多态

北京理工大学计算机学院
金旭亮

两种“香喷喷”的大米

东北大米

泰国香米

“多态性”一词最早用于生物学，指同一种族的生物体虽然具有相同的本质特征，但在不同环境中可能呈现出不同的特性。

狗的多态性

哈士奇　藏獒　贵宾犬　松狮

边境牧羊犬　吉娃娃　德国牧羊犬　秋田犬

蝴蝶犬　博美犬　杜宾犬　柴犬

面向对象开发中的多态

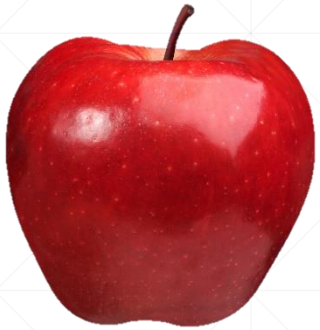
在面向对象理论中，多态是：

同一操作施加于不同的类的实例，不同的类将进行不同的解释，最后产生不同的结果。

从编程角度来看，“多态”表现为：

同样的程序语句，在不同的上下文环境中可能得到不同的运行结果。

多态的实例



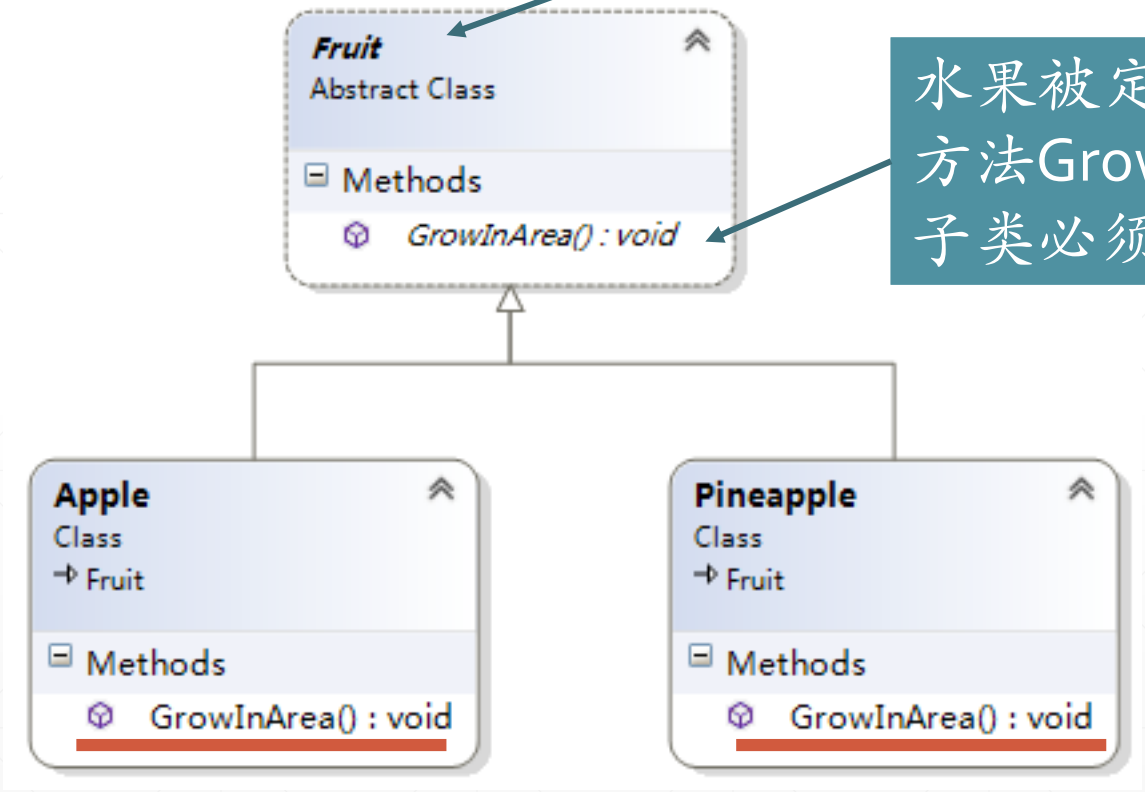
中国的南方、北方均可以种植苹果

菠萝是一种亚热带水果，主要在中国的南方种植

苹果和菠萝都是一种水果，它们都有“适宜种植区域”这个信息值得关注。

为苹果、菠萝建立面向对象的软件模型

抽象类采用“斜体”表示



水果被定义为“抽象类”，其中定义一个抽象方法GrowInArea()，表示“种植区域”，要求子类必须重写。

苹果和菠萝成为Fruit的子类，分别为其抽象方法GrowInArea()提供了不同的实现代码，这种多态编程方式称为“子类重写基类的抽象方法”。

参看示例：Fruit

```
static void ShowPolymorphism()
 {
     Fruit f;

     f = new Apple();
     f.GrowInArea();

     f = new Pineapple();
     f.GrowInArea();
}
```

我是苹果，南方北方都可以种植我。

我是菠萝，喜欢温暖，只能在南方看到我。

相同的一句“f.GrowInArea()”，由于f引用的对象不同，导致其输出结果不同。

“真正的”多态代码

```
//显示特定水果的“适宜种植区域”信息
static void ShowFruitGrowInAreaInfo(Fruit fruit)
{
   fruit.GrowInArea();
}
```

此方法中的代码只调用“基类”中定义的方法，不涉及任何具体的子类，因此，全部都是“多态”代码。

多态代码调用实例

```
//中国哪儿适宜种苹果?
ShowFruitGrowInAreaInfo(new Apple());

//中国哪儿适宜种菠萝?
ShowFruitGrowInAreaInfo(new Pineapple());
```

ShowFruitGrowInAreaInfo()方法可以输出任何一种水果（比如桔子）的“适宜种植地”信息，只要程序中有相应的派生自Fruit类的特定水果类（比如Orange）即可。

多态的代码，只调用“基类”中定义的方法，存取“基类”中定义的字段和属性，简单地说，就是：

针对“基类”编程

多态编程练习

请模仿本讲示例，在现实生活中查找类似的事物，设计一个“抽象基类”-“子类”的继承体系，然后用“多态”特性编写测试代码。

为了更好地理解多态，
下面我们来看一个接近真实的示例.....

在实例中理解多态的含义与用途

假设某动物园管理员每天需要给他所负责饲养的狮子、猴子和鸽子喂食。
我们用一个程序来模拟他喂食的过程。

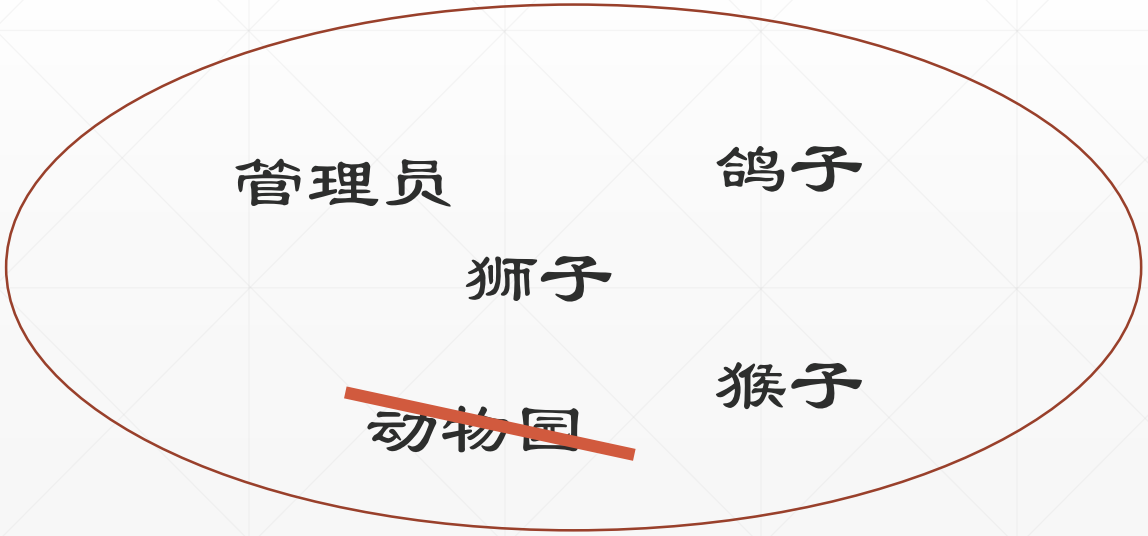
示例： Zoo

面向对象建模中的“名词法”

名词法

用人类的自然语言描述出软件要干的事，挑出其中的名词，它们就是“候选”的“类”。

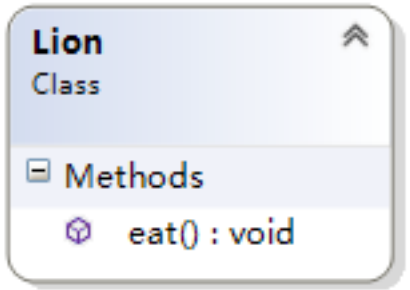
动物园管理员每天需要给他所负责饲养的狮子、猴子和鸽子喂食。

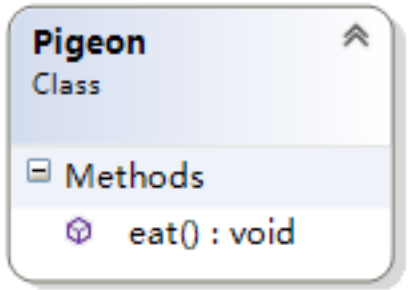


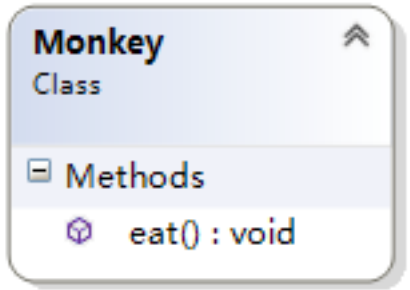
抽取名词

使用“名词法”建立软件模型

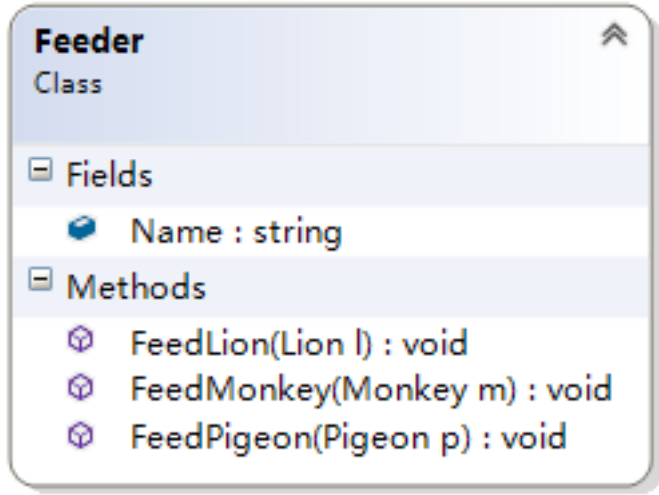
三种动物对应三个类，每个类定义一个eat()方法，表示吃饲养员给它们的食物。







再设计一个Feeder类代表饲养员，其name字段保存饲养员名字，三个方法分别代表喂养三种不同的动物，其参数分别引用三种动物对象。

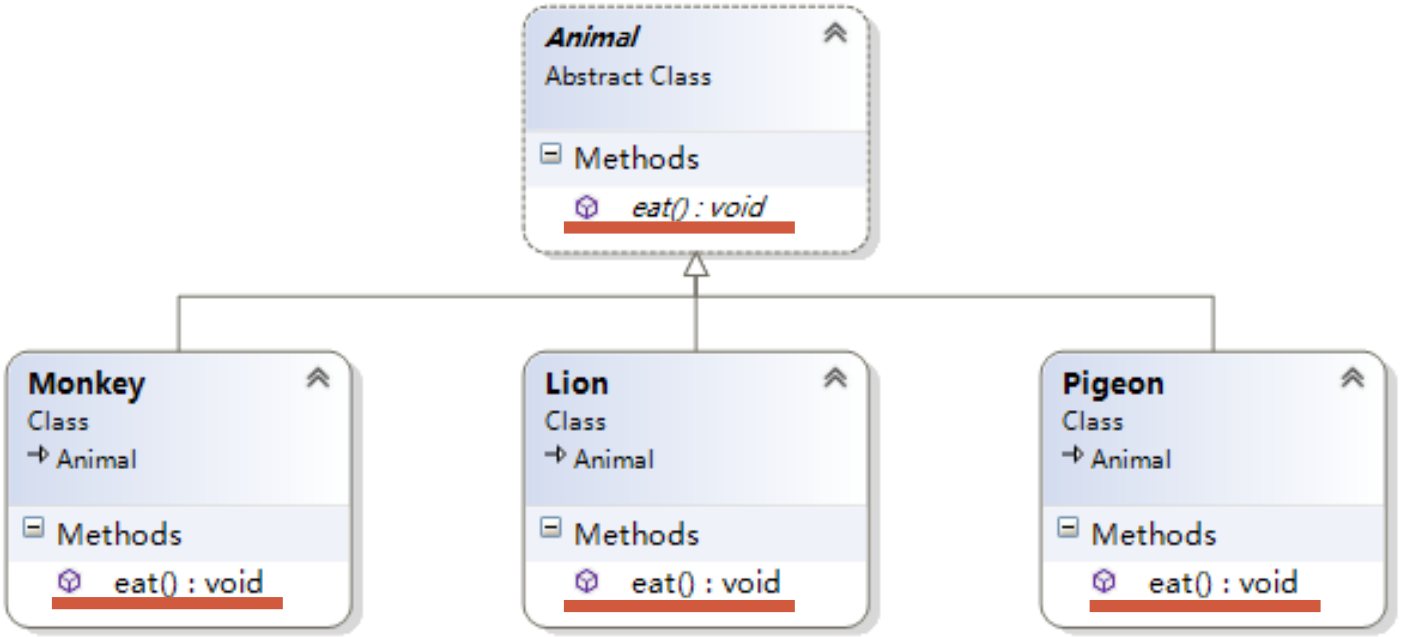


喂食过程

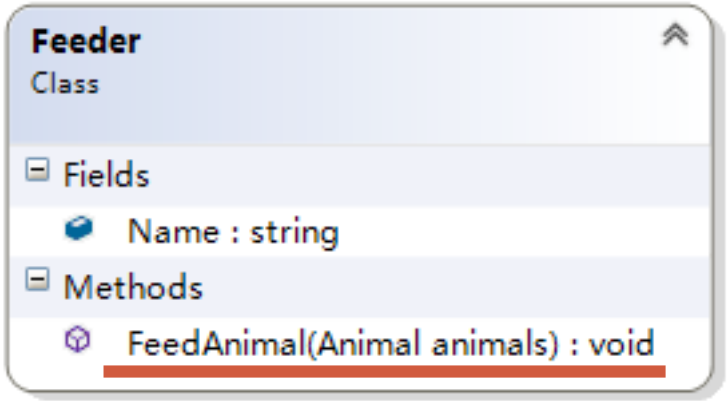
```
0 references
static void Main(string[] args)
{
    Monkey m = new Monkey();
    Pigeon p = new Pigeon();
    Lion l = new Lion();

    Feeder f = new Feeder();
    f.Name = "小李";

    f.FeedMonkey(m);     //喂猴子
    f.FeedPigeon(p);     //喂鸽子
    f.FeedLion(l);       //喂狮子

    Console.ReadKey();
}
```

思索一下：
这种编程方式有什么不合理的地方吗？

程序重构：引入继承



设计抽象类Animal，定义抽象方法eat()，子类重写它



重新设计Feeder类的喂养方法

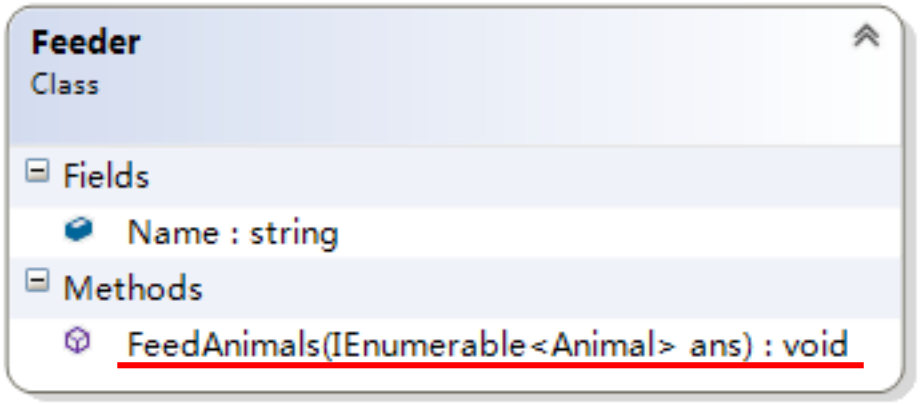
第一次重构结果

```
static void Main(string[] args)
{
    Monkey m = new Monkey();
    Pigeon p = new Pigeon();
    Lion l = new Lion();

    Feeder f = new Feeder();
    f.Name = "小李";

    f.FeedAnimal(m);//喂猴子
    f.FeedAnimal(p);//喂鸽子
    f.FeedAnimal(l);//喂狮子

    Console.ReadKey();
}
```

还可以进一步优化……

再次重构

改喂养**单只**动物为**一群**动物



喂养过程

```
static void Main(string[] args)
{
    //动物数组
    var ans = new List<Animal> {
        new Monkey(), new Pigeon(), new Lion()
    };

    Feeder f = new Feeder();
    f.Name = "小李";
    //喂养一群动物
    f.FeedAnimals(ans);

}
```

“多态”的好处

从这个示例中可以看到，通过在编程中应用多态：

可以使我们的代码具有**更强的适用性**。

当需求变化时，多态特性可以帮助我们**将需要改动的地方减少到最低限度**。

“多态”具体实现方式有两种：

继承多态 使用抽象基类实现

接口多态 使用接口实现

举一反三： 请使用接口，改造“动物园”示例，体会“接口多态”是怎么回事。