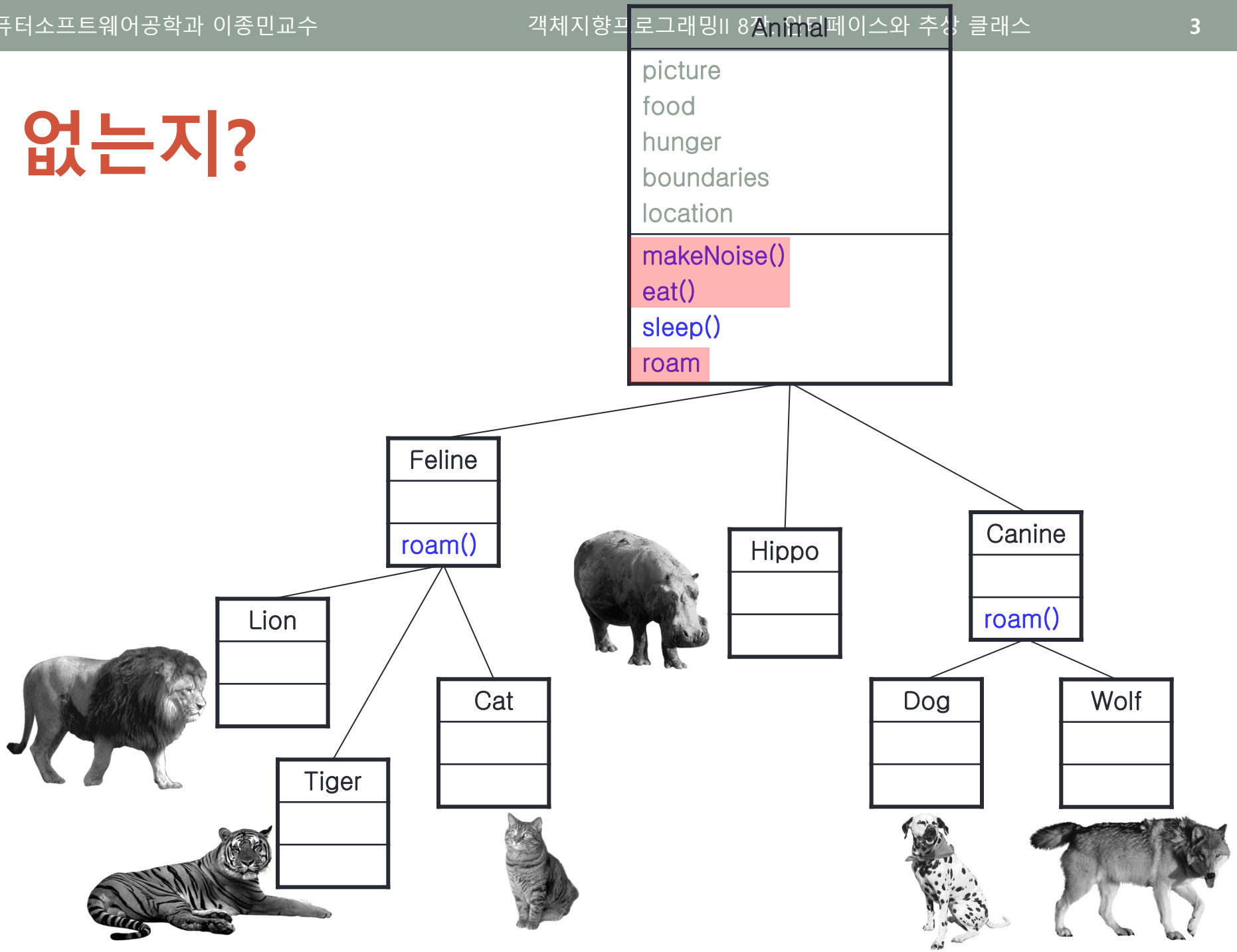
8장. 심각한 다형성

"추상(abstract)"과 "구상(concrete)"의 차이점
java.lang.Object 클래스
다형성
캐스팅
인터페이스

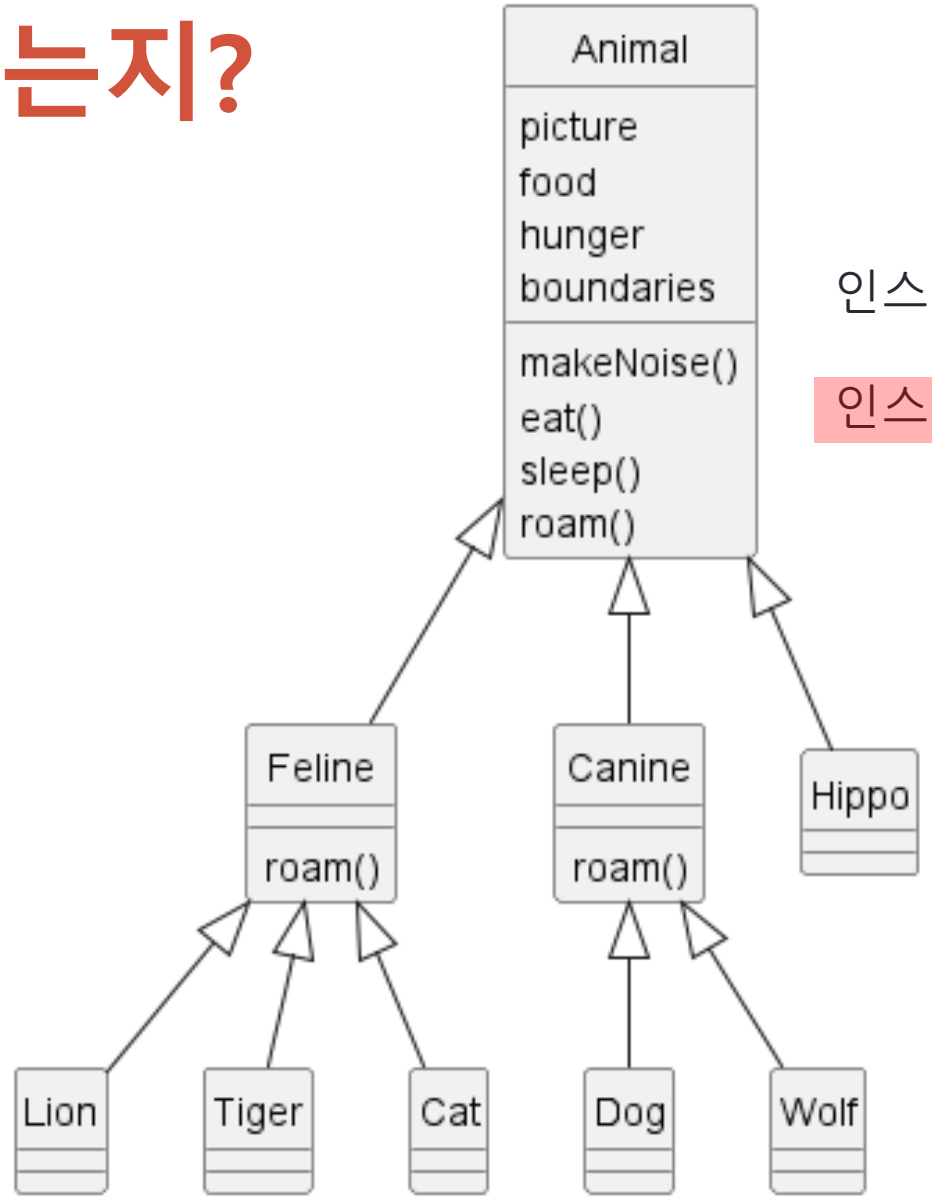
인터페이스와 추상 클래스

문제 없는지?


Animal
picture
food
hunger
boundaries
location
makeNoise()
eat()
sleep()
roam
Feline
roam()
Lion
Tiger
Cat
Hippo
Canine
roam()
Dog
Wolf

p.242

문제 없는지?

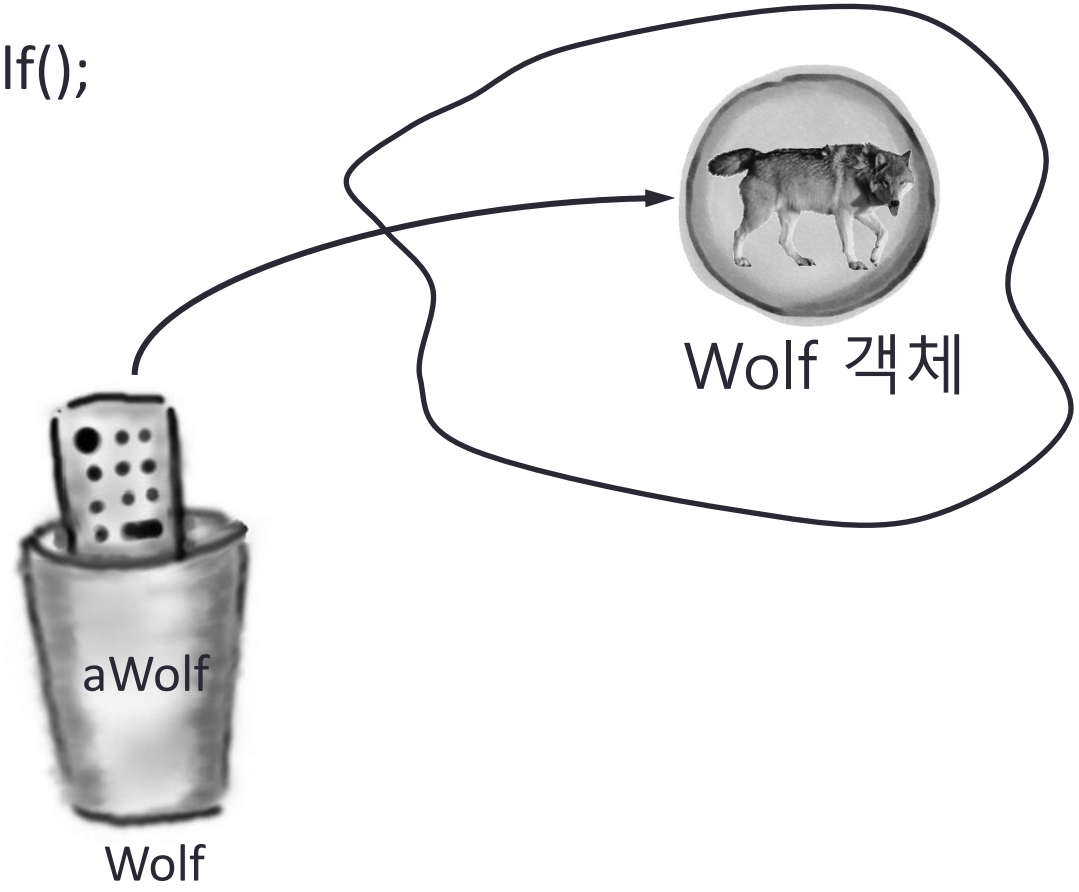


인스턴스가 필요한 것: Lion, Tiger, cat, Dog, Wolf, Hippo

인스턴스가 필요없는 것: Animal, Feline, Canine

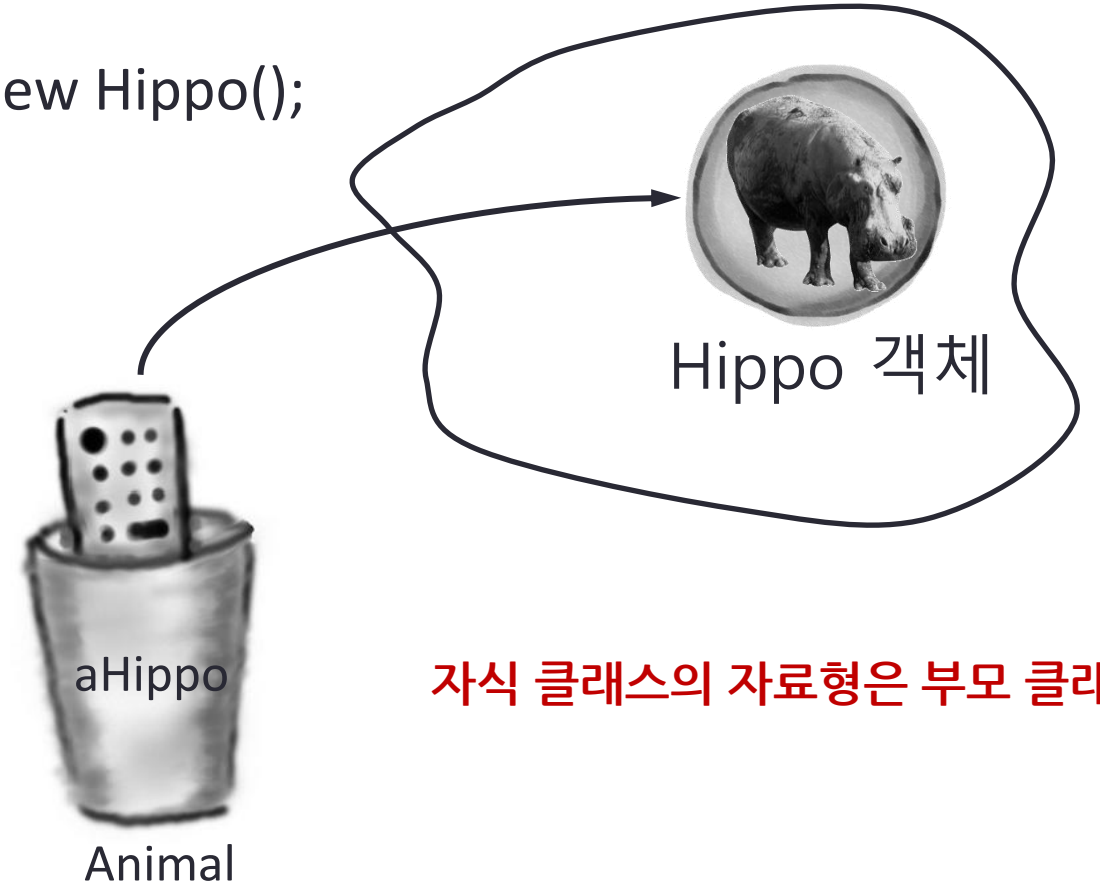
Wolf 객체에 대한 Wolf 레퍼런스

Wolf aWolf = new Wolf();



Hippo 객체에 대한 Animal 레퍼런스

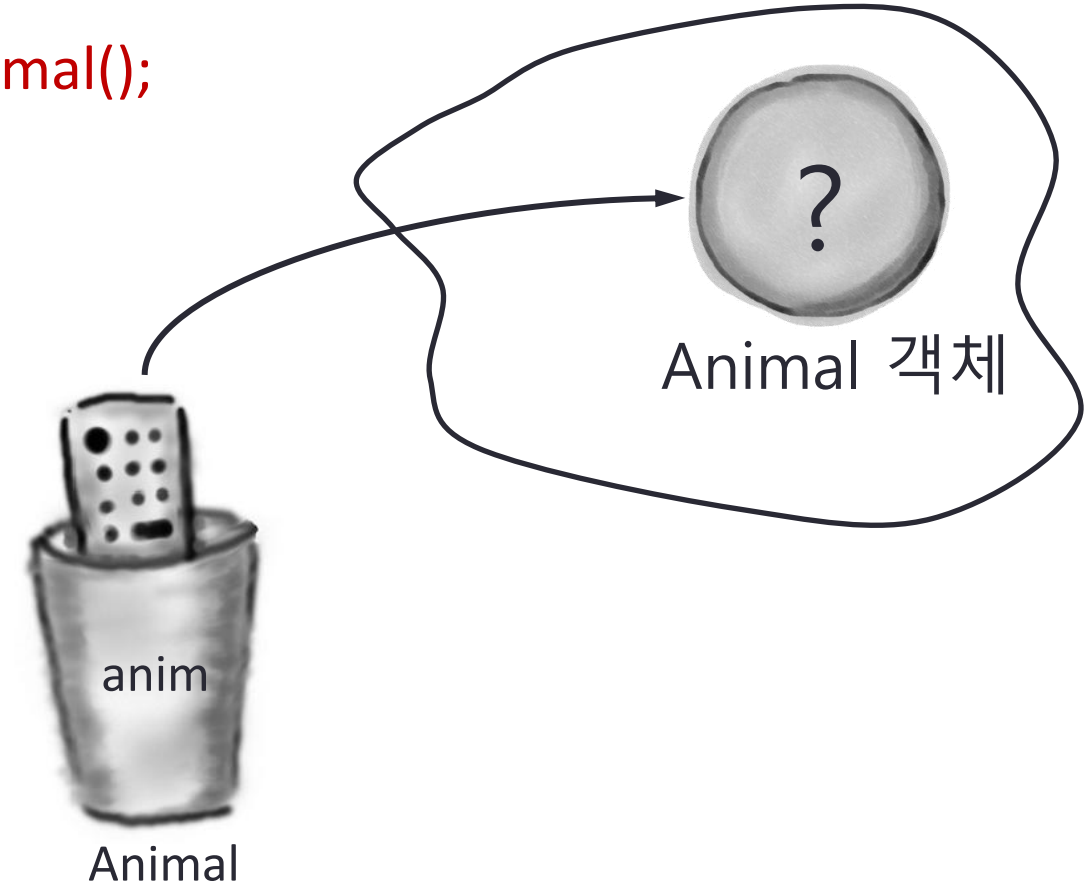
Animal aHippo = new Hippo();



자식 클래스의 자료형은 부모 클래스로 대체 가능 (LSP)

Animal 객체에 대한 Animal 레퍼런스 ?

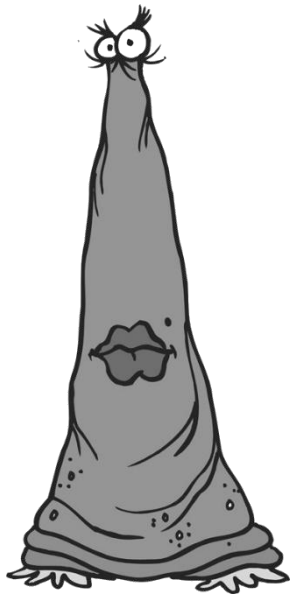
Animal anim = new Animal();

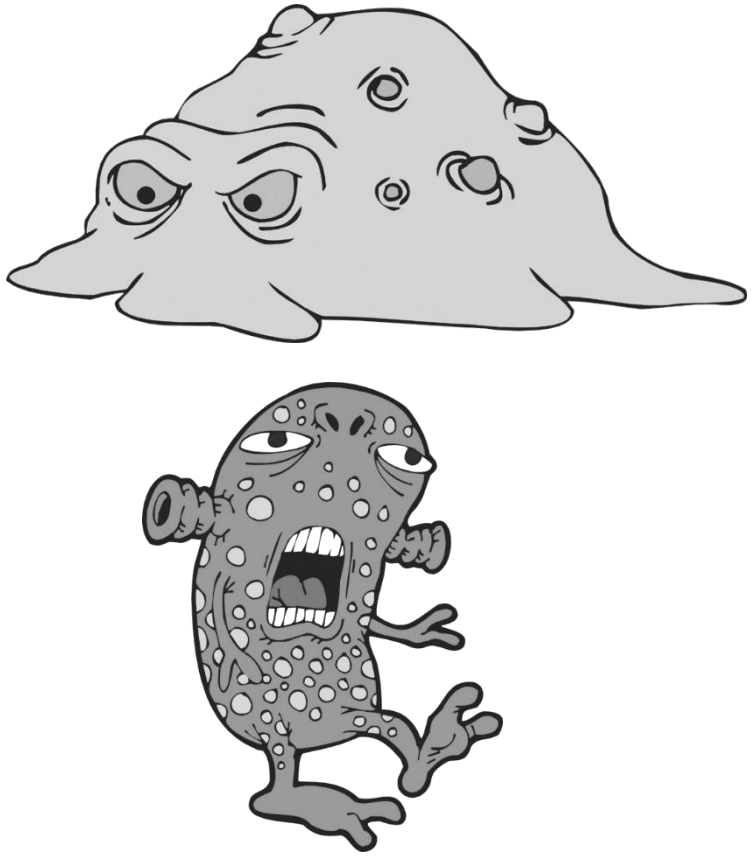


객체는 어떻게 생겼을까?

Animal 객체는 어떻게 생겼을까요?

추상 (abstract) vs. 구상 (concrete)





p.244

추상 클래스

- 추상 클래스(abstract class)란?
  - **인스턴스를 만들 수 없는 클래스**
  - 반드시 확장해야 함:
    하위 클래스에서 추상 메서드를 재정의(overriding)하여 실제 코드를 추가해야 함.
  - 추상 유형을 레퍼런스로 사용할 수는 있음
  - 다형적인 매개변수, 리턴 유형, 배열 등에 활용

```
abstract class Animal {
  abstract public void roam();
}
abstract class Canine extends Animal {
  @Override
    public void roam() { … }  // overriding
}
class Dog extends Canine { ... }  // roam()을 그냥 사용하거나 재정의 가능
```

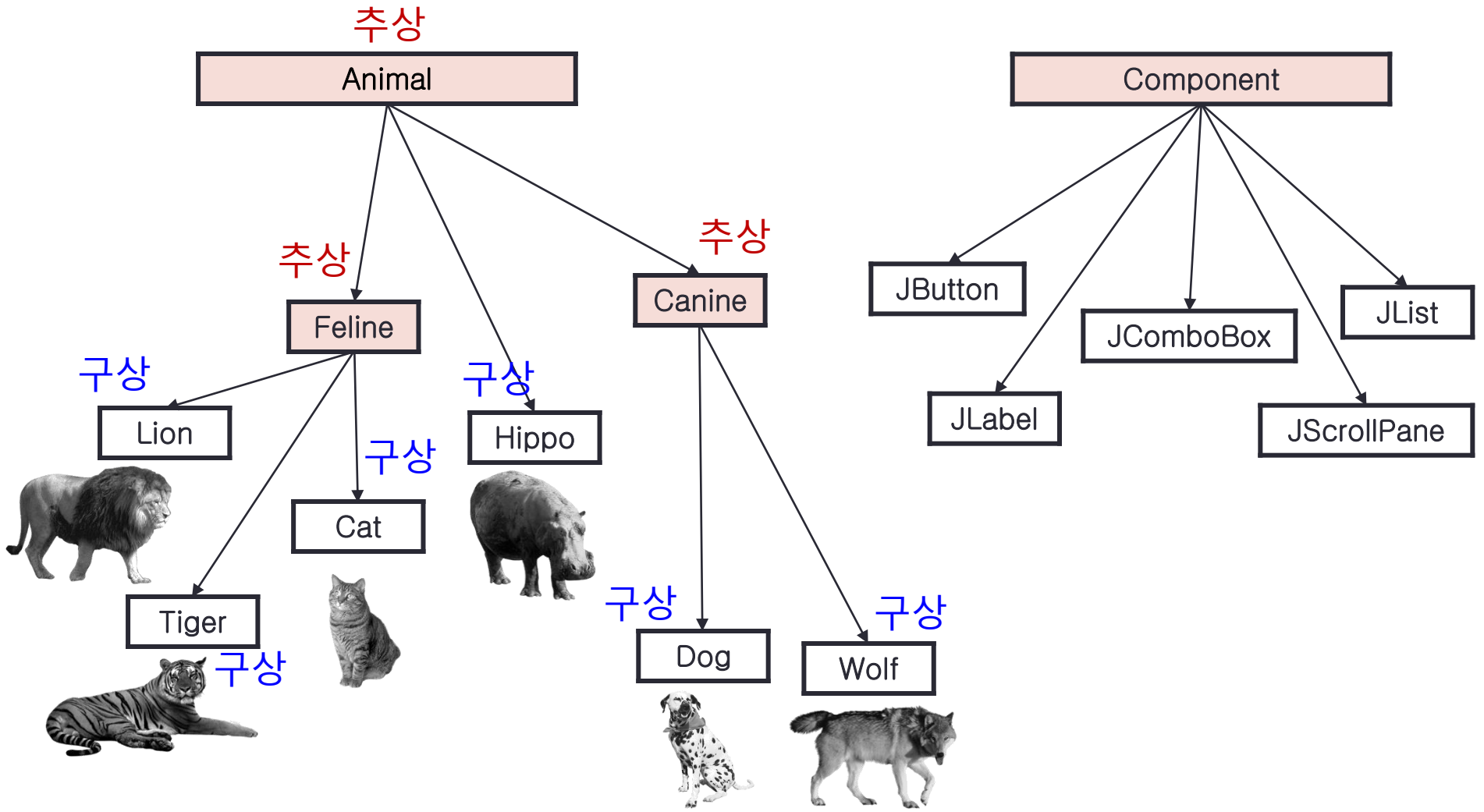
추상 클래스

```
// class를 abstract로 지정하면 더 이상 인스턴스를 만들 수 없음!
public abstract class Canine extends Animal
{
  public void roam() { }
}
```

```
public class MakeCanine {
  public void go() {
    Canine c;
    c = new Dog();
    c = new Canine();
    c.roam();
  }
}
```

```
$ javac MakeCanine.java
MakeCanine.java:5: Canine is abstract;
Cannot be instantiated
                c = new Canine();
                    ^
1 error
```

추상(abstract) vs. 구상(concrete)



p.246

추상 메서드

- 추상 메서드 (abstract method)
  - 몸통이 없는 메서드
  - 반드시 오버라이드해야 합니다.

```
public abstract void eat();
```

  - 추상 메서드를 만들 때는 클래스도 반드시 추상 클래스로 만들어야 합니다.
  - 추상 클래스가 아닌 (구상) 클래스에는 추상 메서드를 집어넣을 수 없습니다.

- 하나 이상의 추상 메서드가 있는 클래스 → **추상 클래스**
- 모든 메서드가 추상이면 **인터페이스**!!!

p.247

인터페이스와 추상 메서드 보충 설명

- **인터페이스 (Interface)**: 클래스가 구현해야 할 **메서드의 집합**을 정의하는 구조
- **추상 메서드**: 구현이 없는 메서드로, 인터페이스에 선언되면 해당 인터페이스를 구현하는 클래스에서 반드시 구현해야 함.

```
interface Animal {
    void makeSound(); // 추상 메서드
}

class Dog implements Animal {
    @Override
    public void makeSound() { … }
}
```

Java 8 이후의 변화

- Java 8부터 인터페이스에 구현이 포함된 메서드를 정의할 수 있게 되었는데, 이를 default method라고 함.
- 목적
  - 기존 인터페이스를 **호환성 깨지지 않게 확장**하기 위해 도입됨.
  - 인터페이스를 구현한 클래스에 **기본 동작을 제공**할 수 있음.
- sleep() 메서드는 기본 구현을 제공하므로, 구현 클래스에서 선택적으로 오버라이드 가능

```
interface Animal {
    void makeSound(); // 추상 메서드

    default void sleep() {
        System.out.println("Sleeping...");
    }
}
```

인터페이스 vs 추상 클래스

- 주의 사항
  - default method는 상속 충돌이 발생할 수 있습니다. 예를 들어, 두 인터페이스에서 동일한 default 메서드를 정의하면 구현 클래스에서 명시적으로 오버라이드해야 함.
  - default method는 상태를 가질 수 없고, 오직 동작만 정의할 수 있음.

항목	인터페이스	추상 클래스
다중 상속	가능	불가능
필드	상수만 가능	인스턴스 변수 가능
메서드 구현	Java 8부터 `default`, `static` 가능	일부 메서드 구현 가능
생성자	없음	있음
목적	기능 명세	공통 기능 제공

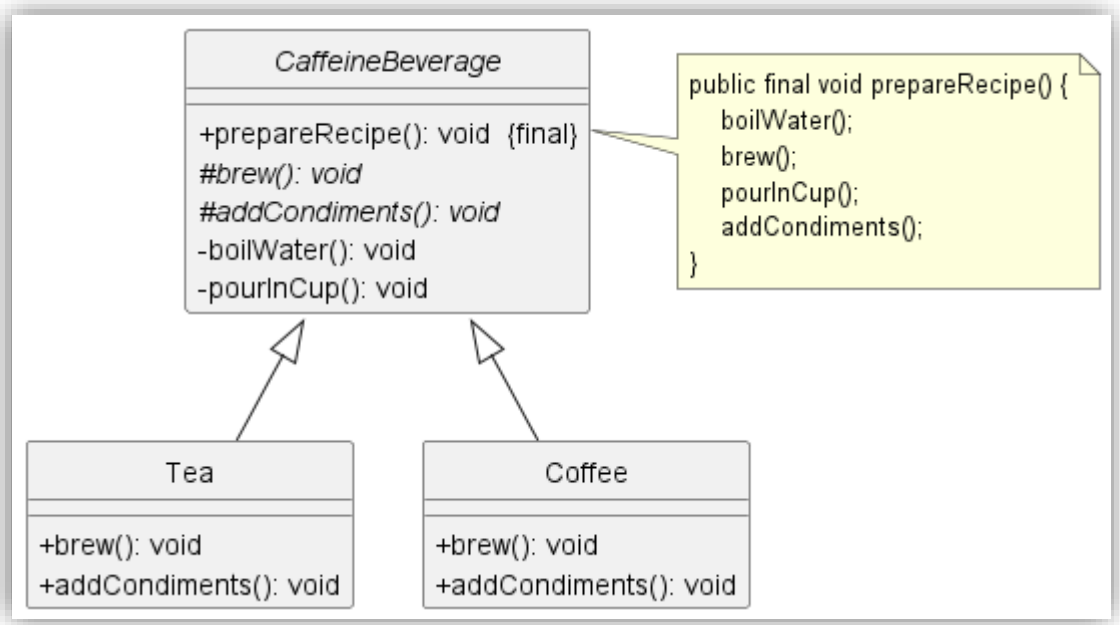
바보 같은 질문은 없습니다.

- 추상 클래스는 하위클래스에서 상속해서 쓸 공통적인 코드를 집어넣기 위해 있는 거라고 볼 수 있지만 **추상 메서드는 왜** 있는 거죠?
  - 추상 클래스에서 **하위 클래스에서 활용할 수 있는 일반적인 메서드 코드를 만들 수 없는 경우**에 추상 메서드를 만듭니다.
  - **하위클래스에 반드시 있어야 하는 일련의 메서드를 정의**하기 위해 추상 메서드가 필요합니다.

p.247

바보 같은 질문은 없습니다.

- 그러면 어떤 장점이 있나요?
  - 다형성이 가장 중요한 장점입니다.
  - 추상 메서드는 다형성을 활용하기 위해 "이 유형에 속하는 모든 하위클래스 유형에는 이 메서드가 있어야 한다"는 것을 지정하기 위해서 반드시 필요합니다.
- 추상 메서드의 활용 예:
  Template Method Pattern!!!



추상 메서드는 모두 구현해야 합니다.

- 추상 메서드 구현 == 메서드 오버라이드
  - 상속 트리에서 처음으로 등장하는 **구상 클래스에서 모든 추상 메서드를 구현**해야만 합니다.
  - 추상 메서드에는 추상 메서드와 구상 메서드가 모두 들어갈 수 있지만 **구상 클래스**에는 추상 메서드가 들어갈 수 없습니다.
  - 메서드 서명(method signature)과 리턴형이 똑같은 (추상 메서드가 아닌 → 구상) 메서드를 만들기만 하면 됩니다. (자바에서는 메서드의 코드가 어떻게 되는지는 신경 쓰지 않습니다.)

p.248

(참고) MethodSignature.java

```java
public class MethodSignature {

    private void add(int e1, int e2) {
        System.out.printf("%d + %d = %d%n", e1, e2, e1 + e2);
    }

    //  error: method add(int,int) is already defined in class MethodSignature
    // method signature는 메서드 이름과 매개변수 목록의 자료형을 의미함. 반환 자료형은 포함되지 않음.
    private int add(int e1, int e2) {
        return e1 + e2;
    }

    private long add(long e1, long e2) {
        return e1 + e2;
    }

    public void run() {
        add(10, 20);
        // add(10L, 20L);
    }

    public static void main(String[] args) {
        new MethodSignature().run();
    }
}
```

```
PS D:₩User₩jongmin₩강의₩강의노트₩2024-2₩2-JAVA₩SW₩LJM추가코드> java .₩MethodSignature.java
.₩MethodSignature.java:11: error: method add(int,int) is already defined in class MethodSignature
    private int add(int e1, int e2) {

1 error
error: compilation failed
```

다형성 활용

- ArrayList 비슷한 클래스를 직접 만듭시다.
  - 배열의 길이는 5로 제한 (Dog 배열로 시작)
  - add() 메서드를 호출하면 새로 받아온 Dog 객체를 추가하고 인덱스(nextIndex)를 증가시킴

```
public class MyDogList {
  private Dog[] dogs = new Dog[5];
  private int nextIndex = 0;

  public void add(Dog d) {
    if (nextIndex < dogs.length) {
      dogs[nextIndex] = d;
      System.out.println("Dog added at " + nextIndex);
      nextIndex++;
    }
  }
}
```

p.250

Cat 객체도 집어넣으려면?

선택 가능한 옵션:

1. Cat 객체를 저장하기 위해 MyCatList라는 클래스를 따로 만듭니다.
2. 서로 다른 두 배열과 서로 다른 두 메서드(addCat(Cat c), addDog(Dog d))가 들어있는 DogAndCatList라는 클래스를 만듭니다.
3. 모든 Animal 하위클래스를 받아들일 수 있는 AnimalList 클래스를 만듭니다.

MyAnimalList

```
public class MyAnimalList {
  private Animal[] animals = new Animal[5];
  private int nextIndex = 0;

  public void add(Animal a) {
    if (nextIndex < animals.length) {
      animals[nextIndex] = a;
      System.out.println("Animal added at " + nextIndex);
      nextIndex++;
    }
  }
}
```

p.251

AnimalTestDrive 클래스

```
public class AnimalTestDrive {
  public static void main(String[] args) {
    MyAnimalList list = new MyAnimalList();
    Dog a = new Dog();
    Cat c = new Cat();
    list.add(a);
    list.add(c);
  }
}
```

```
% java AnimalTestDrive
Animal added at 0
Animal added at 1
```

p.251

Animal 말고 다른 객체는요?

- 가장 포괄적인 클래스는?
  - java.lang.Object 클래스!!!
  - Object 클래스는 모든 클래스의 어머니, 즉 모든 클래스의 상위클래스입니다.
  - 별도로 어떤 클래스를 확장한 것이라고 지정해주지 않으면 자동으로 Object 클래스의 하위클래스가 됩니다.

p.252

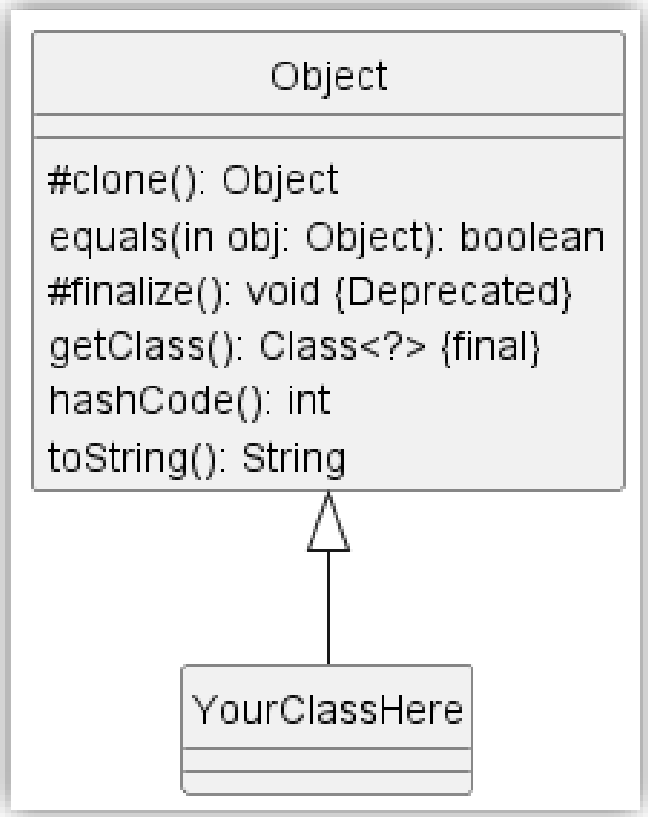
Object 클래스에는 무엇이 있나요?

- equals(Object o)

```
Dog a = new Dog();
Cat c = new Cat();

if (a.equals(c)) {
  System.out.println("true");
} else {
  System.out.println("false");
}
```

```
% java TestObject
false
```



boolean	**equals(Object obj)** Indicates whether some other object is "equal to" this one.

Object 클래스의 메서드

- getClass()

```
% java TestObject
class cse.oop2.ch08.TestObject$Cat
```

```
Cat c = new Cat();
System.out.println(c.getClass());
```

Class<?>	getClass() Returns the runtime class of this Object.

- hashCode()

```
% java TestObject
8202111
```

Cat c = new Cat();

System.out.println(c.hashCode());

int	**hashCode()** Returns a hash code value for the object.

- toString()

```
Cat c = new Cat();
System.out.println(c.toString());
```

```
% java TestObject
Cat@7d277f
```

String	toString() Returns a string representation of the object.

바보 같은 질문은 없습니다.

- Object 클래스는 추상 클래스인가요?
  - 아닙니다. 모든 클래스에서 무조건 오버라이드할 필요 없이 그대로 사용할 수 있는 메서드를 구현해 놓은 코드가 들어있기 때문에 추상 클래스가 아닙니다.

p.254

바보 같은 질문은 없습니다.

- 그러면 Object에 들어있는 메서드를 오버라이드할 수는 있나요?
  - 할 수 있는 것도 있지만 final로 지정되어있어서 오버라이드할 수 없는 것도 있습니다.
  - 될 수 있으면 **hashCode(), equals(), toString()** 메서드는 오버라이드하는 것이 좋습니다.
  - getClass 같은 메서드는 반드시 특정한 방식으로 작동을 해야 하기 때문에 final로 지정되어 있습니다. 그리고 고유 코드(native code)로 만들어진 메서드도 있습니다.

```
public final Class<?> getClass()
public int hashCode()
public boolean equals(Object obj)
```

바보 같은 질문은 없습니다.

- ArrayList는 범용으로 쓸 수 있다고 했는데 왜 ArrayList<DotCom> 같은 식으로 써서 제한을 가하나요?
  - 예전에는 무조건 ArrayList<Object> 같은 식이었습니다.
  - 하지만 자바 5.0에서 <유형> 같은 매개변수화된 유형 기능이 추가되어 이제 특정 유형의 객체만 들어갈 수 있도록 제한할 수 있습니다.
  - 이렇게 되면 나중에 ArrayList에서 객체를 꺼낼 때 훨씬 편하게 쓸 수 있습니다.

바보 같은 질문은 없습니다.

- Object 클래스도 Animal 클래스처럼 "객체"를 만들기가 이상하지 않나요?
  - 주로 포괄적인 개념의 "객체"가 필요한 경우에 Object 객체를 사용합니다. 특히 스레드 동기화를 할 때 많이 사용합니다.
  - Object 객체를 실제로 만들 수는 있지만 그렇게 하는 경우가 그리 흔하지 않다는 정도로만 알아두고 넘어가도 됩니다.

바보 같은 질문은 없습니다.

- Object 유형은 주로 다형적인 인자/리턴형으로 쓰인다고 할 수 있는 건가요? ArrayList에서처럼 말이죠?

  - 임의 클래스에 대해 어떤 작업을 하는 메서드를 만들 때 다형적 유형으로 쓰이는 경우
  - 실행 중 자바에 들어있는 모든 객체에서 필요한 진짜 메서드를 제공하는 경우
  - 스레드와 관련된 메서드도 매우 중요하게 쓰입니다.

바보 같은 질문은 없습니다.

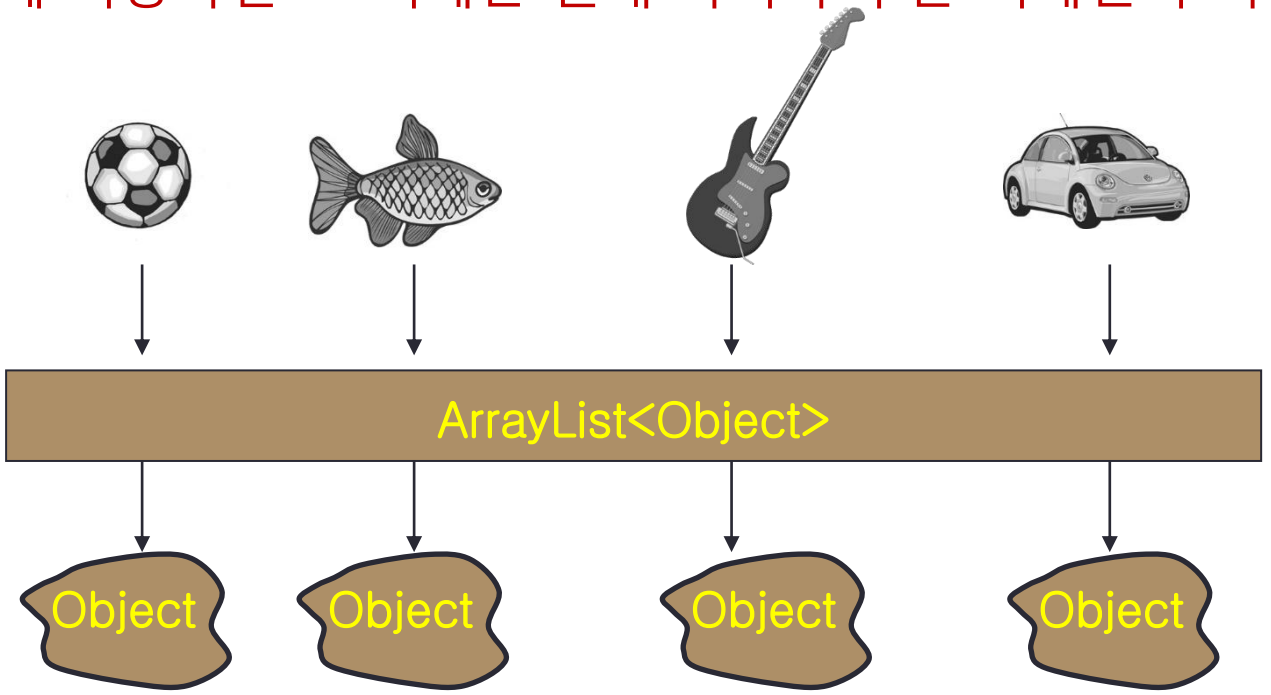
- 왜 모든 메서드의 인자와 리턴 유형을 Object로 하지 않나요?
  - **유형 안전성(type-safety) 문제**
  - 레퍼런스 유형에서 정의되어 있는 메서드만 호출할 수 있습니다.

```
Object o = new Ferrari();
o.goFast();
```

  - Object 유형의 레퍼런스로 참조한 객체에 대해서는 Object 클래스에 정의되어 있는 것만 주문할 수 있습니다.

다형적 레퍼런스

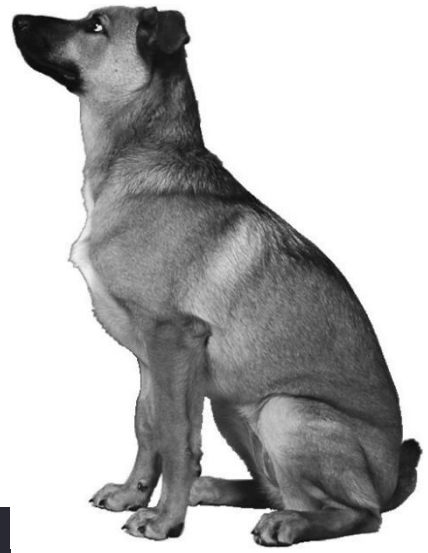
- 어떤 객체를 ArrayList<**Dog**>에 집어넣으면 그 객체는 Dog로 저장되고, 꺼낼 때도 무조건 Dog 객체가 됩니다.
- ArrayList<**Object**>에 저장하면 그 객체는 원래 자기가 무슨 객체인지 기억하지 못합니다.



p.255

기억상실증에 걸린 듯한 객체

```
public void go() {
   Dog aDog = new Dog();
   Dog sameDog = getObject(aDog);
}
public Object getObject(Object o) {
   return o;
}
```

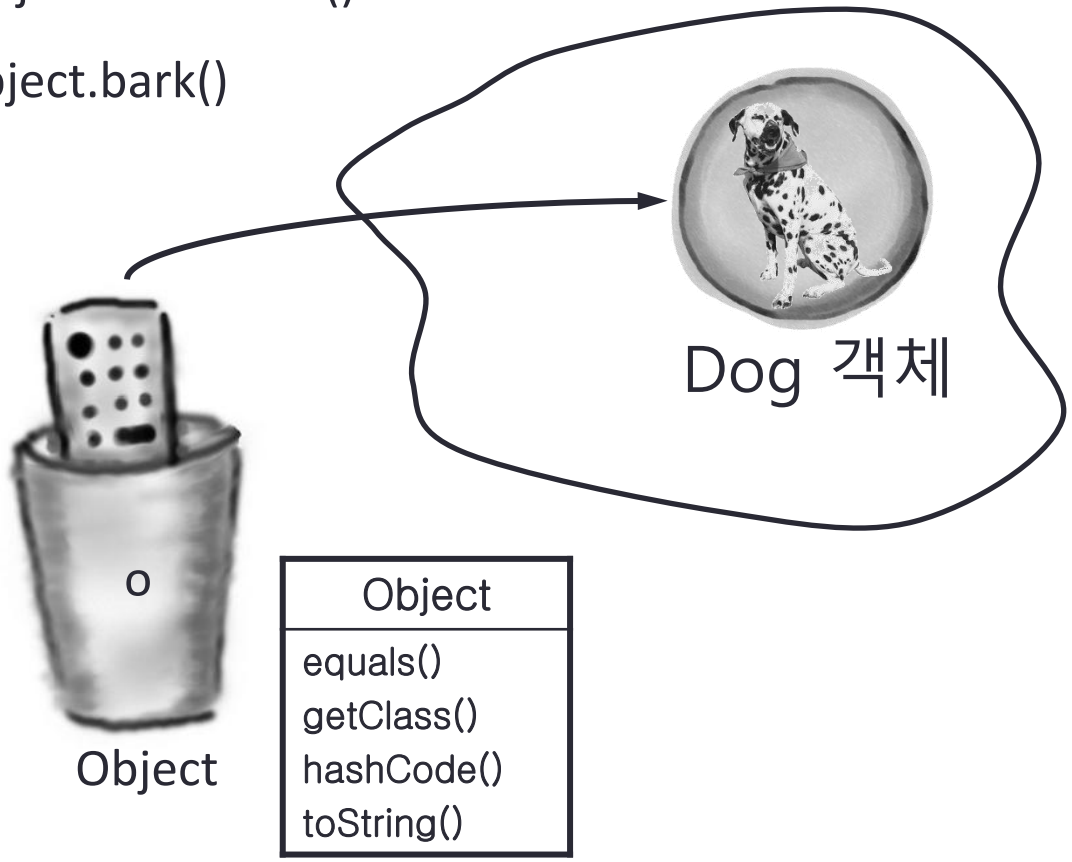


```
DogPolyTest.java:10: incompatible types
found    : java.lang.Object
required  : Dog
    Dog sameDog = getObject(aDog);
                  ^
1 error
```

객체의 유형과 레퍼런스의 유형

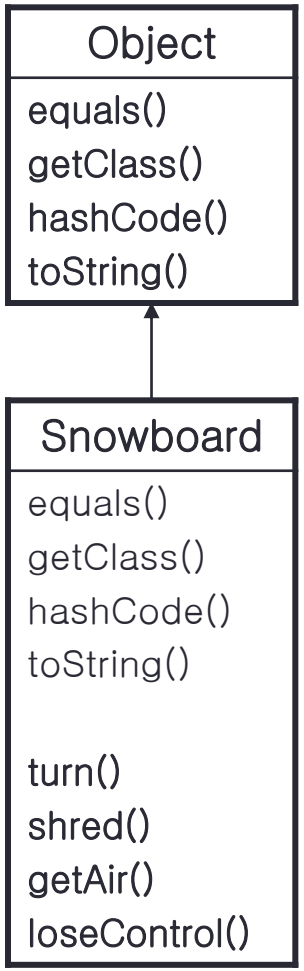
```
Object o = al.get(index);  // al: ArrayList 객체
int i = o.hashCode();  // Object.hashCode()
o.bark();  // Dog.bark(), not Object.bark()
```

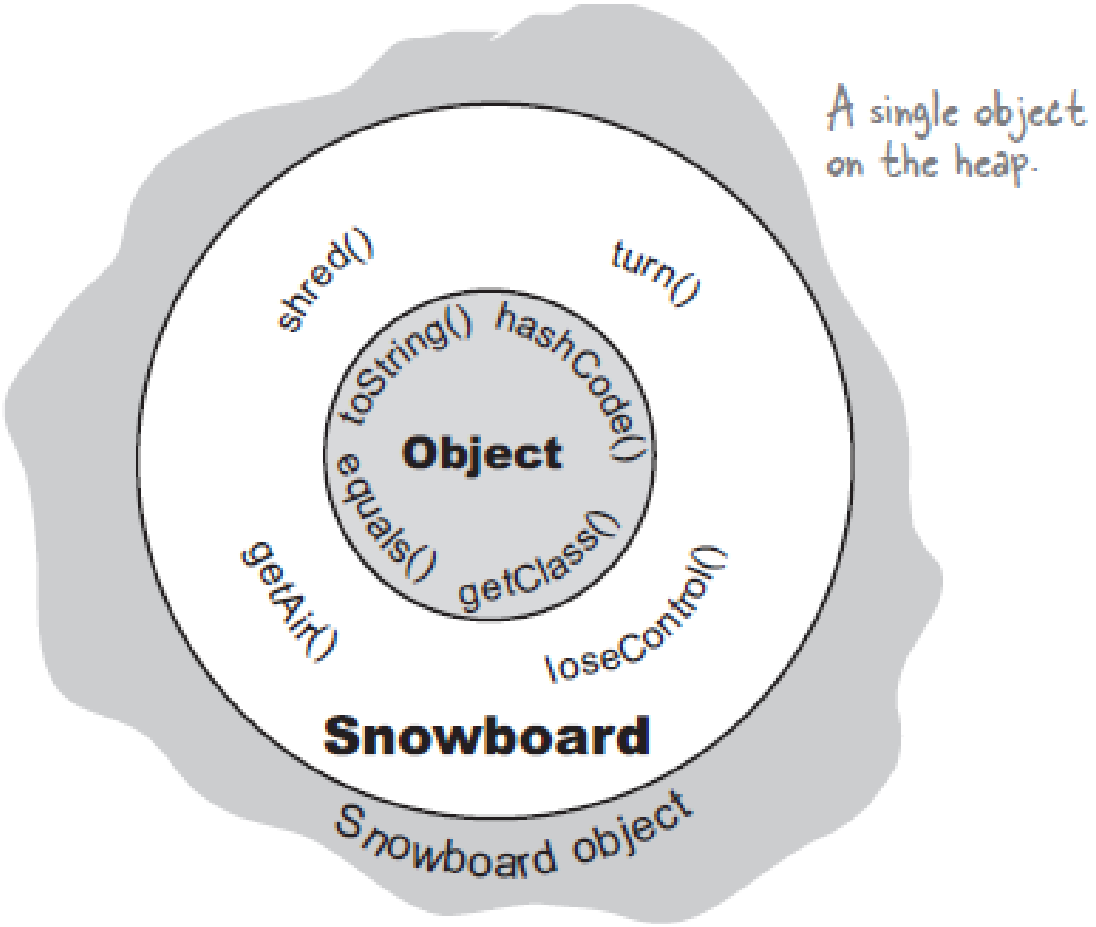
메서드를 호출할 수 있는지 결정할 때는 객체가 아닌 레퍼런스의 유형을 기준으로 합니다.



p.257

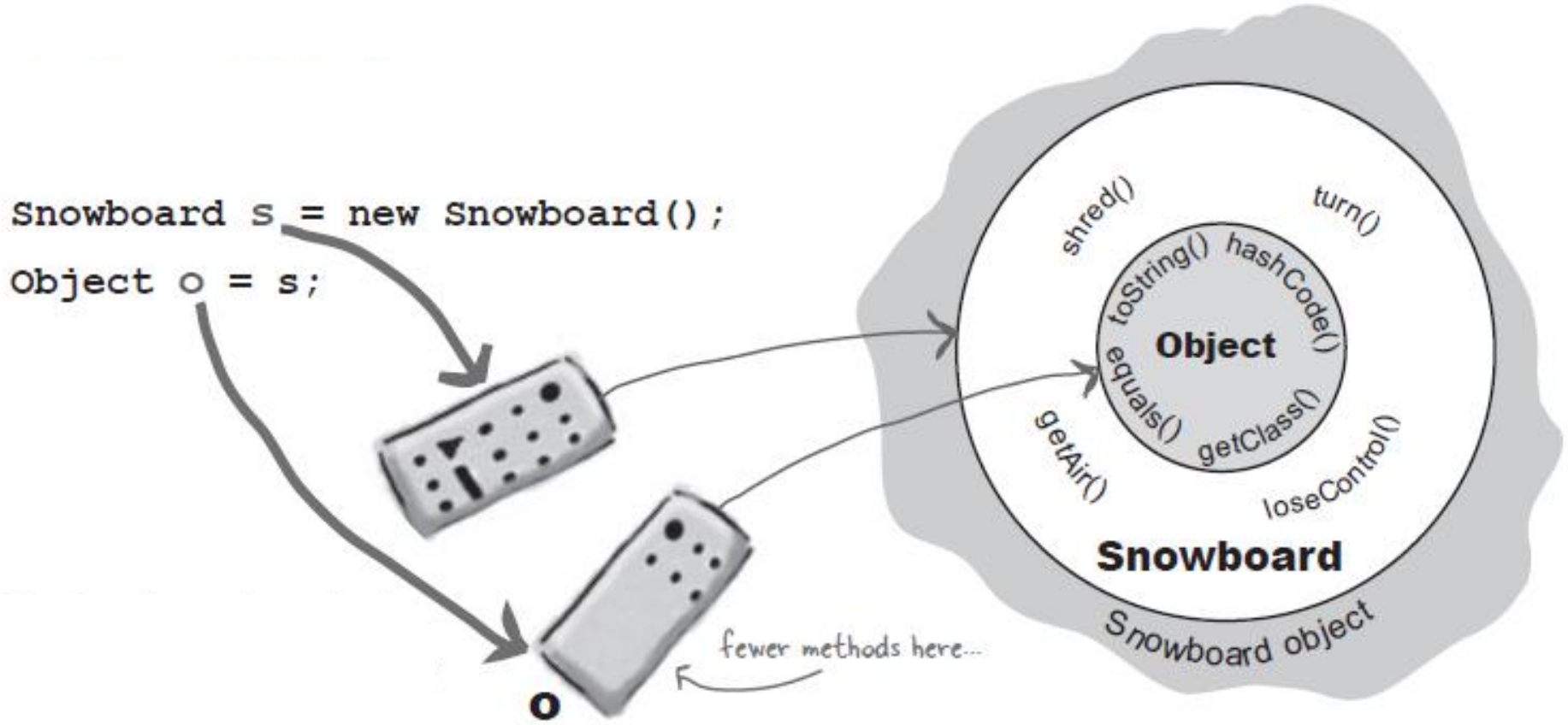
객체 안에 들어있는 알맹이





p.258

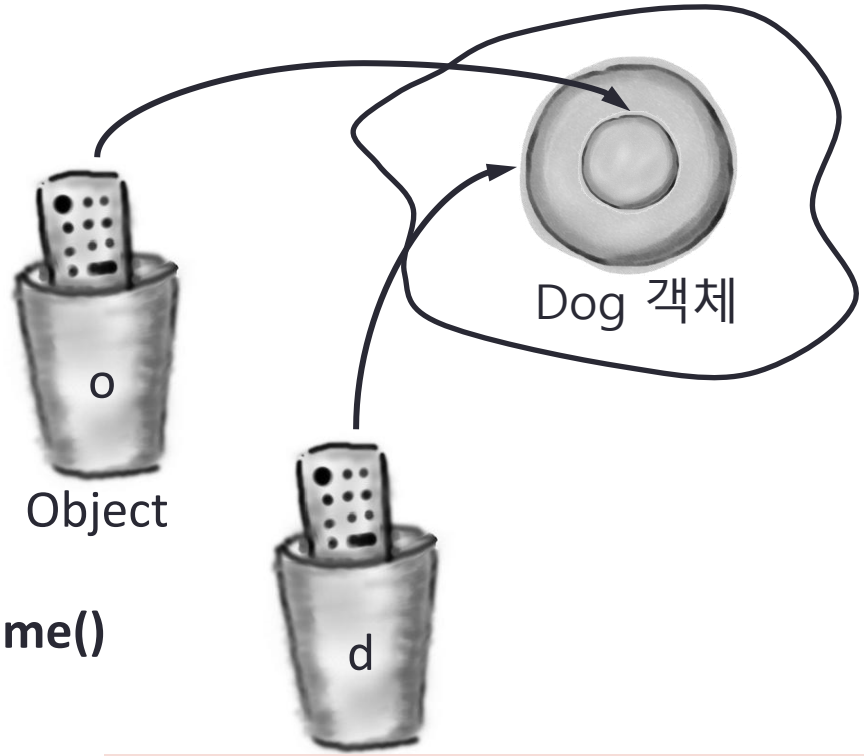
다형성 → 여러 형태



p.259

레퍼런스 캐스팅

```
Object o = al.get(index);
Dog d = (Dog) o;  // type casting
d.roam();
```




```
// 참고. o.getClass().getSimpleName()
if (o instanceof Dog) {
  Dog d = (Dog) o;
  …
```

컴파일러에서는 레퍼런스가 참조하는 실제 객체의 클래스가 아닌 **레퍼런스 변수를 선언할 때 지정한 유형의 클래스를 확인**합니다.

p.260

계약서

- 어떤 객체에 있는 메서드를 호출하려면 그 메서드가 레퍼런스 변수의 클래스에 들어있어야만 합니다.
- 클래스에 있는 **public 메서드는 계약서**, 즉 외부와의 약속이라고 생각하면 됩니다.
- 컴파일러에서는 레퍼런스가 참조하는 실제 객체의 클래스가 아닌 **레퍼런스 변수를 선언할 때 지정한 유형의 클래스**를 확인합니다.

p.261

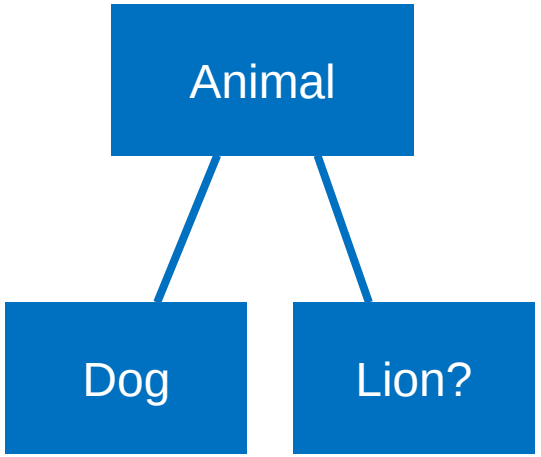
계약서를 고쳐야 한다면?

- Dog 클래스는 동물 시뮬레이션용으로는 적합합니다.
- 하지만 애완동물 가게 프로그램에서 Dog 클래스를 사용하려는 경우를 생각해보면?
- 애완동물(Pet 객체)의 행동인 beFriendly(), play() 같은 메서드가 있어야 합니다.
- 그냥 Dog 클래스에 이런 메서드를 바로 추가하면 될까요?

p.262

Dog를 Pet으로...

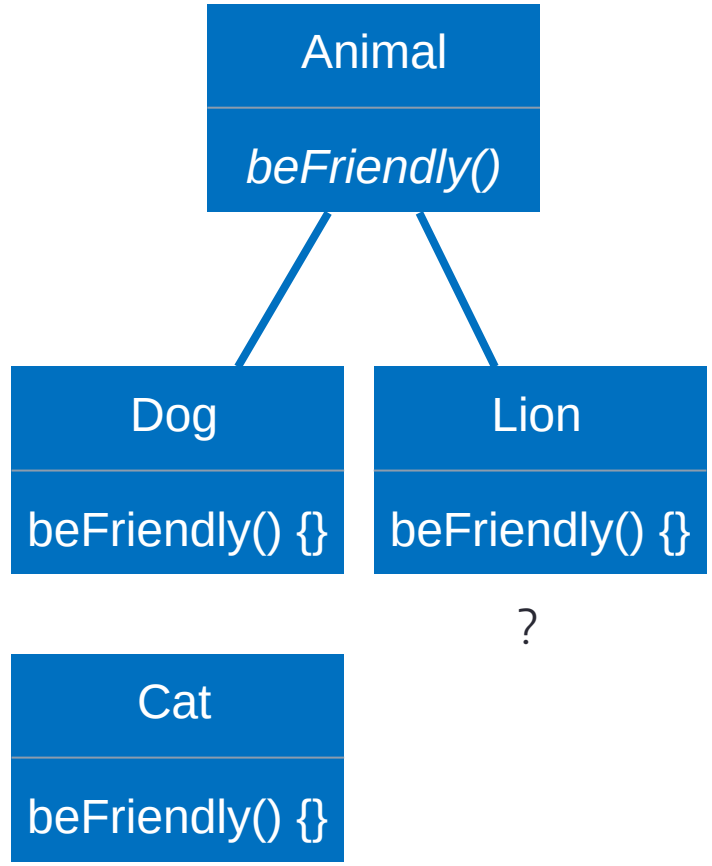
- 첫 번째 방법
  - 애완동물의 성질을 나타내기 위한 메서드를 Animal 클래스에 집어넣습니다.
  - 장점
    - 모든 Animal 객체에 애완동물의 행동이 상속됩니다.
    - 기존의 하위클래스를 전혀 건드리지 않아도 되고 새로 만드는 하위클래스에서도 그런 메서드를 사용할 수 있습니다.
  - 단점
    - 하마, 사자, 늑대 같은 것은 애완동물로 잘 키우지 않죠?
    - 애완동물이 아닌 동물에게 애완동물의 행동을 부여하는 것이 적절치 않습니다.
    - 각 애완동물마다 행동이 많이 다르기 때문에 일일이 수정해야 합니다.



p.263

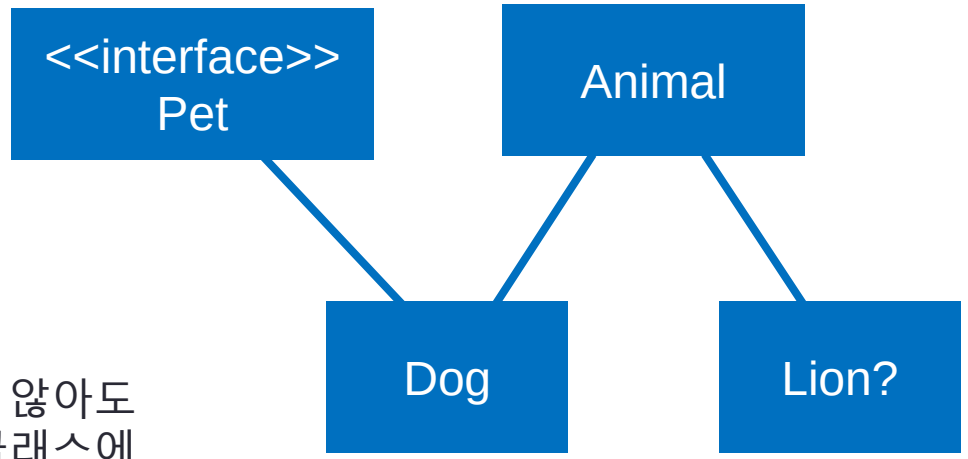
Dog를 Pet으로...

- 두 번째 방법
  - 메서드를 추상 메서드로 만들어서 오버라이드해야만 쓸 수 있도록 만듭니다.
  - 장점
    - 엉뚱한 클래스에서 오버라이드하지 않으면 Pet용 메서드가 실행되지 않도록 할 수 있습니다
  - 단점
    - 구상클래스에서는 무조건 코드를 만들어야 합니다.
    - 일부 유형에만 적용할 것을 Animal 클래스에 집어넣는다는 자체가 잘못된 접근법이라고 할 수 있습니다.



Dog를 Pet으로...

- 세 번째 방법
  - 애완동물용 메서드를 사용할 클래스에만 집어넣습니다.
    → 인터페이스!



  - 장점
    - 애완동물이 아닌 동물에 애완동물용 메서드가 들어갈 걱정을 하지 않아도 됩니다. Dog나 Cat에서는 그런 메서드를 구현할 수 있지만 다른 클래스에서는 전혀 그런 메서드를 쓸 수가 없도록 할 수 있습니다.
  - 단점
    - 제대로 된 계약서를 갖춰야 합니다.
    - 다형성을 적용할 수가 없습니다.

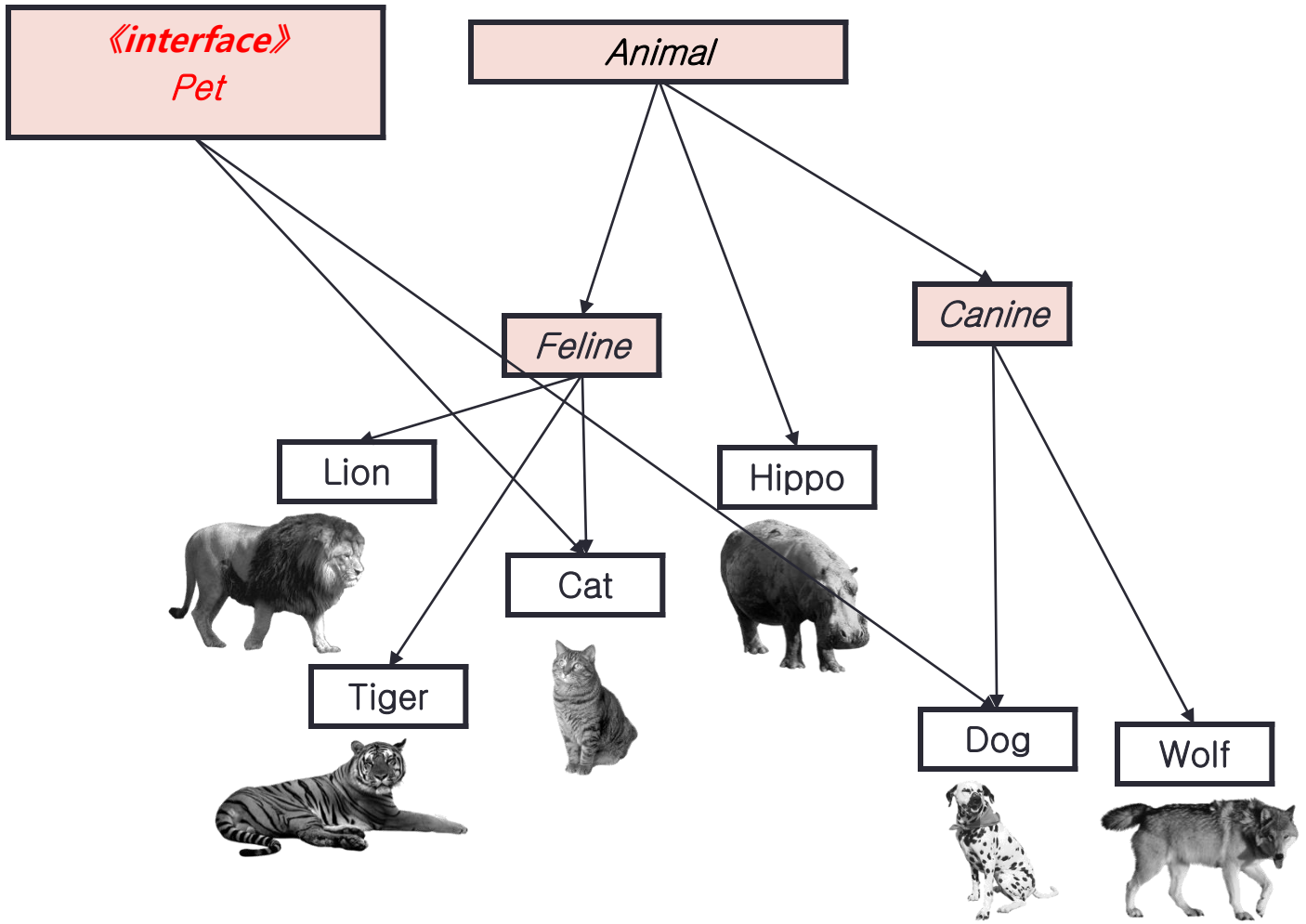
두 개의 상위클래스?

- 필요한 것
  - 애완동물의 행동을 Pet 클래스에만 집어넣는 방법
  - 모든 애완동물 클래스에 똑같은 메서드가 정의되게 하는 방법
  - 각 애완동물마다 다른 인자, 리턴 유형, 배열을 사용하지 않고도 다형성을 활용하여 모든 애완동물에 대해 애완동물용 메서드를 호출할 수 있도록 하는 방법

- 두 개의 상위클래스??

p.266

두 개의 상위클래스?

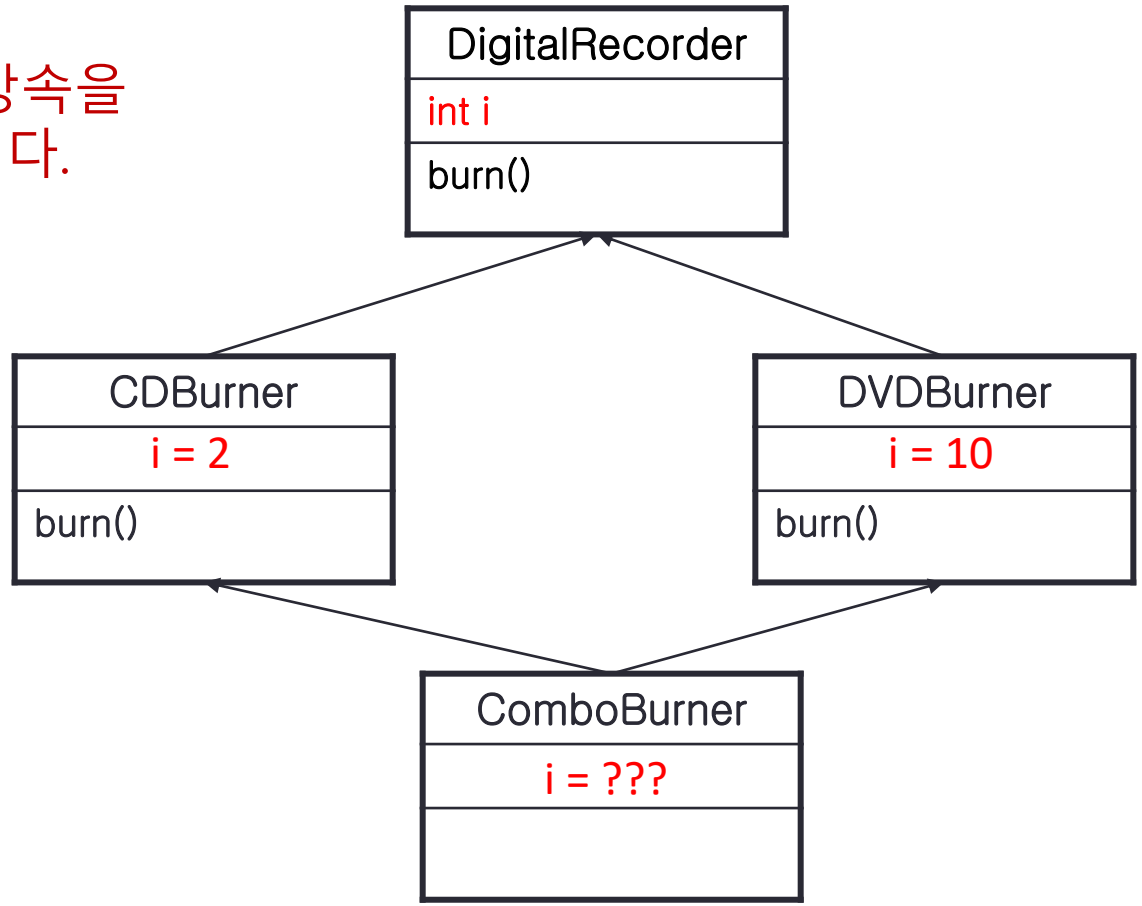


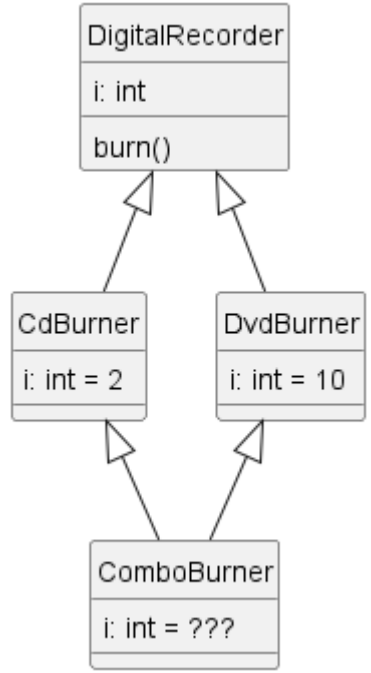
p.266

다중 상속

- 다중 상속(multiple inheritance)의 문제점
  - 죽음의 다이아몬드 (the deadly diamond of death)

자바에서는 다중 상속을
허용하지 않습니다.





p.267

(참고) 죽음의 다이아몬드를 구현한 Python 코드

```
class DigitalRecorder:
  def __init__(self):
    self.i = 0


class CdBurner(DigitalRecorder):
  def __init__(self):
    self.i = 2


class DvdBurner(DigitalRecorder):
  def __init__(self):
    self.i = 10


class ComboBurner(CdBurner, DvdBurner):
  def __init__(self):
    # super().__init__(self)  # 사용 가능하나 상속 받는 클래스 중 앞에 있는 클래스에 영향 받는다.
    CdBurner.__init__(self)
    # DvdBurner.__init__(self)



if __name__ == "__main__":
  combo = ComboBurner()
  print("i = ", combo.i)
```

인터페이스 (interface)

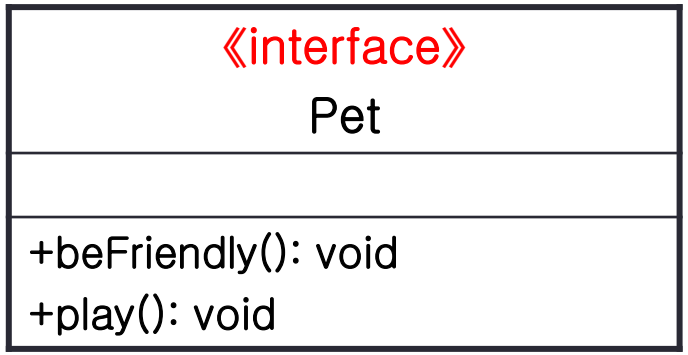
- 인터페이스
  - 모든 메서드가 추상 메서드입니다.
  - 하위클래스에서 반드시 구현해야만 하므로 상속받은 것 중에 어떤 것을 호출해야 할지 결정할 수 없게 되는 문제가 생기지 않습니다.
  - Java 8(2014년 3월 배포)부터는 기본(default) 메서드와 정적(static) 메서드 포함 가능

  - 정의 방법

    public interface Pet {...}

  - 구현 방법

    public class Dog
    extends Canine implements Pet {...}



p.268

(참고) Java 8의 주요 특징

1. **람다 표현식 (Lambda Expressions)**:
   - 함수를 더 간결하게 표현할 수 있게 하며, 코드의 가독성을 높여줍니다.

Java 복사

```
List<String> list = Arrays.asList("a", "b", "c");
list.forEach(s -> System.out.println(s));
```

2. **스트림 API (Stream API)**:
   - 컬렉션을 처리하는 데 있어 보다 선언적이고 간결한 접근 방식을 제공합니다.

Java 복사

```
List<String> list = Arrays.asList("a", "b", "c");
list.stream().filter(s -> s.startsWith("a")).forEach(System.out::println
```

3. **새로운 날짜 및 시간 API (New Date and Time API)**:
   - Java.time 패키지를 도입하여 보다 직관적이고 사용하기 쉬운 날짜 및 시간 처리를 가능하게 합니다.

Java 복사

```java
LocalDate date = LocalDate.now();
LocalTime time = LocalTime.now();
```

4. **기본 메서드 (Default Methods)**:
   - 인터페이스에 메서드 구현을 포함시킬 수 있어, 기존의 인터페이스를 확장하는 것이 보다 유연해졌습니다.

Java 복사

```java
public interface MyInterface {
    default void newDefaultMethod() {
        System.out.println("This is a default method");
    }
}
```

5. **정적 메서드 (Static Methods)**:
   - 인터페이스에 정적 메서드를 포함할 수 있습니다.

Java 복사

```java
public interface MyInterface {
    static void staticMethod() {
        System.out.println("This is a static method");
    }
}
```

6. **메타 애노테이션 (Meta-Annotations)**:
   - 애노테이션에 대한 애노테이션으로, `@Repeatable`과 같은 새로운 애노테이션이 도입되었습니다.

Java 복사

```java
@Repeatable(MyAnnotations.class)
public @interface MyAnnotation {
    String value();
}
```

7. **Nashorn JavaScript 엔진:**
   - JavaScript를 Java 애플리케이션 내에서 실행할 수 있도록 하는 JavaScript 엔진입니다.

AI 검색: Java에서 사용 가능한 Nashone JavaScript 엔진은 어느 버전에서 사용 가능한가?

8. **Optional 클래스:**
   - NullPointerException을 방지하기 위한 새로운 접근 방식으로, 값이 없을 수 있는 변수들을 처리합니다.

Java 복사

```
Optional<String> optional = Optional.ofNullable("Hello");
optional.ifPresent(System.out::println);
```

Pet 인터페이스 정의 및 구현

```
public interface Pet {
  (public) (abstract) void beFriendly();
  (public) (abstract) void play();
}
```

```
public class Dog extends Canine implements Pet {
  public void beFriendly() {…}
  public void play() {…}

  public void roam() {…}
  public void eat() {…}
}
```

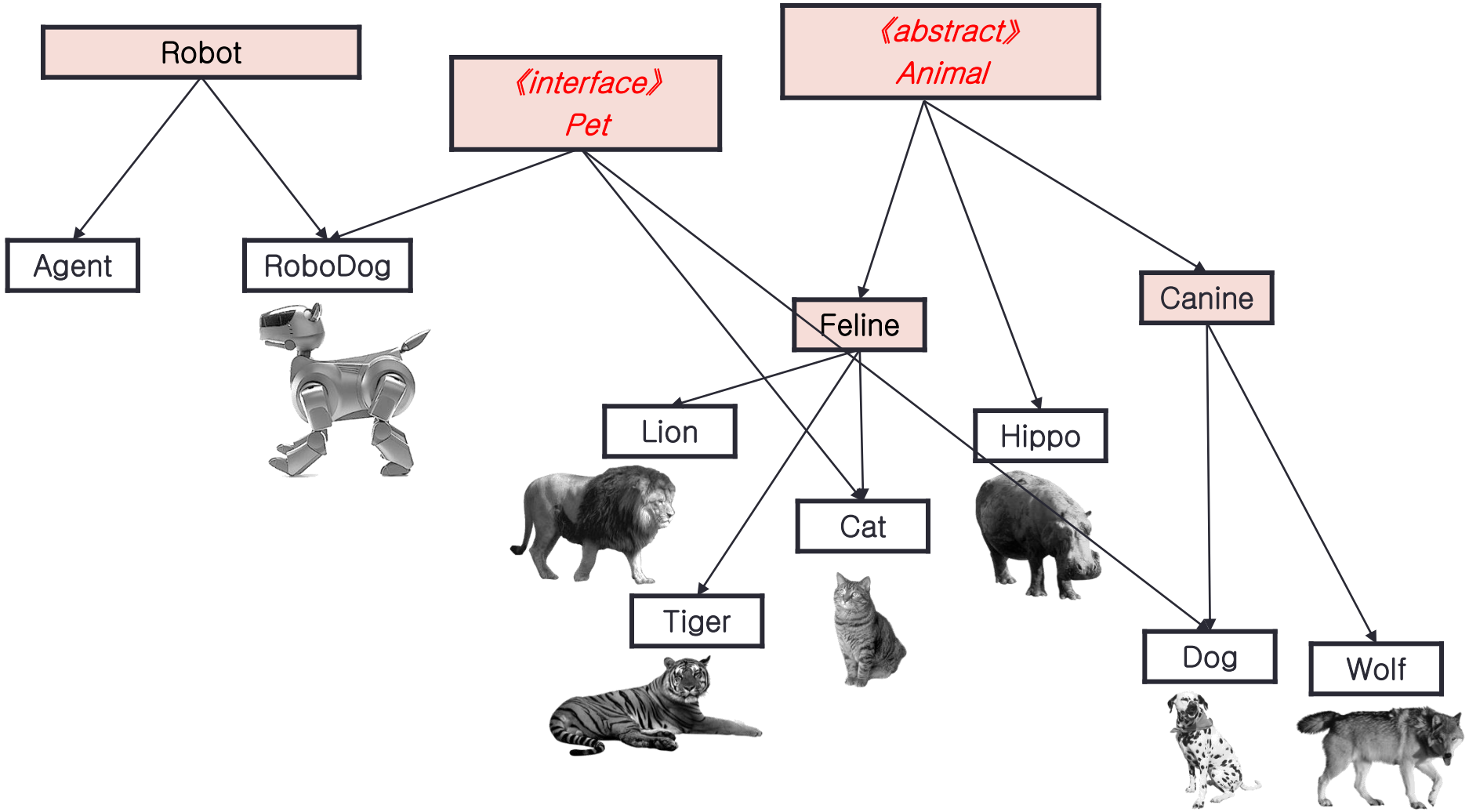
p.269

바보 같은 질문은 없습니다.

- 인터페이스에서는 전혀 코드를 구현할 수 없으니까 진정한 의미에서 다중 상속 기능을 제공한다고 할 수 없지 않나요? 모든 메서드가 추상 메서드라면 인터페이스를 왜 사용해야 하나요?
  - 다형성 때문입니다!!!
  - 인자나 리턴 유형으로 구상 클래스 대신 인터페이스를 사용하면 어떤 객체든 그 자리에 들어갈 수 있습니다.
  - 객체의 역할을 기준으로 해서 처리할 수 있습니다.
  - 인터페이스의 특성상, 구상 메서드를 쓸 수 있더라도 대부분 오버라이드해서 써야 하는 메서드를 정의합니다.

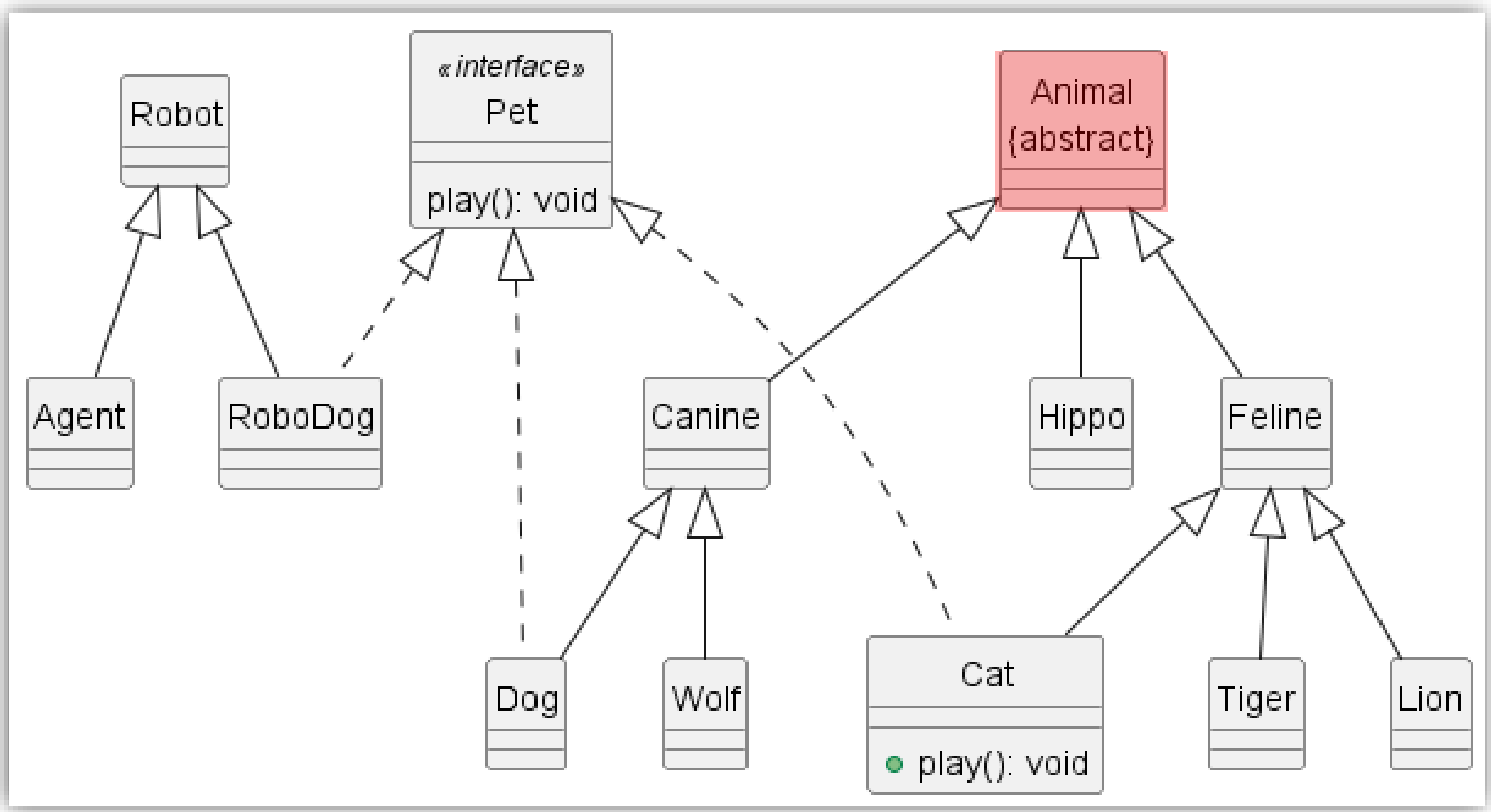
인터페이스

서로 다른 상속 트리에 속한 클래스에서도
같은 인터페이스 구현 가능!

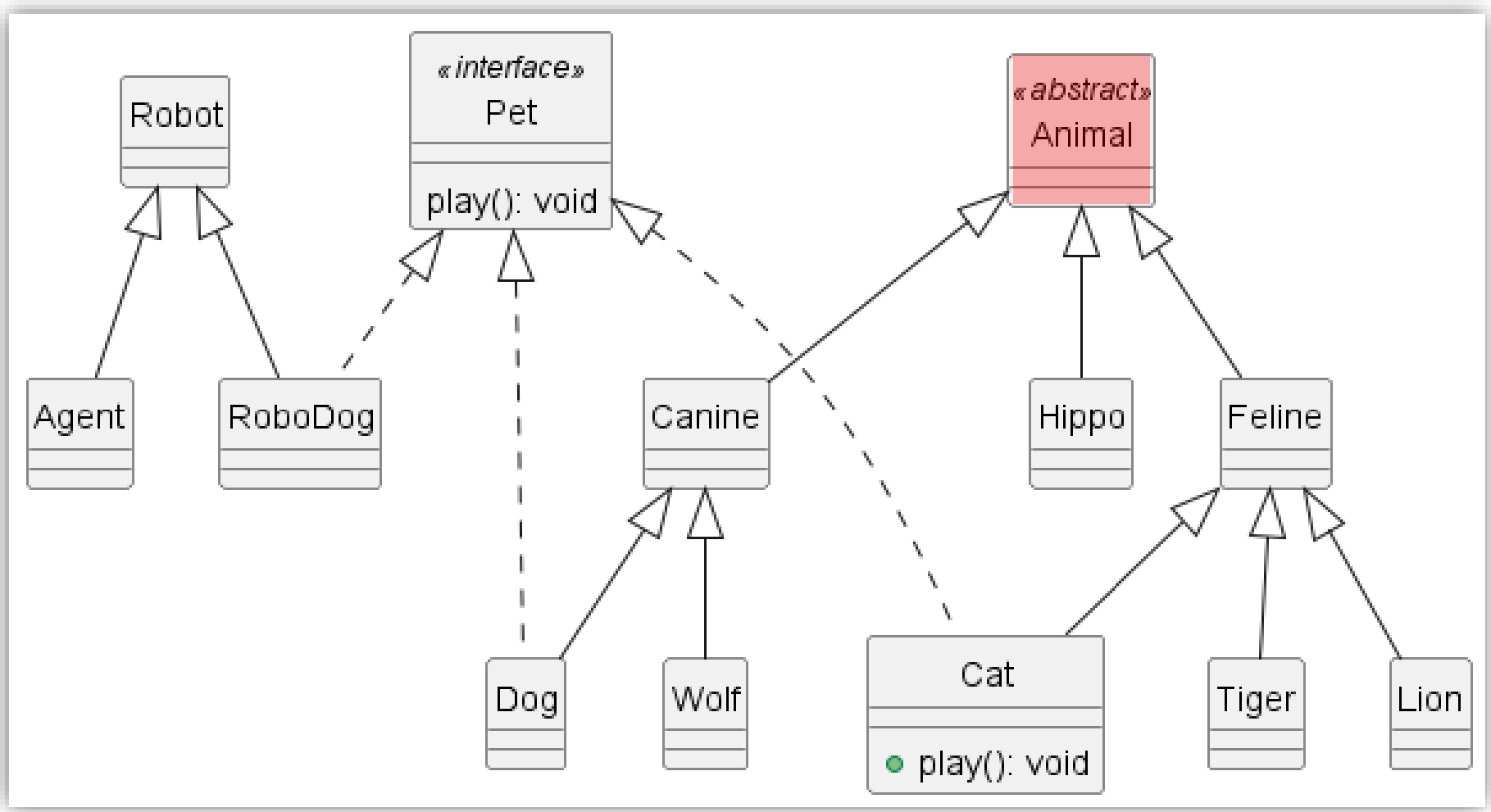


p.270

인터페이스 (PlantUML 사용 1)



인터페이스 (PlantUML 사용 2)



인터페이스

- 서로 다른 상속 트리에서 같은 인터페이스를 구현할 수 있습니다.
- 한 클래스에서 여러 개의 인터페이스를 구현할 수도 있습니다.

```
public class Dog extends Animal implements Pet, Saveable, Paintable {…}
```

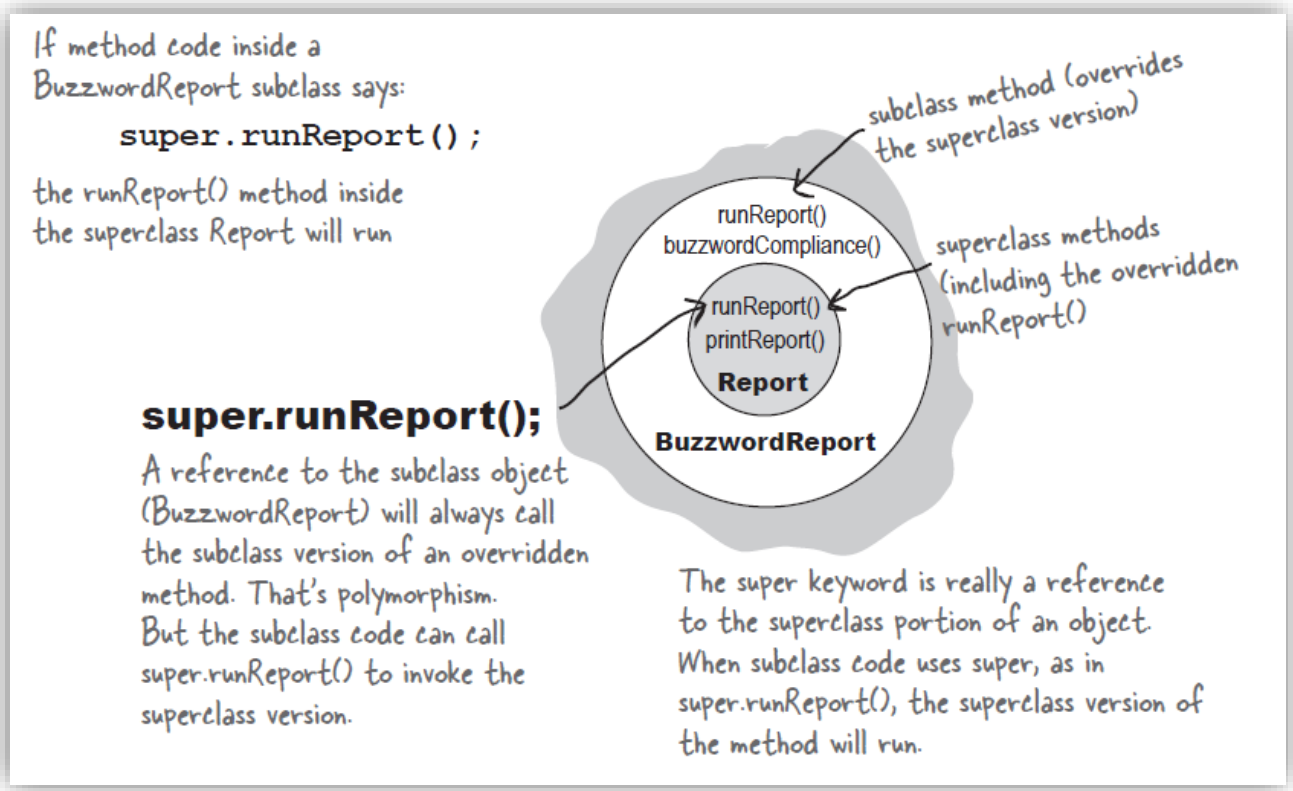
p.270

하위클래스? 추상 클래스? 인터페이스?

- 어떤 클래스가 다른 어떤 클래스에 대해서도 'A는 B다' 테스트를 통과할 수 없다면 그냥 클래스를 만듭니다.
- "더 구체적인 클래스"를 만들고 싶다면 하위클래스를 만듭니다.
- 하위클래스에서 사용할 틀(template)을 정의하고 싶다면, 그리고 구현 코드가 조금이라도 있으면 추상 클래스를 사용합니다.
- 상속 트리에서의 위치에 상관없이 어떤 클래스의 역할을 정의하고 싶다면 인터페이스를 사용하면 됩니다.

super 키워드

- 오버라이드할 때 상위클래스 버전의 메서드의 기능이 필요하다면 어떻게 해야 하나요?
  - super 키워드를 사용하여 상위클래스 버전의 메서드를 호출할 수 있습니다.
  - 추상 클래스에 구상 메서드에서 필요한 포괄적인 작업을 처리하기 위한 코드를 미리 만들어 두고 나중에 구상 메서드에서 더 구체적인 부분만 처리하도록 할 수 있습니다. (템플릿 메서드 패턴)

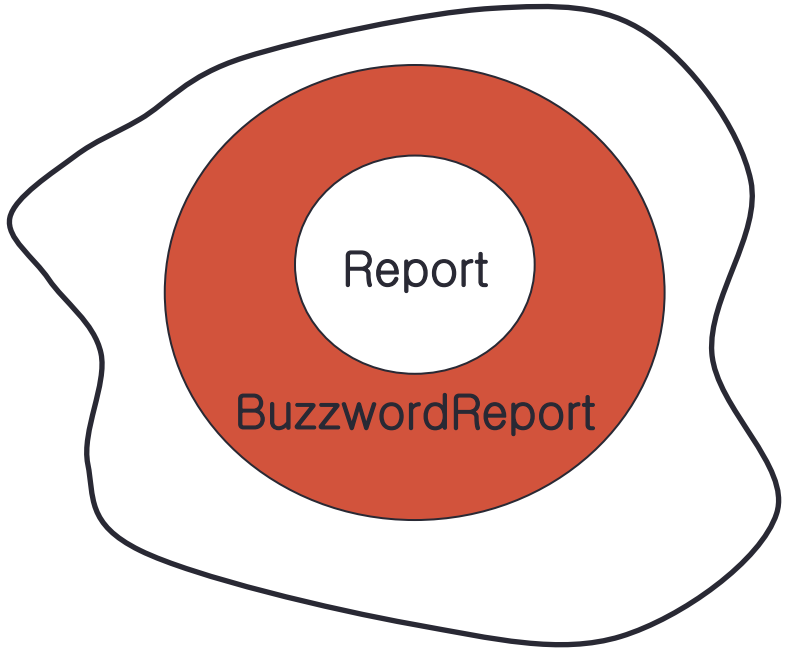


p.272

super 키워드

```
abstract class Report {
  void runReport() {
    // 보고서 설정
  }
  void printReport() {
    // 포괄적인 출력 작업
  }
}

class BuzzWordReport extends Report {
  void runReport() {
    super.runReport();
    buzzwordCompliance();
    printReport();
  }
  void buzzwordCompliance() { … }
}
```



p.272

실습: animal_list

- Animal.java

```
6   package cse.oop2.ch08.animal_list;
7
8   /**...4 lines */
    public abstract class Animal {
13
        public abstract void makeNoise();
15
16      public void sleep() {
17          System.out.println("잠을 자려고합니다.");
18      }
19  }
```

- Dog.java

```
6   package cse.oop2.ch08.animal_list;
7
8   /**...4 lines */
12  public class Dog extends Animal {
13
14      @Override
15      public void makeNoise() {
16          System.out.println("멍멍");
17      }
18
19  }
```

- Cat.java

```
6   package cse.oop2.ch08.animal_list;
7
8   /**...4 lines */
12  public class Cat extends Animal {
13
14      @Override
        public void makeNoise() {
16          System.out.println("야옹 야옹");
17      }
18
19  }
```

- Tiger.java

```
6   package cse.oop2.ch08.animal_list;
7
8   /**...4 lines */
12  public class Tiger extends Animal {
13
14      @Override
        public void makeNoise() {
16          System.out.println("어흥어흥");
17      }
18
19  }
```

- MyAnimalList.java

```
6   package cse.oop2.ch08.animal_list;
7
8   /**...4 lines */
12  public class MyAnimalList {
13
14      private Animal[] animals = new Animal[5];
15      private int nextIndex = 0;
16
17      public void add(Animal a) {
18          if (nextIndex < animals.length) {
19              animals[nextIndex] = a;
20              System.out.println("Animal added at " + nextIndex);
21              nextIndex++;
22          }
23      }
```

```
25      public Animal get(int i) {
26          Animal a = null;
27          if (i >= 0 && i < nextIndex) {
28              a = animals[i];
29          }
30          return a;
31      }
32
33      public int getNextIndex() {
34          return nextIndex;
35      }
36  }
```

- AnimalTestDrive.java

```
6   package cse.oop2.ch08.animal_list;
7
8   /**...4 lines */
12  public class AnimalTestDrive {
13
14      /**...3 lines */
17      public static void main(String[] args) {
18          // TODO code application logic here
19          MyAnimalList list = new MyAnimalList();
20          Dog a = new Dog();
21          Cat c = new Cat();
22          Tiger t = new Tiger();
23
24          list.add(a);
25          list.add(c);
26          list.add(t);
```

```
28              System.out.println("동물이 " + list.getNextIndex() + "마리 있습니다.₩n");
29              for (int i = 0; i < list.getNextIndex(); i++) {
30                  System.out.println(list.get(i).getClass().getName() + ":");
31                  list.get(i).makeNoise();
32                  list.get(i).sleep();
33                  System.out.println();
34              }
35          }
36
37      }
```

- 실행 결과

```
--- exec-maven-plugin:1.5.0:exec (de
Animal added at 0
Animal added at 1
Animal added at 2
동물이 3마리 있습니다.

cse.oop2.ch08.animal_list.Dog:
멍멍
잠을 자려고합니다.

cse.oop2.ch08.animal_list.Cat:
야옹 야옹
잠을 자려고합니다.

cse.oop2.ch08.animal_list.Tiger:
어흥어흥
잠을 자려고합니다.
```

실습: animal_arraylist

- AnimalTestDrive.java (1 / 2)

```
6   package cse.oop2.ch08.animal_arraylist;
7
8   import cse.oop2.ch08.animal_list.Animal;
9   import cse.oop2.ch08.animal_list.Cat;
10  import cse.oop2.ch08.animal_list.Dog;
11  import cse.oop2.ch08.animal_list.Tiger;
12  import java.util.ArrayList;
13  import java.util.List;
14
15  /**...4 lines */
19  public class AnimalTestDrive {
20
21      /**...3 lines */
24      public static void main(String[] args) {
25          // TODO code application logic here
26          List<Animal> list = new ArrayList<>();
27          Dog a = new Dog();
28          Cat c = new Cat();
29          Tiger t = new Tiger();
```

- AnimalTestDrive.java (2 / 2)

```
31              list.add(a);
32              list.add(c);
33              list.add(t);
34
35              System.out.println("동물이 " + list.size() + "마리 있습니다.₩n");
                for (Animal animal : list) {
37                  System.out.println(animal.getClass().getName() + ":");
38                  animal.makeNoise();
39                  animal.sleep();
40                  System.out.println();
41              }
42          }
43
44      }
```

```
for (var animal : list) {
    System.out.println(animal.getClass().getName() + ":");
    animal.makeNoise();
    animal.sleep();
    System.out.println();
}
```

- 실행 결과

```
--- exec-maven-plugin:1.5.0:exec
동물이 3마리 있습니다.

cse.oop2.ch08.animal_list.Dog:
멍멍
잠을 자려고합니다.

cse.oop2.ch08.animal_list.Cat:
야옹 야옹
잠을 자려고합니다.

cse.oop2.ch08.animal_list.Tiger:
어흥어흥
잠을 자려고합니다.
```

실습: pet_interface

- Pet.java (인터페이스 정의)

```
6   package cse.oop2.ch08.pet_interface;
7
8   /**...4 lines */
    public interface Pet {
13
        void play();
15  }
```

- Dog.java

```
6   package cse.oop2.ch08.pet_interface;
7
8   import cse.oop2.ch08.animal_list.*;
9
10  /**...4 lines */
14  public class Dog extends Animal implements Pet {
15
16      @Override
        public void makeNoise() {
18          System.out.println("멍멍");
19      }
20
21      @Override
        public void play() {
23          System.out.println("냄새를 맡으며 뛰어 다닙니다.");
24      }
25
26  }
```

- Cat.java

```
6   package cse.oop2.ch08.pet_interface;
7
8   import cse.oop2.ch08.animal_list.*;
9
10  /**...4 lines */
14  public class Cat extends Animal implements Pet {
15
16      @Override
        public void makeNoise() {
18          System.out.println("야옹 야옹");
19      }
20
21      @Override
        public void play() {
23          System.out.println("그루밍을 하면서 잘 놉니다.");
24      }
25  }
```

- AnimalTestDrive.java (1 / 2)

```
6   package cse.oop2.ch08.pet_interface;
7
8   import cse.oop2.ch08.animal_list.Animal;
9   import java.util.ArrayList;
10  import java.util.List;
11
12  /**...4 lines */
16  public class AnimalTestDrive {
17
18      /**...3 lines */
21      public static void main(String[] args) {
22          // TODO code application logic here
23          List<Animal> list = new ArrayList<>();
24          Dog a = new Dog();
25          Cat c = new Cat();
```

- AnimalTestDrive.java (2 / 2)

```
27              list.add(a);
28              list.add(c);
29
30              System.out.println("동물이 " + list.size() + "마리 있습니다.₩n");
                for (var animal : list) {
32                  System.out.println(animal.getClass().getName() + ":");
33                  animal.makeNoise();
34                  animal.sleep();
35                  if (animal instanceof Dog || animal instanceof Cat) {
36                      Pet p = (Pet) animal;
37                      p.play();
38
39                      ((Pet) animal).play();
40                  }
41                  System.out.println();
42              }
43          }
44      }
```

- 실행 결과

```
--- exec-maven-plugin:1.5.0:exec (de
동물이 2마리 있습니다.

cse.oop2.ch08.pet_interface.Dog:
멍멍
잠을 자려고합니다.
냄새를 맡으며 뛰어 다닙니다.
냄새를 맡으며 뛰어 다닙니다.

cse.oop2.ch08.pet_interface.Cat:
야옹 야옹
잠을 자려고합니다.
그루밍을 하면서 잘 놉니다.
그루밍을 하면서 잘 놉니다.
```

핵심 정리

- 클래스를 만들 때 인스턴스를 만들 수 없게 하고 싶다면 abstract 키워드를 사용하면 됩니다.
- 추상 클래스에는 추상 메서드와 추상 메서드가 아닌 메서드를 모두 집어넣을 수 있습니다.
- 클래스에 추상 메서드가 하나라도 있으면 그 클래스는 추상 클래스로 지정해야 합니다.
- 추상 메서드에는 본체가 없으며 선언 부분은 세미콜론으로 끝납니다.
- 상속 트리에서 처음으로 나오는 구상 클래스에서는 반드시 모든 추상 메서드를 구현해야 합니다.
- 자바에 들어있는 모든 클래스는 직접 또는 간접적으로 Object의 하위클래스입니다.
- ArrayList<Object>에서 꺼내는 객체는 모두 Object 유형입니다.

p.273

핵심 정리

- 레퍼런스 변수를 캐스트해서 객체의 원래 유형으로 돌려놓을 수 있습니다.
- 어떤 객체에 있는 메서드를 호출하려면 그 메서드가 레퍼런스 변수의 클래스에 들어있어야 합니다.
- 자바에서는 다중 상속을 허용하지 않습니다. (죽음의 다이아몬드 문제) 클래스는 단 하나밖에 확장할 수 없습니다.
- 인터페이스는 100% 순수한 추상 클래스입니다.
- 인터페이스를 만들 때는 class 대신 interface라는 키워드를 사용합니다.
- 인터페이스를 구현할 때는 implements라는 키워드를 씁니다.
- 한 클래스에서 여러 개의 인터페이스를 구현할 수 있습니다.
- 인터페이스의 모든 메서드는 public abstract 메서드이므로 인터페이스를 구현하는 클래스에서는 모든 메서드를 구현해야만 합니다. (기본 메서드와 정적 메서드는 제외)

숙제

- 본문을 다시 한 번 꼼꼼히 읽어봅시다.
- 본문 중간에 있는 각종 연습문제와 8장 끝에 있는 연습문제, 퍼즐을 풀어봅시다.
- pp.264-266 반드시 해 볼 것!!!