Lesson5

面向对象的三大特称之二：继承(inheritance)

“is a”

主讲老师：申雪萍

主要内容

- 继承的必要性
- 理解继承
  - 基类与派生类的关系
  - 单继承
  - 多继承
  - 继承的层次结构
  - 子类继承了什么？
  - 继承的好处
- Java继承
  - 子类构造方法
  - 向上映射
  - **进一步理解封装**
  - 保护成员的使用
  - **隐藏**
  - **覆盖**
- **复合与继承**

继承的必要性

基于下面的内容设计类：

- 动物：吃（草）和站着睡觉
- 狮子：吃（肉）、躺着睡觉
- 地鼠：吃（肉），躺着睡觉，并且还可以钻洞的动物

问题：

- 如何创建Animal类
  - 吃（草）和站着睡觉
- 如何创建Lion类
  - 吃（肉）、躺着睡觉
- 如何创建Mouse类
  - 吃（肉），躺着睡觉，并且还可以钻洞的动物

Answer:

- 定义动物类，具有吃饭和睡觉两个方法。
- 嗯，下一步：直接实例化这个Animal对象，就可以得到吃草的并且是站着睡觉的动物。

```
class Animal {
    public void eat() {
        System.out.println("动物吃草ing");
    }
    public void sleep() {
        System.out.println("动物站着睡觉ing");
    }
}
```

Answer:

- 同理，创建Lion类：

```
1  class Lion{
2
3      eat(){
4      Sysotem.out.printf("我在吃肉");
5      }
6      sleep(){
7      System.out.printf("我在躺着睡觉");
8      }
```

Answer:

- 同理，创建Mouse类：

```
1
2  class Mouse{
3
4      eat(){
5      Sysotem.out.printf("我在吃肉");
6      }
7      sleep(){
8      System.out.printf("我在躺着睡觉");
9      }
10     bore(){
11     System.out.printf("我在愉快地钻洞");
12     }
```

分析解决方案的利弊

- 类之间的关系欠考虑；
- 代码重复，代码臃肿；
- 代码可维护性差，未来有更多的动物加入这个大家庭怎么办？
- 违背了我们当初设计面向对象编程语言的初衷。
  - 低耦合
  - 高可维护
  - 高可扩展
  - 代码复用
- 所以，我们要对类进行下一步的优化，就是再抽象操作，或者说刚才的解决方案所做的抽象不足够好。

用继承解决上述问题

我们能否再把类进行抽象一下，抽取一些相同的特质，组成一个动物类呢？

答案是：可以的，抽象提取出合适的父类，通过继承解决上述问题

这不是最好的解决方案，有缺陷

```
class Animal {
    public void eat() {
        System.out.println("动物吃草ing");
    }
    public void sleep() {
        System.out.println("动物站着睡觉ing");
    }
}
```

```
class Lion extends Animal {
    //重写相应的方法
    public void eat() {
        System.out.println("狮子在吃肉");
    }
    //重写相应的方法
    public void sleep() {
        System.out.println("狮子在躺着睡觉");
    }
}
```

用继承解决上述问题

```
class Mouse extends Animal {
    //重写相应的方法
    public void eat() {
        System.out.println("老鼠在吃肉");
    }
    //重写相应的方法
    public void sleep() {
        System.out.println("老鼠在躺着睡觉");
    }
    //添加新的方法
    public void bore() {
        System.out.println("老鼠在愉快地钻洞");
    }
}
```

```
class Giraffe extends Animal {
    //增加
    public void run() {
        System.out.println("长颈鹿四条腿走路");
    }
    //覆盖
    public void eat() {
        System.out.println("长颈鹿在愉快的吃草");
    }
}
```

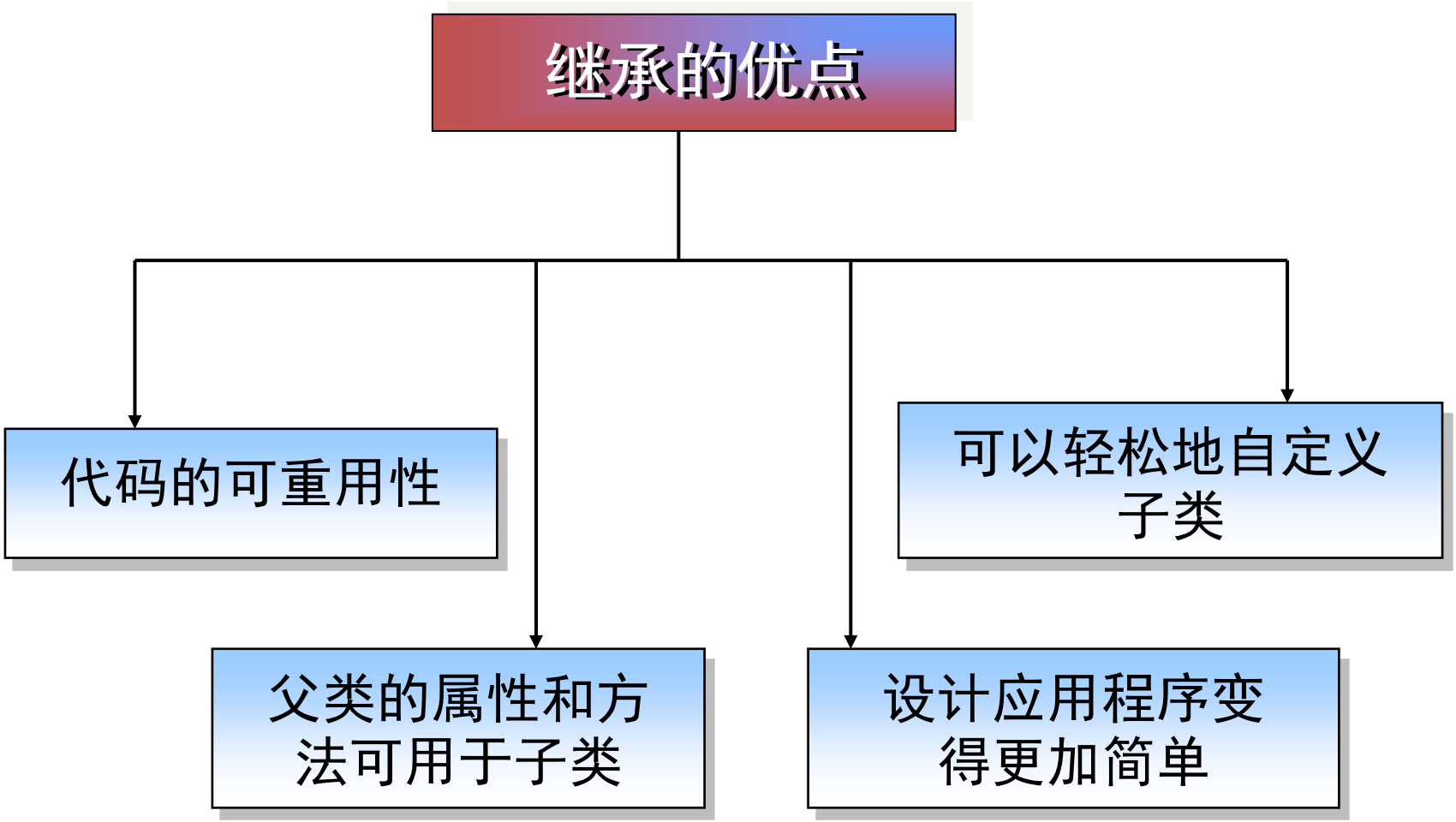
测试类：

```
public class AnimalTest0 {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Mouse aMouse = new Mouse();
        Giraffe aGiraffe = new Giraffe();
        Lion aLion = new Lion();
        aMouse.eat();
        aMouse.sleep();
        aLion.eat();
        aLion.sleep();
        aGiraffe.eat();
        aGiraffe.sleep();
    }

}
```

测试类：com.buaa.inheritance

老鼠在吃肉
老鼠在躺着睡觉
狮子在吃肉
狮子在躺着睡觉
长颈鹿在愉快的吃草
动物站着睡觉ing

为什么使用继承？



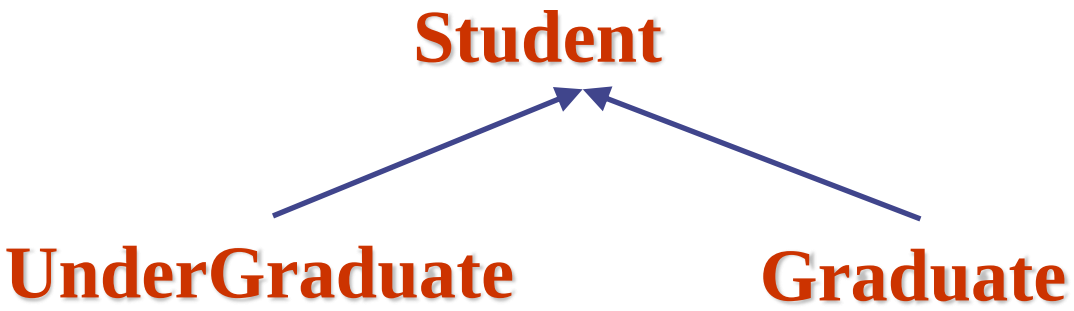
理解继承

主要内容

- 继承的必要性
- 理解继承
  - 基类与派生类的关系
  - 单继承
  - 多继承
  - 继承的层次结构
  - 子类继承了什么？
  - 继承的好处
- Java继承
  - 子类构造方法
  - 向上映射
  - 进一步理解封装
  - 保护成员的使用
  - 变量隐藏
  - 方法覆盖
- 复合与继承

继承是一种“is-a” 的关系

本科生和研究生都有姓名、学号、地址，而且都可以进行选课，**他们都是学生**



继承是一种“is-a” 的关系

继承是一种“is-a” 的关系

- 这种is-a关系，利用继承来实现
  - A is a B (或者A is kind of B)
  - A继承B
- 例如：
  - DateTime(日期时间类)既是Time(时间类)，也是Date(日期类)
  - 食肉动物（Carnivore类）是动物（Animal类），食草动物（Herbivore类）也是动物（Animal类）

注意：但并不是所有的“is a”的关系都可以写成继承

细说继承（下面关于继承的表述都是正确的）：

- 继承是使用已存在的类的定义作为基础，建立新类的技术
- 继承是在保留原有类的数据成员和成员函数的基础上，派生出新类，**新的类可以有某种程度的变异**；
- 通过继承，新类自动具有了原有类的数据成员和成员函数，因而只需定义原有类型没有的新的数据成员和成员函数。实现了**软件复用**，使得类之间具备了层次性；（覆盖、重载和扩展）
- 通过继承和派生形成的类族，反映了面向对象问题域、主题等概念。

类的继承

一. 被继承的类称为父类（superclass）,继承后产生的类称为子类（subclass）。

二. 单继承：如果子类只能有一个直接父类，称为单继承。

- ◆ 例如，轮船、客轮；人、大人 。

三. 多继承：如果子类可以有多个直接父类，称为多继承。

- DateTime(日期时间类)既是Time(时间类)，也是Date(日期类)
- 沙发床是沙发和床的子类

基类和派生类的关系

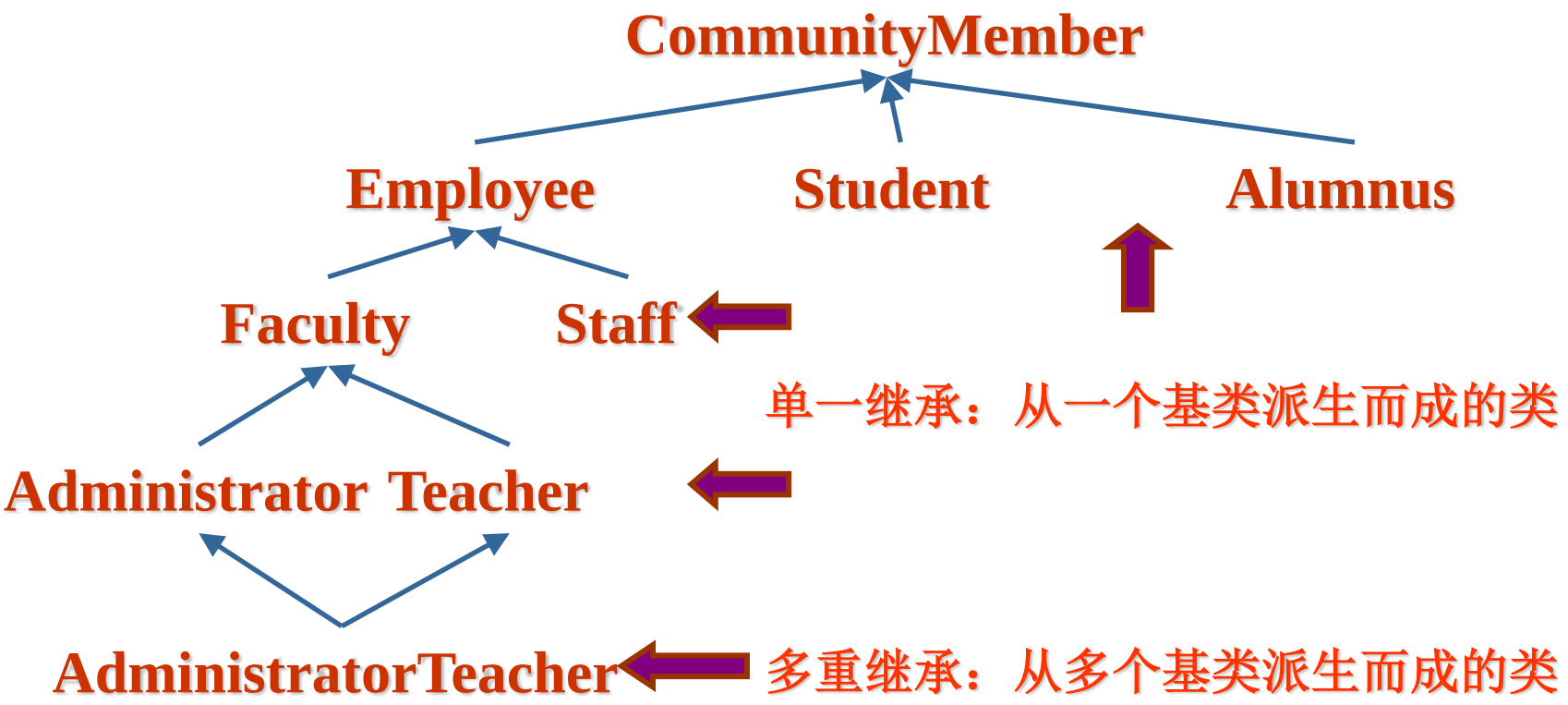
- 基类是对若干个派生类的抽象
  - 基类抽取了派生类的公共特征，在设计类时，**应该将通用的方法放到超类中**
- 派生类是基类的具体化
  - 通过扩展超类定义子类的时候，仅需指出子类与超类的不同之处，**通过增加数据成员或成员函数将基类变为某种更有用的类型**
- ***派生类可以看作基类定义的延续（抽象类在多态中细说）***
  - ***先定义一个抽象基类，该基类中有些操作并未实现***
  - ***然后定义非抽象的派生类，实现抽象基类中未实现的操作***

重新修改上面的Animal类为抽象类：com.buaa.inheritance）

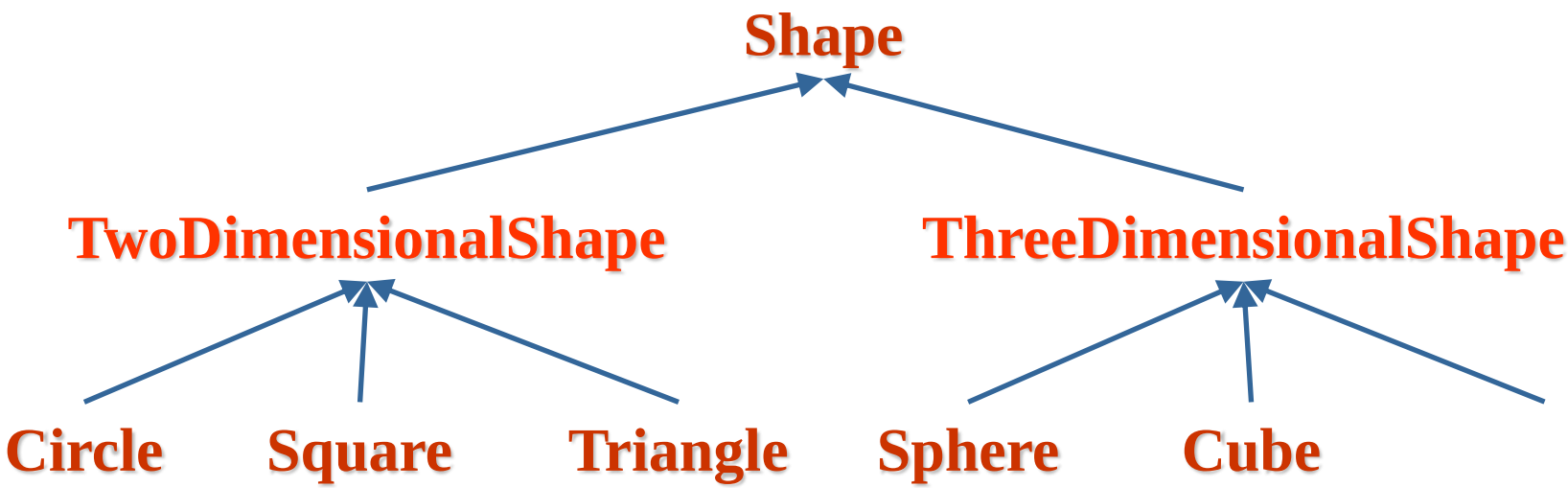
```
abstract class Animal {
    public abstract void eat();
    public abstract void sleep();
}
```

继承层次结构-示例1

基类和派生类构成类的层次结构



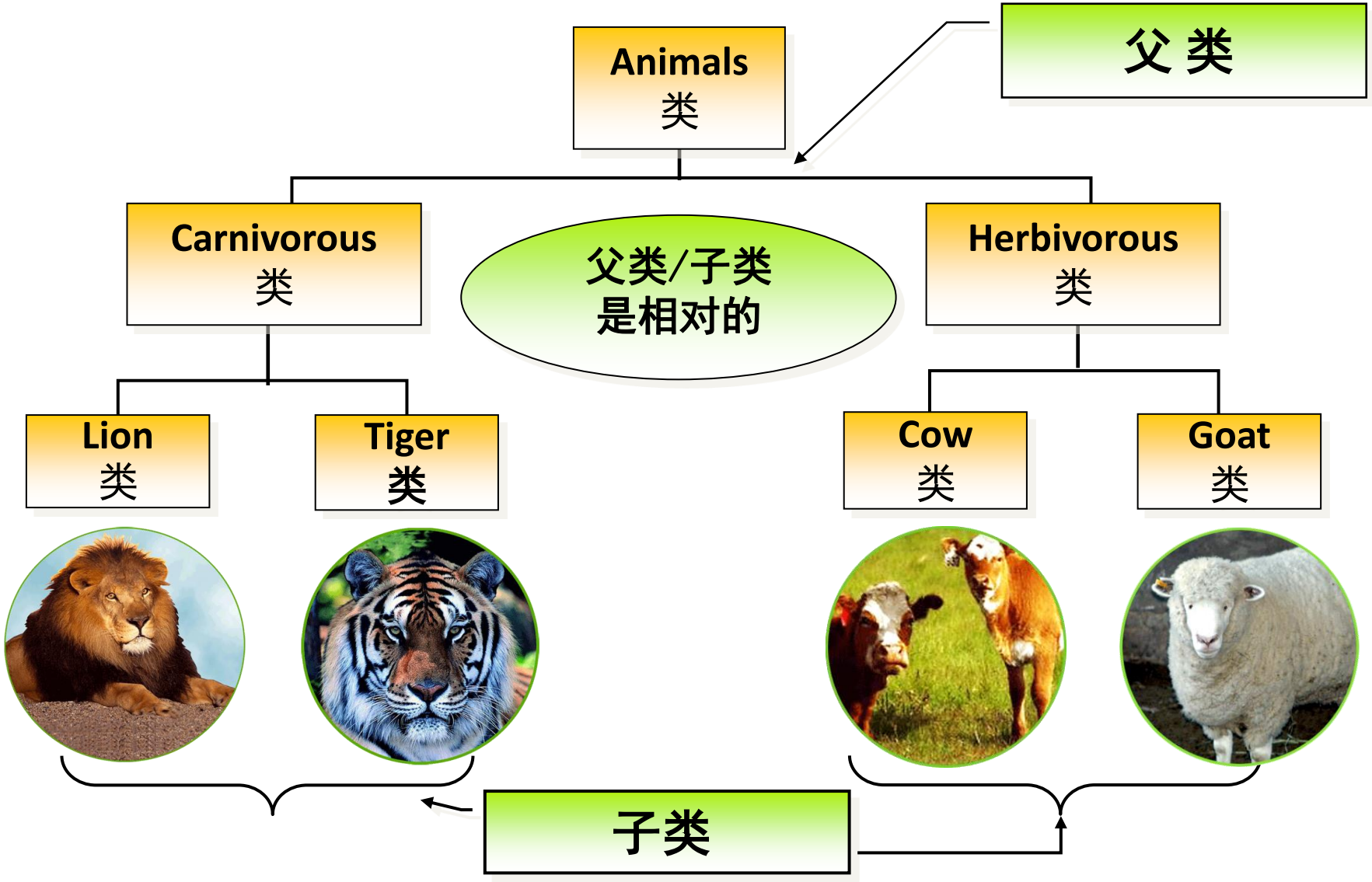
继承的层次结构-示例2



对现实世界中的层次结构进行分门别类！

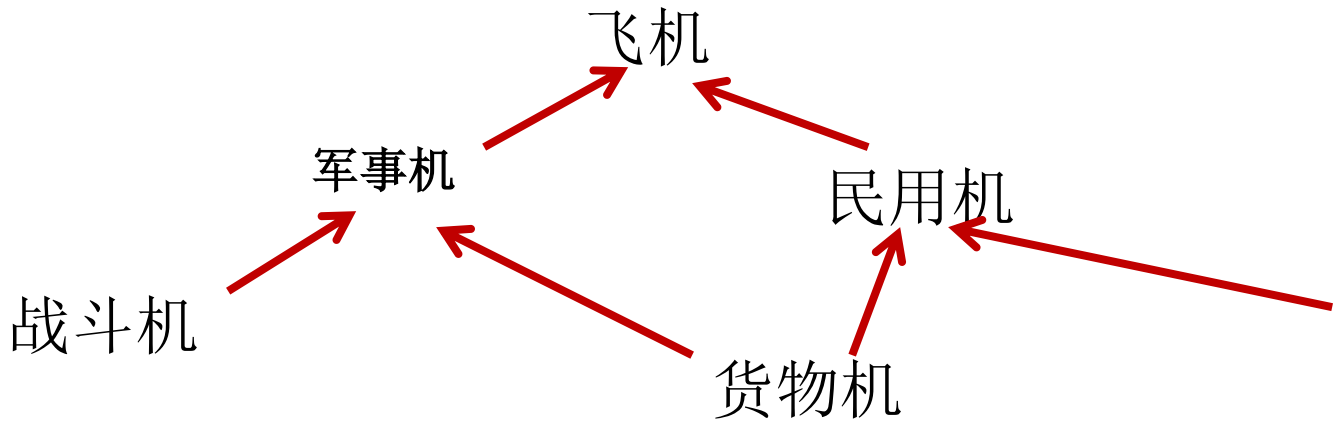
直接基类、直接派生类：Shape与TwoDimensionalShape
间接基类、间接派生类：Shape与Circle

继承的层次结构-示例3


父 类
Animals
类
Carnivorous
类
父类/子类
是相对的
Herbivorous
类
Lion
类
Tiger
类
Cow
类
Goat
类
子类

问答题：请确认各类之间的关系

- plane
- military plane（军事机）
- passenger plane （客机）
- cargo plane （货物机）
- fighter plane （战斗机）
- Airliner （民用机）



子类从父类那里继承了什么？

- 一、子类拥有父类的所有属性和方法，只不过父类的私有属性和方法，子类是无法直接访问到的。

- 二、子类可以对父类进行扩展。子类可以拥有自己属性和方法；子类既可以重载父类的方法，也可以覆盖父类的方法。

子类从父类那里不能继承什么？

- 子类不能继承父类的构造方法
  - 如何解决？

继承的进一步理解

一. 继承避免了公用代码的重复开发，减少代码的冗余，提高程序的复用性；

二. 支持多态（通过向上映射），提高程序的可扩展性；

三. 继承是类实现可重用性和可扩充性的关键持征。在继承关系下类之间组成网状的层次结构。

四. 通过继承增强一致性，从而减少模块间的接口和界面。

Java继承

主要内容

- 继承的必要性
- 理解继承
  - 基类与派生类的关系
  - 单继承
  - 多继承
  - 继承的层次结构
  - 子类继承了什么？
  - 继承的好处
- Java继承
  - 子类构造方法
  - 向上映射
  - **进一步理解封装**
  - 保护成员的使用
  - **变量隐藏**
  - **方法覆盖**
- 复合与继承

Java继承

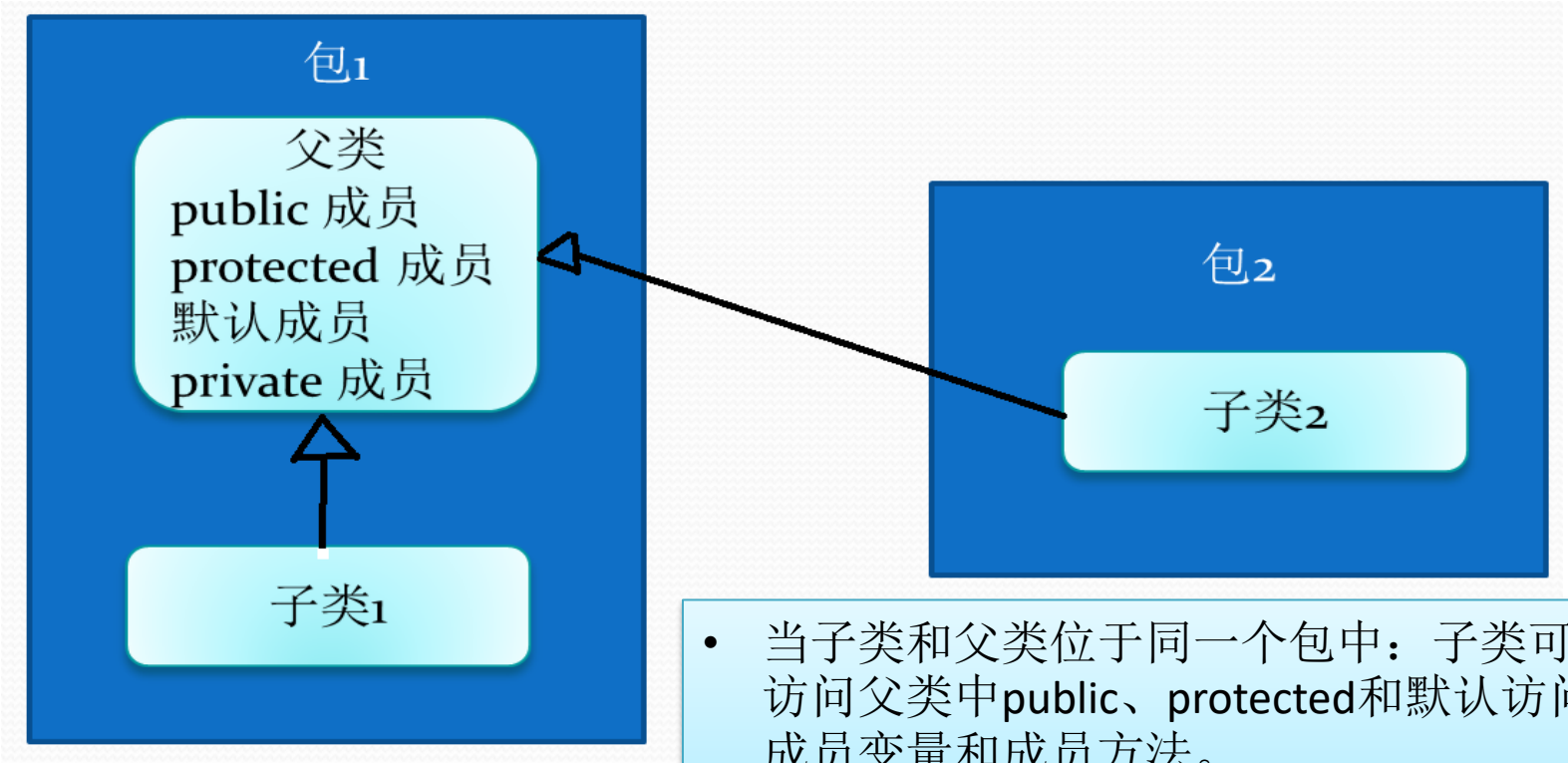
一．Java不支持类的多继承，但支持接口的多继承。

二．Java中的继承通过关键字extends实现。

三．类继承的格式：

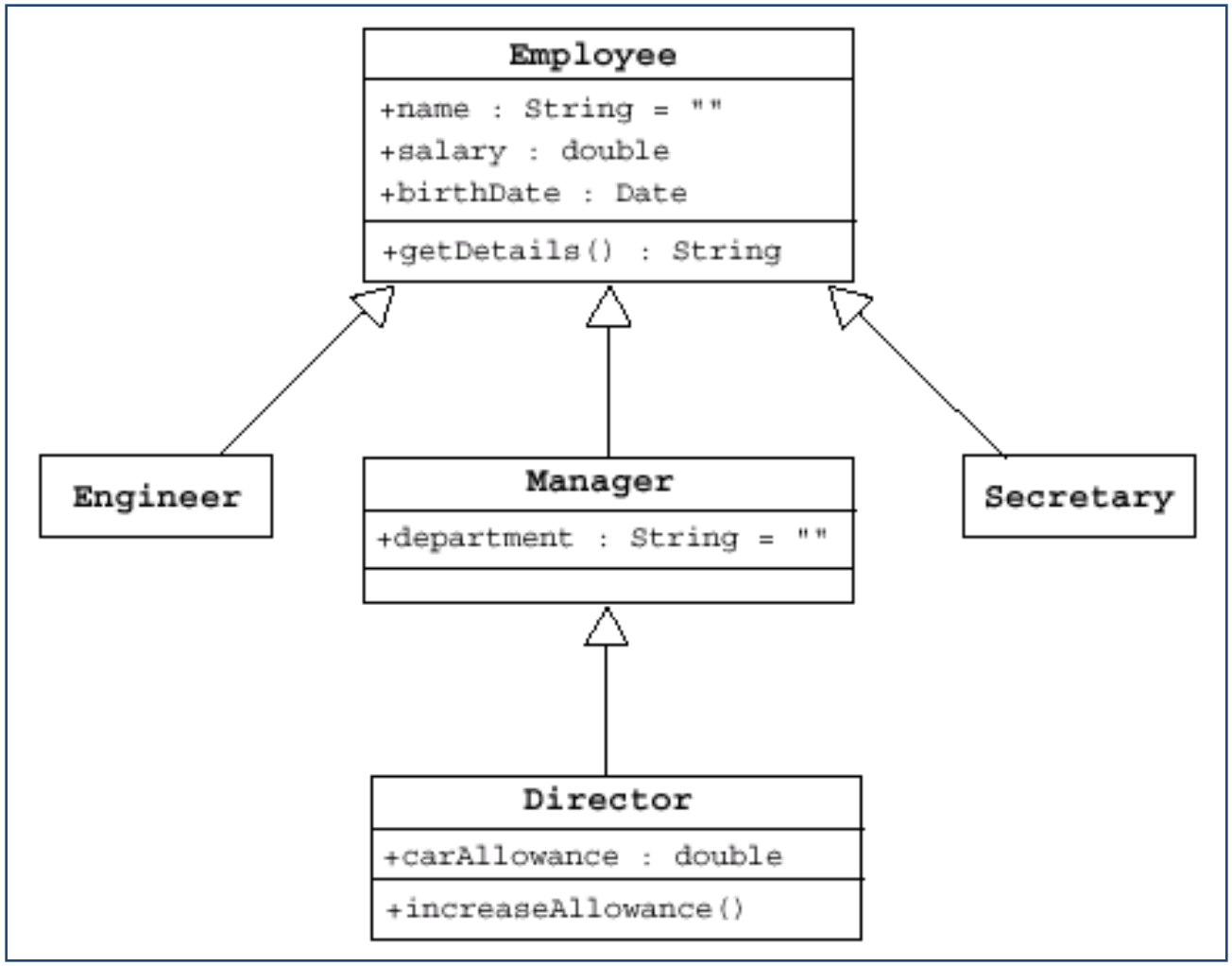
```
[修饰符]class 类名 extends 父类名
{
        类体;
}
```

Java子类可以直接访问父类的哪些内容（复习封装）



- 当子类和父类位于同一个包中：子类可以直接访问父类中public、protected和默认访问级别的成员变量和成员方法。
- 当子类和父类位于不同的包中：上节课讲过，情况比较复杂，具体问题，具体分析。

案例（公司职员）（inheritanceEx）



Person.java
Employee.java
Manager.java
Director.java

案例（公司职员）（inheritanceEx）

- 重点
  - Java 继承（extedns关键字）
  - 子类构造方法编写
    - super关键字的使用
    - 变量隐藏
    - 方法覆盖
  - 子类构建顺序
  - 向上映射
  - is a 和 has a对比

案例（公司职员）（inheritanceEx）

```
package com.dal.inheritaceEx;

public class TestClass {

    public static void main(String[] args) {
        Deirector d0=new Deirector();
        System.out.println("************************************");
        Deirector d1=new  Deirector("Tom",'M',23,2000,"sale",3000);//////
        Employee e1=d1;
        System.out.println(e1.getDetails());
        System.out.println(d1.getDetails());
        System.out.println("************************************");
        Person[] aArray=new Person[4];
        aArray[0]=d1;
        aArray[1]=new Student("Mary",'F',20,80,90);//upcasting
        aArray[2]=new Manager("Penny",'F',30,10000);
        aArray[3]=new Employee("Jack",'M',40,80000);
        for(Person p:aArray){
            System.out.println(p.getDetails());
        }
    }

}
```

案例（公司职员）（inheritanceEx）（多态）

```
父类构造函数
Employee类构造函数
Manager类构造函数
Deirector类构造函数
**********************************************
这个人是：Tom 薪水：2000.0 部门是：sale 津贴是：3000
这个人是：Tom 薪水：2000.0 部门是：sale 津贴是：3000
**********************************************
这个人是：Tom 薪水：2000.0 部门是：sale 津贴是：3000
这个人是：Mary，平均分是85
这个人是：Penny 薪水：10000.0 部门是：null
这个人是：Jack 薪水：80000.0
```

学习继承一定少不了这三个东西

- 编写子类构造器
  - 子类不能继承父类的构造方法
- protected关键字的使用
- 向上转型

子类构造函数：

- 构造函数不能被继承；
- 无参子类构造函数的编写
  - 子类可以通过super()显示调用父类无参的构造函数，也可以隐式调用
- 有参子类构造函数的编写
  - 初始化父类的成员变量；
  - 初始化子类的成员变量
  - 必须显示调用父类有参构造函数
- 无论使用this调用本类构造函数，还是使用super调用父类构造函数，都必须是该方法体中的第一条可以执行语句，否则会产生语法错误。

子类对象的生成(构造函数的调用顺序)

- 创建子类对象时，子类总是按层次结构从上到下的顺序调用所有超类的构造函数。如果继承和组合联用，要先构造基类的构造函数，然后调用组合对象的构造函数（组合按照声明的顺序调用）。
- 如果父类没有不带参数的构造方法，则在子类的构造方法中必须明确的告诉调用父类的某个带参数的构造方法，通过super关键字，这条语句还必须出现在构造方法的第一句。

子类对象的生成（构建）

- 子类创建对象时，子类的构造方法总是先调用父类的某个构造方法，完成父类部分的创建；然后再调用子类自己的构造方法，完成子类部分的创建。
- 如果子类的构造方法没有明显地指明使用父类的哪个构造方法，子类就调用父类的不带参数的构造方法 。
- 子类在创建一个子类对象时，不仅子类中声明的成员变量被分配了内存，而且父类的所有的成员变量也都分配了内存空间。

super关键字 （飞快的过一遍）

一. 如果子类调用父类的构造函数，则通过*super()*调用来实现。

二. 如果子类调用父类的同名方法，则通过super.方法名()来实现。

this关键字 （飞快的过一遍）

一．this变量代表对象本身。

二．当类中有两个同名变量，一个属于类的成员变量，而另一个属于某个特定的方法（方法中的局部变量），使用this区分成员变量和局部变量。

三．使用this简化构造函数的调用。

this引用与super引用的对比　　（飞快的过一遍）

- Java 每个类都默认地具有 null、this、super三个域，所以在任何类中都可以不加说明就可以直接引用它们：

  1. null ：代表“空”，用在定义一个对象但尚未为其开辟内存空间时。

  2. this 和 super ：是常用的指代子类对象和父类对象的关键字

总结：this的用法 （飞快的过一遍）

this用于：

1. 引用自身对象的成员变量
   - this.age;
2. 引用自身对象的成员方法
   - this.diaplay();
3. 调用自身的构造方法
   - this(“Jack”,Male,10);

总结：super的用法　　（飞快的过一遍）

super用于：

1. 引用父类对象的成员变量
   - super.age;
2. 引用父类对象的成员方法
   - super.diaplay();
3. 调用父类的构造方法
   - super(“Jack”,Male,10);

一．无论使用this调用本类构造函数，还是使用super调用父类构造函数，都必须是该方法体中的第一条可以执行语句。

二．否则会产生语法错误。

进一步理解封装 （中速过一遍）

Modifier	Same Class	Same Package	Subclass	Universe
private	Yes			
default	Yes	Yes		
protected	Yes	Yes	Yes	
public	Yes	Yes	Yes	Yes

思考题(不同包)

```
package mypack1;

public class Owner{
   public int v1;
   protected int v2;
   int v3;
   private int v4;

   protected void test(){}
}
```

```
package mypack2;
import mypack1.Owner;

public class Guest{
   public void method(){
      Owner a=new Owner();
      a.v1=1;  //合法
      a.v2=1;  //编译出错
      a.v3=1;  //编译出错
      a.v4=1;  //编译出错
      a.test(); //编译出错
   }
}
```

思考题（同一个包）

```
package mypack1;

public class Owner{
   public int v1;
   protected int v2;
   int v3;
   private int v4;

   protected void test(){}
}
```

```
package mypack1;
import mypack1.Owner;

public class Son extends Owner{
   public void method(){
      v1=1;  //合法
      v2=1;  //合法
      v3=1;  //合法
      test(); //合法
      Owner a=new Owner();
      a.v1=1;  //合法
      a.v2=1;  //合法
      a.v3=1;  //合法
      a.v4=1;  //编译出错
   }
}
```

思考题2（不同包）

```
package mypack1;

public class Owner{
   public int v1;
   protected int v2;
   int v3;
   private int v4;

   protected void test(){}
}
```

```
package mypack2;
import mypack1.Owner;

public class Son extends Owner{
   public void method(){
      v1=1;  //合法
      v2=1;  //合法
      v3=1;  //不合法
      test(); //合法
      Owner a=new Owner();
      a.v1=1;  //合法
      a.v2=1;  //编译出错
      a.v3=1;  //编译出错
      a.v4=1;  //编译出错
   }
}
```

保持基类数据成员私有性（一般而言）

- 将基类数据成员定义成private
  - 派生类中通过相应的访问函数进行访问
  - 优点：保持基类的封装性
  - 缺点：降低了访问效率

关于保护成员的使用　（特殊处理）

- 保护成员会破坏基类的封装性
  - 派生类可以直接访问和修改
- 关于保护成员的使用
  - 如果基类仅向其派生类提供服务，而不对其他客户提供该服务，使用protected成员访问说明符是合适的
  - 例如Owner和Son、Wife、Daughter是父子关系，Owner的车和房产可以被子类直接访问，但是不对其他类开放，那么车和房产可以声明为protected的权限。

向上转型(upcasting)

- 将子类转换成父类，在继承关系上面是向上移动的，所以一般称之为向上转型或者向上映射。
- 由于向上转型是从一个叫专用类型向较通用类型转换，所以它总是安全的，唯一发生变化的可能就是属性和方法的丢失。
- 自动类型转换：这就是为什么编译器在“未曾明确表示转型”或者“未曾指定特殊标记”的情况下，仍然允许向上转型的原因。

向上转型的缺憾：

- 只能调用父类中定义的属性和方法，对于子类中的方法和属性它就望尘莫及了，必须**强制转成子类类型**

案例（公司职员）（inheritanceEx）

```
package com.dal.inheritaceEx;

public class TestClass {

    public static void main(String[] args) {
        Deirector d0=new Deirector();
        System.out.println("************************************");
        Deirector d1=new  Deirector("Tom",'M',23,2000,"sale",3000);//////
        Employee e1=d1;
        System.out.println(e1.getDetails());
        System.out.println(d1.getDetails());
        System.out.println("************************************");
        Person[] aArray=new Person[4];
        aArray[0]=d1;
        aArray[1]=new Student("Mary",'F',20,80,90);//upcasting
        aArray[2]=new Manager("Penny",'F',30,10000);
        aArray[3]=new Employee("Jack",'M',40,80000);
        for(Person p:aArray){
            System.out.println(p.getDetails());
        }
    }

}
```

继承中的变量隐藏

一. 变量隐藏：在子类对父类的继承中，如果子类的成员变量和父类的成员变量同名，此时称为子类隐藏（override）了父类的成员变量。

二. 子类若要引用父类的同名变量。要用super关键字做前缀加圆点操作符引用，即super.变量名

方法隐藏和方法覆盖

- 覆盖就是子类的方法跟父类的方法具有完全一样的签名和参数（**它们的名称、参数以及返回类型完全相同**）
  - 方法覆盖在JAVA中是动态联编（后面详细讲）
- 私有方法、静态方法不能被覆盖，如果在子类出现了同签名的方法，就是方法隐藏
- 用fianl声明的成员方法是最终方法，最终方法不能被子类覆盖（重写）（后面细说final关键字）

```java
package com.buaa.test;

public class Test {
    public static void main(String[] args) {
        Subclass s = new Subclass();
        System.out.println(s.x + " " + s.y + " " + s.z);
        System.out.println(s.method());
        Base b2 = s;//正确
        Base b3 = (Base)s;//正确

        System.out.println(b2.x+" "+b2.y+" "+b2.z);
        System.out.println(b2.method());
    }
}

class Base {
    int x = 1;
    static int y = 2;
    int z = 3;

    int method() {
        return x;
    }
}

class Subclass extends Base {
    int x = 4;
    int y = 5;
    static int z = 6;

    int method() {
        return x;
    }
}
```

```
4 5 6
4
1 2 3
4
```

```
The hide method in Planet.
The override method in Earth.
```

```java
package com.buaa.test;

class Planet {
    public static void hide() {
        System.out.println("The hide method in Planet.");
    }
    public void override() {
        System.out.println("The overrid method in Planet.");
    }
}
public class Earth extends Planet {
    public static void hide() {
        System.out.println("The hide method in Earth.");
    }
    public void override() {
        System.out.println("The override method in Earth.");
    }
    public static void main(String[] args) {
        Earth myEarth = new Earth();
        Planet myPlanet = myEarth;
        myPlanet.hide();
        myPlanet.override();
    }
}
```

综合案例1（电子宠物系统）

- 这两个类图有什么问题？

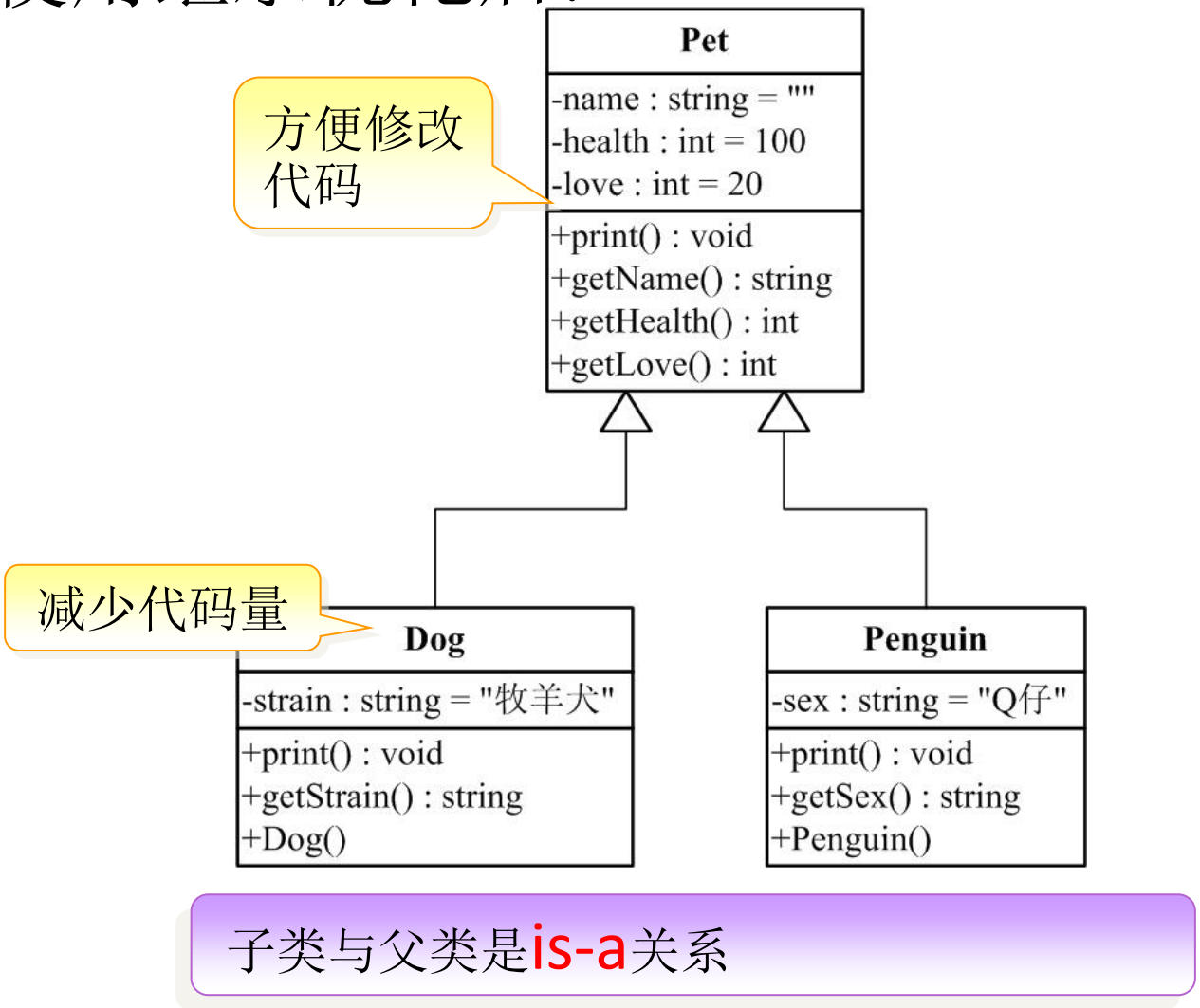
Dog	Penguin
- name:String - health:int - weight:int	- name:String - health:int - weight:int
- strain:String	- sex:String
+ print():void + getName():String + getHealth ():int + getWeight():int	+ print():void + getName():String + getHealth ():int + getWeight():int
+ getStrain:String + Dog()	+ getSex():String + Penguin()

将重复代码抽取到父类中

使用继承优化设计

综合案例1（电子宠物系统）

- 使用继承优化后：



本案例重点

- 继承的使用（extends关键字）
- 抽象类，抽象方法（abstract）
- 子类构造方法的编写
  - 变量隐藏，掌握this,super关键字的使用
- 构造方法重载
- 方法覆盖

示例代码：

```
package com.buaa.pet;
public abstract class Pet {
    protected String name;
    protected int health;
    protected int love;
    public Pet(String name) {
        this.name = name;
        this.health = 100;
        this.love = 20;
    }
    public abstract void print();

    public String getName() {
        return this.name;
    }
    public int getHealth() {
        return this.health;
    }
    public int getLove() {
        return this.love;
    }
}
```

示例代码：

```
package com.buaa.pet;
public class Dog extends Pet {
    private String strain;

    public Dog(String name, String strain) {
        super(name);
        this.strain = strain;
    }

    @Override
    public void print() {
        System.out.println(
            this.strain + " " + this.name + "\n"
            + "健康值: " + this.health + "\n"
            + "好感度: " + this.love
        );
    }

    public String getStrain() {
        return this.strain;
    }
}
```

```java
package com.buaa.pet;
import java.util.Random;
public class Penguin extends Pet {
    private String sex;
    public Penguin(String name) {
        this(name, ((new Random()).nextInt(2) == 1));
    }
    public Penguin(String name, boolean isMale) {
        super(name);
        this.sex = (isMale? "Q仔": "Q妹");
    }
    @Override
    public void print() {
        System.out.println(
            "企鹅"+ this.sex + " " + this.name + "\n"
            + "健康值: " + this.health + "\n"
            + "好感度: " + this.love
        );
    }
    public String getSex() {
        return this.sex;
    }
}
```

多态（下节课分解）

```
package com.buaa.pet;

public class Demo {
    public static void main(String[] args) {
        Pet pet;
        pet = new Dog("小八", "柴犬");
        pet.print();
        System.out.println("----------------------");
        pet = new Penguin("麻花");
        pet.print();
        System.out.println("----------------------");
        pet = new Penguin("马化腾", true);
        pet.print();
    }
}
```

综合案例2（汽车租赁公司出租多种车辆）

- 某汽车租赁公司出租多种车辆，车型及租金情况如下：

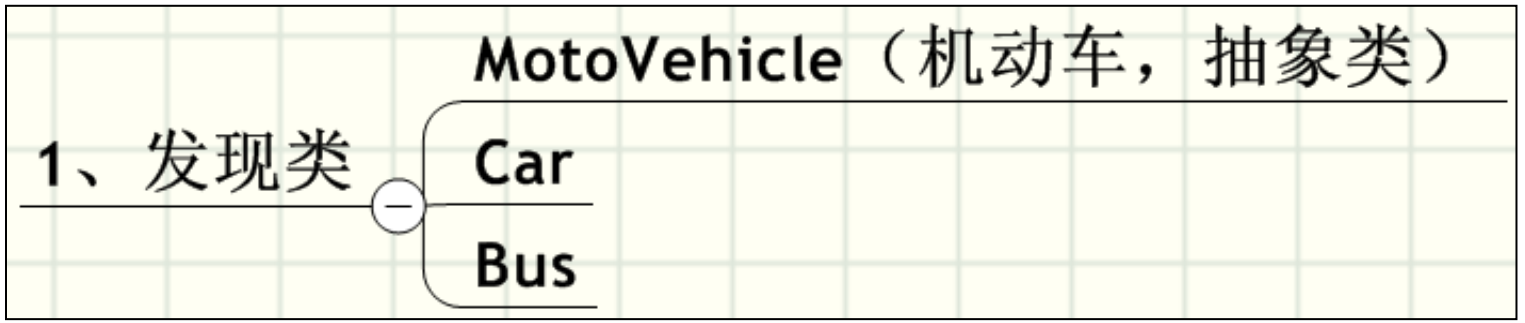
	轿车			客车（金杯、金龙）	
车型	别克商务舱GL8	宝马550i	别克林荫大道	<=16座	>16座
日租费(元/天)	600	500	300	800	1500

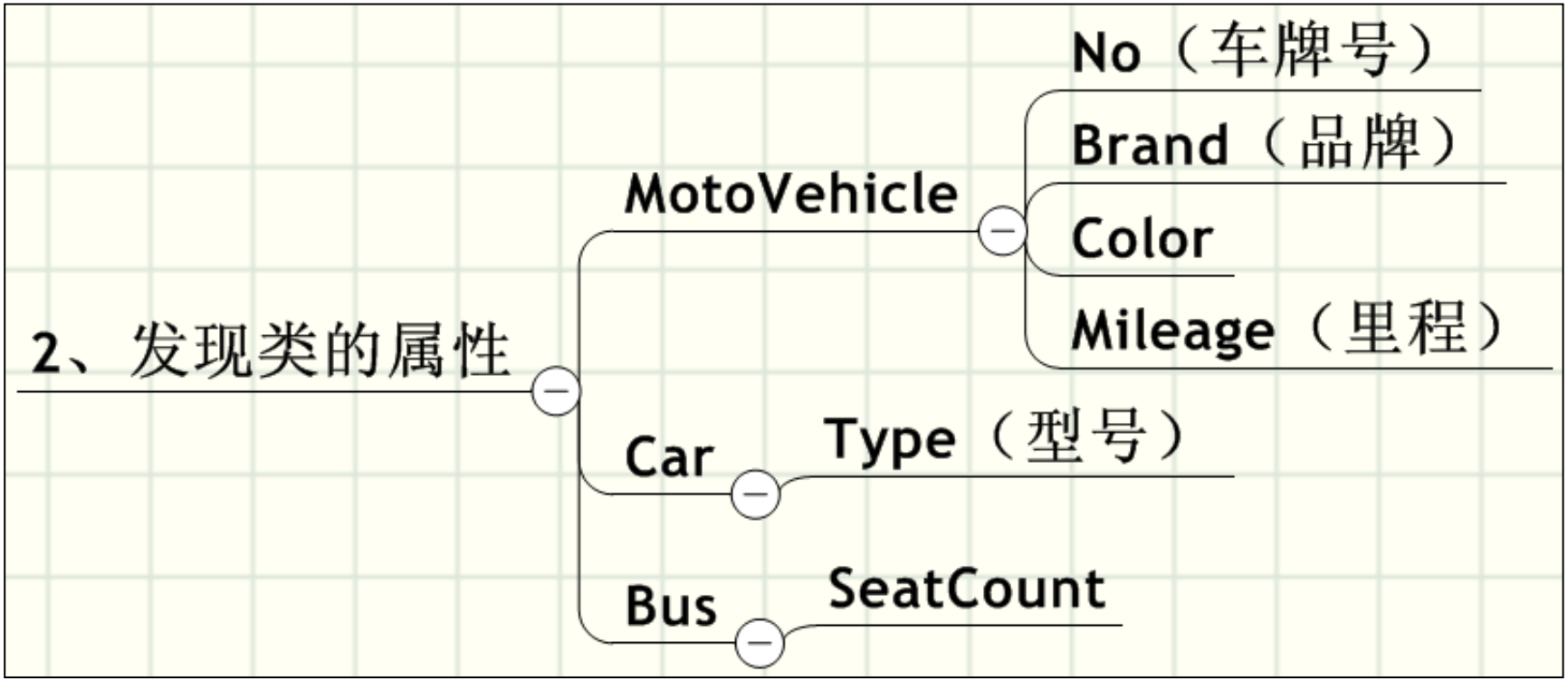
- 编写程序实现计算租赁价

综合案例分析

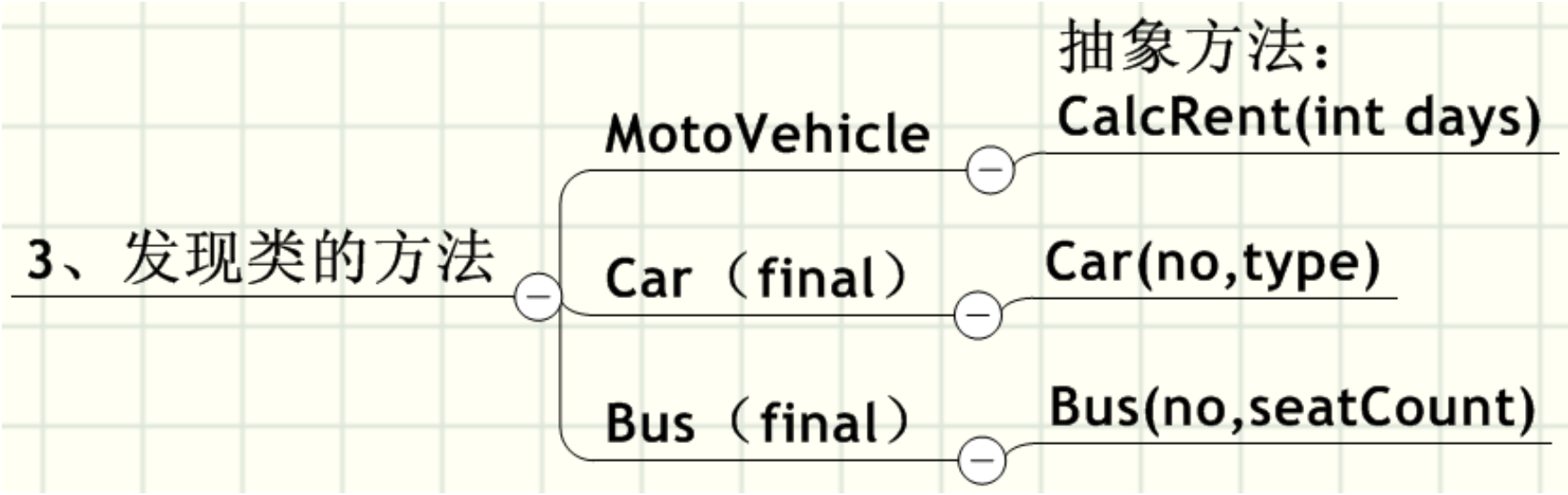
- 需求说明：
  - 根据分析编写MotoVehicle、Car、Bus类
- 分析
  - 抽象出类
  - 抽象出类的属性
  - 抽象出类的方法
  - 优化设计
  - 编写程序入口

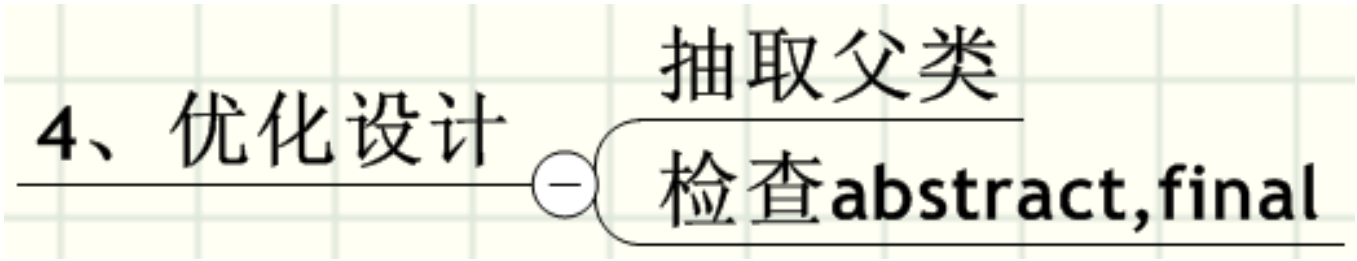
综合案例分析2-1

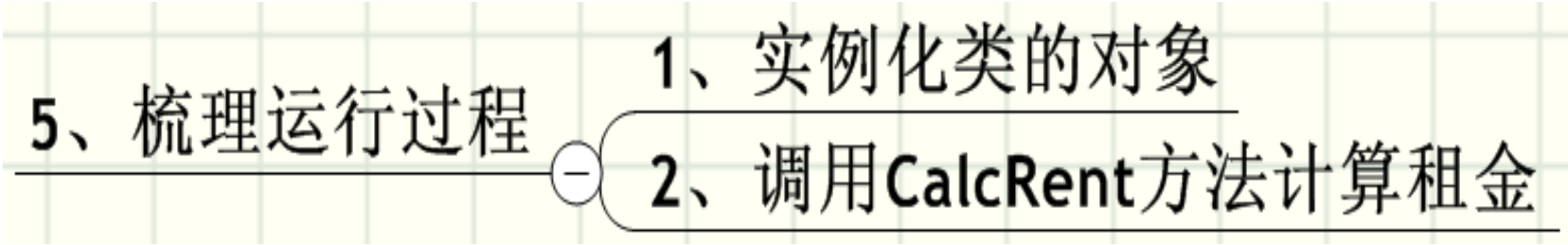

1、发现类
MotoVehicle（机动车，抽象类）
Car
Bus


2、发现类的属性
MotoVehicle
No（车牌号）
Brand（品牌）
Color
Mileage（里程）
Car
Type（型号）
Bus
SeatCount

综合案例分析2-2


3、发现类的方法
MotoVehicle
抽象方法：
CalcRent(int days)
Car（final）
Car(no,type)
Bus（final）
Bus(no,seatCount)


4、优化设计
抽取父类
检查abstract,final


5、梳理运行过程
1、实例化类的对象
2、调用CalcRent方法计算租金

```
package com.buaa.vehicle;
public class MotoVehicle {
    protected double rentPricePerDay;
    protected final String no;
    protected String brand;
    public enum Color {
        BLACK("黑色"),
        WHITE("白色"),
        GRAY("灰色"),
        RED("红色"),
        BLUE("蓝色");
        private final String value;
        private Color(String value) {
            this.value = value;
        }
        @Override
        public String toString() {
            return this.value;
        }
    }
    protected Color color;
```

```java
public MotoVehicle(String no) {
    this.no = no;
}
public void setRentPricePerDay(double price) {
    this.rentPricePerDay = price;
}
public double calcRentPrice(int days) {
    return this.rentPricePerDay * days;
}
public String getNo() {
    return this.no;
}
public String getBrand() {
    return this.brand;
}
public void setBrand(String brand) {
    this.brand = brand;
}
public Color getColor() {
    return this.color;
}
public void setColor(Color color) {
    this.color = color;
}
```

```java
public class Car extends MotoVehicle {
    private String type;
    public Car(String no) {
        super(no);
    }
    public String getType() {
        return this.type;
    }
    public void setType(String type) {
        this.type = type;
    }
    @Override
    public String toString() {
        return String.format(
            "%s %s %s \n车牌: %s\n日租金: %.2f",
            this.brand, this.type, this.color,
            this.no,
            this.rentPricePerDay
        );
    }
}
```

```java
public class Bus extends MotoVehicle {
    private int seatCount;
    public Bus(String no) {
        super(no);
    }
    public int getSeatCount() {
        return this.seatCount;
    }
    public void setSeatCount(int seatCount) {
        this.seatCount = seatCount;
    }
    @Override
    public String toString() {
        return String.format(
            "%s %d座%s \n车牌: %s\n日租金: %.2f",
            this.brand, this.seatCount, this.color,
            this.no,
            this.rentPricePerDay
        );
    }
}
```

Demo.java

- 自己发挥

案例3(is a and has a)

- Item
- CD
- DVD
- Database

```
package com.buaa.demo;

public class Item {
        private String title;//标题
        private int playtime;//播放时间
        private boolean borrow;//是否外借

        public Item(String title, int playtime, boolean borrow) {
                this.title = title;
                this.playtime = playtime;
                this.borrow = borrow;
        }
        public String getTitle() {
                return title;
        }
        public void setTitle(String title) {
                this.title = title;
        }
        public int getPlaytime() {
                return playtime;
        }
```

```
public void setPlaytime(int playtime) {
        this.playtime = playtime;
}
public boolean isBorrow() {
        return borrow;
}
public void setBorrow(boolean borrow) {
        this.borrow = borrow;
}
public void print() {//print方法 输出数据
        System.out.print("标题：  "+title+" 时间：  "+playtim

}
```

```
package com.buaa.demo;

public class CD extends Item{//子类CD继承父类Item
	private String artist;// 艺术家
	public CD(String title, int playtime, boolean borrow,String artist) {
		super(title, playtime, borrow);
		this.artist=artist;
	}
	public String getArtist() {
		return artist;
	}
	public void setArtist(String artist) {
		this.artist = artist;
	}
	public void print() {//print方法重写父类的print
		System.out.print("CD ");
		super.print();//super调用父类的print方法
		System.out.print(" 艺术家： "+artist);//输出子类独有的属性
		System.out.println();
	}

}
```

```
package com.buaa.demo;

public class DVD extends Item{//子类DVD继承父类Item
	private String director;//导演
	public DVD(String title, int playtime, boolean borrow,String director) {
		super(title, playtime, borrow);
		this.director=director;
	}
	public String getDirector() {
		return director;
	}
	public void setDirector(String director) {
		this.director = director;
	}
	public void print() {//print方法重写父类的print
		System.out.print("DVD ");
		super.print();//super调用父类的print方法
		System.out.print(" 导演： "+director);//输出子类独有的属性
		System.out.println();
	}
}
```

Has a

```
package com.buaa.demo;

import java.util.ArrayList;
public class database {
        ArrayList<Item> listItem=new ArrayList<Item>();//创建ArrayList容器，存储类型为Item
        public void add(Item item) {//add方法，传入Item类型，
                listItem.add(item);//添加进入listItem容器中
        }
        public void list() {//list方法 负责遍历容器中所有数据
                for(Item item:listItem) {
                        item.print();
                }
        }
        public static void main(String[] args) {
                database data=new database();//创建database对象
                //添加Item类型对象，添加Item子类对象CD（匿名对象），CD构造器初始化，多态
                data.add(new CD("起风了",3,false,"买辣椒也用券"));
                data.add(new CD("流量", 3, false,"半阳"));
                data.add(new DVD("一出好戏", 125, false,"黄渤"));
                data.list();
        }

}
```

讨论：

- 1、继承带来了哪些好处？
  2、继承是否破坏了类的封装性？

面向对象的三大特征之一：*继承真的很好！*

一. 继承避免了公用代码的重复开发，减少代码的冗余，提高程序的复用性；

二. 支持多态（通过向上映射），提高程序的可扩展性；

三. 继承是类实现可重用性和可扩充性的关键持征。在继承关系下类之间组成网状的层次结构。

四. 通过继承增强一致性，从而减少模块间的接口和界面。

五. 通过向上映射，让我们体验到多态的益处。

继承是否破坏了类的封装性？

- 是的。继承破坏了封装性，换句话说，子类依赖于父类的实现细节。**继承很容易改变父类实现的细节(所以父类中能写成final尽量写成final)，**即使父类整体没有问题，也有可能因为子类细节实现不当，而破坏父类的约束。
- 其实这是一个平衡关系，不是绝对关系，一定程度的封装和一定程度的继承，可以提高开发效率，继承破坏了封装，但是有时继承是必须的，为了继承牺牲一定的封装是允许的。不能绝对的为了封装，就不去继承。

谨慎继承

- 继承是一种强耦合关系。父类变，子类就必须变。
- 继承破坏了封装，子类可以重写父类的方法，子类可以直接访问保护成员。
- 那么到底要不要使用继承呢？
  - 《Think in java》中提供了解决办法：问一问自己是否需要从子类向父类***进行向上转型***。如果必须向上转型，则继承是必要的，但是如果不需要，则应当好好考虑自己是否需要继承。

继承的使用原则

- 合理使用继承，谨慎继承，**继承树的层次不可太多**
- 继承是一种提高程序代码的可重用性、以及提高系统的可扩展性的有效手段
- 但是，如果继承树非常复杂、或者随便扩展本来不是专门为继承而设计的类，反而会削弱系统的可扩展性和可维护性

小结：使用继承的注意点

1. 子类一般比父类包含更多的属性和方法。
2. 父类中的 private 成员在子类中是不可见的，因此在子类中不能直接使用它们。
3. 父类和其子类间必须存在“是一个”即“is-a”的关系，否则不能用继承。但也并不是所有符合“is-a”关系的都应该用继承。例如，正方形是一个矩形，但不能让正方形类来继承矩形类，因为正方形不能从矩形扩展得到任何东西。正确的继承关系是正方形类继承图形类。
4. Java 只允许单一继承（即一个子类只能有一个直接父类），C++ 可以多重继承（即一个子类有多个直接父类）。

小结：继承的优缺点

- 在面向对象语言中，继承是必不可少的、非常优秀的语言机制，它有如下优点：
  1. 实现代码共享，减少创建类的工作量，使子类可以拥有父类的方法和属性。
  2. 提高代码维护性和可重用性。
  3. 提高代码的可扩展性，更好的实现父类的方法。

小结：继承的缺点

- 自然界的所有事物都是优点和缺点并存的，继承的缺点如下：
  1. 继承是侵入性的。只要继承，就必须拥有父类的属性和方法。
  2. 降低代码灵活性。子类拥有父类的属性和方法后多了些约束。
  3. 增强代码耦合性（开发项目的原则为高内聚低耦合）。当父类的常量、变量和方法被修改时，需要考虑子类的修改，有可能会导致大段的代码需要重构。

思考题

```
package com.buaa.fuzi;

class Fu {
    int num = 4;//没有这句会编译失败
}
class Zi extends Fu {
    int num = 5;
}
public class Demo {
    public static void main(String[] args)  {
        Fu f = new Zi();
        System.out.println(f.num);
        Zi z = new Zi();
        System.out.println(z.num);
    }
}
```

思考题

```
package com.buaa.fuzi;
class Father {
    int num = 4;
    void show() {     //没有这个方法，编译失败
        System.out.println("Fu show num");
    }
}
class Son extends Father {
    int num = 5;
    void show() { //重写父类方法
            System.out.println("Zi show num");
    }
    void show_1(){
            System.out.println("Zi show show_1");
    }
}
class Demo1 {
    public static void main(String[] args)  {
        Father f = new Son();
        f.show();
        //f.show_1();
    }
}
```