         virtual void update_total_count(int) = 0;
37         virtual void item_disposal() = 0;
38         virtual int item_pop() = 0;
39     };
40 }
```

Abstract_item

```
1 #pragma once
2 #include <iostream>
3 #include <string>
4 #include <map>
5 #include "money.h"
6 #include "order_management.h"
7 #include "order.h"
8 #include "stock_management.h"
9
10 using namespace std;
11 namespace pos {
12     class account_management {
13     private:
14         map<int, int> sales_log; //일일 매출 기록하는 Map
15         const int ready_money = 100000; //준비금
16         money* money; //Money 접근 포인터
17         stock_management* stock_management; //Stock Management 접근 포인터
18         void dayclose(); //마감 정산
19         void push_sales_log(); //정산 할 때 기록 추가
20         void find_sales_log(int date); //날짜 Key로 해당 일자 매출 조회
21
22     public:
23         void set_stock_management(pos::stock_management&); //Account Management 객체 Setter
24         void set_money(pos::money&); //Money 객체 Setter
25         void account_run(); //Run 함수
26         void refund(order*); // 환불해야 하는 총 금액 다시 계산해서 돈 통에서 빼기
27         void purchase_pay(int, bool); //받은 돈 더하기
28         void calculate_change(int); //거스름 돈 빼기
29     };
30 };
31 
```

account_management

```
1 #pragma once
2 #include <iostream>
3 #include <string>
4 #include <list>
5 using namespace std;
6
7 namespace pos {
8     class cart_item {
9     protected:
10         // Protected property
11         std::string name;
12         int price;
13         int total_count;
14         bool is_over_19;
15         list<int> identifier_array;
16
17     public:
18         // Constructor & Destructor
19         cart_item();
20         ~cart_item();
21
22         // Public method
23         std::string get_name();
24         int get_price();
25         int get_total_count();
26         bool get_is_over_19();
27         list<int> get_identifier_array();
28
29         void set_name(std::string);
30         void set_price(int);
31         void set_total_count(int);
32         void set_is_over_19(char);
33     };
34 }
```

cart_item

```

1  #pragma once
2  #include <iostream>
3  #include <string>
4  #include <list>
5  #include "abstract_item.h"
6  #include "cart_item.h"
7  using namespace std;
8
9  namespace pos {
10  class item_list {
11  protected:
12      // Protected property
13      int item_count;
14
15  public:
16      // Constructor & Destructor
17      item_list();
18      ~item_list();
19
20      // Public property
21      list<abstract_item*> item_array;
22
23      // Public method
24
25      // 받은 수량만큼 total_count 업데이트
26      void update_item_count(int);
27      // 현재 item에 존재하는 수량 다시 세려서 total_count 업데이트
28      void update_item_count();
29      // name 받아서 이름이 동일한 item 반환
30      abstract_item* search_item(string);
31      // 상품 추가
32      void item_push(cart_item*);
33      // 상품 추가
34      void item_push(string, list<int>&);
35      // 상품 추가
36      void item_push(abstract_item*);
37      // list_item에서 name이 동일한 item 찾아 받은 수량만큼 pop
38      cart_item* item_pop(std::string, int);
39      // name과 동일한 item 존재여부 판단
40      bool is_exist_item(string);
41      // 폐기 조건에 맞는 item 폐기 (예: 유통기한 있는 item은 오늘날짜와 비교해 폐기, guarantee 있는 item은 오늘날짜보다 5년 이사 지났으면 폐기)
42      void item_disposal();
43      // item_list 내부에 있는 모든 item 이름 출력
44      void show_item_menu();
45  };
46 }

```

item_list

```

1  #pragma once
2  #include <iostream>
3  #include <string>
4  #include <list>
5  #include "item_list.h"
6  #include "cart_item.h"
7  #include "abstract_item.h"
8  #include "item_with_exdate.h"
9  #include "item_with_guarantee.h"
10
11 namespace pos {
12     class item_list_management {
13     private:
14         //Private property
15         item_list* item_list;
16
17     public:
18         //Public method
19         //item_list_management 객체 setter
20         void set_item_list(pos::item_list&);
21         //item_list에 물품 추가
22         void item_push(cart_item*);
23         //item_list에 물품 추가
24         void item_push(string, list<int>&);
25         //item_list에 물품 추가
26         void item_push(abstract_item*);
27         //item_list에서 물품 끄냄
28         cart_item* item_pop(string, int);
29         // 문자열을 공복을 기준으로 잘라 int형 배열로 반환
30         list<int> string_to_listint(string);
31         // item_list의 name과 이름이 동일한 상품이 존재하는지 판단
32         bool is_exist_item(string);
33         // 상품 입고
34         // 상품 입고시 item_list에 이미 존재하는 item인 경우, 각 상품의 고유 정보(유통기한, 보증기한 등)를 받아서 item에 추가
35         void add_exist_item(string);
36         // 상품 입고시 item_list에 존재하지 않는 item인 경우, item을 모든 정보를 받아 새로 생성한 후, item_list에 추가
37         void create_new_item(string);                                //item_list에 존재하는 메뉴 목록을 나열함
38         //item_list에 존재하는 item 중 폐기 조건에 맞는 item 폐기
39         void item_disposal();
40         //item_list에 존재하는 item 목록 나열
41         void show_item_menu();
42     };
43 }

```

Item_list_management

```
1 #pragma once
2 #include <string>
3 #include <list>
4 #include "abstract_item.h"
5 using namespace std;
6
7 namespace pos {
8
9     class item_with_exdate : public abstract_item {
10 protected:
11     // Protected property
12     std::string name;
13     int price;
14     int total_count;
15     bool is_over_19;
16     list<int> identifier_array;
17
18 public:
19     // Constructor & Destructor
20     item_with_exdate();
21     virtual ~item_with_exdate();
22
23     // Public virtual method
24     virtual std::string get_name();
25     virtual int get_price();
26     virtual int get_total_count();
27     virtual bool get_is_over_19();
28     virtual list<int> get_identifier_array();
29
30     virtual void set_name(std::string);
31     virtual void set_price(int);
32     virtual void set_total_count(int);
33     virtual void set_is_over_19(char);
34     virtual void set_identifier_array(list<int>&);
35
36     // 받은 수량만큼 total_count 업데이트
37     virtual void update_total_count(int);
38     // 현재 item에 존재하는 수량 다시 세려서 total_count 업데이트
39     virtual void update_total_count();
40
41     // 현재 item에 존재하는 폐기 조건에 맞는 것들 폐기
42     virtual void item_disposal();
43     // 가장 오래된 item pop & pick
44     virtual int item_pop();
45
46 };
}
```

item_with_exdate

```
1 #pragma once
2 #include <string>
3 #include <list>
4 #include "abstract_item.h"
5 using namespace std;
6
7 namespace pos {
8     class item_with_guarantee : public abstract_item {
9     protected:
10         // Protected property
11         std::string name;
12         int price;
13         int total_count;
14         bool is_over_19;
15         list<int> identifier_array;
16
17     public:
18         // Constructor & Destructor
19         item_with_guarantee();
20         virtual ~item_with_guarantee();
21
22         // Public virtual method
23         virtual std::string get_name();
24         virtual int get_price();
25         virtual int get_total_count();
26         virtual bool get_is_over_19();
27         virtual list<int> get_identifier_array();
28
29         virtual void set_name(std::string);
30         virtual void set_price(int);
31         virtual void set_total_count(int);
32         virtual void set_is_over_19(char);
33         virtual void set_identifier_array(list<int>&);
34
35         // 받은 수량만큼 total_count 업데이트
36         virtual void update_total_count();
37         // 현재 item에 존재하는 수량 다시 세려서 total_count 업데이트
38         virtual void update_total_count(int);
39
40         // 현재 item에 존재하는 폐기 조건에 맞는 것들 폐기
41         virtual void item_disposal();
42         // 가장 오래된 item pop & pick
43         virtual int item_pop();
44     };
45 }
```

item_with_guarantee

```
1 #pragma once
2 #include <iostream>
3 #include <string>
4 #include <map> //파일(money.h)이 속할 수 있는 다른 프로젝트를 보고 해
5
6 using namespace std;
7 namespace pos {
8
9     class money {
10     private:
11         int card_money; //카드
12         int cash; //현금
13         int today_money; //일일 매출
14         int capital; //총 자본금
15
16     public:
17         //constructor
18         money();
19         //destructor
20         ~money();
21
22         void set_card_money(int); //카드 Set
23         void set_cash(int); // 현금 Set
24         void set_capital(int); //자본금 Set
25         int get_today_money(); //일일 매출 Get
26         int get_capital(); //자본금 Get
27         void calculate_day_revenue(); //일일 정산
28         void plus(int, bool); //돈을 더할 때 사용
29         void minus(int, bool); //돈을 뺄 때 사용
30     };
31 } //형식 Clean Up
```

money

```
1 #pragma once
2 #include <iostream>
3 #include <string>
4 #include <list>
5 #include "shopping_cart.h"
6
7 using namespace std;
8
9 namespace pos {
10     class order {
11     private:
12         //private property
13         shopping_cart order_cart; // 구매 상품 목록
14         int order_id; // 영수증 번호
15         int coupon; // 쿠폰(금액 할인)
16         int payment; //소비자 지불 금액
17         int customer_age; // 구매자 나이
18         int change; //거스름돈
19         bool payment_type; //카드=0 현금=1
20         bool is_refunded = false;//환불 여부 default = false
21
22
23     public:
24
25         //public method
26
27         //setter
28         void set_order_id(int order_id); // order_id. from order_management
29         void set_payment(int payment);
30         void set_shopping_cart(shopping_cart cart);
31         void set_is_refunded(bool answer);
32         void set_payment_type(bool payment_type);
33         void set_coupon(int coupon);
34         void set_customer_age(int age);
35         void set_change(int change);
36         //getter
37         int get_order_id(void);
38         int get_payment(void);
39         shopping_cart get_shopping_cart(void);
40         bool get_is_refunded(void);
41         bool get_payment_type(void);
42
43
44
45     };
46 }
```

order

```

1 #pragma once
2 #include <list>
3 #include <iostream>
4 #include <string>
5 #include "order.h"
6
7 using namespace std;
8
9 namespace pos {
10 class order_log {
11 private:
12     //private property
13     list<order> log_list; //order기록들이 담기는 log_list
14 public:
15     //public method
16     bool is_exist_order(int order_id); //order_log에서 해당되는 order_id 존재하는지 판단
17     void order_push(order* myorder); //order를 log_list에 push
18     order* order_pop(int order_id); //order를 log_list에서 pop
19
20 };
21 }

```

order_log

```

1 #pragma once
2 #include <iostream>
3 #include <list>
4 #include "account_management.h"
5 #include "stock_management.h"
6 #include "shopping_cart.h"
7 #include "cart_item.h"
8 #include "order_log.h"
9
10 using namespace std;
11
12 namespace pos {
13 class order_management {
14 private:
15     int order_id; //영수증 번호
16     order_log* order_log; //order_log의 포인터
17
18 public:
19     //constructor
20     order_management();
21
22     //destructor
23     ~order_management();
24
25     void set_order_log(order_log* order_log); //order_log의 setter
26
27     void order_run(shopping_cart& cart, int coupon, int payment, int age, int change, bool payment_type); //order 생성 후 order_log에 추가
28
29     bool is_exist_order(int order_id); // order_log에서 해당되는 order_id 존재하는지 판단
30
31     order* pop_order_log(int order_id); //order_log에서 해당되는 order pop하기
32
33     void push_to_log(order* new_order); //order_log에 새로운 order push하기
34
35 };
36
37 };
38

```

order_management

```

1  #pragma once
2  #include <iostream>
3  #include <string>
4  #include "account_management.h"
5  #include "stock_management.h"
6  #include "order_management.h"
7  #include "shopping_cart.h"
8  #include "cart_item.h"
9
10 using namespace std;
11
12 namespace pos {
13
14     class purchase_management {
15
16     private:
17
18         account_management* account_management; //account_management에 접근하는 포인터
19         stock_management* stock_management; //stock_management에 접근하는 포인터
20         order_management* order_management; //order_management에 접근하는 포인터
21
22     public:
23
24         //setter
25         void set_stock_management(pos::stock_management* stock_management);
26         void set_account_management(pos::account_management* account_management);
27         void set_order_management(pos::order_management* order_management);
28
29         //메뉴 보여주는 함수
30         void show_menu();
31
32         //입력값 purchase_item_name(구매하고자 하는 상품 이름)으로 저장하는 함수
33         string get_purchase_item_name();
34
35         //입력값 purchase_item_count(구매하고자 하는 상품의 개수)으로 저장하는 함수
36         int get_purchase_item_count();
37
38         //판매를 위한 주 작동 함수
39         void purchase_run();
40
41         //주문 정보를 받아와서 shopping_cart생성
42         //입력값이 0일때 shopping_cart 생성 종료!
43         void put_in_shopping_cart(shopping_cart& cart);
44
45         //결제하는 과정 -> account_management에 접근하여 매출 증가시킴
46         void purchase_pay(int payment, bool payment_type);
47
48     };
}

```

purchase_management

```
1 #pragma once
2 #include <iostream>
3 #include <string>
4 #include <map>
5 #include "account_management.h"
6 #include "stock_management.h"
7 #include "order_management.h"
8
9
10 using namespace std;
11 namespace pos {
12     class refund_management {
13     private:
14         stock_management* stock_management; //Stock Management에 접근하기 위한 포인터
15         account_management* account_management; //Account Management에 접근하기 위한 포인터
16         order_management* order_management; //Order Management에 접근하기 위한 포인터
17
18     public:
19         void set_stock_management(pos::stock_management&); //Stock Management 객체 Setter
20         void set_account_management(pos::account_management&); //Account Management 객체 Setter
21         void set_order_management(pos::order_management&); //Order Management 객체 Setter
22         void refund_run();
23     };
24 }
25 }
```

refund_management

```
1 #pragma once
2 #include <list>
3 #include <iostream>
4 #include "cart_item.h"
5
6 using namespace std;
7
8 namespace pos {
9     class shopping_cart {
10
11     private:
12         //private property
13         list<cart_item*>cart_list; //cart_item이 담기는 리스트
14         int cart_total_count; //카트 내 총 물건의 개수
15         int cart_total_price; //카트 내 물건의 총 가격
16
17     public:
18
19         //constructor
20         shopping_cart();
21         ~shopping_cart();
22
23         //public method
24         void add_item(cart_item* item); //cart_item 객체 추가
25         //getter
26         int get_cart_total_count();
27         int get_cart_total_price();
28         list<cart_item*> get_cart_list();
29     };
30 }
```

shopping_cart

```
1 #pragma once
2 #include<iostream>
3 #include<string>
4 #include "item_list_management.h"
5 #include "cart_item.h"
6 #include "order.h"
7 #include "shopping_cart.h"
8
9 namespace pos {
10     class stock_management {
11     private:
12         //Private property
13         item_list_management* item_list_management;
14     public:
15         //Public method
16         // item_list_management 객체 setter
17         void set_item_list_management(pos::item_list_management* );
18         //환불시 shopping_cart에 담겨있던 상품을 다시 item_list에 push
19         void item_push(cart_item* );
20         //구매시 item의 이름과 수량만큼 item_list에서 pop
21         cart_item* item_pop(string, int);
22         //재고를 직접 추가
23         void add_item(string);
24         //상품 폐기처리
25         void item_disposal();
26         //환불 상품 재고에 다시 추가
27         void refund(order* );
28         //item_list에 담겨있는 모든 item의 이름 출력
29         void show_item_menu();
30         //재고관리 기능 실행
31         bool stock_run();
32     };
33 }
```

stock_management

abstract_item.cpp

```
1 #include "abstract_item.h"
2
3 namespace pos {
4     // Constructor
5     abstract_item::abstract_item() {
6         this->name = "";
7         this->price = 0;
8         this->total_count = 0;
9         this->is_over_19 = false;
10    }
11
12    // Destructor
13    abstract_item::~abstract_item() {}
14 }
```

item_with_exdate.cpp

```
1 #include "item_with_exdate.h"
2 extern int today;
3
4 namespace pos {
5     // Constructor
6     item_with_exdate::item_with_exdate() {
7         this->name = "";
8         this->price = 0;
9         this->total_count = 0;
10        this->is_over_19 = false;
11    }
12
13    // Destructor
14    item_with_exdate::~item_with_exdate() {}
15
16    // Public pure virtual method
17    std::string item_with_exdate::get_name() {
18        return this->name;
19    }
20
21    int item_with_exdate::get_price() {
22        return this->price;
23    }
24
25    int item_with_exdate::get_total_count() {
26        return this->total_count;
27    }
28
29    bool item_with_exdate::get_is_over_19() {
30        return this->is_over_19;
31    }
32
33    list<int> item_with_exdate::get_identifier_array() {
34        return this->identifier_array;
35    }
36
37    void item_with_exdate::set_name(std::string name) {
38        this->name = name;
39    }
40
41    void item_with_exdate::set_price(int price) {
42        this->price = price;
43    }
44
45    void item_with_exdate::set_total_count(int total_count) {
46        this->total_count = total_count;
47    }
48
49    void item_with_exdate::set_is_over_19(char answer) {
50        if (answer == 'y' || answer == 'Y') {
51            this->is_over_19 = true;
52        }
53        else if (answer == 'n' || answer == 'N') {
54            this->is_over_19 = false;
55        }
56    }
57 }
```

```
55     }
56     else {
57         cout << "잘못된 입력입니다." << endl;
58     }
59 }
60 void item_with_exdate::set_identifier_array(list<int>& identifier_array) {
61     this->identifier_array = identifier_array;
62 }
63 // 받은 수량만큼 total_count 업데이트
64 void item_with_exdate::update_total_count(int update_count) {
65     this->total_count += update_count;
66 }
67 // 현재 item에 존재하는 수량 다시 세려서 total_count 업데이트
68 void item_with_exdate::update_total_count() {
69     this->total_count = this->identifier_array.size();
70 }
71 // 현재 item에 존재하는 폐기 조건에 맞는 것을 폐기
72 void item_with_exdate::item_disposal() {
73     list<int>::iterator it; //반복자 선언
74     for (it = this->identifier_array.begin(); it != this->identifier_array.end(); it++) {
75         // 유통기한이 오늘 날짜보다 작으면 폐기
76         if (*it < today) {
77             this->identifier_array.remove(*it);
78         }
79     }
80 }
81 // 가장 오래된 item pop & pick
82 int item_with_exdate::item_pop() {
83     int item = this->identifier_array.front();
84     this->identifier_array.pop_front();
85     return item;
86 }
87 
```

item_with_guarantee.cpp

```
1 #include "item_with_guarantee.h"
2 extern int today;
3
4 namespace pos {
5     // Constructor
6     item_with_guarantee::item_with_guarantee() {
7         this->name = "";
8         this->price = 0;
9         this->total_count = 0;
10        this->is_over_19 = false;
11    }
12
13     // Destructor
14     item_with_guarantee::~item_with_guarantee() {}
15
16     // Public pure virtual method
17     std::string item_with_guarantee::get_name() {
18         return this->name;
19     }
20
21     int item_with_guarantee::get_price() {
22         return this->price;
23     }
24
25     int item_with_guarantee::get_total_count() {
26         return this->total_count;
27     }
28
29     bool item_with_guarantee::get_is_over_19() {
30         return this->is_over_19;
31     }
32
33     list<int> item_with_guarantee::get_identifier_array() {
34         return this->identifier_array;
35     }
36
37     void item_with_guarantee::set_name(std::string name) {
38         this->name = name;
39     }
40
41     void item_with_guarantee::set_price(int price) {
42         this->price = price;
43     }
44
45     void item_with_guarantee::set_total_count(int total_count) {
46         this->total_count = total_count;
47     }
48
49     void item_with_guarantee::set_is_over_19(char answer) {
50         if (answer == 'y' || answer == 'Y') {
51             this->is_over_19 = true;
52         }
53         else if (answer == 'n' || answer == 'N') {
54             this->is_over_19 = false;
55         }
56     }
57 }
```

```
55     }
56     else {
57         cout << "잘못된 입력입니다." << endl;
58     }
59 }
60
61 void item_with_guarantee::set_identifier_array(list<int>& identifier_array) {
62     this->identifier_array = identifier_array;
63 }
64
65 // 받은 수량만큼 total_count 업데이트
66 void item_with_guarantee::update_total_count(int update_count) {
67     this->total_count += update_count;
68 }
69
70 // 받은 수량만큼 total_count 업데이트
71 void item_with_guarantee::update_total_count() {
72     this->total_count = this->identifier_array.size();
73 }
74
75 // 현재 item에 존재하는 폐기 조건에 맞는 것을 폐기
76 void item_with_guarantee::item_disposal() {
77     list<int>::iterator it; //반복자 선언
78     for (it = this->identifier_array.begin(); it != this->identifier_array.end(); it++) {
79         // 보증기한이 오늘 날짜보다 5년이전이면 폐기
80         if (*it < today - 50000) {
81             this->identifier_array.remove(*it);
82         }
83     }
84 }
85
86 // 가장 오래된 item pop & pick
87 int item_with_guarantee::item_pop() {
88     int item = this->identifier_array.front();
89     this->identifier_array.pop_front();
90     return item;
91 }
92 []
93 }
```

item_list.cpp

```
1 #include "item_list.h"
2
3 namespace pos {
4     // Constructor
5     item_list::item_list() {
6         this->item_count = 0;
7     }
8
9     // Destructor: item_list내의 item_array에 담긴 item 객체를 delete
10    item_list::~item_list() {
11        // 반복자 생성
12        list<abstract_item*>::iterator iter = this->item_array.begin();
13        abstract_item* item;
14
15        for (iter = this->item_array.begin(); iter != this->item_array.end(); iter++) {
16            item = *iter;
17            delete(item);
18        }
19    }
20
21    // Public method
22
23    // 받은 수량만큼 total_count 업데이트
24    void item_list::update_item_count(int update_count) {
25        this->item_count += update_count;
26    }
27
28 // 현재 item에 존재하는 수량 다시 세려서 total_count 업데이트
29 void item_list::update_item_count() {
30     int total_item_count = 0;
31     // 반복자 생성
32     list<abstract_item*>::iterator iter = this->item_array.begin();
33     abstract_item* item;
34
35     for (iter = this->item_array.begin(); iter != this->item_array.end(); iter++)
36     {
37         item = *iter;
38         total_item_count += item->get_total_count();
39     }
40     this->item_count = total_item_count;
41
42 // name 받아서 이름이 동일한 item 반환
43 abstract_item* item_list::search_item(string name) {
44     // 반복자 생성
45     list<abstract_item*>::iterator iter = this->item_array.begin();
46     abstract_item* item = NULL;
47
48     for (iter = this->item_array.begin(); iter != this->item_array.end(); iter++)
49     {
50         item = *iter;
51         if (item->get_name() == name) break;
52     }
53     return item;
54 }
55
56 // 상품 추가
57 void item_list::item_push(cart_item* cart_item) {
58     abstract_item* item = search_item(cart_item->get_name());
59     // cart_item 내부 상품을 item에 넣은 후 경합
60     item->get_identifier_array().merge(cart_item->get_identifier_array());
61     item->get_identifier_array().sort();
62     // 상품 수량 변경
63     item->update_total_count(cart_item->get_total_count());
64     this->update_item_count(cart_item->get_total_count());
65
66     delete(cart_item);
67 }
68
69 // 상품 추가
70 void item_list::item_push(string name, list<int>& identifier_array) {
71     abstract_item* item = search_item(name);
72     item->get_identifier_array().merge(identifier_array);
73     item->update_total_count();
74 }
75
76 // 상품 추가
77 void item_list::item_push(abstract_item* new_item) {
78     this->item_array.push_back(new_item);
79     new_item->update_total_count();
80 }
81
82 // list_item에서 name이 동일한 item 찾아 받은 수량만큼 pop
83 void item_list::item_pop(string name, int amount) {
84 }
```

```
82     cart_item* item_list::item_pop(string name, int count) {
83         abstract_item* item = search_item(name);
84         // cart item 생성
85         cart_item* my_cart = new cart_item();
86         my_cart->set_name(item->get_name());
87         my_cart->set_price(item->get_price());
88         my_cart->set_total_count(count);
89         my_cart->set_is_over_19(item->get_is_over_19());
90         // cart item에 상품 담기
91         for (int item_count = 0; item_count < count; item_count++) {
92             my_cart->get_identifier_array().push_back(item->item_pop());
93         }
94         // 변동 수량 update
95         item->update_total_count(-count);
96         this->update_item_count(-count);
97
98         return my_cart;
99     }
100
101    // name과 동일한 item 존재여부 판단
102    bool item_list::is_exist_item(string name) {
103        bool exist = false;
104        // 반복자 생성
105        list<abstract_item*>::iterator iter = this->item_array.begin();
106        abstract_item* item;
107
108        for (iter = this->item_array.begin(); iter != this->item_array.end(); iter++)
109        {
110            item = *iter;
111            if (item->get_name() == name) {
112                exist = true;
113                break;
114            }
115        }
116        return exist;
117    }
118
119    // 폐기 조건에 맞는 item 폐기 (예: 유통기한 있는 item은 오늘날짜와 비교해 폐기, guarantee 있는 item은 오늘날짜보다 5년 이사 지났으면 폐기)
120    void item_list::item_disposal() {
121        // 반복자 생성
122        list<abstract_item*>::iterator iter = this->item_array.begin();
123        abstract_item* item;
124        for (iter = this->item_array.begin(); iter != this->item_array.end(); iter++)
125        {
126            item = *iter;
127            item->item_disposal();
128        }
129        this->update_item_count();
130    }
131
132    // item_list 내부에 있는 모든 item 이름 출력
133    void item_list::show_item_menu() {
134        // 반복자 생성
135        list<abstract_item*>::iterator iter = this->item_array.begin();
136
137        abstract_item* item;
138        int count = 0;
139        for (iter = this->item_array.begin(); iter != this->item_array.end(); iter++)
140        {
141            count = ++count;
142            item = *iter;
143            cout << count << ". " << item->get_name() << " " << "가격: " << item->get_price() << " " << "남은 수량: " << item->get_total_count() << endl;
144        }
145    }
```

```
100 % 문제가 검색되지 않음 < > 줄: 1 문자: 1 템 CRI
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
```

```
100 % 문제가 검색되지 않음 < > 줄: 1 문자: 1 템 CRI
136
137
138
139
140
141
142
143
144
145
```

```
100 % 문제가 검색되지 않음 < > 줄: 1 문자: 1 템 CRI
```

shopping_cart.cpp

```
1 #include "shopping_cart.h"
2
3 namespace pos {
4
5     //constructor
6     shopping_cart::shopping_cart() {
7         cart_total_count = 0;
8         cart_total_price = 0;
9     }
10
11    //destructor
12    shopping_cart::~shopping_cart() { //cart_list 내의 cart_item 하나씩 delete
13        list<cart_item*>::iterator iter = this->cart_list.begin();
14        cart_item* cart_item;
15        for (iter = this->cart_list.begin(); iter != this->cart_list.end(); iter++) {
16            cart_item = &(*iter);
17            delete(cart_item);
18        }
19    }
20
21    //public method
22    int shopping_cart::get_cart_total_count() {
23        return this->cart_total_count;
24    }
25    int shopping_cart::get_cart_total_price() {
26        return this->cart_total_price;
27    }
28
29    void shopping_cart::add_item(cart_item* item) {
30        cart_list.push_back(item);
31        int count = item->get_total_count();
32        cart_total_count += count; //개수 더해주기
33        cart_total_price += count * item->get_price(); //개수*가격
34    }
35
36    list<cart_item*> shopping_cart::get_cart_list() { //getter of cart_list
37        return this->cart_list;
38    }
}
```

cart_item.cpp

```
1 #include "cart_item.h"
2
3 namespace pos {
4     // Constructor
5     cart_item::cart_item() {
6         this->name = "";
7         this->price = 0;
8         this->total_count = 0;
9         this->is_over_19 = false;
10    }
11
12     // Destructor
13     cart_item::~cart_item() {}
14
15     // Public pure virtual method
16     std::string cart_item::get_name() {
17         return this->name;
18     }
19
20     int cart_item::get_price() {
21         return this->price;
22     }
23
24     int cart_item::get_total_count() {
25         return this->total_count;
26     }
27
28     bool cart_item::get_is_over_19() {
29         return this->is_over_19;
30     }
31
32     list<int> cart_item::get_identifier_array() {
33         return this->identifier_array;
34     }
35
36     void cart_item::set_name(std::string name) {
37         this->name = name;
38     }
39
40     void cart_item::set_price(int price) {
41         this->price = price;
42     }
43
44     void cart_item::set_total_count(int total_count) {
45         this->total_count = total_count;
46     }
47
48     void cart_item::set_is_over_19(char answer) {
49         int age = answer - '0';
50         if (age > 19) {
51             this->is_over_19 = true;
52         }
53         else {
54             this->is_over_19 = false;
55         }
56     }
57
58 }
```

order.cpp

```
1 #include "order.h"
2
3 namespace pos {
4     //public method
5     //setter
6     void order::set_order_id(int order_id) {
7         this->order_id = order_id;
8     }
9
10    void order::set_payment(int payment) {
11        this->payment = payment;
12    }
13
14    void order::set_payment_type(bool payment_type) {
15        this->payment_type = payment_type;
16    }
17
18    void order::set_shopping_cart(shopping_cart cart) {
19        this->order_cart = cart;
20    }
21
22    void order::set_is_refunded(bool answer) {
23        this->is_refunded = answer;
24    }
25
26    void order::set_coupon(int coupon) {
27        this->coupon = coupon;
28    }
29
30    void order::set_customer_age(int age) {
31        this->customer_age = age;
32    }
33
34    void order::set_change(int change) {
35        this->change = change;
36    }
37
38    //getter
39    int order::get_order_id(void) {
40        return this->order_id;
41    }
42
43    int order::get_payment(void) {
44        return this->payment;
45    }
46
47    bool order::get_payment_type(void) {
48        return this->payment_type;
49    }
50
51    shopping_cart order::get_shopping_cart(void) {
52        return this->order_cart;
53    }
54
55    bool order::get_is_refunded(void) {
56        return this->is_refunded;
57    }
58 }
```

100 % ⌂ 문제가 검색되지 않음 ⌂ 줄: 1 문자: 1 템 CRI

```
28    void order::set_customer_age(int age) {
29        this->customer_age = age;
30    }
31    void order::set_change(int change) {
32        this->change = change;
33    }
34
35    //getter
36    int order::get_order_id(void) {
37        return this->order_id;
38    }
39    int order::get_payment(void) {
40        return this->payment;
41    }
42
43    bool order::get_payment_type(void) {
44        return this->payment_type;
45    }
46    shopping_cart order::get_shopping_cart(void) {
47        return this->order_cart;
48    }
49
50    bool order::get_is_refunded(void) {
51        return this->is_refunded;
52    }
53
54 }
```

100 % ⌂ 문제가 검색되지 않음 ⌂ 줄: 1 문자: 1 템 CRI

order_log.cpp

```
1 #include "order_log.h"
2
3 namespace pos {
4
5     //public method
6
7     bool order_log::is_exist_order(int order_id) {
8         bool exist = false;
9         for (list<order>::iterator iter = log_list.begin(); iter != log_list.end(); iter++) {
10             if (iter->get_order_id() == order_id && iter->get_is_refunded() == false) {
11                 exist = true;
12                 break;
13             }
14         }
15         return exist;
16     }
17
18     void order_log::order_push(order* myorder) { //order를 log_list에 push
19         log_list.push_back(*myorder); //값을 빼서 push back
20         delete myorder; //동적할당 delete
21     }
22
23     order* order_log::order_pop(int order_id) { //order를 log_list에서 pop
24         for (list<order>::iterator iter = log_list.begin(); iter != log_list.end(); iter++) {
25             if (iter->get_order_id() == order_id && iter->get_is_refunded() == false) {
26                 iter->set_is_refunded(true); //is_refunded = true, 환불 완료.
27                 order* myorder = &(*iter);
28                 return myorder;
29             }
30         }
31     }
32 }
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100 %
```

● 문제가 검색되지 않음 줄: 1 문자: 1 SPC CRL

main.cpp

```
1 #include <iostream>
2 #include <sstream>
3 #include <string>
4 #include <list>
5 #include "abstract_item.h"
6 #include "account_management.h"
7 #include "cart_item.h"
8 #include "item_list.h"
9 #include "item_list_management.h"
10 #include "item_with_exdate.h"
11 #include "item_with_guarantee.h"
12 #include "money.h"
13 #include "order.h"
14 #include "order_log.h"
15 #include "order_management.h"
16 #include "purchase_management.h"
17 #include "refund_management.h"
18 #include "shopping_cart.h"
19 #include "stock_management.h"
20
21 using namespace pos;
22 using namespace std;
23
24 // 구현을 위해 today를 전역변수로 설정
25 int today = 20221207;
26
27 int main() {
28 }
```

● 문제가 검색되지 않음 줄: 1 문자: 1 SPC CRL

```

28     {
29         item_list item_list;
30         account_management account_management;
31         order_management order_management;
32         refund_management refund_management;
33         stock_management stock_management;
34         purchase_management purchase_management;
35         item_list_management item_list_management;
36         order_log order_log;
37         money money;
38
39         account_management.set_stock_management(stock_management);
40         account_management.set_money(money);
41         order_management.set_order_log(&order_log);
42         refund_management.set_stock_management(stock_management);
43         refund_management.set_account_management(account_management);
44         refund_management.set_order_management(order_management);
45         stock_management.set_item_list_management(&item_list_management);
46         purchase_management.set_stock_management(&stock_management);
47         purchase_management.set_account_management(&account_management);
48         purchase_management.set_order_management(&order_management);
49         item_list_management.set_item_list(item_list);
50
51         int response = 0;
52         cout << endl;
53
54         cout << "POS 시스템을 시작합니다." << endl;
55
56     while (1) {
57         cout << endl;
58         cout << "오늘의 날짜는 " << today << " 입니다." << endl;
59         cout << "무엇을 하시겠습니까?" << endl;
60         cout << endl;
61         cout << "===== POS 목록 =====" << endl;
62         cout << " 1. 재고 관리   2. 구매" << endl;
63         cout << " 3. 환불   4. 회계" << endl;
64         cout << " 5. POS 시스템 종료(모든 정보 초기화)" << endl;
65         cout << "===== =====" << endl;
66         cout << "입력 (입력 예시: 1) : ";
67         cin >> response;
68         if (response == 1) {
69             stock_management.stock_run();
70         }
71         else if (response == 2) {
72             purchase_management.purchase_run();
73         }
74         else if (response == 3) {
75             refund_management.refund_run();
76         }
77         else if (response == 4) {
78             account_management.account_run();
79         }
80         else if (response == 5) {
81             cout << endl;
82             cout << "POS 시스템을 종료합니다." << endl;
83             break;
84         }
85         else {
86             cout << "유효하지 않은 입력입니다. 다시 입력하세요." << endl;
87         }
88     }
89
90
91
92     return 0;

```

refund_management.cpp

```
1 #include "refund_management.h"
2
3 namespace pos {
4     //Stock Management 객체 Setter
5     void refund_management::set_stock_management(pos::stock_management& stock_management) {
6         this->stock_management = &stock_management;
7     }
8     //Account Management 객체 Setter
9     void refund_management::set_account_management(pos::account_management& account_management) {
10        this->account_management = &account_management;
11    }
12    //Order Management 객체 Setter
13    void refund_management::set_order_management(pos::order_management& order_management) {
14        this->order_management = &order_management;
15    }
16
17 void refund_management::refund_run() {
18     cout << endl;
19     order* refund_order = NULL;
20     cout << "환불을 진행하시겠습니까? (부분 환불은 가능하지 않습니다.) (Y / N)";
21     char answer;
22     cin >> answer;
23     if (answer == 'y' || answer == 'Y') {
24
25         cout << endl;
26         cout << "===== 환불 시작 =====" << endl;
27         cout << "===== 환불 여스즈 비우트 인터내 즈리오 (이전 에너지)" << endl;
28
29         cout << "환불 영수증 번호를 입력해 주세요. (입력 예시: 0)" ;
30         int refund_order_id;
31         cin >> refund_order_id;
32         refund_order = this->order_management->pop_order_log(refund_order_id); //Order Management 통해 영수증 번호 조회하기 위해 Null 포인터 매개 변수 보냄.
33         if (refund_order == NULL) //My Order가 그대로 Null이면 실패
34         {
35             cout << "존재하지 않는 영수증 번호입니다." << endl;
36             cout << "===== 환불 실패 =====" << endl;
37             cout << endl;
38             cout << endl;
39         }
40         else { //My Order가 주소 값을 가져 오면 성공
41             cout << "내역 조회에 성공했습니다. 환불을 진행합니다." << endl;
42             this->stock_management->refund(refund_order); //Stock Management에서 환불 진행
43             this->account_management->refund(refund_order); //Account Management에서 환불 진행
44             cout << "환불을 성공했습니다." << endl;
45             cout << "===== 환불 종료 =====" << endl;
46             cout << endl;
47             cout << endl;
48         }
49         else if (answer == 'n' || answer == 'N') {
50             cout << "환불 결차를 취소합니다." << endl;
51             cout << "===== 환불 취소 =====" << endl;
52             cout << endl;
53             cout << endl;
54         }
55     }
56 }
```

stock_management.cpp

```
1 #include "stock_management.h"
2
3 namespace pos {
4
5     //Public method
6     //item_list_management 객체 setter
7     void stock_management::set_item_list_management(pos::item_list_management* item_list_management) {
8         this->item_list_management = item_list_management;
9     }
10
11     //환불시 shopping_cart에 담겨있던 물품을 다시 item_list에 push
12     void stock_management::item_push(cart_item* cart_item) {
13         this->item_list_management->item_push(cart_item);
14     }
15
16     //구매시 item의 이름과 수량만큼 item_list에서 pop
17     cart_item* stock_management::item_pop(string name, int num) {
18         return this->item_list_management->item_pop(name, num);
19     }
20
21     //재고를 직접 추가
22     void stock_management::add_item(string name) {
23         this->item_list_management->add_item(name);
24     }
25
26     //상품 폐기처리
27     void stock_management::item_disposal() {
28         this->item_list_management->item_disposal();
29     }
30
31     //환불 상품 재고에 다시 추가
32     void stock_management::refund(order* refund_order) {
33         shopping_cart* cart = refund_order->get_shopping_cart();
34         int count = cart.get_cart_total_count();
35
36         list<cart_item*>::iterator it;
37         it = cart.get_cart_list().begin();
38         for (int i = 0; i < count; i++) {
39             this->item_push(*it);
40         }
41     }
42
43     //item_list에 담겨있는 모든 item의 이름 출력
44     void stock_management::show_item_menu() {
45         this->item_list_management->show_item_menu();
46     }
47
48     //재고관리 기능 실행
49     bool stock_management::stock_run() {
50         cout << endl;
51         char answer;
52         while (true) {
53             cout << "재고를 추가하시겠습니까? (Y / N) ";
54             cin >> answer;
55             if (answer == 'y' || answer == 'Y') {
56                 while (true) {
57                     string name;
58                     cout << endl;
59                     cout << endl;
60                     cout << "===== 재고 추가 시작 =====" << endl;
61                     cout << "추가할 상품의 이름을 입력해주세요. (입력 예시: banana, 더이상 추가할 상품이 없을 경우 0입력) ";
62                     cin >> name;
63                     if (name.compare("0") == 0) break;
64                     this->add_item(name);
65                     cout << "===== 재고 추가 완료 =====" << endl;
66                     cout << endl;
67                     cout << endl;
68                 }
69             }
70             else {
71                 cout << "재고를 추가하지 않습니다." << endl;
72                 cout << "===== 재고 추가 취소 =====" << endl;
73                 cout << endl;
74                 cout << endl;
75                 break;
76             }
77         }
78         return true;
79     }
80 }
```

account_management.cpp

```
1 #include "account_management.h"
2 extern int today;
3
4 namespace pos {
5     //Stock Management 객체 Setter
6     void account_management::set_stock_management(pos::stock_management& stock_management) {
7         this->stock_management = &stock_management;
8     }
9
10    void account_management::set_money(pos::money& money) {
11        this->money = &money;
12    }
13
14    //마감 경산
15    void account_management::dayclose() {
16        this->money->calculate_day_revenue(); //매출 합산 실행
17        this->stock_management->item_disposal(); //Stock Management에 폐기 전달
18        push_sales_log(); //날짜와 매출 Push
19        this->money->set_card_money(0); //카드 매출 초기화
20        this->money->set_cash(this->ready_money); //현금 통 초기화
21        int new_capital;
22        new_capital = this->money->get_capital() - this->ready_money;
23        this->money->set_capital(new_capital); //현금 통 채운 만큼 자본금에서 제하기
24    }
25
26    //경산 할 때 기록 추가
27    void account_management::push_sales_log() {
28        int today_money;
29        today_money = this->money->get_today_money() - 100000; //Get 함수로 Money에서 일일 매출 가져 오기
30
31        (this->sales_log).insert({{today, today_money}}); //날짜랑 일일 매출 Sales_Log에 넣기
32        cout << "매출 로그 업데이트 완료" << endl;
33    }
34
35    //날짜 Key로 해당 일자 매출 조회
36    void account_management::find_sales_log(int date) {
37        if ((this->sales_log.count(date)) {
38            cout << date << " 일자의 매출: " << (this->sales_log).find(date)->second << endl; //Map 기능 통해 매출 찾기
39            cout << "===== 일일 매출을 조회 완료 =====" << endl;
40            cout << endl;
41            cout << endl;
42        } else {
43            cout << "매출 기록을 찾을 수 없습니다." << endl;
44            cout << "===== 일일 매출을 조회 실패 =====" << endl;
45            cout << endl;
46            cout << endl;
47        }
48    }
49
50
51    //Run 함수
52    void account_management::account_run() {
53        cout << endl;
54        cout << "원하시는 메뉴를 선택해 주세요." << endl;
55        cout << "===== 1. 마감 경산 2 : 매출 조회" << endl;
56        cout << "===== =====" << endl;
57        cout << "===== 1. 마감 경산 2 : 매출 조회" << endl;
58        cout << "===== =====" << endl;
59        cout << endl;
60        int menu_choice;
61        cout << "입력 (입력 예시: 1) : ";
62        cin >> menu_choice;
63        if (menu_choice == 1)
64        {
65            cout << endl;
66            cout << endl;
67            cout << "===== 마감 경산 =====" << endl;
68            cout << "마감 경산을 시작하시겠습니까? (Y / N)" << endl;
69            char answer;
70            cin >> answer;
71            if ((answer == 'Y') || (answer == 'y'))
72            {
73                this->dayclose(); //매출 합산 + 폐기 + Money 초기화
74                today += 1; //날짜 +1
75                cout << "마감 경산이 완료되었습니다. 오늘 날짜가 변경됩니다." << endl;
76                cout << "===== 마감 경산 완료 =====" << endl;
77                cout << endl;
78                cout << endl;
79            }
80        }
81        else if ((answer == 'N') || (answer == 'n')) {
82            cout << "마감 경산을 취소합니다." << endl;
83        }
84    }
85
86    //문제가 검색되지 않음
87
88    //문제가 검색되지 않음
89
90    //문제가 검색되지 않음
91
92    //문제가 검색되지 않음
93
94    //문제가 검색되지 않음
95
96    //문제가 검색되지 않음
97
98    //문제가 검색되지 않음
99
100 //문제가 검색되지 않음
```

```

82     cout << "마감 경산을 취소합니다." << endl;
83     cout << "===== 마감 경산 취소 =====" << endl;
84     cout << endl;
85     cout << endl;
86 }
87 }
88 else if (menu_choice == 2)
89 {
90     cout << "일일 매출을 조회하시겠습니까? (Y / N)" << endl;
91     char answer;
92     cin >> answer;
93     if ((answer == 'Y') || (answer == 'y'))
94     {
95         cout << endl;
96         cout << endl;
97         cout << "===== 일일 매출을 조회 시작 =====" << endl;
98         cout << "조회할 날짜를 입력해 주세요. (입력 예시 : YYYYMMDD)";
99         int key_date;
100        cin >> key_date;//조회하고자 하는 날짜 입력
101        this->find_sales_log(key_date);
102    }
103    else if ((answer == 'N') || (answer == 'n'))
104    {
105        cout << "매출 조회를 취소합니다." << endl;
106        cout << "===== 일일 매출을 조회 취소 =====" << endl;
107        cout << endl;
108    }
109 }
110
111 // 환불해야 하는 총 금액 다시 계산해서 돈 통에서 빼기
112 void account_management::refund(order* refund_order) {
113     int refund_money, refund_payment_type;
114     refund_money = refund_order->get_payment(); //환불 기록의 총 금액
115     refund_payment_type = refund_order->get_payment_type(); //환불 기록의 결제 수단
116     money->minus(refund_money, refund_payment_type); //환불해야 하는 금액 만큼 차감
117 }
118
119 //받은 돈 더하기
120 void account_management::purchase_pay(int received_money, bool payment_type) {
121     money->plus(received_money, payment_type);
122 }
123
124 //거스름 돈 빼기
125 void account_management::calculate_change(int change) {
126     money->minus(change, false);
127 }
128
129 }

```

money.cpp

```

1 #include "money.h"
2
3 namespace pos {
4
5     money::money() {
6         card_money = 0;
7         cash = 100000; //초기 준비금
8         today_money = 0;
9         capital = -100000; //초기에는 현금으로 준비금을 빼서 미리 10만 원을 빼 둠
10    }
11
12    money::~money() {};
13
14    void money::set_card_money(int set_amount) { this->card_money = set_amount; }
15
16    void money::set_cash(int set_amount) { this->cash = set_amount; }
17
18    void money::set_capital(int set_amount) { this->capital = set_amount; }
19
20    void money::plus(int money, bool payment_type) {
21        if (payment_type == false)
22            this->card_money += money; // 카드 매출 증가
23        else if (payment_type == true)
24            this->cash += money; // 현금 매출 증가
25    }
26
27    void money::minus(int money, bool payment_type) {
28        if (payment_type == false)
29            this->card_money -= money; // 카드 매출 감소
30        else if (payment_type == true)
31            this->cash -= money; // 현금 매출 감소
32    }
33 }
34
35 문제가 검색되지 않음

```

```
28     if (payment_type == false)
29         this->card_money -= money; // 카드 매출 감소
30     else if (payment_type == true)
31         this->cash -= money; // 현금 매출 감소
32 }
33
34 void money::calculate_day_revenue() {
35     this->today_money = this->card_money + this->cash; //카드 현금 합산
36     this->capital += this->today_money; //자본금에 하루 매출 합산
37 }
38
39 int money::get_today_money() { return this->today_money; }
40
41 int money::get_capital() { return this->capital; }
42 }
```

100% 문제가 검색되지 않음 줄: 1 문자: 1 템 CRI

purchase_management.cpp

```
1 #include "purchase_management.h"
2
3 namespace pos {
4
5     //setter of management pointer
6
7     void purchase_management::set_stock_management(pos::stock_management* stock_management) {
8
9         this->stock_management = stock_management;
10    }
11
12     void purchase_management::set_account_management(pos::account_management* account_management) {
13
14         this->account_management = account_management;
15    }
16
17     void purchase_management::set_order_management(pos::order_management* order_management) {
18
19         this->order_management = order_management;
20    }
21
22     //메뉴 보여주는 함수
23     void purchase_management::show_menu() {
24
25         cout << "=====상품 목록=====" << endl;
26         cout << "          상품 목록          " << endl;
27         stock_management->show_item_menu();
28         cout << "=====상품 목록===== " << endl;
29     }
30
31 }
```

100% 문제가 검색되지 않음 줄: 1 문자: 1 템 CRI

```
28 //입력값 purchase_item_name(구매하고자 하는 상품 이름)으로 저장하는 함수
29
30 string purchase_management::get_purchase_item_name() {
31     string temp_purchase_item_name;
32     cin >> temp_purchase_item_name;
33
34     return temp_purchase_item_name;
35 }
36
37
38 //입력값 purchase_item_count(구매하고자 하는 상품의 개수)으로 저장하는 함수
39
40 int purchase_management::get_purchase_item_count() {
41
42     int temp_purchase_item_count;
43     cin >> temp_purchase_item_count;
44     return temp_purchase_item_count;
45 }
46
47
48 //판매를 위한 주 작성 함수
49
50 void purchase_management::purchase_run() {
51     cout << endl;
52     shopping_cart cart; // 구매할 상품 목록
53
54     int age = 0; //손님 나이 받아와서 저장
55     int coupon = 0; //소년에게서 쿠폰 할인가지 넘겨오
```

100% 문제가 검색되지 않음 줄: 1 문자: 1 템 CRI

```

55     int coupon = 0; //손님에게서 쿠폰 할인가 받아옴
56     int total_order_price = 0; //구매하는 상품 총 가격
57     int change = 0; //거스름돈
58     int paid_money = 0; //손님이 실제로 지불한 금액
59     int payment = 0; // total_order_price - coupon
60     bool payment_type;
61
62     this->show_menu(); //메뉴 보여줌
63     cout << endl;
64     cout << endl;
65     cout << "===== 구매 시작 =====" << endl;
66
67     cout << "손님의 나이를 입력해주세요: (입력 예시 : 20) : "; //손님 나이 물어봄
68     cin >> age; //age에 손님 나이 저장
69
70     put_in_shopping_cart(cart); //주문에 대한 것 : 구매할 item정보 받기 + item을 shopping_cart에 담기 + shopping_cart를 order.management로 보내기
71
72     total_order_price = cart.get_cart_total_price();
73
74     cout << endl;
75     cout << "결제를 시작합니다." << endl;
76     cout << "쿠폰이 있다면 할인가를 입력하세요 (입력 예시 : 500, 없다면 0입력) : ";
77     cin >> coupon; //coupon에 할인가 저장
78
79     payment = total_order_price - coupon;
80
81     payment = total_order_price - coupon;
82
83     cout << "총 " << payment << "원 입니다." << endl;
84
85     cout << "결제 방식을 선택해주세요 (0: 카드 1: 현금) : ";
86     cin >> payment_type;
87
88     if (payment_type == false) {
89         paid_money = payment; //카드이므로 금액 딱 맞춤
90         cout << paid_money << "원 결제 완료." << endl;
91     }
92     else {
93         cout << "손님에게 받은 금액을 입력하세요. (입력 예시: 20000) : " << endl;
94         cin >> paid_money;
95         change = paid_money - payment;
96         cout << payment << "원 결제 완료." << endl;
97         cout << "거스를 돈은 " << change << "원입니다." << endl;
98         this->account_management->calculate_change(change);
99     }
100
101     this->purchase_pay(paid_money, payment_type); // account_management에 payment와 payment_type 전달하는 내부 메서드
102
103     order_management->order_run(cart, coupon, payment, age, change, payment_type);
104     cout << "===== 구매 완료 =====" << endl;
105     cout << endl;
106 }
107
108 //주문 정보를 받아와서 shopping_cart 생성
109 //이렇게만 주면 shopping_cart 생성 충분!
110 void purchase_management::put_in_shopping_cart(shopping_cart& cart) {
111
112     string purchase_item_name = ""; //상품이름
113     int purchase_item_count = 0; //해당 상품 개수
114     cart_item* my_item; //stock_management에서 실제 item 객체 받아서 저장할거임, 이후 shopping cart에 넘겨줄거
115
116     while (1) {
117
118         //1.손님으로부터 주문 정보를 가져옴
119         cout << endl;
120         cout << "위의 상품 목록을 참고하여 구매할 상품을 하나씩 입력해주세요." << endl;
121         cout << "(입력 예시: banana, 없다면 0입력)(한 번에 한 종류의 상품만 입력하세요.) : ";
122         cin >> purchase_item_name;
123
124         if ((purchase_item_name.compare("0")) == 0) {
125             cout << "장바구니에 구매할 상품이 모두 추가되었습니다." ;
126             break;
127         }
128         cout << "구매할 상품의 개수를 입력하세요. (입력 예시: 1) : ";
129         cin >> purchase_item_count;
130
131         //2-1. stock management로부터 item 객체 정보를 복사하여 cart_item 자료형으로 받아오기
132         my_item = stock_management->item_pop(purchase_item_name, purchase_item_count);
133
134         //2-2. 받아온 cart_item 자료형 객체를 shopping cart에 담기
135
136     }
137
138 }
```

```
136     cart.add_item(my_item);
137     cout << "쇼핑카트에 상품이 담겼습니다." << endl;
138     cout << endl;
139 }
140
141 //shopping_cart의 객체를 order_management에게 주문 정보 넘겨줌
142 //이후 order_management는 order를 생성하여 order_log의 log_array에 넣음
143
144 }
145
146 //결제하는 과정 -> account_management에 접근하여 매출 증가시킴
147 void purchase_management::purchase_pay(int payment, bool payment_type) {
148
149     this->account_management->purchase_pay(payment, payment_type);
150
151
152
153 }
154
```

100% ④ 문제가 검색되지 않음 ← → 줄: 1 문자: 1 템 CRI