Assignment 3 – Mart Simulation

컴퓨터소프트웨어 2020018159 이종준

## 0. 실행환경

1) OS: Windows 10

2) JAVA 버전: 15.0.2

3) IDE: VS Code

**# 디자인 패턴에 관한 내용의 경우, 보라색으로 표시하였습니다.**

## 1. MartSimulation.java

BufferedReader와 FileReader를 이용하여, 기본 위치에 있는 CustomerWallets.txt와 Mart.txt를 가져오도록 하였다 (./CustomerWallets.txt, ./Mart.txt). 또한, Mart 클래스에 현재 시각(timeNow)을 LocalDateTime을 이용하여 명세에 맞게 설정해주었다. 가져온 CustomerWallets와 Mart 정보를 통해, 새로 선언한 배열에 각각의 정보들을 초기화하였다 (split(“, ”)로 정보 구분). 입력을 받아 managerMode 와 customerMode 로 분기시켜주었다. while 문 안에 try catch 구문을 넣어, exception 이 발생할 경우, 오류 메시지를 출력하고 계속해서 해당 동작을 반복하도록 하였다 (해당 로직은 이 Assignment 내 모든 exception handling에 동일하게 사용되었다).

1) printMainMenu

➔ 메인 메뉴 번호를 입력 받아 반환하는 함수. main 함수에서 사용된다. 1~3 이외의 범위의 입력이 들어오면 예외를 발생시키도록 하였다.

2) managerMode

➔ 0 을 입력하기 전까지 무한 반복하며, 메뉴 입력을 통해 addInventory 와 replaceExpired 로 분기시켜주었다.

3) addInventory

➔ 재고를 추가할 상품의 번호와 그 개수를 차례대로 입력 받고, Mart 클래스의 addInventory 함수를 통해 해당 상품의 재고를 늘려준다. Observer 패턴이 적용되어,

해당 상품 구매대기를 신청한 고객의 장바구니에 알맞은 개수로 들어갈 수 있게끔 하였다. 0 을 입력하면 managerMode 로 돌아가는 기능을 추가하였다.

4) printProductList

➔ 마트가 가지고 있는 상품의 리스트를 출력하는 함수. 총 상품의 개수를 반환하도록 하여 addInventory 함수에서 사용될 수 있도록 하였다.

5) replaceExpired

➔ 마트에 있는 식품 중에서 유통기한이 지난 상품의 리스트를 출력하고, 연장할 유통기한을 입력 받아 일괄적으로 유통기한을 연장(LocalDateTime 의 plus 함수 이용)시키는 함수. 연장할 유통기한의 형식은, “년, 월, 일”의 형식으로 받도록 하였다. 만약 유통기한이 지난 식품이 하나도 없다면, “There are no expired products.”를 출력하도록 하였다.

6) customerMode

➔ customerList 에서 고객을 선택한 후, customer mode 로 진입한다. 진입할 메뉴를 입력 받아 각각 shopping, printShoppingCart, paying, printWallet 함수로 분기시켜주었다. 모든 단계에서 0 을 입력 받으면 main 으로 돌아가도록 하였다. 그 전까지는 모든 단계에서 무한 반복한다.

7) printCustomerList

➔ 고객의 리스트를 출력하는 함수. 총 고객 수를 반환하도록 하여 customerMode 함수에서 사용될 수 있도록 하였다.

8) shopping

➔ printProductList 를 호출하여 상품의 리스트를 출력한 뒤, 장바구니에 넣으려는 상품의 번호와 개수를 입력 받는다. 해당 customer 의 addToCart 함수를 호출하여 장바구니에 넣을 수 있는지 체크한 후에 추가할 수 있도록 해주었다.

9) printShoppingCart

➔ System.out.printf 를 통해 정렬된 형식으로 장바구니 내 상품의 이름, 개수, 금액을 리스트 형식으로 보여줄 수 있도록 하였다. paying 함수에서 사용될 수 있도록 형식을 맞춰주었으며, 장바구니 내의 총 금액을 반환하도록 하였다.

10) paying

➔ 장바구니 목록, 총 금액, 결제 수단 순서로 출력한 뒤, 결제 수단을 입력 받아 해당 고객의 requestPay 함수를 호출하도록 하였다. 만약 결제가 성공적으로 이루어졌다면, 영수증을 출력하고, 장바구니를 초기화할 수 있도록 printReceipt 함수와 clearCart 함수를 호출해주었다. 또한, 주문번호를 1 씩 증가시켜주었다. 0 을 입력하면 customerMode 로 돌아갈 수 있도록 하였다.

11) printWallet

➔ 고객의 결제 수단을 리스트 형식으로 출력해주었다. 총 결제 수단의 개수를 반환하도록 하였다.

12) printReceipt

➔ PrintWriter 를 통해 기본 위치에 `Receipt${주문번호}.txt`의 형식으로 영수증이 출력될 수 있도록 하였다. 출력 형식은 paying 의 형식과 같으며, 마지막에 결제 수단에 대한 내용을 추가로 출력할 수 있도록 하였다.

2. InventoryManager.java

Observer 패턴을 이용하기 위한 InventoryManager. addObserver 를 통해 observers ArrayList 에 구매대기를 신청한 고객을 넣어 놓고, 재입고가 되면 notifyObservers 를 호출할 수 있도록 하는 형태로 설계하였다. notify 를 한 후, 구매대기가 완전히 끝난 고객의 경우, 안전하게 delete 가 될 수 있도록 for 문 순회가 끝난 후에 ArrayList 의 removeIf 를 이용하여 삭제해주었다. getObservers 를 추가하여, 장바구니 추가 과정에서 동일한 고객이 observers 배열에 들어가지 않도록 하는데 사용되도록 하였다.

3. Mart.java (Singleton: Eager Initialization)

private static 타입으로 mart 객체를 미리 생성해주어, Singleton 패턴을 구현하였다. 이렇게 생성된 객체는, Mart 클래스의 static 메소드인 getInstance 를 통해 접근할 수 있게 된다. 주문번호(transactionNum)와 현재 시각(timeNow)도 static 변수로 미리 생성해주어, 언제든지 접근할 수 있도록 해주었다.

1) addProduct

➔ Mart.txt 에서 가져온 정보들로 product 를 추가해주는 함수.

2) addInventory

➔ 상품 재입고를 해주는 함수. 상품 재고를 추가한 후, notifyObservers 를 호출하여 고객들이 확인할 수 있도록 하였다.

4. Customer.java

기본적으로 이름(name), 결제수단(wallet), 장바구니(shoppingCart)를 가지며, 구매대기를 신청한 상품의 목록을 따로 저장하기 위한 ArrayList 인 waitingList 를 추가해 주었다.

1) addToCart

➔ customerMode 에서 선택한 상품의 번호와 개수를 가지고, 해당 상품이 장바구니에 들어갈 수 있는 상황인지를 체크하는 함수. 만약 해당 상품이 유통기한이 지난 식품이라면 ExpiredException 에러를 throw 하고, 재고가 부족한 경우에는 고객을 observers 배열에 추가하여 구매대기가 이루어질 수 있도록 하였다. 이 때, 원하는 개수의 일부를 구매할 수 있는 상황이라면, 가능한 만큼 장바구니에 넣고, 나머지 개수를 구매대기로 신청한다. 마지막으로, 정상적으로 장바구니에 넣을 수 있는 상황이라면, 상품을 장바구니에 넣은 후, 마트에서 해당 상품의 재고를 감소시킨다.

2) requestPay

➔ customerMode 에서 선택한 결제수단의 번호와 개수를 가지고, 결제가 가능한지 체크하는 함수. Payable 타입의 payment 를 통해, 각 결제수단에 맞는 pay 함수가 호출될 수 있도록 한다. 각 결제수단의 pay 함수에서 결제가 불가능한 상황으로 판정됐다면, pay 함수에서 throw 한 NotEnoughBalanceException 이 customerMode 에서 catch 될 수 있도록 throws 를 추가해주었다. 결제가 정상적으로 이루어졌다면, 결제완료 메시지와 함께 주문번호가 출력되도록 하였다.

3) clearCart

➔ 결제가 완료된 후, 장바구니를 초기화해주는 함수.

4) update

➔ Observer 패턴을 이용하기 위한 update 함수. addInventory 를 통해 상품의 재고가 추가되었다면, 구매대기를 신청한 고객들의 update 함수가 호출되어 각 고객별로 자신이 신청한 상품의 재고가 구매가능한 상황인지를 addToCart 함수를 호출하여 판단하게 된다. addToCart 에서 또다시 여러 조건들을 판단한 후, 아직도 재고가 부족한 상황이라면 waitingList 에 새로운 구매대기를 넣고, 구매대기가 완전히 끝난 상황이라면 waitingList 에서 해당 품목이 삭제되도록 구현하였다.

5. Product.java, Food.java, Manufactured.java

상품에 관한 클래스. Food 이면 유통기한을 가지게 되고, Manufactured 이면 브랜드를 추가로 가지게 된다. Food 에는 유통기한이 지났는지를 판별해주는 isExpired 함수를 구현해 주었다.

6. Cash.java, Credit.java

Payable 인터페이스를 implement 하는 결제수단에 관한 클래스. Cash 의 경우에는 현재 소지한 금액보다 작은지에 대해서 결제 가능 여부가 갈리고, Credit 의 경우에는 현재 누적 사용 금액과 사용한도에 따라 결제 가능 여부가 갈리도록 pay 함수를 구현해 주었다. 결제가 가능하다면, 결제 금액을 보유 금액 및 누적 사용 금액에 반영해주었고, 결제가 불가능한 상황이라면, NotEnoughBalanceException 을 throw 하도록 하였다.

7. ExpiredException.java, InvalidAccessException.java, NotEnoughBalanceException.java

각각 유통기한이 지난 상품을 구매하려는 경우, 주어진 메뉴 번호의 범위를 벗어난 입력을 받은 경우, 잔고 부족 및 한도 초과로 결제가 불가능한 경우에 예외가 발생할 수 있도록 해주었다.

8. Payable.java, Observer.java

다른 클래스에서 implement 할 수 있도록 메소드들을 정의해주었다.