인터페이스

주차	42주차
스터디 일자	@2024/09/20

인터페이스 (Interface)

> 인터페이스란? **추상 메소드(Abstract Method)만으로 이루어진 클래스**이다.

- 추상메소드는 구현 내용 없이 선언만 해놓은 형태로 정의된다.

  즉, 인터페이스 내의 모든 메소드는 선언만 되어 있을 뿐, 그 내용은 없다.

```
public interface Car{
    // 인터페이스 내의 변수는 반드시 static final로 지정한다.
    public static final int variable = 10;

    // 인터페이스 내에서 지정한 메소드는
    // 반드시 이 인터페이스를 구현하는 클래스가 구현해야 한다.
    // -> 즉, 클래스에게 특정 메소드들을 구현하도록 강제할 수 있다.
    public void Ride();

    public void RideReverse();

    public void ChangeGear();
```

```
    public double Acceleration();
}
```

```
public class GasolineCar implements Car{
    public void Ride(){
        //...
    }

    public void RideReverse(){
        //...
    }

    public void ChangeGear(){
        //...
    }

    public double Acceleration(){
        //...
    }

    public void ChargeGasoline(int Amount){
        //...
    }
}
```

→ 클래스 옆에 implements 키워드를 붙여 해당 클래스가 어떤 인터페이스의 구현체인지 나타낸다.

- 어떤 인터페이스를 구현하는(implements) **클래스는 반드시 자신이 구현하는 인터페이스에 선언된 모든 메소드를 구현해야 한다**. 그렇지 않으면 오류 발생
- 또한, 인터페이스 내부의 변수는 반드시 **static final**로 설정되어야 한다.

→ 이는 곧, 해당 인터페이스를 구현하는 클래스는 선언된 변수 값을 변경할 수 없음을 말한다.

💡 이처럼 인터페이스는 클래스에 특정 규칙을 찍어주는 것과 같은 역할을 한다!

인터페이스가 필요한 이유?

- 동물원의 사육사가 하는 일은 아래와 같다.

```
동물원(zoo)의 사육사(zookeeper)가 있다.
육식동물(predator)이 들어오면 사육사(zookeeper)는 먹이를 던져준다(feed).
호랑이(tiger)가 오면 사과(apple)를 던져준다.
사자(lion)가 오면 바나나(banana)를 던져준다.
```

위 조건을 코드로 나타내면 아래와 같다.

```
class Animal {
    String name;
```

```
    void setName(String name) {
        this.name = name;
    }
}

class Tiger extends Animal {
}

class Lion extends Animal {
}

class ZooKeeper {
    void feed(Tiger tiger) {  // 호랑이가 오면 사과를 던져 준다.
        System.out.println("feed apple");
    }

    void feed(Lion lion) {  // 사자가 오면 바나나를 던져준다.
        System.out.println("feed banana");
    }
}

public class Sample {
    public static void main(String[] args) {
        ZooKeeper zooKeeper = new ZooKeeper();
        Tiger tiger = new Tiger();
        Lion lion = new Lion();
        zooKeeper.feed(tiger);  // feed apple 출력
        zooKeeper.feed(lion);  // feed banana 출력
    }
}
```

🚨 이렇게 하면 원하는 조건을 만족하지만 악어, 표범 등등 동물들이 계속 추가된다면 ZooKeeper는 클래스가 추가될 때마다 매번 동물 먹이에 맞는 feed 메서드를 추가해야 한다.

→ 이럴때 인터페이스를 사용하면 간단히 문제를 해결할 수 있다.

```
interface Predator {
}
```

```
class Tiger extends Animal implements Predator {
    public String getFood() {
        return "apple";
    }
}

class Lion extends Animal implements Predator {
    public String getFood() {
        return "banana";
    }
}
```

이렇게 같은 인터페이스를 사용하도록 해주면 아래와 같이 하나의 메서드로 모든 동물들의 `feed()` 를 표현할 수 있다.

```
class ZooKeeper {
    void feed(Predator predator) { // 인터페이스로 대체 o
        System.out.println("feed " + predator.getFood());
    }
}
```

추상클래스 (Abstract Class)

> 추상 클래스란? **추상 메소드(Abstract Method)를 포함한 클래스**를 말한다.

```
public abstract class Car{
    public void Ride(){
        //...
    }

    public void RideReverse(){
        //...
    }

    // 이 부분은 추상 메소드로, Car를 상속받는 클래스 내부에서 구현한다.
    public abstract void ChangeGear();

    public abstract double Acceleration();
}
```

일반 메소드들도 가지고 있지만 **추상 메소드를 포함한 형태**의 클래스로 구현된다.

추상 클래스의 구체화는 **클래스 간의 상속**을 통해 이루어진다.

```
public class GasolineCar extends Car{
    // Car 내에 구현된 일반 메소드는 구현하지 않아도 사용 가능
```

```
    // Ride, RideReverse

    // 추상 메소드 구현
    public void ChangeGear(){
        //...
    }

    // 추상 메소드 구현
    public double Acceleration(){
        //...
    }

    // 객체 내의 일반 메소드
    public void ChargeGasoline(int Amount){
        //...
    }
}
```

추상 클래스를 상속받는 클래스는 반드시 부모 추상 클래스의 모든 추상 메소드를 구현해야 한다.

구현하지 않으면 오류 발생 !

인터페이스와 추상 클래스가 주는 이점

인터페이스는 그를 구체화하는 클래스에 어떠한 규칙을 주는 것이고,

추상 클래스 또한 추상 클래스 내부에 선언된 추상 메소드를 반드시 구현하도록 해 인터페이스의 기능을 수행하는 것이다.

이렇게 클래스에 특정 규칙을 부여하는 것은 **곧 클래스에 하위 개념을 두는 것과 같은데,** 이는 **다형성을 통한 유지보수성의 증가**로 이어지기 때문에 큰 장점을 갖는다.

인터페이스와 추상 클래스의 차이

implements와 extends

인터페이스의 구현체임을 나타내는 implements 키워드 뒤에는 여러 인터페이스가 올 수 있다.

→ 즉, **한 클래스는 여러 인터페이스의 구현체가 될 수 있다.**

```
public class Vehicle implements Car, Bicycle {
    // Car 클래스의 추상 메소드 구현
    public void RideCar(){
        //...
    }

    // Bicycle 클래스의 추상 메소드 구현
    public double RideBicycle(){
        //...
    }
}
```

but, 추상 클래스는 클래스 간의 **상속을 통해** 추상 메소드를 이어받기 때문에,

→ **하나의 클래스는 단 하나의 추상 클래스를 상속받을 수 있다.**

❓ 그렇다면 추상 클래스는 왜 쓰는 걸까?

추상 클래스의 의의

만약 Car 인터페이스를 구현하는 클래스의 종류가 100가지 정도 된다고 가정해보면

만약 이 모든 Car 인터페이스를 구현하는 클래스들의 `Ride()` 메소드가 동일한 방식으로 작동한다고 해도, 개발자는 같은 내용의 `Ride()` 메소드를 100번 작성해야 한다.

why? 인터페이스에 선언된 **모든 메소드**는 내용이 없기 때문이다. (추상 메소드)

```
public class GasolineCar implements Car{
    public void Ride(){
        //구현 내용
    }
    //(생략)
}

public class GasCar implements Car{
    public void Ride(){
        //구현 내용
    }
    //(생략)
}

public class ElectricCar implements Car{
    public void Ride(){
        //구현 내용
    }
    //(생략)
}

public class HydroCar implements Car{
    public void Ride(){
        //구현 내용
    }
    //(생략)
```

```
}

... 100개
```

이런 경우 Car 추상 클래스를 상속받으면, Ride와 같이 공통적인 메소드를 반복해서 구현하지 않아도 되겠죠? ^^

결론

이외에도 인터페이스를 사용하는 여러 디자인 패턴이 있어요.
상세히 알고 싶으시다면, 이 부분은 자바 디자인 패턴을 따로 공부하시면 되요.

인터페이스는 잘 사용하면 프레임워크나 배포되는 공용 컴포넌트, 유틸 등을 개발할 때 아주 효과적이에요.
다만, 전제가 있어요. **잘 사용해야되요.**

반대로, 일반 웹 개발 프로젝트에서는 오히려 복잡도만 증대시키는 상황이 많아요.
다시 말씀드리지만, 인터페이스는 **잘 사용해야되요.**
많은 사람들이 불편해하고 필요성을 못느낀다면 과감히 버리는게 맞아요.

결론은? **잘 사용해야되요.**

https://gofnrk.tistory.com/22

https://wikidocs.net/217

https://4legs-study.tistory.com/m/158