상속 [Java]

소유자	종수 김
태그	

[상속]

전기차, 가스차 모두 자동차의 하위 개념이다.
그렇다면, 자동차가 가지는 고유 기능 - 이동, 정지, 주차 등 공통된 부분을 각각 다른 객체로 정의할 필요가 없다.

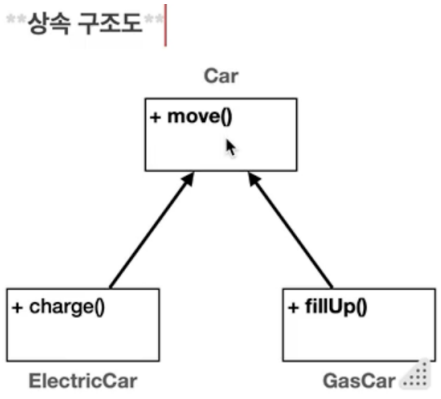
이를 위한 공통 기능의 묶음.

[상속 관계]

객체 지향 프로그래밍의 핵심 요소
기존 클래스의 필드와 메서드를 새로운 클래스에서 재사용하게 해줌.

이름 그대로 속성과 기능을 물려받는 것.

- 부모 클래스 (슈퍼 클래스) : 상속을 통해 자신의 피륻와 메서드를 다른 클래스에 제공하는 클래스
- 자식 클래스 (서브 클래스) : 부모 클래스로부터 필드와 메서드를 상속 받는 클래스
- extends 키워드를 사용. ⇒ public class ElectricCar extends Car



전기차와 가솔린차가 `Car`를 상속 받은 덕분에 `electricCar.move()`, `gasCar.move()`를 사용할 수 있다.

- 자식은 부모에 대해 extends 키워드를 통해 알고 있지만, 부모는 자식에 대해 알지 못함.
  때문에, 자식은 부모의 기능을 사용할 수 있지만, 부모는 자식의 기능을 사용하지 못함.

[단일 상속]

자바는 다중 상속을 지원하지 않으므로, extends의 대상은 하나만 선택이 가능하다.
하지만, 부모가 또 다른 부모를 가지는 것은 상관 없다.

다중 상속 시 자식 입장에서 기능을 사용할 때 어떤 부모의 기능을 사용해야하는 지 알 수 없으며, 다중 상속 사용시 클래스 구조가 매우 복잡해 질 수 있으므로 다중 상속을 허용하지 않음.

[상속과 메모리 구조]

상속을 받는 서브 클래스 객체를 생성 시 상속 관계에 있는 부모 클래스를 포함해서 인스턴스를 생성.
참조 값은 하나지만, 실제로 그 안에는 Car, ElectricCar 두가지 클래스가 공존.

상속이라고 해서 단순하게 부모의 필드와 메서드만 물려받는 것이 아닌, 상속 관계를 사용하면 부모 클래스도 함께 포함해서 생성.

- 외부에서 볼 때는 하나의 인스턴스를 생성하지만, 내부에서는 부모와 자식이 생성되고 공간도 구분.

[이 때, electricCar.charge()를 호출하면 ?]

하나의 참조 값에 접근을 했을 때, 내부에 부모와 자식 모두가 존재.
이 때 부모에서 함수를 찾을 지, 자식에서 찾을 지 결정 해야함.

이 때 '호출하는 변수의 타입(클래스)를 기준으로 선택' Ex) Car car = new ElectricCar();
즉, electricCar의 변수 타입이 ElectricCar이므로 ElectricCar의 함수를 찾아서 호출.
만약, 호출하는 변수의 타입에 함수가 존재하지 않는다면, 부모를 찾아 올라가며 해당 함수를 호출.

- 상속 관계의 객체를 생성하면 그 내부에는 부모와 자식이 모두 생성된다.
- 상속 관계의 객체를 호출할 때, 대상 타입을 정해야 한다. 이 때, 호출자의 타입을 통해 대상 타입을 찾는다.
- 현 타입에서 기능을 찾지 못하면 상위 부모 타입으로 기능을 찾아서 실행한다. 기능을 찾지 못하면 컴파일 오류

[상속과 기능 추가]

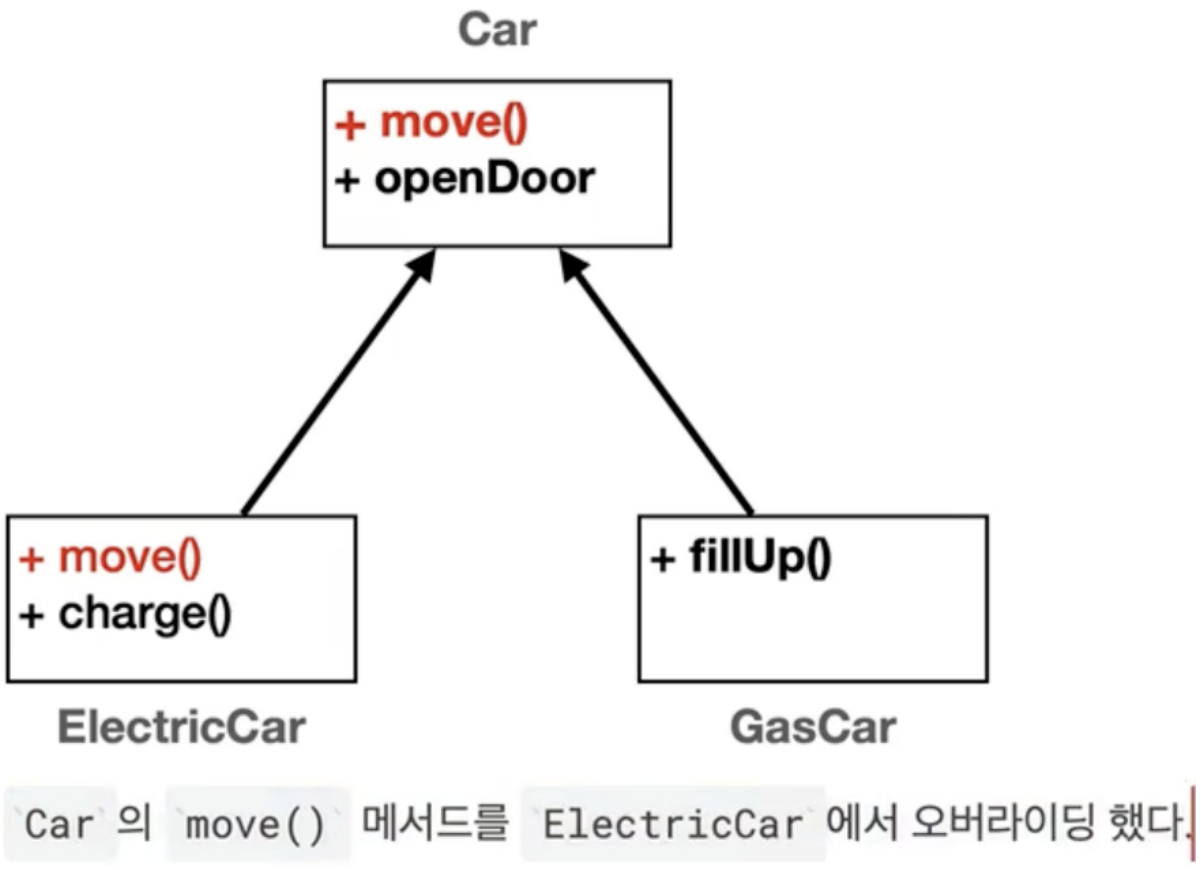
모든 자식 클래스에 기능을 추가하는 것은, 부모에만 추가해주면 됨.
공통적인 기능을 부모에만 추가함으로써,
중복을 제거할 수 있으며, 자식을 추가해감으로써 확장을 용이하게 하는 것이 가능하다.

= 어떤 상위의 개념을 가지고있는지 명시가 가능하며, 상위의 개념의 기능을 간단하게 가져갈 수 있음.

[상속과 메서드 오버라이딩]

부모 타입의 기능을 자식에서는 다르게 재정의 할 수 있음.
이렇게, 부모에게서 상속 받은 기능을 자식이 재정의 하는 것을 메서드 오버라이딩(Method Overriding)이라고 함.

- @Override
  주석과 비슷하지만, 프로그램이 읽을 수 있는 특별한 주석
  - 없어도 동작하지만, 휴먼 에러를 컴파일 단계에서 잡아줄 수 있어서 명시하는 것이 좋음.



1. electricCar.move() 호출
2. 호출한 electricCar의 타입은 ElectricCar이므로, 인스턴스 내부의 ElectricCar타입에서 시작.
3. ElectricCar 타입에 move()가 있으므로, 해당 메서드를 실행. 실행할 메서드를 찾았으므로 부모 타입을 찾지 않음.

- [오버로딩과 오버라이딩]
  - 오버로딩 : 메서드 이름이 같고, 매개변수(파라미터)가 다른 메서드를 여러개 정의하는 것.
  - 오버라이딩 : 하위 클래스에서 상위클래스의 메서드를 재정의 하는 과정.
    상속 관계에서 사용, 부모의 기능을 자식이 다시 정의하는 것.
- [메서드 오버라이딩 조건]
  - 메서드 이름이 같아야 한다.
  - 접근 제어자가 상위 클래스의 메서드보다 더 제한적이어서는 안된다.
    부모 public, 자식 오버라이딩 protected 불가능.
    부모 protected, 자식 오버라이딩 pulbic 가능.
  - 파라미터 타입, 순서, 개수가 같아야 한다.
  - 오버라이딩 메서드는 상위 클래스의 메서드보다 더 많은 체크 예외를 throw로 선언할 수 없다.
    더 적거나, 같은 수의 예외, 또는 하위 타입의 예외는 선언 가능.
  - static, final, private 키워드가 붙은 메서드는 오버라이딩 될 수 없다
    - static은 클래스 레벨에서 동작하므로 인스턴스 레벨에서 사용하는 오버라이딩이 의미가 없음.
    - final은 재정의를 금지한다
    - private은 해당 클래스에서만 접근 가능하기에 하위 클래스에서 보이지 않는다. 따라서 오버라이딩 할 수 없다.
  - 생성자는 오버라이딩 할 수 없다.

[상속과 접근제어]

protected : 같은 패키지안에서 호출은 허용함, 패키지가 달라도 상속 관계의 호출은 허용한다.

Ex.

▼ Parent

```
package extend.access.parent;

public class Parent {
    public int publicValue;
    protected int protectedValue;
```

```
    int defaultValue;
    private int privateValue;

    public void publicMethod() {
        System.out.println("Parent.public");
    }

    private void privateMethod() {
        System.out.println("Parent.private");
    }

    void defaultMethod() {
        System.out.println("Parent.default");
    }

    protected void protectedMethod() {
        System.out.println("Parent.protected");
    }

    public void printParent() {
        System.out.println("==Parent 메서드==");
        System.out.println("publicValue = " + publicValue)
        System.out.println("protectedValue = " + protected
        System.out.println("defaultValue = " + defaultValu
        System.out.println("privateValue = " + privateValu

        publicMethod();
        defaultMethod();
        privateMethod();
        protectedMethod();
    }

}
```

▼ Child

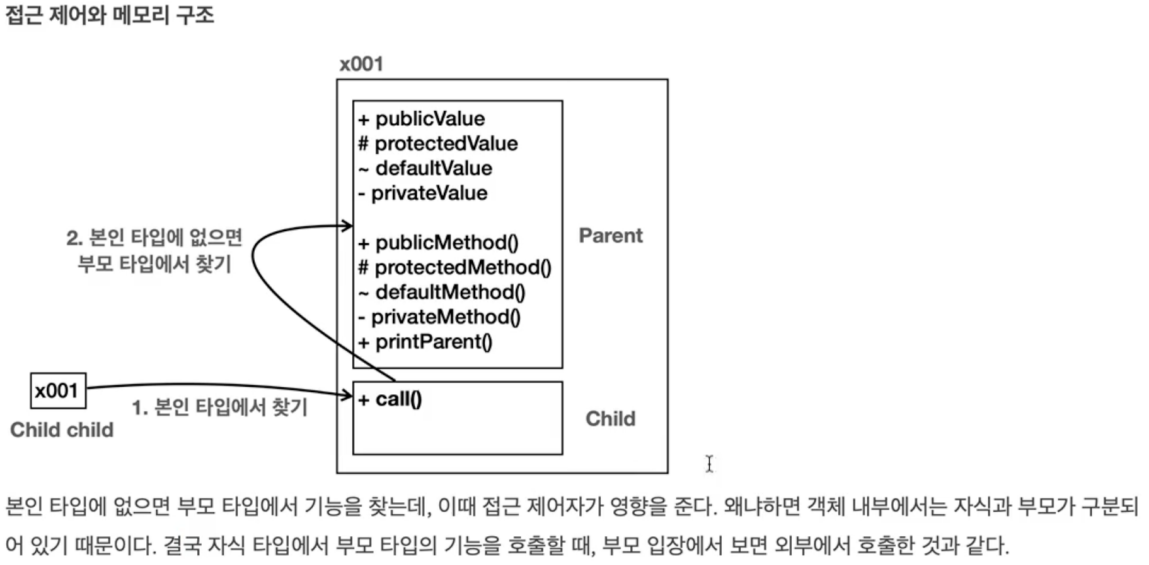
```
package extend.access.child;
```

```
import extend.access.parent.Parent;

public class Child extends Parent {
    public void call() {
        publicValue = 1;
        protectedValue = 2; // 상속 관계 or 같은 패키지, 현재는
//        defaultValue = 3; // 다른 패키지이므로 불가능 Compile
//        privateValue = 4; // 접근 불가 Compile Error

        publicMethod();
        protectedMethod();
//        defaultMethod();
//        privateMethod();
    }
}
```

[접근 제어와 메모리 구조]



본인 타입에 없으면 부모 타입에서 기능을 찾는데, 이때 접근 제어자가 영향을 준다. 왜냐하면 객체 내부에서는 자식과 부모가 구분되어 있기 때문이다. 결국 자식 타입에서 부모 타입의 기능을 호출할 때, 부모 입장에서 보면 외부에서 호출한 것과 같다.

[super - 부모참조]

부모와 자식의 필드명이 같거나, 메서드가 오버라이딩 되어 있으면, 자식에서 부모의 필드나 메서드를 호출할 수 없음.

이 때, super 키워드를 사용하면 부모를 참조할 수 있다.
super는 이름 그대로 부모 클래스에 대한 참조를 나타낸다.

super.value / super.value()

[super - 생성자]

상속 관계의 인스턴스를 생성하면 결국 메모리 내부에는 자식과 부모 클래스가 각각 다 만들어짐.

Child를 만들면 Parent까지 함께 만들어지는 것.
즉, 각각 생성자도 모두 호출되어야 하는 것.

- **상속 관계를 사용하면 자식 클래스의 생성자에서 부모 클래스의 생성자를 반드시 호출해야 한다.**
- 부모의 기본 생성자(매개변수 없는)는 생략이 가능.
  기본 생성자만 생략 가능하며, 기본 생성자가 없이 정의한 생성자만 있는 경우 super(매개,변수)로 직접 선언해줘야 함.

ClassA → ClassB → ClassC 라면 생성자 호출 순서는, ClassA → ClassB → ClassC
super()는, 각 클래스 생성자의 맨 위에만 호출이 가능함. (같이 생성되기 때문인듯.)

- 초기화는 최상위 부모부터 이루어진다.
- 자식은 부모의 값을 가져다 쓰므로, 부모부터 초기화 되는게 맞는듯.

- 상속 관계의 생성자 호출은 결과적으로 부모 → 자식 순서.
- 상속 관계에서 첫줄에는 this를 제외하면 super가 호출 되어야 함.
  this를 통해 내 생성자로 아무리 보내더라도 언젠가 마지막에 한번은 맨 윗줄에 super()가 들어가 있어야 함.