JavaStudyBasic 8

인터페이스 만들기

인터페이스 : 서로 관계가 없는 물체들이 상호작용 하기 위해서 사용하는 장치나 시스템

- TV라는 객체 - 켜고 끄는 기능, 볼륨 조절 기능, 채널 변경 기능 있어야 함.
- 꼭 필요한 기능들을 **구현하지는 않고**, 선언해서 **가지고만 있는 것**

---

1. 인터페이스 만들기 -> 구현하지 않는다

`interface`

```
public interface TV
```

```
public interface TV{
    public int MAX_VOLUME = 100;    //interface에 상수 정의, 변경 불가능
    public int MIN_VOLUME =0;

    public void turnOn();
    public void turnOff();
    public void changeVolume(int volume);
    public void changeChannel(int channel);

}
```

2. 인터페이스 사용하기

`class 생성`

- 인터페이스는 사용할 때 해당 인터페이스를 구현하는 클래스에서 Implements 키워드를 이용

```
public class LedTV implements TV{
    public void on(){
        System.out.println("켜다");
    }
    public void off(){
        System.out.println("끄다");
    }
    public void volume(int value){
        System.out.println(value + "로 볼륨조정하다.");
    }
    public void channel(int number){
```

```
        System.out.println(number + "로 채널조정하다.");
    }
}
```

```
public class LedTVExam{
    public static void main(String args[]){
        TV tv = new LedTV();     //Led TV 객체 생성
        tv.on();                 //메소드 구현
        tv.volume(50);           //메소드 구현
        tv.channel(6);           //메소드 구현
        tv.off();                //메소드 구현
    }
}
```

- 인터페이스가 가지고 있는 메소드를 하나라도 구현하지 않는다면 해당 클래스는 추상 클래스가 된다.
- *(추상클래스는 인스턴스를 만들 수 없음)*
- 참조 변수 타입으로 인터페이스 사용 가능

3. 인터페이스의 default method

원래는 abstract 만 가능했지만, 이제 default랑 static 도 됨

> 인터페이스가 변경이 되면, 인터페이스를 구현하는 모든 클래스들이 해당 메소드를 구현해야했었음. 이런 문제를 해결하기 위하여 인터페이스에 메소드를 구현해 놓을 수 있도록 하였다.

`default`

- 인터페이스가 default 키워드로 선언되면 메소드가 구현될 수 있다.
- 또한 이를 구현하는 클래스는 default 메소드를 `오버라이딩` 할 수 있다.

오버라이딩

- 부모클래스의 메소드를 자식 클래스에서 재정의하여 사용하는 것

1. 인터페이스 생성

```
public interface Calculator {
        public int plus(int i, int j);      //더하는 기능
        public int multiple(int i, int j); //곱하는 기능

        default int exec(int i, int j){
        //default로 선언함으로 메소드를 구현할 수 있다.
        //원래는 추상 클래스만 선언 가능했으니까 구현 불가능
            return i + j;
        }
    }
```

2. Calculator 인터페이스를 구현한 MyCalculator 클래스

```
public class MyCalculator implements Calculator {

        @Override
        public int plus(int i, int j) {
            return i + j;
        }

        @Override
        public int multiple(int i, int j) {
            return i * j;
        }
    }
```

3. 사용 가능한 Class 생성

```
public class MyCalculatorExam {
        public static void main(String[] args){
            Calculator cal = new MyCalculator();
            int value = cal.exec(5, 10);    //default로 구현한 메소드도 사용 가능
            System.out.println(value);
        }
    }
```

4. 인터페이스의 static method

1. 인터페이스 생성

```
public interface Calculator {
    public int plus(int i, int j);
    public int multiple(int i, int j);
    default int exec(int i, int j){
        return i + j;
    }
    public static int exec2(int i, int j){   //static 메소드
        return i * j;
    }
}
```

2. Calculator 인터페이스 사용 가능하게 한 MyCalculatorExam

```
public class MyCalculatorExam {
    public static void main(String[] args){
        Calculator cal = new MyCalculator();
        int value = cal.exec(5, 10);
        System.out.println(value);

        int value2 = Calculator.exec2(5, 10);  //static메소드 호출
        System.out.println(value2);
    }
}
```

4. 인터페이스의 static method

- 오버라이드 필요
- 반드시 인터페이스.메소드 로 호출시킴

1. 인터페이스 생성

```
public interface Calculator {
    public int plus(int i, int j);
    public int multiple(int i, int j);
    default int exec(int i, int j){
        return i + j;
    }
    public static int exec2(int i, int j){   //static 메소드
        return i * j;
    }
}
```

2. Calculator 인터페이스 사용 가능하게 한 MyCalculatorExam

```
public class MyCalculatorExam {
        public static void main(String[] args){
            Calculator cal = new MyCalculator();
            int value = cal.exec(5, 10);
            System.out.println(value);

            int value2 = Calculator.exec2(5, 10);  //static메소드 호출
            //interface_name.method 형식으로만 가능.
            //cal.exec2 안됨!! (참조변수명.메소드)
            System.out.println(value2);
        }
    }
```

- 클래스 안에 선언된 클래스
- 중첩 클래스 & 인스턴스 클래스

---

5. 내부 클래스

`내부 클래스`

1.클래스 안에 인스턴스 변수, 즉 필드를 선언하는 위치에 선언되는 경우

```
    public class InnerExam1{
        class Cal{
            int value = 0;
            public void plus(){
                value++;
            }
        }

        public static void main(String args[]){
            InnerExam1 t = new InnerExam1();  //우선 원래 클래스 불러와야 함
            InnerExam1.Cal cal = t.new Cal(); //내부 클래스
            cal.plus();
            System.out.println(cal.value);

        }
    }
```

2.내부 클래스가 static으로 정의된 경우 (정적 중첩 클래스/static 클래스)

- new InnerExam2.Cal() 로 객체를 생성 가능

```
public class InnerExam2{
    static class Cal{
        int value = 0;
        public void plus(){
            value++;
        }
    }

    public static void main(String args[]){
        InnerExam2.Cal cal = new InnerExam2.Cal();
        // new InnerExam2.Cal()바로 생성 - Static한 필드니까
        // 정적인 필드라 위와는 다름.
        cal.plus();
        System.out.println(cal.value);

    }
}
```

3. 메소드 안에 클래스를 선언한 경우 (지역 중첩 클래스/지역 클래스)

- 인스턴스 변수로 선언되는 것이 아니라, 메소드로.

```
public class InnerExam3{
    public void exec(){
        class Cal{                //메소드 안에서 클래스 생성
            int value = 0;
            public void plus(){
                value++;
            }
        }
        Cal cal = new Cal();    //객체 생성
        cal.plus();             //객체.메서드
        System.out.println(cal.value);
    }


    public static void main(String args[]){
        InnerExam3 t = new InnerExam3();
        t.exec();
    }
}
```

6. 익명 클래스 (익명 중첩 클래스)

- 내부클래스 이기도 함
- 마지막 네 번째 내부 클래스

1. 원래라면...

`추상클래스 Action`

```
public abstract class Action{
    public abstract void exec();
}
```

`추상클래스 Action을 상속받은 클래스 MyAction` -> 추상클래스 사용하기 위해선.

```
public class MyAction extends Action{
    @Override
    public void exec(){
        System.out.println("exec");
    }
}
```

`MyAction을 사용하는 클래스 ActionExam`

```
public class ActionExam{
    public static void main(String args[]){
        Action action = new MyAction(); //자식클래스로 만들어주어야 함
        action.exec();    //액션이 가진 메소드
    }
}
```

---

2. 익명 클래스

- MyAction을 사용하지 않고 Action을 상속받는 익명 클래스를 만들어서 사용하도록 수정
- 괄호 안에는 메소드를 구현하거나 메소드를 추가할 수 있다. 이렇게 생성된 이름 없는 객체를 action이라는 참조변수가 참조하도록 하고, exec()메소드를 호출.
- 익명클래스를 만드는 이유는 Action을 상속받는 클래스를 만들 필요가 없을 경우이다.
- Action을 상속받는 클래스가 `해당 클래스에서만 사용되고 다른 클래스에서는 사용되지 않는 경우`이다.

```
public class ActionExam{
     public static void main(String args[]){
         Action action = new Action(){ // 이름없는 객체!
             public void exec(){
                 System.out.println("exec");
             }
         };
         action.exec();
     }
 }
```