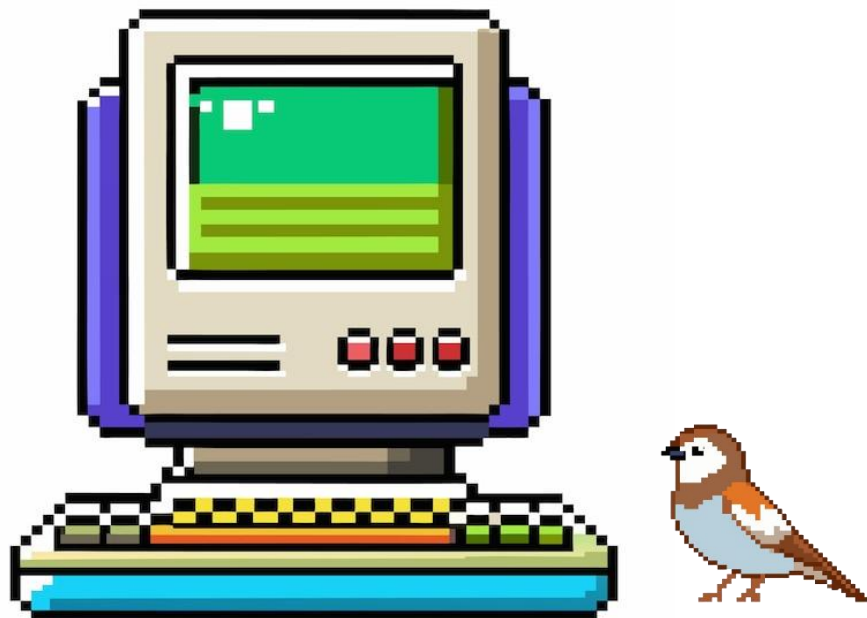


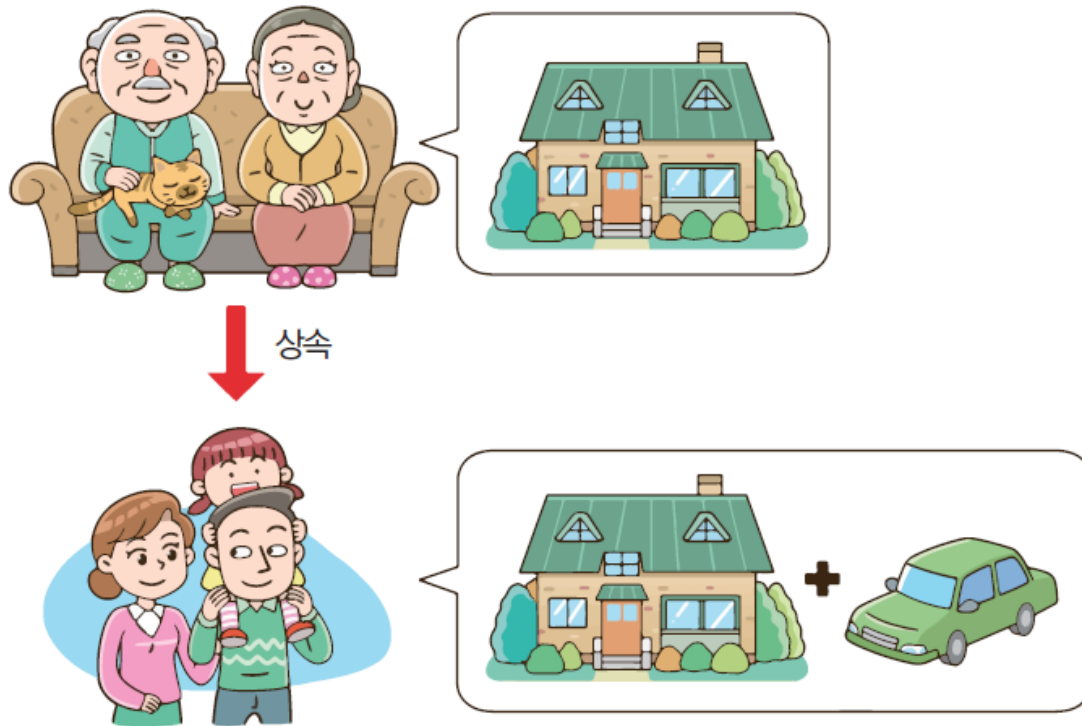
## 07. 상속



강동기

# 상속이란

- 현실 세계에도 존재하는 상속의 개념



상속을 이용하면 쉽게  
재산을 모을 수 있는 것처럼  
소프트웨어도 쉽게 개발할  
수 있습니다.

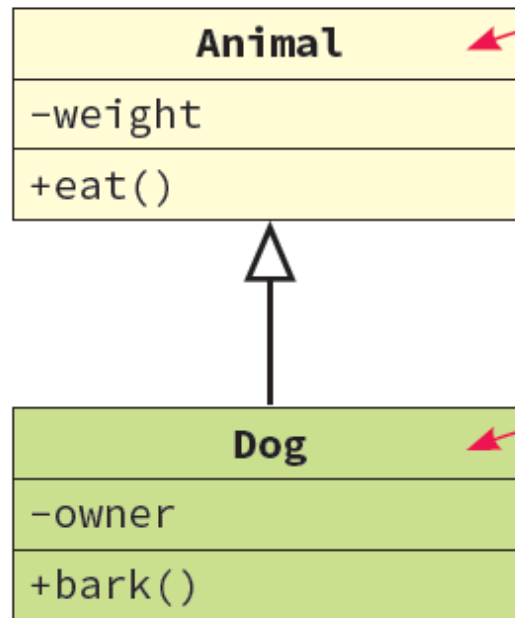
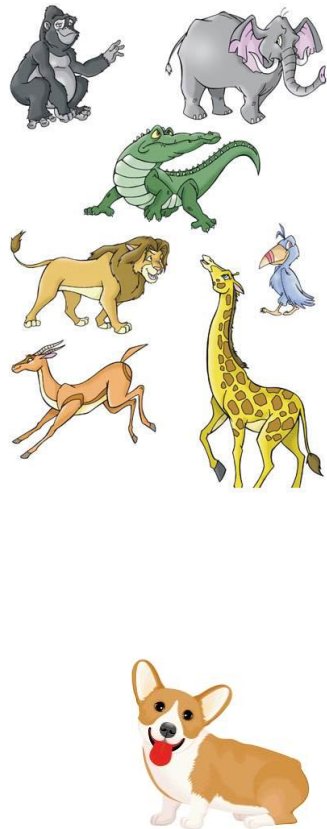


# 부모클래스와 자식클래스

- 부모 클래스 (parent class): 상속해주는 클래스
- 자식 클래스 (child class): 상속받는 클래스
- 부모 클래스는 추상적이고 자식 클래스는 구체적

부모 클래스	자식 클래스
Animal(동물)	Lion(사자), Dog(개), Cat(고양이)
Bike(자전거)	MountainBike(산악자전거), RoadBike, TandemBike
Vehicle(탈것)	Car(자동차), Bus(버스), Truck(트럭), Boat(보트), Motorcycle(오토바이), Bicycle(자전거)
Student(학생)	GraduateStudent(대학원생), UnderGraduate(학부생)
Employee(직원)	Manager(관리자)
Shape(도형)	Rectangle(사각형), Triangle(삼각형), Circle(원)

# 부모클래스와 자식클래스



부모 클래스 또는  
수퍼 클래스라고 한다.

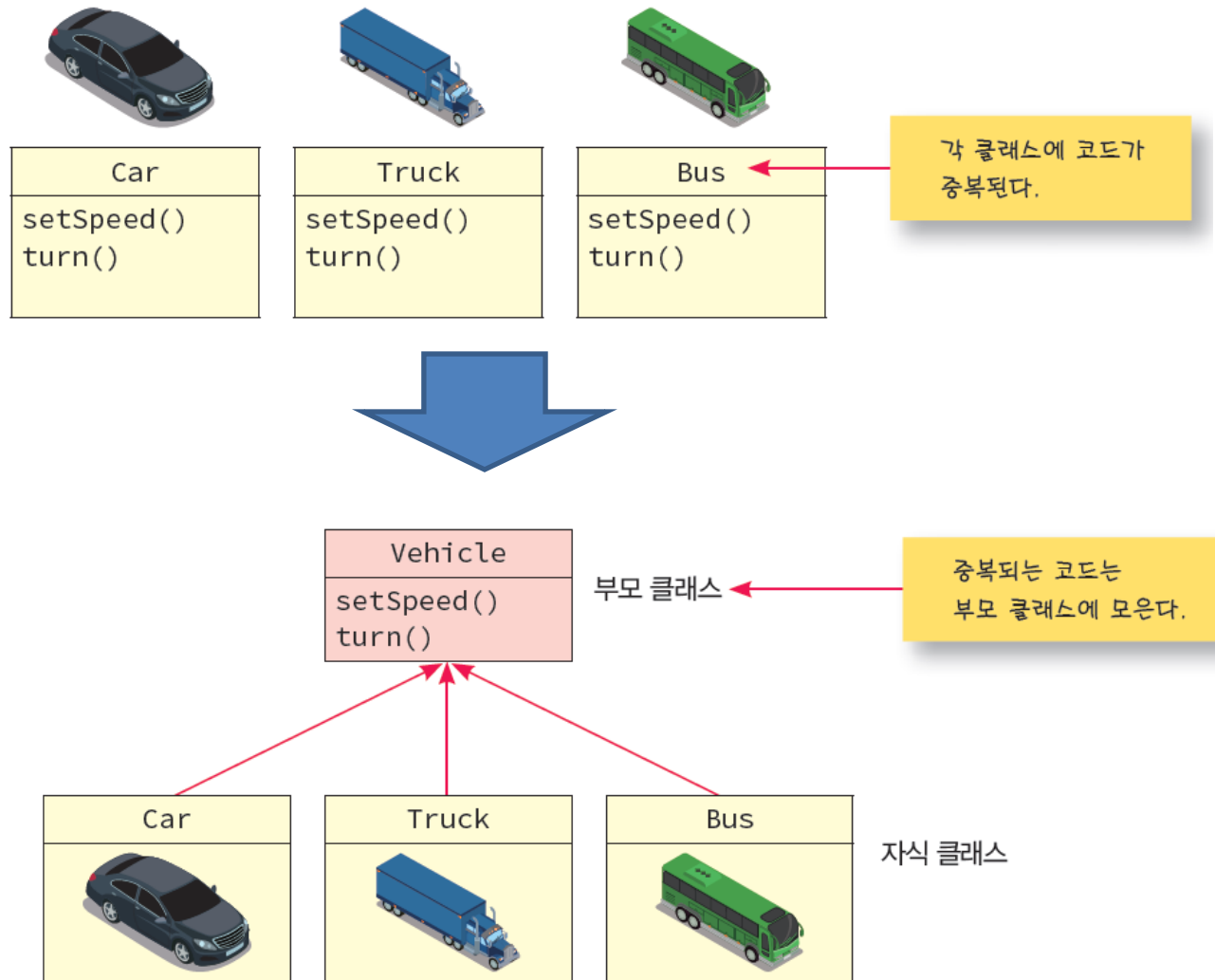
자식 클래스 또는  
서브 클래스라고 한다.

# 상속의 장점

---

- 상속을 통하여 기존 클래스의 필드와 메소드를 재사용
  - 기존 클래스의 내용 변경 가능
  - 상속을 통해 복잡한 GUI 프로그램을 손쉽게 작성
- 이미 검증된 소프트웨어를 재사용
  - 신뢰성 있는 소프트웨어를 손쉽게 개발, 유지 보수
  - 코드의 중복을 줄일 수 있음

# 상속의 장점



# 상속의 형식

전체적인 구조



형식

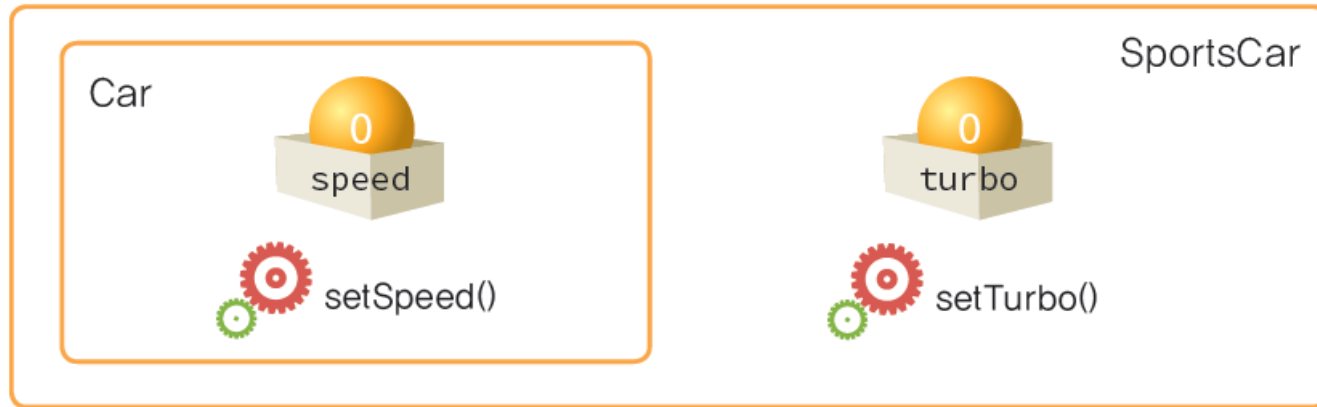
자식 클래스 또는 서브 클래스라고 한다.

```
class Childclass extends Parentclass
{
    // 여기에 필드를 추가한다.
    // 여기에 메소드를 추가한다.
}
```

부모 클래스 또는 슈퍼 클래스라고 한다.

# 상속의 형식

- 자식 클래스는 부모 클래스 내용을 포함



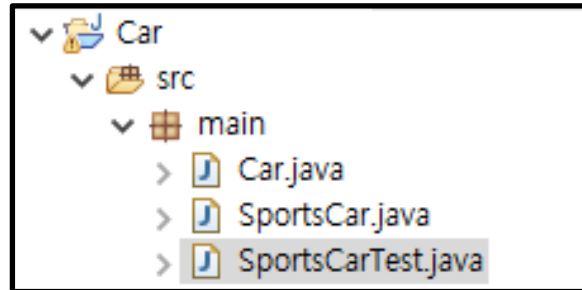
Car =  + 

SportsCar =  +  +  + 



# 예제: 자동차클래스 상속

- 프로젝트 구조



# 예제: 자동차클래스 상속

- Car.Java

```
package main;

public class Car {
    int speed;

    public void setSpeed(int speed) {
        this.speed = speed;
    }
}
```

# 예제: 자동차클래스 상속

- SportsCar.Java

```
package main;

public class SportsCar extends Car {
    boolean turbo;

    public void setTurbo(boolean flag) {
        turbo = flag;
    }
}
```

# 예제: 자동차클래스 상속

- SportsCarTest.Java

```
package main;

public class SportsCarTest {
    public static void main(String[] args) {
        SportsCar sportsCar = new SportsCar();

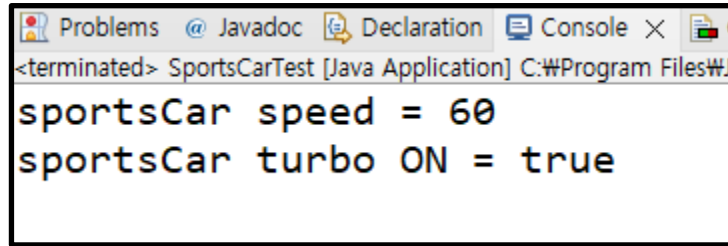
        sportsCar.speed = 10;
        sportsCar.setSpeed(60);

        sportsCar.setTurbo(true);

        System.out.println("sportsCar speed = " + sportsCar.speed);
        System.out.println("sportsCar turbo ON = " + sportsCar.turbo);
    }
}
```

# 예제: 자동차클래스 상속

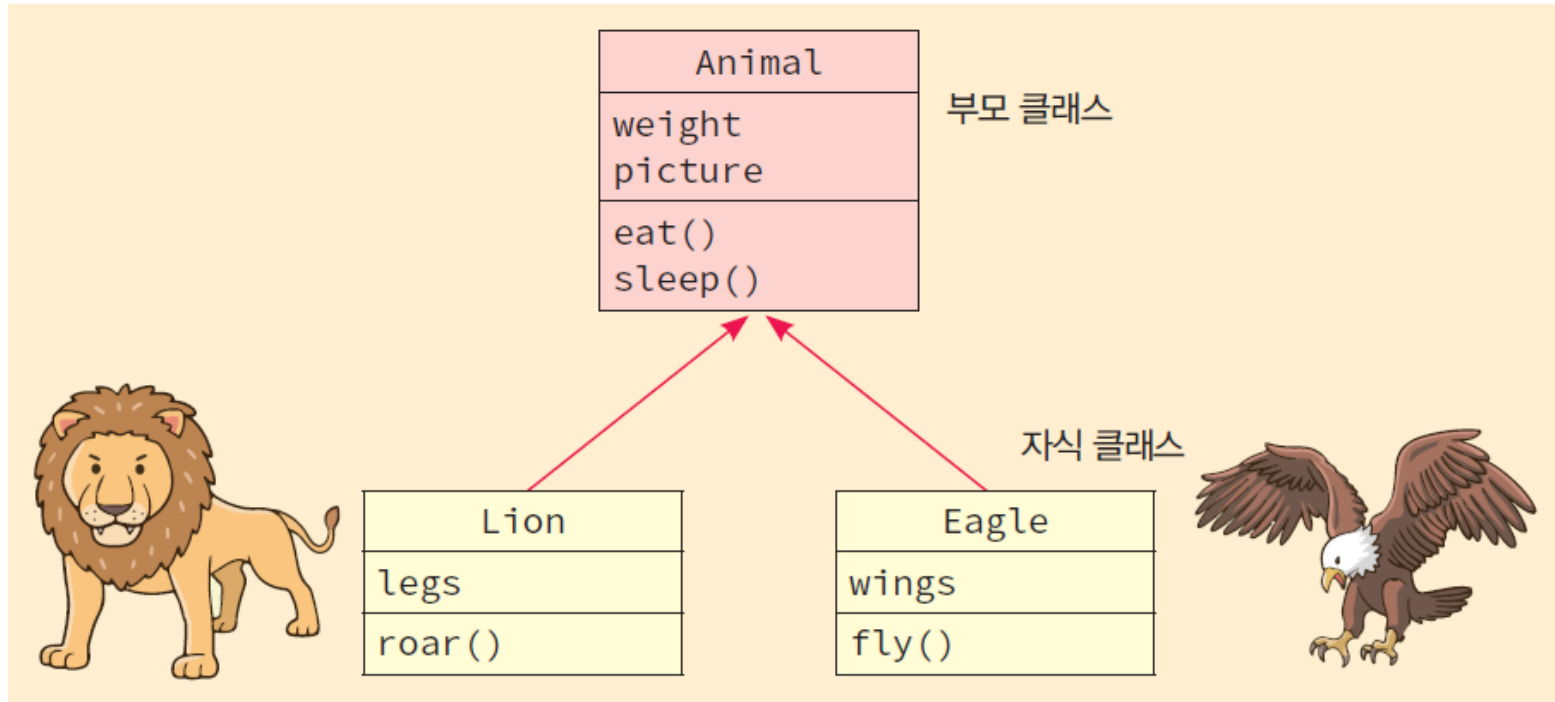
- SportsCarTest.Java 실행결과



The screenshot shows a Java IDE window with a console tab. The title bar indicates the application is "SportsCarTest [Java Application]" and the file path is "C:\Program Files\WJ...". The console output displays two lines of text: "sportsCar speed = 60" and "sportsCar turbo ON = true".

```
<terminated> SportsCarTest [Java Application] C:\Program Files\WJ...  
sportsCar speed = 60  
sportsCar turbo ON = true
```

# 예제: 동물클래스 상속



# 예제: 동물클래스 상속

- 프로젝트 구조



# 예제: 동물클래스 상속

- Animal.java

```
package main;

public class Animal {
    private double weight;
    private String picture;

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

    public void sleep() {
        System.out.println("get some sleep");
    }
}
```



# 예제: 동물클래스 상속

- Lion.java

```
package main;

public class Lion extends Animal {
    private int legs = 4;

    public void roar() {
        System.out.println("The lion is roaring");
    }
}
```

## 예제: 동물클래스 상속

- Eagle.java

```
package main;

public class Eagle extends Animal {
    private int wings = 2;

    public void fly() {
        System.out.println("The eagle is flying");
    }
}
```

# 예제: 동물클래스 상속

- Test.java

```
package main;

public class Test {
    public static void main(String[] args) {
        Animal animal = new Animal();
        Lion lion = new Lion();
        Eagle eagle = new Eagle();

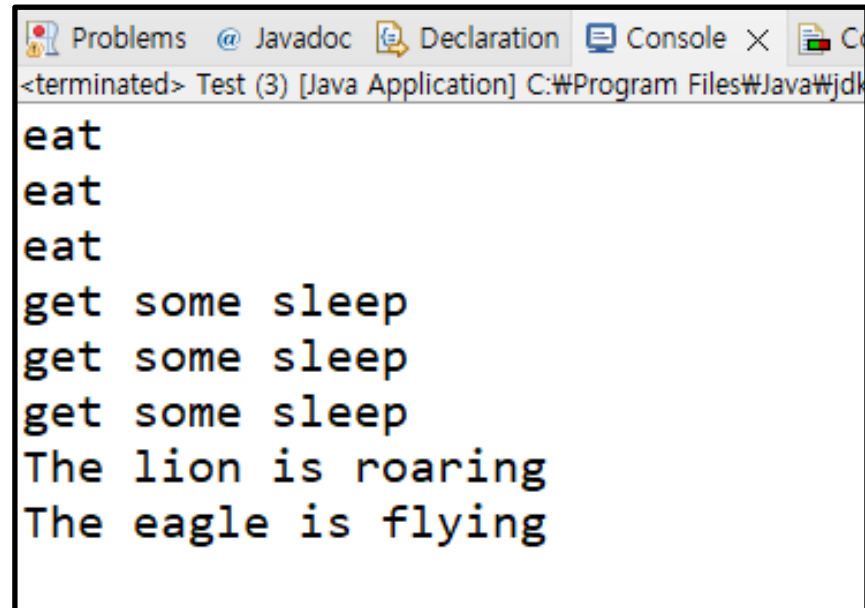
        animal.eat();
        lion.eat();
        eagle.eat();

        animal.sleep();
        lion.sleep();
        eagle.sleep();

        lion.roar();
        eagle.fly();
    }
}
```

# 예제: 동물클래스 상속

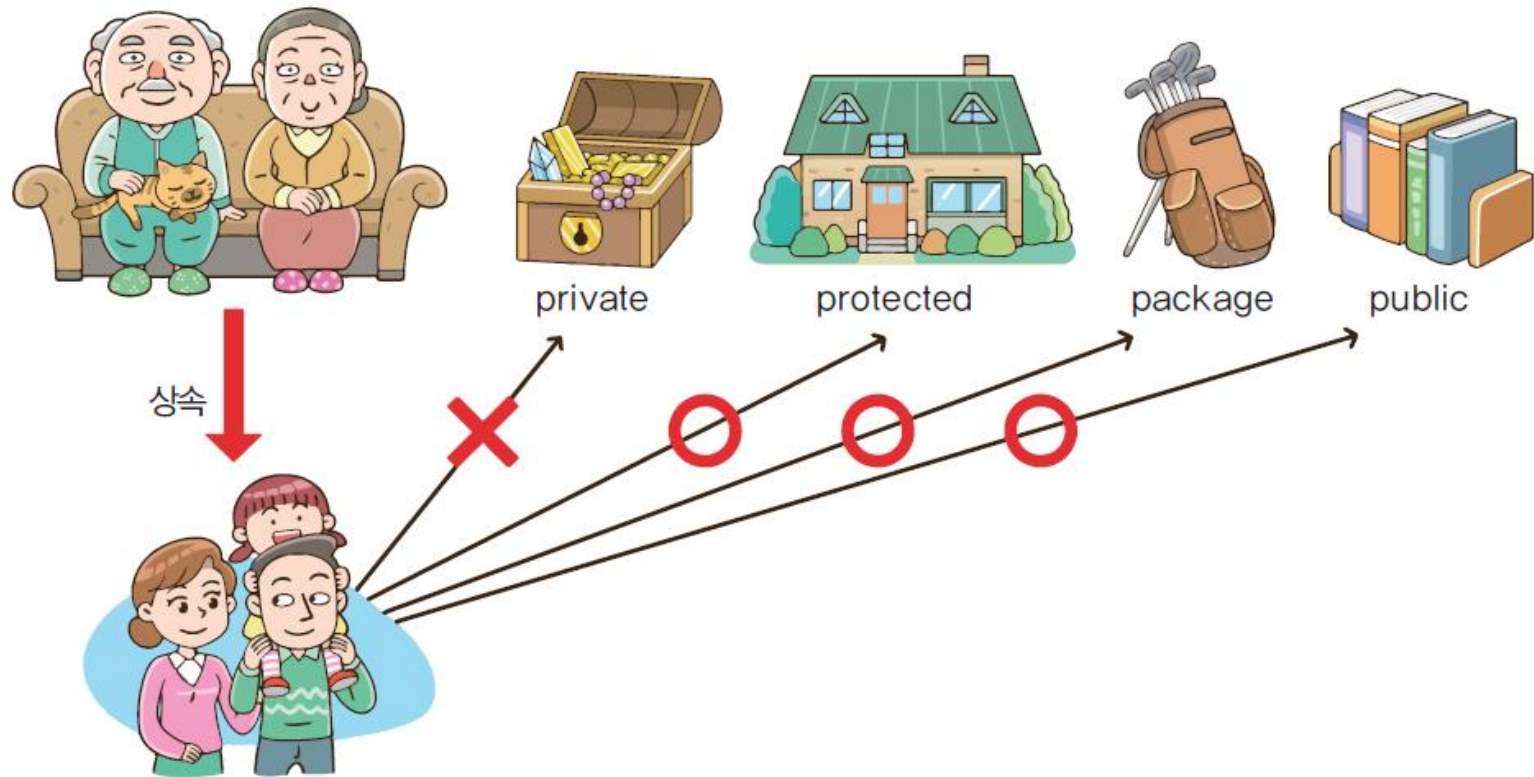
- Test.java 실행결과



The screenshot shows a Java IDE window with a console tab. The title bar includes 'Problems', 'Javadoc', 'Declaration', 'Console', and a close button. The console text reads: '<terminated> Test (3) [Java Application] C:\Program Files\Java\jdk...'. The output of the program is displayed in a monospaced font:

```
eat
eat
eat
get some sleep
get some sleep
get some sleep
The lion is roaring
The eagle is flying
```

# 상속과 접근제어

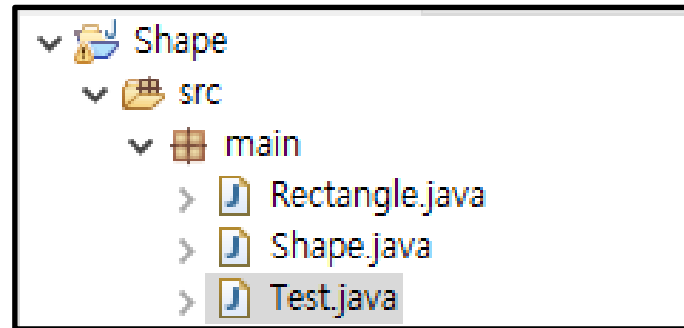


# 상속과 접근제어

접근 지정자	클래스	패키지	자식 클래스	전체 세계
public	O	O	O	O
protected	O	O	O	X
없음	O	O	X	X
private	O	X	X	X

# 예제: 상속과 접근제어

- 프로젝트 구조



# 예제: 상속과 접근제어

- Shape.java

```
package main;

public class Shape {
    private int x;
    private int y;

    void print() {
        System.out.println("x: " + x + " y: " + y);
    }
}
```



# 예제: 상속과 접근제어

- Rectangle.java (에러 발생)
  - 에러가 발생하지 않게 하려면??

```
package main;

public class Rectangle extends Shape {
    private int width;
    private int height;

    double calcArea() {
        return width * height;
    }

    void draw() {
        System.out.println("At (" + x + ", " + y + ") " + "width: "
            + width + " height: " + height);
    }
}
```

# 예제: 상속과 접근제어

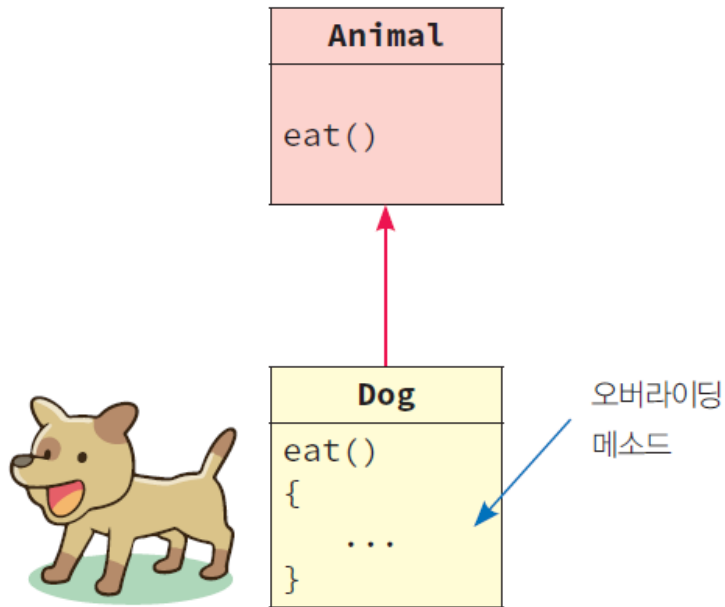
- Test.java

```
package main;

public class Test {
    public static void main(String[] args) {
        Rectangle rect = new Rectangle();
        rect.draw();
    }
}
```

# 메소드 오버라이딩

- method overriding
- 자식 클래스가 필요에 따라 상속된 메소드를 재 정의하는 것
- 메소드 오버로딩(method overloading)과 헷갈리지 말 것



메소드 오버라이딩이란 부모 클래스의 메소드를 자식 클래스가 자신의 필요에 맞추어서 변경하는 것입니다.

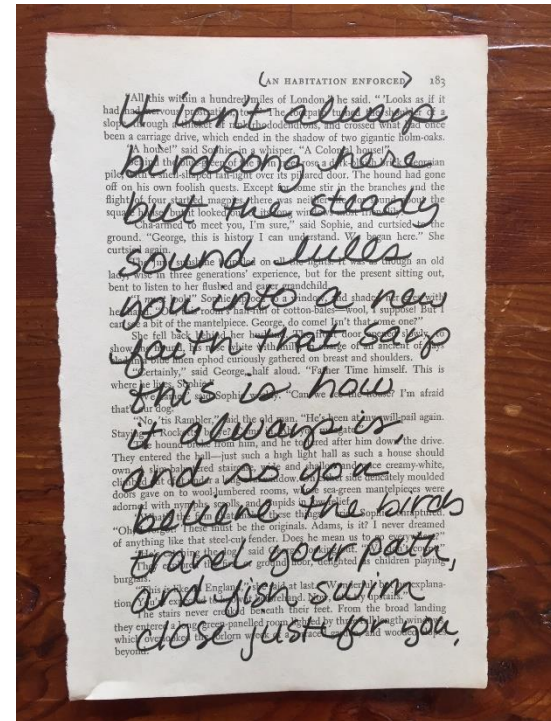


# 메소드 오버라이딩

- 메소드 오버로딩 (method overloading)
  - 이름은 동일하고 파라미터 자료형이 서로 다른 여러개의 메소드를 같은 공간에 정의하는 것
- 메소드 오버라이딩 (method overriding)
  - 자식 클래스가 필요에 따라 상속된 메소드를 재 정의하는 것



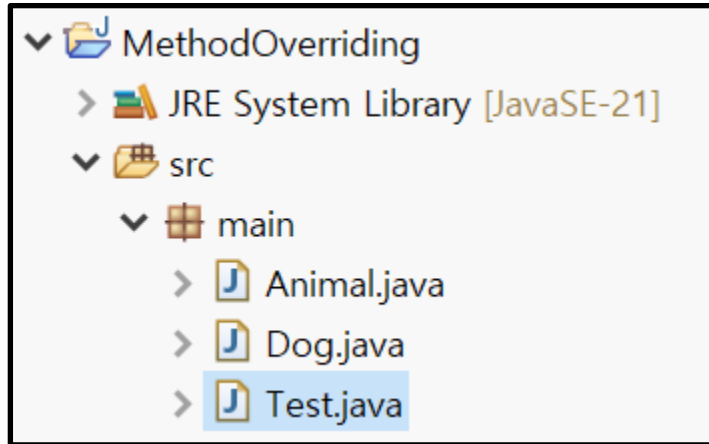
\* 주의: 스타 유닛 스펠링은 Overlord



덮어쓰다: overwriting

# 예제: 메소드 오버라이딩

- 프로젝트 구조



# 예제: 메소드 오버라이딩

- Animal.java

```
package main;

public class Animal {
    public void eat() {
        System.out.println("An animal eats it.");
    }
}
```

# 예제: 메소드 오버라이딩

- Dog.java

```
package main;

public class Dog extends Animal {
    public void eat() {
        System.out.println("A dog eats it.");
    }
}
```

# 예제: 메소드 오버라이딩

- Test.java

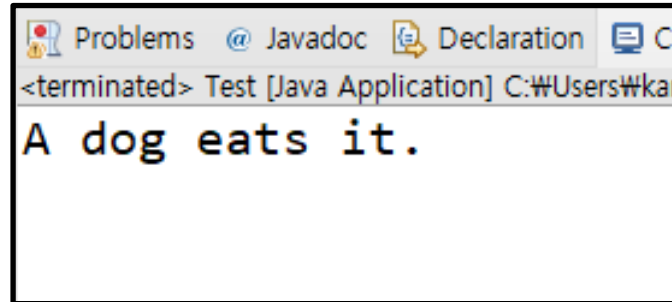
```
package main;

public class Test {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.eat();
    }
}
```



# 예제: 메소드 오버라이딩

- Test.java 실행결과



The screenshot shows a Java IDE's console window. The title bar includes icons for 'Problems', 'Javadoc', 'Declaration', and 'Console'. The text in the console reads: '<terminated> Test [Java Application] C:\Users\kai...' followed by the output 'A dog eats it.' on a new line.

```
<terminated> Test [Java Application] C:\Users\kai...  
A dog eats it.
```

# 예제: 메소드 오버라이딩

- 메소드 오버라이딩 조건
  - 접근 지정자와 리턴 타입이 일치하여야 함

```
package main;

public class Animal {
    public void eat() {
        System.out.println("An animal eats it.");
    }

    public void sound() {
        System.out.println("Grrrrr...");
    }
}
```

sound()의 리턴 타입이 public, private 으로  
서로 일치하지 않아 에러 발생!

```
package main;

public class Dog extends Animal {

    public void eat() {
        System.out.println("A dog eats it.");
    }

    private void sound() {
        System.out.println("Bow~ Wow~");
    }
}
```

# 예제: 메소드 오버라이딩

- 메소드 오버라이딩 조건
  - 접근 지정자와 리턴 타입이 일치하여야 함

```
package main;

public class Animal {
    public void eat() {
        System.out.println("An animal eats it.");
    }

    public void sound(int a) {
        System.out.println("Grrrrr...");
    }
}
```

sound()의 파라미터(시그니처)가 다른 경우  
오버라이딩이 아닌 오버로딩이 적용되어서  
에러는 발생하지 않음

```
package main;

public class Dog extends Animal {

    public void eat() {
        System.out.println("A dog eats it.");
    }

    public void sound(double a) {
        System.out.println("Bow~ Wow~");
    }
}
```

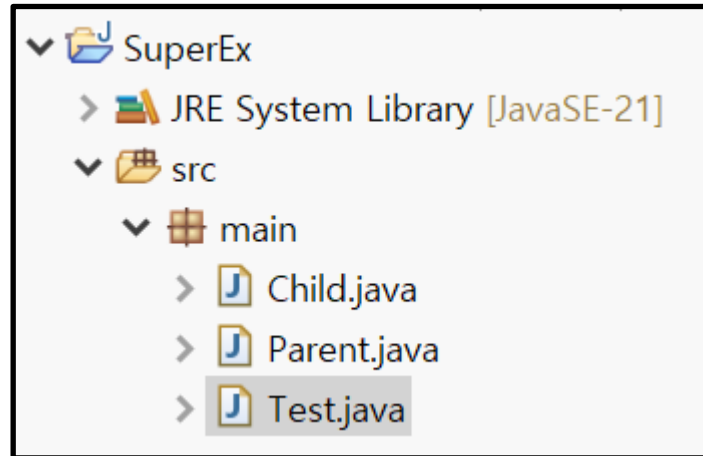
# super

---

- this: 자신의 객체
- **super: 부모의 객체**
- super는 상속 관계에서만 (자식 클래스에서만) 사용할 수 있음

# 예제: super

- 프로젝트 구조



# 예제: super

- Parent.java

```
package main;

public class Parent {
    public void print() {
        System.out.println("I love my child.");
    }
}
```

## 예제: super

- Child.java

```
package main;

public class Child extends Parent{
    public void print() {
        super.print();
        System.out.println("I want to be away from my parents.");
    }
}
```

# 예제: super

- Test.java

```
package main;

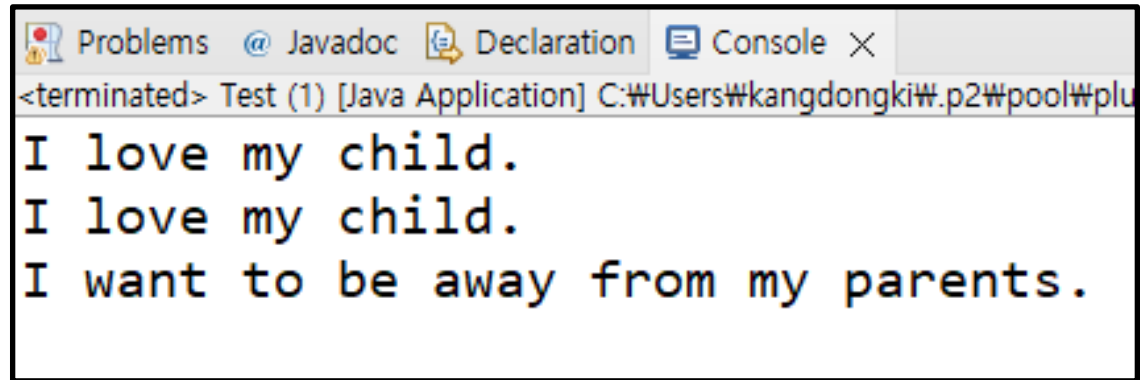
public class Test {
    public static void main(String[] args) {
        Parent parent = new Parent();
        Child child = new Child();

        parent.print();
        child.print();
    }
}
```



# 예제: super

- Test.java 실행결과



The screenshot shows an IDE console window with the following tabs: Problems, Javadoc, Declaration, and Console. The console title bar indicates the application is terminated. The output text is as follows:

```
<terminated> Test (1) [Java Application] C:\Users\kangdongki\p2\pool\plu  
I love my child.  
I love my child.  
I want to be away from my parents.
```

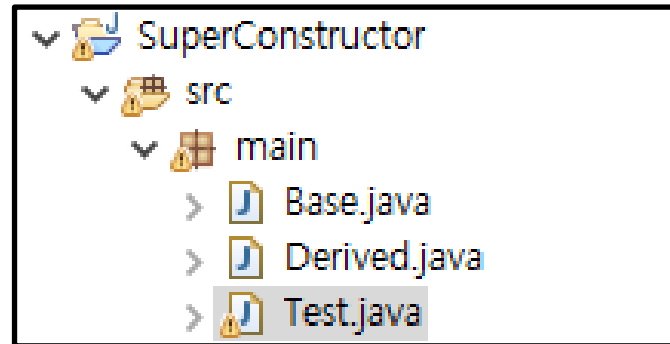
# 상속과 생성자

---

- 자식 클래스는...
  - 부모 클래스의 상속 부분과 자식 클래스의 추가 부분으로 구성되었음
  - 자식 클래스의 객체를 생성할 때 자식 클래스의 안의 부모 클래스 부분을 초기화하기 위하여 부모 클래스의 생성자가 호출됨
- 생성자의 실행 순서는...
  - 생성자 호출
  - 부모 클래스의 생성자 코드 실행
  - 자식 클래스의 생성자 코드 실행

# 예제: 상속과 생성자

- 프로젝트 구조



# 예제: 상속과 생성자

- Base.java

```
package main;

public class Base {
    public Base(){
        System.out.println("in Base()");
    }
}
```

# 예제: 상속과 생성자

- Derived.java

```
package main;

public class Derived extends Base{
    public Derived() {
        System.out.println("in Derived()");
    }
}
```

# 예제: 상속과 생성자

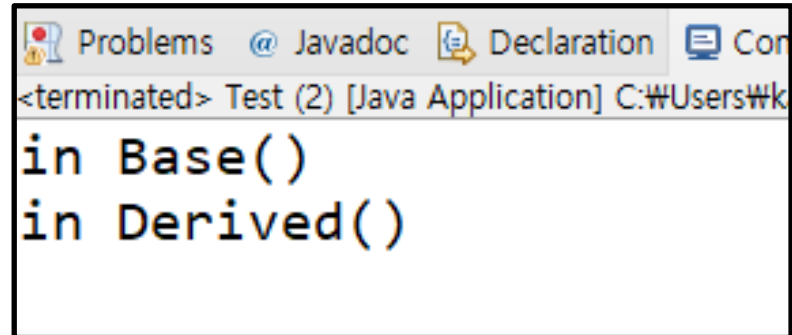
- Test.java

```
package main;

public class Test {
    public static void main(String[] args) {
        Derived d = new Derived();
    }
}
```

# 예제: 상속과 생성자

- Test.java 실행결과



```
<terminated> Test (2) [Java Application] C:\Users\Wk
in Base()
in Derived()
```

# super()

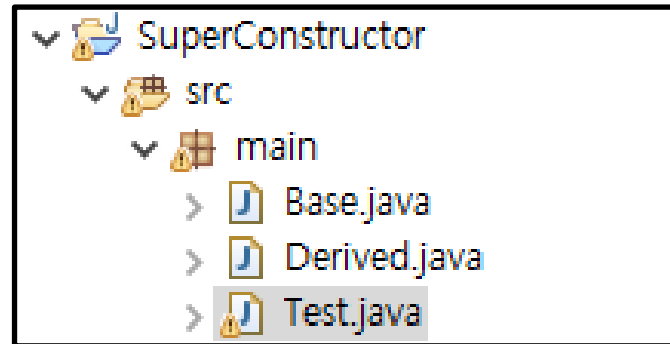
---

- this(): 자신의 생성자
- **super(): 부모의 생성자**
- this()와 super()는 생성자 안에서만, 그리고 첫 줄에만 올 수 있음



# 예제: super()

- 프로젝트 구조



# 예제: super()

- Base.java

```
package main;

public class Base {
    public Base(){
        System.out.println("in Base()");
    }
    public Base(int a){
        System.out.println("in Base() with " + a);
    }
}
```

## 예제: super()

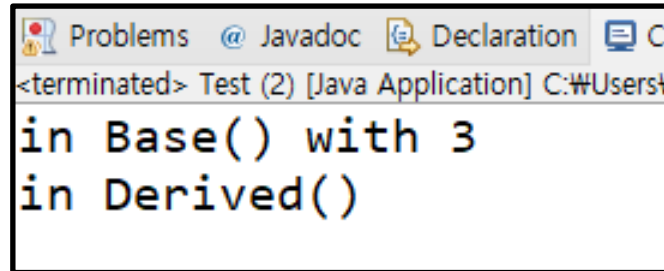
- Derived.java

```
package main;

public class Derived extends Base{
    public Derived() {
        super(3);
        System.out.println("in Derived()");
    }
}
```

# 예제: super()

- Test.java 실행결과



```
<terminated> Test (2) [Java Application] C:\Users\...  
in Base() with 3  
in Derived()
```

# 연습문제 1

- 페이지 14-20을 참고하자
- BabyLion 클래스를 추가하고, Test.java를 아래와 같이 수정 후 실행하자

```
package main;

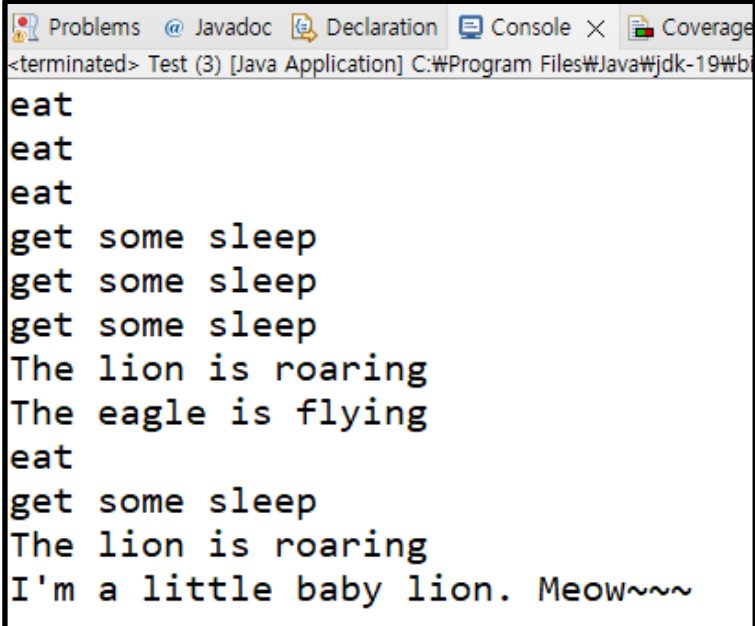
public class Test {
    public static void main(String[] args) {
        Animal animal = new Animal();
        Lion lion = new Lion();
        Eagle eagle = new Eagle();

        animal.eat();
        lion.eat();
        eagle.eat();

        animal.sleep();
        lion.sleep();
        eagle.sleep();

        lion.roar();
        eagle.fly();

        BabyLion babyLion = new BabyLion();
        babyLion.eat();
        babyLion.sleep();
        babyLion.roar();
        babyLion.meow();
    }
}
```



The screenshot shows a Java IDE window with a console output. The title bar includes 'Problems', '@ Javadoc', 'Declaration', 'Console', and 'Coverage'. The console text is as follows:

```
<terminated> Test (3) [Java Application] C:\Program Files\Java\jdk-19\bin
eat
eat
eat
get some sleep
get some sleep
get some sleep
The lion is roaring
The eagle is flying
eat
get some sleep
The lion is roaring
I'm a little baby lion. Meow~~~
```

## 연습문제 2

- 페이지 23-26을 참고하자.
- Rectangle 클래스에서 발생하는 에러를 해결하기 위해...
  - 적절한 지정자를 가진 setter / getter 메소드를 구현하여 접근할 것

```
package main;

public class Shape {
    private int x;
    private int y;

    void print() {
        System.out.println("x: " + x + " y: " + y);
    }

    ??????

}
```

## 연습문제 3

---

- 다음의 조건을 만족하는 프로젝트를 개발하자.
  - (자유롭게) 부모클래스와 자식클래스를 생성
  - 부모클래스의 생성자를 3개 작성 (생성자 오버로딩 사용)
  - 부모클래스의 메소드를 3개 작성
  - 자식클래스의 생성자를 3개 작성 (생성자 오버로딩 사용)
  - 자식클래스의 메소드를 3개 작성
  - 자식클래스의 생성자 코드안에 `super()` 키워드 넣기
  - 자식 클래스의 메소드 1개는 부모클래스 메소드 1개를 오버라이딩 함

**감사합니다! XD**

