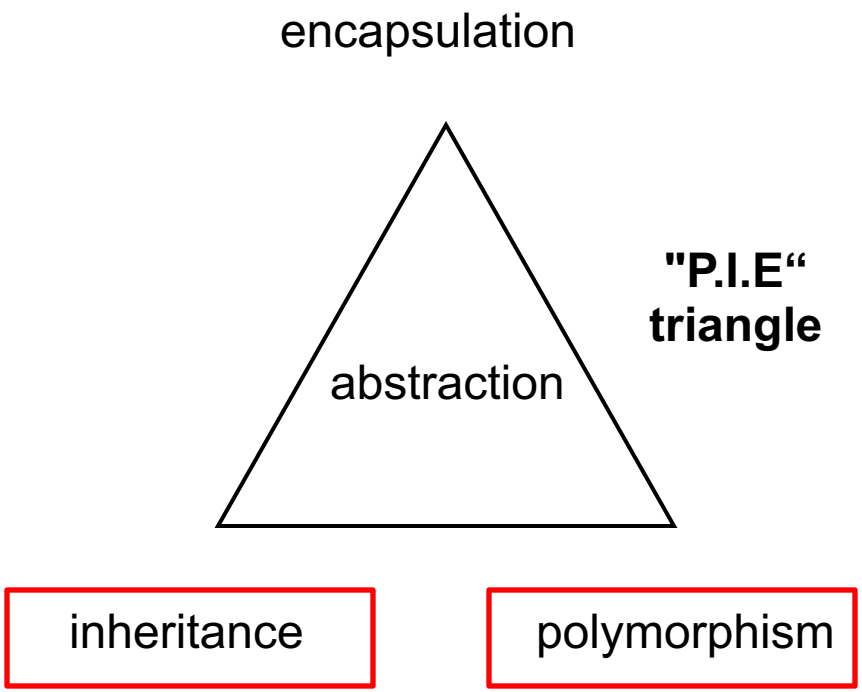
Object-Oriented Programming

Inheritance & Polymorphism

Contents

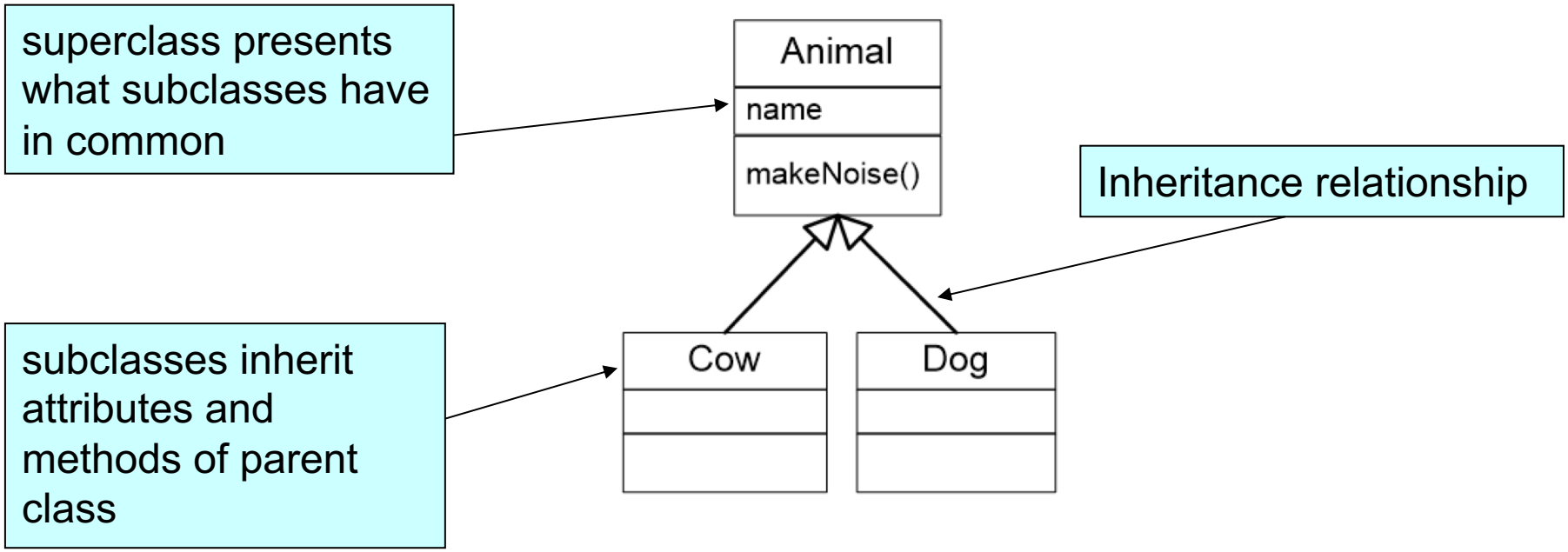
- Concept of inheritance
- Overriding
- IS-A & HAS-A relationship
- Design an inheritance structure
- Concept of polymorphism
- Object class

Important OO Concepts



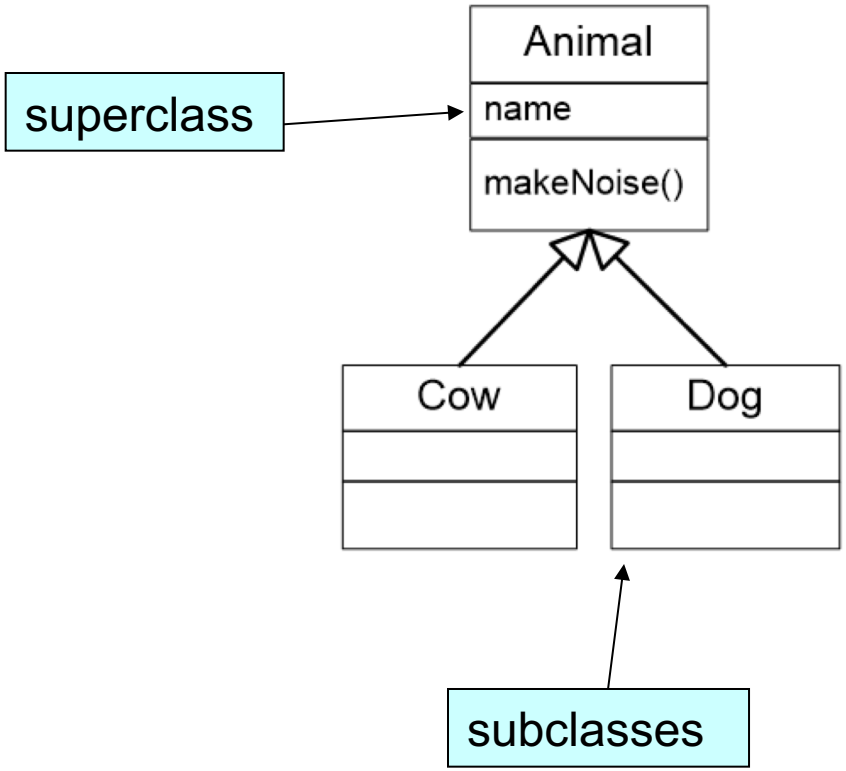
What is Inheritance?

- Inheritance is a relationship where a child class inherits members, i.e. instance variables and methods, of a parent class:
  - The child class is known as subclass or derived class
  - The parent class is known as superclass or base class



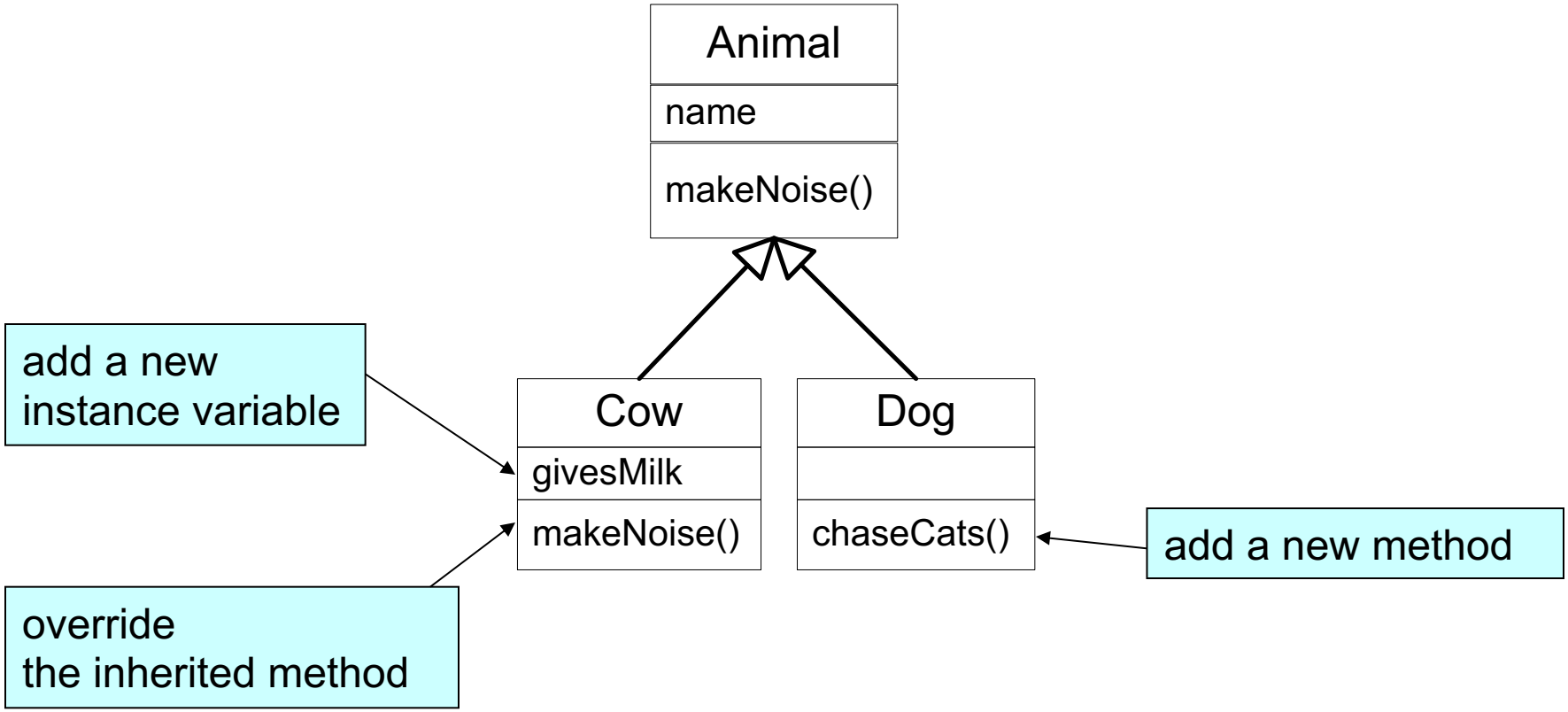
What is Inheritance?

- In inheritance:
  - The superclass is more abstract
  - The subclass is more specific



What is Inheritance?

- In inheritance, the subclass specializes the superclass:
  - It can add new variables and methods
  - It can override inherited methods



Inheritance Declaration

- In Java, extends keyword is used to express inheritance relationship between two classes
- syntax:

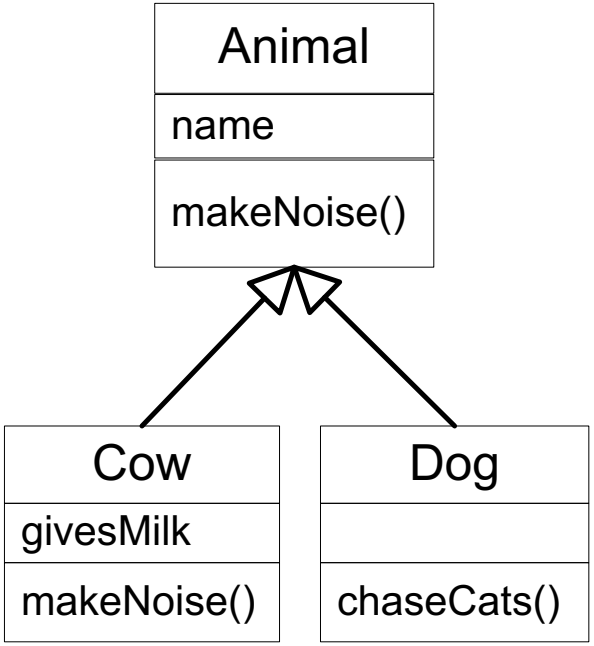
```
class Parent {
        .....
        .....
}
class Child extends Parent {
        .....
        .....
 }
```

Example

```
class Animal {
  String name;
  void makeNoise() {
    System.out.print("Hmm");
  }
}

class Cow extends Animal {
  boolean givesMilk;
  void makeNoise() {
    System.out.print("Moooooooo...");
  }
}

class Dog extends Animal {
  void chaseCats() {
    System.out.print("I'm coming, cat!");
  }
}
```



the overriding method

newly added attribute and method

Overriding - Which method is called?

- Which version of the methods get called?

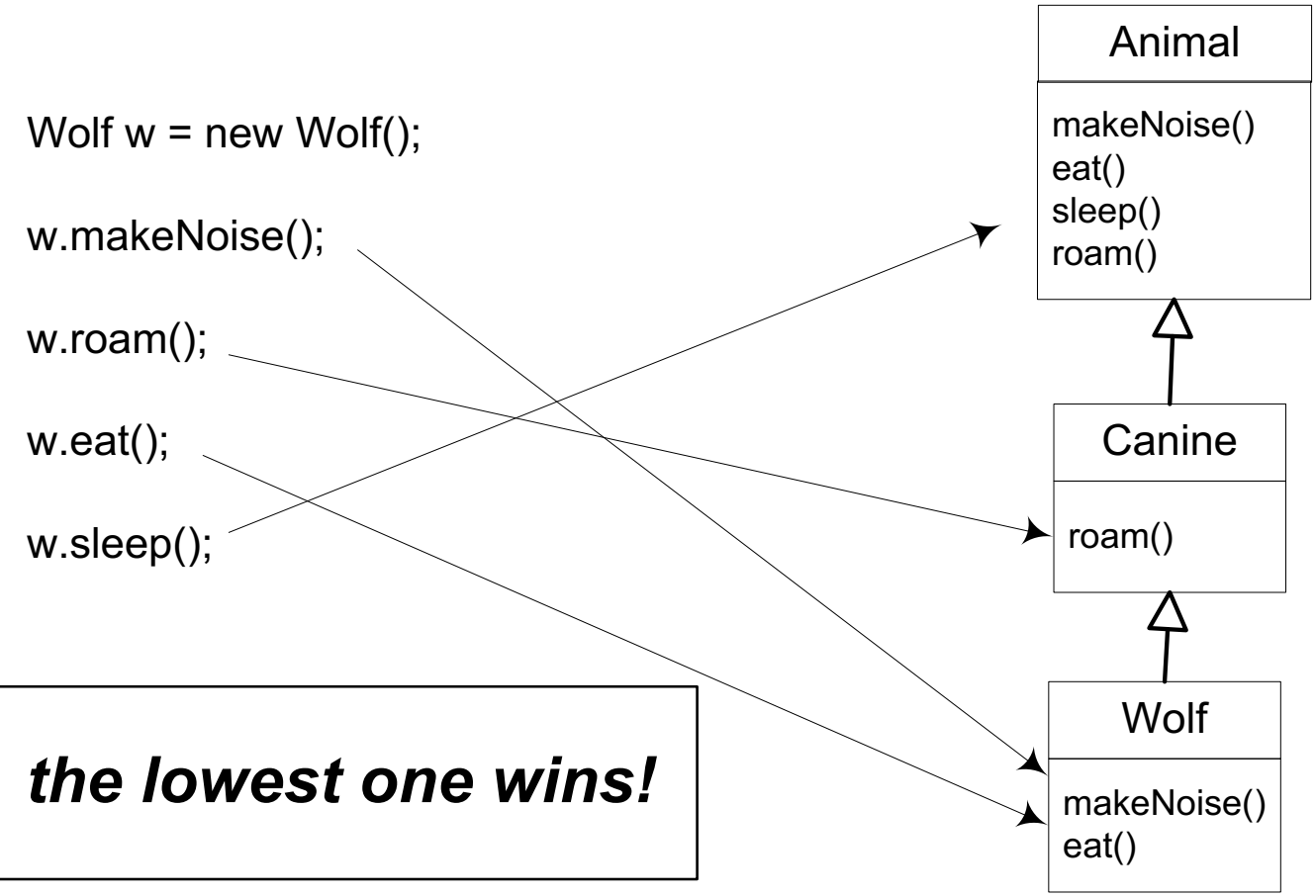
```
Wolf w = new Wolf();

w.makeNoise();

w.roam();

w.eat();

w.sleep();
```

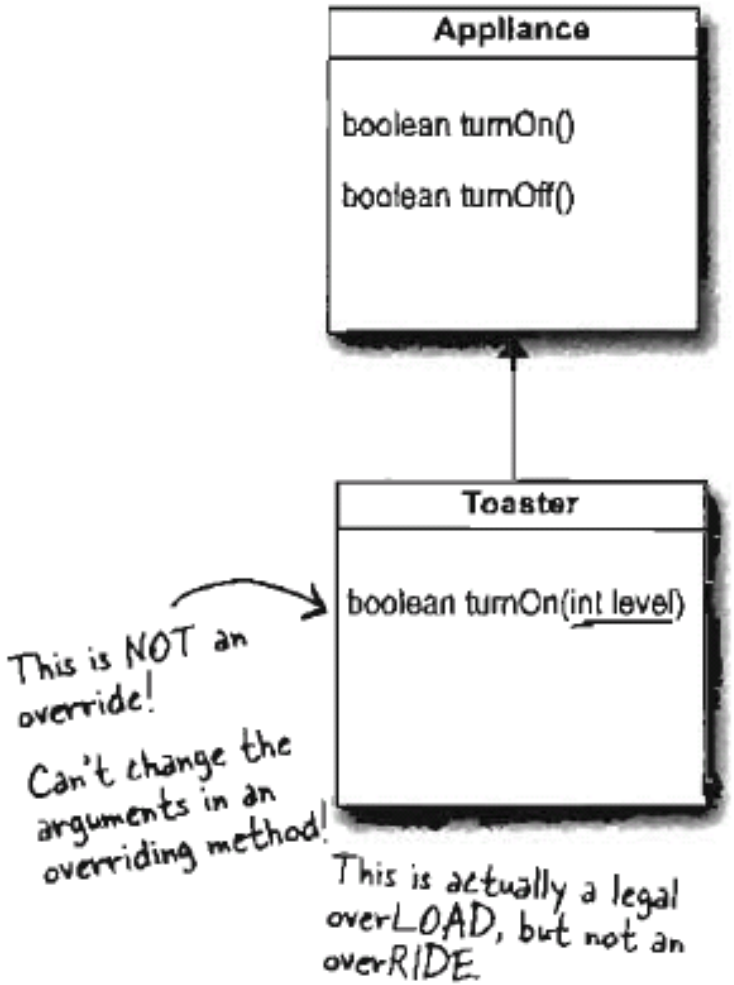


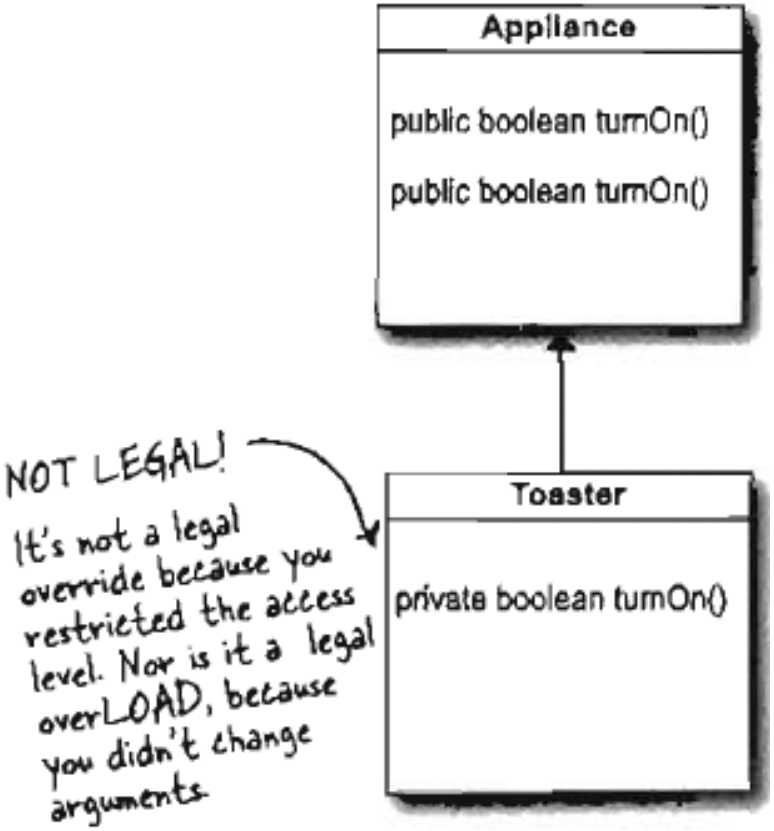
the lowest one wins!

Rules for Overriding

- The principle: the subclass must be able to do anything the superclass declares
- Overriding rules:
  - Parameter types must be the same
    - whatever the superclass takes as an argument, the subclass overriding the method must be able to take that same argument
  - Return types must be compatible
    - whatever the superclass declares as return type, the subclass must return the same type or a subclass type
  - The method can't be less accessible
    - a public method cannot be overridden by a private version

Wrong Overriding

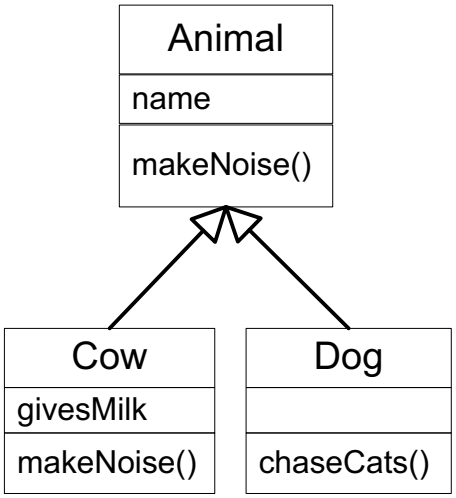




IS-A & HAS-A relationship

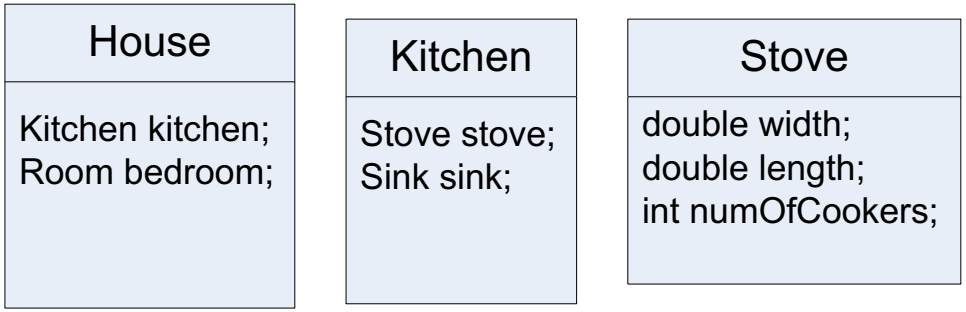
- Triangle IS-A Shape
- Cow IS-An Animal
- Dog IS-An Animal

➡ ***Inheritance***



- House HAS-A Kitchen
- Kitchen HAS-A Sink
- Kitchen HAS-A Stove

➡ ***Composition***



IS-A & HAS-A relationship

- Composition – “HAS-A” relationship
  - the new class is composed of objects of existing classes
  - reuse the functionality of the existing class, but not its form
- Inheritance – “IS-A” relationship
  - create a new class as a *type of* an existing class
  - new class absorbs the existing class's members and extends them with new or modified capabilities

Protected Access Level

Modifier	accessible within			
	same class	**same package**	**subclasses**	**universe**
`private`	Yes			
`package` (*default*)	Yes	Yes		
`protected`	Yes	Yes	Yes	
`public`	Yes	Yes	Yes	Yes

Protected Access Level

Protected attributes of a superclass are directly accessible from inside its subclasses

```
public class Person {
   protected String name;
   protected String birthday;
   ...
}
```

Subclass can directly access superclass's protected attributes

```
public class Employee extends Person {
   protected int salary;
   public String toString() {
      String s;
      s = name + "," + birthday;
      s += "," + salary;
      return s;
   }
}
```

Protected Access Level

Protected methods of a superclass are directly accessible from inside its subclasses.

```
public class Person {
   private String name;
   private String birthday;

   protected String getName()…
}
```

Subclass can directly access superclass's protected methods

```
public class Employee extends Person {
   protected int salary;
   public String toString() {
      String s;
      s = getName() + "," + getBirthday();
      s += "," + salary;
      return s;
   }
}
```

Design an Inheritance Structure

Tiger

HouseCat

- Which one should be subclass/superclass?
- Or, should they both be subclasses to some *other* class?
- How should you design an inheritance structure?

Design an Inheritance Structure

- Case study:
  - Having a number of animals of different species: tigers, lions, wolves, dogs, hippos, cats…

→ how to design the corresponding inheritance structure?

Design an Inheritance Structure

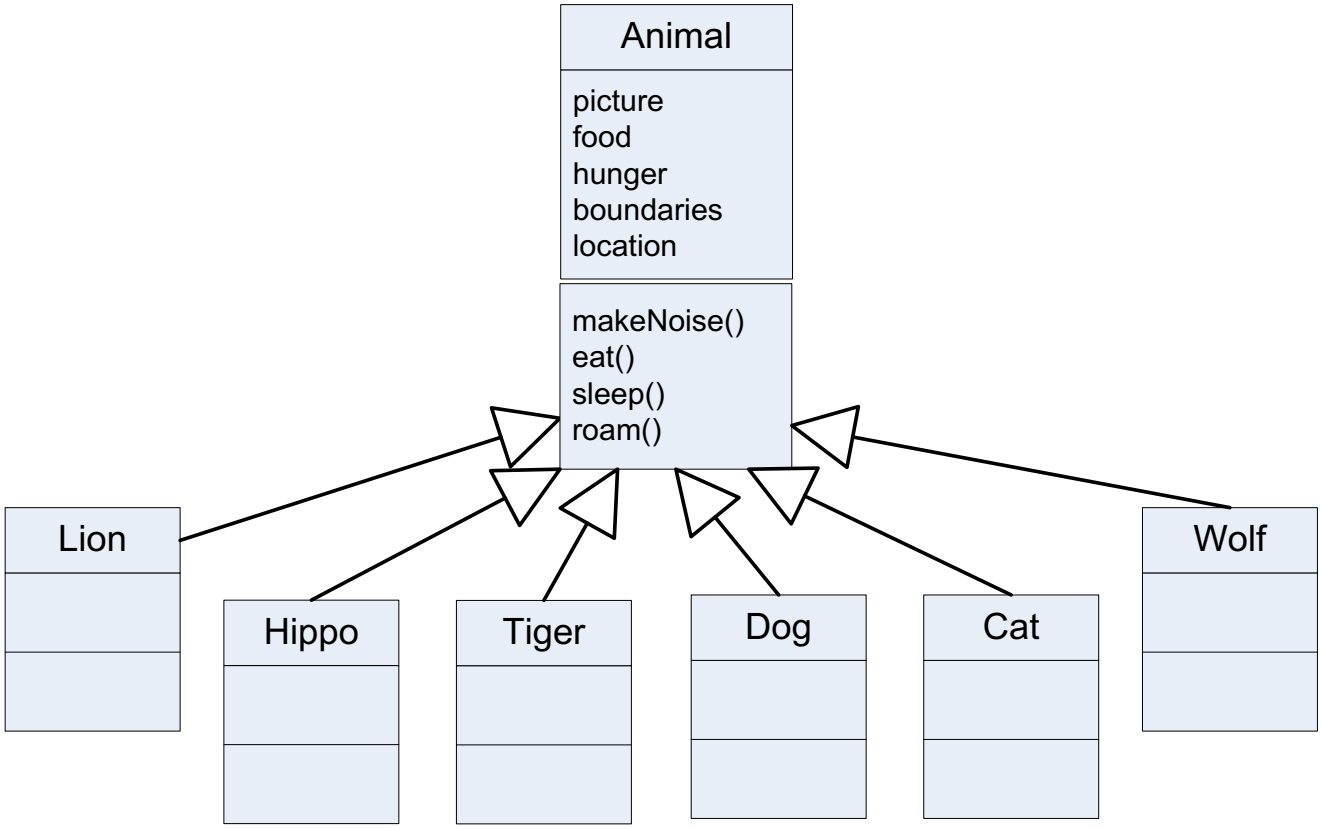
- Step 1: Figure out the common abstract characteristics that all animals have

❑ instance variables
- food
- hunger
- location

❑ methods
- makeNoise()
- eat()
- sleep()
- roam()

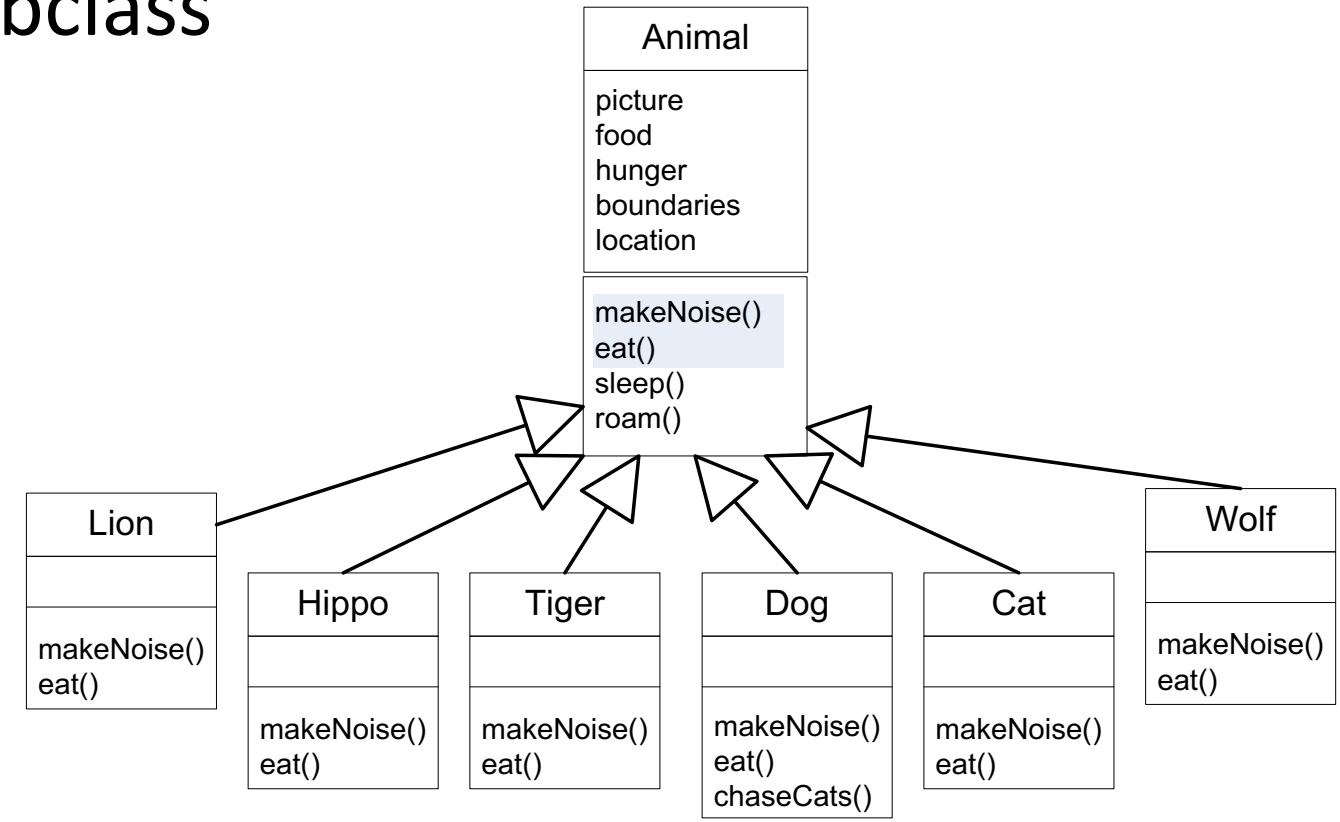
Design an Inheritance Structure

- Step 2: Design a class that represents all common states and behaviors



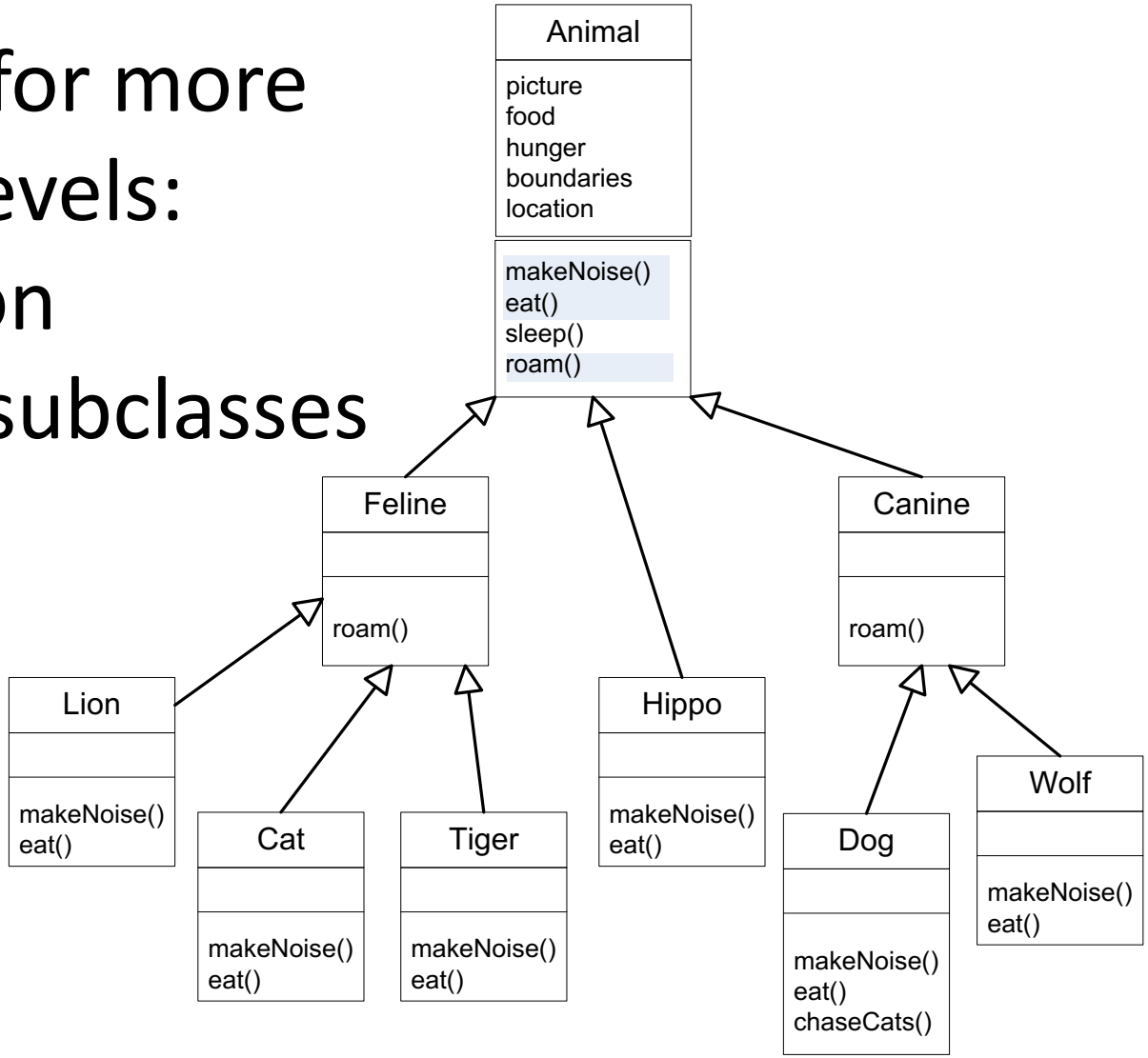
Design an Inheritance Structure

- Step 3: Decide if a subclass needs any behaviors that are specific to that particular subclass



Design an Inheritance Structure

- Step 4: Look for more inheritance levels: more common behaviors in subclasses



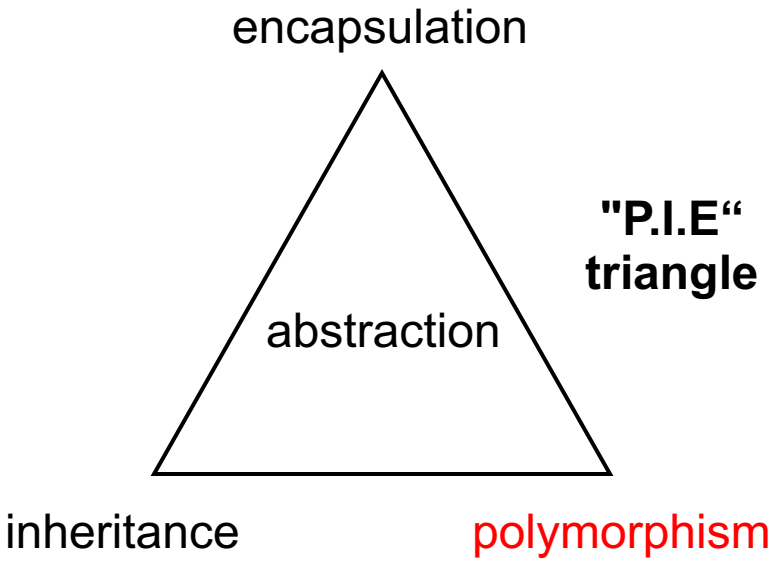
Types of inheritance structure

Single Inheritance
Class A
↑
Class B
	public class A {
	.....
	}
	public class B **extends** A {
	.....
	}

Multi Level Inheritance Class A ↑ Class B ↑ Class C	public class A { .....} public class B **extends** A {.....} public class C **extends** B {..... }
Hierarchical Inheritance Class A ↑ ↑ Class B Class C	public class A { .....} public class B **extends** A {.....} public class C **extends** A {..... }

Multiple Inheritance
Class A Class B
↑ ↑
Class C
	public class A { .....}
	public class B {.....}
	public class C **extends** A,B {
	.....
	} // Java does not support mutiple Inheritance

Important OO Concepts

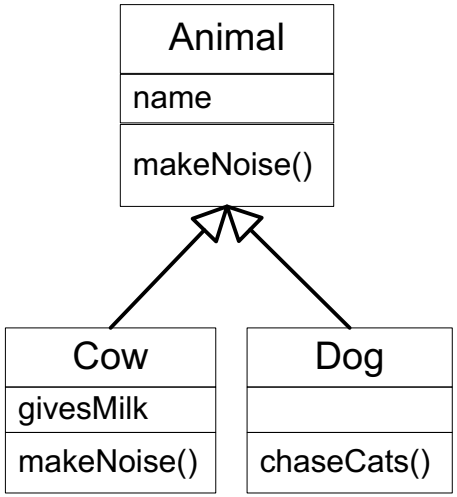


"P.I.E" triangle

What is Polymorphism?

- Polymorphism means "exist in many forms"
- Object polymorphism : objects of subclasses can be treated as if they are all objects of the superclass
- Example:

```
Dog dog = new Dog();
Animal dog = new Dog();
```

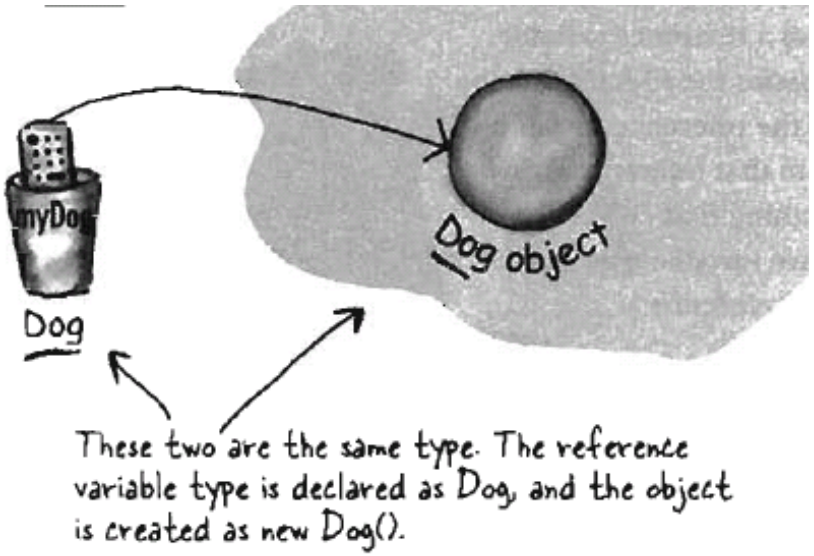


→A Dog object can be seen as an Animal object as well

Polymorphism Example

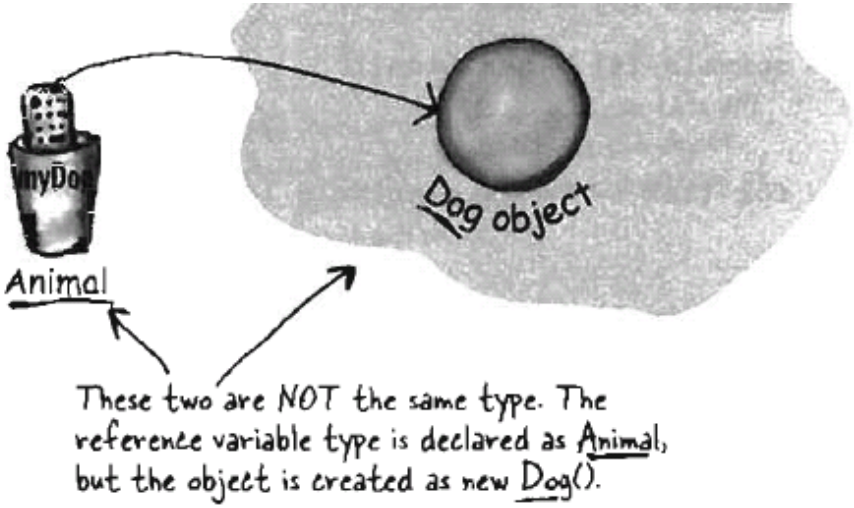
- Normally,

```
Dog dog = new Dog();
```



- With polymorphism:

```
Animal dog = new Dog();
```



→ The reference type can be a superclass of the actual object type

Polymorphism Example

- An array is declared of type Animal. It can hold objects of Animal's subclasses

```
Animal[] animals = new Animal[5];

animals[0] = new Dog();
animals[1] = new Cat();
animals[2] = new Wolf();
animals[3] = new Hippo();
animals[4] = new Lion();

for (int i = 0; i < animals.length; i++) {
  animals[i].makeNoise();
}
```

we put objects of any subclasses of Animal in the Animal array

we can loop through the array and call Animal-class methods

the cat runs Cat's version of makeNoise(), the dog runs Dog's version,…

Polymorphic Arguments & Return Types

- Parameters of type Animal can take arguments of any subclasses of Animal

```
class Pet {
  public void giveVaccine(Animal a) {
    a.makeNoise();
  }
}
```

it takes arguments of types Dog and Cat

```
Pet p = new Pet();
Dog d = new Dog();
Cat c = new Cat();
p.giveVaccine(d);
p.giveVaccine(c);
```

the Dog's makeNoise() is invoked

the Cat's makeNoise() is invoked

```java
class Animal {
   String name;
   ...
   public void makeNoise() {
      System.out.print ("Hmm.");
   }
   public void introduce() {
      makeNoise();
      System.out.println(" I'm " + name);
   }
}
class Cat extends Animal {
   ...
   public void makeNoise() {
      System.out.print("Meow...");
   }
}
class Cow extends Animal {
   ...
   public void makeNoise() {
      System.out.print("Moo...");
   }
}
```

Polymorphism: The same message "makeNoise" is interpreted differently, depending on the type of the owner object

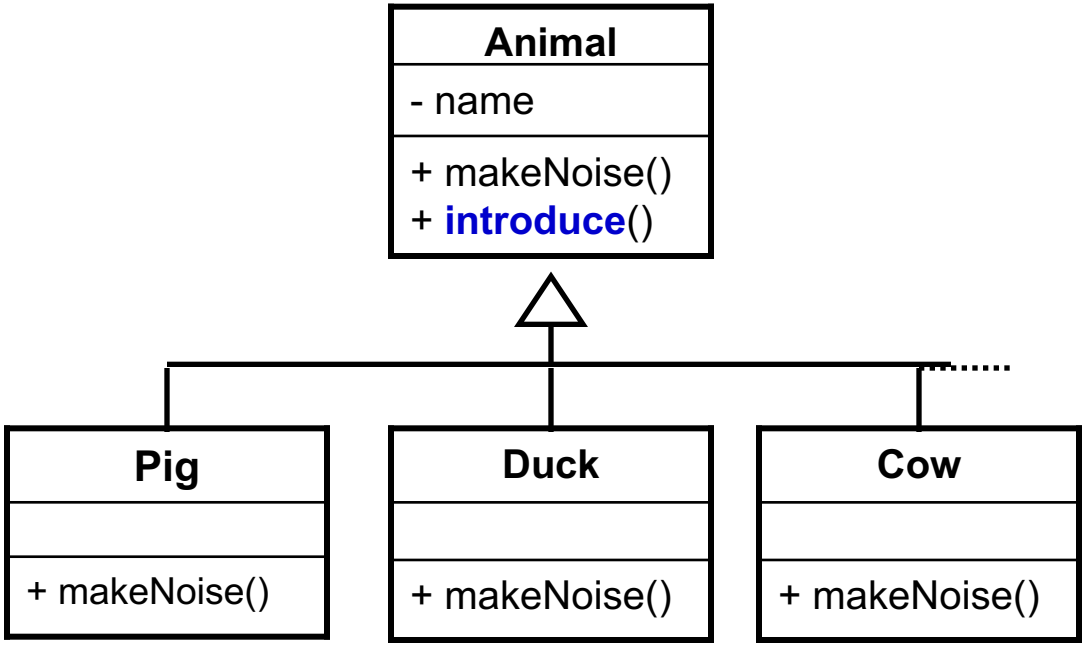
```java
Animal pet1 = new Cat("Tom Cat");
Animal pet2 = new Cow("Mini Cow");
pet1.introduce();
pet2.introduce();
```

Meow... I'm Tom Cat

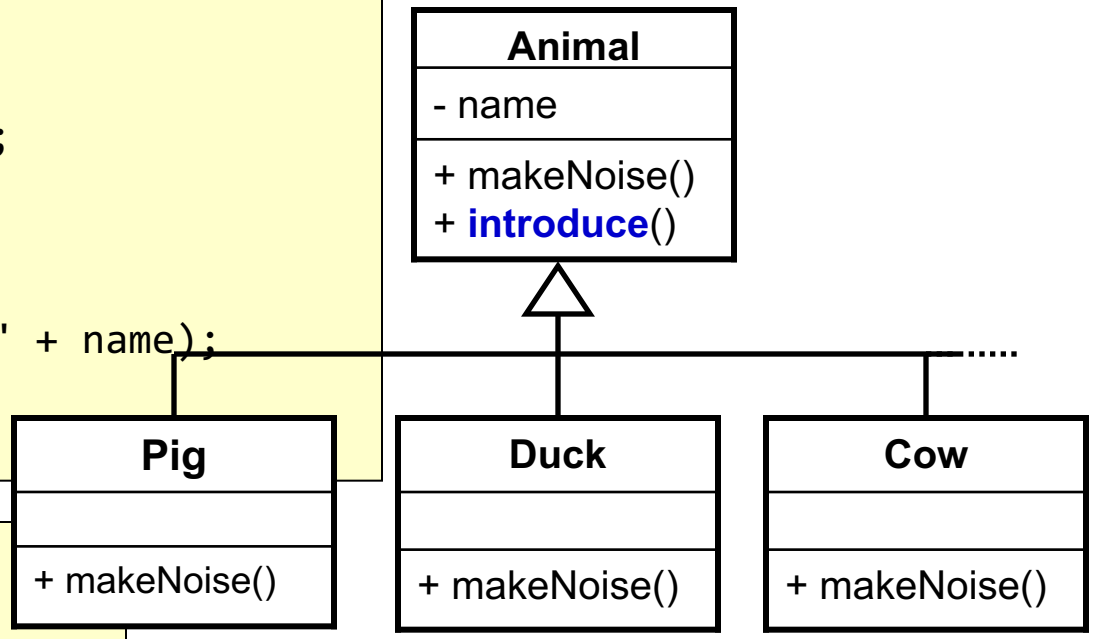
Moo... I'm Mini Cow

Why care about polymorphism?

- With polymorphism, you can write code that doesn't have to change when you introduce new subclass types into the program



```
class Animal {
   ...
   public void makeNoise() {
      System.out.print ("Hmm.");
   }
   public void introduce() {
      makeNoise();
      System.out.println(" I'm " + name);
   }
}
```




```
class Pig extends Animal {
   public void makeNoise() {
      System.out.print("Oi oi...");
   }
}
```

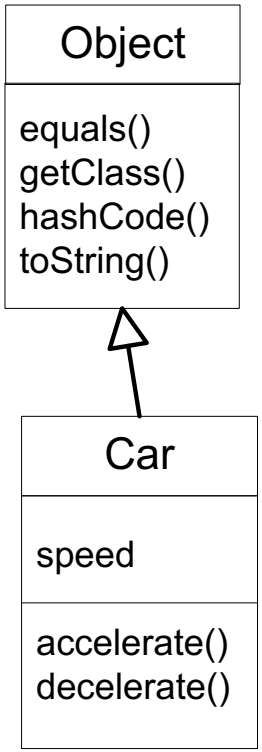
```
class Duck extends Animal {
   public void makeNoise() {
      System.out.print("Quack quack...");
   }
}
```

You can add as many new animal types as you want without having to modify the **`introduce()`** method !

Object Class

- All classes are subclasses to the class Object
- inherited methods:
  - Class getClass()
  - int hashCode()
  - boolean equals()
  - String toString()

equals() and toString() should be overridden to work properly




```
Car c1 = new Car();
Car c2 = new Car();

System.out.println(c1.equals(c2));
System.out.println(c1.getClass() + c1.hashCode());
System.out.println(c1.toString());
```