JAVA interface

insterface란
인터페이스 사용법
인터페이스가 적용된 클래스의 객체 생성
인터페이스의 사용 목적

실생활에서도 인터페이스라는 단어는 많이 사용된다. 인터페이스라는 용어가 무엇인지 물어보면 말로 설명하기는 어렵지만 대략적인 느낌을 알고 있을 것이다.

interface는 inter(사이) + face(얼굴)이라는 어원에서 시작된 것으로 두 사람, 혹은 사물 중간에서 둘을 이어주는 매개체를 말한다.

컴퓨터와 관련하여 정의한다면 인터페이스는 서로 다른 시스템 혹은 장치 간의 상호작용을 정의하는 방식 혹은 상호작용의 매개체를 말한다.

Tv 리모콘의 버튼을 인터페이스라 말한다. 리모콘이 사람과 Tv가 상호작용 할 수 있는 매개체이기 때문이다.

커피 자판기의 버튼도 인터페이스라 칭한다. 사람과 기계의 동작이 상호작용 할 수 있는 매개체이기 때문이다.

여기서 중요한 것은 이러한 Tv 리모콘, 커피 자판기는 우리가 버튼을 누르면 어떠한 결과를 주는지 알지만, 그 결과가 나오는 원리나 과정은 알지 못하며,

알 필요도 없다.

이러한 내용을 자바에 대입해보겠다.

Tv 리모콘의 전원버튼을 누르면 Tv가 켜진다. -> powerOn() 메서드를 호출하면 Tv 전원을 켜는 기능이 실행된다.

전원버튼을 누르면 어떤 원리나 과정을 거치는지는 모른다. -> powerOn() 메서드를 호출하면 기능이 실행되나, 메서드 안의 내용은 모른다.

이렇게 특정 메서드를 호출하면 어떠한 결과가 도출되지만, 메서드 안에 작성된 코드는 모르는것, 이러한 메서드를 추상메서드라 한다.

이러한 설명을 코드로 보자면 다음과 같을 것이다.

```
public void method_1(){
   System.out.println(“123”);
}
```

위 method_1 메서드는 실행하면 “123”이 출력된다. 우리는 작성된 코드를 통해 어떤 동작을 통해 “123”이 출력되는지 알 수 있다. 일반적인 메서드이다.

아래 코드는 추상메서드를 작성한 것이다.

```
public void method_2(); //이 메서드를 실행하면 “123”이 출력된다.
```

추상메서드는 위와 같이 메서드는 정의되어 있지만 메서드의 내용부{} 가 없다. 어떤 메서드가 정의는 되어 있지만 그 내용이 없는 것이다.

자바의 클래스와 비슷한 형태를 지녔지만 위와 같은 추상메서드만 존재하는 자료형을 인터페이스(interface)라고 한다.

그렇다... 인터페이스는 자료형이다... 또 다른 자료형이 나왔다...ㅜㅜ. 하지만 클래스와 별 차이 없으니 미리 겁먹을 필요는 없다!

인터페이스를 코드로 보면 다음과 같다.

```
interface Printable{
    public void aaa();
    public int bbb();
    public String ccc(int a);
    ...
}
```

전체적인 틀은 class와 동일하다. 위 코드에서 interface라는 단어를 class로 변경하면 클래스와 정말 큰 차이가 없다. 하지만 보기와는 다르게 interface는 class와 큰 차이점이 있다. 아래는 class와 interface의 공통점과 차이점을 나열한 것이다.

공통점

- 전체적인 틀이 똑같음. class Aaa{...} , interface Aaa {}
- interface명과 class명은 앞 글자를 대문자로 작성 함.
- 둘 다 사용자 정의 자료형임.

차이점

- 클래스 안에는 맴버변수, 메서드의 정의, 생성자가 올 수 있다.
- 인터페이스 안에는 추상메서드만 존재한다. 즉, 맴버변수도 없고, 일반적인 메서드(내용부가 있는 메서드)도 없으며, 생성자도 없다.

여기서 주목해야 할 차이점은 인터페이스에는 생성자가 없다는 것이다.

생성자가 없다는 말은 인터페이스는 클래스처럼 객체를 생성해서 사용할 수 없다는 말이다. 그럼 인터페이스는 어떻게 사용해야 할까...

인터페이스는 사실 인터페이스만으로는 어떤 것도 할 수 없다. 결국, 인터페이스는 혼자선 아무 것도 못하기 때문에 누군가에게 얹혀살아야 한다.

그리고 얹혀 살게 될 대상은 바로 클래스다. 인터페이스의 이러한 특징은 사실 인터페이스라는 문법이 나온 본질적 의미를 알아야한다.

인터페이스의 탄생 배경을 알면 왜 인터페이스는 생성자가 없는지, 왜 클래스와 함께 사용했을 때에만 의미가 있는지 조금은 파악이 될 것이다.

인터페이스의 탄생 배경과 본질적 의미는 말로 설명이 어렵기 때문에 강의 시간에 말로 풀겠다.

새로운 문법을 배우면 해당 문법을 사용했을 때의 이점은 무엇인지, 언제 사용하면 좋은지, 왜 써야하는지 등에 대한 고민을 하는 것은 아주 좋은 학습방법이다. 이러한 내용을 고민하지 않으면 문법이 머리 속에는 있지만, 문제가 주어지면 필요한 문법을 머리에서 끄집어내지 못할 뿐더러, 응용하지 못한다. 실습을 통해 어려운 문제를 여러분들이 힘들어하는 걸 알면서도 풀어보라는 것은 주어진 요구사항을 해결하기 위해 배운 문법 중 어떤 문법을 사용해야하는지 최대한 다수의 경험을 통해 '이 문법은 이럴때 사용하면 되는 거구나, 이 문법을 이렇게도 활용할 수 있구나'를 알려주기 위해서다. 이런 경험이 쌓여야 배운 문법을 문제 해결에 적용할 수 있는 힘이 생긴다. *(내용이 어려워 여러분들 표정이 좋지 않으면 저도 기분이 좋지 않아요. 어려운 문제 풀어보라고 하는게 마음 편하지 않습니다... 그래도 해봐야해요. 피하고 싶은걸 다 피해버리면 성장하지 못하잖아요? 화이팅!!)*

인터페이스 학습 도중 뜬금없이 이런 말을 하는 이유는 인터페이스의 사용 이유와 목적 등을 이야기하겠지만 실무를 경험하지 않으면 인터페이스를 왜 사용해야하는지 받아들이기 힘들기 때문이다. 상속은 내용이 많고 어렵다. 그럼에도 상속을 사용했을 때의 이점을 아주 조금은 느껴봤을 것이다. 인터페이스의 학습량은 상속 내용의 반도 되지 않으며, 익혀야할 문법의 분량이 적다. 하지만.. 인터페이스는 어렵게 느껴질 것이다. 사용 목적과 이점을 잘 느끼지 못하기 때문이다. 하지만 실무에서 무조건 사용한다. 그러니 현재는 인터페이스를 왜 쓰는지 감이 잘 오지 않아도 일단 문법만이라도 익힌다고 생각하자!

인터페이스는 클래스의 붙은 인증마크라고 생각하면 된다.

우리가 물건을 살 때, 잘보면 여러 인증 마크가 붙어 있는 것을 볼 수 있다. 이러한 인증마크는 제품이 어떠한 기준을 충족시킬 경우 부여된다.

클래스와 인터페이스가 이러한 관계라 생각하면 된다. 클래스는 제품이고 인터페이스는 클래스라는 제품이 어떠한 기준을 충족한다는 것을 확인시켜주는 인증마크와 비슷한 것이다. 그러니, 인증 마크만으로는 아무런 의미가 없다. 인증마크는 특정 제품에 부여됨으로써 그 제품을 더 가치있게 만들어주는 역할을 한다. 결국 인터페이스는 단독으로 사용이 불가능하고, 클래스에 얹혀 있을 때, 구실을 할 수 있도록 설계되었다.

인터페이스의 사용을 위해 먼저 인터페이스를 만든다. 아래는 인터페이스 예시이다.

```
interface Printable{
  //흑백 프린터 실행
   public void blackPrint();
   //컬러 프린터 실행
   public void colorPrint();
   //남은 잉크량 확인
   public int getLeftInk();
}
```

앞서 말한것처럼 인터페이스는 위와 같이 추상메서드만 가진다. 이러한 추상메서드는 메서드의 기능 설명은 있지만 그 기능이 어떠한 코드로 구현되는지는 작성하지 않는다. 이러한 메서드는 마치 제품에 인증마크를 부여할 수 있는 충족조건이 된다. 클래스가 인증마크를 받기 위한 조건을 충족시키기 위해서는 인터페이스에 선언된 추상메서드를 구현해야 한다.

인터페이스를 클래스에 적용하기 위해선 다음과 같이 클래스명 뒤에 'implements 인터페이스'를 추가해준다.

```
class SamsungPrinter implements Printable{
  ...
}
```

implements는 프로그래밍에서 많이 사용하는 용어로 '구현 ' 이라는 뜻이다. 결국, 위 클래스의 선언문을 해석하면

Printable 인터페이스를 구현하여 SamsungPrinter 클래스를 만들겠다. 라는 의미이다.

Printable 인터페이스를 구현한다는 것은 인터페이스에 정의된 추상메서드의 내용 부분을 구현한다는 것이다.

이렇게 클래스에 'implements Printable'을 사용하면 class에 바로 오류가 나는 것을 확인할 수 있다. 그 이유는 앞서 설명한대로

해당 클래스는 이제 Printable 인터페이스에서 정의된 추상메서드를 구현해야 하지만 아직 하지 않았기 때문이다.

그럼 이제 실제 인터페이스에서 정의된 추상메서드를 클래스에서 구현한다. 구현한다는 것은 메서드의 내용부를 작성한다는 것이다.

```
class SamsungPrinter implements Printable{
  //흑백 프린터 실행
  public void blackPrint(){
    ...
  }
  //컬러 프린터 실행
  public void colorPrint(){
    ...
  }
  //남은 잉크량 확인
  public int getLeftInk(){
    ...
  }
}
```

좌측과 같이 인터페이스에서 정의한 추상메서드를 클래스에서 구현하면 인터페이스의 기능은 끝이다. 참고로 인터페이스가 구현된 클래스도 우리가 알고 있는 맴버변수, 메서드의 정의, 생성자 등은 사용할 수 있다.

인터페이스의 추상메서드를 구현한 클래스는 지금껏 해왔던 것처럼 객체를 생성하여 사용하면 된다.

Printable 인터페이스를 구현한 객체를 생성하는 또 다른 방법이 존재한다.

```
class SamsungPrinter implements Printable{
  //흑백 프린터 실행
   public void blackPrint(){ ...}
   //컬러 프린터 실행
   public void colorPrint(){ ...}
   //남은 잉크량 확인
   public int getLeftInk(){ ...}
}

class LgPrinter implements Printable{
  //흑백 프린터 실행
   public void blackPrint(){ ...}
   //컬러 프린터 실행
   public void colorPrint(){ ...}
   //남은 잉크량 확인
   public int getLeftInk(){ ...}
}
```

인터페이스가 적용된 클래스의 객체의 생성 방법은 우리가 아는 방식과 같다.

```
SamsungPrinter p1 = new SamsungPrinter();
LgPrinter p2 = new LgPrinter();
```

추가적으로 다음과 같이 인터페이스를 구현한 클래스의 객체는 인터페이스형으로 생성 가능하다.

```
Printable p3 = new SamsungPrinter();
Printable p4 = new LgPrinter();
```

이러한 인터페이스는 사용하면 어떤 이점이 있을까? 인터페이스는 기능 구현을 위해 꼭 필요한 문법은 아니다. 오히려 설계와 관련된 문법이기 때문에 개발 경험이 없는 우리가 인터페이스의 사용목적과 이유를 느끼기에는 한계가 있다. 하지만 유지보수 측면에서 분명한 장점은 있다. 다음 예시를 보자.

```
class SamsungTV{
   public void powerOn(){
      System.out.println("삼성TV-전원 켬");
   }
    public void powerOff(){
      System.out.println("삼성TV-전원 끔");
   }
   public void volumeUp(){
      System.out.println("삼성TV-볼륨 업");
   }
   public void volumeDown(){
      System.out.println("삼성TV-볼륨 다운");
   }
}
```

```
class LgTV{
   public void turnOn(){
      System.out.println("엘지TV-전원 켬");
   }
    public void turnOff(){
      System.out.println("엘지TV-전원 끔");
   }
   public void soundUp(){
      System.out.println("엘지TV-볼륨 업");
   }
   public void soundeDown(){
      System.out.println("엘지TV-볼륨 다운");
   }
}
```

위의 클래스는 삼성티비와 엘지티비를 구현한 클래스들이다. 티비 제조사가 다르기 때문에 당연히 다른 개발자가 각 제조사의 티비를 만들었을 것이며, 동일한 기능이지만 메서드명과 구현 내용이 다를 수 밖에 없다.

이러한 티비 클래스 구현 후 티비 설치 기사가 삼성 티비를 설치한다고 가정해보자

```
public static void main(String[] args){
    SamsungTV tv = new SamsungTV();
    tv.powerOn();
    tv. volumeUp();
    tv. volumeDown();
    tv.powerOff();
}
```

왼쪽의 코드를 보면 알겠지만, 티비를 바꾸기 실행하기 위해

거의 대부분의 코드가 수정되어야 한다.

더욱이, 각 제조사의 기능을 실행하는 메서드명이 모두 다르기 때문에

설치기사는 각 제조사의 기능을 익히고 있어야 한다.

설치한 삼성티비를 엘지 티비로 변경하려면?

```
public static void main(String[] args){
    LgTV tv = new LgTV();
    tv.turnOn();
    tv. soundUp();
    tv. soundDown();
    tv.turnOff();
}
```

앞선, 각 제조사의 티비 클래스에 인터페이스를 적용해보자.

```
interface TV{
   //전원 ON
   public void turnOn();
   //전원 OFF
   public void turnOff();
   //음량 업
   public void volumeUp();
   //음량 다운
   public void volumeDown();
}
```

```
class SamsungTV implements TV{
   public void turnOn(){
      System.out.println("삼성TV-전원 켬");
   }
    public void turnOff(){
      System.out.println("삼성TV-전원 끔");
   }
   public void volumeUp(){
      System.out.println("삼성TV-볼륨 업");
   }
   public void volumeDown(){
      System.out.println("삼성TV-볼륨 다운");
   }
}
```

```
class LgTV implements TV{
   public void turnOn(){
      System.out.println("엘지TV-전원 켬");
   }
    public void turnOff(){
      System.out.println("엘지TV-전원 끔");
   }
   public void volumeUp(){
      System.out.println("엘지TV-볼륨 업");
   }
   public void volumeDown(){
      System.out.println("엘지TV-볼륨 다운");
   }
}
```

티비 기능을 정의하는 인터페이스TV를 만들었다. 이 인터페이스는 모든 티비 기능의 표준을 정한것이라고 보면 된다.

그리고 각 제조사가 티비를 만들 때 이러한 티비 표준에 맞춰 제작을 하기 위해 인터페이스를 구현하여 각 기능을 만들었다.

이렇게 표준을 맞추면 삼성티비와 엘지티비를 개발하는 사람이 다르고, 각 기능의 구현 방법은 다르지만, 설치기사는 기능 사용법(메서드명)만 알면 된다.

이렇게 인터페이스를 구현한 티비 중 삼성티비를 설치기사가 설치한다고 보자.

```
public static void main(String[] args){
    TV tv = new SamsungTV();
    tv.turnOn();
    tv. volumeUp();
    tv. volumeDown();
    tv.turnOff();
}
```

설치한 삼성티비를 엘지 티비로 변경하려면?

```
public static void main(String[] args){
    TV tv = new LgTV();
    tv.turnOn();
    tv. volumeUp();
    tv. volumeDown();
    tv.turnOff();
}
```

인터페이스를 구현한 클래스들은 인터페이스에 정해진대로 동일한 메서드명을 사용하기에 티비 객체를 변경한다 하더라도 사용법이 달라지지 않는다. 그렇기 때문에 각 객체마다 제각기 달랐던 여러 메서드명을 모두 숙지할 필요도 없어졌다. 또한, 동일한 사용법으로 삼성티비와 엘지티비 개발자가 구현한 내용대로 실행되기 때문에 설치 기사는 구현 내용까지 신경 쓸 필요가 없다. 삼성티비든 엘지티비는 전원을 켜려면 turnOn()메서드만 호출하면 된다는 것만 인지하면 되는 것이다.

이렇게 인터페이스를 사용하면 객체 변경이 일어날 때 수정해야 하는 코드를 최소화 할 수 있기 때문에, 인터페이스 사용 여부에 따라 유지보수 측면에서 꽤나 큰 이점을 가져올 수 있다.