|  |
| --- |
| 인공지능 소프트웨어과 |
| 딕셔너리를 활용한 자판기 프로그램 |
| 파이썬기초 |

|  |
| --- |
| 학번: 2401110262  이한석 |

목차

[개요 3](#_Toc169644571)

[과제의 요구사항 3](#_Toc169644572)

[프로그램 구동방식 3](#_Toc169644573)

[코드 5](#_Toc169644574)

[Drink 클래스 5](#_Toc169644575)

[VendingMachineClass 클래스 5](#_Toc169644576)

[World 클래스 8](#_Toc169644577)

[스크린샷 9](#_Toc169644578)

[고찰 12](#_Toc169644579)

# 개요

이번 과제는 파이썬과 파이썬에서 제공하는 딕셔너리 자료형을 활용해 자판기 프로그램을 만드는 것이다.

## 과제의 요구사항

이번 과제의 목표는 파이썬을 사용해 자판기 프로그램을 작성하는 것이다. 특히, 자판기 메뉴에 딕셔너리 자료형을 적용하여 메뉴 출력과 금액 계산을 보다 효율적으로 처리할 수 있도록 프로그래밍하는 데 중점을 둔다.

음료수를 클래스화하여 자판기 내에서 객체화하여 관리하고, 자판기 또한 클래스화하여 사용자가 두 개 이상의 자판기를 통해 음료수를 구매할 수 있도록 하는 것이다.

각 자판기의 판매 방식은 다음과 같다. 먼저 사용자가 돈을 투입(매개변수로 정수 타입의 돈에 해당하는 변수를 입력)하면, “지갑에 있는 돈을 다 넣었다.”란 안내 문구가 출력된다. 다음으로 자판기는 판매하고 있는 음료수 이름과 가격을 메뉴식으로 출력해준 뒤 스캐너를 통해 사용자로부터 메뉴번호를 입력 받는다. 사용자가 메뉴번호를 입력하면 사용자 자신이 어느 음료를 선택했는지 확인할 수 있도록 자판기는 사용자가 선택한 번호를 토대로 ‘[몇]번 [무슨 음료]를 선택하셨습니다.’란 안내 문구를 출력한다. 마지막 계산 과정에서는 사용자의 투입금액이 사용자가 선택한 음료의 가격 이상인 경우 음료가 나오고, 그렇지 않은 경우는 잔액이 부족하다는 안내 문구를 출력하면서 한 번의 판매행위 루프가 끝나게 된다.

자판기의 판매행위 루프는 사용자가 특정 번호를 입력하여 구매를 종료하기로 선택하거나, 사용자의 투입금액이 자판기가 판매하고 있는 음료의 최저가격 미만이 될 때까지 반복된다..

## 프로그램 구동방식

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

이 프로그램은 첫번째로 음료 클래스가 정의되어 있는 Drink.java, 두번째로 자판기 클래스가 정의되어 있는 VendingMachineClass.java, 마지막으로 main문이 존재하고, 자판기의 판매행위(=소비자의 구매행위)가 이루어지는 World.java 총 세 개의 java파일로 구성된다. 음료수 객체는 자판기 클래스에서 객체화 되며, 자판기 객체는 World 클래스의 main문에서 객체화 된다.

각 클래스에 대해 설명하자면, 먼저 음료수의 속성과 메소드를 정의한 Drink 클래스는 이름과 가격, 재고량 총 세 개의 속성을 갖는다. 생성자에서 위 세 가지 속성이 매개변수로도 들어간다. 해당 클래스는 총 네 가지 메소드를 갖는데, 첫 번째로 음료의 이름을 가져오는데 사용되는 GetName 메소드, 두 번째로 음료의 가격을 가져오는데 사용되는 GetPrice 메소드, 세 번째로 음료의 재고량을 가져오는 GetStock 메소드, 마지막으로 판매된 음료의 재고량을 1 차감시키는 Sell 메소드가 있다. 해당 메소드는 객체 간의 상호작용을 위해 VendingMachineClass의 ReduceStock 메소드가 재고량 속성에 접근하기 위한 목적도 겸한다. 음료수의 클래스화는 지난 과제와의 차이점이기도 한데, 클래스화를 통해서 자판기 관리자는 음료의 종류와 가격을 자유롭게 바꿀 수 있고, 재고량 데이터가 추가됨에 따라 재고관리가 가능해졌다는 장점이 생겼다.

VendingMachineClass 클래스는 속성은 가지고 있지 않은 대신 Drink 클래스를 객체화해서 자신이 판매할 음료수를 생성한다. 메소드로는 판매행위(=소비자 입장에서는 구매행위)를 개시하는 Buy 메소드, 사용자의 선택사항을 계산과정으로 넘겨주는 input 메소드, 음료수 값을 계산하는 Calculate 메소드까지 총 세 가지 메소드를 가진다. 각 메소드는 실생활에서의 제품 판매과정와 마찬가지로 서로 긴밀하게 연결되어 있다. 자판기 객체가 가진 메소드를 통해 사용자의 구매행위를 어떤 단계를 거치는지 따라가보면 아래와 같다.

Buy 메소드 호출

1. (printf문) 최초 잔액 표시
2. (printf문) 메뉴 출력
3. (Scanner 객체 생성 및 사용자 입력 받기) 사용자로부터 음료번호 입력 받기
4. Input 메소드 호출 후 사용자가 입력한 음료 번호 값과 투입 금액 값을 인자로 전달

input 메소드 호출

1. (printf문) 사용자의 금액 투입에 대한 안내 문구 출력
2. (switch-case문) Buy 메소드로부터 전달받은 음료번호 값에 따라 사용자의 선택 확인 안내 문구 출력

Calculate 메소드 호출 후 사용자의 입력 값에 따른 각 음료 객체의 이름, 가격, 사용자의 투입 금액 값을 인자로 전달

Calculate 메소드 호출

1. (printf문) 음료수 받기에 대한 안내 문구 출력
2. 전달받은 사용자의 투입 금액 값에서 음료 객체의 가격을 차감 후 반환

이렇게 Calculate 메소드에서 계산된 최종 잔액 값은 input 메소드로 반환되고, input 메소드는 그 값을 다시 한번 Buy 메소드로 반환한다.(9번 단계) 최종적으로 최종 잔액 값은 input 메소드를 호출했던 코드에서 insertedMoney 변수에 다시 저장되어, 남은 금액에 대한 안내 문구에서 출력되며(10번 단계), Buy 메소드 안의 루프가 실행되는 동안 유지된다.

World 클래스는 main문에서 VendingMachineClass 객체를 인스턴스화하고, 각 객체의 Buy 메소드를 호출하여 음료 구매 절차를 수행한다. 사용자는 각 자판기 객체에 자신의 보유 금액을 전달하기만 하면 음료 구매를 완료할 수 있으며, 자판기 및 음료 객체의 내부 속성과 메소드는 외부에서 접근할 수 없다. 이는 객체 지향 프로그래밍의 캡슐화를 구현하고자 하는 의도로 객체의 내부 상태를 보호하고, 사용자에게 필요한 기능만을 제공하고자 함이다.

# 코드

## Drink 클래스

**package** ch08;

**public** **class** Drink {

//속성

**private** String name;

**private** **int** price;

**private** **int** stock;

//생성자

**public** Drink(String name, **int** price, **int** stock) {

**this**.name = name;

**this**.price = price;

**this**.stock = stock;

}

//메소드

**public** String GetName() {**return** **this**.name;}

**public** **int** GetPrice() {**return** **this**.price;}

**public** **int** GetStock() {**return** **this**.stock;}

**public** **void** sell() {**this**.stock -= 1;}

}

## VendingMachineClass 클래스

**package** ch08;

**import** java.util.Scanner;

**import** java.lang.Math;

**public** **class** VendingMachineClass {

//판매할 음료 객체 생성

Drink drink1 = **new** Drink("콜라", 500, 1);

Drink drink2 = **new** Drink("사이다", 700, 3);

Drink drink3 = **new** Drink("물", 700, 3);

Drink drink4 = **new** Drink("파워에이드", 1000, 3);

String drink1name = drink1.GetName();

String drink2name = drink2.GetName();

String drink3name = drink3.GetName();

String drink4name = drink4.GetName();

**int** drink1price = drink1.GetPrice();

**int** drink2price = drink2.GetPrice();

**int** drink3price = drink3.GetPrice();

**int** drink4price = drink4.GetPrice();

**int** drink1stock = drink1.GetStock();

**int** drink2stock = drink2.GetStock();

**int** drink3stock = drink3.GetStock();

**int** drink4stock = drink4.GetStock();

**public** **void** Buy(**int** insertedMoney) {

//음료 판매 메소드 (매개변수 : 사용자가 투입한 돈)

**while** (insertedMoney>0) {

System.***out***.println("---------------------------------------------------");

System.***out***.printf("현재 잔액: %d원\n",insertedMoney); //1. 최초 잔액 표시

System.***out***.printf("1)%s %d원 2)%s %d원 3)%s %d원 4)%s %d원 0) 구입종료\n",

drink1name,drink1price, drink2name, drink2price,drink3name, drink3price,

drink4name, drink4price); //2. 메뉴 출력

System.***out***.println("---------------------------------------------------");

//3.사용자로부터 음료번호 입력받기

Scanner input = **new** Scanner(System.***in***);

System.***out***.print("음료번호: ");

**int** choice = input.nextInt();

//사용자가 0번을 입력하여 구매종료를 선택했을 때

**if** (choice==0) **break**;

//사용자가 투입한 가격이 판매하고 있는 음료의 최저가격 미만인 경우

**if** (insertedMoney<500) {

System.***out***.println("잔액이 부족합니다.");

**break**;

}

//4.input메소드 호출 후 사용자의 음료선택 번호와 투입 금액값들을 인자로 전달

insertedMoney = input(choice, insertedMoney);

System.***out***.printf("남은 잔액은 %d원 입니다.\n", insertedMoney); //10. 잔액 표시

System.***out***.println("===================================================\n");

}

}

**public** **int** input(**int** passedNum, **int** inserted) {

/\* input 메소드

\* 매개변수 :

\* int passedNum : 사용자가 선택한 음료의 번호

\* int inserted : 사용자가 투입한 금액

\* 반환값 :

\* int inserted : 구매 후 남은 금액(=output메소드에 인수전달 후 반환받은 값)

\*/

System.***out***.println("지갑에 있는 돈을 다 넣었다."); //5. 안내 문구 출력

**switch**(passedNum) {

**case** 1 :

**if**(drink1.GetStock()>=1) {

//6. 사용자의 선택 확인 안내 문구 출력

System.***out***.printf("%d번 %s를 선택하셨습니다.\n", passedNum, drink1name);

inserted = Calculate(drink1name, drink1price, inserted);

ReduceStock(drink1);

**break**;

}

**else** {

//사용자가 선택한 음료의 재고가 0일 때

System.***out***.println("재고가 부족합니다. 다른 제품을 선택해주십시오.");

**break**;

}

**case** 2 :

**if**(drink2.GetStock()>=1) {

System.***out***.printf("%d번 %s를 선택하셨습니다.\n", passedNum, drink2name);

inserted = Calculate(drink2name, drink2price, inserted);

ReduceStock(drink2);

**break**;

}

**else** {

System.***out***.println("재고가 부족합니다. 다른 제품을 선택해주십시오.");

**break**;

}

**case** 3 :

**if**(drink3.GetStock()>=1) {

System.***out***.printf("%d번 %s을 선택하셨습니다.\n", passedNum, drink3name);

inserted = Calculate(drink3name, drink3price, inserted);

ReduceStock(drink3);

**break**;

}

**else** {

System.***out***.println("재고가 부족합니다. 다른 제품을 선택해주십시오.");

**break**;

}

**case** 4 :

**if**(drink4.GetStock()>=1) {

System.***out***.printf("%d번 %s를 선택하셨습니다.\n", passedNum, drink4name);

inserted = Calculate(drink4name, drink4price, inserted);

ReduceStock(drink4);

**break**;

}

**else** {

System.***out***.println("재고가 부족합니다. 다른 제품을 선택해주십시오.");

**break**;

}

**default** : //예외사항 처리(메뉴에 없는 번호 선택)

System.***out***.println("없는 번호입니다. 메뉴에 있는 번호를 선택해주십시오.");

**break**;

}

**return** inserted; //9. 잔돈받기(반환)

}

**public** **int** Calculate(String menuName, **int** price, **int** inserted2) {

/\* Calculate 메소드

\* 매개변수 :

\* String menuName: 사용자가 선택한 음료명

\* int price : 사용자가 선택한 음료의 가격

\* int inserted2 : 사용자가 투입한 금액(input메소드에서 전달받은 값)

\* 반환값 :

\* int inserted2 : inserted2에서 price를 뺀 최종잔액값

\*/

**if** (inserted2 >= price) {

System.***out***.printf("%s가(이) 나옵니다.\n", menuName); //7. 음료수 받기

inserted2 -= price;

**return** inserted2; //8. 계산 후 잔돈받기(반환)

}

**else** { //예외사항 처리(음료값 계산하지 않음)

System.***out***.println("잔액이 부족합니다.");

**return** inserted2;

}

}

**public** **void** ReduceStock(Drink drink) {

/\* ReduceStock 메소드

\* 매개변수:

\* Drink drink - 재고를 줄일 Drink 객체

\*/

drink.sell();

}

}

## World 클래스

**package** ch08;

**import** java.util.Scanner;

**public** **class** World {

**public** **static** **void** main(String[] args) {

//첫 번째 자판기 객체 생성 및 음료구입 메소드 호출

VendingMachineClass test1 = **new** VendingMachineClass();

test1.Buy(3200);

//두 번째 자판기 객체 생성 및 음료구입 메소드 호출

VendingMachineClass test2 = **new** VendingMachineClass();

test2.Buy(1840);

}

}

# 스크린샷

1. 두 개의 자판기 객체를 생성한 뒤 첫 번째 자판기 객체에서는 0을 입력하여 구입을 종료하고, 두번째 자판기 객체로 넘어가서 (4)파워에이드→(2)사이다 순으로 구입한 뒤 마지막으로 (1)콜라를 구입하려고 할 때는 금액이 모자라서 구입이 강제 종료되는 경우

텍스트, 스크린샷, 문서, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 자판기에 없는 번호 선택하는 경우

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. 자판기 객체 두 개 생성 후 첫 번째 자판기 객체에서 투입 금액을 소진하여 구매가 강제 종료된 후 두 번째 자판기 객체에서의 구매로 넘어가는 경우

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

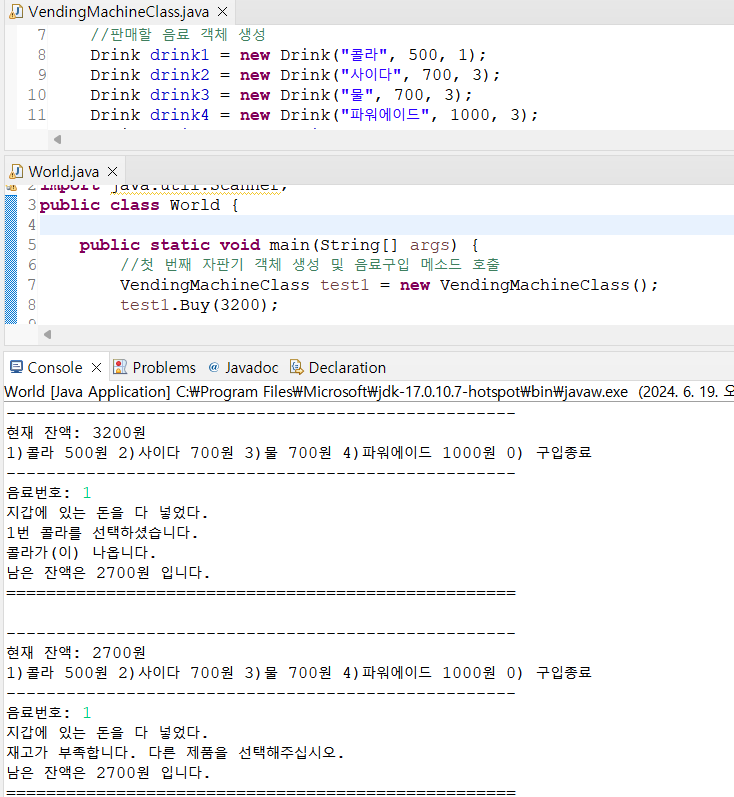
자동 생성된 설명

1. 두 개의 자판기 객체 생성 이후 0을 입력해 어느 자판기에서도 구매하지 않는 경우

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. 콜라 객체 생성시 재고를 1로 하고 자판기 객체 생성 후 콜라를 두 번 구매해서 재고부족으로 구매불가한 경우



# 고찰

처음에는 딕셔너리형 메뉴를 menu = {'콜라': 500, '사이다': 500, '물': 800, '파워에이드': 1000}라고 정의하였다. 딕셔너리를 사용함으로써 메뉴를 출력하는 것은 매우 편리했다. 그러나 메뉴 선택 함수를 설계하면서 딕셔너리는 리스트와 달리 숫자 인덱스를 사용하여 각 항목에 접근할 수는 없다는 점을 깨닫게 되었고, 메뉴 번호를 통해 주문을 받으려는 계획에 차질이 생겼다. 이러한 문제를 해결하기 위해 중첩 딕셔너리라는 형식을 택하게 되었다.

하지만, 딕셔너리에서는 리스트와 달리 숫자 인덱스를 사용하여 각 항목에 접근할 수 없다는 것을

지난 번 과제에서는 사용자가 투입한 금액이 판매하고 있는 음료의 최저가격 미만으로 내려가지 않는 한 while문의 루프가 반복되어 사용자의 의사와 무관하게 투입한 금액을 끊임없이 사용할 수밖에 없는 한계가 있었다. 이번 과제에서는 그런 문제점을 개선하고 사용자의 선택을 존중하는 의도에서 음료 선택과정에서 구입종료라는 선택지(0)를 넣었다.

객체를 활용한 마법사 게임 과제에서 객체와 객체의 상호작용-마법사 클래스의 공격 메소드에 몬스터 객체를 전달하여 몬스터의 hp를 차감하는 식의 상호작용- 방식에 익숙하지 않아서 공격 클래스를 따로 생성했었다. 이번 과제에서는 그런 점을 보완하기 위해 자판기의 재고 차감 방식에 있어서 고민했고, 재고를 차감하는 메소드(ReduceStock)를 Drink 객체를 매개변수로 하여 호출 시 인자로 전달한 Drink 객체의 재고량을 1 차감하는 기능을 수행하도록 했다.

절차지향 방식으로 작성했던 프로그램을 객체지향 방식으로 바꾸면서 결과적으로 코드가 더 길어지긴 했지만, 결과적으로 코드의 재사용성, 유지보수성, 보안성 등 측면에서 더 개선되었다는 점을 직접 체감해볼 수 있어서 유익했던 시간이었다.