# 01.09

## 📌상속

### 상속이란 무엇일까요?

기존의 클래스의 메소드와 필드를 물려 받아 새로운 기능을 추가하거나 재정의하여 새로운 클래스를 정의하는 것을 의미합니다.

상속을 통해 다형성 및 계층 구조가 발생할 수 있습니다. 또한 중복 코드 제거 및 느슨한 결합이 가능합니다.

### 다형성이란 무엇일까요?

하나의 객체나 메소드가 여러 다른 형태를 가질 수 있음을 의미합니다.

따라서 하나의 타입에 여러 객체를 대입할 수 있고, 기능을 확장하거나 객체를 변경해야 할 때 타입 변경 없이 객체 주입만으로 수정이 일어나게 할 수 있습니다.

### 다형성 구현 방식

- 오버 로딩 - 여러 타입을 받아들여 같은 기능을 하도록 함 → 메소드 동적 호출 가능(다형성)
- 오버 라이딩 - 상위 클래스의 메서드를 하위 클래스에서 재정의하는 것. 기능의 확장과 객체의 수정에 유연한 구조. 부모 클래스가 하위 클래스 메서드를 동적 바인딩할 수 있음

# 📌Object 클래스

## Object 클래스에 대해 설명해주세요

자바의 모든 클래스의 조상 클래스이자, 클래스의 기본 행동을 정의한 클래스입니다. 객체 처리, 쓰레드 처리와 관련된 메소드를 담고 있습니다.

## 주요 메서드로는 무엇이 있을까요?

먼저 객체 처리에 대한 3가지를 말할 수 있습니다.

먼저 **toString()**이 있습니다. sout이나 객체에 대한 더하기  연산 시 해당 객체에 대한 정보를 반환하며, 기본 설정으로는 getClass().getName() + '@' + Integer.toHexString(hashcode()) 를 반환합니다.

**equals()**가 있습니다. 참조 자료형의 비교에 동등 연산자를 사용하면, 객체의 메모리 주소, 즉 hashcode를 비교하게 됩니다. 이 때 객체가 가진 값을 비교하게 하려면 equals()를 오버라이딩 해야 합니다.

**hashcode()**가 있습니다. 객체의 주소를 int 타입의 16진수로 리턴합니다. 오버라이딩 시 한 애플리케이션에서 메서드를 수행했을 때 동일한 int 값을 리턴하도록 해야 합니다. 자바 실행마다 동일하지는 않아도 됩니다. equals()가 true를 반환 시 hashcode 역시 동일한 int를 반환해야 하며, false 반환 시 무조건 다를 필요는 없습니다. 그러나 다를 경우 hashtable의 성능 향상에 도움이 됩니다. (java 8), java 13 부터는 실용적인 이유로 다른 int를 반환해야 합니다. 따라서 equals 재정의 시 hashcode로 재정의 해야 합니다.

# 📌Object.clone

요약 | `Object.clone`은 native 메소드로, bitwise copy를 합니다.

*bitwise copy: 새로운 객체용으로 충분한 메모리를 확보하기 위해 객체가 얼마나 큰 지를 알아내고 이전 객체에서 새로운 객체로 모든 비트를 복사합니다.

## clone()을 사용하려면

`Object` 클래스에 있는 `clone` 메소드를 사용하려면 다음과 같이 합니다.

1. 클래스가 `Cloneable` 인터페이스를 구현하게 한다.

   X → `CloneNotSupportedException`

2. `clone` 메소드를 public으로 오버라이드 합니다. (Object 에서는 protected)
3. 오버라이드한 `clone` 메소드에서는 `super.clone`을 호출합니다.

## 왜 `super.clone()`을 호출하는가?

- `clone`은 생성자가 아닙니다. 따라서 자동으로 호출되지 않습니다.
- `super.clone()`을 호출해서 재귀적으로 부모 클래스의 `clone` 메소드를 호출하는 방식.

## `clone()`은 기본적으로 얕은 복사를 수행합니다.

- 깊은 복사가 필요하다면 `clone`을 오버라이드할 때 코딩해서 구현할 것.
- ex) ArrayList
  - ArrayList는 각 요소에 대해 원형 타입 → 값 복사, 참조 타입 → 메모리 주소 복사
  - 깊은 복사를 위해서는 각 요소를 하나하나 clone 하여 전달하는 함수를 오버라이드 해야 합니다.
  - ArrayList가 포함한 객체들에 대해 `clone()`을 시도하지 않는 것은 그 객체들이 복제 가능(cloneable)한지 보장할 수 없기 때문입니다.

## `Cloneable` 인터페이스는 메서드가 없다.

- JVM에 이 객체를 복제할 수 있다고 알리는 용도 이다. 이처럼 메서드 없이 인터페이스를 통해 의미를 전달하는 것을 마커 인터페이스 패턴(marker interface pattern)이라고 합니다.

## 메소드가 리턴하는 객체 `this` 객체(복제대상 객체)와는 독립적이어야 합니다.

이러한 독립성을 달성하기 위해서 `super.clone`에 의해 리턴된 객체의 필드를 하나 이상 수정해야 할 수도 있습니다.

이런 독립성을 위한 수정 작업이 무슨 뜻이냐면, 복제되는 객체 "내부의 구조체(deep structure)"를 구성하는 변경 가능한(mutable) 객체들을 복사하고, 이 객체들에 대한 참조를 복사본에 대한 참조로 바꾸는 것을 의미합니다. 만약 클래스가 primitive 필드들만 갖고 있거

나, 불변(immutable) 객체들 대한 레퍼런스만 갖고 있다면, 보통은 `super.clone`이 리턴한 객체의 필드를 수정하지 않아도 됩니다.

### clone()의 복잡성 이유

웹에서 자바가 사용되면서, 보안 이슈로 인해 간단한 설계에서 많은 패치가 더해지면서 구현이 복잡하게 되었습니다.

참고

https://twinw.tistory.com/24

https://johngrib.github.io/wiki/java/object-clone/

## 📌final이란?

불변한 값을 가진 변수 혹은 클래스, 메서드에 붙이는 예약어입니다. 변수 선언과 동시에 값을 할당해야 합니다.

- 생성자나 메소드에서 초기화 시 → 중복으로 선언될 수 있어 기본 의도를 벗어남
- 매개 변수나 지역 변수를 final로 선언할 경우, 반드시 선언할 필요 없음
    - 매개 변수는 이미 초기화된 값이 들어오기 때문이고, 지역 변수는 외부에서 변경되지 않기 때문

## 📌인터페이스

### 인터페이스란?

**인터페이스는** 클래스들이 구현 해야하는 **동작**을 지정하는 용도로 사용되는 **추상 자료형**입니다. 구현 객체가 **같은 동작을 한다는 것을 보장**하기 위해 사용하는 것에 초점을 두고 있습니다. OOP 적으로보면 결합도(Coupling)을 낮춰 유지보수성을 늘리는 디자인 패턴이라고 볼 수도 있습니다.

### 메서드와 필드

인터페이스는 자바 8 이전에는 abstract 메서드만을 가졌고, 자바 8에서는 default, static method, 자바 9에서는 private 메서드가 추가되었습니다. 필드는 기본적으로 static이기 때문에 구현체를 따라가지 않습니다. (독립 상수)

### 장점

- 클래스의 **다중 구현** 및 인터페이스 간 **다중 상속**이 가능합니다.
- ocp를 지키며 클래스 간 코드 결합도가 낮은 프로그래밍이 가능해집니다.
- 마커 인터페이스, 함수형 인터페이스로 이용 가능합니다.

## 인터페이스를 사용하는 경우

- 어플리케이션의 기능을 정의해야 하지만 그 구현 방식이나 대상에 대해 추상화 할 때
- **서로 관련성이 없는 클래스들을 묶어 주고** 싶을때 (형제 관계)
- **다중 상속(구현)**을 통한 추상화 설계를 해야할때
- 특정 데이터 타입의 행동을 명시하고 싶은데, 어디서 그 행동이 구현되는지는 신경쓰지 않는 경우
- 클래스와 별도로 **구현 객체가 같은 동작을 한다는 것을 보장**하기 위해 사용