다형성

다형성

- **다형성(Polymorphism)**

여러가지의 형태를 가질 수 있는 개념
(오버로딩, 오버라이딩, ...)

1. 코드의 재사용성을 높이고, 중복을 최소화
2. 계층적인 구조를 통해 객체 간의 관계를 나타낼 수 있음

- **오버라이딩 vs 오버로딩**

오버라이딩
→ 부모 클래스에서 상속받은 메소드를 재정의 하는 행위

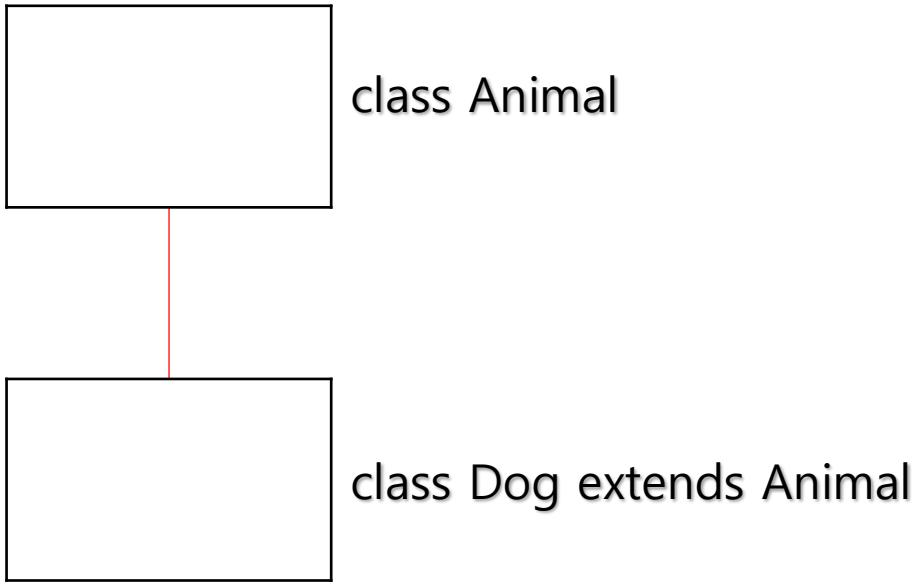
오버로딩
→ 같은 클래스 내에서 메소드의 매개변수 타입/위치를 종류별로 만드는 행위

다형성을 통한 객체 참조

- **다형성을 통한 객체 참조**

객체를 생성할 때 부모 클래스의 타입으로 객체 생성이 가능
이를 통해 동일한 타입(부모 클래스)을 가진 여러가지 하위 클래스 객체를 생성할 수 있음

```
기존에 객체를 생성하던 방식
Cat cat = new Cat("Whiskers", 5);
cat.eat();
cat.sleep();
cat.meow();

다형성을 통한 객체 참조
Animal a = new Dog("강아지",3);
a.eat();
```

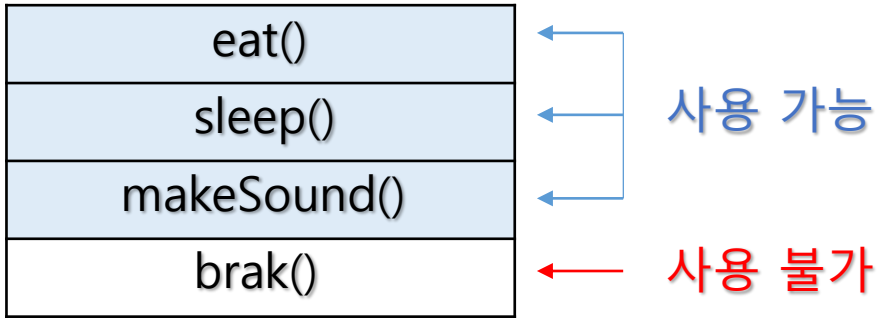


다형성을 통한 객체 참조

- **다형성을 통한 객체 참조**

부모 클래스가 가지고 있는 필드를 활용할 수 있으나, 내가(Dog) 가진 필드는
일반적인 방법으로 사용할 수 없음

```
다형성을 통한 객체 참조
Animal a = new Dog("강아지",3);
a.eat();
```



다형성을 통한 객체 참조

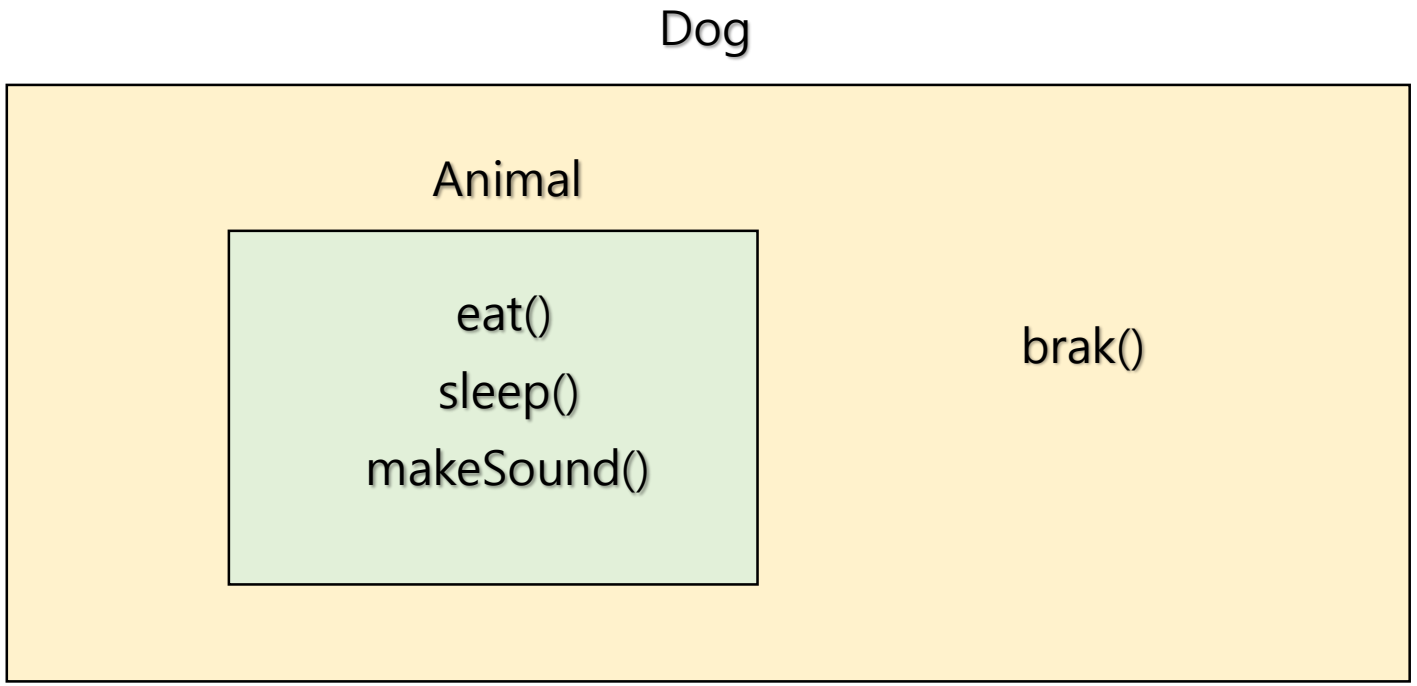
- 참고

1. 부모클래스 변수명 = new 자식클래스();
   → 가능
2. 자식클래스 변수명 = new 부모클래스();
   → 불가

다형성을 통한 객체 참조

- **다형성을 통한 객체 참조**

Animal 클래스 : eat(), sleep(), makeSound()
Dog 클래스 : brak(), eat(), sleep(), makeSound()



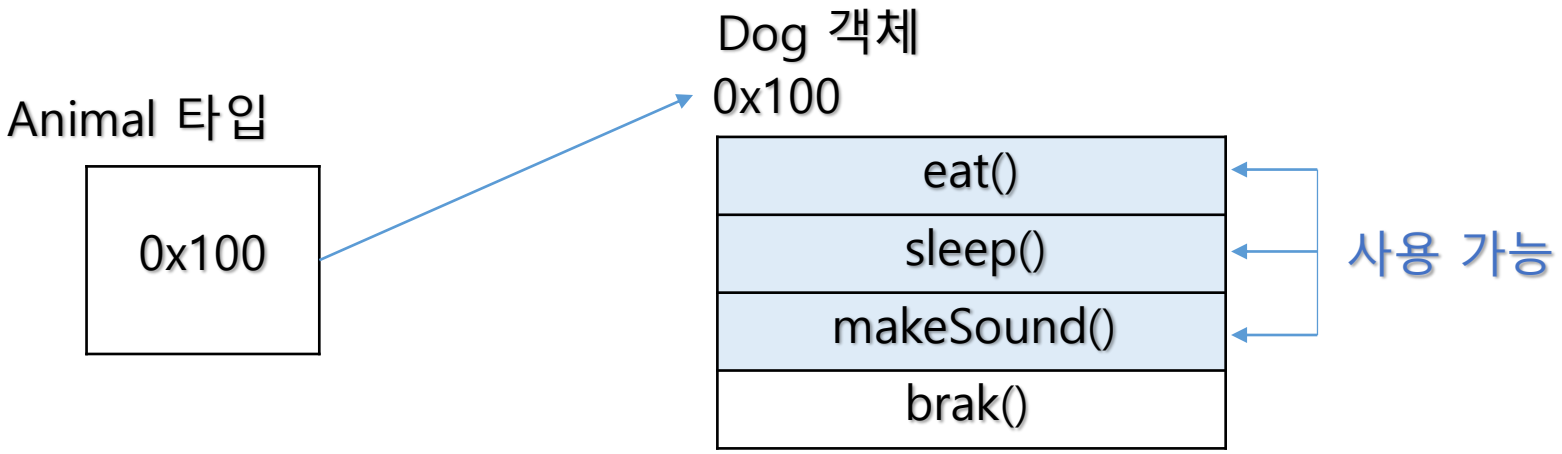
다형성을 통한 객체 참조

- **다형성을 통한 객체 참조**

Animal 클래스 : eat(), sleep(), makeSound()
Dog 클래스 : brak(), eat(), sleep(), makeSound()

Animal 타입은 3개의 메소드만 알고 있기 때문에 3개만 사용 가능

```
다형성을 통한 객체 참조
Animal a = new Dog("강아지",3);
a.eat();
```



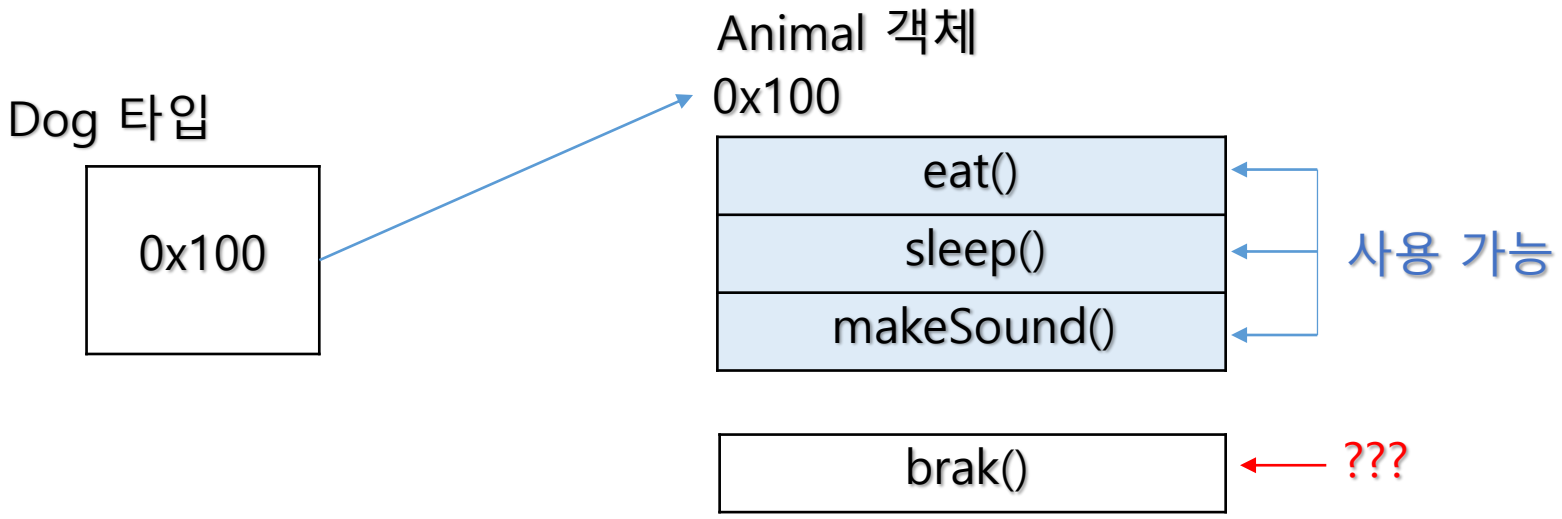
다형성을 통한 객체 참조

- **다형성을 통한 객체 참조**

Animal 클래스 : eat(), sleep(), makeSound()
Dog 클래스 : brak(), eat(), sleep(), makeSound()

Dog 타입은 4개의 메소드를 알고 있는데, Animal 객체는 3개의 메소드만 알고 있으므로 사용 불가

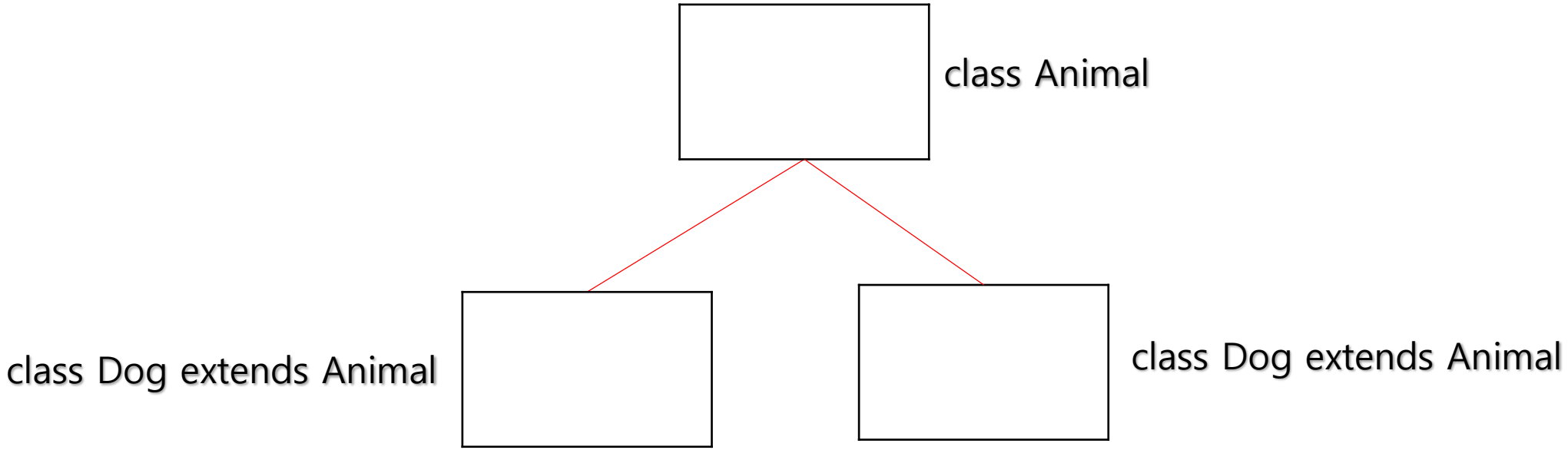
```
Dog d = new Animal();
d.eat();
```



다형성을 통한 객체 참조

- **사용하는 이유 - 1**

1. 일관된 방식으로 처리 가능
   → 여러 종류의 동물을 다루는 프로그램에서 Animal 타입으로 다양한 동물들의 객체를 동일한 메서드를 활용해 다양한 동물을 관리할 수 있음
   → 코드의 가독성과 유지보수성 향상



다형성을 통한 객체 참조

- **사용하는 이유 - 2**

1. 다형성을 활용해 매개변수로 여러 개의 자식 클래스를 컨트롤할 수 있음
   → 참조 매개변수 이므로 메모리 주소값을 전달받고 해당 메모리 주소에 있는 객체(Dog, Cat)을 찾아감
   → 부모와 자식이 있을 때 오버라이딩된 메소드가 있을 경우 자식의 우선권이 높기 때문에 오버라이딩된 makeSound() 메소드를 출력
   → 코드의 재사용성과 확장성을 높임

```java
public class Depoly {
    public static void main(String[] args) {
        Zoo zoo = new Zoo();

        Animal dog = new Dog();
        zoo.performSound(dog);  // "멍멍!" 출력

        Animal cat = new Cat();
        zoo.performSound(cat);  // "야옹~" 출력
    }
}

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

class Dog extends Animal {
    @Override
    public void makeSound() {
        System.out.println("멍멍!");
    }
}

class Cat extends Animal {
    @Override
    public void makeSound() {
        System.out.println("야옹~");
    }
}

class Zoo {
    public void performSound(Animal animal) {
        animal.makeSound();
    }
}
```

다형성을 통한 객체 참조

- **사용하는 이유 - 3**

1. 다형성을 활용해 다형적 컬렉션 사용 가능
   → Dog, Cat 등의 객체를 ArrayList<Animal>에 저장하면
   여러가지 동물 객체를 다룰 수 있는 컬렉션을
   생성하고 관리할 수 있음

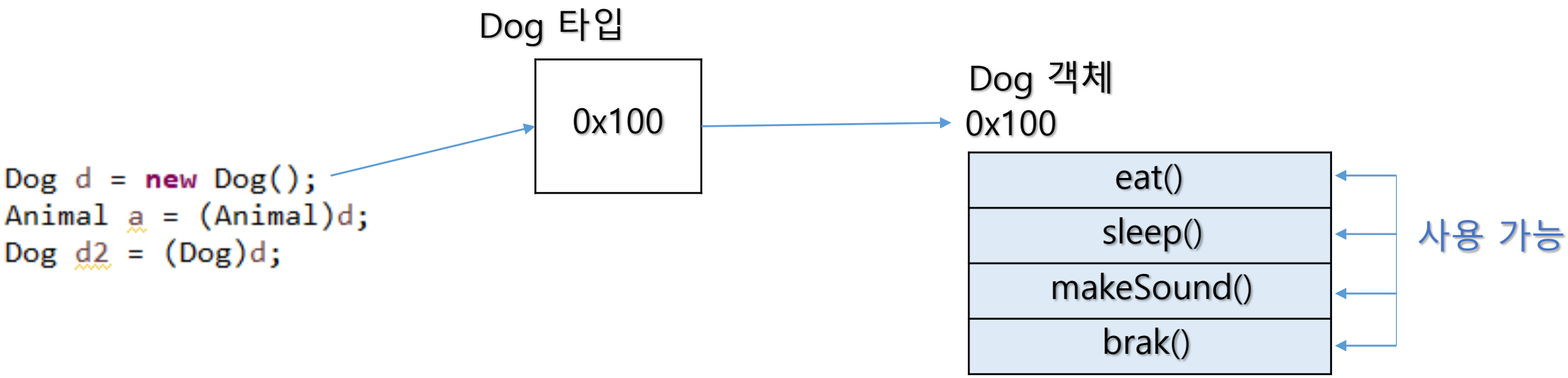
```java
public class Depoly {
    public static void main(String[] args) {
        ArrayList<Animal> animals = new ArrayList<>();

        animals.add(new Dog("강아지", 3));
        animals.add(new Cat("고양이", 5));
        animals.add(new Dog("슈나우저", 2));

        for (Animal animal : animals) {
            animal.makeSound();
        }
    }
}

class Animal {
    public String name;
    public int age;

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

    public void makeSound() {
        System.out.println(name + "이 소리를 내고 있습니다.");
    }
}

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

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

class Cat extends Animal {
    public Cat(String name, int age) {
        super(name, age);
    }

    public void makeSound() {
        System.out.println(name + "이 야옹하고 울부짖습니다.");
    }
}
```

참조변수 형변환

참조변수 형변환 - 1

- **참조변수 형변환**

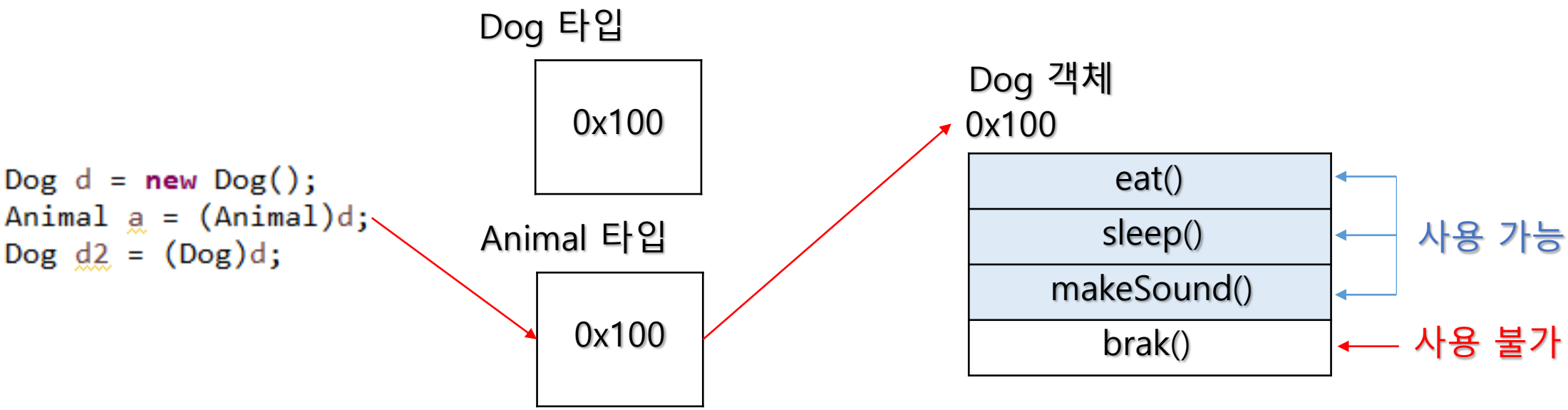
일반적으로 상속 관계에서 형변환(Casting)을 하는 것을 의미



참조변수 형변환 - 2

- **참조변수 형변환**

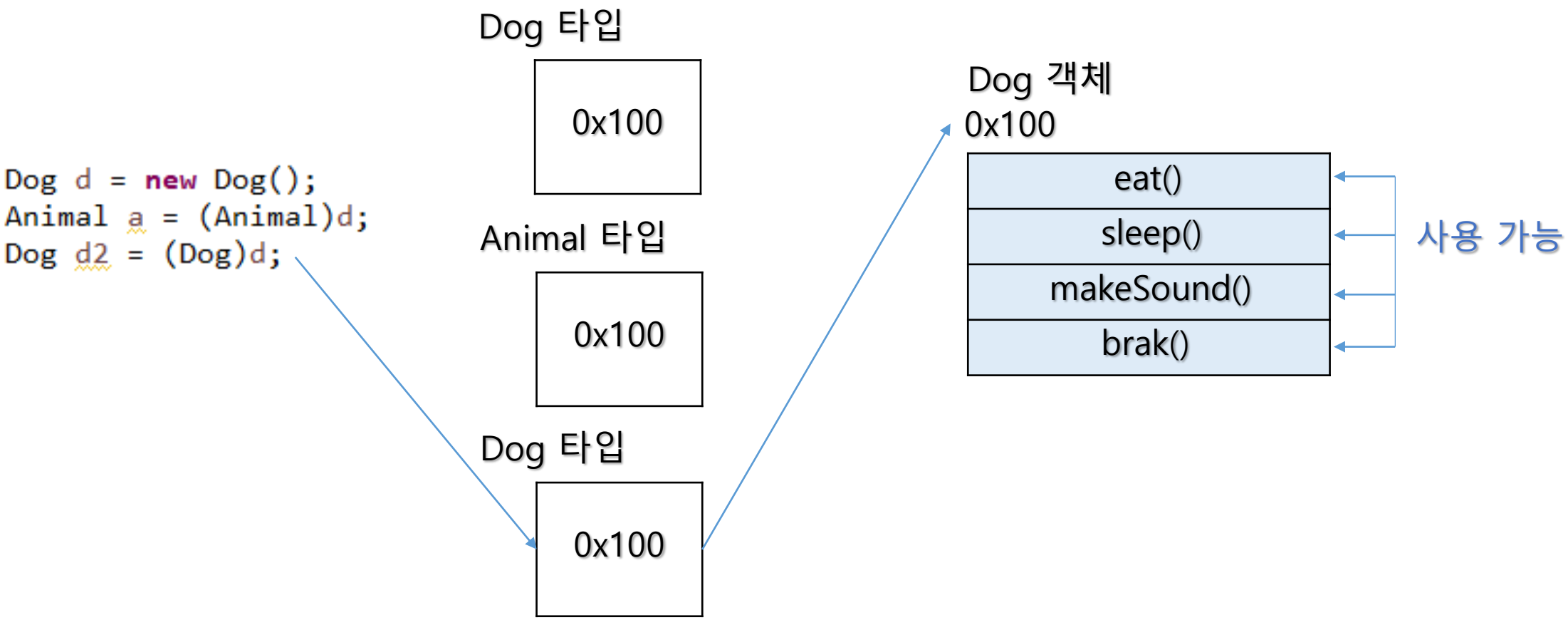
일반적으로 상속 관계에서 형변환(Casting)을 하는 것을 의미



참조변수 형변환 - 3

- **참조변수 형변환**

일반적으로 상속 관계에서 형변환(Casting)을 하는 것을 의미



참조변수 형변환 - 4

- **형변환 가능 여부 확인 (instanceof)**

형변환을 하기 전, 아래와 같이 instanceof로 형변환이 가능한지 먼저 확인해주는 작업이 필요함

```
public static void main(String[] args) {
    Dog d = new Dog();
    QuizExtends.checkCasting(d);
}

public static void checkCasting(Dog d) {
    if(d instanceof Animal) {
        System.out.println("캐스팅 가능!");
    }
}
```

참조변수 형변환 - 5

- **참조변수 형변환이 햇갈린다면?**

1. 자기 자신, 부모 클래스는 형변환 가능
2. 자식 클래스는 형변환 불가
3. 상속관계가 아닌 클래스는 형변환 불가
   → 자바에서는 단일상속을 지원하기 때문에 부모 클래스 외에는 동등 또는 자식 관계
   이므로 "자기 자신과 부모 클래스 외에는 형변환이 불가능하다"라고 외워도 됨