

8주차 과제

study 할래?

스터디내용

자바의 인터페이스에 대해 학습하세요.

1. 인터페이스 정의하는 방법
2. 인터페이스 구현하는 방법
3. 인터페이스 레퍼런스를 통해 구현체를 사용하는 방법
4. 인터페이스 상속
5. 인터페이스의 기본 메소드 (Default Method), 자파8
6. 인터페이스의 `static` 메소드, 자바8
7. 인터페이스의 `private` 메소드, 자바9

0. 인터페이스의 필요성

우리가 개발작업을 하기 위해서는 정해진 요구사항이 있어야한다. 동시에 이 요구사항들은 구체적이며 명확해야 한다. 이러한 가이드라인을 제공해 주는 역할을 인터페이스가 맡고 있으며 최종적 목표는 자바의 다형성을 이용하여 개발코드 수정을 줄이고 유지보수성을 높이기 위함이다.

자세한 사항은 6주차 과제를 살펴보도록한다.

1. 인터페이스 정의하는 법

```
interface i { //main이 아니어서 public이 붙지 않음
    //상수
    //타입 상수명 = 값;
    //인터페이스에서 값을 정해줄테니 바꾸지 말고 제공해주는 값만 참조해라 (절대적)
    String name = "은수";
    //추상 메소드
    //타입 메소드명 (매개변수, ...);
    //가이드만 줄테니 오버라이딩해서 재구현해라 (강제적)
    public String method1();
    public String method2(String input);
    //===== JAVA VERSION 8 =====//
    //디폴트 메소드
    //default 타입 메소드명 (매개변수, ...) {구현부}
    //인터페이스에서 기본적인 제공은 해주지만, 말에 만들면 각자 구현해서 써라 (선택적)
    default String defaultMethod1(){return "defaultMethod1";};
    default String defaultMethod2(String input){return "defaultMethod2";};
    //정적 메소드
    //static 타입 메소드명 (매개변수) {구현부}
    //인터페이스에서 제공해주는것으로 무조건 사용 (절대적)
    static String staticMethod1(){return "staticMethod1";};
    static String staticMethod2(String input){return "staticMethod2";};
}
```

추상메소드를 인터페이스에 추가한다면, 이를 implements한 모든 클래스에서 강제적으로 추상메소드를 구현해야하고 구현하지 않을시 전부 에러가 난다.

디폴트 메소드를 정의하고 기본 구현부를 제공하고 만약 기본 구현부가 맘에 들지 않으면 각자가 오버라이딩을 하여 재구현할 수 있도록 선택적인 메소드를 가이드한다면 시스템 운영 유지보수성이 확보가 될 것이다.

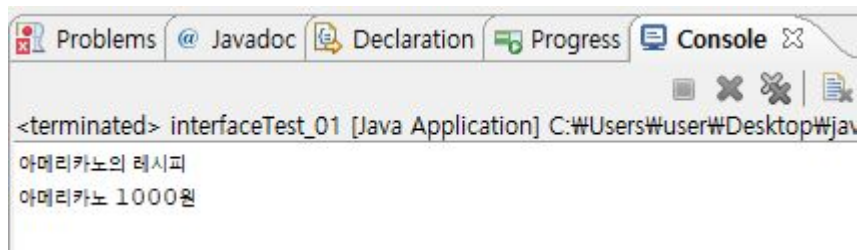
결국 이미 운영되고 있는 시스템에서 추가 요건으로 인해 불가피하게 반영을 해야할 때 디폴트 메소드를 쓰면 효과적이란 소리다.

2. 인터페이스 구현하는 법(6. static in interface)

```
3 interface MainBranch {
4     static void recipe(int itemNumber) {
5         switch (itemNumber) {
6             case 1: System.out.println("아메리카노의 레시피"); break;
7             case 2: System.out.println("카피라떼의 레시피"); break;
8             case 3: System.out.println("카라멜마끼아또의 레시피"); break;
9         }
10    }
11    static void price(int itemNumber) {
12        switch (itemNumber) {
13            case 1: System.out.println("아메리카노 1000원"); break;
14            case 2: System.out.println("카피라떼 2000원"); break;
15            case 3: System.out.println("카라멜마끼아또 3000원"); break;
16        }
17    }
18 }
19
20 class gimpoAirportBranch implements MainBranch {
21 }
22
23 public class interfaceTest_01 {
24
25     public static void main(String[] args) {
26         MainBranch gimpo = new gimpoAirportBranch();
27         MainBranch.recipe(1);
28         MainBranch.price(1);
29     }
30
31 }
```

커피 체인점을 운영중이다. 레시피와 가격은 모든 분사가 동일하게 운영해야한다. 변경이 불가하고 모두 동일해야 하기 때문에 정적 메소드를 이용했다.

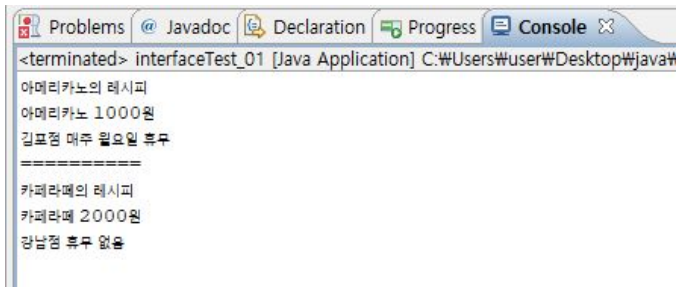
하지만 운영을 하다보니 운영시간은 매장별로 선택할 수 있게 해야했다.



2. 인터페이스 구현하는 법

운영시간은 매장별로 오버라이딩 후 재구현할 수 있게 본사에서 인터페이스를 제공해 주었다.

그렇다면 본사에서 진행하는 이벤트를 진행하는데 특정 매장만 진행이 된다는 가정을 해보자 예를들어 아메리카노 1잔 무료 이벤트를 진행하는데 강남점은 참여하지않고 김포공항점만 참여를 하게된다면?



```
<terminated> interfaceTest_01 [Java Application] C:\Users\User\Desktop\java#
아메리카노의 레시피
아메리카노 1000원
김포점 매우 필요일 후두
=====
카페라떼의 레시피
카페라떼 2000원
강남점 후두 없음
```

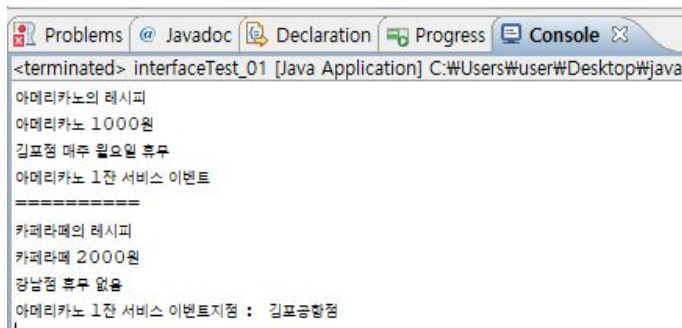
```
3 interface MainBranch {
4     static void recipe(int itemNumber){
5         switch (itemNumber){
6             case 1: System.out.println("아메리카노의 레시피"); break;
7             case 2: System.out.println("카페라떼의 레시피"); break;
8             case 3: System.out.println("카라멜마끼아또의 레시피"); break;
9         }
10    }
11    static void price(int itemNumber){
12        switch (itemNumber){
13            case 1: System.out.println("아메리카노 1000원"); break;
14            case 2: System.out.println("카페라떼 2000원"); break;
15            case 3: System.out.println("카라멜마끼아또 3000원"); break;
16        }
17    }
18    public void hours();
19 }
20
21 class gimpoAirportBranch implements MainBranch{
22     @Override
23     public void hours() {
24         System.out.println("김포점 매우 필요일 후두");
25     }
26 }
27
28 class gangnamBranch implements MainBranch{
29     @Override
30     public void hours() {
31         System.out.println("강남점 후두 없음");
32     }
33 }
34
35 public class interfaceTest_01 {
36
37     public static void main(String[] args) {
38         MainBranch gimpo = new gimpoAirportBranch();
39         MainBranch.recipe(1);
40         MainBranch.price(1);
41         gimpo.hours();
42         System.out.println("=====");
43         MainBranch gangnam = new gangnamBranch();
44         MainBranch.recipe(2);
45         MainBranch.price(2);
46         gangnam.hours();
```

2. 인터페이스 구현하는 법 (5. default in interface)

그렇다면 default 메소드로 김포점만 받아서 재구현하면 된다.

이렇게 본사에서 제공해주는 가이드를 받아 사용하면 편리하다. 하지만 implements하지 않고 자신만의 메서드를 구현한다면 본사에서 주는 recipe나 price 정책의 도움을 받을 수 없을 것이다.

이렇게 인터페이스를 알아보았다. 정리하자면 인터페이스는 추상메소드와 상수를 통해 강력한 강제성을 가지게하여 인터페이스를 implements한 클래스가 동일한 동작을 수행하도록 보장한다. 그리고 Java8부터 추가된 사항으로 유연성까지 확보해 주었다.



```
<terminated> interfaceTest_01 [Java Application] C:\Users\user\Desktop\Wjava
아메리카노의 레시피
아메리카노 1000원
김포점 매주 월요일 휴무
아메리카노 1잔 서비스 이벤트
=====
카페라떼의 레시피
카페라떼 2000원
강남점 휴무 없음
아메리카노 1잔 서비스 이벤트지점 : 김포공항점
```

```
19 public void hours();
20 //JAVAS
21 default void event(){
22     System.out.println("아메리카노 1잔 서비스 이벤트지점 : 김포공항점");
23 }
24 }
25
26 class gimpoAirportBranch implements MainBranch{
27     @Override
28     public void hours() {
29         System.out.println("김포점 매주 월요일 휴무");
30     }
31
32     @Override
33     public void event() {
34         System.out.println("아메리카노 1잔 서비스 이벤트");
35     }
36 }
37
38 }
39 class gangnamBranch implements MainBranch{
40     @Override
41     public void hours() {
42         System.out.println("강남점 휴무 없음");
43     }
44 }
45
46 public class interfaceTest_01 {
47
48     public static void main(String[] args) {
49         MainBranch gimpo = new gimpoAirportBranch();
50         MainBranch.recipe(1);
51         MainBranch.price(1);
52         gimpo.hours();
53         gimpo.event();
54         System.out.println("=====");
55         MainBranch gangnam = new gangnamBranch();
56         MainBranch.recipe(2);
57         MainBranch.price(2);
58         gangnam.hours();
59         gangnam.event();
60         //MainBranch.event(); 요류 static이 아니어서 안됨
61     }
62 }
```

3. 인터페이스 레퍼런스를 통해 구현체를 사용

