**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.

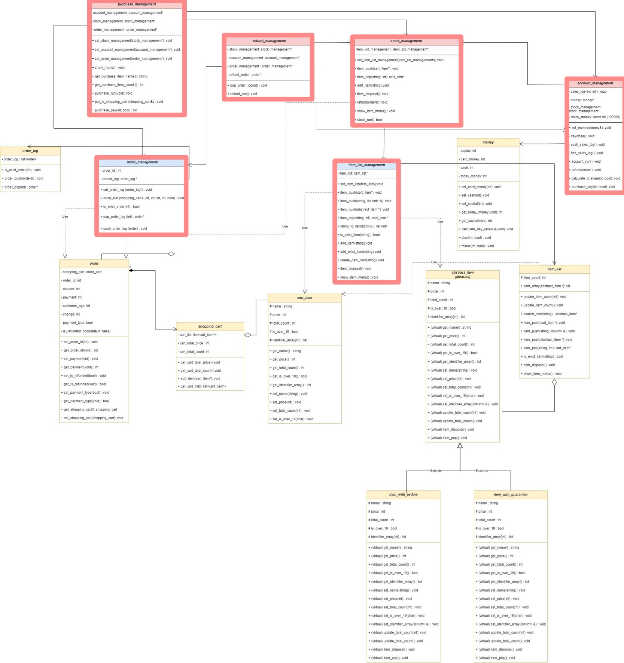
매우 복잡하게 얽혀있는 각 기능의 상호작용을 효율적으로 관리할 수 있도록 설계의 방향성을 잡았고 설계 과정에서 고려한 주요 쟁점은 다음과 같다.

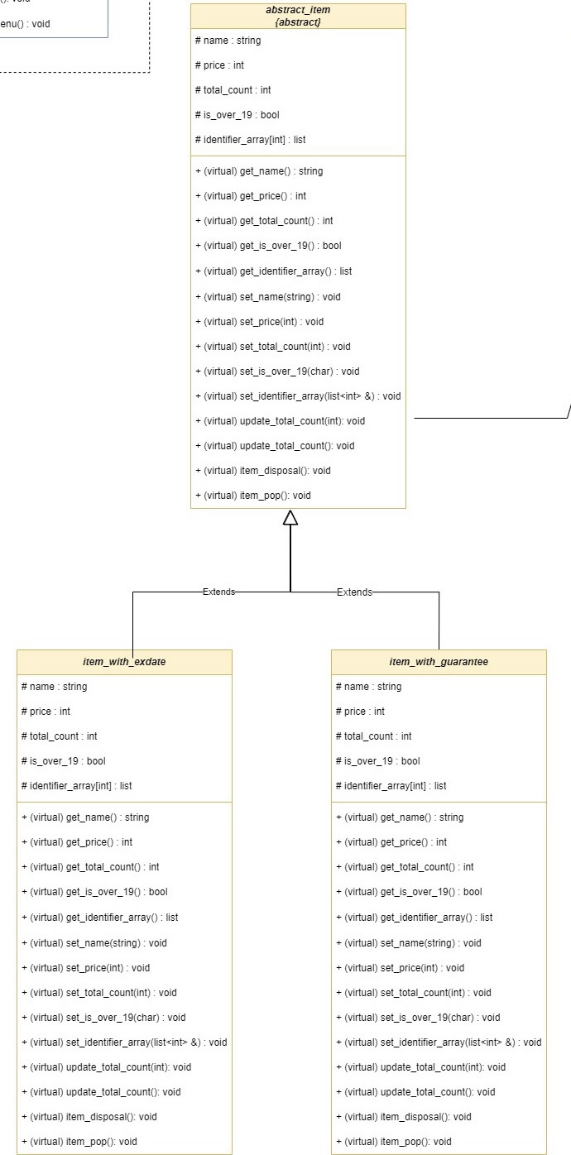
1. **OOP적 관점을 토대로 각 클래스를 어떻게 분업하는가?**
2. **상품을 어떻게 추상화해야 하는가?**
3. **상품을 어떻게 저장하여 관리하는가?**
4. **핵심 기능이라 할 수 있는 판매 상황에서, 손님에게 어떤 식으로 여러 개의 구매 상품 목록을 받아오는가?**

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

1. **설계 소개**
2. **OOP적 관점을 토대로 각 클래스를 어떻게 분업하는가?**

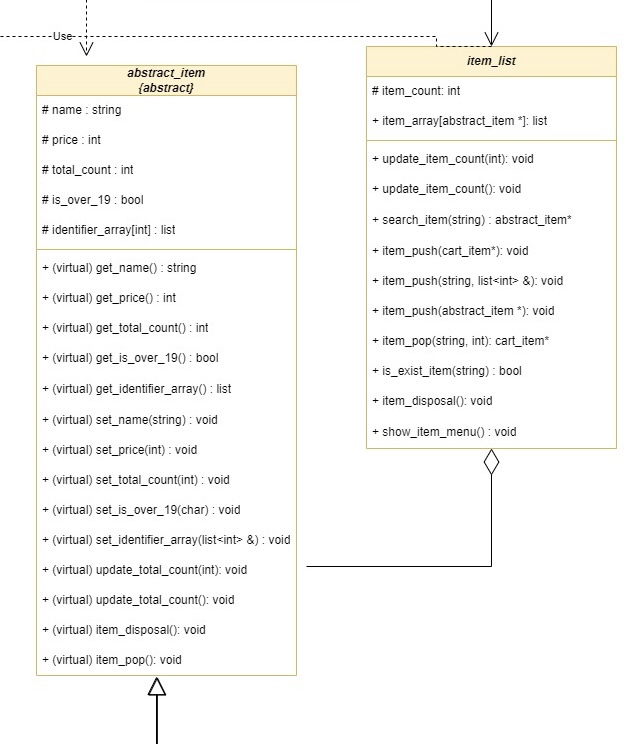
구매 (Purchase) / 주문(order) / 환불 (Refund) / 회계 (Account) / 재고 (Stock) / 상품 (Item) 6가지에 대해 management를 두어 각각을 담당하여 관리하도록 하고, 상호 작용에 있어서도 이 5가지 management를 통해서만 정보를 주고 받도록 설계하였다. 이 방식을 택함으로써 다음과 같은 이점을 얻을 수 있다.

* 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 책임를 가질 수 있다. 예를 들어 purchase\_management를 통해서는 손님과의 상호 작용의 역할만을 수행할 수 있다. 이후 재고의 변동은 stock\_management가, 자본의 변동은 account\_management만이 관여할 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.

1. **상품을 어떻게 추상화해야 하는가?**

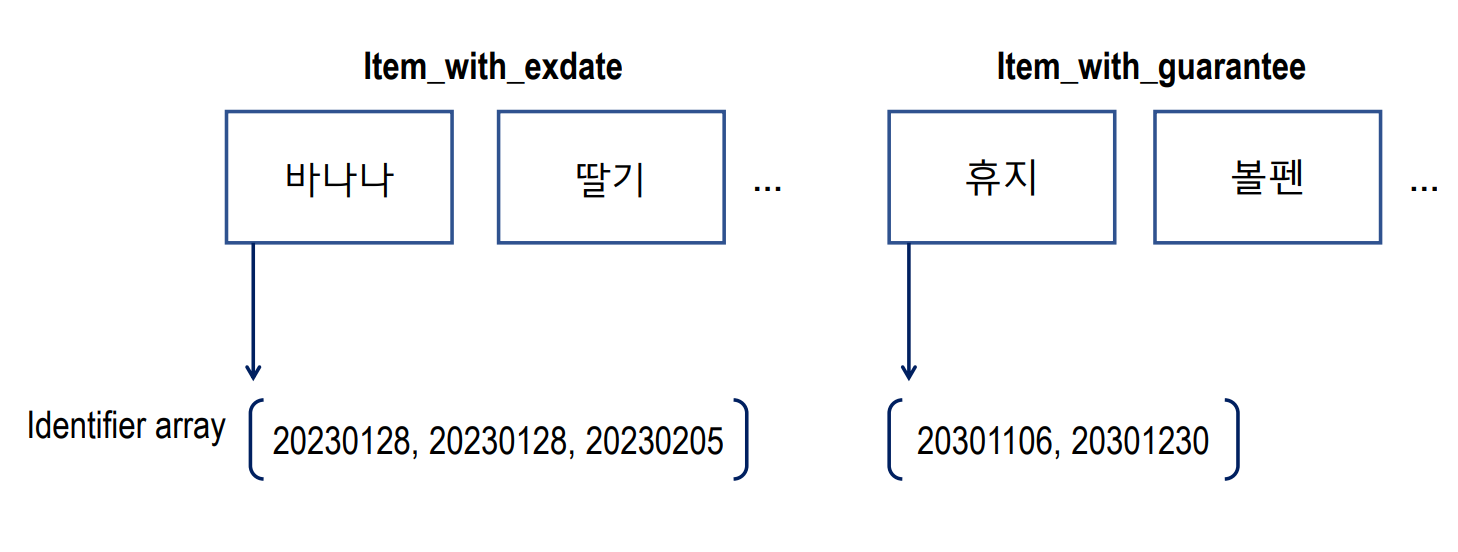
모든 상품이 동일한 정보 구조를 가지고 있지 않으므로, 모든 상품의 공통점을 토대로 추상화하여 큰 틀로 두고 이를 상속받는 하위 클래스를 두는 구조를 이용하기로 했다. 우리 설계에서는, 재고 관리를 하는 상황에서 유통 기한의 유무가 상품을 다루는 데에 있어 크게 좌우될 것을 고려하였다.  
 오른쪽 그림과 같이 item 전체를 가리키는 추상클래스인 abstract\_item를 상위에 두고, 유통기한이 있는 상품(item\_with\_exdate)과 없는 상품(item\_with\_guarantee)이 item 추상클래스를 상속받도록 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.

1. **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.

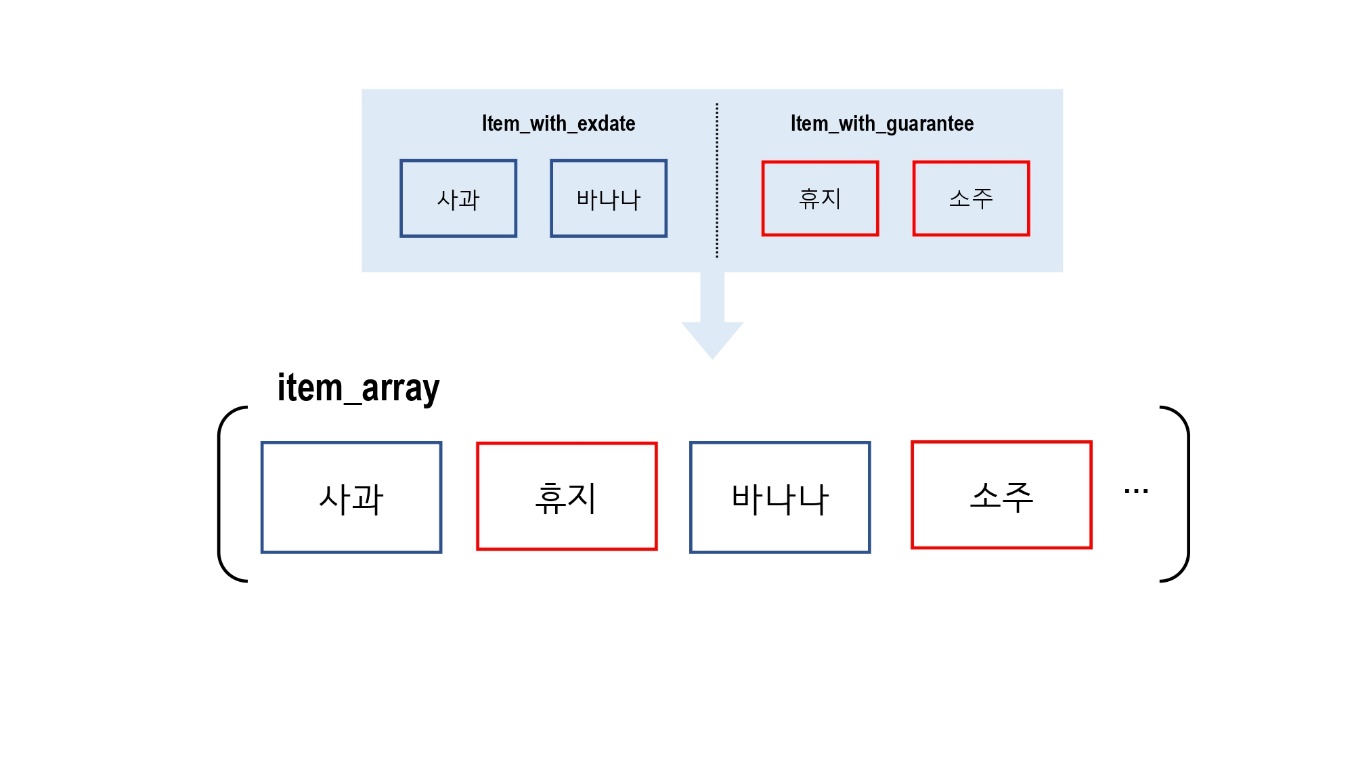
위의 예시에서는, 바나나 객체 내의 identifier\_array에는 20230128, 20230128, 20230205 세 개의 날짜가 저장되어 있다. 이는 바나나라는 품목은 유통 기한이 2023년 1월 28일인 것 2개, 2023년 2월 5일인 것 1개, 총 3개가 재고로 저장되어 있다는 것을 의미한다. 마찬가지로, 휴지 객체 내에 있는 identifier\_array를 보면 보증기한이 2030년 11월 06일인 휴지 1개, 2030년 12월 30일까지인 휴지가 1개씩, 총 2개가 존재하는 것을 알 수 있다.   
 정리하자면, 클래스로 찍어낸 객체는 품목이 되고, 각 상품의 종류 내에 identifier\_array는 각 멤버변수 안에 있는 배열의 원소 하나하나는 각각 실제 상품을 의미하게 된다.

이러한 방식을 선택함으로써 우리 설계가 취한 주요 이점은 다음과 같다.

* 효율적인 관리 : 유통기한과 보증기한이 지난 상품을 폐기하는 상황, 상품을 입고하는 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**

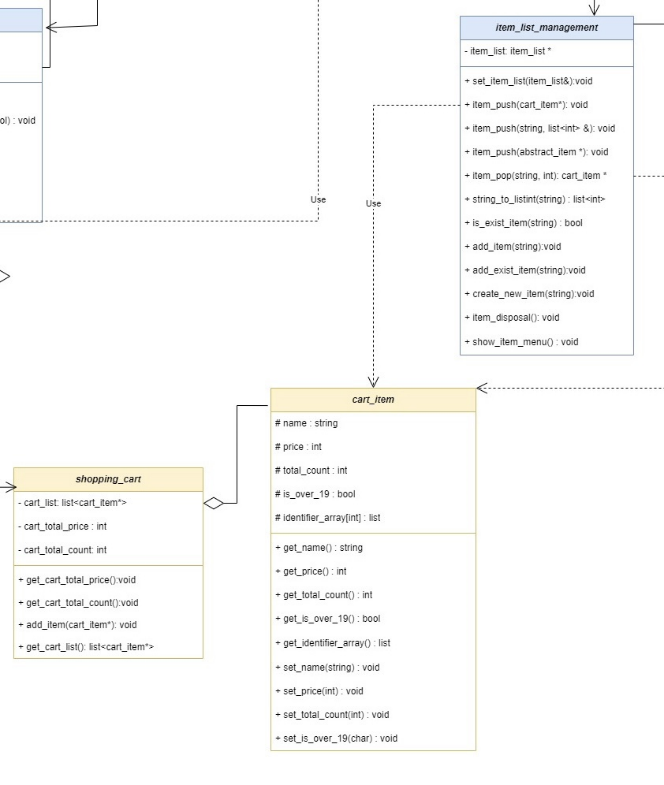
상품의 저장은 검색에 있어서 용이하게 하기 위해, Item\_with\_exdate 객체와 item\_with\_guarantee 객체를 구분 없이 한 컨테이너에 두기로 했다. 이는 2-1)에서와 같이, item\_with\_exdate와 item\_with\_guarantee가 그 상위의 추상 클래스인 abstract\_item을 공통으로 상속받는 클래스여서 가능해졌다.



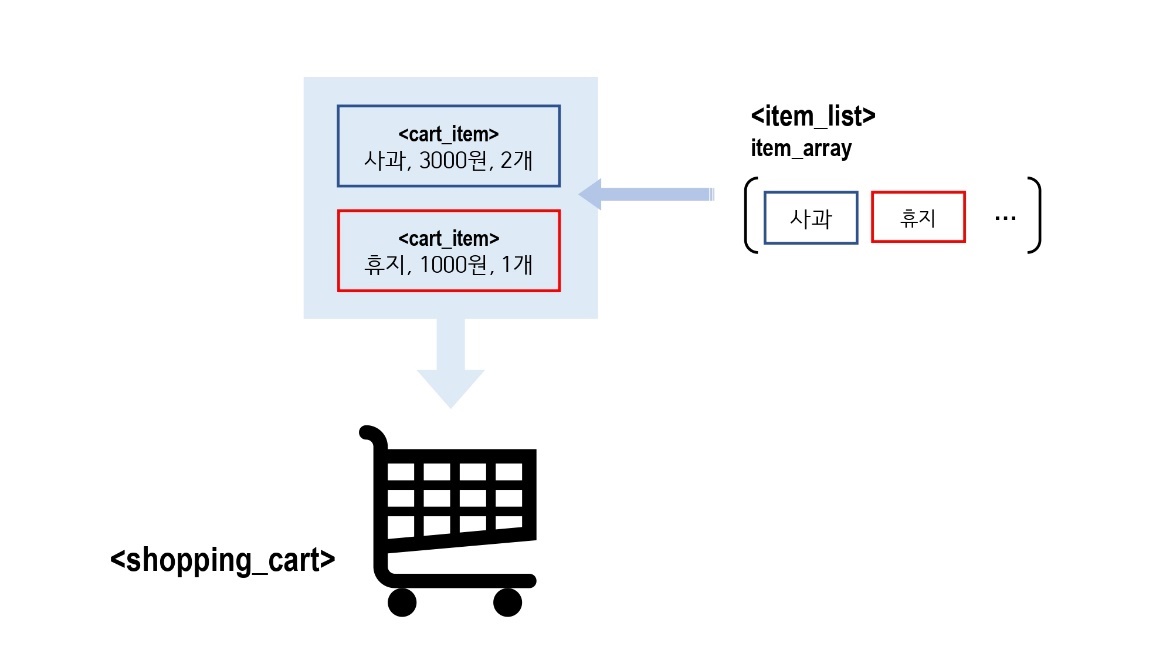
이때 단순히 Array 형식을 사용하기보다 STL 컨테이너인 List를 사용한 이유는 사이즈가 고정적이면서도 메모리와 성능의 최적화할 수 있기 때문이다. 편의점과 같은 작은 상점은 재고를 보관할 수 있는 공간이 한정되어 있고 상품의 총 종류와 그 개수가 크게 가변적이지 않기 때문에 List를 사용하는 게 더 적절할 것이라 판단하였다.

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.

1. **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.

1. **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. **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\_price를 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 때 마다 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.

1. payment\_type, 즉 지불 수단이 현금이면 calculate\_change()를 통해 change (거스름 돈) 를 계산한다.
2. 구매로 인해 **(1) 구매 기록의 추가**, **(2) 재고의 감소**와 **(3) POS기 회계 상의 변경**이 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. **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 읽어 온다.
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 상**의 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 폐기**와 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.

1. **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]**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

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]**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

* 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.

**텍스트이(가) 표시된 사진

자동 생성된 설명**

* 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 판매했으니 하루 매출을 정산한다. [MAIN]에서 4를 입력한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* 마감 정산 후 매출이 한 개 이상 기록되어야 매출 조회가 가능하다. 따라서 1. 마감 정산을 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.

텍스트이(가) 표시된 사진

자동 생성된 설명

* 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 하러 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.

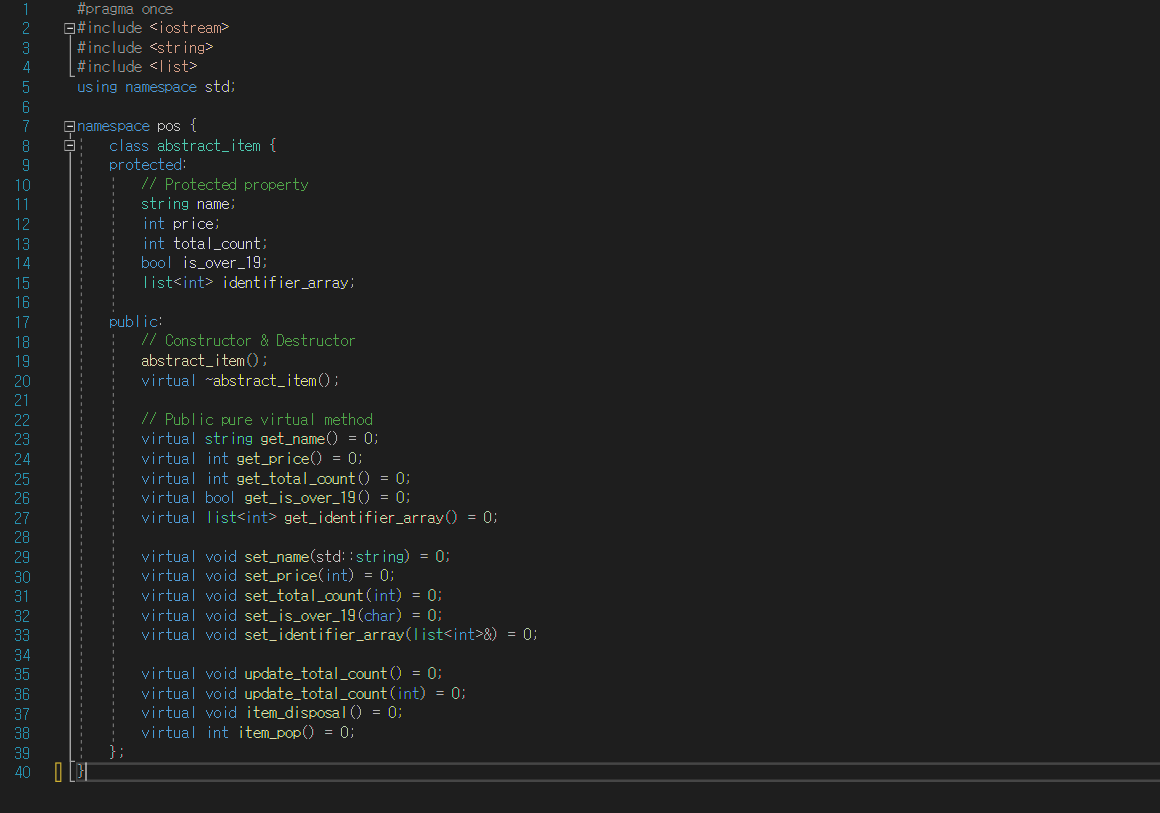
**[POS 시스템 종료]**

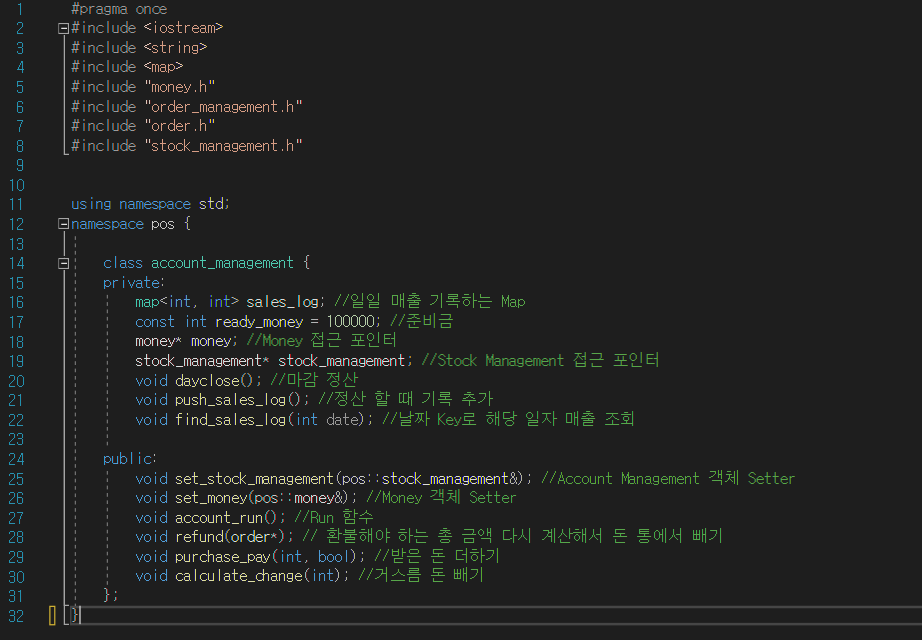
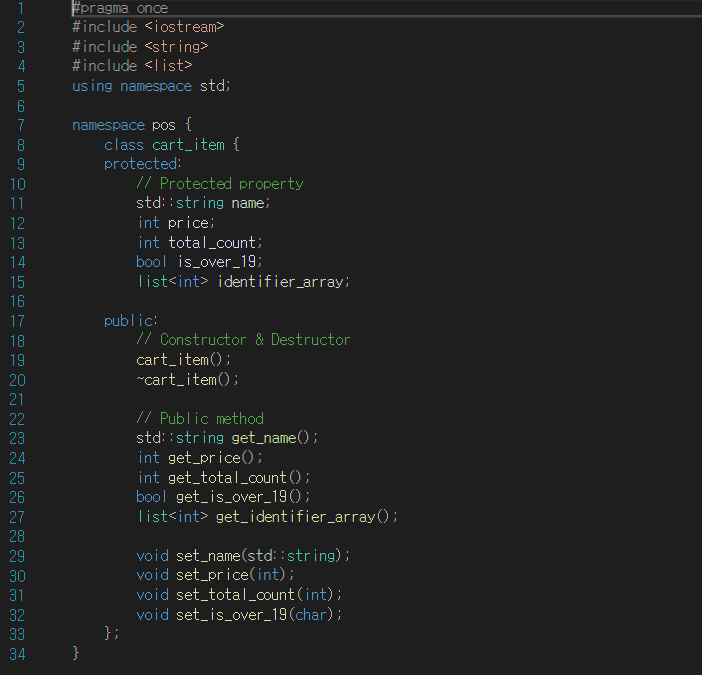
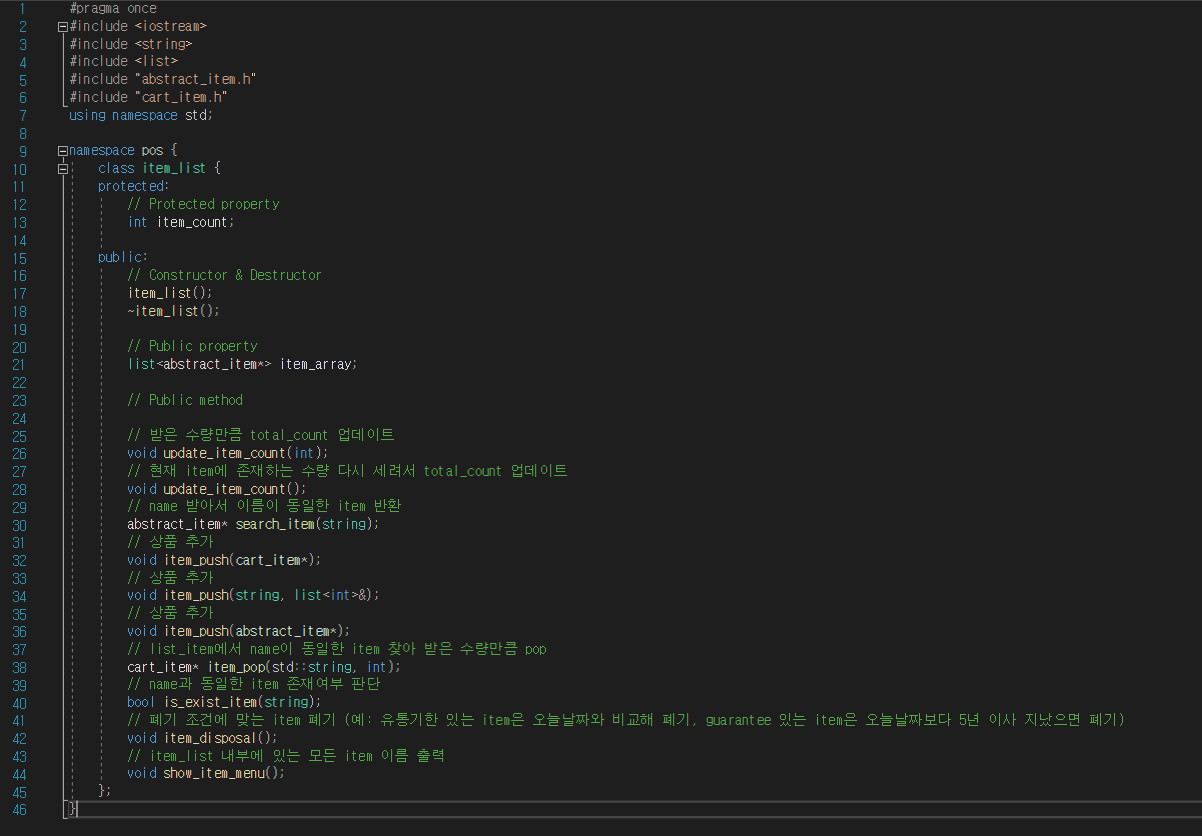
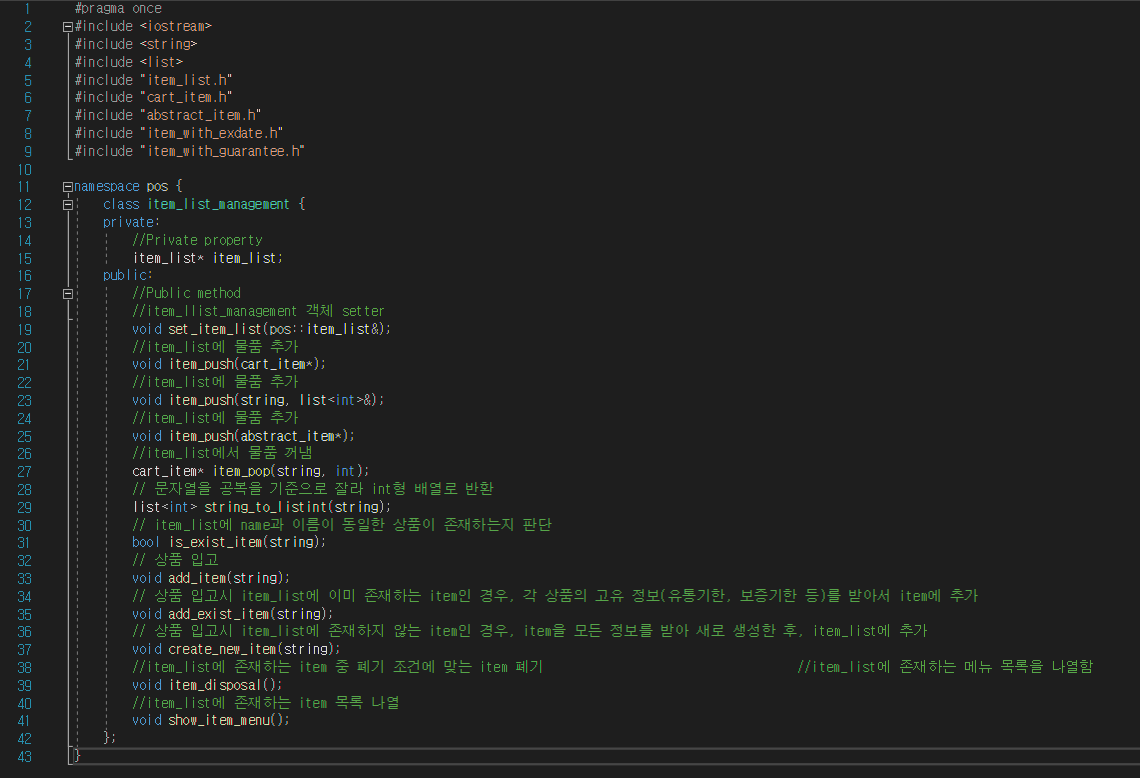
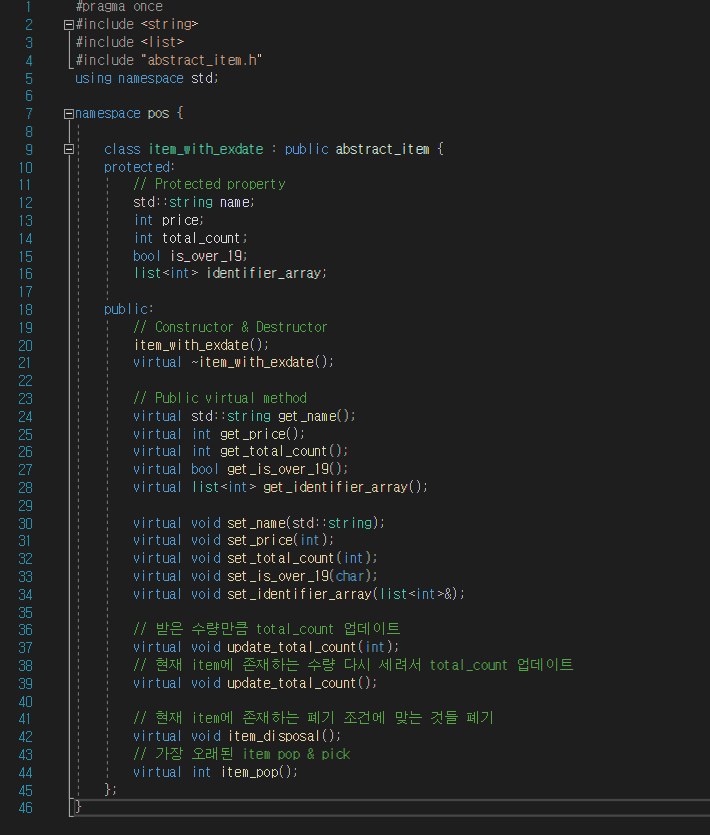
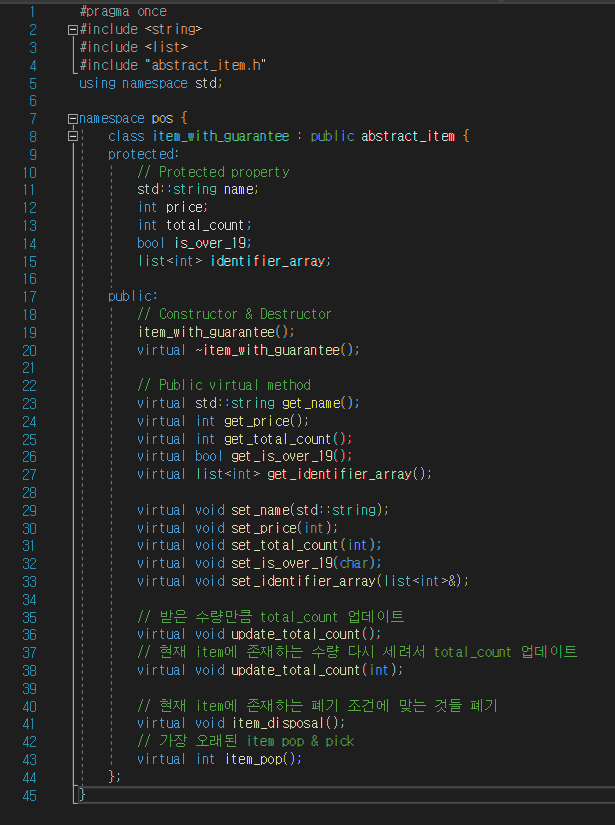
[MAIN]에서 5를 입력하면텍스트이(가) 표시된 사진

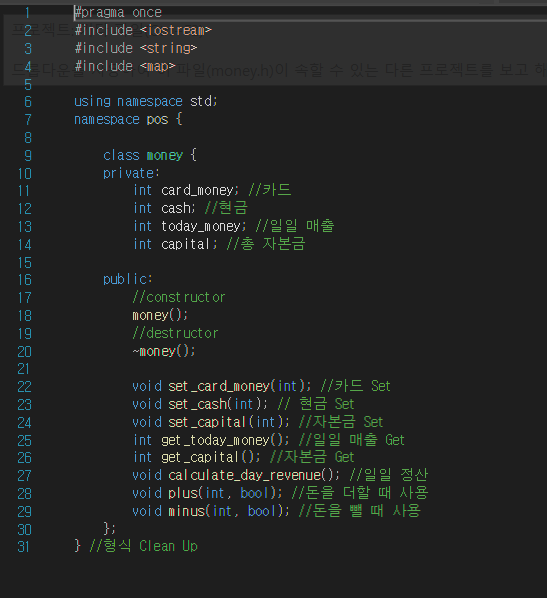
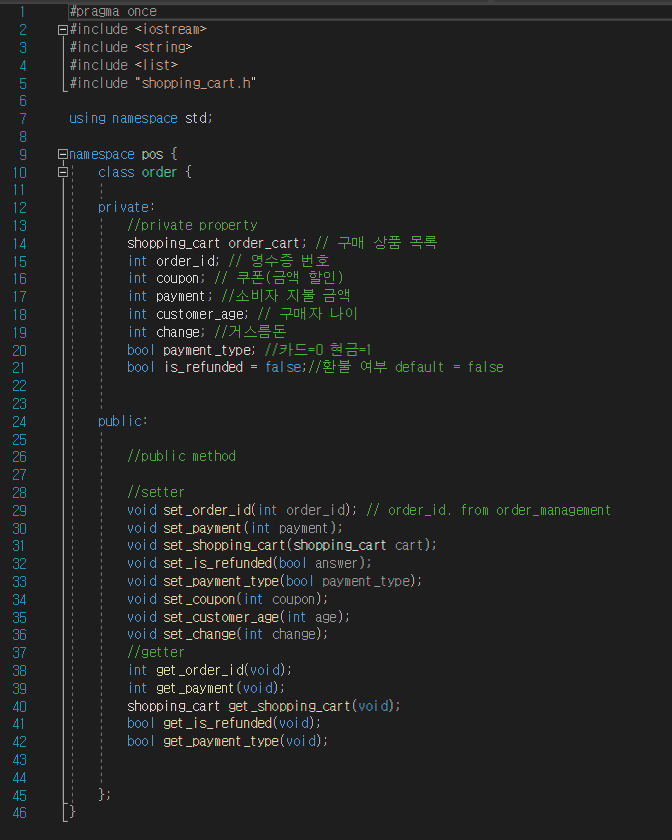
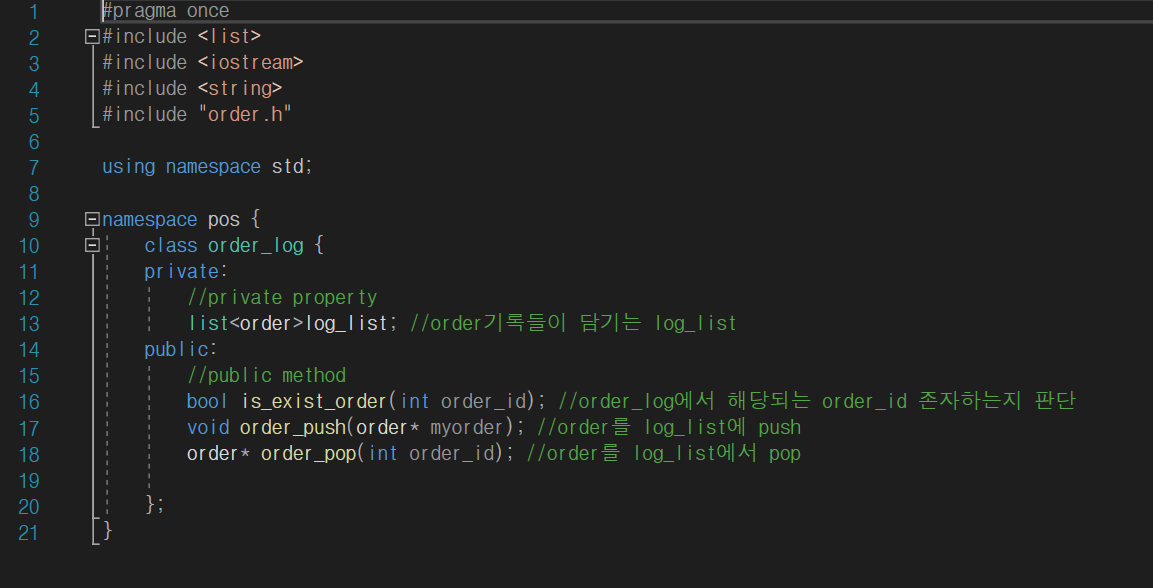
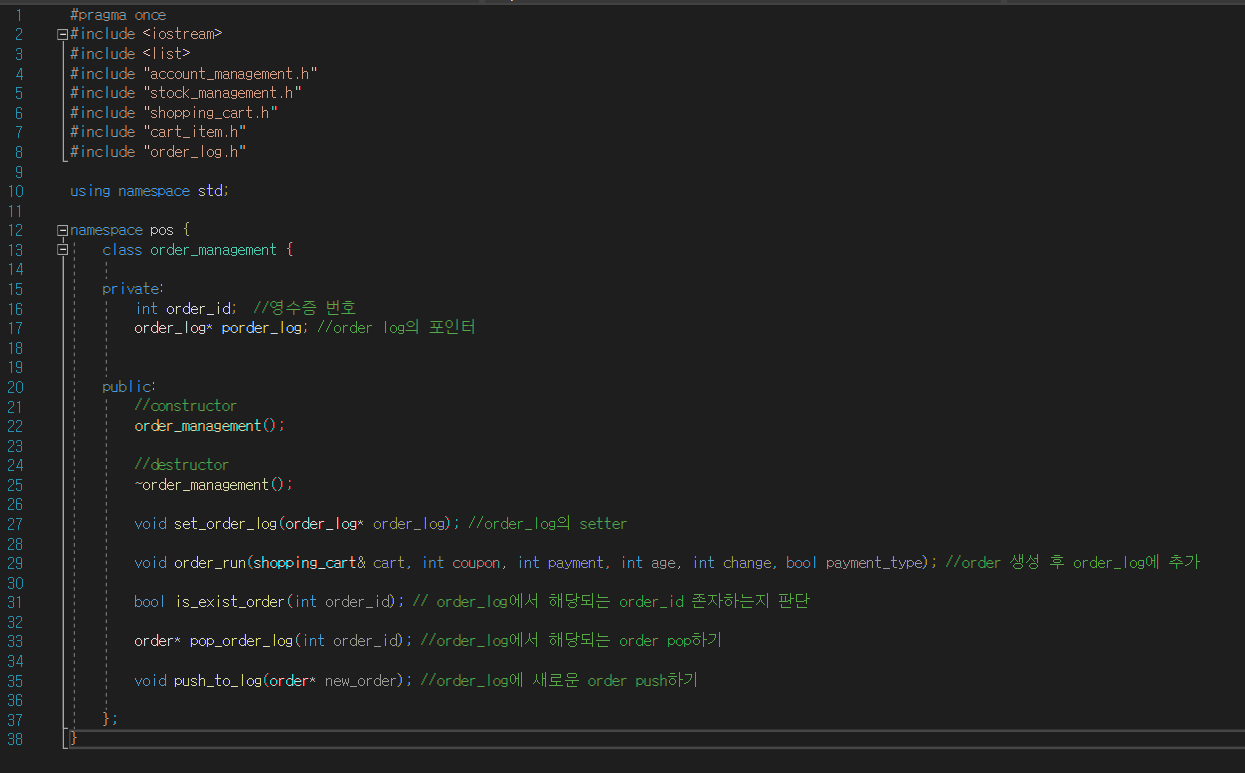
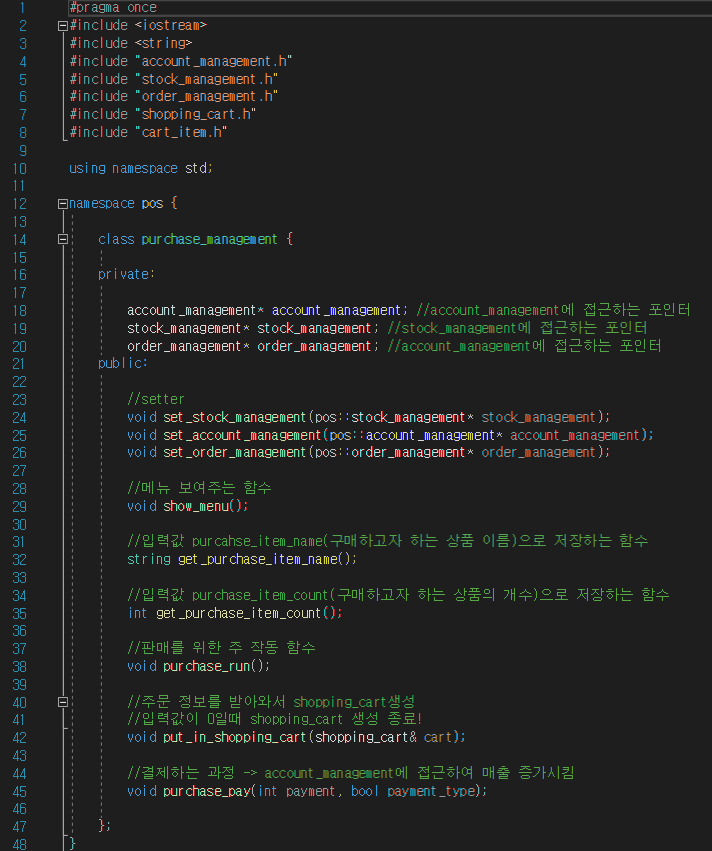
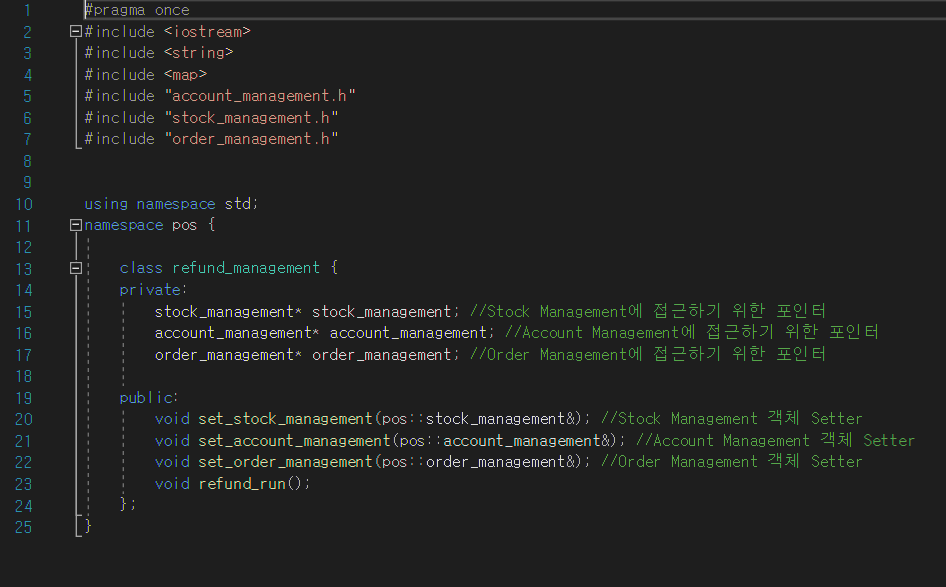
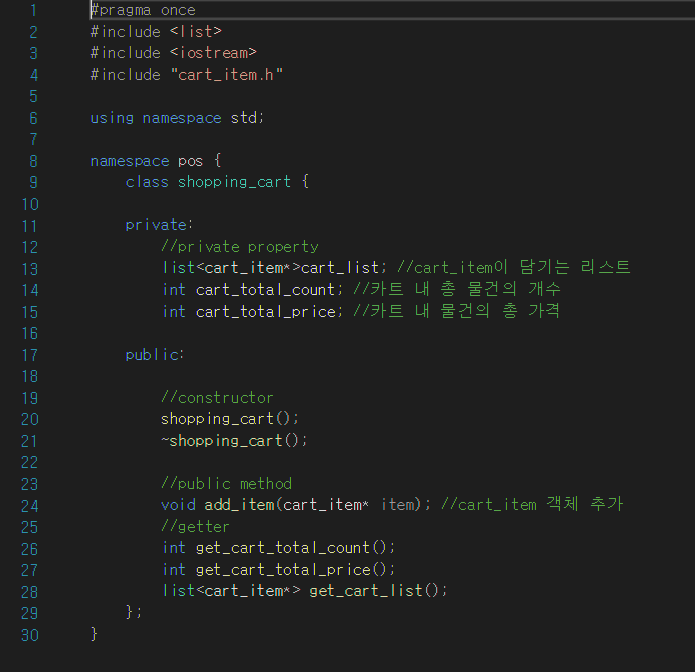
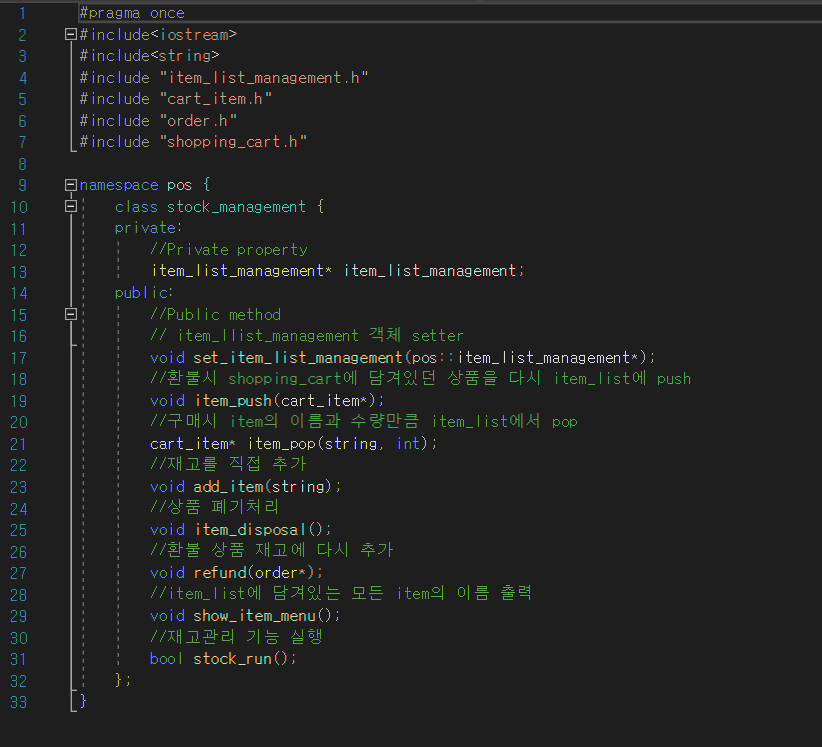
자동 생성된 설명 POS 시스템이 종료된다.

**\*\*이후 내용은 실제 코드 구현 내용입니다.**

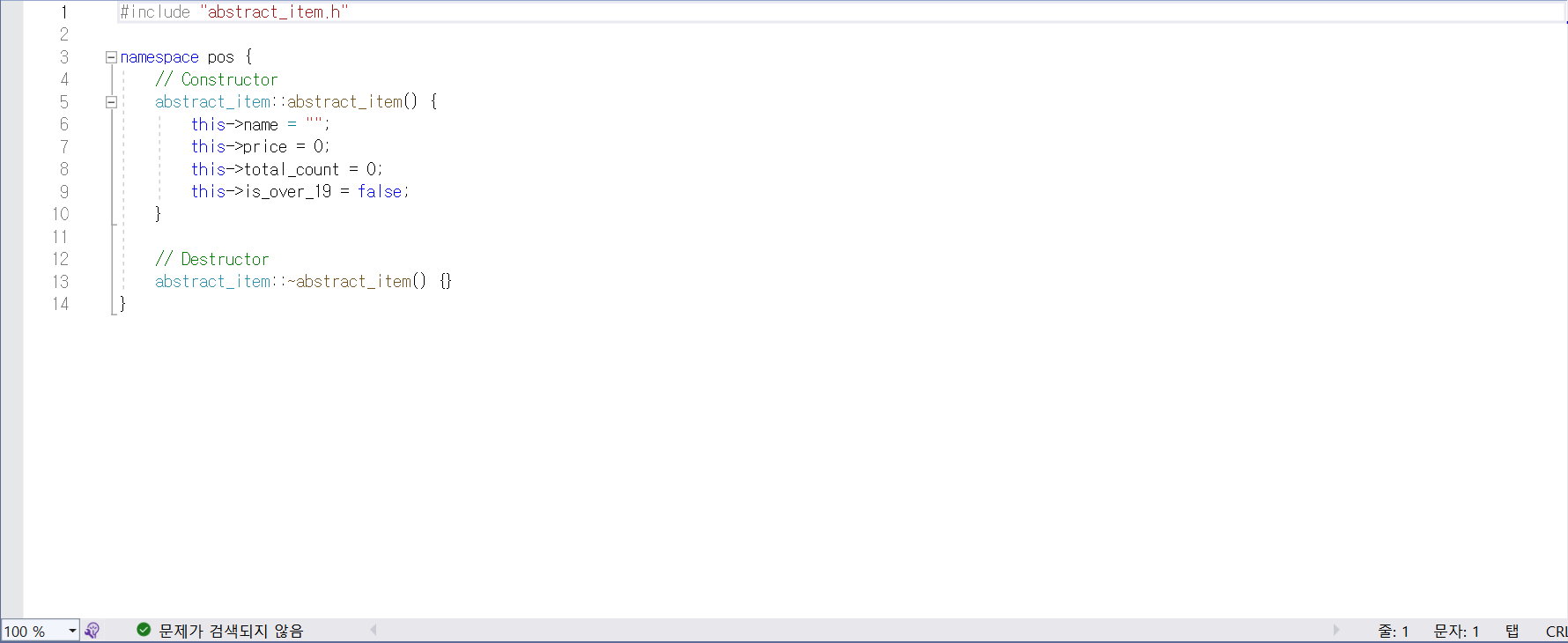
**<헤더파일>**

** Abstract\_item**

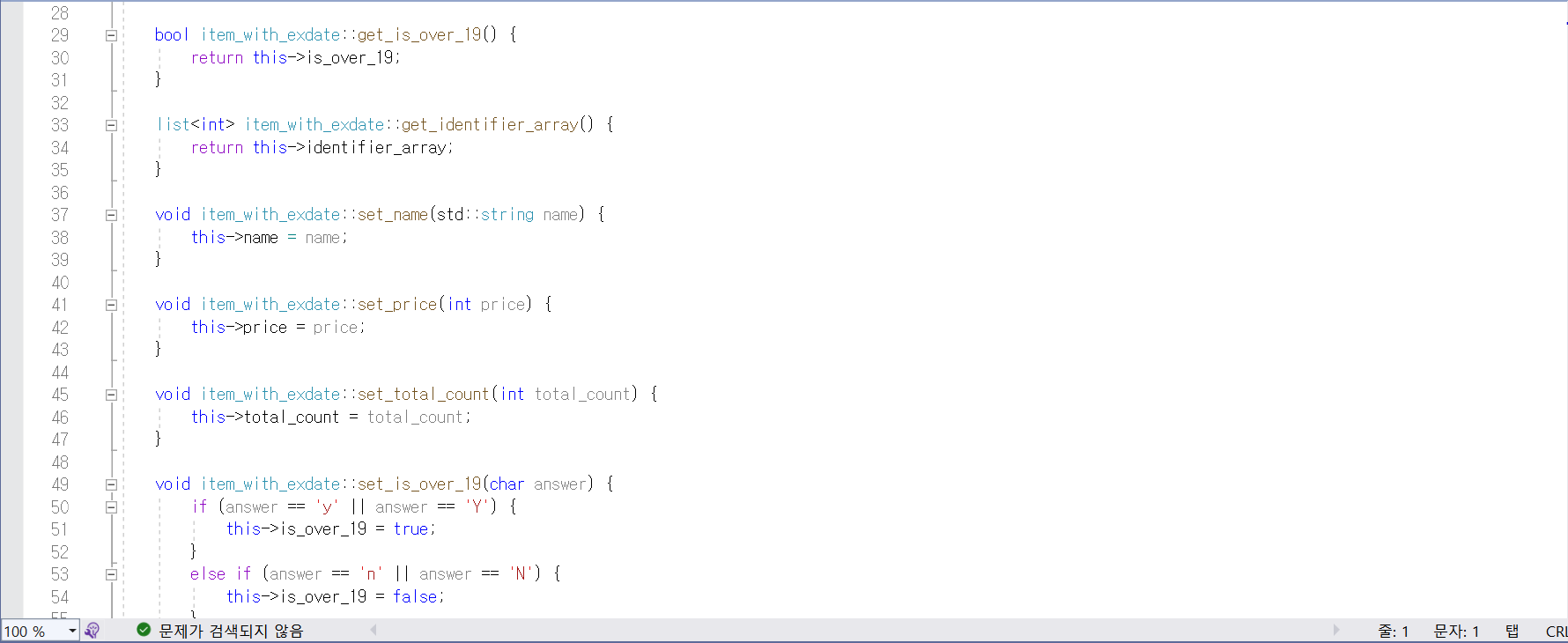
**account\_managementcart\_itemitem\_listItem\_list\_managementitem\_with\_exdateitem\_with\_guarantee**

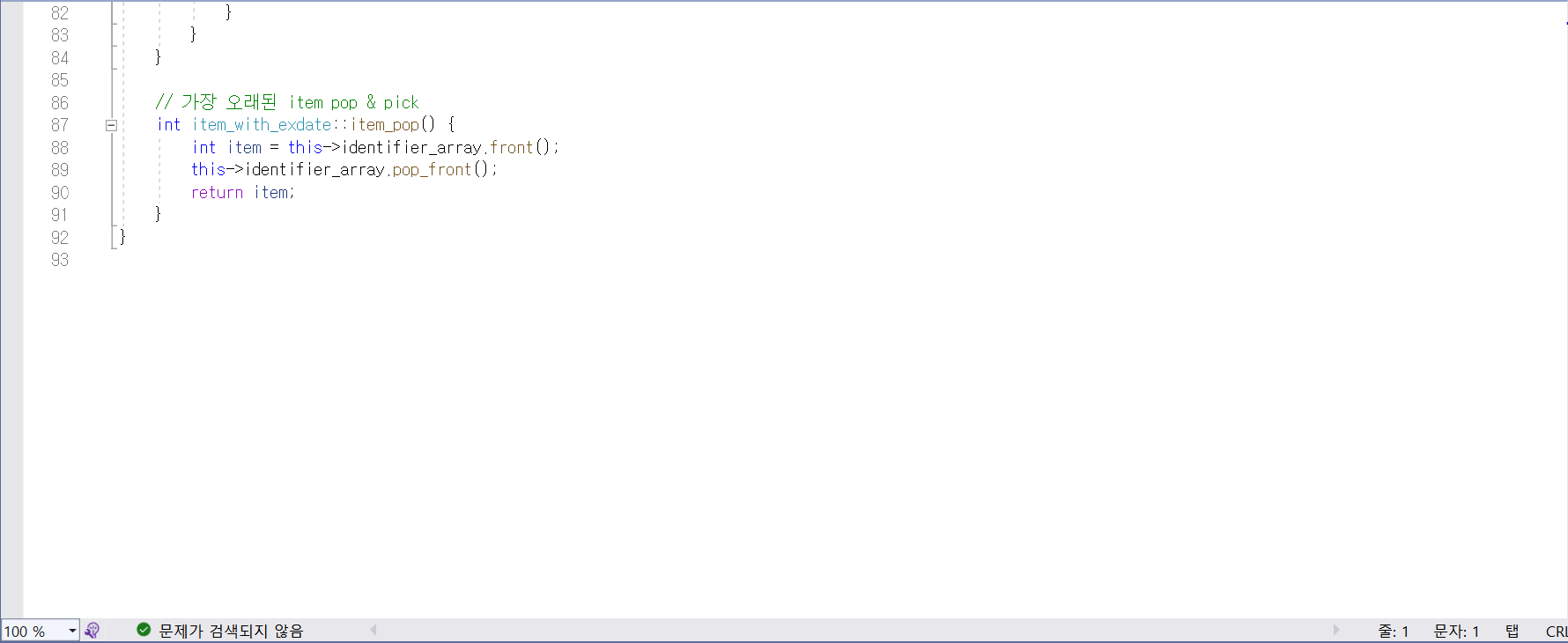
**moneyorderorder\_logorder\_managementpurchase\_managementrefund\_managementshopping\_cartstock\_management**

**abstract\_item.cpp**



**item\_with\_exdate.cpp**

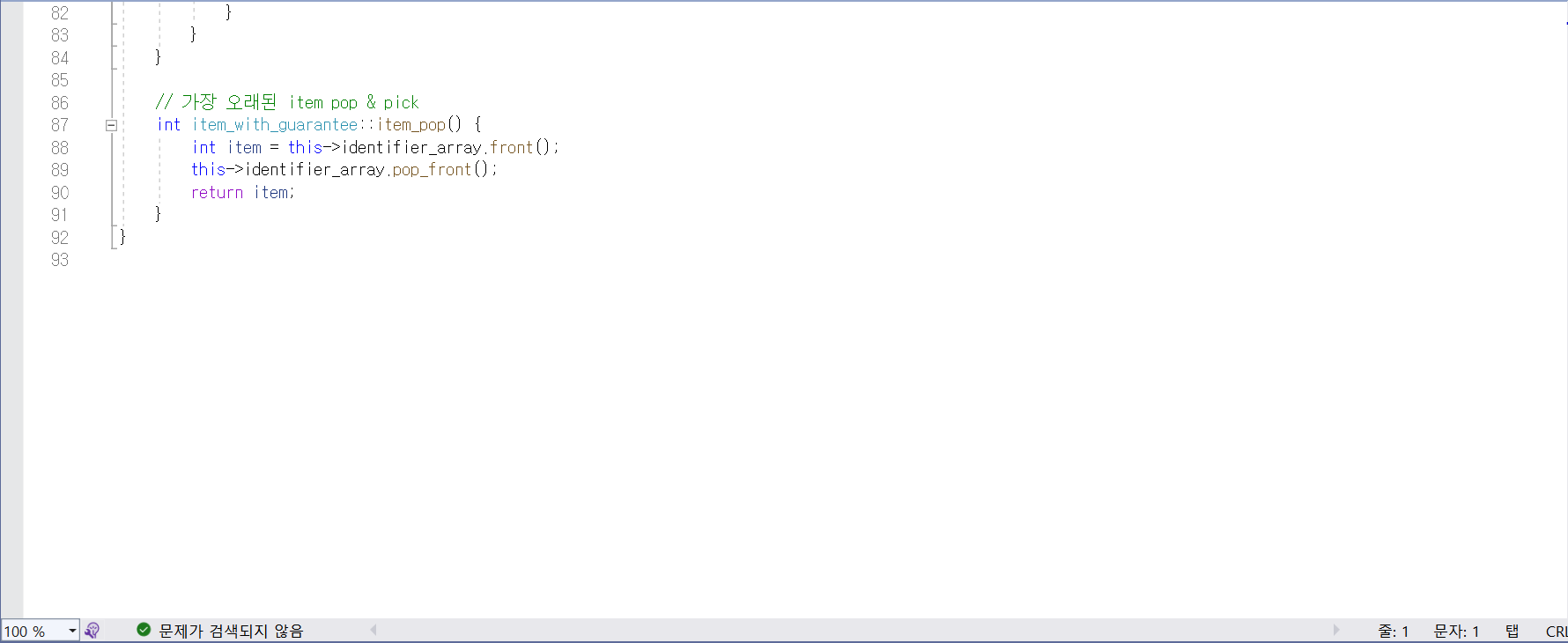
**텍스트이(가) 표시된 사진

자동 생성된 설명**

**item\_with\_guarantee.cpp**

**텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명**

**item\_list.cpp**

**텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명**

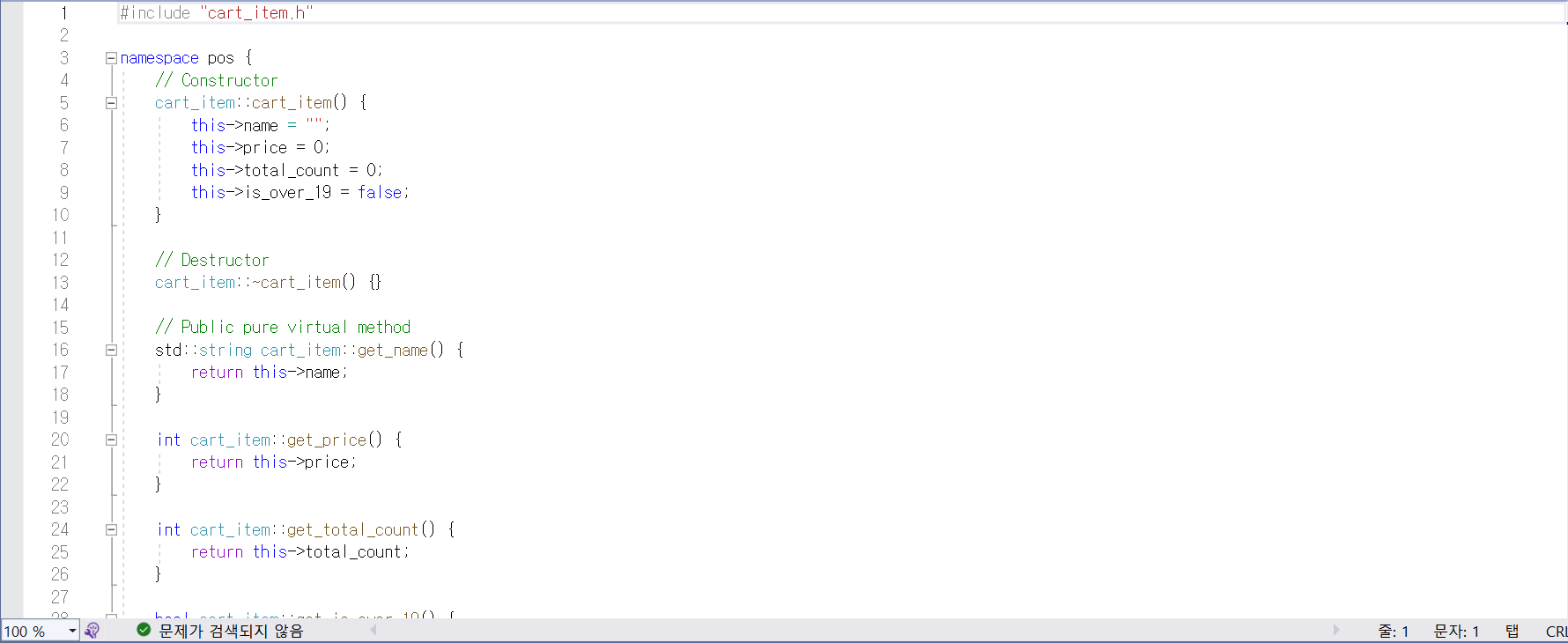
**shopping\_cart.cpp**

**텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

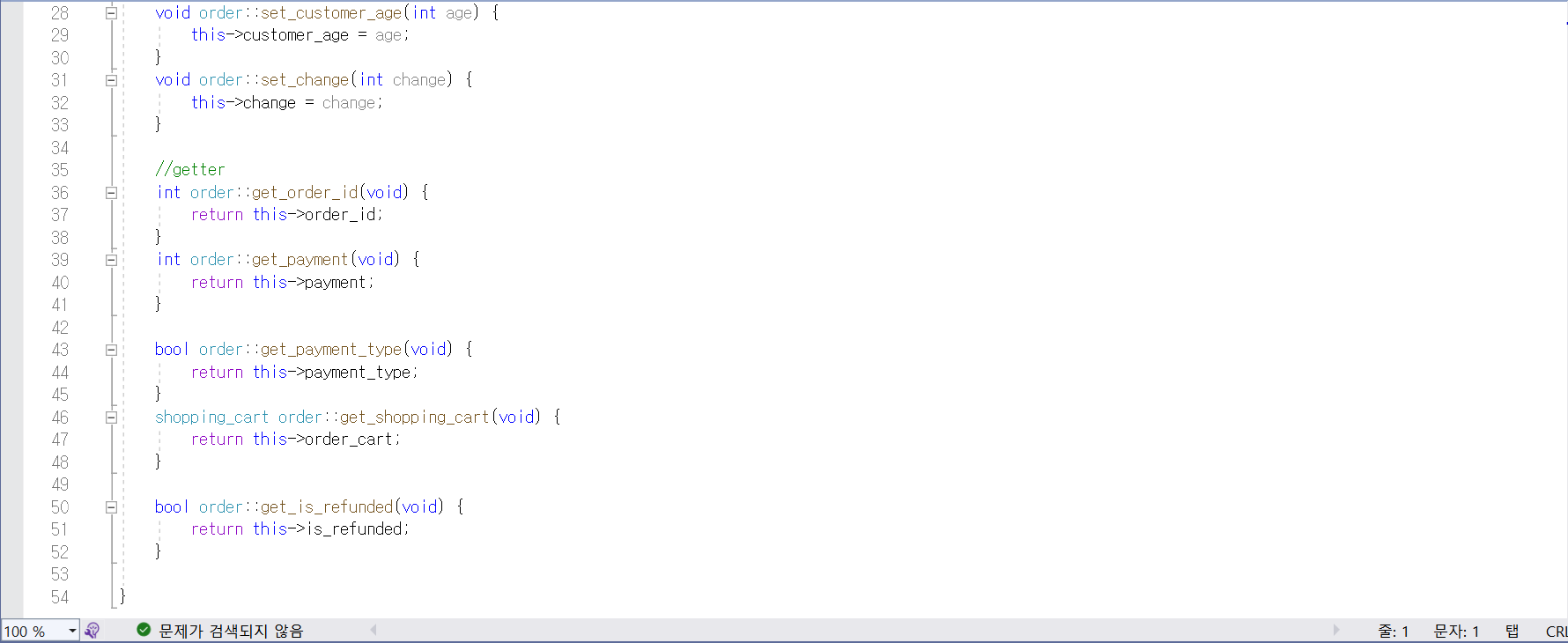
자동 생성된 설명**

**cart\_item.cpp**

텍스트이(가) 표시된 사진

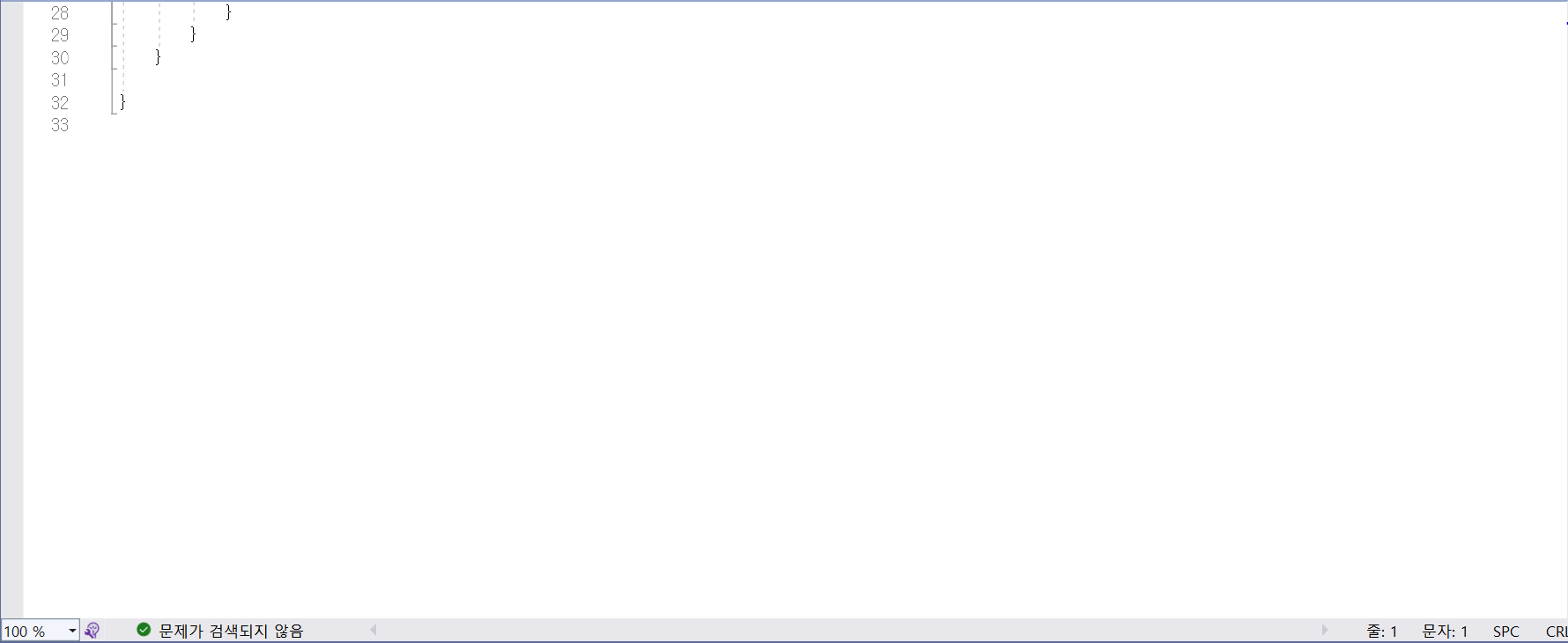
자동 생성된 설명

**order.cpp**

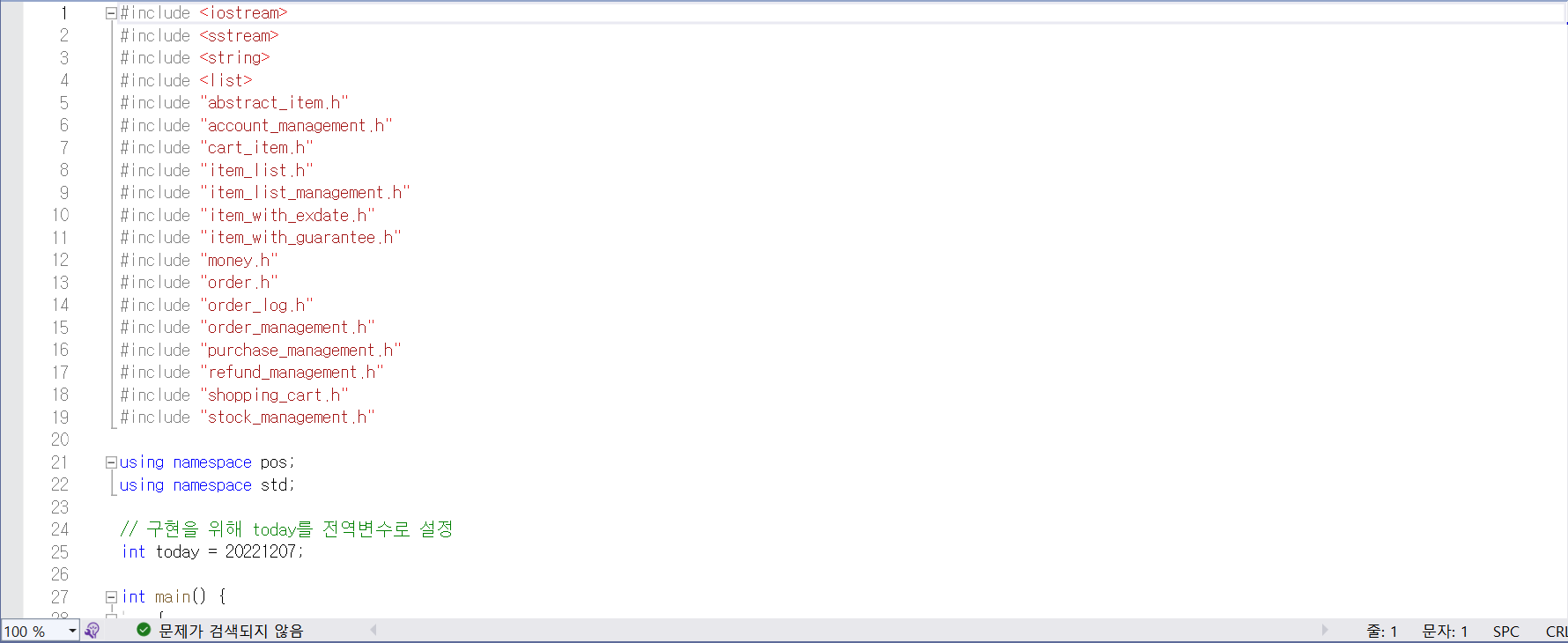
****

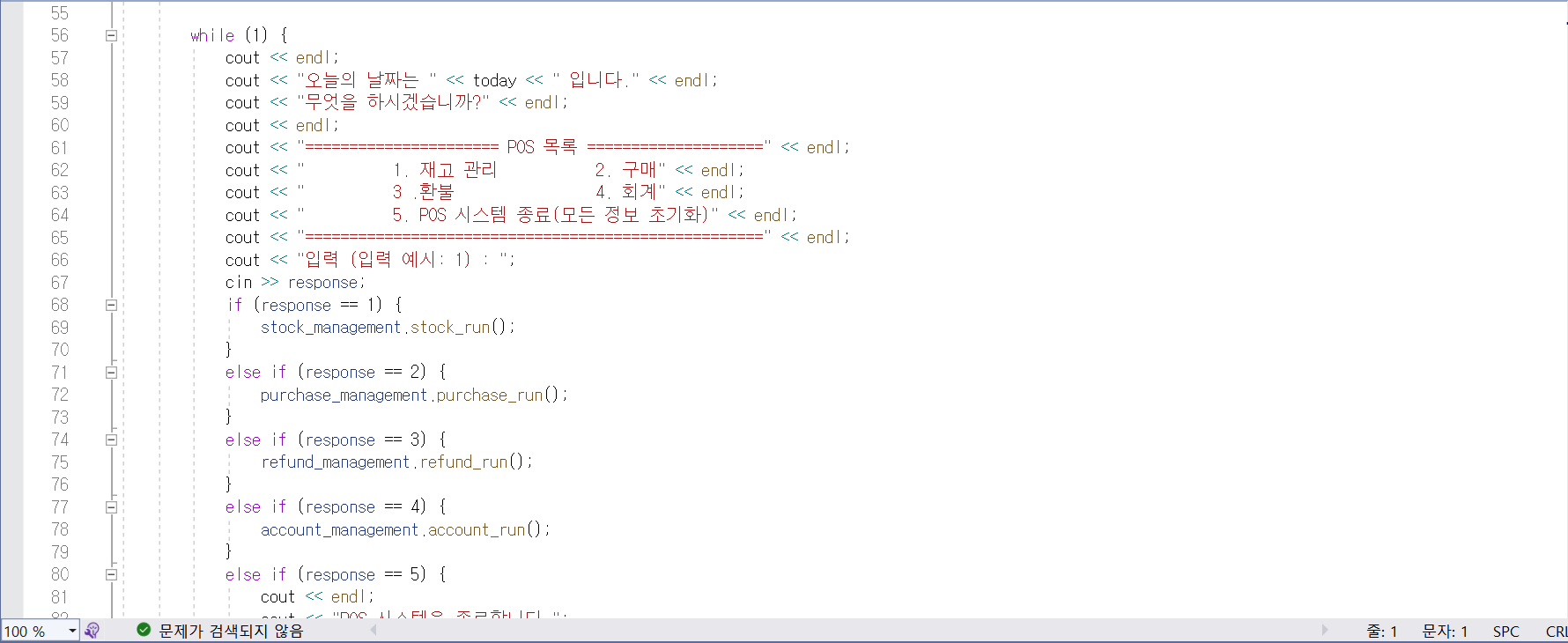
**order\_log.cpp**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**main.cpp**

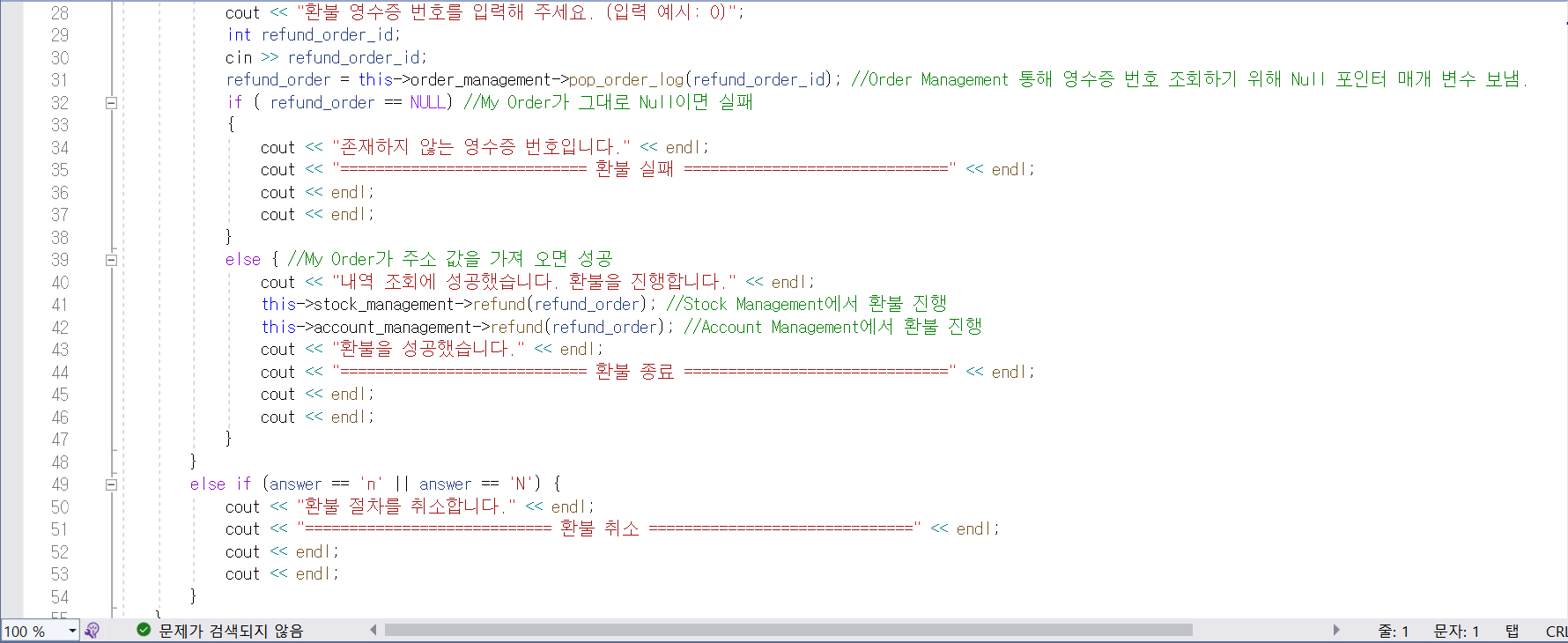
텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

**refund\_management.cpp**

텍스트이(가) 표시된 사진

자동 생성된 설명

**stock\_management.cpp**

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

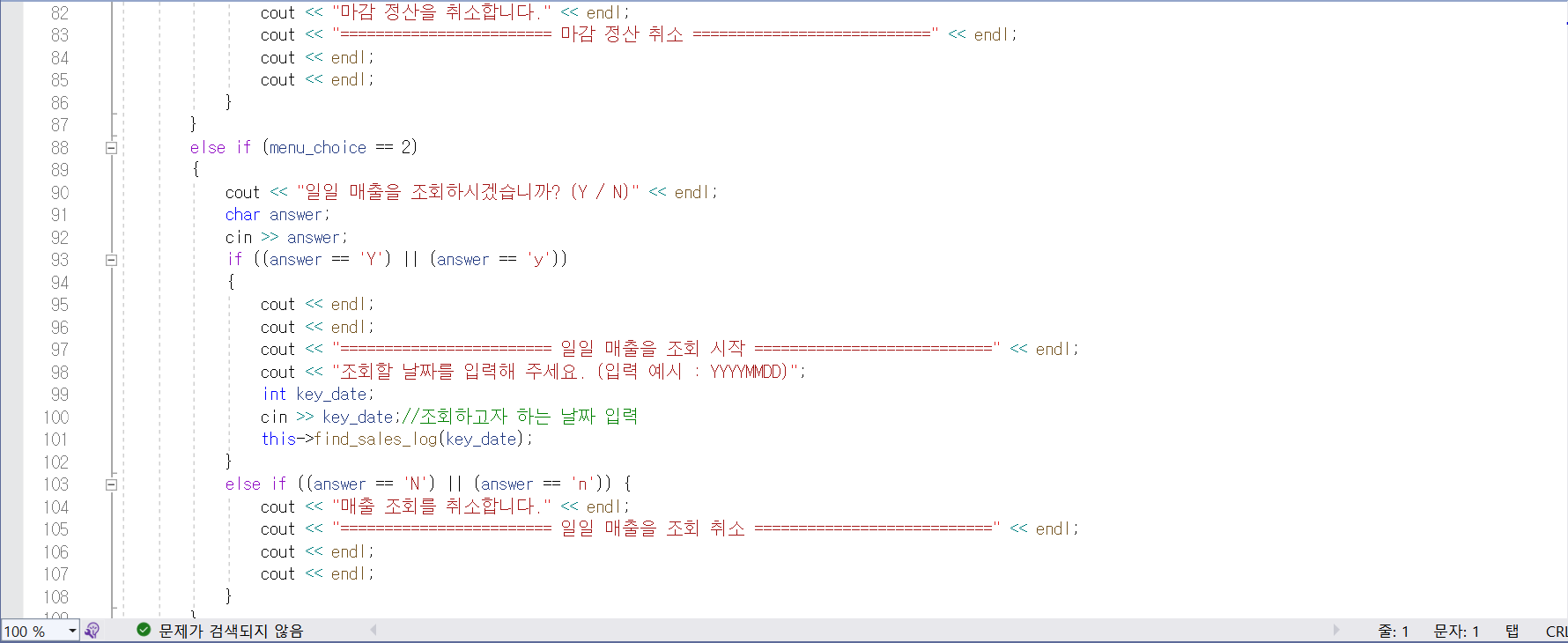
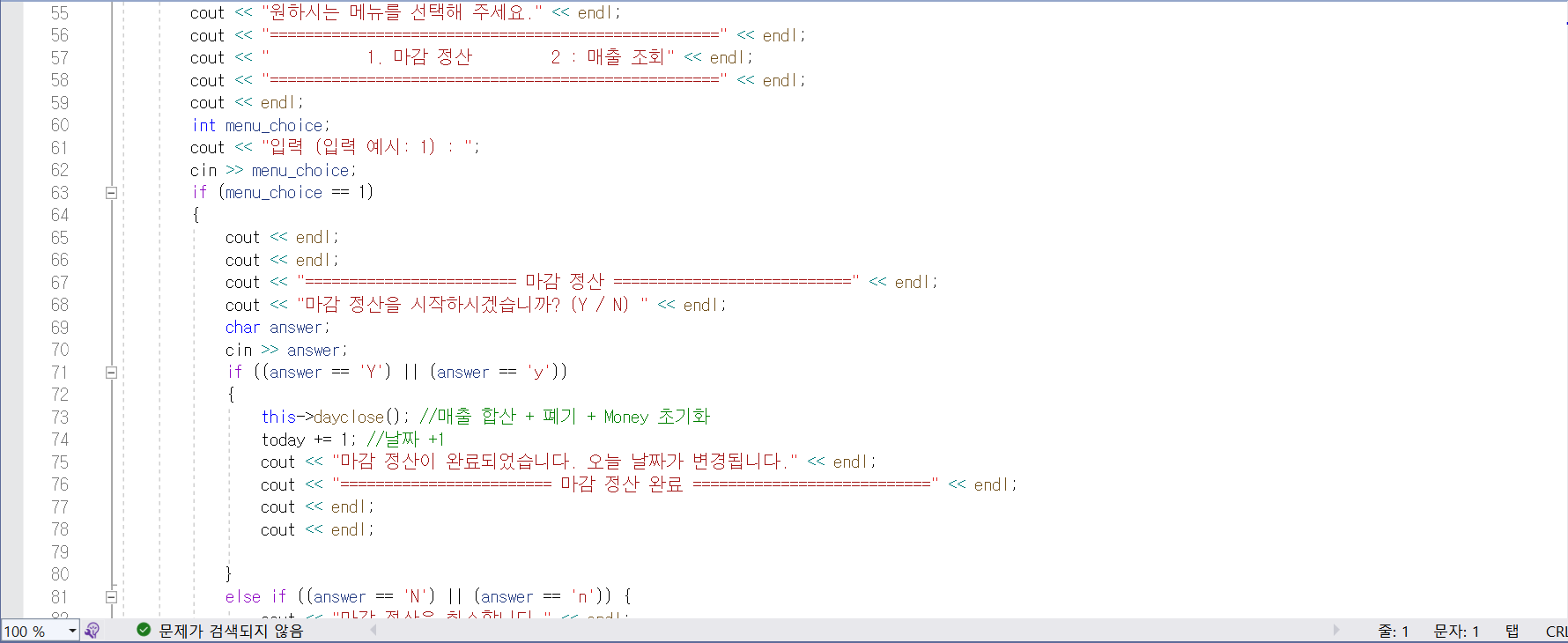
자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

**account\_management.cpp**

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

**money.cpp**

**텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명**

**purchase\_management.cpp**

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명