탈잉 개발자와 프로그래밍 무작정 배워보기

6. 추상클래스와 인터페이스

강대규

목차

1. 추상클래스

추상클래스

클래스란 ?

추상클래스

클래스란 ?

변수와 메소드의 집합으로 이루어진 구성체

추상클래스

클래스란 ?

변수와 메소드의 집합으로 이루어진 구성체

→ 부품의 종류 (설계도)

추상클래스

그렇다면 추상클래스는 ?

추상클래스

그렇다면 추상클래스는 ?

추상메소드를 가지고 있는 클래스

추상메소드

메소드의 이름만 선언하고

메소드의 기능은 비어있는 메소드

추상메소드

메소드의 이름만 선언하고

메소드의 기능은 비어있는 메소드

→ 메소드 기능의 구현은 자식 클래스에게 맡김

추상클래스와 추상메소드

```
public abstract class Pet {

    public void bark() {
        String barkSound = getBarkSound();
        System.out.println(barkSound);
    }

    public abstract String getBarkSound();

}
```

추상클래스와 추상메소드

```
public abstract class Pet {

    public void bark() {
        String barkSound = getBarkSound();
        System.out.println(barkSound);
    }

    public abstract String getBarkSound();

}
```

추상클래스와 추상매소드를
사용하기 위한 예약어

추상클래스와 추상메소드

```
public abstract class Pet {

    public void bark() {
        String barkSound = getBarkSound();
        System.out.println(barkSound);
    }

    public abstract String getBarkSound();

}
```

추상메소드

→ 선언만 해두고 자식 메소드에게 기능을 맡김

추상클래스와 추상메소드

```
public abstract class Pet {

    public void bark() {
        String barkSound = getBarkSound();
        System.out.println(barkSound);
    }

    public abstract String getBarkSound();

}
```

추상메소드 호출은 가능함

추상메소드

→ 선언만 해두고 자식 메소드에게 기능을 맡김

추상클래스와 추상메소드

```
public class Dog extends Pet {

    @Override
    public String getBarkSound() {
        return “bow”;
    }
}
```

```
public class Cat extends Pet {

    @Override
    public String getBarkSound() {
        return “meow”;
    }
}
```

추상클래스와 추상메소드

```
public class Dog extends Pet {

    @Override
    public String getBarkSound() {
        return “bow”;
    }
}
```

```
public class Cat extends Pet {

    @Override
    public String getBarkSound() {
        return “meow”;
    }
}
```

```
public static void main(String[] args) {
    Dog dog = new Dog();
    Cat cat = new Cat();

    dog.bark();
    cat.bark();
}
```

추상클래스는 왜 사용할까?

큰 프로젝트의 경우 기능 하나하나를 세세히 하는 것 보다는

대략적인 기능을 정의하는 것이 더욱 중요하기 때문

추상클래스는 왜 사용할까?

큰 프로젝트의 경우 기능 하나하나를 세세히 하는 것 보다는

대략적인 기능을 정의하는 것이 더욱 중요하기 때문

→ 자세한 설계도를 그리기 전에 스케치

추상클래스는 왜 사용할까?

큰 프로젝트의 경우 기능 하나하나를 세세히 하는 것 보다는

대략적인 기능을 정의하는 것이 더욱 중요하기 때문

→ 자세한 설계도를 그리기 전에 스케치

즉 **아키텍쳐 설계**에 유용함

추상클래스의 단점

1. 잦은 상속으로 복잡도가 증가한다.
2. 자바는 다중 상속을 지원하지 않기 때문에 한계점이 존재한다.

2. 인터페이스

인터페이스

메소드 선언 만을 가지고 있는 것

인터페이스

메소드 선언 만을 가지고 있는 것

인터페이스를 implements 한 클래스는 그 메소드를 구현해야 한다.

인터페이스의 생성

```
public interface Flyable {
    public void fly();
}
```

인터페이스의 사용

```
public class MockingBird extends Pet implements Flyable {

    @Override
    public void getBarkSound() {
        return “짹짹”;
    }

    @Override
    public void fly() {
        System.out.println(“앵무새 날다.”);
    }
}
```

인터페이스의 사용

Can - do 라는 관계가 성립한다면 인터페이스 구현을 시키는 것이 좋다.

→ Mocking Bird can fly

→ MockingBird implements Flyable

인터페이스는 왜 사용할까?

큰 프로젝트의 경우 기능 하나하나를 세세히 하는 것 보다는

대략적인 기능을 정의하는 것이 더욱 중요하기 때문

인터페이스는 왜 사용할까?

큰 프로젝트의 경우 기능 하나하나를 세세히 하는 것 보다는

대략적인 기능을 정의하는 것이 더욱 중요하기 때문

→ 자세한 설계도를 그리기 전에 스케치

즉 **아키텍쳐 설계**에 유용함

인터페이스 vs 추상클래스

1. 인터페이스는 메소드만을 가지지만, 추상클래스는 자세한 로직을 가질 수 있다.
2. 인터페이스는 다중상속을 못하는 문제의 대안이 될 수 있다.
3. 인터페이스는 추상클래스의 복잡성을 줄일 수 있다.

인터페이스 vs 추상클래스

1. 인터페이스는 메소드만을 가지지만, 추상클래스는 자세한 로직을 가질 수 있다.
2. 인터페이스는 다중상속을 못하는 문제의 대안이 될 수 있다.
3. 인터페이스는 추상클래스의 복잡성을 줄일 수 있다.

상황에 맞춰서 쓰고싶은 걸 쓰면 됨.