07 상속

자바에서 상속은 객체 지향 프로그래밍(OOP)의 중요한 개념 중 하나로, 클래스 간의 계층 구조를 형성하는 메커니즘입니다. 자바에서 상속을 정의하기 위해 extends 키워드를 사용합니다. extends 키워드를 사용하여 서브 클래스가 상속할 슈퍼 클래스를 지정합니다.

상속의 핵심 개념

- 슈퍼 클래스 (Superclass) 또는 부모 클래스 (Parent Class): 상속의 기초가 되는 클래스로, 다른 클래스에게 특성을 제공하는 클래스를 가리킵니다. 슈퍼 클래스에서 정의된 필드(멤버 변수)와 메서드는 하위 클래스에 상속됩니다.
- 서브 클래스 (Subclass) 또는 자식 클래스 (Child Class): 슈퍼 클래스로부터 상속받은 특성을 확장하거나 수정하는 클래스를 가리킵니다. 서브 클래스는 슈퍼 클래스의 특성을 확장하고 새로운 특성을 추가할 수 있습니다.
- extends 키워드: 자바에서 상속을 정의하기 위해 extends 키워드를 사용합니다. extends 키워드를 사용하여 서브 클래스가 상속할 슈퍼 클래스를 지정합니다. 예를 들어, class Subclass extends Superclass와 같이 사용합니다.
- super 키워드: super 키워드는 슈퍼 클래스의 멤버(필드 또는 메서드)에 접근하는 데 사용됩니다. 서브 클래스에서 슈퍼 클래스의 멤버를 참조할 때 super 키워드를 사용할 수 있습니다.
- 메서드 오버라이딩 (Method Overriding): 서브 클래스에서 슈퍼 클래스의 메서드를 다시 정의하고 구현을 재정의할 수 있습니다. 이를 통해 서브 클래스는 특정 메서드를 슈퍼 클래스의 버전 대신에 자체 버전으로 사용할 수 있습니다.

상속의 특징

- 여러 개의 부모 클래슬르 상속할 수 없다.
- 부모의 메서드와 필드를 사용할 수 있다.
- 부모 클래스에 생성자가 없거나 매개변수를 갖는 경우 자식 클래스에서 super() 를 호출시켜야 한다.
- 부모 클래스를 상속받은 자식 클래스는 객체를 생성할 때 부모 객체가 먼저 생성된다.

```
class Animal {
    void eat() {
        System.out.println("동물이 먹이를 먹습니다.");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("개가 짖습니다.");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog();
```

```
        myDog.eat();  // 슈퍼 클래스의 메서드 호출
        myDog.bark(); // 서브 클래스의 메서드 호출
    }
}
```

메소드 오버라이딩

자바에서 메서드 오버라이딩(Method Overriding)은 상속 관계에 있는 클래스에서 슈퍼 클래스의 메서드를 서브 클래스에서 다시 정의하고 구현하는 것을 의미합니다. 부모 메서드는 숨겨지고, 자식 메서드가 우선적으로 사용됩니다. 메서드 오버라이딩은 다형성의 핵심 원리 중 하나로, 같은 메서드 이름을 사용하여 다양한 동작을 제공하는데 사용됩니다.

메소드 오버라이딩의 특징

- 부모 메소드의 선언부(리턴 타입, 메소드 이름, 매개변수)와 동일해야 한다.
- 접근 제한을 더 강하게 오버라이딩 할 수 없다.(public -> private으로 변경 불가)
- 새로운 예외를 throws 할 수 없다.

상속의 핵심 개념

- 메서드 시그니처 일치: 서브 클래스에서 오버라이딩된 메서드는 메서드 시그니처(메서드 이름, 매개변수 목록, 반환 타입)가 슈퍼 클래스의 메서드와 일치해야 합니다. 매개변수 목록과 반환 타입은 정확하게 일치해야 합니다.

- @Override 어노테이션 (Annotation): 자바에서는 메서드를 오버라이딩할 때 @Override 어노테이션을 사용하는 것이 좋습니다. 이 어노테이션을 사용하면 컴파일러가 오버라이딩의 유효성을 검사할 수 있습니다.

- 접근 제어자 (Access Modifier): 오버라이딩된 메서드는 슈퍼 클래스의 메서드와 동일하거나 더 큰 접근 범위(더 넓은 접근 제어자)를 가질 수 있으나, 더 작은 접근 범위로 변경할 수 없습니다. 예를 들어, 슈퍼 클래스의 메서드가 protected로 선언되었다면, 서브 클래스에서 public으로 변경할 수 있지만, private로 변경할 수 없습니다.

- 예외 처리 (Exception Handling): 오버라이딩된 메서드에서 던지는 예외(checked 예외)는 슈퍼 클래스의 메서드에서 던지는 예외와 일치하거나 그 예외의 서브 클래스를 던질 수 있습니다. 하지만 오버라이딩된 메서드에서 던지는 예외는 슈퍼 클래스의 메서드에서 던지는 예외와 다르거나 더 넓은 예외를 던질 수 없습니다.

```
class Animal {
    void makeSound() {
        System.out.println("동물이 소리를 내고 있습니다.");
    }
}

class Dog extends Animal {
```

```
    @Override // 오버라이딩이 되었는지 체크한다.
    void makeSound() {
        System.out.println("개가 짖고 있습니다.");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.makeSound(); // 서브 클래스의 메서드가 호출됨
    }
}
```

부모 메소드 호출

부모 클래스의 메서드를 오버라이딩할 때 메서드의 일부만 수정하고 나머지 부분은 부모 클래스의 동작을 그대로 유지하려면 다음과 같이 처리할 수 있습니다:

super 키워드 사용: 자식 클래스에서 오버라이딩된 메서드 내에서 super 키워드를 사용하여 부모 클래스의 메서드를 명시적으로 호출할 수 있습니다. 이를 통해 부모 클래스의 메서드 동작 중 일부를 유지하고 추가적인 동작을 수행할 수 있습니다.

```
class Animal {
    void makeSound() {
        System.out.println("동물이 소리를 내고 있습니다.");
    }
}

class Dog extends Animal {
    @Override
    void makeSound() {
        // 슈퍼 클래스의 메서드를 호출하여 일부 동작을 유지
        super.makeSound();

        // 자식 클래스에서 추가적인 동작을 수행
        System.out.println("개가 짖고 있습니다.");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.makeSound(); // 서브 클래스의 메서드가 호출되고, 부모 클래스의 메서드 일부도 실행됨
    }
}
```

final 클래스와 final 메소드

자바에서 final 키워드는 클래스, 메서드 또는 변수에 적용될 수 있으며, 사용되는 맥락에 따라 다른 의미를 가집니다. final 클래스는 클래스 레벨에서 사용되며, 클래스를 final로 선언하면 해당 클래스는 더 이상 상속할 수 없는 클래스가 됩니다.

```
final class FinalClass {
    // 클래스 내용
}

// 아래와 같이 FinalClass를 상속하려고 시도하면 컴파일 오류가 발생
// class Subclass extends FinalClass {
//     // 서브 클래스 내용
// }
```

final 클래스를 사용해야 하는 경우:

클래스가 더 이상 확장되어서는 안 되는 경우. 클래스의 구현을 변경하지 않도록 보장해야 하는 경우 (불변 클래스 또는 보안 관련 클래스). 상속과 다형성을 사용하지 않아야 하는 경우 (일부 설계에는 상속을 사용하지 않고 final 클래스를 선호하는 경우도 있음)

protected 접근 제한자

protected 접근 제어자는 해당 멤버(필드 또는 메서드)를 선언한 클래스와 동일한 패키지 내에서 그리고 해당 멤버를 상속한 서브 클래스에서 접근할 수 있는 권한을 부여합니다. 이것은 멤버를 패키지 외부에서 직접 접근하지 못하게 하는 private과 달리 상속 관계의 클래스에서도 접근이 가능합니다.

protected 를 사용하는 이유

- 상속 관계에서 재사용: protected를 사용하면 슈퍼 클래스의 멤버를 서브 클래스에서 재사용할 수 있습니다. 이를 통해 코드의 재사용성이 향상되고 유지 관리가 쉬워집니다.
- 캡슐화 유지: 멤버를 protected로 선언하면 클래스의 내부 구현을 외부로부터 숨길 수 있으며, 필요한 경우 서브 클래스에서만 접근할 수 있도록 허용합니다. 이는 캡슐화를 유지하면서 상속을 통해 확장할 수 있게 합니다.
- 부모 클래스의 상태 변경: 서브 클래스에서 protected로 선언된 필드에 접근하여 부모 클래스의 상태를 변경할 수 있으므로, 상속 관계에서 부모 클래스의 동작을 커스터마이징할 수 있습니다.

```
class Animal {
    protected String name;

    public Animal(String name) {
        this.name = name;
    }

    protected void eat() {
        System.out.println(name + "이(가) 먹이를 먹습니다.");
    }
}

class Dog extends Animal {
    public Dog(String name) {
        super(name);
    }

    public void bark() {
        System.out.println(name + "이(가) 짖습니다.");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog("멍멍이");
        myDog.eat(); // 서브 클래스에서 상속받은 protected 메서드 호출
        myDog.bark();
    }
}
```

위 예제에서 Animal 클래스의 name 필드와 eat 메서드는 protected로 선언되었습니다. 이것은 Dog 클래스에서 Animal 클래스의 name 필드를 사용하고 eat 메서드를 호출할 수 있도록 합니다. 이렇게 protected를 사용하여 상속 관계에서 멤버를 공유하고 재사용할 수 있습니다.