# final 제어자와 abstract 제어자



## final 제어자 - 마지막의, 변경될 수 없는

> **필드, 지역 변수, 메서드, 클래스 앞**에 가능하며 어디에 위치하느냐에 따라 의미가 달라진다.

### 1. final 변수

- 대입된 값이 **최종(final) 값**이 된다.
    - 즉, **한 번 대입된 값은 수정할 수 없다**.
- **선언과 동시에 값을 대입**할 수도 있고, **선언과 대입을 분리**할 수도 있다.
    - 만약 분리하는 경우라면 **반드시 생성자에서 초기화를 진행**해야 한다.

> 💡 *final 필드는 일반 필드와 같이 강제 초기화되지 않기 때문*

- **final 필드의 실제 데이터**는 첫번째 메모리 영역 중 **상수(final) 영역에 자리**한다.
    - final 역시 필드이고, **필드는 멤버**에 **속하기** 때문에 **객체를 생성하면 객체 속에 포함**이 되지만, **실제 데이터는 첫번째 영역**에 있고 **그 데이터에 대한 위치값을 공유**한다.
- 지역 변수에도 final을 붙일 수 있는데, 그런 경우 **상수 영역에 값 1개가 복사**된다.
    - 값의 복사는 **값을 선언한 후 최초로 값이 초기화될 때 딱 한 번만** 일어난다.
    - 원본의 복사는 값을 대입할 때 한 번만 일어난다는 것이다.
    - 스택 메모리의 변숫값은 자신이 만들어진 메서드가 종료되면 사라지지만, 복사한 final 변수는 첫번째 메모리 영역에 있기 때문에 사라지지 않는다.

즉, **어떤 필요에 따라 복사본**을 하나 만들어 놓음으로써 **원본이 삭제된 이후에도 그 값을 활용**할 수 있도록 하는 것이 **final 변수의 기능**이다.

### 2. fianl 메서드와 final 클래스

- 각각 **최종 메서드, 최종 클래스**의 의미를 지닌다.
    - 상속 시 메서드를 오버라이딩하면 메서드 기능이 변경되는데 `final 메서드`의 경우 기능을 변경할 수 없다. **즉, 오버라이딩이 불가능하다는 것.**
        - 변경될 수 없는, 오버라이딩(재정의)이 불가능한 메서드

```
class A{
    void abc() { }
    final void bcd() { }
}

class B extends A {
    void abc() { }
    void bcd() { }
}
```

    - `final 클래스`의 경우 **더 이상 자식이 없다** 즉, **상속 자체가 불가능한 클래스**라는 것.
        - 변경될 수 없는, 확장될 수 없는 클래스

```
final class A { }

class B extends A {} // 불가능
```

표 11-1 final 변수, 메서드, 클래스 정리

| final 변수 | final 메서드 | final 클래스 |
|---|---|---|
| class A {<br>  int m = 3;<br>  final int n = 4;<br>}<br><br>A a = new A();<br>a.m = 5;<br>a.n = 9;  // (X) | class A {<br>  void abc() {...}<br>  final void bcd() {...}<br>}<br><br>class B extends A {<br>  void abc() {...}<br>  void bcd() {...} // (X)<br>} | final class A {<br>  int m;<br>  void bcd() {...}<br>}<br><br>class B extends A { // (X)<br>  ...<br>} |
| 값 변경 불가능 | 오버라이딩 불가능 | 상속 불가능 |

# abstract - 추상의, 미완성의 (추상화)

➡️ 추상화란?

불필요한 정보를 숨기고 중요한 정보만을 나타내는 것을 의미

- 어떤 영역에서 필요한 공통의 속성이나 행동을 추출하여 사용

자동차 속도 줄이기 기능을 사용한다고 브레이크를 밟으면, 브레이크 페달을 밟았을 때 속도가 어떻게 감소하고 멈추게 되는지 운전자는 내부까지 알고 있지 않다. 이처럼 자바 프로그래밍에서 사용자에게 필요하지 않은 세부적인 기능 구현 정보를 숨기는 것이 추상화이다.

[장점]

- 객체 간의 복잡성이 줄어든다
- 코드의 중복을 막고 재사용을 높인다
- 사용자에게 중요한 세부 정보만 제공 → 보안에 도움

> **메서드, 클래스** 앞에만 붙일 수 있으며 추상적이라는 의미가 붙어 미완성이라는 뜻이 된다.

- 추상 메서드 구조

```
abstract 리턴_타입 메서드명();
```

추상 메서드는 중괄호가 없는 메서드로, 중괄호 안의 구현부가 없는 형태를 띤다.

아직 무슨 기능을 정의할지 정해지지 않은 **미완성 메서드**인 것이다.

- 추상 클래스

```
abstract class 클래스명 {
    // 추상 메서드 포함
}
```

  - 아무 기능도 하지 않는 추상 메서드를 1개 이상 포함하고 있다면 무조건 클래스는 추상 클래스라고 정의해야 한다.
    - 포함되어 있는 메서드에 미완성이 있기 때문이다.
  - 추상 클래스는 미완성 클래스이기 때문에 객체를 생성해서 사용할 수 없다.

        - 설계도가 완성되지 않았으니 제품을 만들지 못하는 것과 같은 의미이다.
    - 추상 클래스를 사용하고 싶다면 상속을 통해 자식 클래스를 하나 만들어주고, 자식 클래스에서 메서드를 재정의(즉, 해당 기능을 완성)하여 사용해야 한다.
- 추상 클래스를 상속 받은 자식 클래스
    - 상속받은 클래스 내부에 추상 메서드가 1개라도 있다면, 해당 클래스는 추상 메서드를 일반 메서드로 오버라이딩하거나 자신을 추상 클래스로 정의하여 또 상속해 주어야 한다.
        - 만약 둘 중 어떤 정의도 하지 않는다면 오류가 발생한다.

    즉, 추상 클래스가 있을 때 '이를 상속한 모든 자식 클래스 내부에는 항상 추상 메서드가 정의돼 있다.'는 것이 보장된다.