학번: 2401110252

이름: 박지수

1. 개요

1.1. 과제 목표

요구사항은 각 메뉴에 대한 이름과 가격이 제시되고 지갑에 돈을 넣어준다.
작동 방식은 제시된 지갑의 돈이 출력되고

메뉴 출력 -> 지갑에 있는 모든 돈 넣기 -> 메뉴 선택 -> 음료수 수령 -> 잔돈 수령 -> 잔액 표시
이 과정을 3번 행하면 된다.

다만 예외 사항으로 선택한 음료보다 가진 돈이 적으면 경고 문구가 뜨고
구매가 안되며 곧바로 중단된다.
이 작동 방식을 구현하기 위해 다음과 같은 코드의 순서로 작성하였다.

1.2. 목표를 구현하기 위한 작동 방식 및 설명

1.2.1. import, 전역 상수와 main문

장바구니 배열의 원소를 프로그램 실행 도중 삽입, 삭제 및 접근을 하기 위함과
주문메뉴를 입력받거나 구입을 재개 및 중단을 입력 받기 위해서
java.util.ArrayList java.util.Scanner를 import 했다.

변숫값을 코드 내에서 변경하지 않으면서 어디서든 접근할 수 있는 변수를 만들기 위해
static, final 이용해서 전역 상수의 느낌을 주었다.

이 외에도 배열의 길이, 제일 싼 음료수와 같은 값도 전역 상수로 작성하려 했지만
필요할 때 실행되도록 두었다.

main문에서는 함수들을 실행시키기 위한 변수들을 선언하고 필요한 값들을 불러온다.
목표로 제시한 초기 지갑의 돈이 출력되는 것은 main의 printWallet 함수에서 실행되고
마지막에는 RunningVMachine 함수를 실행시키며 main의 내용은 끝난다.

1.2.2. 이 프로그램의 심장과 같은 함수 RunningVMachine

첫 번째 while문에 해당하는 boolean값이 설정되면 이 프로그램은 더 이상 반복 실행되지 않는다.

안쪽에는 PreOrder라는 선주문받는 함수가 실행되고 그다음
주문을 받는 함수가 실행되는 순서이다.
마지막은 계산을 끝내는 함수가 실행되고 거슬러 올라가 반환 값을 _wallet에 받는다.

그다음 IfNoWallet 함수는 구매를 한 뒤에 지갑에 돈을 검사하고 아무것도 살 수 없는 금액만 남았다면 첫 번째 두 번째 While문에 들어가지 않게 값을 제어해서 프로그램을 종료한다.

두 번째 while문에 해당하는 boolean값은 추가로 살 계획의 여부를 검사하는 값이다.
WhileAdditional 함수가 실행되면 구입에 성공하거나 실패했을 때 추가로 살 계획이 있는지 받아오고 아니라면 프로그램을 종료한다.

1.2.3. 메뉴를 보여주고 주문을 텍스트로 받는 함수

먼저 메뉴를 보여주는 ViewMenu 함수가 실행되고 ParsingInputInt 함수를 통해
숫자 이외의 값을 입력받지 못하게 한다.

1.2.2. 에서도 쓴 것처럼 계산이 다 끝난 값을 받아오고 _wallet을 통해 반환한다.

1.2.4. 주문받은 메뉴를 정리

현재 주문받을 이름과 가격을 담을 변수를 선언 및 초기화 한다.
그리고 금액이 부족하거나 없는 메뉴를 입력받아 살 수 없는 상황을 생각해서 구매가
실제로 되었는 체크하는 boolean 값도 선언, 초기화 한다.

메뉴가 진짜 있는지를 조건 검사하는데 Switch문을 통해 이루어진다.
간단하게 있는 메뉴인지 확인하고 있으면 해당하는 메뉴를 currentDrinkName에 저장하고
currentDrinkPrice에 값 또한 저장한다.

그다음 선별한 변수들의 값을 OrderCondition 함수에 넘기고 종료된다.

1.2.5. 구입 여부를 검사하는 함수

첫 if 조건문에서는 있는 메뉴는 그대로 출력시키고
없는 메뉴는 계산을 할 필요가 없어 안내문만 출력한 다음 가지고 있는 금액을 반환한다.

1.2.4. OrderDrink 함수 에서도 실행시켜도 되지만 좀 더 역할을 분리하기 위해서
여기까지 끌고 왔다.

그 다음 if 조건문은 가지고 있는 돈보다 선택한 메뉴의 가격보다 많으면
금액을 계산하는 CalculateDrink 함수를 실행시키고 _wallet 변수에 저장한다.

계산이 완료됐으면 구매도 완료되었으니 boolean값도 변경하고 계산이 끝나면
출력되는 함수를 실행한다. 그 다음 _wallet을 반환하고 종료된다.

만약 가지고 있는 금액이 부족해서 구입할 수 없다면
잔액이 부족하다는 출력문과 현재 가지고 있는 잔액, 주문을 더이상 할수없을때 나오는
출력문이 나오고 _wallet을 반환시키고 종료된다.

1.2.6. 계산이 다 끝나면 실행되는 함수

한 개 살 때마다 계산해야 하는데 그때 마다 장바구니에 담아야 한다. 하지만 담을 물건이 없는데 담았다고 기록되는 것은 말도 안 되는 일이다.

그렇기 때문에 구입이 진짜 완료되었는지 확인하는 boolean 값으로
if 조건을 걸어주고 정말 구매가 되었다면 장바구니에 담고 샀던 내용물을
전부를 출력시킨다. 그다음은 지갑의 금액을 출력시키고 종료된다.

1.2.7. 주문이 다 끝나면 실행되는 함수

한 개씩 사면서 계산하고 장바구니에 담다 보면 무엇을 샀는지
총 얼마나 샀는지 궁금하게 된다. 이를 위해 만든 함수라고 이해하면 된다.
그런데 살려고 할 때 메뉴에 없는 것을 누르고 안 사게 될 경우도 있다.

이때를 위해서 _basket.isEmpty를 조건으로 걸어서 해당하는 출력문을 실행시키고
종료되게 한다. 당연히 살 때에도 관련 출력문이 실행된다.

1.2.8. 아무것도 살 수 없는 상황일 때 실행되는 함수

가지고 있는 돈을 다 넣었지만 살 수 있는 메뉴가 아무것도 없고 메뉴를 고르라고
창이 뜨면 매우 당혹스럽다.

그렇기 때문에 메뉴 중 가장 싼 값을 불러와서 지금 가지고 있는 지갑의 액수와 비교해서
관련 내용을 출력시킨 후 1.2.7. 함수를 실행시키고 종료된다.

1.2.9. 추가로 주문받을 때 실행되는 함수

먼저 안내문이 출력되고 사용자에게 입력값을 받는다.
이때 입력 받는 함수는 똑같이 숫자로 받기 때문에 메뉴 받을 때와 똑같다.

만약 값이 1번 아니면 2번으로 잘 들어왔고 1번이면 현재 걸린 while 제어문에 탈출하고
2번으로 들어온다면 더 이상 살 계획이 없기 때문에 프로그램을 종료시키기 위해
boolean 값을 전부 false로 바꾸고 EndOrderInform 함수를 실행시키고 종료된다.

1.2.10. 메뉴를 보기 위한 함수

메뉴를 배열로 받아왔기 때문에 하나하나 다 빼 내줘야 한다.
방법은 간단하게 for 반복문을 통해서 출력한다.

다만 주의해야 할 점은 “1. 콜라”인 경우 i는 0부터 시작하고
메뉴의 원소도 0으로 시작하지만, 메뉴 순서를 적을 때 “0. 콜라” 라고는

적지 않기 때문에 순서 적을 때에만 i에 1을 덧셈한다.

1.2.11. 장바구니를 보기 위한 함수

장바구니는 List 배열로 받아왔기 때문에 반복문을 실행시키지 않아도 전체값이 출력된다.
하지만 출력문에는 괄호가 붙여서 나오기 때문에 보기 싫은 관계로 또 똑같이 가져온다.

배열과는 다르게 List속성 안에 길이를 측정하는 size() 함수와
해당 인덱스에 접근하는 get() 함수가 있기 때문에 활용해서 프린트문을 작성했다.

1.2.12. 장바구니를 보기 위한 함수_2

큰 골자는 똑같지만 1.2.11 함수는 딱 물건만 쉼표를 넣어 출력하는 것이고
이 함수는 비어 있으면 그대로 종료하는 경우와 앞뒤로 출력문이 들어간 형태이다.

List 배열이 비어 있는지 확인하는 함수도 있는데 isEmpty()를 사용해
비어 있는지를 if 조건문을 걸었다.

1.2.13. 지갑 액수를 확인하는 함수

입력받은 지갑의 값을 가져와서 출력시키는 간단한 함수이다.
다만 조건으로 boolean 값을 걸었다.

아무것도 구입하지 않았을 때도 이 함수가 실행되는데 그때에는 거스름돈 이란 텍스트를
출력하면 안되기 때문에 boolean값으로 조건을 걸었다.

1.2.14. 입력 시 에러를 막기 위해 예외 처리를 하는 함수

Scanner를 통해 입력값을 받게 되면 지정된 자료형으로만 입력받을 수 있다.
반대로 지정된 자료형이 아닌 형태를 입력 받게 되면 에러를 일으키며 프로그램이
강제로 중단되는 사고가 일어난다.

이를 해결하기 위해 try catch문을 사용하고 그 안에 NumberFormatException을 넣었다.

이 함수 구조는 입력을 저장 받을 int형 inputIntText 변수와
예외처리가 발생하지 않았을 때를 검사하기 위해 boolean형 noneException 변수를 선언,
초기화 했다.

그 다음 noneException 값을 조건으로 while문이 실행된다.
에러가 나지 않을 때까지 실행된다는 조건이다.

바로 try 문이 나오는데 try 안쪽에서 코드가 진행될때 catch로 지정한 값이 나올때 바로
catch 안에 있는 코드가 실행된다.

즉 예외 처리가 나지 않아야 noneException = true; 라는 코드가 실행된다.
예외처리 없이 정수형 타입만 받아왔다면 inputIntText 변수를 반환하며 함수가 종료된다.

1.2.15. 음료 값을 계산하는 함수

정말 간단하고 직관적인 함수이다.
함수에 입력받은 값 두 개를 빨셈한 다음 그 값을 반환한다.

1.2.16. 현재 메뉴의 최솟값을 찾는 함수

입력받은 배열의 원소 아무것이라도 가져와서 최솟값으로 int형 선언, 초기화 한다.
그다음 for 문을 실행하는데 간단하다.

만약 내가 지정한 변수인 minPrice 값이 배열의 i번째 값보다 클 경우 minPrice 보다
작은 값인 i번째 배열의 원소를 minPrice값으로 변경하면 된다.
배열의 길이만큼 다 검사하면 배열 중 제일 작은 값을 minPrice에 저장하게 된다.

그리고 그 값을 반환하면 다른 함수의 조건식에 사용할 수 있다.

1.3. 작동 결과

1.3.1. 정수형 입력을 못하여 아무것도 못 산 경우

```
현재 잔액은 2000원 있습니다.

1. 콜라 500원 2. 사이다 500원 3. 물 800원 4. 파워에이드 1000원
제품에 해당하는 숫자를 입력하시면 구매가 진행됩니다.
5
잘못된 선택을 하였습니다.
거스름돈을 2000원 받아,현재 잔액은 2000원 있습니다.

더 주문하실 사항이 있으면 1번 아니면 2번을 입력해주세요
2
오늘 구입 하신 물품은 없습니다.
반환되는 금액은 2000원 입니다.
Process finished with exit code 0
```

1.3.2. 정수형 이외의 입력을 하여 아무것도 못 산 경우

```
현재 잔액은 2000원 있습니다.

1. 콜라 500원 2. 사이다 500원 3. 물 800원 4. 파워에이드 1000원
제품에 해당하는 숫자를 입력하시면 구매가 진행됩니다.
asd
정수의 숫자만 적어 주십시오.
5
잘못된 선택을 하였습니다.
거스름돈을 2000원 받아,현재 잔액은 2000원 있습니다.

더 주문하실 사항이 있으면 1번 아니면 2번을 입력해주세요
asd
정수의 숫자만 적어 주십시오.
3
더 주문하실 사항이 있으면 1번 아니면 2번을 입력해주세요
2
오늘 구입 하신 물품은 없습니다.
반환되는 금액은 2000원 입니다.
Process finished with exit code 0
```

1.3.3. 구입을 하고 중단한 경우

```
현재 잔액은 2000원 있습니다.

1. 콜라 500원 2. 사이다 500원 3. 물 800원 4. 파워에이드 1000원
제품에 해당하는 숫자를 입력하시면 구매가 진행됩니다.
1
콜라을(를) 선택하였습니다.
현재까지 장바구니에 담긴 물건은 콜라입니다.
거스름돈을 1500원 받아,현재 잔액은 1500원 있습니다.

더 주문하실 사항이 있으면 1번 아니면 2번을 입력해주세요
2
오늘 구입 하신 물품은 콜라입니다.
사용한 금액은 500원 반환되는 금액은 1500원입니다.
이용해 주셔서 감사합니다.

Process finished with exit code 0
```

1.3.4. 남은 금액을 0원에 맞게 산 경우

```
현재 잔액은 2000원 있습니다.

1. 콜라 500원 2. 사이다 500원 3. 물 800원 4. 파워에이드 1000원
제품에 해당하는 숫자를 입력하시면 구매가 진행됩니다.
4
파워에이드을(를) 선택하였습니다.
현재까지 장바구니에 담긴 물건은 파워에이드입니다.
거스름돈을 1000원 받아,현재 잔액은 1000원 있습니다.

더 주문하실 사항이 있으면 1번 아니면 2번을 입력해주세요
1
1. 콜라 500원 2. 사이다 500원 3. 물 800원 4. 파워에이드 1000원
제품에 해당하는 숫자를 입력하시면 구매가 진행됩니다.
4
파워에이드을(를) 선택하였습니다.
현재까지 장바구니에 담긴 물건은 파워에이드, 파워에이드입니다.
거스름돈을 0원 받아,현재 잔액은 0원 있습니다.

더 이상 살 수 있는 메뉴가 없어 자동으로 구입이 종료됩니다.
오늘 구입 하신 물품은 파워에이드, 파워에이드입니다.
사용한 금액은 2000원 반환되는 금액은 0원입니다.
이용해 주셔서 감사합니다.

Process finished with exit code 0
```

1.3.5. 남은 금액이 0원 보다는 크지만, 최소 금액보다 적은 경우

```
현재 잔액은 2000원 있습니다.

1. 콜라 500원 2. 사이다 500원 3. 물 800원 4. 파워에이드 1000원
제품에 해당하는 숫자를 입력하시면 구매가 진행됩니다.
4
파워에이드을(를) 선택하였습니다.
현재까지 장바구니에 담긴 물건은 파워에이드입니다.
거스름돈을 1000원 받아,현재 잔액은 1000원 있습니다.

더 주문하실 사항이 있으면 1번 아니면 2번을 입력해주세요
1
1. 콜라 500원 2. 사이다 500원 3. 물 800원 4. 파워에이드 1000원
제품에 해당하는 숫자를 입력하시면 구매가 진행됩니다.
3
물을(를) 선택하였습니다.
현재까지 장바구니에 담긴 물건은 파워에이드, 물입니다.
거스름돈을 200원 받아,현재 잔액은 200원 있습니다.

더 이상 살 수 있는 메뉴가 없어 자동으로 구입이 종료됩니다.
오늘 구입 하신 물품은 파워에이드, 물입니다.
사용한 금액은 1800원 반환되는 금액은 200원입니다.
이용해 주셔서 감사합니다.

Process finished with exit code 0
```

1.3.6. 잔액 보다 비싼 메뉴를 시켜 강제로 종료되는 경우

```
현재 잔액은 2000원 있습니다.

1. 콜라 500원 2. 사이다 500원 3. 물 800원 4. 파워에이드 1000원
제품에 해당하는 숫자를 입력하시면 구매가 진행됩니다.
4
파워에이드을(를) 선택하였습니다.
현재까지 장바구니에 담긴 물건은 파워에이드입니다.
거스름돈을 1000원 받아,현재 잔액은 1000원 있습니다.

더 주문하실 사항이 있으면 1번 아니면 2번을 입력해주세요
1
1. 콜라 500원 2. 사이다 500원 3. 물 800원 4. 파워에이드 1000원
제품에 해당하는 숫자를 입력하시면 구매가 진행됩니다.
1
콜라을(를) 선택하였습니다.
현재까지 장바구니에 담긴 물건은 파워에이드, 콜라입니다.
거스름돈을 500원 받아,현재 잔액은 500원 있습니다.

더 주문하실 사항이 있으면 1번 아니면 2번을 입력해주세요
1
1. 콜라 500원 2. 사이다 500원 3. 물 800원 4. 파워에이드 1000원
제품에 해당하는 숫자를 입력하시면 구매가 진행됩니다.
4
파워에이드을(를) 선택하였습니다.
파워에이드을(를) 사기에는 잔액이 부족합니다.
거스름돈을 500원 받아,현재 잔액은 500원 있습니다.

오늘 구입 하신 물품은 파워에이드, 콜라입니다.
사용한 금액은 1500원 반환되는 금액은 500원입니다.
이용해 주셔서 감사합니다.

Process finished with exit code 0
```

1.3.7. 3번 실행하고 중단하는 경우

```
현재 잔액은 2000원 있습니다.

1. 콜라 500원 2. 사이다 500원 3. 물 800원 4. 파워에이드 1000원
제품에 해당하는 숫자를 입력하시면 구매가 진행됩니다.
1
콜라을(를) 선택하였습니다.
현재까지 장바구니에 담긴 물건은 콜라입니다.
거스름돈을 1500원 받아,현재 잔액은 1500원 있습니다.

더 주문하실 사항이 있으면 1번 아니면 2번을 입력해주세요
1
1. 콜라 500원 2. 사이다 500원 3. 물 800원 4. 파워에이드 1000원
제품에 해당하는 숫자를 입력하시면 구매가 진행됩니다.
2
사이다을(를) 선택하였습니다.
현재까지 장바구니에 담긴 물건은 콜라, 사이다입니다.
거스름돈을 1000원 받아,현재 잔액은 1000원 있습니다.

더 주문하실 사항이 있으면 1번 아니면 2번을 입력해주세요
1
1. 콜라 500원 2. 사이다 500원 3. 물 800원 4. 파워에이드 1000원
제품에 해당하는 숫자를 입력하시면 구매가 진행됩니다.
1
콜라을(를) 선택하였습니다.
현재까지 장바구니에 담긴 물건은 콜라, 사이다, 콜라입니다.
거스름돈을 500원 받아,현재 잔액은 500원 있습니다.

더 주문하실 사항이 있으면 1번 아니면 2번을 입력해주세요
2
오늘 구입 하신 물품은 콜라, 사이다, 콜라입니다.
사용한 금액은 1500원 반환되는 금액은 500원입니다.
이용해 주셔서 감사합니다.

Process finished with exit code 0
```

2. 코드

```java
//2024_04_04
//한국폴리텍대학_서울정수캠퍼스_인공지능소프트웨어학과
//2401110252_박지수
//자동음료수판매기


import java.util.ArrayList;
import java.util.Scanner;

public class VendingMachine
{
    static final String DRINKMENU[] = {"콜라", "사이다", "물", "파워에이드"};
    // DRINKMENU[0] = "콜라" DRINKMENU[3] = "파워에이드"
    static final int DRINKPRICE[] = {500, 500, 800, 1000};
    // 콜라, 사이다, 물, 파워에이드의 값을 배열로 표현
    static final int INIT_WALLET = 2000; // 초기 제시된 지갑액
```

```
public static void main(String[] args)
{
    int wallet = INIT_WALLET; //지갑의 돈을 초기화.
    ArrayList<String> basket = new ArrayList<>();
    // 장바구니 인스턴스 생성
    ArrayList<Boolean> isPurchaseFinished = new ArrayList<>();
    isPurchaseFinished.add(0,true);
    //isPurchaseFinished.add(1,true);
    isPurchaseFinished.add(1,true);
    // {WhileAdditionalOrder, RunningVMachine}
    // 각각 순서대로 해당됨
    // While문을 제어하기 위한 Boolean 값
    Scanner inputText = new Scanner(System.in);
    // 스캐너를 통해 입력받을 변수
    PrintWallet(wallet,true);
    // 지갑에 있는 잔액 출력 처음나올때 출력 할 내용과
    // 그 후 출력 할 내용이 달라서 뒤에 boolean이 들어감
    RunningVMachine(wallet, basket, isPurchaseFinished,
          inputText,FindMinPrice(DRINKPRICE));
    // 안에 While문이 내장되어 있으며 이 함수에서
    // 대부분의 함수를 호출해서 진행해 나감.
}

private static void RunningVMachine(int _wallet,ArrayList<String> _basket,
                                    ArrayList<Boolean> _isPurchaseFinished,
                                    Scanner _inputText,int _minPrice)

{
    while (_isPurchaseFinished.get(1)) // 이 프로그램이 작동 할 것인지
    {

        _wallet = PreOrder(_wallet, _basket, _isPurchaseFinished , _inputText);
        IfNoWallet(_wallet, _basket, _isPurchaseFinished, _minPrice);

        while (_isPurchaseFinished.get(0)) // 추가로 더 살 계획이 있는지
        {
            WhileAdditionalOrder(_wallet, _basket, _isPurchaseFinished, _inputText);
        }
    }
```

```
    }
    private static int PreOrder(int _wallet,ArrayList<String> _basket,
                            ArrayList<Boolean> _isPurchaseFinished ,Scanner _inputText)
    {
        ViewMenu();
        int inputOrderDrinkMenu = ParsingInputInt(_inputText);
        _wallet = OrderDrink(inputOrderDrinkMenu, _wallet,
               _basket, _isPurchaseFinished);
        return _wallet;
    }
    private static int OrderDrink
           (int _selectMenu, int _wallet, ArrayList<String> _basket, ArrayList<Boolean>
_isPurchaseFinished)
    // 구매시 실행되는함수, 중요 함수들이 함수를 타고 들어가면서 실행됨.
    {
        String currentDrinkName = "";
        int currentDrinkPrice = 0;
        boolean checkPurchase = false;
        switch (_selectMenu)
        {
            case 1: currentDrinkName = DRINKMENU[0];
                currentDrinkPrice = DRINKPRICE[0];break;
            case 2: currentDrinkName = DRINKMENU[1];
                currentDrinkPrice = DRINKPRICE[1];break;
            case 3: currentDrinkName = DRINKMENU[2];
                currentDrinkPrice = DRINKPRICE[2];break;
            case 4: currentDrinkName = DRINKMENU[3];
                currentDrinkPrice = DRINKPRICE[3];break;
            default: break;
        }
        return OrderCondition(_selectMenu, currentDrinkName, currentDrinkPrice,
               _wallet, _basket, _isPurchaseFinished, checkPurchase);
        // 메뉴 선택후 지금 선택한 메뉴가 돈이 있는지 없는지 검사하러감.
    }
    private static int OrderCondition(int _selectMenu, String _currentDrinkName, int
_currentDrinkPrice, int _wallet
          , ArrayList<String> _basket, ArrayList<Boolean> _isPurchaseFinished,
                                boolean _checkPurchase) {
        if (_selectMenu > 0 && DRINKMENU.length >= _selectMenu)    // 메뉴에 있는 값을 넣을시
        {
            System.out.printf("%s을(를) 선택하였습니다.\n", _currentDrinkName);
        }
        else    // 메뉴에 없는 값을 넣을시 여기서 종료
        {
```

```
            System.out.print("잘못된 선택을 하였습니다.\n");
            CalculationAfterInform(_checkPurchase, _wallet, _basket, _currentDrinkName);
            _isPurchaseFinished.set(0,true);
            return _wallet;
        }

        if (_wallet >= _currentDrinkPrice)
        // 메뉴에 있는데 넣은 돈과 선택한 메뉴의 가격과
        // 검사 살 수 있으면 계산하고 구입한 물건, 지갑 액수 출력
        {
            _wallet = CalculateDrink(_currentDrinkPrice, _wallet);
            _checkPurchase = true;
            CalculationAfterInform(_checkPurchase, _wallet, _basket, _currentDrinkName);
            _isPurchaseFinished.set(0,true);
            return _wallet;
        }
        else // 구입할 수 없다면 부족 알림 및 지갑 액수 출력
        {
            System.out.printf("%s을(를) 사기에는 잔액이 부족합니다.\n", _currentDrinkName);
            PrintWallet(_wallet,false);
            _isPurchaseFinished.set(0,false);
            _isPurchaseFinished.set(1,false);
            EndOrderInform(_wallet, _basket);
            return _wallet;
        }
    }
    private static void CalculationAfterInform // 계산이 다 끝나면 나오는 함수
    (Boolean _checkPurchase, int _inputWalletOrCompleteCash,
     ArrayList<String> _basket, String _currentDrinkName)
    {
        if (_checkPurchase)
        {
            _basket.add(_currentDrinkName);
            // 구매가능 하면 장바구니에 담고 아니면 현재 목록 및 지갑 출력
        }
        ViewBasket(_basket);
        PrintWallet(_inputWalletOrCompleteCash,false);
    }
    private static void EndOrderInform(int _wallet, ArrayList<String> _basket)
    //주문이 전부 끝나면 나오는 알림들
    {
        if (_basket.isEmpty()) //장바구니가 비어있으면 종료
```

```
        {
            System.out.print("오늘 구입 하신 물품은 없습니다.\n");
            System.out.printf("반환되는 금액은 %d원 입니다.", _wallet);
            return;
        }
        System.out.print("오늘 구입 하신 물품은 ");
        PrintBasket(_basket);
        System.out.print("입니다.\n");
        System.out.printf("사용한 금액은 %d원 반환되는 금액은 %d원입니다.\n" +
                "이용해 주셔서 감사합니다.\n", (INIT_WALLET - _wallet),_wallet );
    }
    private static void IfNoWallet(int _wallet,ArrayList<String> _basket,
                                   ArrayList<Boolean> _isPurchaseFinished,int _minPrice ) {
        // 아무것도 살 수 없는 잔액상태일때 실행
        if (_wallet < _minPrice)
        {
            System.out.println
                    ("더 이상 살 수 있는 메뉴가 없어 자동으로 구입이 종료됩니다.");
            EndOrderInform(_wallet, _basket);
            _isPurchaseFinished.set(0, false);
            _isPurchaseFinished.set(1, false);
        }
    }

    private static void WhileAdditionalOrder(int _wallet,ArrayList<String> _basket,
                                             ArrayList<Boolean> _isPurchaseFinished ,Scanner
_inputText)
            // 주문을 더 할 수 있을때 입력받음.
    {
        System.out.print("더 주문하실 사항이 있으면 1번 아니면 2번을 입력해주세요\n");
        int _inputIsPurchaseFinishedValue = ParsingInputInt(_inputText);
        if (_inputIsPurchaseFinishedValue == 1)
        {
            _isPurchaseFinished.set(0,false);
        }
        else if (_inputIsPurchaseFinishedValue == 2)
        {
            _isPurchaseFinished.set(0,false);
            _isPurchaseFinished.set(1,false);
            EndOrderInform(_wallet, _basket);
        }
```

```
}
private static void ViewMenu()
{
    for (int i = 0; i < DRINKMENU.length; i++)
    {   //메뉴의 원소는 0부터 시작하지만 메뉴판 출력할때는 1부터 시작해야한다.
        System.out.printf("%d", i + 1);System.out.print(". ");
        System.out.printf("%s ", DRINKMENU[i]);
        System.out.printf("%d원 ", DRINKPRICE[i]);
    }
    System.out.println("");
    System.out.println("제품에 해당하는 숫자를 입력하시면 구매가 진행됩니다.");
}
private static void PrintBasket(ArrayList<String> _basket)
        // 딱 장바구니 내용물만 출력한다.
{
    for (int i = 0; i < _basket.size(); i++)
    {
        System.out.printf("%s", _basket.get(i));
        if (i < _basket.size() - 1)
        {
            System.out.print(", ");
        }
    }
}
private static void ViewBasket(ArrayList<String> _basket)
// 장바구니와 관련된 출력 함수
{
    if (_basket.isEmpty())  //비어있으면 종료 ,
    // 처음에 메뉴 잘못선택했을때 더 구입할건지 안할건지
    // 입력창이 뜰때 구입 안한다고 입력하면 발생
    {
        return;
    }
    System.out.print("현재까지 장바구니에 담긴 물건은 ");
    PrintBasket(_basket);
    System.out.print("입니다.\n");
}
private static void PrintWallet(int _nowWallet,boolean _firstPrint)
{
    if (_firstPrint)
        System.out.printf("현재 잔액은 %d원 있습니다.\n\n",_nowWallet);
```

```
            else System.out.printf("거스름돈을 %d원 받아,현재 잔액은 %d원 있습니다.\n\n"
                    , _nowWallet,_nowWallet);
        }
        private static int ParsingInputInt(Scanner _nextInt)
        // 여기서 메뉴의 입력값, 주문 계속 할지 안할지의 입력값을
        // 검사해서 숫자 이외의 값이 들어올경우 에러가 안생기게 해줌
        {
            int inputIntText = 0;
            boolean noneException = false;
            while (!noneException)
            {
                try
                {
                    inputIntText = Integer.parseInt(_nextInt.nextLine());
                    noneException = true;
                }
                catch (NumberFormatException noneInt)
                {
                    System.out.println("정수의 숫자만 적어 주십시오.");
                }
            }
            return inputIntText;
        }

        private static int CalculateDrink(int _currentDrinkPrice, int _inputWallet) // 메뉴 값
을 계산 하는 함수
        {
            return (_inputWallet - _currentDrinkPrice);
        }

        private static int FindMinPrice(int[] _menuPriceArray) // 메뉴 최솟값을 계산하는 함수
        {
            int minPrice = _menuPriceArray[0];
            for (int i = 0; i < _menuPriceArray.length; i++)
            {
                if ( minPrice > _menuPriceArray[i])
                {
                    minPrice = _menuPriceArray[i];
                }
            }
            return minPrice;
        }
}
```

3. 스크린샷

```
public class VendingMachine
{
    7 usages
    static final String DRINKMENU[] = {"콜라", "사이다", "물", "파워에이드"};
    // DRINKMENU[0] = "콜라" DRINKMENU[3] = "파워에이드"
    6 usages
    static final int DRINKPRICE[] = {500, 500, 800, 1000};
    // 콜라, 사이다, 물, 파워에이드의 값을 배열로 표현
    2 usages
    static final int INIT_WALLET = 2000; // 초기 제시된 지갑액

    public static void main(String[] args)
    {
        int wallet = INIT_WALLET; //지갑의 돈을 초기화.
        ArrayList<String> basket = new ArrayList<>();
        // 장바구니 인스턴스 생성
        ArrayList<Boolean> isPurchaseFinished = new ArrayList<>();
        isPurchaseFinished.add( index: 0, element: true);
        //isPurchaseFinished.add(1,true);
        isPurchaseFinished.add( index: 1, element: true);
        // {WhileAdditionalOrder, RunningVMachine}
        // 각각 순서대로 해당됨
        // While문을 제어하기 위한 Boolean 값
        Scanner inputText = new Scanner(System.in);
        // 스캐너를 통해 입력받을 변수
        PrintWallet(wallet, _firstPrint: true);
        // 지갑에 있는 잔액 출력 처음나올때 출력 할 내용과
        // 그 후 출력 할 내용이 달라서 뒤에 boolean이 들어감
        RunningVMachine(wallet, basket, isPurchaseFinished,
                inputText,FindMinPrice(DRINKPRICE));
        // 안에 While문이 내장되어 있으며 이 함수에서
        // 대부분의 함수를 호출해서 진행해 나감.
    }
```

3.1.

3.2.

```
private static void RunningVMachine(int _wallet,ArrayList<String> _basket,
                                    ArrayList<Boolean> _isPurchaseFinished,
                                    Scanner _inputText,int _minPrice)

{
    while (_isPurchaseFinished.get(1)) // 이 프로그램이 작동 할 것인지
    {

        _wallet = PreOrder(_wallet, _basket, _isPurchaseFinished , _inputText);
        IfNoWallet(_wallet, _basket, _isPurchaseFinished, _minPrice);

        while (_isPurchaseFinished.get(0)) // 추가로 더 살 계획이 있는지
        {
            WhileAdditionalOrder(_wallet, _basket, _isPurchaseFinished, _inputText);
        }
    }
}
```

3.3.

```
private static int PreOrder(int _wallet,ArrayList<String> _basket,
                            ArrayList<Boolean> _isPurchaseFinished ,Scanner _inputText)
{
    ViewMenu();
    int inputOrderDrinkMenu = ParsingInputInt(_inputText);
    _wallet = OrderDrink(inputOrderDrinkMenu, _wallet,
            _basket, _isPurchaseFinished);
    return _wallet;
}
```

3.4.

```
private static int OrderDrink
        (int _selectMenu, int _wallet, ArrayList<String> _basket, ArrayList<Boolean> _isPurchaseFinished)
// 구매시 실행되는함수, 중요 함수들이 함수를 타고 들어가면서 실행됨.
{
    String currentDrinkName = "";
    int currentDrinkPrice = 0;
    boolean checkPurchase = false;
    switch (_selectMenu)
    {
        case 1: currentDrinkName = DRINKMENU[0];
            currentDrinkPrice = DRINKPRICE[0];break;
        case 2: currentDrinkName = DRINKMENU[1];
            currentDrinkPrice = DRINKPRICE[1];break;
        case 3: currentDrinkName = DRINKMENU[2];
            currentDrinkPrice = DRINKPRICE[2];break;
        case 4: currentDrinkName = DRINKMENU[3];
            currentDrinkPrice = DRINKPRICE[3];break;
        default: break;
    }
    return OrderCondition(_selectMenu, currentDrinkName, currentDrinkPrice,
            _wallet, _basket, _isPurchaseFinished, checkPurchase);
    // 메뉴 선택후 지금 선택한 메뉴가 돈이 있는지 없는지 검사하러감.
}
```

3.5.

```java
private static int OrderCondition(int _selectMenu, String _currentDrinkName, int _currentDrinkPrice, int _wallet
        , ArrayList<String> _basket, ArrayList<Boolean> _isPurchaseFinished,
                                  boolean _checkPurchase) {
    if (_selectMenu > 0 && DRINKMENU.length ≥ _selectMenu)    // 메뉴에 있는 값을 넣을시
    {
        System.out.printf("%s를(를) 선택하였습니다.\n", _currentDrinkName);
    }
    else    // 메뉴에 없는 값을 넣을시 여기서 종료
    {
        System.out.print("잘못된 선택을 하였습니다.\n");
        CalculationAfterInform(_checkPurchase, _wallet, _basket, _currentDrinkName);
        _isPurchaseFinished.set(0,true);
        return _wallet;
    }

    if (_wallet ≥ _currentDrinkPrice)
    // 메뉴에 있는데 넣은 돈과 선택한 메뉴의 가격과
    // 검사 살 수 있으면 계산하고 구입한 물건, 지갑 액수 출력
    {
        _wallet = CalculateDrink(_currentDrinkPrice, _wallet);
        _checkPurchase = true;
        CalculationAfterInform(_checkPurchase, _wallet, _basket, _currentDrinkName);
        _isPurchaseFinished.set(0,true);
        return _wallet;
    }
    else // 구입할 수 없다면 부족 알림 및 지갑 액수 출력
    {
        System.out.printf("%s를(를) 사기에는 잔액이 부족합니다.\n", _currentDrinkName);
        PrintWallet(_wallet, _firstPrint: false);
        _isPurchaseFinished.set(0,false);
        _isPurchaseFinished.set(1,false);
        EndOrderInform(_wallet, _basket);
        return _wallet;
    }
}
```

3.6.

```
private static void CalculationAfterInform // 계산이 다 끝나면 나오는 함수
(Boolean _checkPurchase, int _inputWalletOrCompleteCash,
 ArrayList<String> _basket, String _currentDrinkName)
{
    if (_checkPurchase)
    {
        _basket.add(_currentDrinkName);
        // 구매가능 하면 장바구니에 담고 아니면 현재 목록 및 지갑 출력
    }
    ViewBasket(_basket);
    PrintWallet(_inputWalletOrCompleteCash, _firstPrint: false);
}
```

3.7.

```
private static void EndOrderInform(int _wallet, ArrayList<String> _basket)
//주문이 전부 끝나면 나오는 알림들
{
    if (_basket.isEmpty()) //장바구니가 비어있으면 종료
    {
        System.out.print("오늘 구입 하신 물품은 없습니다.\n");
        System.out.printf("반환되는 금액은 %d원 입니다.", _wallet);
        return;
    }
    System.out.print("오늘 구입 하신 물품은 ");
    PrintBasket(_basket);
    System.out.print("입니다.\n");
    System.out.printf("사용한 금액은 %d원 반환되는 금액은 %d원입니다.\n" +
            "이용해 주셔서 감사합니다.\n", (INIT_WALLET - _wallet),_wallet );
}
```

```
private static void IfNoWallet(int _wallet,ArrayList<String> _basket,
                               ArrayList<Boolean> _isPurchaseFinished,int _minPrice ) {
    // 아무것도 살 수 없는 잔액상태일때 실행
    if (_wallet < _minPrice)
    {
        System.out.println
                ("더 이상 살 수 있는 메뉴가 없어 자동으로 구입이 종료됩니다.");
        EndOrderInform(_wallet, _basket);
        _isPurchaseFinished.set(0, false);
        _isPurchaseFinished.set(1, false);
    }
}
```

3.8.

3.9.

```
private static void WhileAdditionalOrder(int _wallet,ArrayList<String> _basket,
                                         ArrayList<Boolean> _isPurchaseFinished ,Scanner _inputText)
        // 주문을 더 할 수 있을때 입력받음.
{
    System.out.print("더 주문하실 사항이 있으면 1번 아니면 2번을 입력해주세요\n");
    int _inputIsPurchaseFinishedValue = ParsingInputInt(_inputText);
    if (_inputIsPurchaseFinishedValue == 1)
    {
        _isPurchaseFinished.set(0,false);
    }
    else if (_inputIsPurchaseFinishedValue == 2)
    {
        _isPurchaseFinished.set(0,false);
        _isPurchaseFinished.set(1,false);
        EndOrderInform(_wallet, _basket);
    }
}
```

3.10.

```
private static void ViewMenu()
{
    for (int i = 0; i < DRINKMENU.length; i++)
    {   //메뉴의 원소는 0부터 시작하지만 메뉴판 출력할때는 1부터 시작해야한다.
        System.out.printf("%d", i + 1);System.out.print(". ");
        System.out.printf("%s ", DRINKMENU[i]);
        System.out.printf("%d원 ", DRINKPRICE[i]);
    }
    System.out.println("");
    System.out.println("제품에 해당하는 숫자를 입력하시면 구매가 진행됩니다.");
}
```

3.11.

```
private static void PrintBasket(ArrayList<String> _basket)
        // 딱 장바구니 내용물만 출력한다.
{
    for (int i = 0; i < _basket.size(); i++)
    {
        System.out.printf("%s", _basket.get(i));
        if (i < _basket.size() - 1)
        {
            System.out.print(", ");
        }
    }
}
```

3.12.

```
private static void ViewBasket(ArrayList<String> _basket)
// 장바구니와 관련된 출력 함수
{
    if (_basket.isEmpty())  //비어있으면 종료 ,
    // 처음에 메뉴 잘못선택했을때 더 구입할건지 안할건지
    // 입력창이 뜰때 구입 안한다고 입력하면 발생
    {
        return;
    }
    System.out.print("현재까지 장바구니에 담긴 물건은 ");
    PrintBasket(_basket);
    System.out.print("입니다.\n");
}
```

3.13.

```
private static void PrintWallet(int _nowWallet,boolean _firstPrint)
{
    if (_firstPrint)
        System.out.printf("현재 잔액은 %d원 있습니다.\n\n",_nowWallet);
    else System.out.printf("거스름돈을 %d원 받아,현재 잔액은 %d원 있습니다.\n\n"
            , _nowWallet,_nowWallet);
}
```

3.14.

```
private static int ParsingInputInt(Scanner _nextInt)
// 여기서 메뉴의 입력값, 주문 계속 할지 안할지의 입력값을
// 검사해서 숫자 이외의 값이 들어올경우 에러가 안생기게 해줌
{
    int inputIntText = 0;
    boolean noneException = false;
    while (!noneException)
    {
        try
        {
            inputIntText = Integer.parseInt(_nextInt.nextLine());
            noneException = true;
        }
        catch (NumberFormatException noneInt)
        {
            System.out.println("정수의 숫자만 적어 주십시오.");
        }
    }
    return inputIntText;
}
```

3.15.

```
private static int CalculateDrink(int _currentDrinkPrice, int _inputWallet) // 메뉴 값을 계산 하는 함수
{
    return (_inputWallet - _currentDrinkPrice);
}
```

3.16.

```
private static int FindMinPrice(int[] _menuPriceArray) // 메뉴 최솟값을 계산하는 함수
{
    int minPrice = _menuPriceArray[0];
    for (int i = 0; i < _menuPriceArray.length; i++)
    {
        if ( minPrice > _menuPriceArray[i])
        {
            minPrice = _menuPriceArray[i];
        }
    }
    return minPrice;
}
```

4. 고찰

간단한 로직으로 작동하는 음료수 자판기를 구현하는 과제였지만 생각 외로
고민도 많이 하고 설계를 신중하게 해야 아슬아슬하게 쌓아 올린 불법 건축물은
만들지 않아야겠다는 것을 뼈저리게 느꼈다.

어디서 건드려야 될지도 의문이고 추후 기능 추가, 수정 등을 수월하게 할 수 있다.
너무 못 만들면 수리하는 것보다 다 부숴버리고 새로 만드는 게 나을 생각이 들 정도이다.

현재 제출하는 코드도 처음에는 지갑에 해당하는 변수를 final이 없는 static으로 지정하고
각 함수 void로 설정하고 마음대로 접근하면서 값을 수정했다.
하지만 그렇게 됐을 때 가독성이나 유지보수 측면 및 코드를 설계하는 실력이 전혀 늘지 않을 것으로
판단하고 전부 지우고 새로 작성했다.

다시 새로 작성해 보니 반환 값을 하나밖에 못 받아 불러오고 싶은 것은 많은데 답답할 나름이었다.
어쩔 수 없이 검색을 통해 List 배열로 인스턴스를 생성하면 접근도 할 수 있고 삽입, 삭제, 수정도
가능해서 이용해 보았다.

새로운 함수들이나 예약어를 사용하면서 조금 더 폭넓게 생각하고 구성을 할 수 있었지만
코드의 짜임새는 크게 바뀐 것이 없었다.
그래서 나름대로 변화를 주려고 계속해서 고치기를 반복해 보니 너무 생각 없이 시작했다고 느껴졌다.

코드를 작성하기 전에 글로 써보면서 시작하였지만 조금 더 세밀하게 구성을 짜고 통일성을 지켜
헷갈리지 않게 작성해야 했는데 너무 성급하게 했다.

나중에는 시간에 쫓기다 보니 마음이 급해서 어느 정도 수정한 다음 보고서를 작성하려고
내가 작성한 코드를 켠 다음 설명하는 텍스트 작성하기 위해 생각하면서 입력하는데
너무 말이 안 되는 코드를 작성했고 그게 여러 번 수정본이라는 것을 알아차리는 시간은 길지 않았다.

이미 true인 값을 true로 변경한다고 입력하거나, while문을 일회성 코드로 쓰고 있거나
없어도 될 값들을 많이 만들어서 더욱더 헷갈리게 만드는 등 수많은 기행을 경험했고
아직도 남아 있을 것이다.
경험이 너무 없으니, 시작을 잘못한 것도 있고 구성을 여러 번 안 만든 경험 부족이기 때문에
계속해서 의심하고 새로 짜보고 함수의 역할을 좀 더 세분화 하는 훈련을 해야겠다.