< 추상 클래스 / 인터페이스 >

1. 추상클래스 (public abtract class 클래스명 { })

- 미완성 클래스라고 함 → 객체 생성 못하는 클래스

  ➔ 추상클래스명 레퍼런스 ; // ok 레퍼런스 = new 추상클래스생성자(); // error

- 클래스 안에 한 개 이상의 추상 메소드를 가지고 있다면, 그 클래스는 반드시 추상클래스여야 함

- 추상클래스를 상속받은 후손 클래스는 반드시 부모의 추상 메소드를 완성시켜야 되는 강제성이 부여됨

  ➔ 후손이 추상 메소드를 반드시 오버라이딩 해야 함

  < 추상 메소드 >

  public abstract 반환형 메소드명 ([자료형 매개변수]) ;

  - 메소드의 헤드(head)만 있고, 몸체(body : 구현부)가 없는 메소드
  - 표준화된 인터페이스를 제공할 목적으로 추상 메소드 사용됨
  - 메소드 사용의 통일성을 확보할 목적
  - **어떤 기능 처리용 메소드는 어떤 이름으로 어떻게 사용할 것인지에 대한 약속(규칙)을 미리 정해놓은 것**
  - **정해놓은 규칙을 상속을 통해 각자 정해진 규칙은 따르되, 처리할 내용은 각자의 처리 내용에 맞게 작성하도록 함**

**** 주의 사항 ****

추상 메소드가 없어도 추상클래스로 만들 수 있음 → 단지 객체 생성 못하게 할 목적

2. 인터페이스

- 상수형 필드와 추상 메소드로만 구성된 추상 클래스의 변형체
- 클래스에 추상 메소드를 포함시키는 구조보다는 **메소드의 통일성을 부여하기 위해 추상 메소드만 따로 모아놓은 추상 클래스**

[public] interface 인터페이스명 {

```
// 상수형 필드 : 필드명은 대문자로 지어줌
[public static final] 자료형 필드명 = 초기값;  // 명시적으로 해줌

// 추상 메소드
[public abstract] 반환형 메소드명([자료형 매개변수]) ;
```

}

➔ 인터페이스의 사용

- 상속을 통해서 후손이 부모 인터페이스의 추상메소드를 완성시키는 방식
- 클래스가 인터페이스를 상속받을 때 : implements 인터페이스명 (다중상속 가능)

```
[public] class 클래스명 extends 클래스명 implements 인터페이스명 {  }
[public] class 클래스명 implements 인터페이스명, 인터페이스명, .. {  }
```

- 인터페이스간에도 상속이 가능, 다중 상속 가능

```
[public] interface 인터페이스명 extends 인터페이스명, 인터페이스명, .. {  }
```

< 정리 >

상속 → 추상클래스 → 인터페이스

오른쪽으로 갈수록 강제성이 커진다.