



计算机学院

北華航天工業學院
NORTH CHINA INSTITUTE OF AEROSPACE ENGINEERING

第6章 面向对象技术-II

回顾上堂课的内容

◆面向对象的基本概念

- 面向对象的设计思想
- 类之间的关系
- 类和对象

◆类的创建

- 构造方法

◆对象的创建、使用、销毁

◆面向对象的特征

- 封装
- 继承
- 多态

本讲目标

◆方法的重载

◆this关键字

◆static关键字

◆继承

- 继承的概念、特征及优点
- 访问修饰符对类成员的访问限制

◆方法的重写

◆super关键字

◆final关键字

◆多态

- 为什么使用多态
- 多态的概念及分类

方法的重载

方法的重载是指一个类中可以定义有**相同的名字**，但**参数不同**的多个方法。调用时，会根据不同的参数列表选择对应方法。

Note:

- 1、参数不同：
 - 参数的**个数**不同；
 - 参数的**类型**不同。
- 2、方法的**名字一样**，参数的**类型和个数**都一样，但返回值类型不一样，构成方法的重载吗？

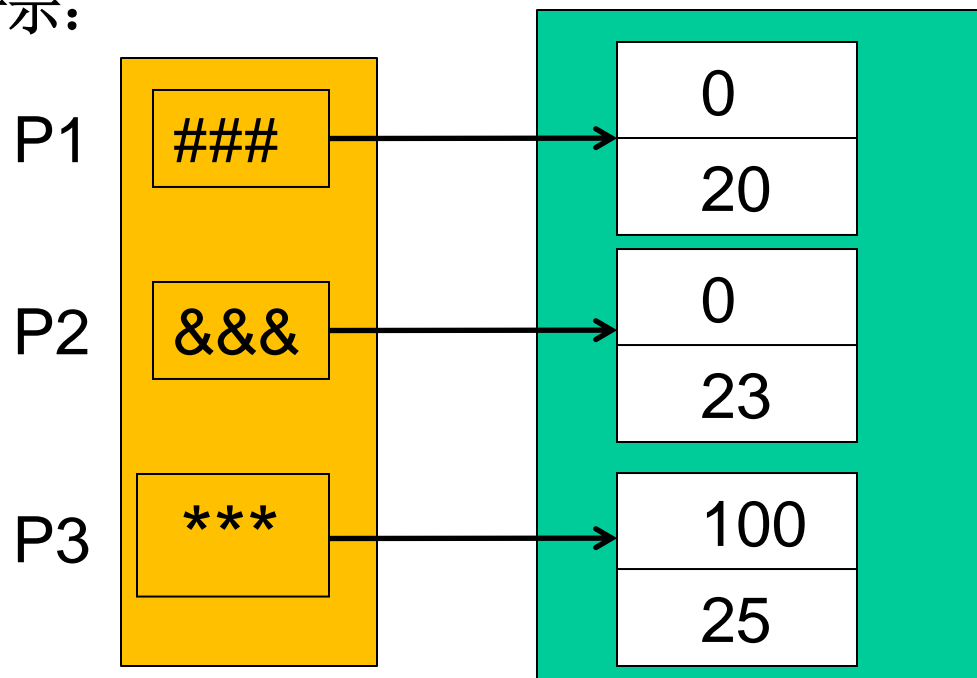
例题：Test.java

构造方法的重载

- 与普通方法一样，构造方法也可以重载；
- 例如，修改Person类的构造方法。

```
Person() {  
    id = 0;  
    age = 20;  
}  
Person(int i) {  
    id = 0;  
    age = i;  
}  
Person(int n, int i) {  
    id = n;  
    age = i;  
}
```

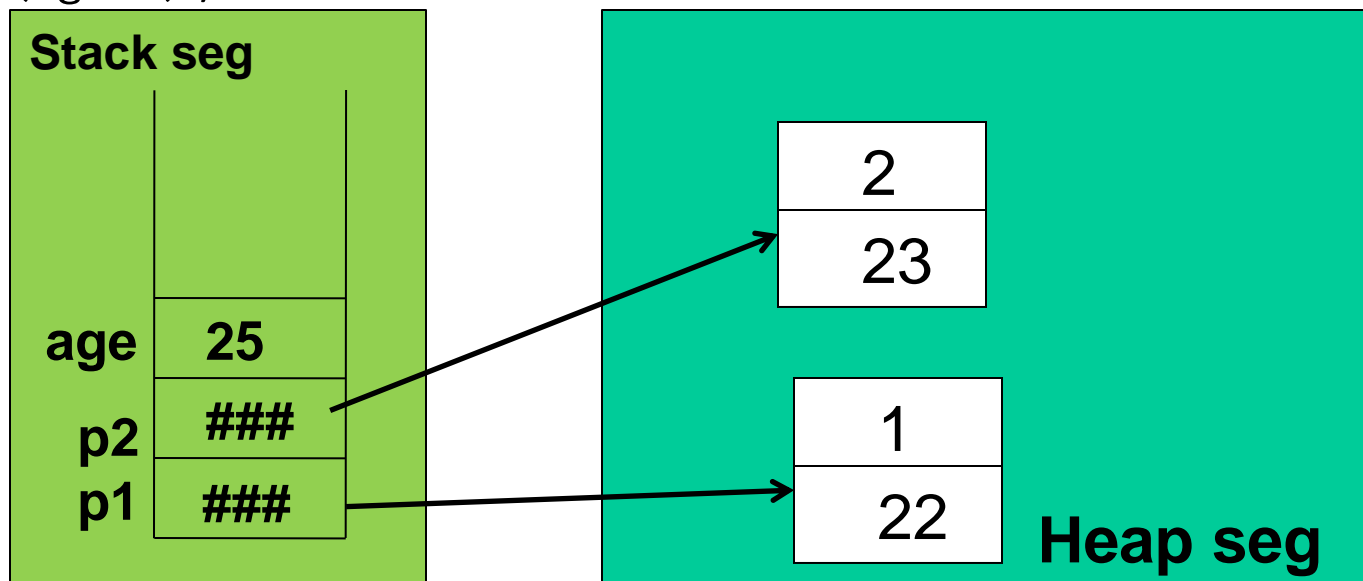
课堂练习：利用修改过的Person类，编写程序，分别用三种构造方法创建三Person对象，如下图所示：



内存分析:

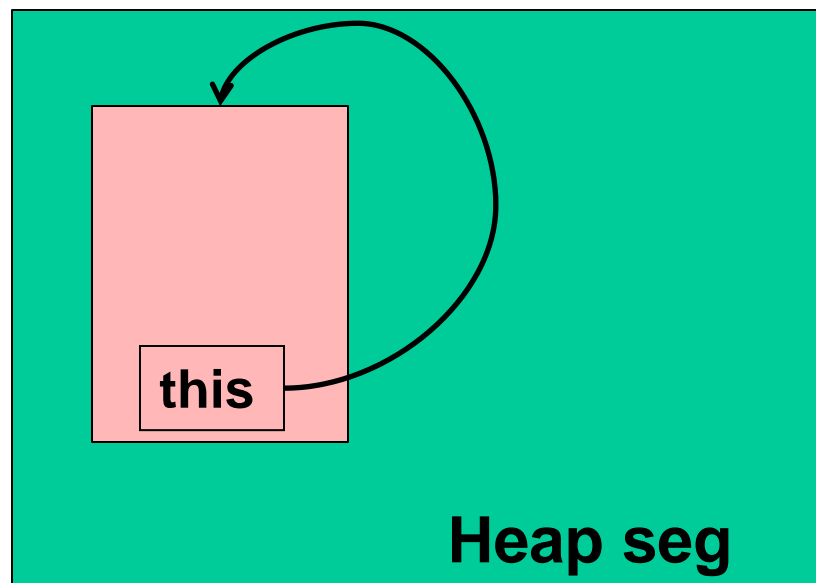
```
public class Test{  
    public static void main(String args[]) {  
        Person p1 = new Person(1, 22);  
        Person p2 = new Person(2, 23);  
        int age = 25;  
        p1.setAge(age);  
        p2.setAge(age++);  
    }  
}
```

例题: **TestOverload.java**



this 关键字

- 在类的**方法定义**中使用的this关键字代表**使用该方法**的对象的引用。
- 当必须指出当前使用方法的对象是谁时要使用this。
- 有时使用this可以处理方法中成员变量和参数重名的情况。
- this可以看作一个变量，它的值是当前对象的引用。



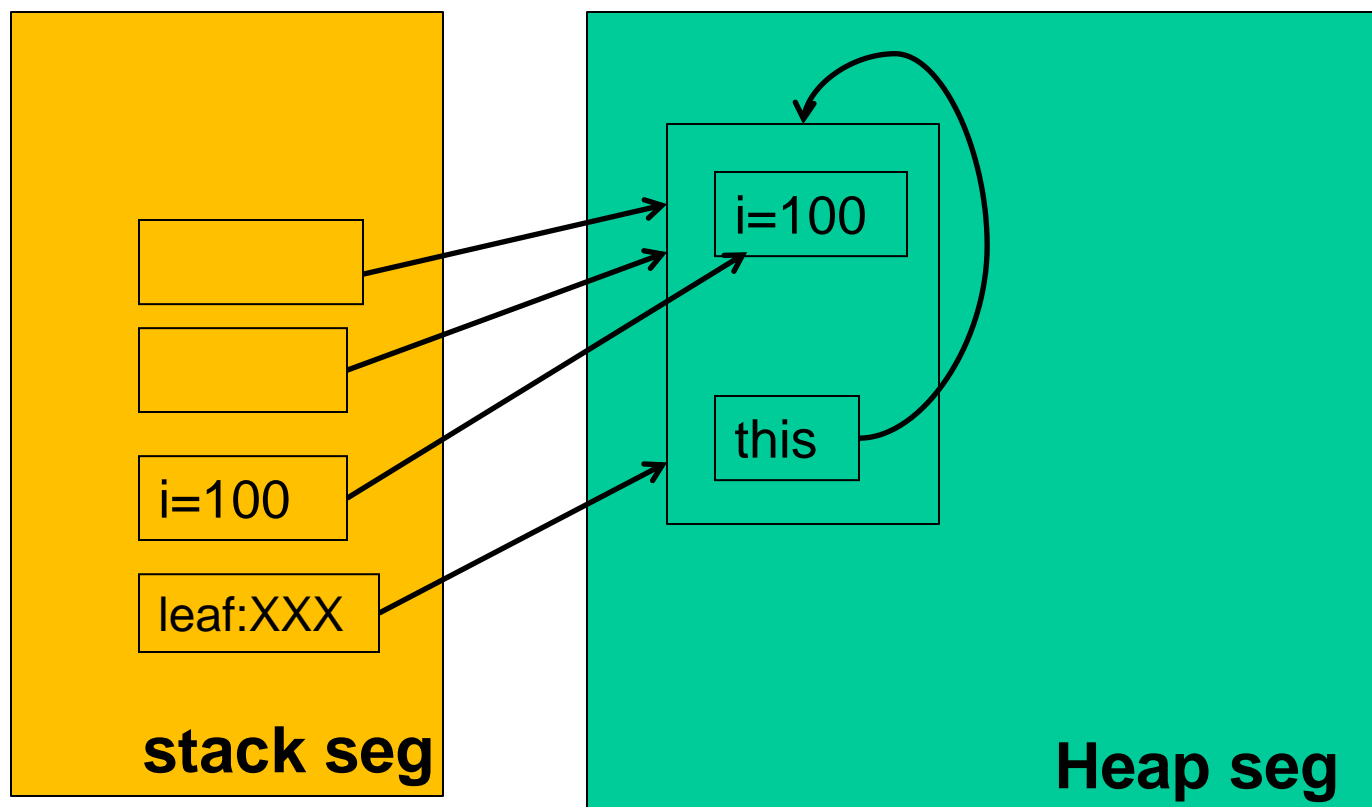
this 关键字

例题:

```
Public class Leaf {  
    int i = 0;  
    Leaf(int i) {this.i = i; }  
    Leaf increament() {  
        i++;  
        return this;  
    }  
    void print() { System.out.println( "i = " +i); }  
    public static void main(String[] args) {  
        Leaf leaf = new Leaf(100);  
        leaf.increament().increament().print();  
    }  
}
```


this 关键字

例题：



static 关键字

- 在类中，用static声明的成员变量为静态成员变量，它为该类的公用变量，在第一次使用时被初始化，对于该类的所有对象来说，static成员变量**只有一份**。
- 用static声明的方法为静态方法，在调用该方法时，不会将对象的引用传递给它，所以static方法中不可访问非static的成员。
※静态方法不再是针对某个对象调用，所以不能访问非静态成员。
- 可以通过对象引用或类名（不需要实例化）访问静态变量(System.out)

例题：Cat.java

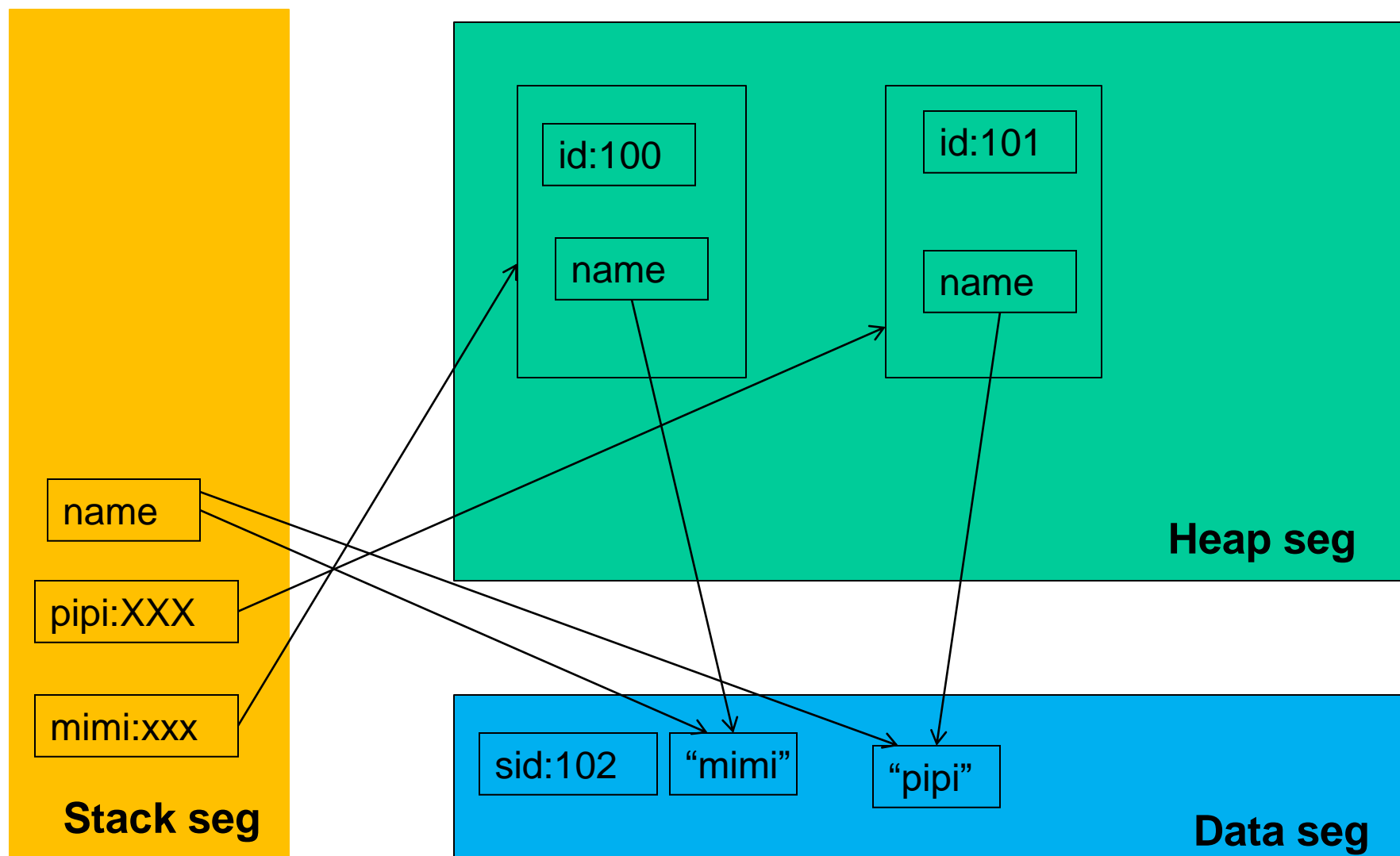
static关键字

Cat.java

```
public class Cat {  
    private static int sid = 0;  
    private String name;  
    int id;  
    Cat(String name) {  
        this.name = name;  
        id = sid++;  
    }  
    public void info() {  
        System.out.println("My name  
is "+name+" No. "+id);  
    }  
}
```

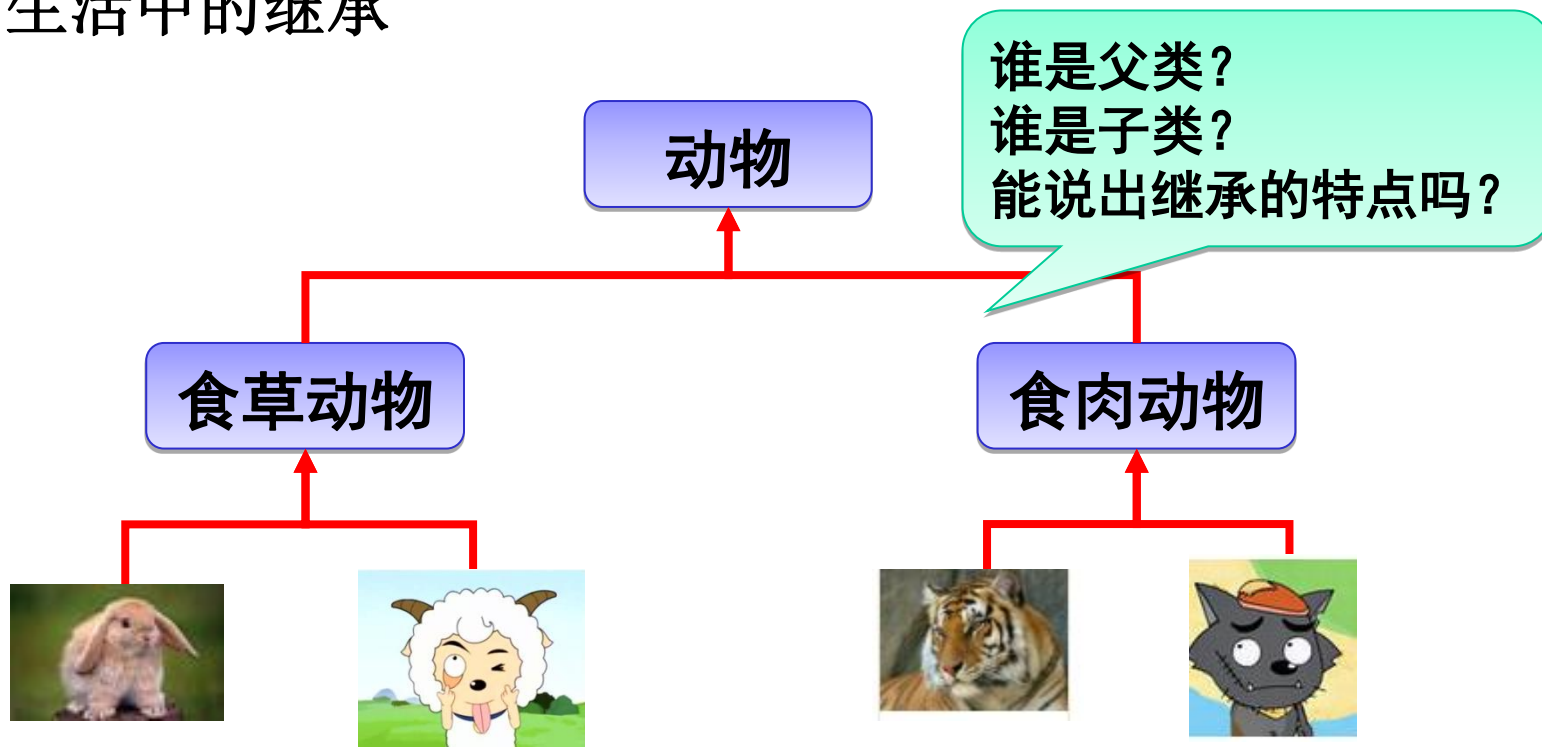
```
public static void main(String arg[]) {  
    Cat.sid = 100;  
    Cat mimi = new Cat("mimi");  
    Cat pipi = new Cat("pipi");  
    mimi.info();  
    pipi.info();  
}
```

static关键字



继承

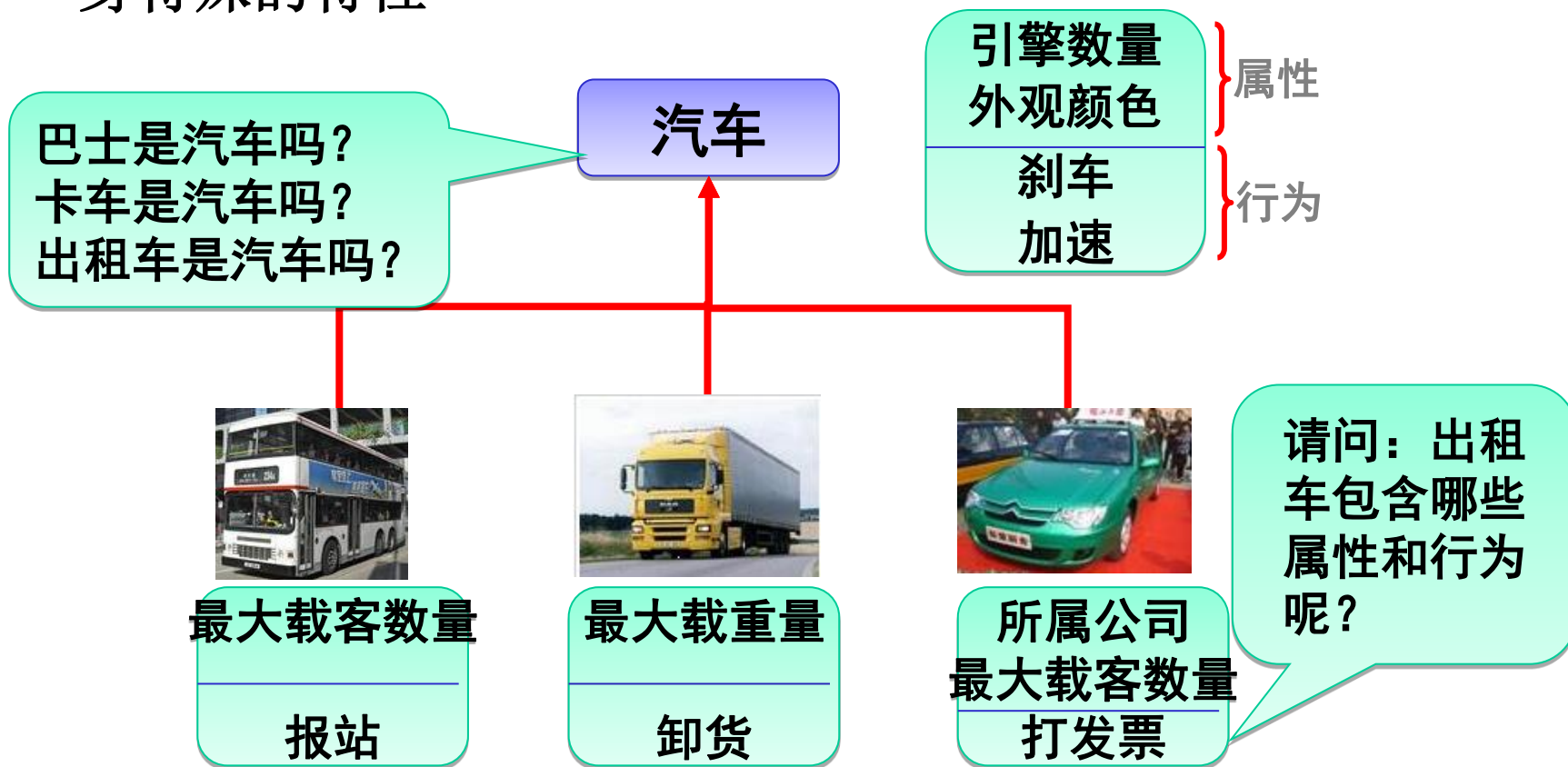
◆生活中的继承



继承需要符合的关系：**is-a**，父类更通用、子类更具体

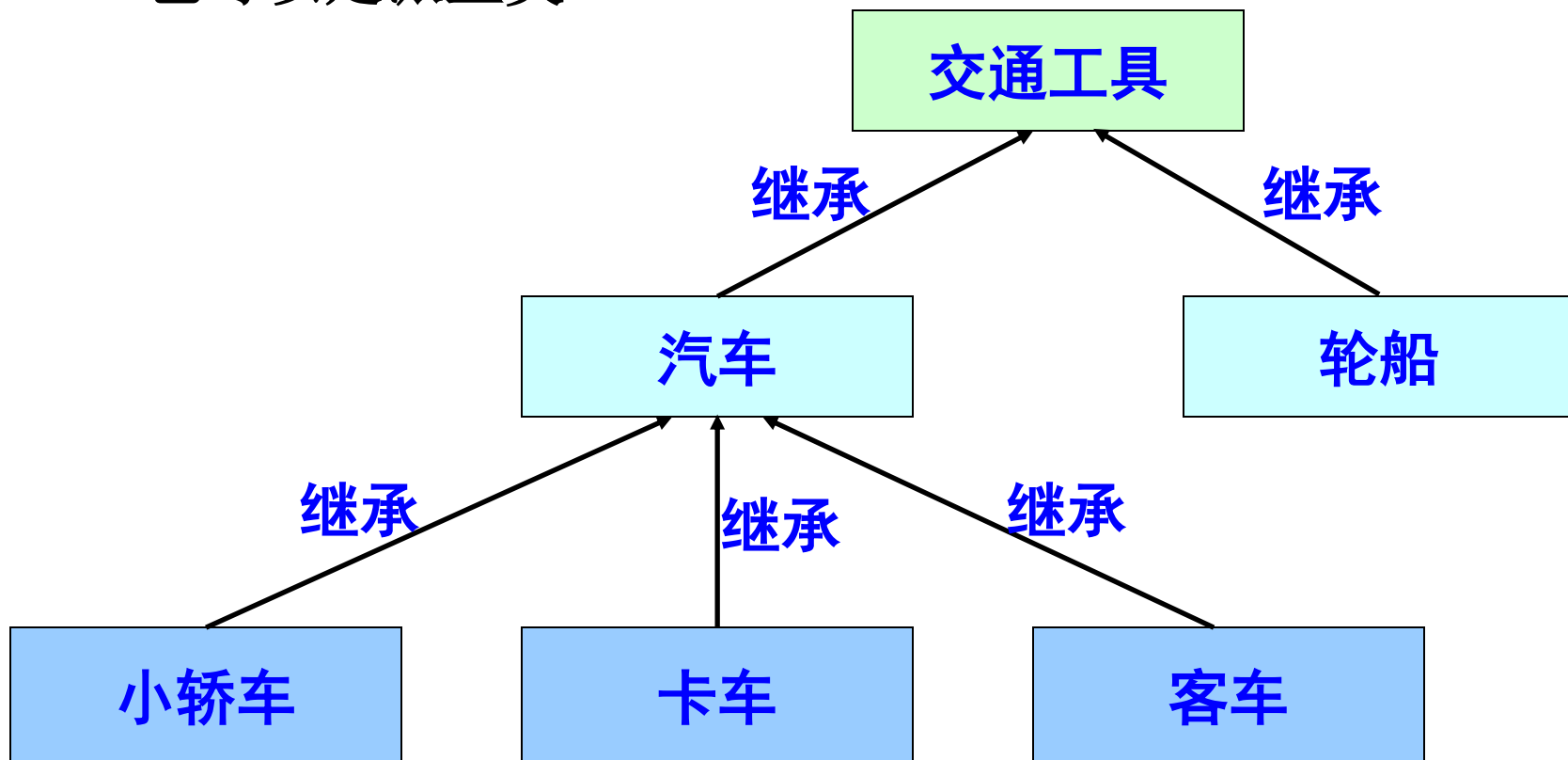
生活中的继承(续)

- ◆ 子类具有父类的一般特性（包括属性和行为），以及自身特殊的特性



基类和派生类

◆ 基类和派生类的概念是相对的，一个类既可以是基类，也可以是派生类



为什么需要继承

◆描述两种动物羊Sheep和狼Wolf，各自的要求如下：

● Sheep

□ 属性：名称，重量

□ 方法：吃饭(吃草)、睡觉、呼吸、动物描述

● Wolf

□ 属性：名称，重量

□ 方法：吃饭(吃肉)、睡觉、呼吸、动物描述

为什么需要继承(续)



◆ 以下这个解决方案中存在着什么问题？

```
public class Sheep {  
    {  
        private String name;// 名称  
        private int weight;// 重量  
    }  
    public Sheep(String sname, int sweight) {  
        name= sname;  
        weight = sweight;  
    }  
    public void eat() {  
        System.out.println ("I'm hungry");  
        System.out.println("I like to eat grass");  
    }  
    {  
        public void sleep() {  
            System.out.println("I'm sleeping");  
        }  
        public void breathe() {  
            System.out.println("I'm breathing");  
        }  
        public void info(){  
            System.out.println("Hi,I'm a "+name);  
            System.out.println("My weigth is"+weight);  
        }  
    }  
}
```

为什么需要继承(续)

```
public class Wolf{  
    {  
        private String name;// 名称  
        private int weight;// 重量  
    }  
    {  
        public Wolf(String sname, int sweight) {  
            name= sname;  
            weight = sweight;  
        }  
    }  
    {  
        public void eat() {  
            System.out.println("I'm hungry");  
            System.out.println("I like to eat meat");  
        }  
    }  
    {  
        public void sleep() {  
            System.out.println("I'm sleeping");  
        }  
    }  
    {  
        public void breathe() {  
            System.out.println("I'm breathing");  
        }  
    }  
    {  
        public void info(){  
            System.out.println("Hi,I'm a "+name);  
            System.out.println("My weighth is"+weight);  
        }  
    }  
}
```

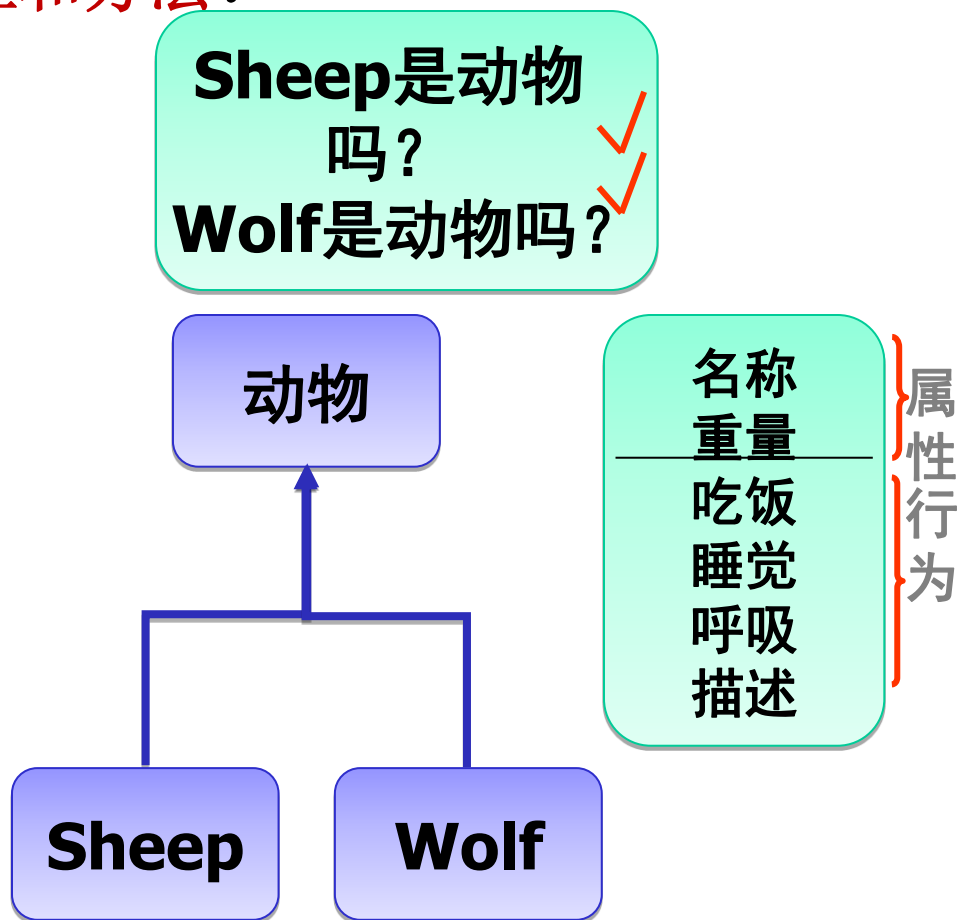
为什么需要继承(续)

◆存在的问题

两段代码存在重复

为什么需要继承(续)

- ◆ 如何改进？有没有可能建立继承关系，让子类自动继承父类的属性和方法？



JAVA中的继承

◆ 语法

```
class subclass-name extends superclass-name {  
    //类体  
}
```

◆ 特征

- 父类通用，它具有更一般的特征和行为，较为抽象。子类具体，它除了具有父类的特征和行为之外，还具有自己的特征和行为
- 在继承关系中，父类和子类必须满足is-a的关系。
- 父类叫基类、超类，子类叫派生类。
- 如果子类没有写extends，则默认该子类的父类为Object
- Java中 **只支持单继承**
- 子类不能继承父类中访问权限为private的成员变量和方法。

JAVA中的继承

◆优点

- 通过继承，能够复用原有代码，节省编程时间，并能够减少程序代码出错的可能性。
- 在JAVA编程语言中，通过继承可利用已有的类，并扩展它的属性和方法。这个已有的类可以是语言本身提供的或其他程序员编写的。
- JAVA中所有的类（包括我们自己定义的类）都是直接或间接继承自java.lang.Object类。



定义父类Animal及其子类Sheep、Wolf类

```
public class Animal {  
    private String name;//名称  
    private int weight;//重量  
  
    public Animal(String sname,int sweight){  
        name = sname;  
        weight =sweight;  
    }  
  
    public void eat() {  
        System.out.println("I'm hungry");  
    }  
  
    public void sleep() {  
        System.out.println("I'm sleeping");  
    }  
  
    public void breathe() {  
        System.out.println("I'm breathing");  
    }  
  
    public void info(){  
        System.out.println("Hi,I'm a "+name);  
        System.out.println("My weighth is"+weight);  
    }  
}
```

现在，所有子类中相同的代码都抽取到父类中了



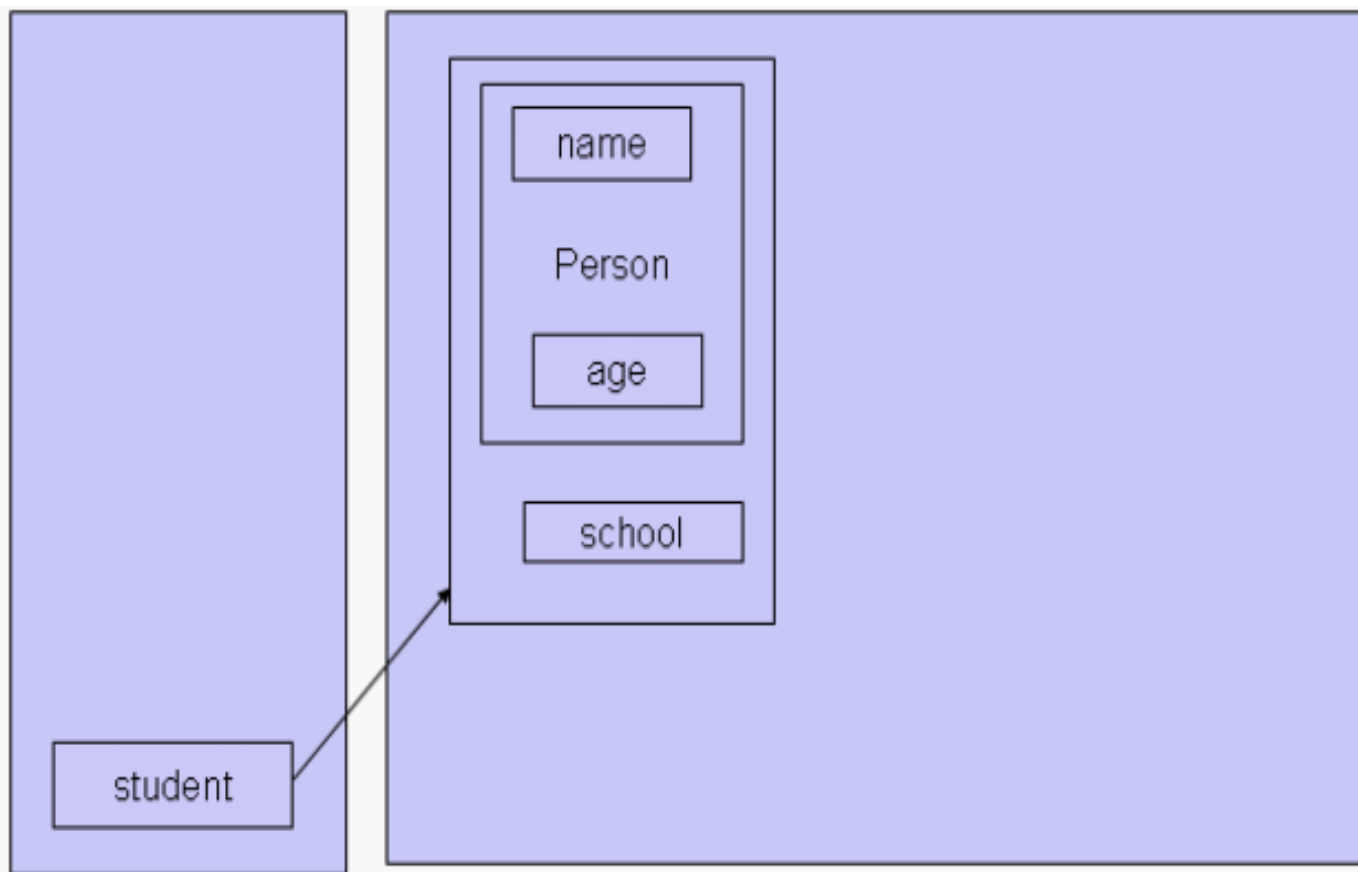
```
public class Sheep extends Animal {  
    public Sheep (String sname, int sweight){  
        super(sname,sweight);  
    }  
  
    public void eat(){  
        System.out.println("I like to eat grass")  
    }  
}
```

由于**eat()**方法与父类的实现有所不同，因此**重写**了该方法

子类自动继承父类的属性和方法，子类中不再存在重复代码

```
public class Wolf extends Animal {  
    public Wolf (String sname, int sweight) {  
        super(sname,sweight);  
    }  
  
    public void eat(){  
        System.out.println("I like to eat meat");  
    }  
}
```


例题：TestPerson.java





Lab6_1:编写一个应用程序，设计一个汽车类 Vehicle，包含的属性有车轮个数 wheels和车重weight。小车类Car 是Vehicle的子类，其中包含的属性有载人数 loader。卡车类 Truck 是Car类的子类，其中包含的属性有载重量 payload。每个类都有构造方法和输出相关数据的方法 。编写测试类CarTest进行测试。

JAVA中的访问控制

◆访问修饰符对类成员的访问限制

访问修饰符 访问权限	<code>private</code>	缺省的 (<code>default</code>)	<code>protected</code>	<code>public</code>
当前类	√	√	√	√
同一包中的类(和继承无关系)	×	√	√	√
不同包中的子类	×	×	√	√
任何类	×	×	×	√

Note: 对class的权限修饰只可以用public和default。

1、public可以在任意地方被访问

2、default只可以被同一个包内部类访问。

方法的重写

- 在子类中可以根据需要对基类中继承的方法进行重写。
- 重写方法必须和被重写方法具有**相同方法名称、参数列表和返回类型**。
- 重写方法不能使用比被重写方法更**严格的访问权限**。

例题：TestOverWrite.java

super关键字

◆ super

- java中通过super来实现对父类成员的访问，super用来引用当前对象的父类。
- 每个子类构造方法的第一句，都隐含了调用super()。

◆ 语法：

- super(参数列表)：调用父类的构造方法
- super. 属性名：调用子类继承的父类属性
- super. 方法：调用子类继承的方法

例题：TestInherit.java

继承中的构造方法

- ◆ 子类的构造的过程中**必须**调用其基类的构造方法
- ◆ 子类可以在自己的构造方法中使用`super(argument_list)`调用基类的构造方法
 - 使用`this(argument_list)`调用本类另外的构造方法。
 - 如果使用`super()`，必须写在子类构造方法的第一行。
- ◆ 如果子类的构造方法中没有显示地调用基类的构造方法，则系统默认调用基类无参的构造方法。
- ◆ 如果子类的构造方法中既没有显式的调用基类的构造方法，而基类中又没有无参的构造方法，则编译出错。

例题：TestSuperSub.java



```
public class Sheep extends Animal {  
    public Sheep (String sname, int sweight){  
        super(sname,sweight);  
  
    }  
  
    public void eat(){  
        System.out.println("I like to eat grass");  
    }  
}
```

main() 方法

- main() 方法就是一个静态的方法，main() 方法也是一个特殊的方法，在Java应用程序中可以有許多类，每个类也可以有許多方法。但解释器在装入程序后首先运行的是main() 方法。

- 格式如下：

```
public static void main(String [] args)
{
    //方法体定义
    .....
}
```


main() 方法

- 其中：

- 1) 声明为public以便解释器能够访问它。
- 2) 修饰为static是为了不必创建类的实例而直接调用它。
- 3) void 表明它无返回值。
- 4) 它带有一个字符串数组参数。

main() 方法

- 我们只能从装入运行程序的命令行传递参数给它。其一般格式如下：

java 程序名 实参列表

例： java HelloWorld "hello world"

final关键字

◆ final关键字

- final修饰类：是最终类，不能被其他类所继承

```
public final class String {}
```

- final修饰变量：（成员变量或局部变量）即为常量，只能赋值一次，只能访问，不能修改其值。而且通常变量名用大写字母。如：

```
public static final int INCREMENT=5
```

- final修饰成员方法：是最终方法，不能被子类重写。

```
public final void display() {};
```

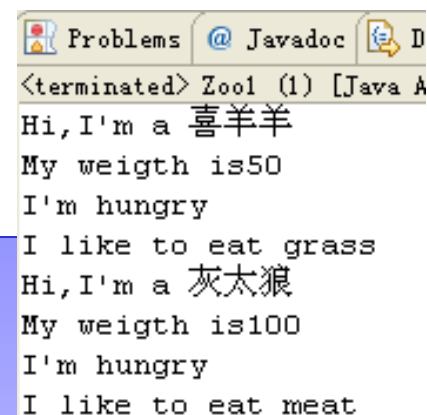
为什么需要多态

- ◆ 在Animal类的基础上，开发一个类代表动物园，负责对饲养动物，包括：
 - 1、查看动物的状态
 - 2、查看动物信息

为什么需要多态

◆ 第一种解决方案

```
public class Zoo {  
    public void CheckAnimal(Sheep s) {  
        s.info();  
        s.eat();  
    }  
    public void CheckAnimal(Wolf w) {  
        w.info();  
        w.eat();  
    }  
    public static void main(String[] args) {  
        Zoo z = new Zoo();  
        z.CheckAnimal(new Sheep("喜羊羊", 50));  
        z.CheckAnimal(new Wolf("灰太狼", 100));  
    }  
}
```



```
Problems @ Javadoc D  
<terminated> Zool (1) [Java A  
Hi, I'm a 喜羊羊  
My weight is 50  
I'm hungry  
I like to eat grass  
Hi, I'm a 灰太狼  
My weight is 100  
I'm hungry  
I like to eat meat
```

为什么需要多态

- ◆现在，升级Animal类，增加一种新动物： Panda，该动物吃的食物是竹子，要求如下：
 - 属性：名称、重量
 - 方法：吃饭(吃竹子)、睡觉、呼吸、动物描述
- ◆同时，动物园也需要查看该动物的情况。

为什么需要多态

- ◆ 如果在第一种解决方案的基础上去解决，我们需要如下两个步骤：
 - 建立Panda类
 - 动物园类，增加方法：CheckAnimal (Panda t)



能找到这个解决方案的弊端吗？

每增加一种新的动物类型，都要修改动物园类，增加相应的 **CheckAnimal**（查看）方法，**代码的可扩展性及可维护性极差**

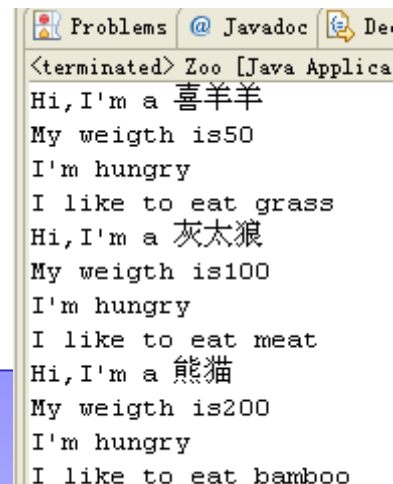
为什么需要多态

◆第二种解决方案——使用多态

```
public class Zoo {  
    public void CheckAnimal(Animal an) {  
        an.info();  
        an.eat();  
    }  
    public static void main(String[] args) {  
        Zoo z = new Zoo();  
        z.CheckAnimal(new Sheep2("喜羊羊", 50));  
        z.CheckAnimal(new Wolf2("灰太狼", 100));  
        z.CheckAnimal(new Panda("熊猫", 200));  
    }  
}
```

可以接收子类类型

根据实际创建的对象类型调用相应方法



```
Problems @ Javadoc De  
<terminated> Zoo [Java Applica  
Hi, I'm a 喜羊羊  
My weigth is50  
I'm hungry  
I like to eat grass  
Hi, I'm a 灰太狼  
My weigth is100  
I'm hungry  
I like to eat meat  
Hi, I'm a 熊猫  
My weigth is200  
I'm hungry  
I like to eat bamboo
```

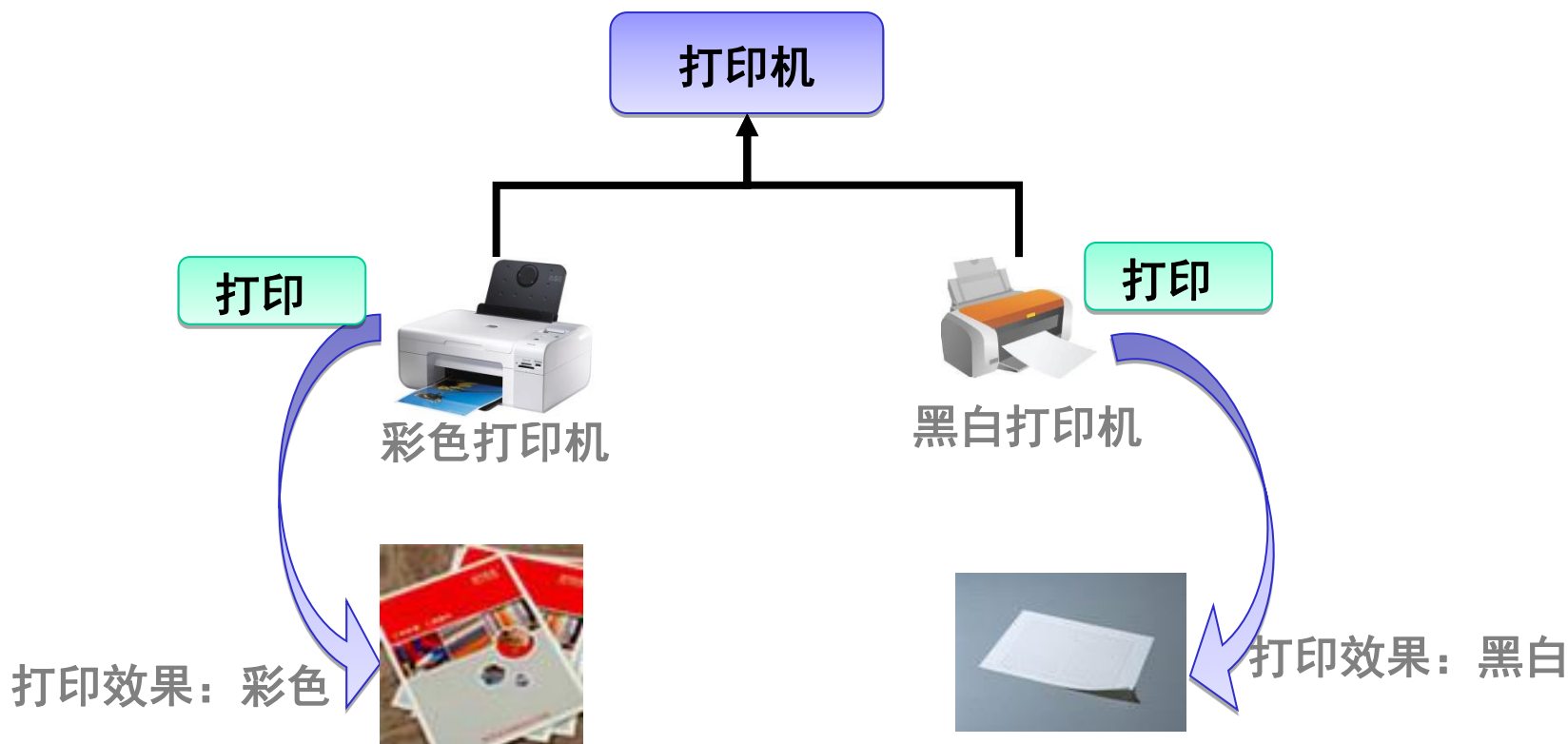
使用多态之后，当需要增加新的子类类型时，无需更改总的调用类，程序的可扩展性及可维护性增强

什么是多态

◆简单来说，多态是具有**表现多种形态**的能力的特征

同一个实现接口

使用不同的实例而执行不同操作



多态的分类

◆ 方法重载

● 第一层含义

方法重载，即在一个类中通过多种方式完成同一种行为。如几何图形的“绘图”方法，通过多种方式都可以完成绘图。

● 第二层含义

方法重载又叫编译时多态，即在编译阶段，具体调用哪个被重载的方法，编译器会根据参数的不同来静态确定调用相应的方法。

多态的分类

◆ 重载的特征

- 同一个类中多个方法有相同的名字，不同的参数列表（参数类型不同或参数顺序不同）。
- 返回类型不同并不足以构成方法重载。

◆ 示例：

```
public void add(int a, int b) {  
    int c = a + b;  
    System.out.println("两个整数相加得 " + c);  
}  
  
public void add(float a, float b) {  
    float c = a + b;  
    System.out.println("两个float型浮点数相加得" + c);  
}
```

多态的分类

◆ 方法重写

● 第一层含义

指在父类中定义的属性或方法被子类继承之后，可以具有不同的数据类型或表现出不同的行为。这使得同一个属性或方法在父类及其各个子类中具有不同的语义。例如：“几何图形”的“绘图”方法，“椭圆”和“多边形”都是“几何图”的子类，其“绘图”方法功能不同。

● 第二层含义

方法重写又叫运行时多态，即在java运行时，系统根据调用该方法的实例，来决定调用哪个方法。

对子类的一个实例，如果子类重写了父类的方法，则运行时系统调用子类的方法。

如果子类继承了父类的方法（未重写），则运行时系统调用父类的方法。

多态的分类

◆ 方法重写的特征

- 在类层次结构中，如果子类中的一个方法与父类中的方法有相同的方法名并具有相同数量和类型的参数列表，即方法签名相同，这种情况称为方法覆盖。

如何实现多态

- ◆ 第一步：子类重写父类的方法
- ◆ 第二步：编写方法时，使用父类定义的方法
- ◆ 第三步：运行时，根据实际创建的对象类型动态决定使用哪个方法

关键字 instanceof

- ◆ instanceof 是 Java 的一个二元操作符。它的作用是测试它左边的对象是否是它右边的类的实例，返回 boolean 类型的数据。可以用在 **继承中的子类的实例是否为父类的实现**

```
Animal an = new Animal();
Sheep2 sheep = new Sheep2();
if(an instanceof Animal)
{
    System.out.println("an is Animal's instance");
}
if(sheep instanceof Sheep2)
{
    System.out.println("sheep is Sheep2's instance");
}
if(sheep instanceof Animal)
{
    System.out.println("sheep is Animal's instance");
}
```

内部类

◆定义

在类中定义的类型称为内部类，内部类之外的类称为外部类

◆语法：

```
修饰符  class  外部类名称 {  
    //外部类成员  
    修饰符  class  内部类名称 {  
        //内部类成员  
    }  
}
```

◆分类

- 静态的内部类，非静态的内部类及匿名类

内部类

◆特征

- 内部类可以访问其外部类的所有变量和方法，并能够以和外部类的其他非静态成员相同的方式直接引用它们
- 内部编译后生成的class文件为：外部类\$内部类.class

◆优点

- 可以实现多重继承
- 可以方便的访问外部类的私有属性



```
public class Outer {  
    private String info="hello world";  
    class Inner{  
        public void print() {  
            System.out.println("outer class :"+info);  
        }  
    }  
}  
  
public class TestOuter {  
    public static void main(String[] args) {  
        Outer outer=new Outer();  
        Outer.Inner inner=outer.new Inner();  
        inner.print();  
    }  
}
```



总结

◆ static与final

◆ 继承

- Java中只支持单继承
- 在继承关系中，父类更通用，子类更具体。父类具有一般的特征和行为，而子类除了具有父类的特征和行为外，还具有一些自己的特殊的特征和行为
- 使用继承可以实现代码的复用

◆ this与super

◆ 多态

- 多态是具有表现多种形态的能力的特征。更术语化的说法是：同一个实现接口，使用不同的实例而执行不同的操作
- 多态分为编译时多态（重载）和运行时多态（重写）

◆ 内部类

- 在类的内部定义类称为内部类
- 内部类分为静态内部类、非静态内部类和匿名类



计算机学院

北华航天工业学院

NORTH CHINA INSTITUTE OF AEROSPACE ENGINEERING

谢谢!