

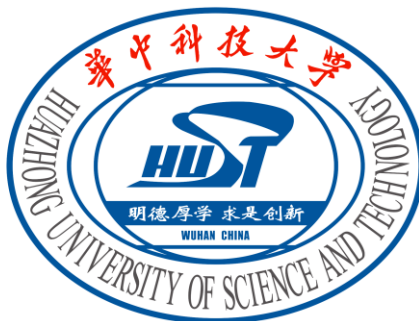
# 基于Java的面向对象程序设计

陈维亚

*weiya\_chen@hust.edu.cn*

华中科技大学软件学院

## 第11讲：多态

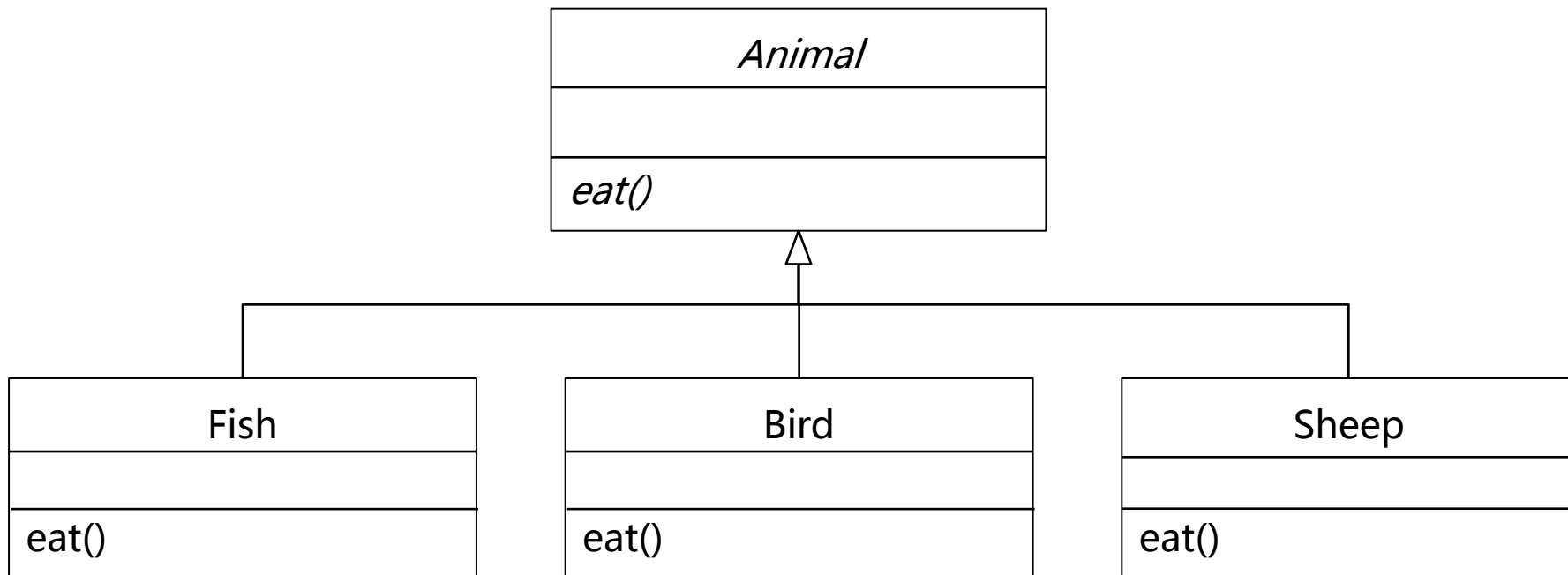


1. 多态是什么
2. 多态从哪来
3. 怎么用多态

# 1. 多态是什么

## □ 定义 Polymorphism

多态就是指一种名称定义不同的方法



## □ 定义

- 编译时多态 - 方法的重载 ( overload )

```
public double createRect(Point p1, Point p2){}  
  
public double createRect(double x1, double y1, double  
x2, double y2){}
```

- 运行时多态 ( runtime polymorphism )

程序中定义的引用变量所指向的具体类型和通过该引用变量发出的方法调用在编程时并不确定，而是在程序运行期间才确定，即一个引用变量到底会指向哪个类的实例对象，该引用变量发出的方法调用到底是哪个类中实现的方法，必须在由程序运行期间才能决定。

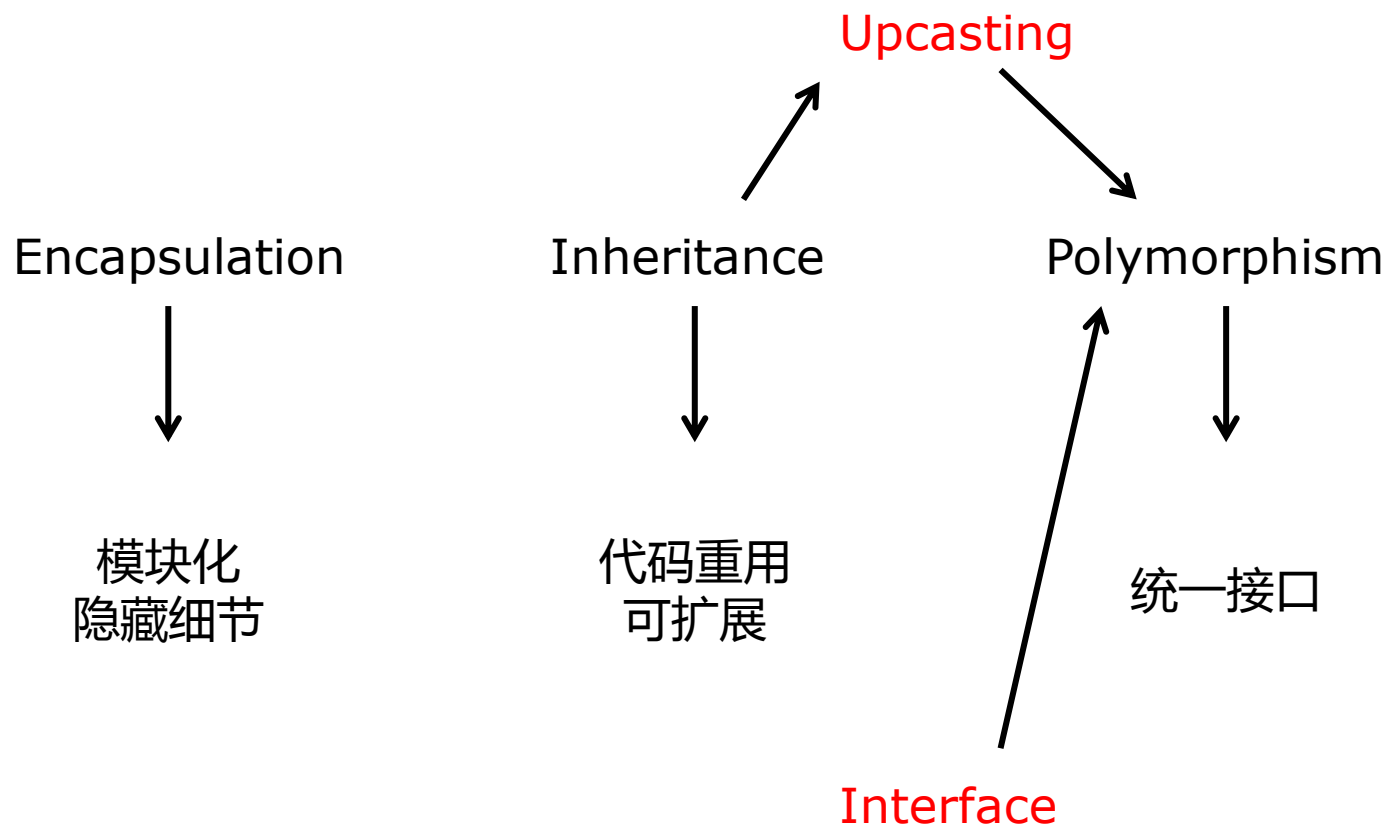
```
...  
public void add(Animal a){}  
...
```

```
Zoo zoo = new Zoo();  
zoo.add(new Fish());
```

## 2. 多态从哪来

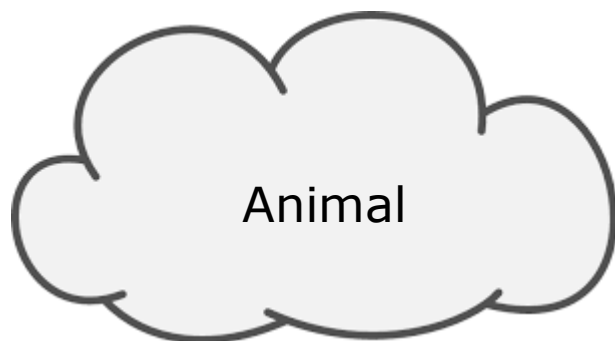


### □ 面向对象的三大特点

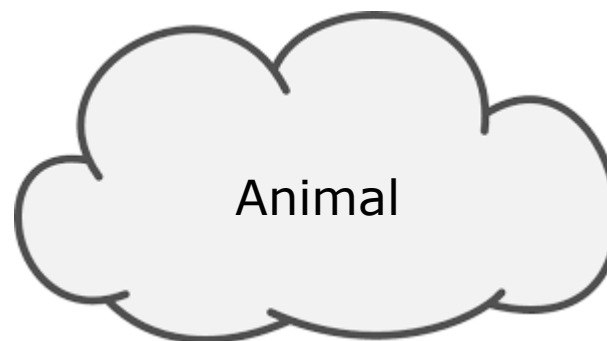


## 2. 多态从哪来

### □ 继承中的向上转型 - 动物王国的面向对象（IBM刘欣）



a1

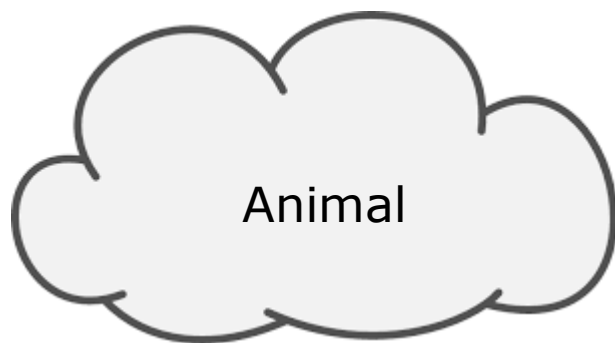


a2

```
Animal :  
public String getFavoriteFood(){  
    return "不知道";  
}
```

```
int age;  
System.out.println(a2.age)
```

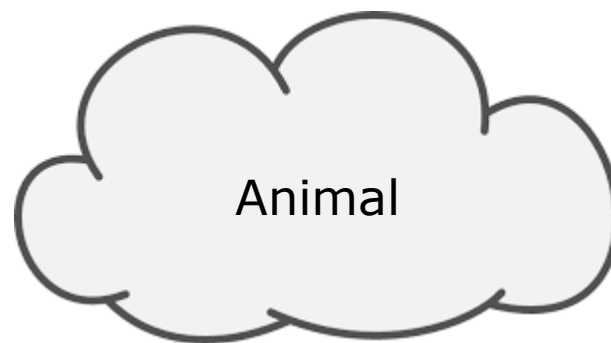
## 2. 多态从哪来



a1

**Animal :**  
`public String getFavoriteFood(){  
 return "不知道";  
}`

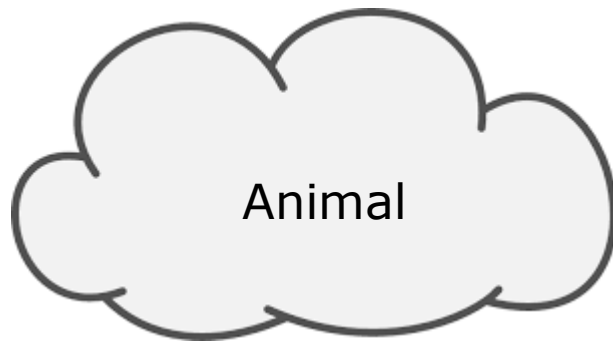
`a1.getFavoriteFood(),`  
天哪，它返回了“骨头”！



a2

`a2.getFavoriteFood(),`  
它竟然返回了“鱼”！

## 2. 多态从哪来



a1

**Animal :**

```
public String getFavoriteFood(){  
    return "不知道";  
}
```

a1.getFavoriteFood(),  
天哪，它返回了“骨头”！



Cat

```
if(a2 instanceof Cat)  
    Cat cat = (Cat) a2
```



## 2. 多态从哪来



Dog

```
if(a1 instanceof Dog)
    Dog dog = (Dog)a1
```



Cat

```
if(a2 instanceof Cat)
    Cat cat = (Cat) a2
```

```
Animail a2 = new Cat()
```

## 2. 多态从哪来

```
public String eat(Food d);
```



```
public void enterGate(Rabbit r);
```

```
enterGate(dog);
```

```
Rabbit r = (Rabbit)cat;
```



Dog



Cat

## 2. 多态从哪来

```
public String eat(Food d);
```



Dog



Cat

```
public void enterGate(List<Rabbit> rabbits){  
    Iterator<Rabbit> iter = rabbits.iterator();  
    while(iter.hasNext()){  
        Rabbit r = iter.next();  
        r.eat(food);  
    }  
}
```

```
List list = new ArrayList();  
list.add(cat);  
list.add(dog);  
enterGate(list);
```

java.lang.ClassCastException: Cat cannot be cast to Rabbit

## 2. 多态从哪来

```
public String eat(Food d);
```



```
public void enterGate(Animal a);
```



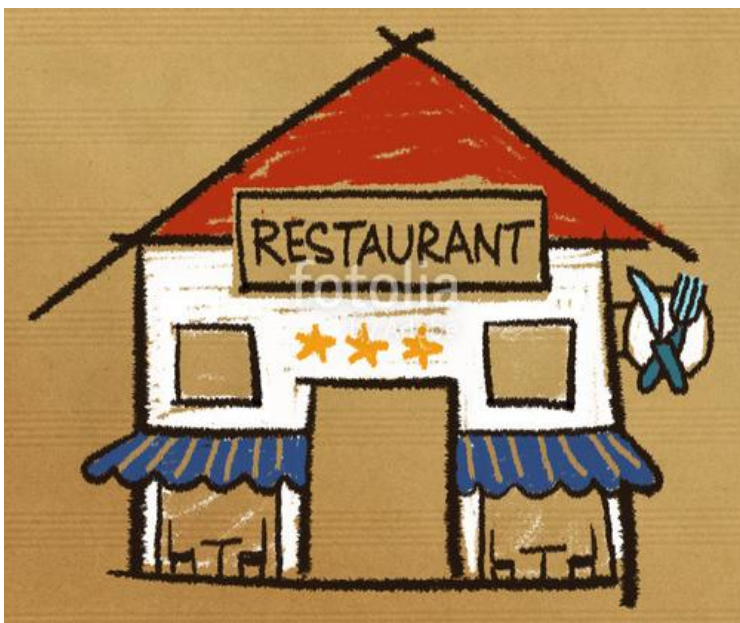
Dog



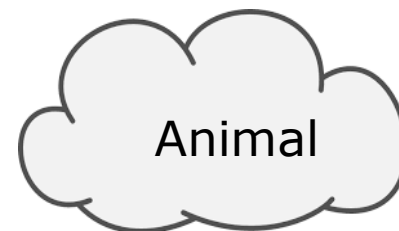
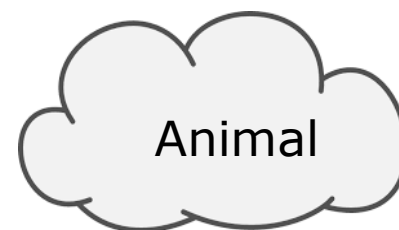
Cat

## 2. 多态从哪来

```
public String eat(Food d);
```



```
public void enterGate(Animal a);
```



a2:

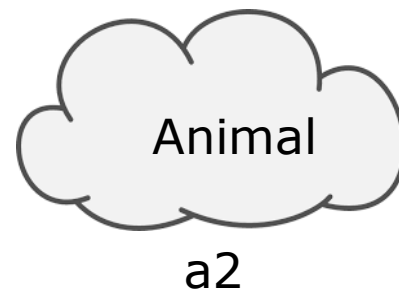
Dog兄，你还在吗？(没有回音)  
恩，我是谁啊？

## 2. 多态从哪来



别琢磨了，你是第一次来吧，新人都有这个毛病，老是爱琢磨自己的具体类。  
请提供你最喜欢的食物？

服务员：“稍等，鱼马上就来”



Dog兄，你还在吗？(没有回音)  
恩，我是谁啊？

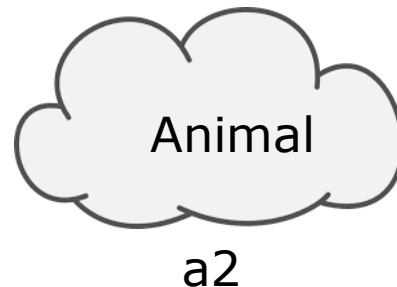
“鱼”（调用getFavoriteFood()方法）

## 2. 多态从哪来



服务员：我不管你是谁，只要你是Animal 就行，你不是有个eat方法吗？

我们这个餐馆，使用的是“**面向接口编程，而不是面向具体类编程**”，要不然，每个动物都判断一下，我们不亏死了。



“你也不问问我到底是谁吗？”

哦，我有点明白那帮家伙把我 new 出来以后，只是给我贴上 Animal 标签的原因了。

```
public class A {  
    public String show(D obj) {  
        return ("A and D");  
    }  
    public String show(A obj) {  
        return ("A and A");  
    }  
}
```

```
public class B extends A{  
    public String show(B obj){  
        return ("B and B");  
    }  
    public String show(A obj){  
        return ("B and A");  
    }  
}
```

```
public class C extends B{ }  
public class D extends B{ }
```

main:

```
A a1 = new A();  
A a2 = new B();  
B b = new B();  
C c = new C();  
D d = new D();  
System.out.println("1--"+a1.show(b));  
System.out.println("2--"+a1.show(c));  
System.out.println("3--"+a1.show(d));  
System.out.println("4--"+a2.show(b));  
System.out.println("5--"+a2.show(c));  
System.out.println("6--"+a2.show(d));  
System.out.println("7--"+b.show(b));  
System.out.println("8--"+b.show(c));  
System.out.println("9--"+b.show(d));
```



## 2. 多态从哪来



### □ 以继承实现多态（IBM刘欣）

```
public abstract class Fruit{
    public String getName(){
        return "Fruit";
    }
    public abstract float getPrice();
}

class Apple extends Fruit{

    public String getName() {
        return "Apple";
    }

    public float getPrice() {
        return 5.0f;
    }
}

Fruit f = new Apple();
f.getPrice() --> 返回5.0f
f.getName() --> 返回 "Apple"
```

Apple a = new Apple();  
a.getPrice() --> 返回5.0f  
a.getName() --> 返回 "Apple"

好像没有必要？

## 2. 多态从哪来



如果没有多态

```
class Apple {
    public String getName() {
        return "Apple";
    }
    public float getPrice() {
        return 5.0f;
    }
}

class Orange {
    public String getName() {
        return "Orange";
    }

    public float getUnitPrice() {
        return 6.0f;
    }
}
```

```
class ShopCart{
    List items = new ArrayList();
    public void addApple(Apple apple){
        items.add(apple);
    }
    public void addOrange(Orange orange){
        items.add(orange);
    }

    public float calculateTotalPrice(){
        float total = 0.0f;
        for(Object o : items){
            if(o instanceof Apple){
                Apple apple = (Apple) o;
                total += apple.getPrice();
            }
            if(o instanceof Orange){
                Orange orange = (Orange) o;
                total += orange.getUnitPrice();
            }
        }
        return total;
    }
}
```


## 2. 多态从哪来



为什么购物车类能变得简单而清晰？

```
class ShopCart{
    List items = new ArrayList();
    public void addApple(Apple apple){
        items.add(apple);
    }
    public void addOrange(Orange orange){
        items.add(orange);
    }

    public float calculateTotalPrice(){
        float total = 0.0f;
        for(Object o : items){
            if(o instanceof Apple){
                Apple apple = (Apple) o;
                total += apple.getPrice();
            }
            if(o instanceof Orange){
                Orange orange = (Orange) o;
                total += orange.getUnitPrice();
            }
        }
        return total;
    }
}
```



```
public class ShopCart {

    List<Fruit> items = new ArrayList<Fruit>();
    public void addFruit(Fruit f){
        items.add(f);
    }

    public float calculateTotalPrice(){
        float total = 0.0f;
        for(Fruit f : items){
            total += f.getPrice();
        }
        return total;
    }
}
```

快递员小张负责小区里所有住户的物资投递，从快递到信件，以及住户订阅的报纸，都在运送之列。请为系统合理建模，使快递员类Postman实现为每天要投递的物资计数和计算总运费收益两项功能。

运费计算方法：

- 快递：重量 \* 1
- 信：重量 \* 5
- 报纸：固定价格

ps：如何得到某个收件人所收物品的总运费？

### □ 以继承实现多态

向上转型、方法重写，以一般代表特殊，以特殊代替一般

```
Animal a = new Cat();
```

```
a.getFavoriteFood();
```

## 2. 多态从哪来



### □ 以接口实现多态

```
public interface Foodable {  
    boolean isDelicious();  
    int getCalory();  
}
```

```
public class Fish implements Foodable {}  
public class Chocolate implements Foodable {}
```

```
Client c = new Client();  
Foodable[] food_list = new Foodable[];  
// Initialization of food_list  
...  
for(Foodable f : food_list){  
    if(f.isDelicious() || f.getCalory() < 100)  
        c.eat(f);  
}
```



忍者神龟武力超群，可以攻击敌人、一些场景道具以及自己人。。。请合理使用Damageable接口，简化玩家Player类的实现。

```
public class Player{  
    private int damage;  
    ...  
    public void hitEnemy(Enemy e){  
        e.getDamage(this.damage);  
    }  
  
    public void collideItem(Item i){  
        i.receiveDamage(this.damage)  
    }  
  
    public void hitFriend(Player p){  
        p.getHit(this.damage);  
    }  
}
```



### □ 多态在方法中的位置

可以在方法的**参数**中传入其父类（或接口）类型，在运行时会根据实际的运行时类型来在方法中进行相应的操作。

多态用于**返回值**，可以在方法的返回值类型上使用其实际返回值的父（接口）类型，不关心返回值的实际类型。

### □ 多态的选择

根据具体问题**合理选择**使用继承和（或）接口；

多态可以使代码变得更通用，以适应需求的变化；

发现变化，封装变化；预判变化，运用抽象。



#### 多态 ( Polymorphism )

- 定义
- 如何实现
- 如何使用

综合练习