1회차 : 객체지향개념2 - 6. 추상클래스

날짜 @2023년 6월 14일

- 클래스
  - 정의 : 특정 객체를 생성하기 위해 **변수**와 **메소드**를 정의하는 일종의 틀
  - 사용하는 이유 : 클래스나 변수의 중복 정의를 하지 않아도 되므로 효율적으로 코딩이 가능

```
class Animal {
    String name;
    int age;
    Animal(int name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

6. 추상 클래스

"추상적으로 말하지 말자…"
구체성(具體性)이 없어서 그 뜻이 분명하지 않은 것을 추상적이라고 한다.
이러한 애매모호한 메서드를 **추상 메서드**라고 한다.

애매모호한데 왜 써?

세부 코드를 구체적으로 작성하지 않음으로써

추상클래스를 상속받은 자식클래스는 **구현해야 될 메소드를 명시적으로 전달**받을 수 있고, 추상메소드를 **자식클래스의 특징에 맞게 세부 코드를 구현**할 수 있다.

⇒ 다른 클래스를 작성하는 데 도움을 줄 목적으로 사용한다.

추상클래스 : 추상메서드(미완성 메서드)를 포함하고 있는 클래스

- 클래스→ 설계도, 추상클래스 → **미완성 설계도**
- 완성된 설계도가 아니므로 인스턴스를 생성할 수 없다.
- 일반메서드가 추상메서드를 호출할 수 있다.

```
abstract class Animal {
    String name;
    int age;
    Animal(int name, int age) {
        this.name = name;
        this.age = age;
    }
    abstract void crying();
}
```

***추상메서드 : 선언부만 있고** 구현부(몸통, body)가 없는 메서드

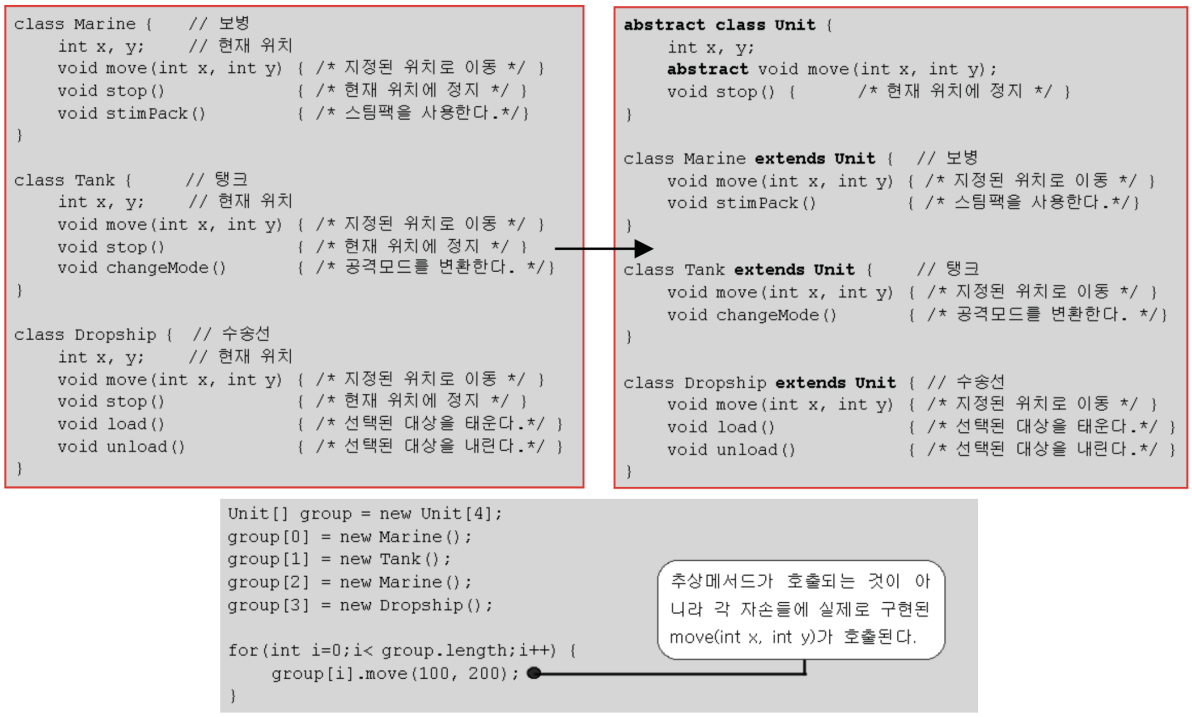
목적 : 꼭 필요하지만, 자손마다 다르게 구현될 것으로 예상되는 경우 사용

추상클래스를 상속받는 **자손클래스에서 추상메서드의 구현부를 완성해야함**

```
abstract class Player {
     int currentPos;          // 현재 Play되고 있는 위치를 저장하기 위한 변수

     Player() {               // 추상클래스도 생성자가 있어야 한다.
          currentPos = 0;
     }

     abstract void play(int pos);      // 추상메서드
     abstract void stop();             // 추상메서드

     void play() {
          play(currentPos);            // 추상메서드를 사용할 수 있다.
     }
    ...
}
```

***추상클래스 작성 요령**

: 여러 클래스에서 **공통적으로 사용되는 메서드**를 추상메서드로 뽑기



7. 인터페이스(interface)

- 일종의 추상클래스. 추상클래스(미완성 설계도)보다 추상화 정도가 높다.(밑그림 정도)
- 실제 구현된 것이 전혀 없는 기본 설계도(알맹이 없는 껍데기)
- **인스턴스를 생성할 수 없고, 클래스 작성에 도움을 줄 목적**
- **미리 정해진 규칙에 맞게 구현하도록 표준을 제시하는 데 사용**
- 추상클래스는 단일 상속. 인터페이스는 **다중 상속** 가능.
- 클래스는 추상메서드 구현의 강제성을 띄어 결합도 증가

  인터페이스는 강제성 없어 **결합도 낮음.**
- 추상메서드와 상수만을 멤버로 가질 수 있다.

인터페이스 작성법

- 클래스 작성할 때 class 대신 interface 사용. 이외 동일.
- 구성요소(멤버) : 상수와 추상메서드
  - 상수 : public static final
  - 추상메서드 : public abstract

인터페이스 상속

- 클래스와 동일하게 상속 가능. (**다중 상속** 허용)
- Object클래스와 같은 **최고 조상이 없다.**

```
interface Movable {
     /** 지정된 위치(x, y)로 이동하는 기능의 메서드 */
     void move(int x, int y);
}

interface Attackable {
     /** 지정된 대상(u)을 공격하는 기능의 메서드 */
     void attack(Unit u);
}

interface Fightable extends Movable, Attackable { }
```

인터페이스 구현

- 상속받은 후에 직접 구현
- 인터페이스에 정의된 추상메서드를 완성해야 한다.
- 상속과 구현이 동시에 가능하다.

인터페이스를 이용한 다형성

인터페이스 장점

1. 개발시간 단축
2. 표준화 가능
3. 서로 관계없는 클래스들에게 관계를 맺어 줄 수 있음
4. 독립적인 프로그래밍 가능

예제

인터페이스 이해

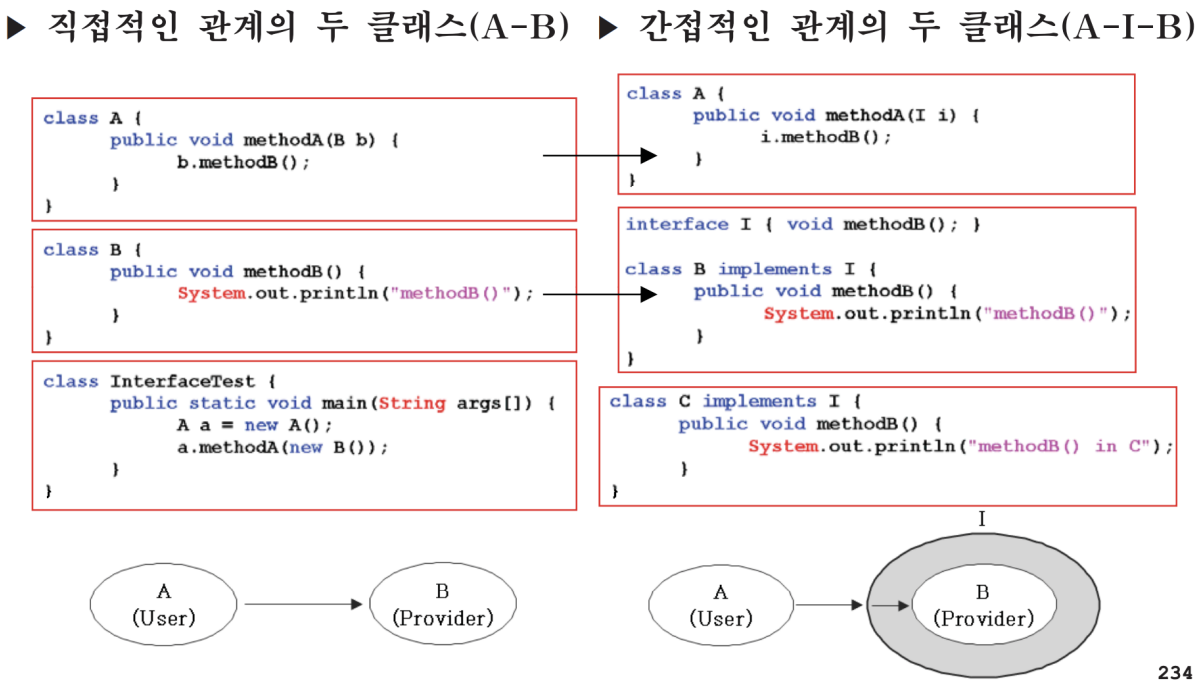
(1)

- 두 대상(객체) 간의 '연결, 대화, 소통'을 돕는 '**중간 역할**' 레츠고

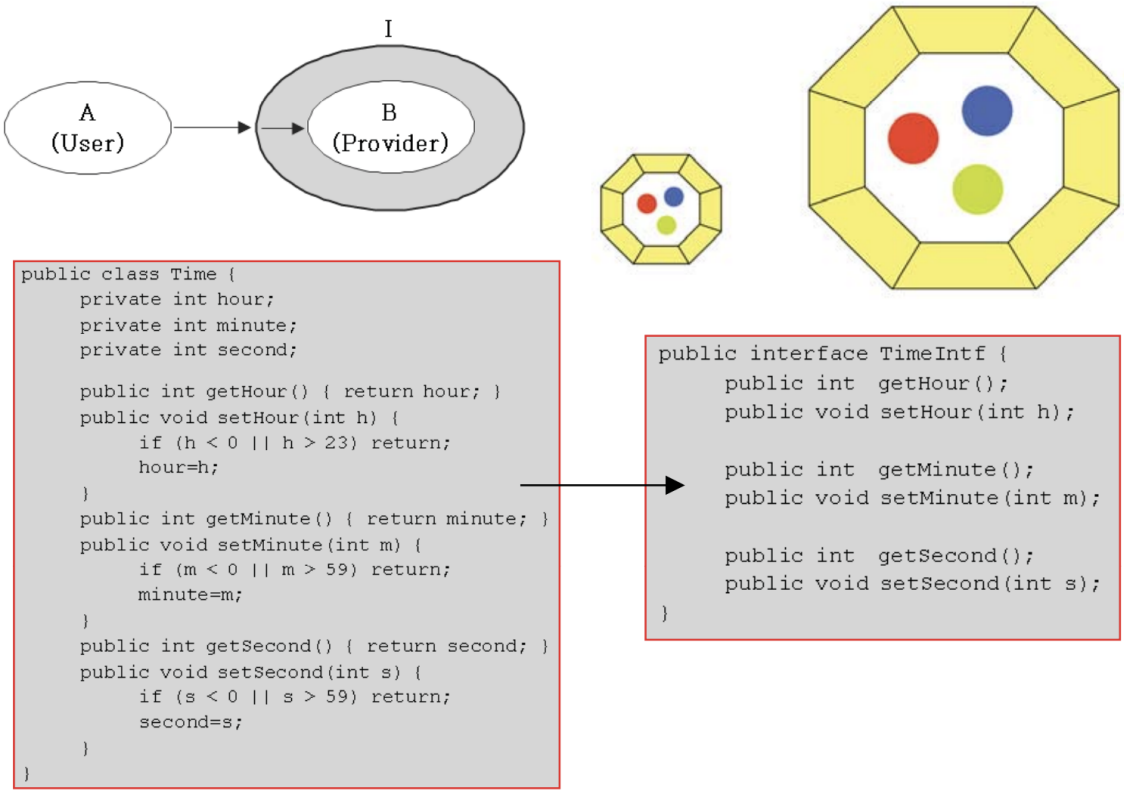
  선언과 구현 분리를 가능하게 한다.

- 인터페이스를 이해하려면 두 가지 기억하기!
  - 클래스를 사용하는 쪽(User)와 클래스를 제공하는 쪽(Provider)
  - 메서드를 사용(호출)하는 쪽(User)에서 사용하려는 메서드(Provider) 선언부만 알면 된다.

(2)



(3)



결론

추상클래스, 인터페이스 : 클래스를 작성하는데 도움을 줄 목적으로 사용

추상클래스는 일반적인 클래스인데 미완성 메서드(추상메서드)를 포함

인터페이스는 상수와 선언부만 있는 메서드만을 포함

(인터페이스는 관계없는 클래스 간 관계를 맺어주고 표준화를 시키며 개인작업이 가능하게 해줘 프로그래밍 시간을 단축시킬수 있다.)

	클래스	추상클래스	인터페이스
비유	설계도	미완성설계도(목차)	밑그림
목적	구현	목차	목차
인스턴스 생성	가능	불가능	불가능
구성	속성(변수), 기능(메소드)	일반메소드,상수,변수필드,	only 추상메소드, 상수
다중상속	불가능(extends)	불가능(extends)	가능(implements)