

## 상속( Inheritance)

1. Super class: 상속해 주는 class  
sub class : 상속받는 class
2. 모든 class는 object class로 부터 상속 받는다.
3. 단일 상속만 가능
4. 부모클래스의 모든 것을 상속받는다

형식> class sub클래스명 extends super클래스명

Frame 클래스를 상속받아서  
윈도우창을 하나 만들자

```
import java.awt.Color;
import java.awt.Frame;

public class WindowTest extends Frame {
    public WindowTest ()
    {
        this.setSize(300, 400);
        this.setVisible(true);
        this.setBackground(Color. YELLOW);
    }

    public static void main(String[] args) {
        WindowTest m= new WindowTest ();
    }
}
```



상속

## 상속 (Inheritance)

코드의 재사용, 코드 집중화  
코드의 확장, 유지보수 용이

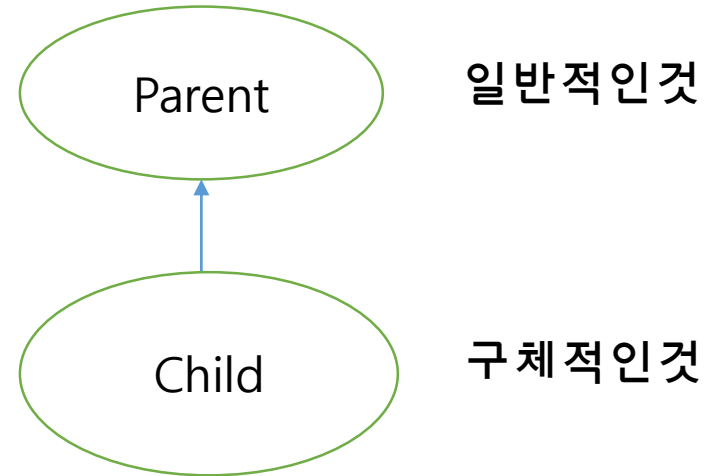
코드중복 제거  
클래스 간결

# 상속 (Inheritance)

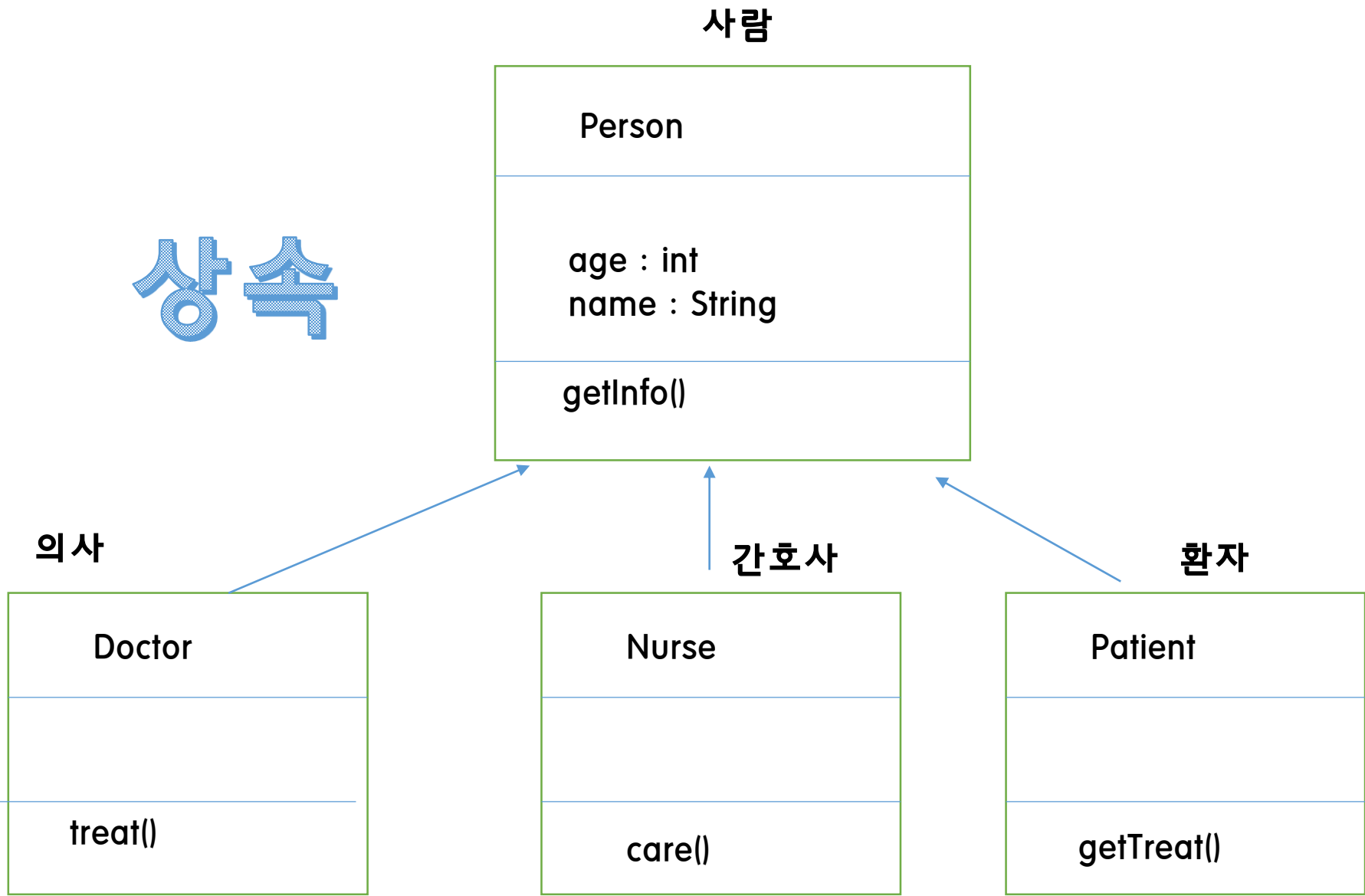
기존의 클래스로 새로운 클래스를 작성하는 것(코드의 재사용)  
두 클래스를 부모와 자식으로 관계를 맺어 주는 것

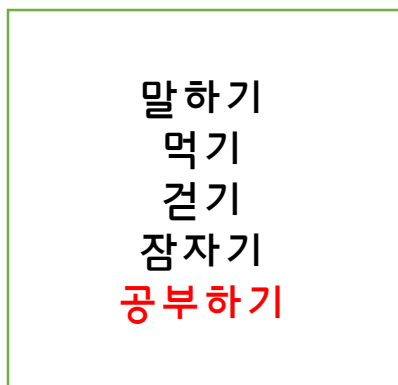
```
class 자식클래스 extends 부모클래스 {  
}
```

```
class Parent{ }  
class Child extends parent{ }
```

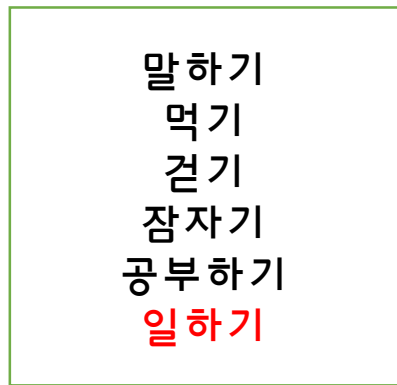


상속

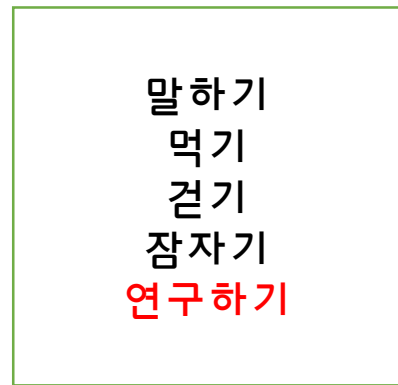




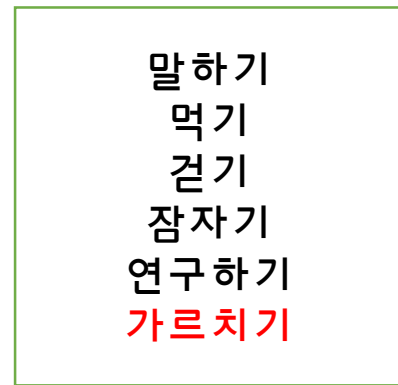
Student



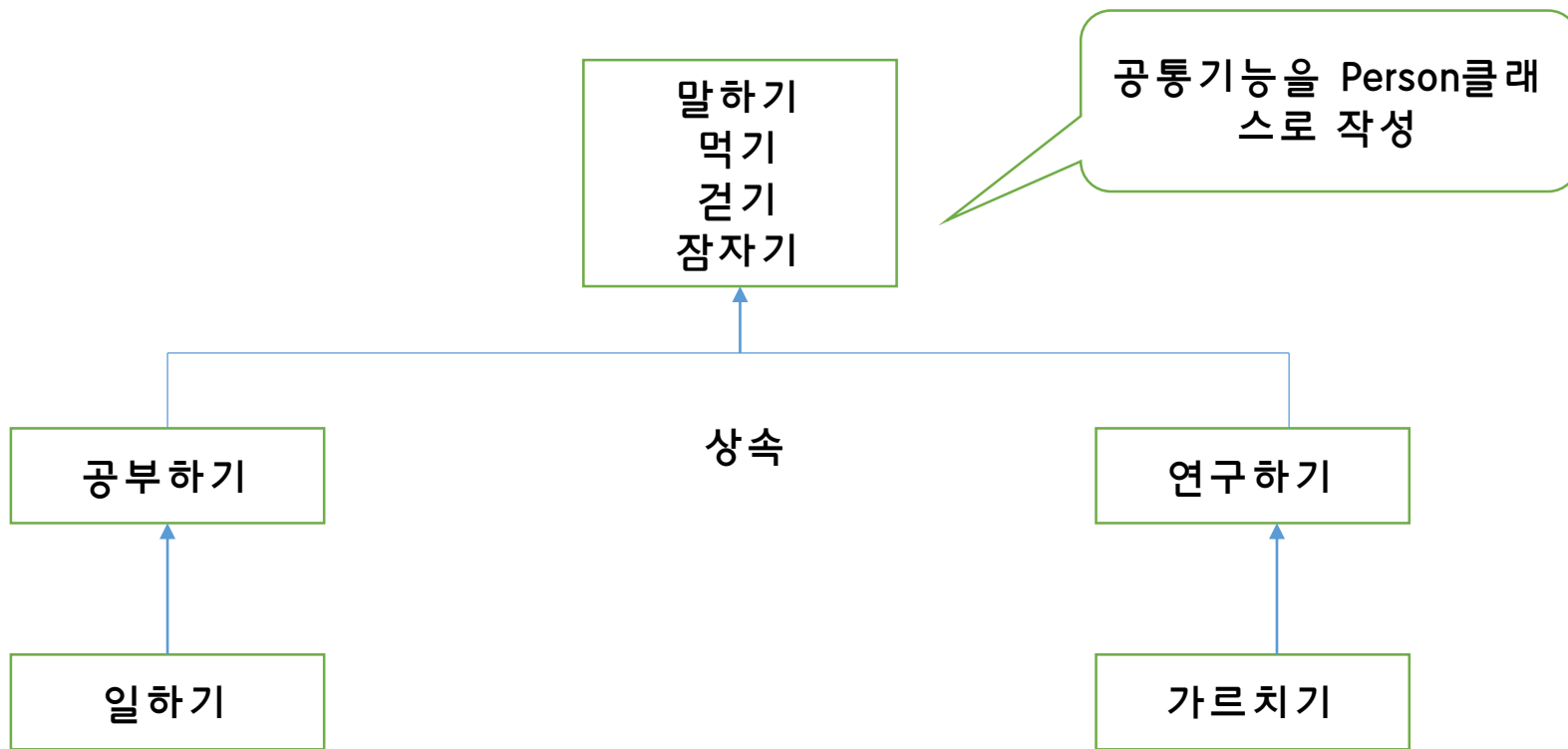
StudentWorker



Researcher



Professor



상속관계에서  
생성자 호출

```
public class Person {
    protected String name;
    protected String id;

    public Person(String name, String id)
    {
        System.out.println("부모 생성자 호출");
        this.name = name;
        this.id = id;
    }

    public Person()
    {
        System.out.println("부모 디폴트 생성자 호출");
    }

    public void disp()
    {
        System.out.println("이름 = " + name + " 아이디 = " + id);
    }
}
```



```

public class Student extends Person{

    int classNo;

    public Student( String name, String id, int classNo)
    {
        super(name, id );
        // 부모의 생성자 호출, (명시적으로 호출) 하지 않으면 부모의 기본생성자를 호출함,
        //부모의 기본생성자가 없으면 에러가 발생함 !!

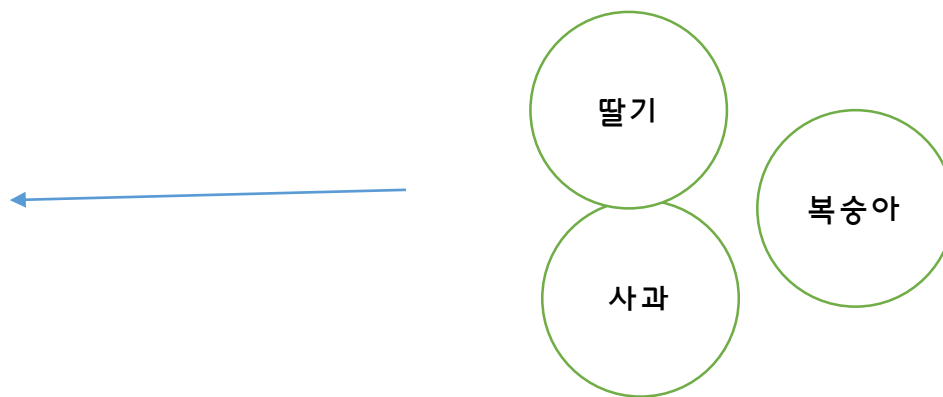
        this.classNo= classNo;
        System.out.println("자식 생성자 호출");
    }
    public void output()
    {
        disp();
        System.out.println(    "반="+ classNo);
    }
    public static void main(String[] args) {
        Student s = new Student(  "김솔" ,  "kimssol"    , 403);
        s.output();
    }
}

```

# 업캐스팅

자식객체는 부모형 참조변수에  
담을 수 있다

과일



사과를 과일이라고 부를 수 있다. 그러나  
과일이라고 부르는 순간 과일의 공통성질만 알 수 있다.

자료형은 크기와 해석방법을 결정한다



사과



과일

```
class A{
```

```
    int a=5, b=2;
```

```
    void sum( )
```

```
    {
```

```
        System.out.println( a+b);
```

```
    }
```

```
}
```

```
class B extends A{
```

```
    int c=10 ,d=2;
```

```
    void minus( )
```

```
    {
```

```
        System.out.println( c-d);
```

```
    }
```

```
}
```

부모형으로 받은 경우 부모  
꺼만 호출할 수 있다.

```
Class TestMain{  
    public static void main( String[] args)
```

```
    {
```

```
        B b = new B();
```

```
        b.sum() ;
```

```
        b.minus();
```



```
    }
```

```
}
```

```
Class TestMain{  
    public static void main( String[] args)
```

```
        A b = new B();
```

```
        b.sum() ;
```

```
        //b.minus(); 호출할 수 없다
```

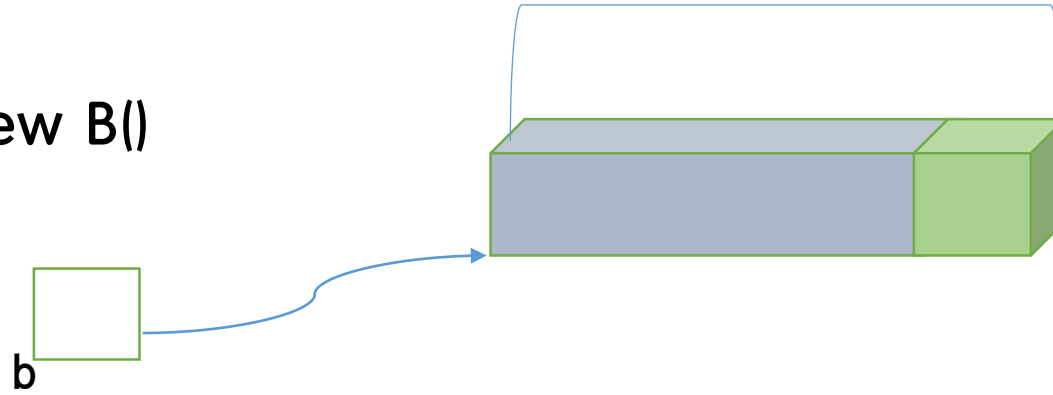


```
}
```

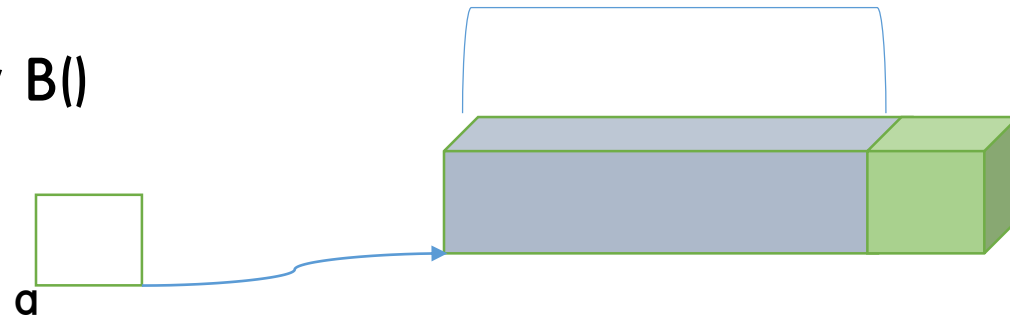
# 상속관계에서 참조형 변수

```
Class TestMain{  
    public static void main( String[] args)  
    {  
        B b  = new B();  
        A a  = new B();  
    }  
}
```

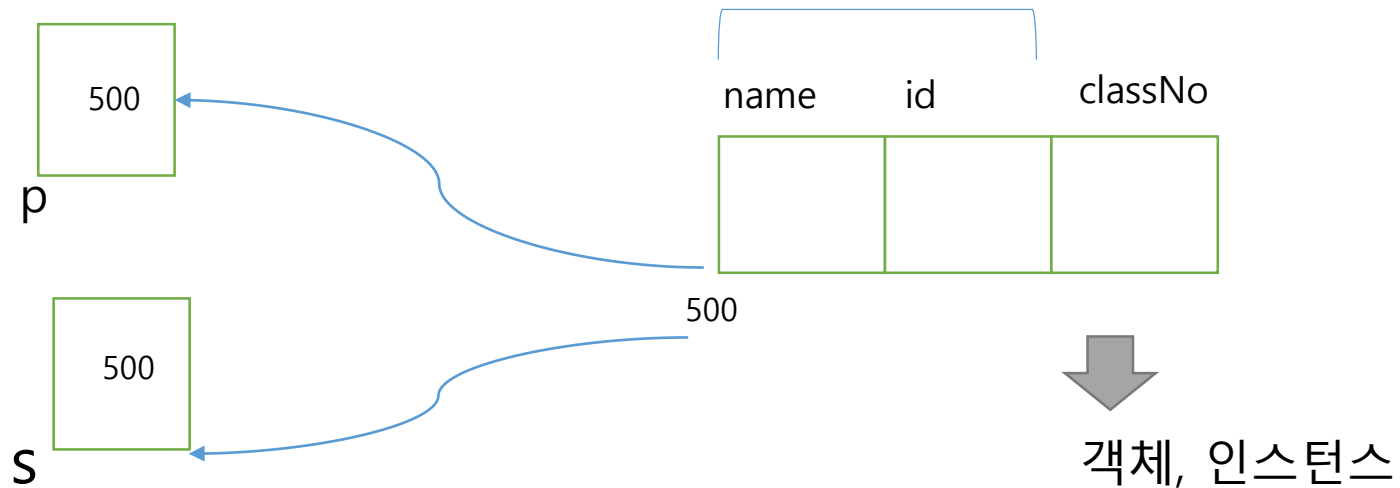
**B** b= new B()



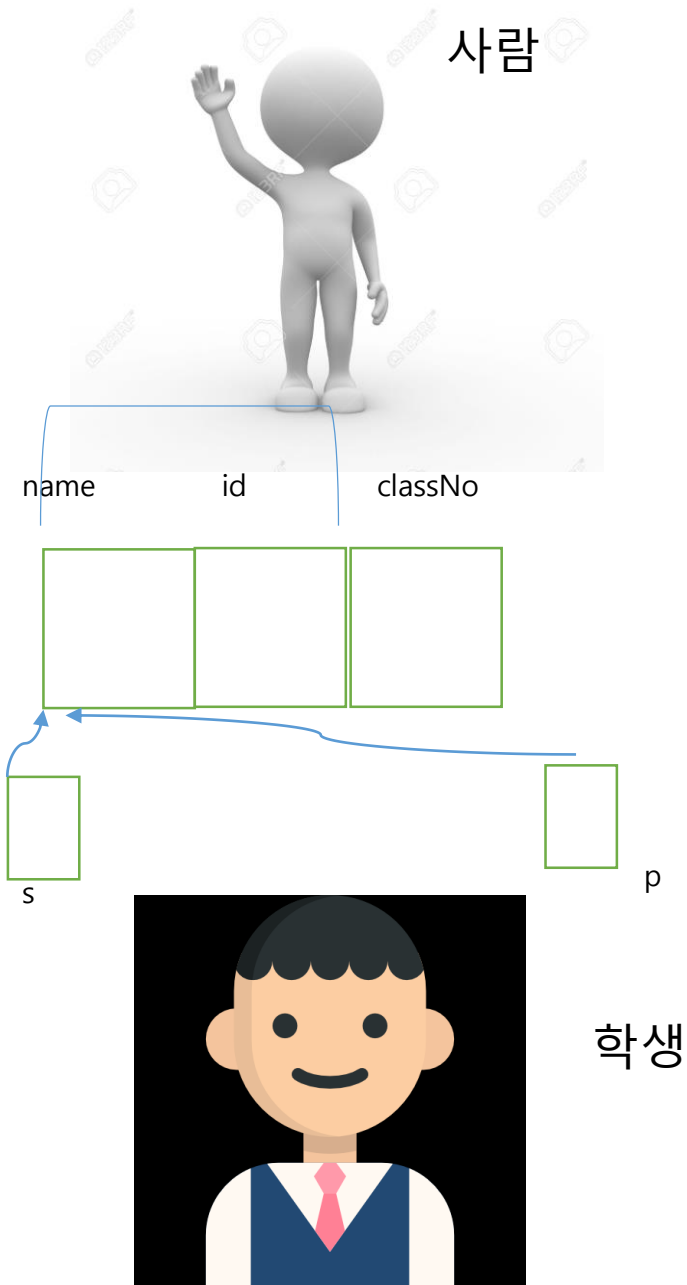
**A** a= new B()



```
Person p;  
Student s= new Student();  
p=s;
```







```
class Person{
    String name; //이름
    String id;    //주민번호

    public Person(String name, String id){
        this.name =name;
        this.id = id;
    }
}
```

```
class Student extends Person{
    int classNo; //반

    public Student(String name, String id, int classNo ){
        super(name , id); //부모의 생성자 호출
        this.classNo = classNo;
    }
}
```

```
class Student UpcastingEx{
    public static void main(String[] args){

        Student s = new Student( "홍길동" , " 090111" , 3);
        Person p;
        p=s; //코드 ok

    }
}
```

상속관계에서 생성자 호출 순서

# 업캐스팅

## 생각해 봅시다.

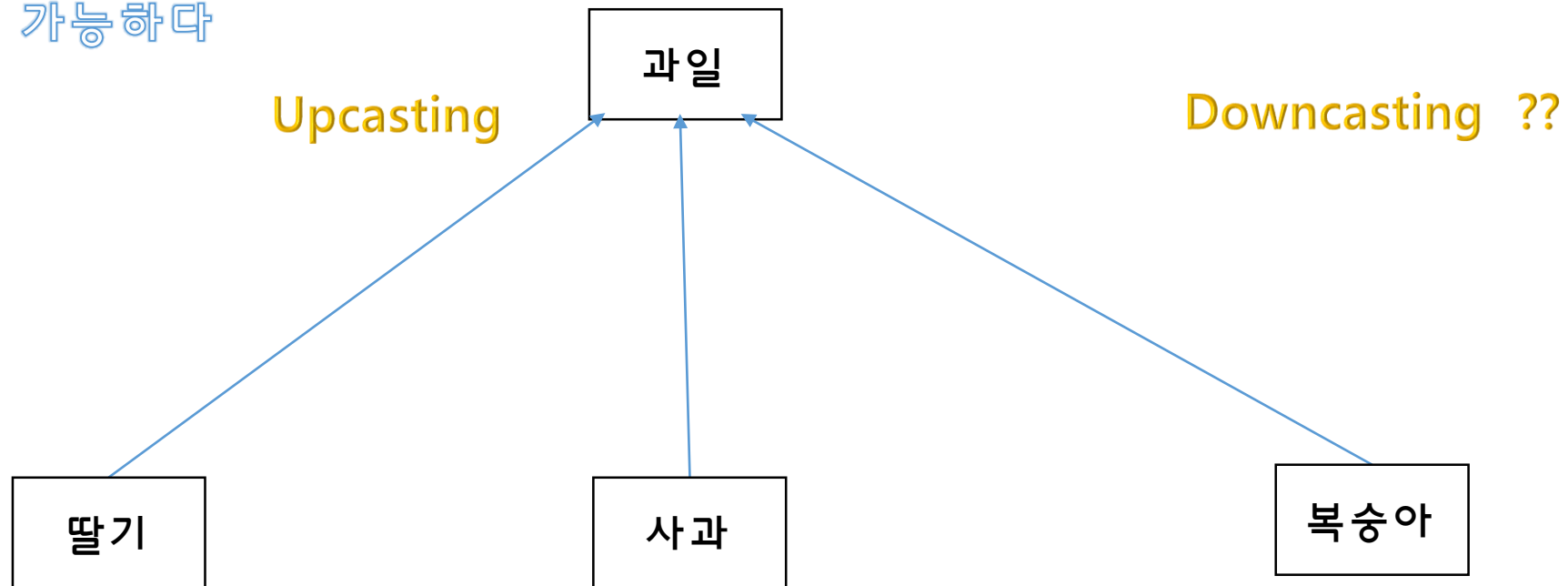
전화기 넘어 상대방에게 과일을  
먹는다고  
하면 상대는 내가 먹는 사과의 모든  
특성을 알 수 있나요.  
그냥 과일의 특성만을 알 수 있죠

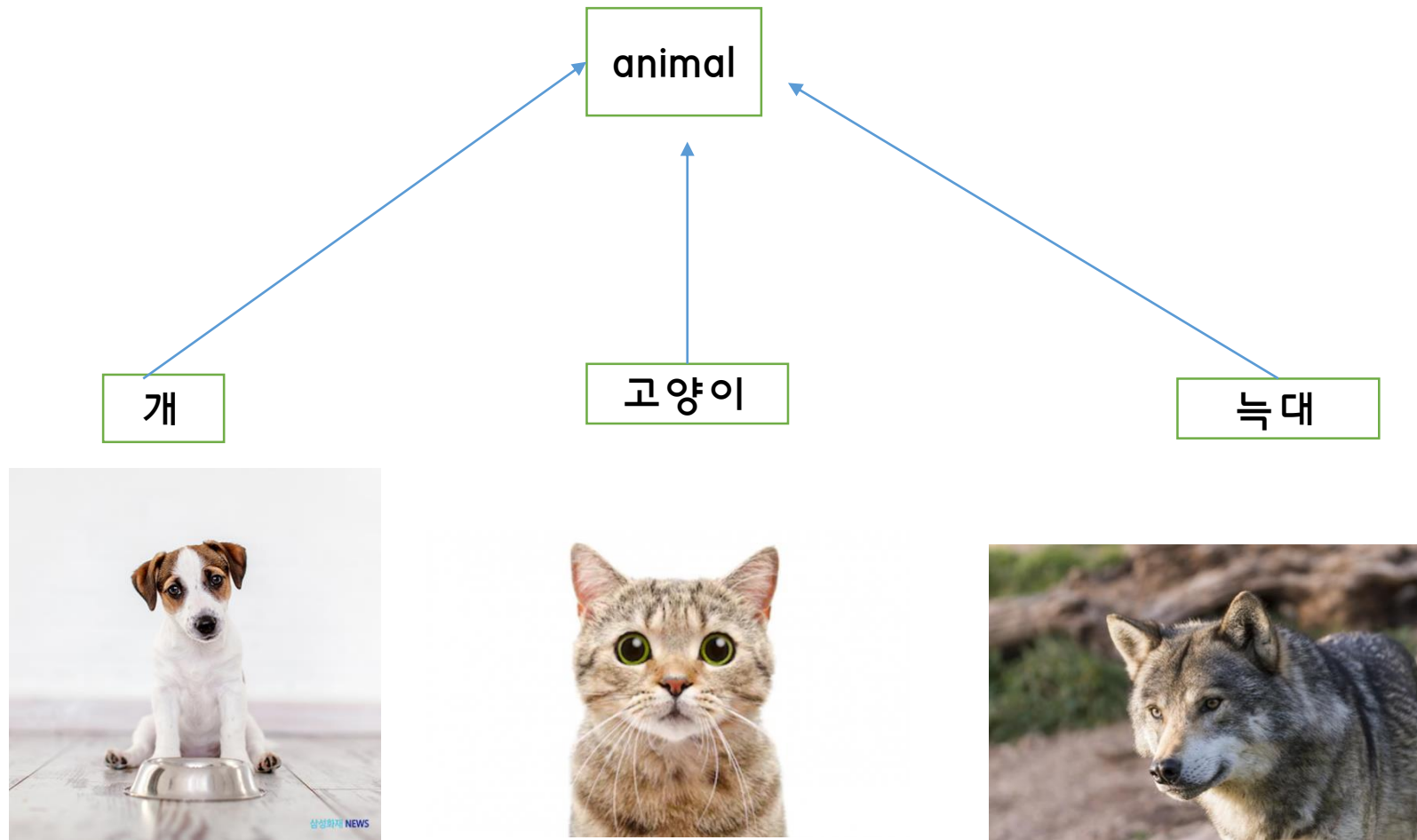
부모형으로 다루게 되면  
부모형으로 의미가 축소된다.

잊지말자  
자료형

크기 / 해석방법

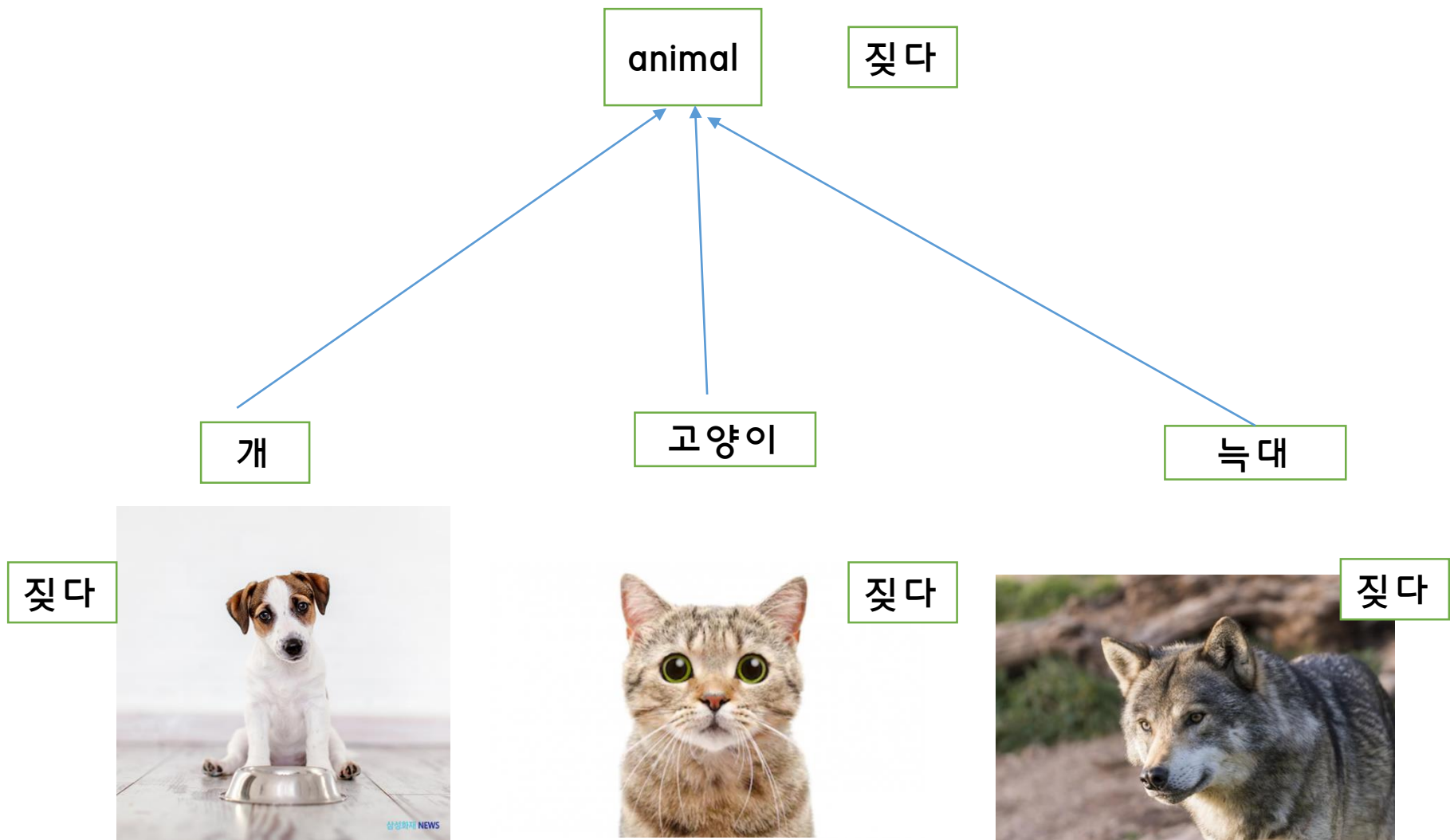
상속관계에서  
타입변환이 가능하다

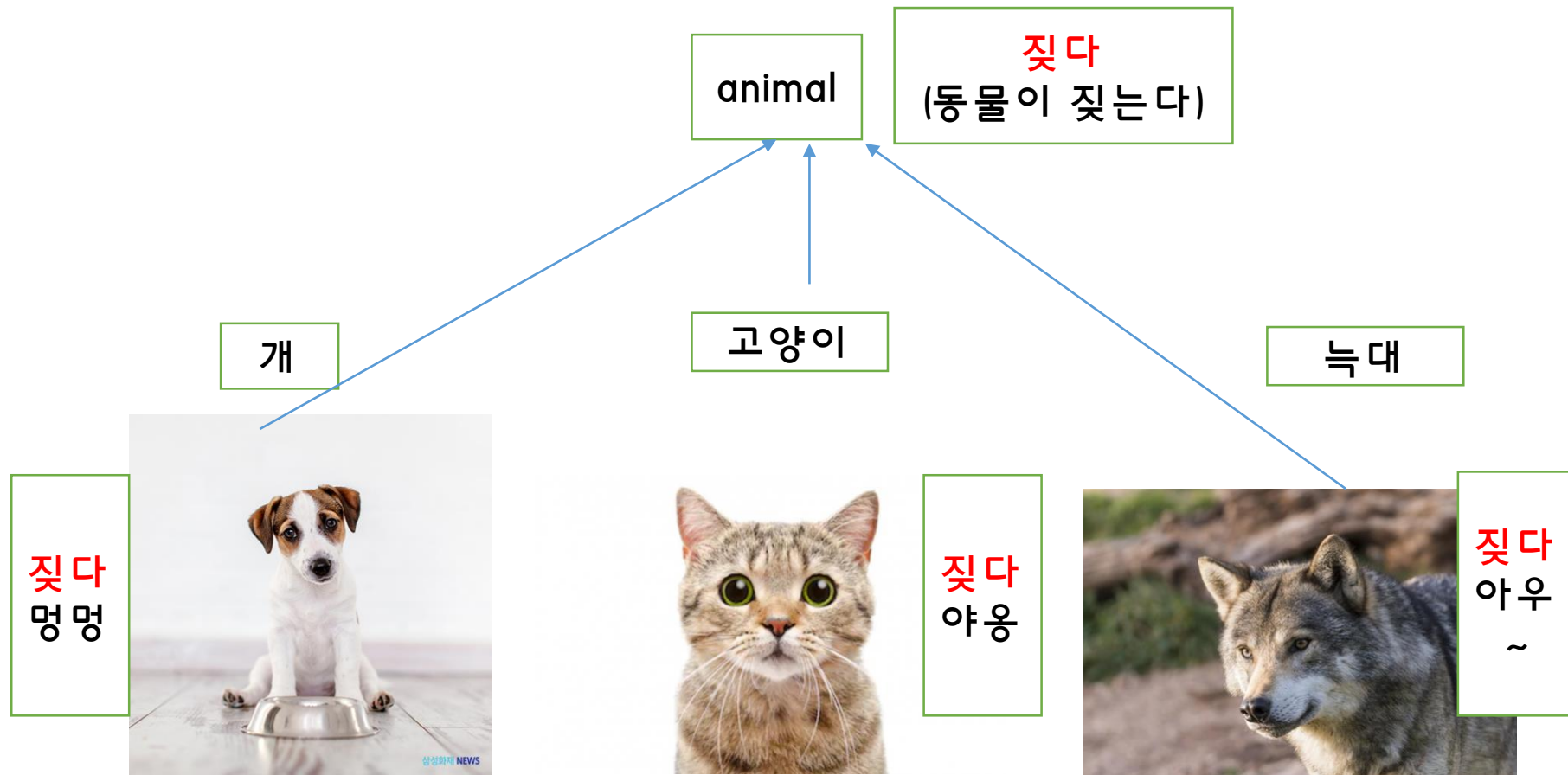




그래서 ?

어떤 잇점?





각 각에 맞게 오버라이딩 합니다



One message  
Multiple implements

하나의 메세지

동물 짖다

다형성



다양한 구현

짖다

야옹~



짖다

아우~



짖다

멍멍

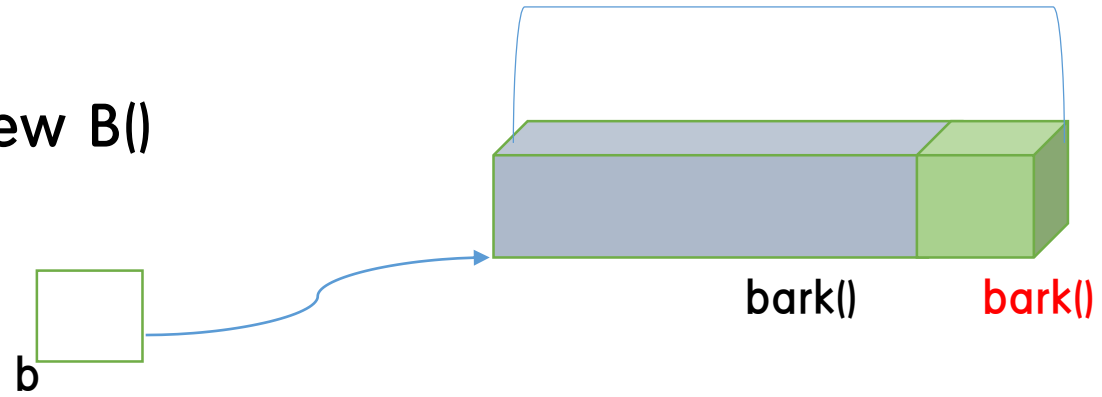
다른 무엇보다 더 중요한,  
최우선시 되는

## overriding

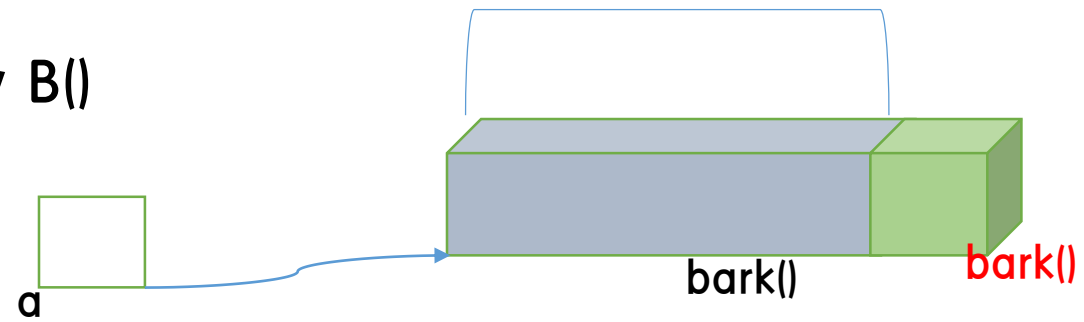
상속시 super 클래스 와 sub 클래스에 똑같은 메소드가 존재하는 것  
( 상속관계에서 부모의 메소드를 자식이 다시 재정의 하는 것)  
: **overriding된 메소드가 우선순위를 갖는다.**

상속받으면서 부모의 메서드를 재정의 하는것을 말한다.

**B** b= new B()



**A** a= new B()



# 오버라이딩

부모의 메서드를 자식이 재정의 했다면  
자식의 메서드가 실행된다

**A** a = new B()

a.disp();

a

disp( "A" )

disp( "B" )

누구의 disp()가  
호출될 것인가?

```
class Animal {  
  
    public void bark() { System.out.println("동물이 짖는다");}  
}
```

```
class Cat extends Animal {  
  
    public void bark()  
    {  
        System.out.println("야옹");  
    }  
}
```

```
class Dog extends Animal {  
  
    public void bark()  
    {  
        System.out.println("멍멍");  
    }  
}
```

```
public class Test{  
    public static void main( String[] args) {  
        Cat cat = new Cat();  
        cat.bark();    //야옹~  
        Dog dog = new Dog();  
        dog.bark();    // 멍멍  
        Animal a = cat;  
        a.bark();    // 야옹~  
        a = dog;  
        a.bark();    // 멍멍!  
    }  
}
```



# 오버라이딩 규칙

**부모의 메소드와 동일한 완벽히 일치**

리턴형 ,메소드명, 매개변수개수 자료형 완벽 일치

**부모의 접근지정자보다 좁혀  
오버라이딩 할 수 없다**

final, static, private은 오버라이딩 안됨

부모형으로 참조하더라도  
 오버라이딩 된 것은

실제 참조하는 객체의  
메소드가 호출되는 것

이게 되지 않으면 부모형으로  
받는것이 의미가 없다



1. 상속관계에서 생성자 호출 순서

2. 상속관계에서 접근제어자 protected

```
public class Animal {  
    public Animal() {    System.out.println( "Animal 생성자" ); }  
    public void bark() { System.out.println( "동물이 짖는다" ); }  
}
```

```
public class Cat extends Animal {  
  
    public Cat() {System.out.println("cat 생성자");}  
    public void bark()  
    {  
        System.out.println("야옹");  
    }  
}
```

```
public class Dog extends Animal{  
  
    public Dog() {System.out.println("dog 생성자");}  
    public void bark()  
    {  
        System.out.println("멍멍");  
    }  
}
```

```
public class Wolf extends Animal{

    public Wolf() {
        System.out.println("늑대 생성자");
    }
    public void bike()
    {
        System.out.println("아우 ~");
    }
}
```

```
public class Poly {  
    public static void main(String[] args) {  
        Cat c = new Cat();  
        Dog d = new Dog();  
        Wolf w = new Wolf();  
        c.bark();  
        d.bark();  
        w.bark();  
  
        Animal a1= new Cat() ; a1.bark();  
        Animal a2= new Dog(); a2.bark();  
        Animal a3= new Wolf; a3.bark();  
  
        // 아래배열로 처리가능함  
        Animal[] arr = new Animal[3];  
        arr[0]= new Cat();  
        arr[1] = new Dog();  
        arr[2] = new Wolf();  
  
        for(int i=0 ; i< arr.length ;i++)  
            arr[i].bark();  
    }  
}
```



**Animal형 배열로  
관리할 수 있다.**

상속 관계에서  
오버라이딩  
예시

```
public class Car {  
  
    protected String name;  
    protected int velocity;  
  
    public Car() {}  
    public Car(String name, int velocity)  
    {  
        this.name = name;  
        this.velocity= velocity;  
  
    }  
  
    public void speedUp() {  
        velocity+=1;  
    }  
  
    public void disp() {  
        System.out.println("자동차 입니다.");  
    }  
  
}
```

```
public class K5 extends Car{

    public void speedUp()
    {
        velocity +=5;
    }
    public void disp()
    {
        System.out.println("K5 입니다. 속도=" + velocity);
    }
}
```

```
public class SantaFeCar extends Car {

    public void speedUp()
    {
        velocity +=10;
    }
    public void disp()
    {
        System.out.println("산타페 입니다. 속도=" + velocity );
    }
}
```

```
public class CarMain {  
    public static void main(String[] args) {  
        /*Car c = new SantaFeCar();  
        c.speedUp();  
        c.disp();  
  
        Car c1= new K5();  
        c1.speedUp();  
        c1.disp();  
        */  
        //다형성  
        Car[ ] c = new Car[2];  
        c[0]= new SantaFeCar();  
        c[1]= new K5();  
  
        for(int i=0; i<c.length;i++ )  
        {  
            c[i].speedUp();  
            c[i].disp();  
        }  
    }  
}
```

# 다형성 예제 속제

과일

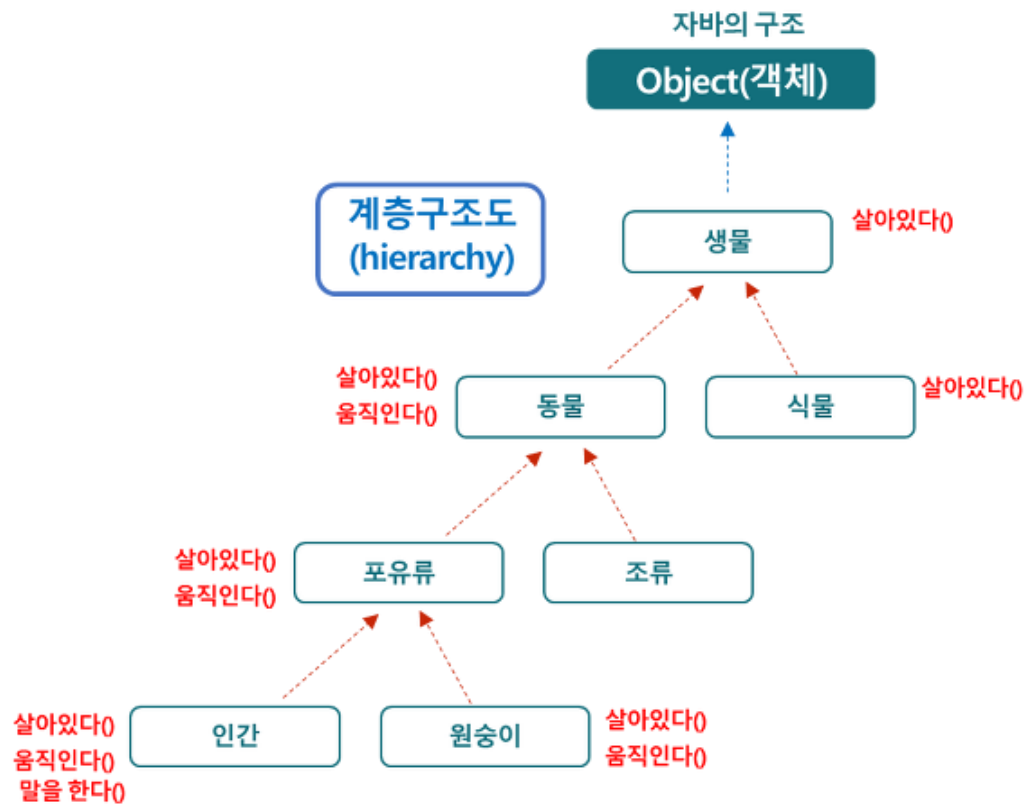
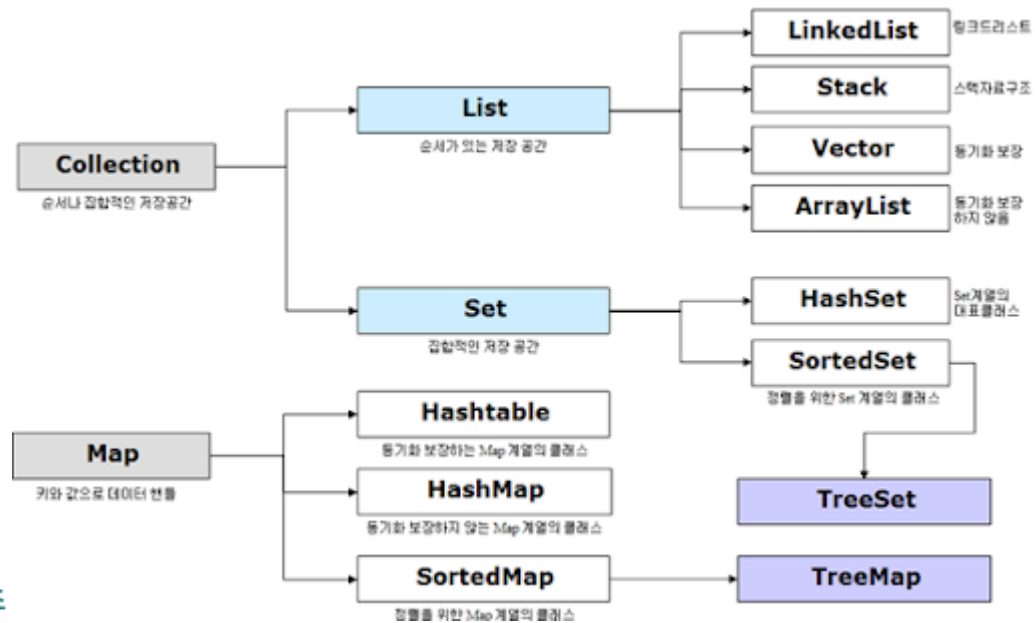
동물

고기



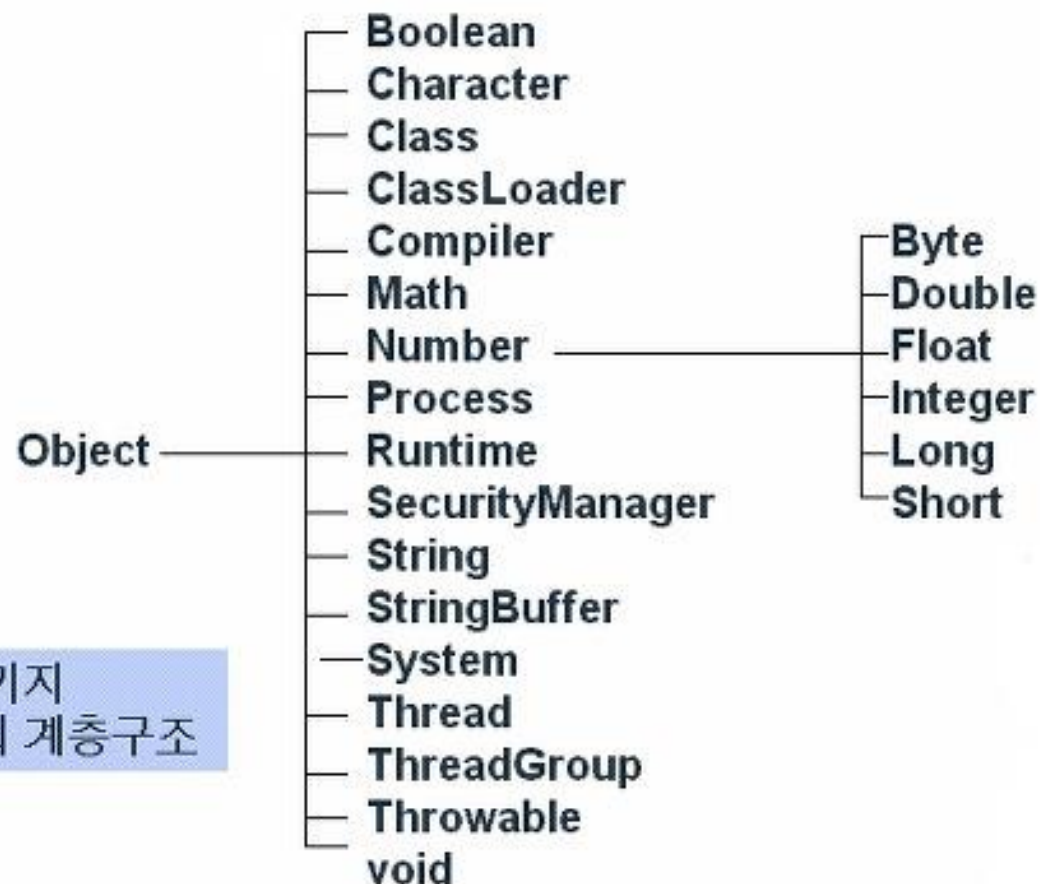
# Object

모든 클래스에 강제로  
상속받는다.  
최상위 클래스



## 11.1 java.lang 패키지의 개요

✓ 모든 자바 프로그램에 자동으로 포함되는 패키지














java.lang 패키지  
주요 클래스의 계층구조

## 주요메서드

String toString()  
boolean equals(Object)

```
Object obj = new Object();
```

- ✓  omok
- ✓  prj1
  -  JRE System Library [JavaSE-1.8]
  - ✓  src
    - ✓  child
      -  Cat.java
      -  Dog.java
    - ✓  parent
      -  Animal.java
      -  AnimalTest.java
  - ✓  test

```
package parent;
import child.Cat;
import child.Dog;
```

```
public class AnimalTest {
    public static void main(String[] args) {
        Cat c = new Cat();
        Dog d = new Dog();
        c.bark();
        d.bark();
        //////////////////////////////////
        Animal[] animals = new Animal[2];
        animals[0]= new Cat();
        animals[1]= new Dog();

        for( Animal animal : animals)
        {
            animal.bark();
            //System.out.println(animal.toString());
            System.out.println(animal);
        }
    }
}
```

```
package child;  
import parent.*;
```

```
public class Cat extends Animal {  
    public void bark() {  
        System.out.println("야옹");  
    }  
  
    public String toString() {  
        return "야옹이 입니다.";  
    }  
}
```

```
package child;  
import parent.*;  
public class Dog extends Animal {  
    public void bark()  
    {  
        System.out.println("멍멍");  
    }  
  
    public String toString() {  
        return "고양이 입니다.";  
    }  
}
```

```
package parent;  
public class Animal {  
  
    public void bark()  
    {  
        System.out.println("동물이 짖는다");  
    }  
  
}
```



Object 클래스 멤버 오버라이딩

```
public String toString()
```

```
public boolean equals(Object obj)
```

## String toString()

```
class Point2{
    int x,y;
    public Point2(int x, int y)
    {
        this.x =x;
        this.y = y;

    }
    //오버라이딩 (오버라이딩 하지 않으면 기본적인 값 출력)
    public String toString() {
        return "Point(" +x + "," + y + ")";
    }
}
```

```
public class ToStringTest {  
    public static void main(String[] args) {  
        Point2 p = new Point2(2,3);  
        System.out.println(p.toString());  
        System.out.println(p);  
    }  
}
```

```
class Point{
    int x,y;
    public Point(int x, int y) {
        this.x =x;
        this.y=y;
    }
    // equals 오버라이딩
    public boolean equals(Object obj)
    {
        Point p= (Point)obj;
        if(x == p.x && y== p.y)return true;
        else return false;
    }
}
```

```
public class EqualsEx {  
  
    public static void main(String[] args) {  
  
        Point a = new Point(2,3);  
        Point b = new Point(2,3);  
        Point c = new Point(3,4);  
  
        if(a==b)  
            System.out.println("a==b");  
        if(a.equals(b))  
            System.out.println("a is equal to b");  
        if(a.equals(c))  
            System.out.println("a is equal to c");  
  
    }  
}
```

# 부모형 참조변수로 받을 수 있다

그러나

```
String str="AWESOMW";  
Object obj;  
obj =str; //업캐스팅 ok
```

```
String str1;  
str1= (String)obj; //다운캐스팅
```



Obj를 통해서는 string객체의 메서드를 사용할 수 없다