

第4章 面向对象 (下)

老师：李沁如



/4.1

类的继承



面向对象三大特征

封装

继承

多态

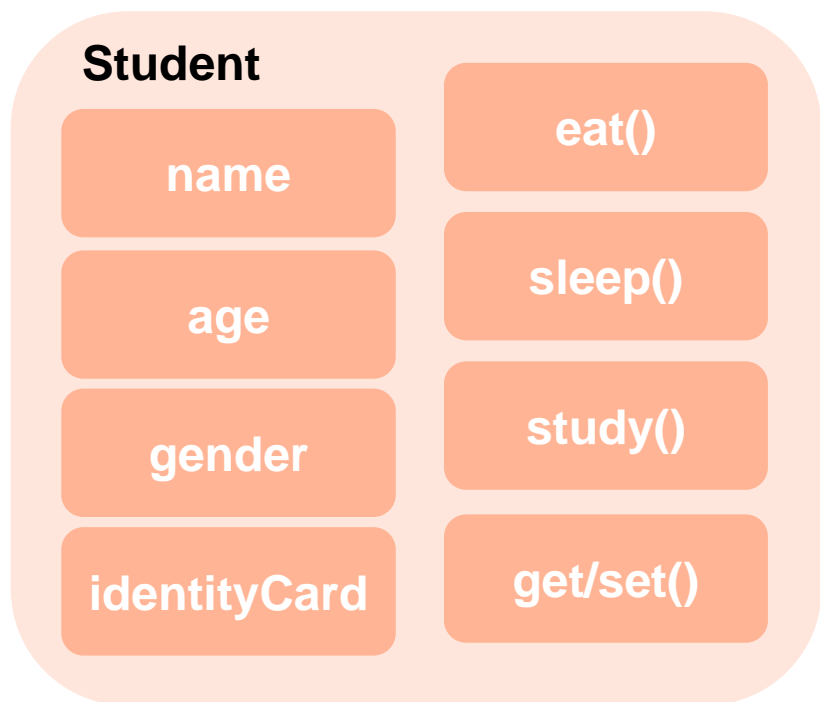
继承：

- 现实生活中，继承一般是指子女继承父辈的财产。
- 在程序中，继承描述的是事物之间的所属关系。

4.1.1 继承的概念

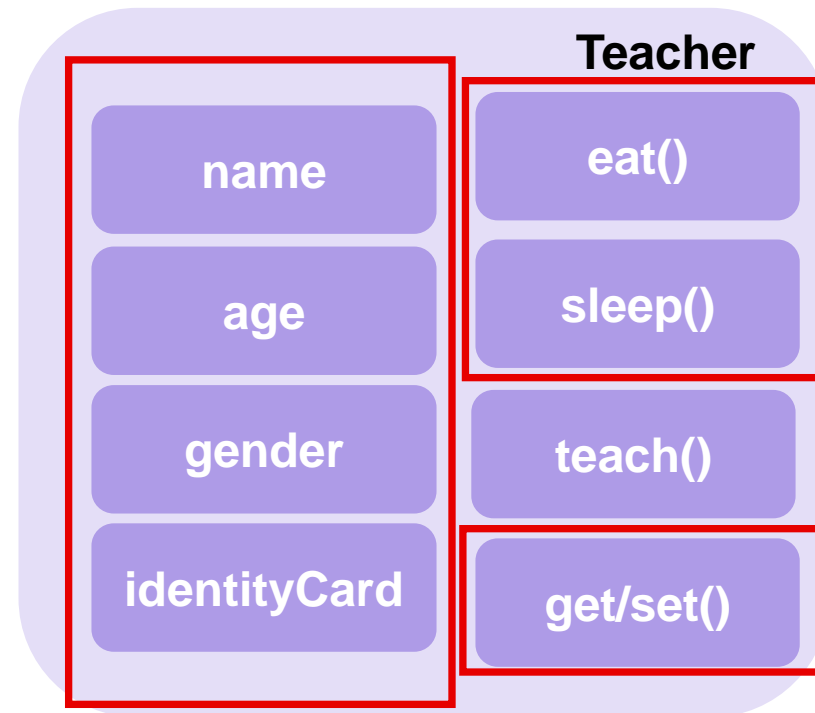
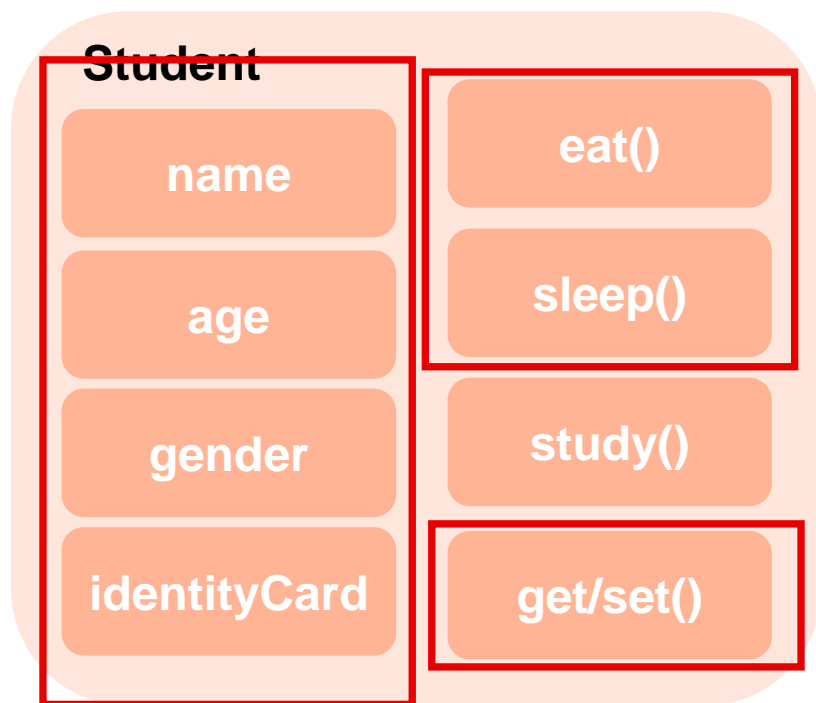
封装:

对象代表什么，就得封装对应的数据，并提供数据对应的行为。

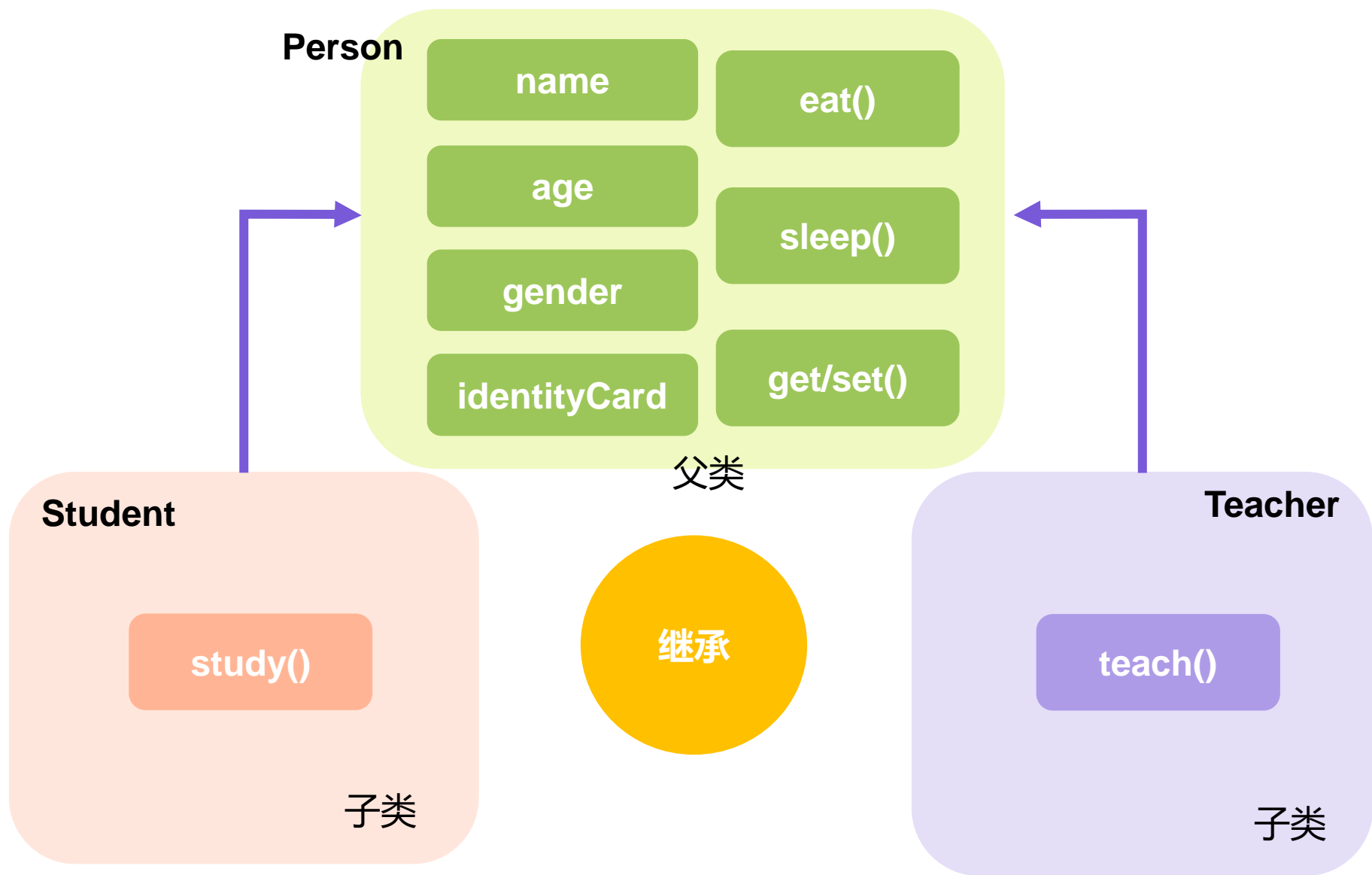


```
public class TestClass {  
    public static void printStudentInfo(Student s){  
        //...  
    }  
}
```

4.1.1 继承的概念



4.1.1 继承的概念



4.1.1 继承的概念

继承

- Java中提供一个关键字extends，用这个关键字，我们可以让一个类和另一个类建立起继承关系。

```
public class Student extends Person{ }
```

- Student称为子类（派生类），Person称为父类（基类或超类）。

使用继承的好处

- 可以把多个子类中重复的代码抽取到父类中了，提高代码的复用性。
- 子类可以在父类的基础上，增加其他的功能，使子类更强大。

4.1.1 继承的概念

继承需要学习什么?

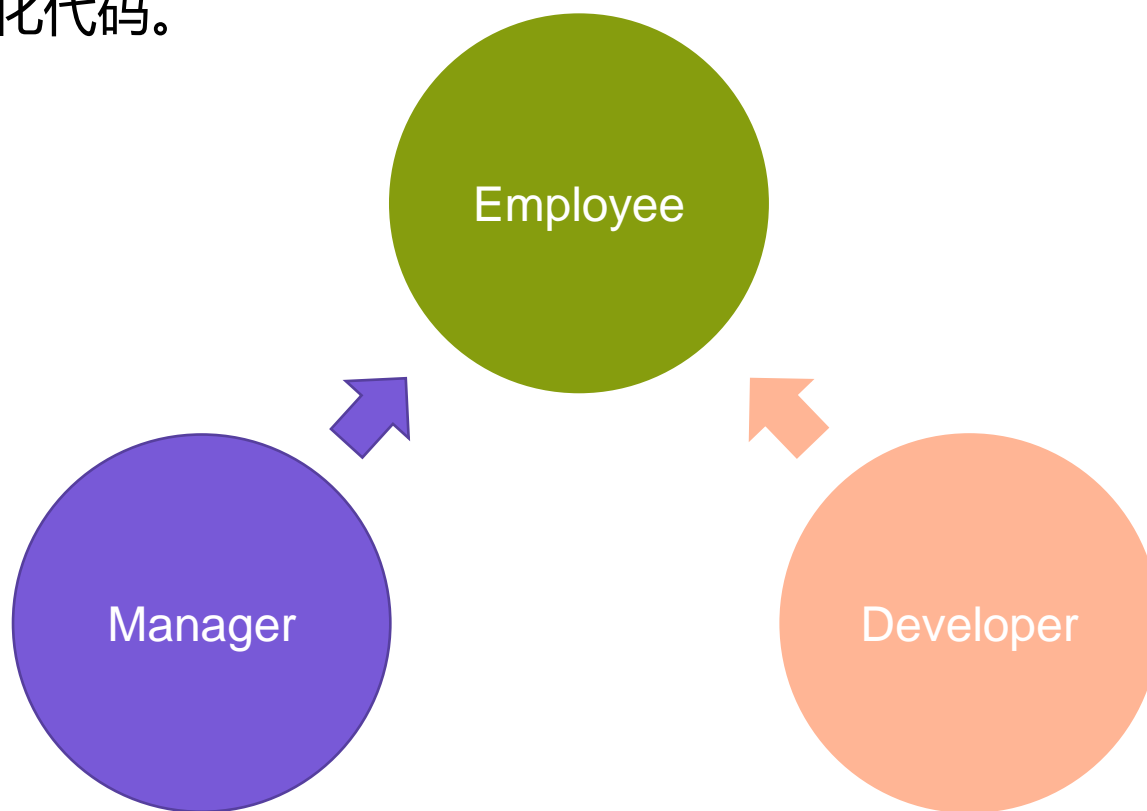
自己设计

用别人的

4.1.1 继承的概念

什么时候用继承？

当类与类之间，存在相同（共性）的内容，并满足子类是父类中的一种，就可以考虑使用继承，来优化代码。



4.1.1 继承的概念

Java只支持单继承，不支持多继承，但支持多层继承。

单继承：一个子类只能继承一个父类

不支持多继承：子类不能同时继承多个父类

4.1.1 继承的概念

Java只支持单继承，不支持多继承，但支持多层继承。

为什么不支持多继承？

```
public class 父类A{  
    public void method(){  
        System.out.println("复习数学");  
    }  
}
```

```
public class 父类B{  
    public void method(){  
        System.out.println("复习语文");  
    }  
}
```

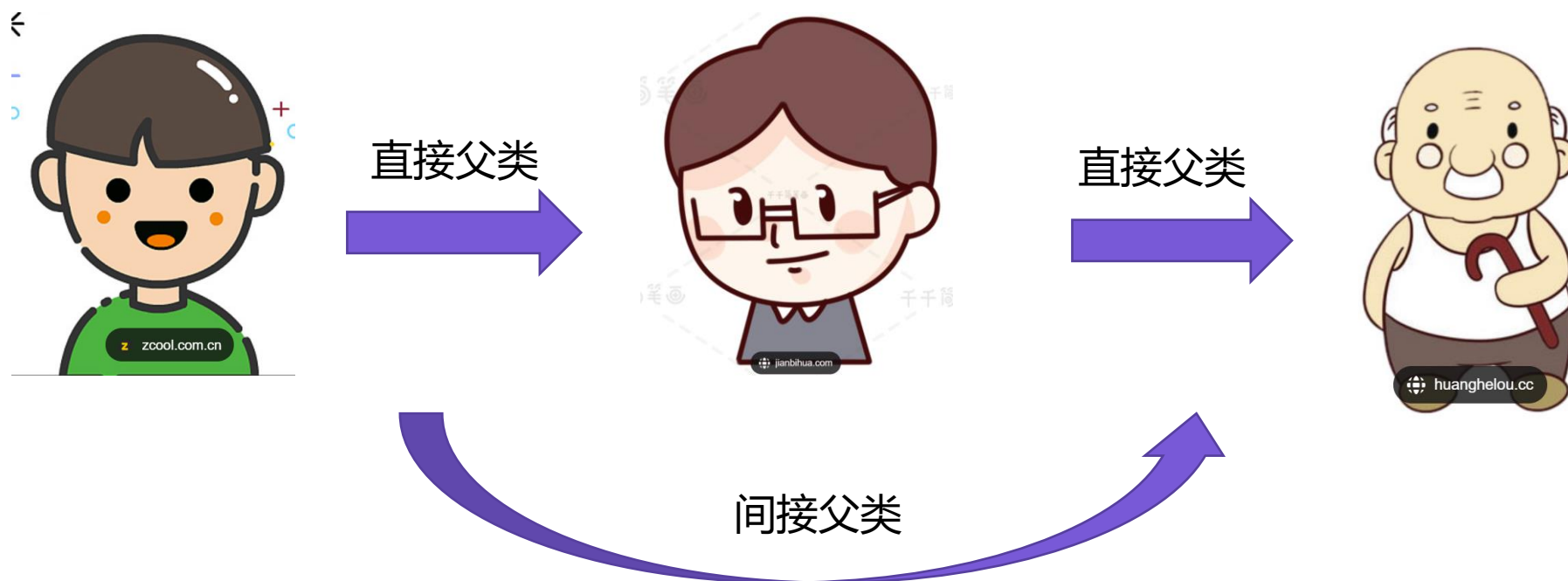
```
public class 子类 extends 父类A,父类B{  
    }  
}
```

```
public class Test{  
    public static void main(String[] args){  
        子类 z = new 子类();  
        z.method();  
    }  
}
```

4.1.1 继承的概念

Java只支持单继承，不支持多继承，但支持多层继承。

多层继承：子类A继承父类B，父类B可以继承父类C



4.1.1 继承的概念

多层继承：子类A继承父类B，父类B可以继承父类C

每一个类都直接或者间接的继承于Object

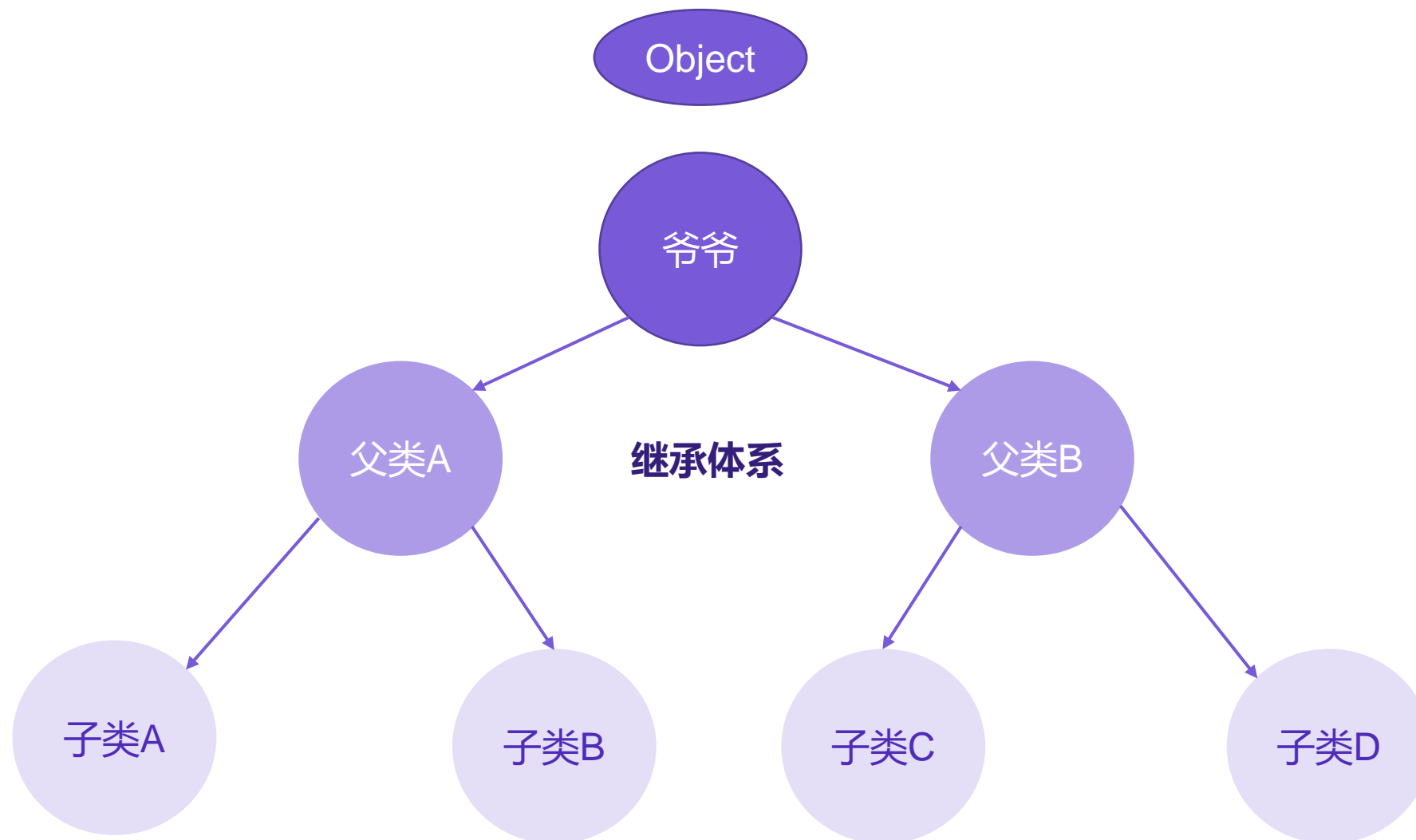
```
public class A{  
    .....  
}
```



```
public class A extends Object{  
    .....  
}
```

虚拟机会默认让class A继承Object类

4.1.1 继承的概念



4.1.1 继承的概念

继承的练习（自己设计一个继承体系）

现在有四种动物：布偶猫，中国狸花猫，哈士奇，柴犬。

暂时不考虑属性，只要考虑行为。

请按照继承的思想特点进行继承体系的设计。

四种动物分别有以下行为：

布偶猫：吃饭，喝水，抓老鼠

中国狸花猫：吃饭，喝水，抓老鼠

哈士奇：吃饭，喝水，看家，拆家

柴犬：吃饭，喝水，看家，狩猎



4.1.1 继承的概念

画图：从下往上画

下面：子类

上面：父类

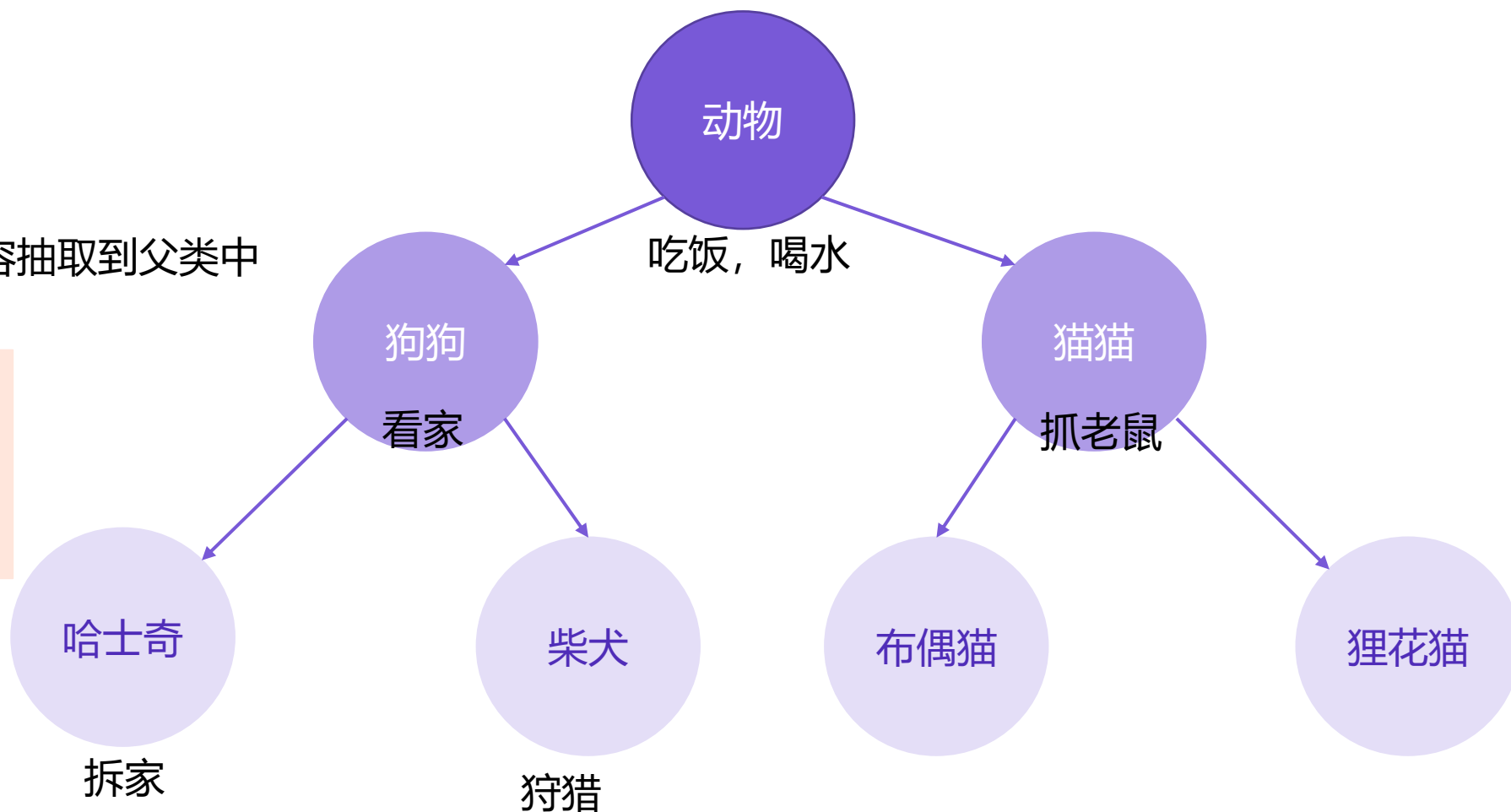
需要把子类中的共性内容抽取到父类中

核心：

1. 共性内容抽取
2. 子类是父类中的一种

书写代码：

从上往下写



4.1.1 继承的概念

```
public class Animal {  
    private String color;  
    public String getColor() {  
        return color;  
    }  
    public void setColor(String color) {  
        this.color = color;  
    }  
}
```

权限修饰符:

Private: 子类就无法访问了, 只能在本类中使用

比如: 爸爸的私房钱只能自己使用

注意事项:

- 子类只能访问父类中非私有成员
- 子类可以调用父类的非私有方法

4.1.1 继承的概念

复习访问控制

访问控制符	同一类中	同包子类	同包其他类	不同包子类	不同包其他类
public	√	√	√	√	√
protected	√	√	√	√	
default	√	√	√		
private	√				

子类到底能够继承父类中的哪些内容？

父类私有的东西，子类就无法继承？

父类中非私有方法，就被子类继承下来了？

子类到底能够继承父类中的哪些内容？

构造方法

非私有 不能

private 不能

成员变量

非私有 能

private 能

成员方法

非私有 能

private 不能

子类到底能够继承父类中的哪些内容?

构造方法是否可以被继承

```
public class Father{  
    String name;  
    int age;  
    public Father(){}  
    public Father(String name, int age){  
        this.name = name;  
        this.age = age;  
    }  
}
```

父类的构造方法不能被子类继承

```
public class son extends Father{  
    public Father(){}  
    public Father(String name, int age){  
        this.name = name;  
        this.age = age;  
    }  
}
```

子类到底能够继承父类中的哪些内容？

成员变量是否可以被继承

```
public class Father{  
    private String appearance;  
}
```



被封印了

```
public class Son extends Father{  
  
}
```

子类到底能够继承父类中的哪些内容？

继承的内存图

```
public class Father {  
    String name;  
    int age;  
}
```

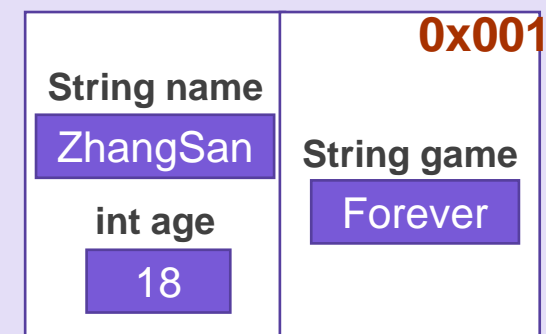
```
public class Son extends Father{  
    String game;  
}
```

```
public class TestMain {  
    public static void main(String[] args) {  
        Son s = new Son();  
        System.out.println(s);  
        s.name = "ZhangSan";  
        s.age = 18;  
        s.game = "Forever";  
        System.out.println(s.name+", "+s.age+", "+s.game);  
    }  
}
```

栈内存

方法: main
Son s **0x001**
sout(s);
s.name = "ZhangSan";
s.age = 18;
s.game = "Forever";
sout(s.name+", "+s.age+", "+s.game);

堆内存



TestMain.class

main();

Son.class

String name

Father.class

String name
int age

方法区

子类到底能够继承父类中的哪些内容？

继承的内存图

```
public class Father {  
    private String name;  
    private int age;  
}
```

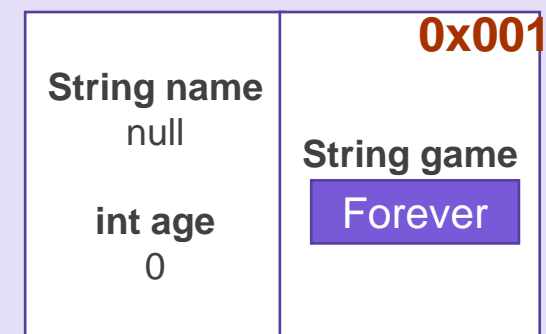
```
public class Son extends Father{  
    String game;  
}
```

```
public class TestMain {  
    public static void main(String[] args) {  
        Son s = new Son();  
        System.out.println(s);  
        s.name = "ZhangSan";  
        s.age = 18;  
        s.game = "Forever";  
        System.out.println(s.name+", "+s.age+", "+s.game);  
    }  
}
```

栈内存

方法: main
Son s **0x001**
sout(s);
s.name = "ZhangSan";
s.age = 18;
s.game = "Forever";
sout(s.name+", "+s.age+", "+s.game);

堆内存



TestMain.class

main();

Son.class

String game

Father.class

String name
int age

方法区

子类到底能够继承父类中的哪些内容?

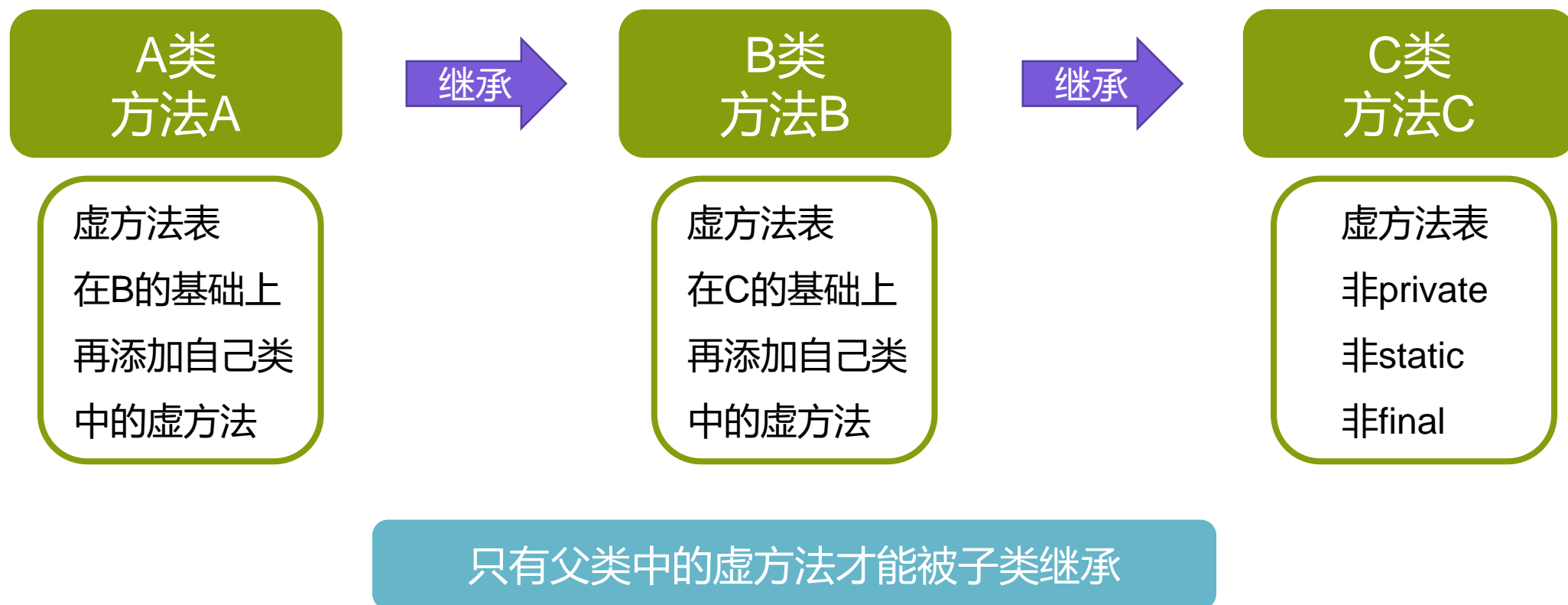
成员方法是否可以被继承



```
public class Test{  
    public static void main(String[] args){  
        A a = new A();  
        a.方法c();  
    }  
}
```

子类到底能够继承父类中的哪些内容？

成员方法是否可以被继承



子类到底能够继承父类中的哪些内容?

```
public class Father {  
    public void fShow1(){  
        System.out.println("Father show1 ----- public");  
    }  
    private void fShow2(){  
        System.out.println("Father show2 ----- private");  
    }  
}
```

```
public class Son extends Father{  
    public void sShow(){  
        System.out.println("Son show ----- public");  
    }  
}
```

```
public class TestMain {  
    public static void main(String[] args) {  
        Son s = new Son();  
        System.out.println(s);  
        s.sShow();  
        s.fShow1();  
        s.fShow2();  
    }  
}
```

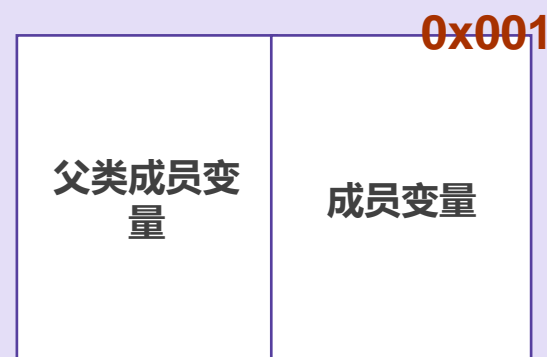
栈内存

方法: fShow1();
sout("Father show1 ----
public")

方法: sShow();
sout("Son show ----public")

方法: main
Son s 0x001
sout(s);
s.sShow();
s.fShow1();
s.fShow2();

堆内存



TestMain.class

main();

Son.class

所有成员方法

虚方法表
6个方法
sShow();

方法区

Father.class

所有成员方法

虚方法表
5个方法
fShow1();

Object.class

所有成员方法

虚方法表
5个方法

子类到底能够继承父类中的哪些内容？

构造方法

非私有 不能

private 不能

成员变量

非私有 能

private 能

成员方法

虚方法表能

否则 不能

4.1.2 方法的重写

方法的重写

当父类的方法不能满足子类现在的需求时，需要进行方法重写。

书写格式

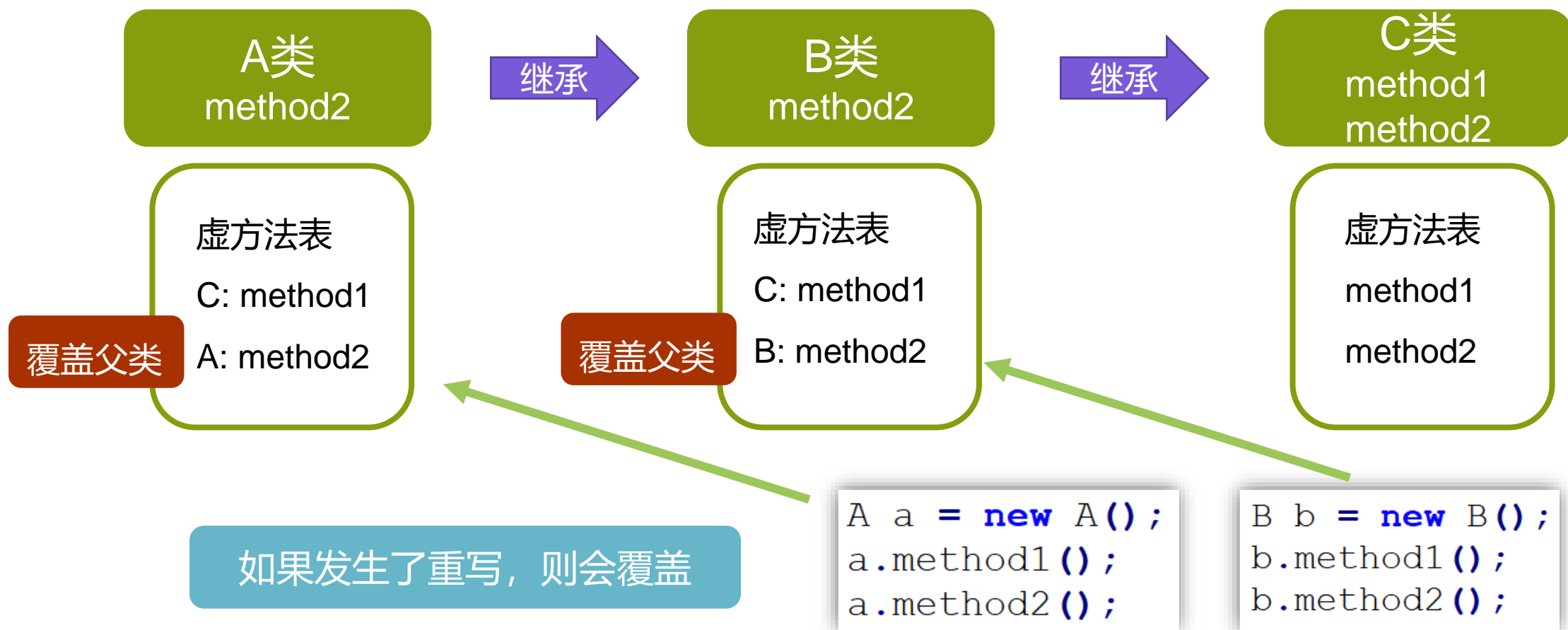
在继承体系中，子类出现了和父类一模一样的方法声明，我们就称子类这个方法是重写的方法。

@Override重写注解

1. @Override是放在重写后的方法上，校验子类重写时语法是否正确。
2. 加上注解后如果有红色波浪线，表示语法错误。
3. **建议重写方法都加@Override注解，代码安全，优雅！**

4.1.2 方法的重写

方法重写的本质



4.1.2 方法的重写

方法重写的本质

1. 重写方法的名称，形参列表必须与父类中的一致。
2. 子类重写父类方法时，访问权限子类必须大于等于父类 (default < protected < public)
3. 子类重写父类方法时，返回值类型子类必须等于父类。
4. **建议：重写的方法尽量和父类保持一致。**
5. 私有方法不能被重写。
6. 子类不能重写父类的静态方法，如果重写会报错的。

只有被添加到虚方法表中的方法才能被重写

继承中：成员变量的访问特点

```
public class Father{
    String name = "Father";
}
public class Son extends Father{
    String name = "Son";
    public void sShow(){
        String name = "sShow";
        System.out.println(name);
    }
}
```

就近原则：谁离我近，我就用谁

先在局部位置找，本类成员位置找，父类成员位置找，逐级往上。

继承中：成员变量的访问特点

```
public class Father{
    String name = "Father";
}
public class Son extends Father{
    String name = "Son";
    public void sShow(){
        String name = "sShow";
        System.out.println(name);
        System.out.println(this.name);
        System.out.println(super.name);
    }
}
```

从局部位置开始往上找

从本类成员位置开始往上找

从父类成员位置开始往上找

继承中：成员变量的访问特点

直接调用满足就近原则：谁离我近，我就用谁

super调用，直接访问父类

继承中：构造方法的特点

继承中：构造方法的访问特点

- 父类中的构造方法不会被子类继承。
- 子类中所有的构造方法默认先访问父类中的无参构造，再执行自己。

默认初始化

```
public class Father{  
    String name;  
    int age;  
    public Father() {}  
    public Father(String name, int age){  
        this.name = name;  
        this.age = age;  
    }  
}
```

```
public class Son extends Father{  
    public Son() {  
    }  
}
```

继承中：构造方法的特点

为什么？

- 子类在初始化的时候，有可能会使用到父类中的数据，如果父类没有完成初始化，子类将无法使用父类的数据。
- 子类初始化之前，一定要调用父类构造方法先完成父类数据空间的初始化。

怎么调用父类构造方法？

- 子类构造方法的第一行语句默认都是`super()`，不写也存在，且必须在第一行。
- 如果想调用父类有参构造，必须手写`super`进行调用。

继承中：构造方法的特点

继承中构造方法的访问特点是什么？

- 子类不能继承父类的构造方法，但是可以通过super调用。
- 子类构造方法的第一行，有一个默认的super();
- 默认先访问父类中无参的构造方法，再执行自己。
- 如果想要访问父类有参构造，必须要手动书写。

总结



4.1.3 super关键字

super关键字

当子类重写父类的方法后，子类对象将无法访问父类被重写的方法，而super关键字可以在子类中调用父类的普通属性，方法和构造方法。

书写格式

super.成员变量

super.成员方法（参数1， 参数2）

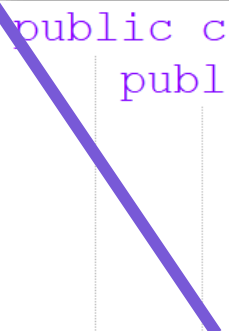
4.1.3 super关键字

this, super使用总结

- **this**: 理解为一个变量, 表示当前方法调用者的地址值;

```
public class Student{  
    String name;  
    int age;  
    public void show(Student this){  
        System.out.println(name + ", " + age);  
    }  
}
```

```
public class Test{  
    public static void main(String[] args){  
        Student s = new Student();  
        s.name = "ZhangSan";  
        s.age = 18;  
        s.show();  
    }  
}
```



4.1.3 super关键字

this, super使用总结

- **this**: 理解为一个变量，表示当前方法调用者的地址值；
- **super**: 代表父类存储空间。

关键字	访问成员变量	访问成员方法	访问构造方法
this	this.成员变量 访问本类成员变量	this.成员方法 访问本类成员方法	this(...) 访问本类构造方法
super	super.成员变量 访问父类成员变量	super.成员方法(...) 访问父类成员方法	super(...) 访问父类构造方法

什么是包

包就是文件夹。用来管理各种不同功能的Java类，方便后期代码维护。

包名的规则： 公司域名反写+包的作用，需要全部英文小写，见名知意。

Eg. cn.edu.cwnu.cs.xxx

```
package cn.edu.cwnu.cs.lqr03;
```

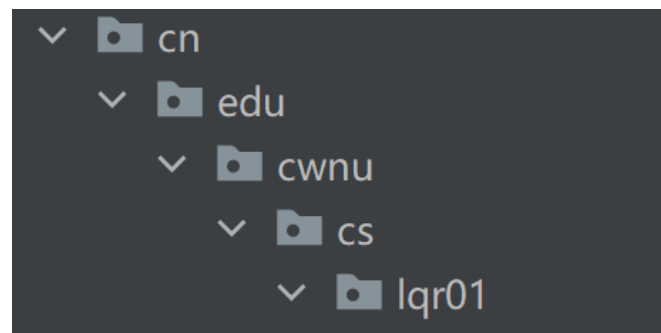
```
/**
```

```
 * @author: Li Qinru
```

```
 * @Date: 2022/10/25 11:28 PM
```

```
 */
```

```
public class OverseaStudent extends Person{
```



全类名

全限定名

cn.edu.cwnu.cs.lqr03.OverseaStudent

什么是包

使用其他类的规则

使用其他类时，需要使用全类名。

```
public class MainTest {  
    public static void main(String[] args) {  
        cn.edu.cwnu.cs.lqr03.OverseaStudent s = new cn.edu.cwnu.cs.lqr03.OverseaStudent();  
    }  
}
```

```
package cn.edu.cwnu.cs.lqr03;  
import cn.edu.cwnu.cs.lqr03.OverseaStudent;  
  
public class MainTest {  
    public static void main(String[] args) {  
        OverseaStudent s = new OverseaStudent();  
    }  
}
```

什么是包

使用其他类的规则

- 使用同一个包中的类时，不需要导包。
- 使用java. lang包中的类时，不需要导包。
- 其他情况都需要导包
- 如果同时使用两个包中的同名类，需要用全类名。

什么是包

包的作用?

包就是文件夹，用来管理各种不同功能的Java类

包名书写的规则?

公司域名反写+包的作用，需要全部英文小写，见名知意。

什么是全类名?

包名+类名

什么时候需要导包? 什么时候不需要导包?

- 使用同一个包中的类时，不需要导包。
- 使用java.lang包中的类时，不需要导包。
- 其他情况都需要导包
- 如果同时使用两个包中的同名类，需要用全类名。

总结



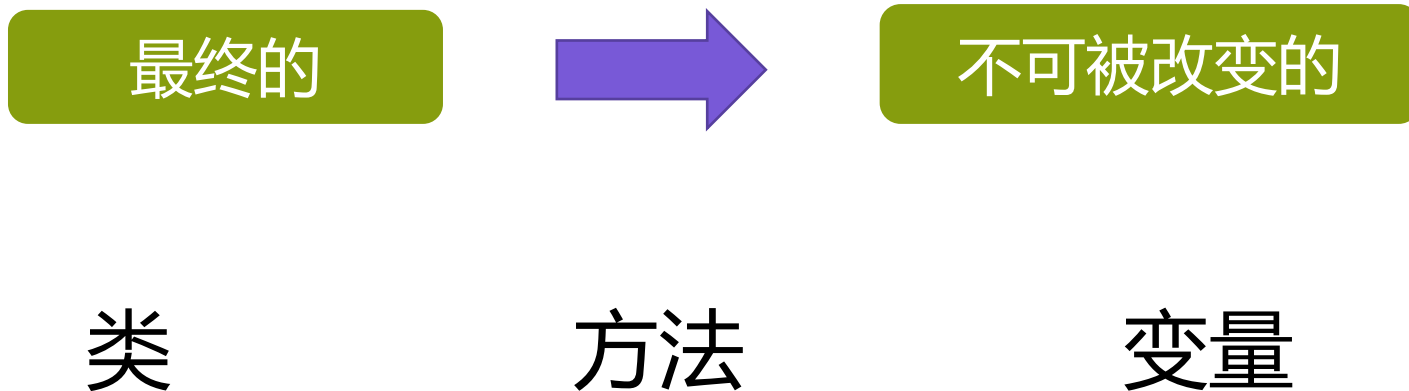
/4.2

final关键字



4.2 final关键字

final英文意思是最终



4.2 final关键字

类

表明该类是最终类，不能被继承

方法

表明该方法是最最终方法，不能被重写

变量

叫做常量，只能被赋值一次

4.2.1 final关键字修饰类

类

表明该类是最终类，不能被继承

```
public final class Father {  
    public void show(){  
        System.out.println("Father's show method");  
    }  
}
```

```
public class Son extends Father{  
    public void show(){  
        System.out.println("Son's show method");  
    }  
}
```

Cannot inherit from final 'cn.edu.cwnu.cs.lqr05.Father'

Make 'Father' not final Alt+Shift+Enter More actions... Alt+Enter

4.2.2 final关键字修饰方法

方法

表明该方法是最最终方法，不能被重写

```
public class Father {  
    1 related problem  
    public final void show(){  
        System.out.println("Father's show method");  
    }  
}
```

```
public class Son extends Father{  
    public void show(){  
        System.out.println("Father's show method");  
    }  
}
```

'show()' cannot override 'show()' in 'cn.edu.cwnu.cs.lqr05.Father'; overridden method is final

Make 'Father.show' not final Alt+Shift+Enter More actions... Alt+Enter

4.2.3 final关键字修饰变量

变量

叫做常量，只能被赋值一次

```
public class Test {  
    public static void main(String[] args) {  
        final int a = 10;  
        System.out.println(a);  
        a = 20;  
    }  
}
```

二次赋值时编译器会报错

Cannot assign a value to final variable 'a'

Make 'a' not final Alt+Shift+Enter

More actions... Alt+Enter

4.2.3 final关键字修饰变量

注意：

实际开发中，常量一般作为系统的配置信息，方便维护，提高可读性。

public static final声明，则此变量将成为全局变量。

常量的命名规范：

- 单个单词：全部大写
- 多个单词：全部大写，单词之间用下划线隔开

细节：

Final修饰的变量是基本类型：那么变量存储的**数据值**不能发生改变。

Final修饰的变量是引用类型：那么变量存储的**地址值**不能发生改变，对象内部的可以改变。

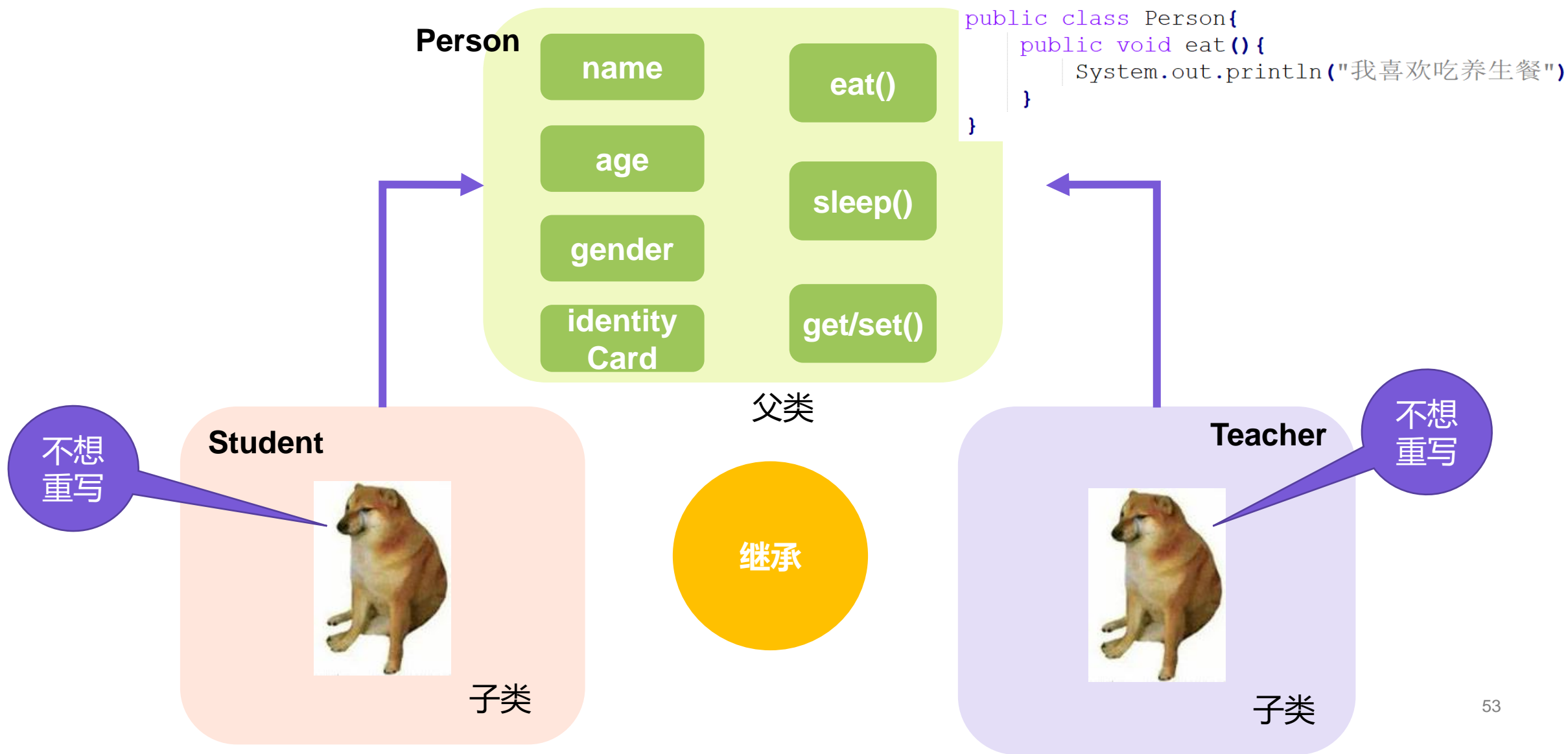
核心：常量记录的数据是不能发生改变的。

/4.3

抽象类和接口



4.3.1 抽象类



4.3.1 抽象类



```
public abstract class Person{  
    public abstract void eat() {  
        System.out.println("我喜欢吃养生餐")  
    }  
}
```

抽象类的定义格式

抽象方法的定义格式

子类继承抽象类之后，如何重写抽象方法

4.3.1 抽象类

抽象方法

- 抽象方法：将**共性的**行为（**方法**）抽取到父类之后。

由于每一个子类执行的内容是不一样的，

所以，在父类中不能确定**具体的方法体**。

该方法就可以定义为抽象方法。

- 抽象类：如果一个**类中存在抽象方法**，那么该类就**必须声明为抽象类**

4.3.1 抽象类

抽象类和抽象方法的定义格式

- 抽象方法的定义格式:

public **abstract** 返回值类型 方法名(参数列表);

- 抽象类的定义格式:

public **abstract** class 类名{

4.3.1 抽象类

抽象类和抽象方法的注意事项

- 抽象类不能实例化(抽象类不能创建对象)
- 抽象类中不一定有抽象方法，有抽象方法的类一定是抽象类
- 可以有构造方法
- 抽象类的子类

要么重写抽象类中的所有抽象方法

要么是抽象类

4.3.1 抽象类

抽象类和抽象方法的意义

疑问：

- 把子类中共性的内容抽取到父类后，
- 由于方法体不确定，需要定义为抽象。子类使用时需要重写。
- 那么我不抽取到父类，直接在子类写不是更节约代码？

4.3.1 抽象类

抽象类和抽象方法的意义

```
public abstract class Person{  
    public void drink(){  
        System.out.println("我喜欢喝水");  
    }  
}
```

```
public class Student extends Person{  
    public void drink(){  
        System.out.println("爱喝饮料");  
    }  
}
```

```
public class Teacher extends Person{  
    public void drink(){  
        System.out.println("爱喝枸杞泡大枣");  
    }  
}
```

4.3.1 抽象类

抽象类和抽象方法的意义

程序员，公务员，销售

```
public abstract class Person{  
    public void drink(){  
        System.out.println("我喜欢喝水");  
    }  
}
```

```
public class Student extends Person{  
    public void heShui(String name){  
        System.out.println("爱喝"+ name);  
    }  
}
```

```
public class Teacher extends Person{  
    public String drink(){  
        System.out.println("爱喝枸杞泡大枣");  
        return "咕噜咕噜";  
    }  
}
```

4.3.1 抽象类

抽象类和抽象方法的意义

```
public abstract class Person{  
    public abstract void drink();  
}
```

强制子类必须按照这种格式进行重写

```
public class Student extends Person{  
    @Override  
    public void drink(){  
        System.out.println("爱喝饮料");  
    }  
}
```

```
public class Teacher extends Person{  
    @Override  
    public void drink(){  
        System.out.println("爱喝枸杞泡红枣");  
    }  
}
```

4.3.1 抽象类

1. 抽象类的作用是什么？

- 抽取共性时，无法确定方法体，就把方法定义为抽象的。
- 强制让子类按照某种格式重写。
- 抽象方法所在的类，必须是抽象类。

2. 格式：

- `public abstract 返回值类型 方法名(参数列表);`
- `public abstract class 类名{`

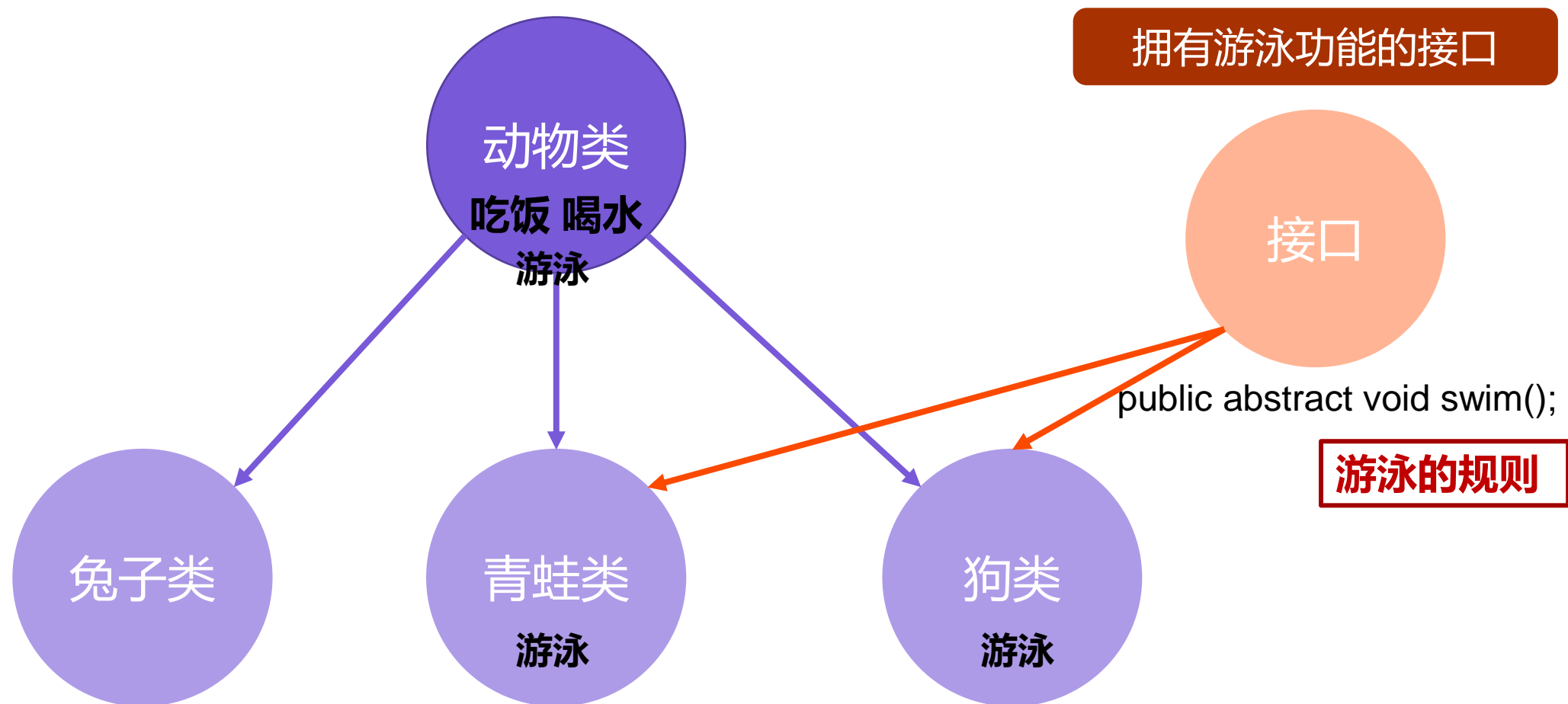
3. 继承抽象类注意事项：

- 要么重写抽象类中的所有抽象方法
- 要么是抽象类

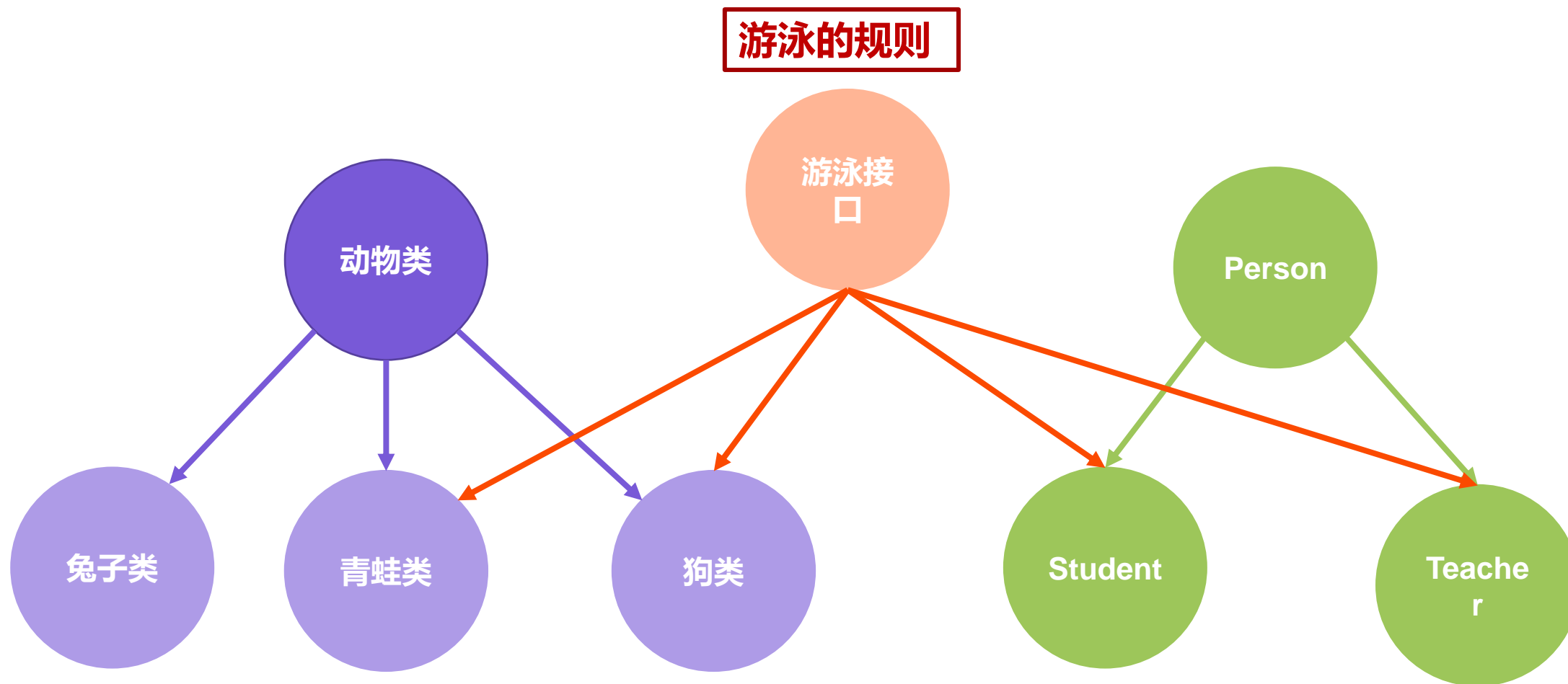
总结



4.3.2 接口



4.3.2 接口和抽象类的异同



接口就是一种规则，是对行为的抽象

4.3.2 接口的定义和使用

如何定义一个接口？

如何使用一个接口？

接口里面成员的特点？

4.3.2 接口的定义和使用

- 接口用关键字interface来定义

public **interface** 接口名{

- 接口不能实例化

- 接口和类之间是实现关系，通过implements关键字表示

public class 类名 **implements** 接口名{

- 接口的子类(实现类)

要么重写接口中的所有抽象方法

要么是抽象类

注意1： 接口和类的实现关系，可以单实现，也可以多实现。

public class 类名 **implements** 接口名1, 接口名2{

注意2： 实现类还可以在继承一个类的同时实现多个接口

public class 类名 extends 父类 **implements** 接口名1, 接口名2{

4.3.2 接口中成员的特点

- 成员变量

只能是常量

默认修饰符: **public** static final

- 构造方法

没有

- 成员方法

抽象方法: public abstract(如果什么都不写, 默认)

静态方法: (public) static void + 方法名

默认方法: (public) default void + 方法名

- **JDK7以前: 接口中只能定义抽象方法。**

- **JDK8的新特性: 接口中可以定义有方法体的方法**

4.3.2 接口和类之间的关系

- 类和类的关系

继承关系，只能单继承，不能多继承，但是可以多层继承

- 类和接口的关系

实现关系，可以单实现，也可以多实现，还可以在继承一个类的同时实现多个接口

- 接口和接口的关系

继承关系，可以单继承，也可以多继承

4.3.2 接口和抽象类的标准Javabeans类

我们现在有乒乓球运动员和篮球运动员，乒乓球教练和篮球教练。

为了出国交流，跟乒乓球相关人员都需要学习英语。

请用所有只是分析，在这个案例中，哪些是具体类，哪些是抽象类，哪些是接口？

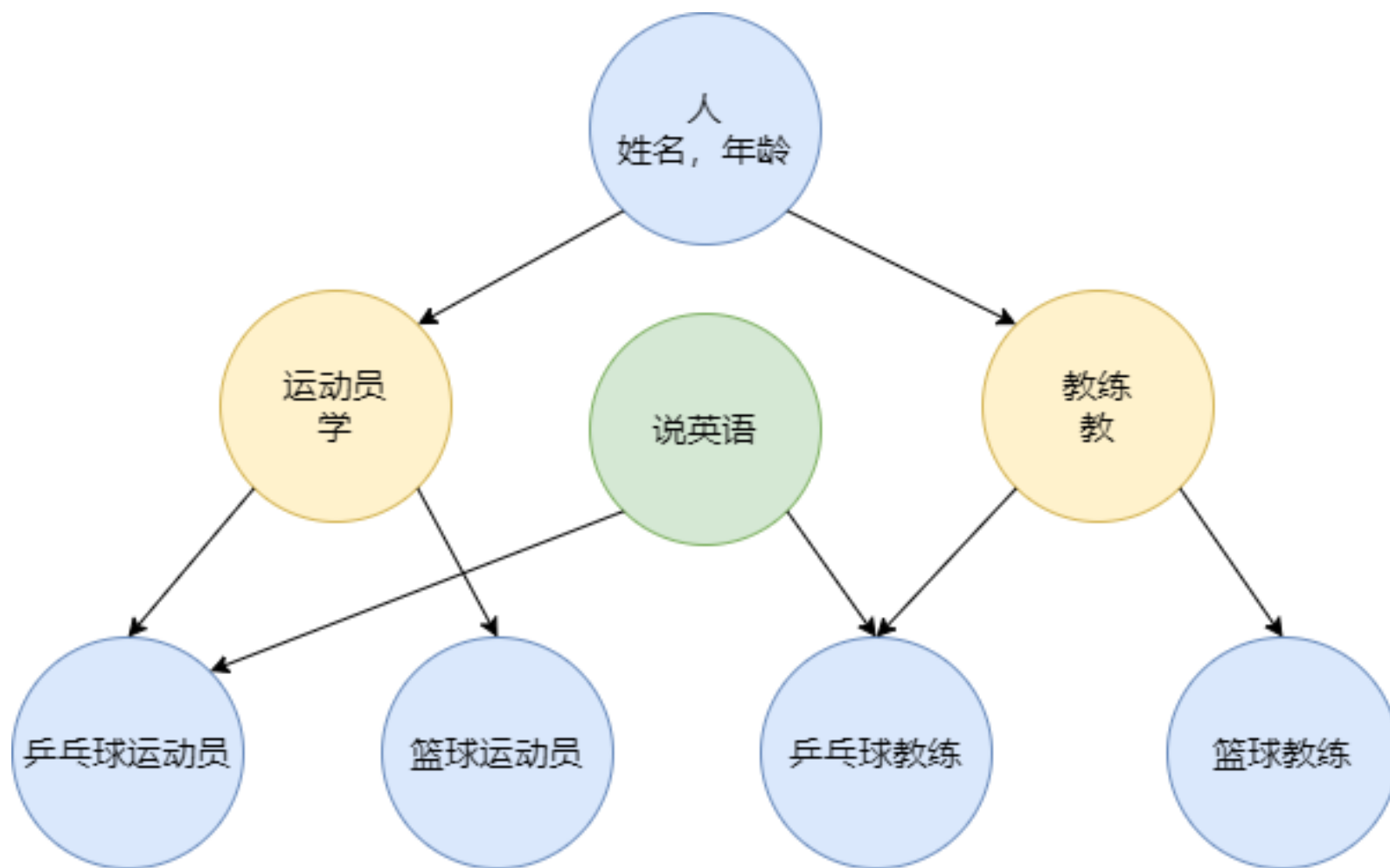
乒乓球运动员：姓名，年龄，学打乒乓球，说英语

篮球运动员：姓名，年龄，学打篮球

乒乓球教练：姓名，年龄，教打乒乓球，说英语

篮球教练：姓名，年龄，教打篮球

4.3.2 接口和抽象类的标准Javabean类



4.3.2 接口的一些小扩展

JDK8开始接口中新增的方法

接口的应用

适配器设计模式

4.3.2 接口的一些小扩展

JDK8开始接口中新增的方法

- **JDK7以前：** 接口中只能定义抽象方法。
- **JDK8的新特性：** 接口中可以定义有方法体的方法。（默认, 静态）
- **JDK9的新特性：** 接口中可以定义私有方法。

4.3.2 接口的一些小扩展

```
public interface Inter{  
    public abstract int method();  
}
```

此时实现类就不需要立马修改了，等以后用到某个规则了，就重写。

```
public class InterImpl1 implements Inter{  
    public int method(){  
        ...  
    }  
}
```

```
public class InterImpl2 implements Inter{  
    public int method(){  
        ...  
    }  
}
```

4.3.2 接口的一些小扩展

JDK8以后接口中新增的方法

允许在接口中定义默认方法，需要使用关键字default修饰

作用：解决接口升级的问题

接口中默认方法的定义格式：

- 格式：public default 返回值类型 方法名(参数列表){ }
- 范例：public default void show(){ }

接口中默认方法的注意事项：

- 默认方法不是抽象方法，所以不强制被重写。但是如果被重写，重写的时候去掉default关键字
- public可以省略，default不能省略
- 如果实现了多个接口，多个接口中存在相同名字的默认方法，子类就必须对该方法进行重写

4.3.2 接口的一些小扩展

JDK8以后接口中新增的方法

允许在接口中定义静态方法，需要用static修饰

接口中静态方法的定义格式：

- 格式：public static 返回值类型 方法名(参数列表){ }
- 范例：public static void show(){ }

接口中静态方法的注意事项：

- 静态方法只能通过接口名调用，不能通过实现类名或者对象名调用
- public可以省略，static不能省略

4.3.2 接口的一些小扩展

JDK9新增的方法

```
interface Inter{  
    public default void start(){  
        System.out.println("start方法执行...");  
        System.out.println("记录日志的N多行代码...");  
    }  
  
    public default void end(){  
        System.out.println("end方法执行...");  
        System.out.println("记录日志的N多行代码...");  
    }  
}
```

4.3.2 接口的一些小扩展

JDK9以前的做法

```
interface Inter{  
    public default void start(){  
        System.out.println("start方法执行...");  
        log();  
    }  
  
    public default void end(){  
        System.out.println("end方法执行...");  
        log();  
    }  
}
```

如果这么写有一个问题，外类可以访问

```
p private void log(){  
    System.out.println("日志记录");  
}
```

4.3.2 接口的一些小扩展

JDK9以后接口中新增的方法

接口中私有方法的定义格式:

- 格式1: `private返回值类型 方法名(参数列表){ }`
- 范例1: `private void show(){ }`
- 格式2: `private static 返回值类型 方法名(参数列表){ }`
- 范例2: `private static void show(){ }`

4.3.2 接口的一些小扩展

总结

1. JDK7以前：接口中只能定义抽象方法。
2. JDK8：接口中可以定义有方法体的方法。（默认，静态）
3. JDK9：接口中可以定义私有方法。
4. 私有方法分为两种：普通的私有方法，静态的私有方法



/4.4

多态



4.4 多态

面向对象三大特征

封装

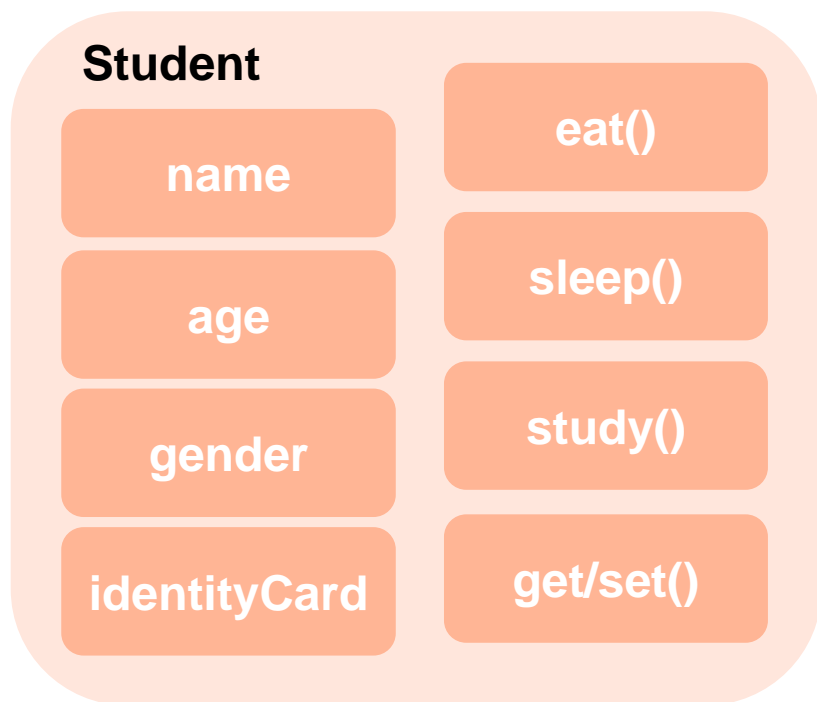
继承

多态

4.4.1 多态概述

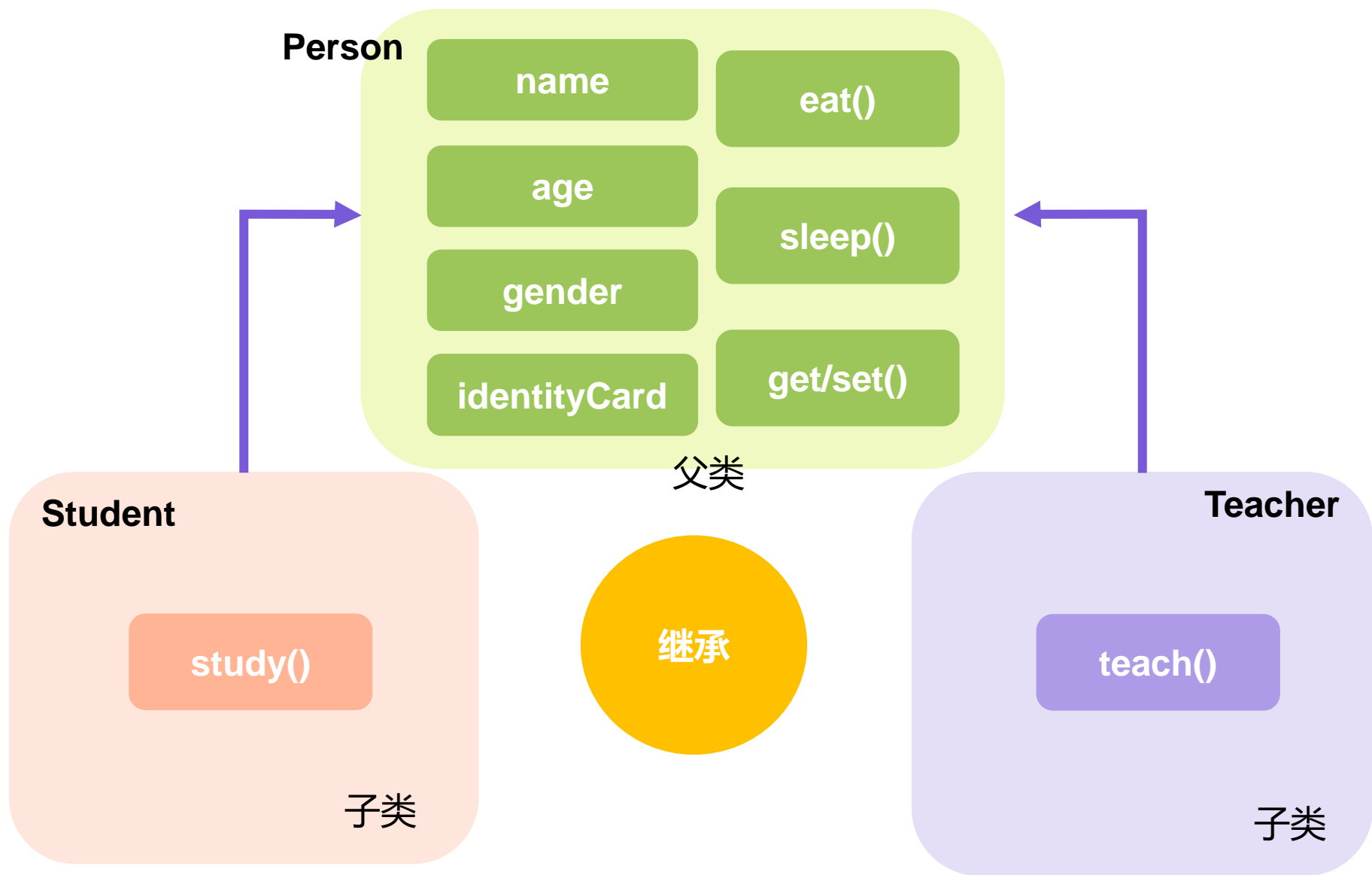
封装:

对象代表什么，就得封装对应的数据，并提供数据对应的行为。



```
public class TestClass {  
    public static void printStudentInfo(Student s){  
        //...  
    }  
}
```

4.4.1 多态概述



4.4.1 多态概述

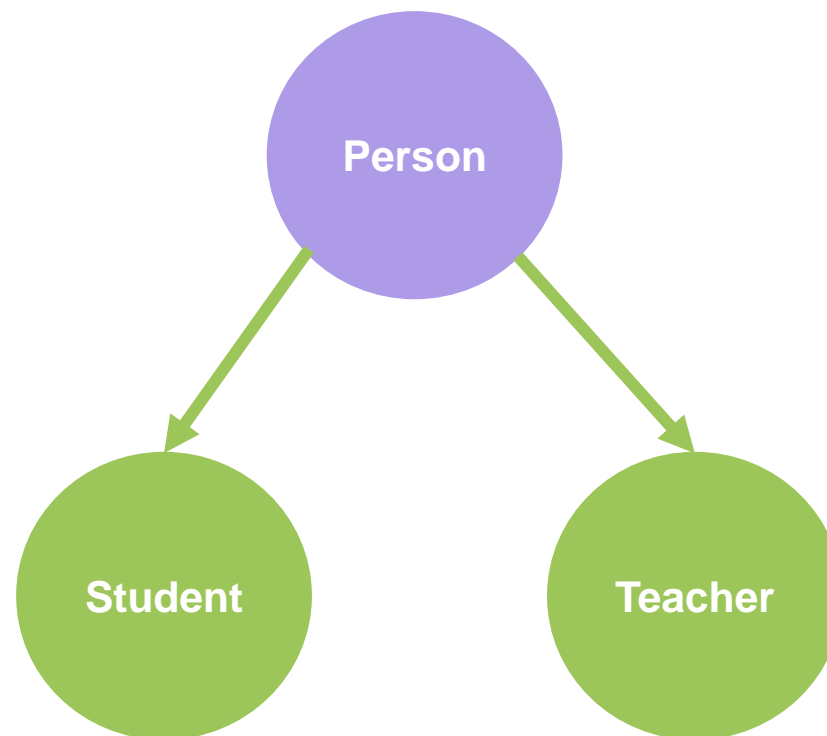
指对象的多种形态

学生形态 **对象**

```
Student s = new Student();
```

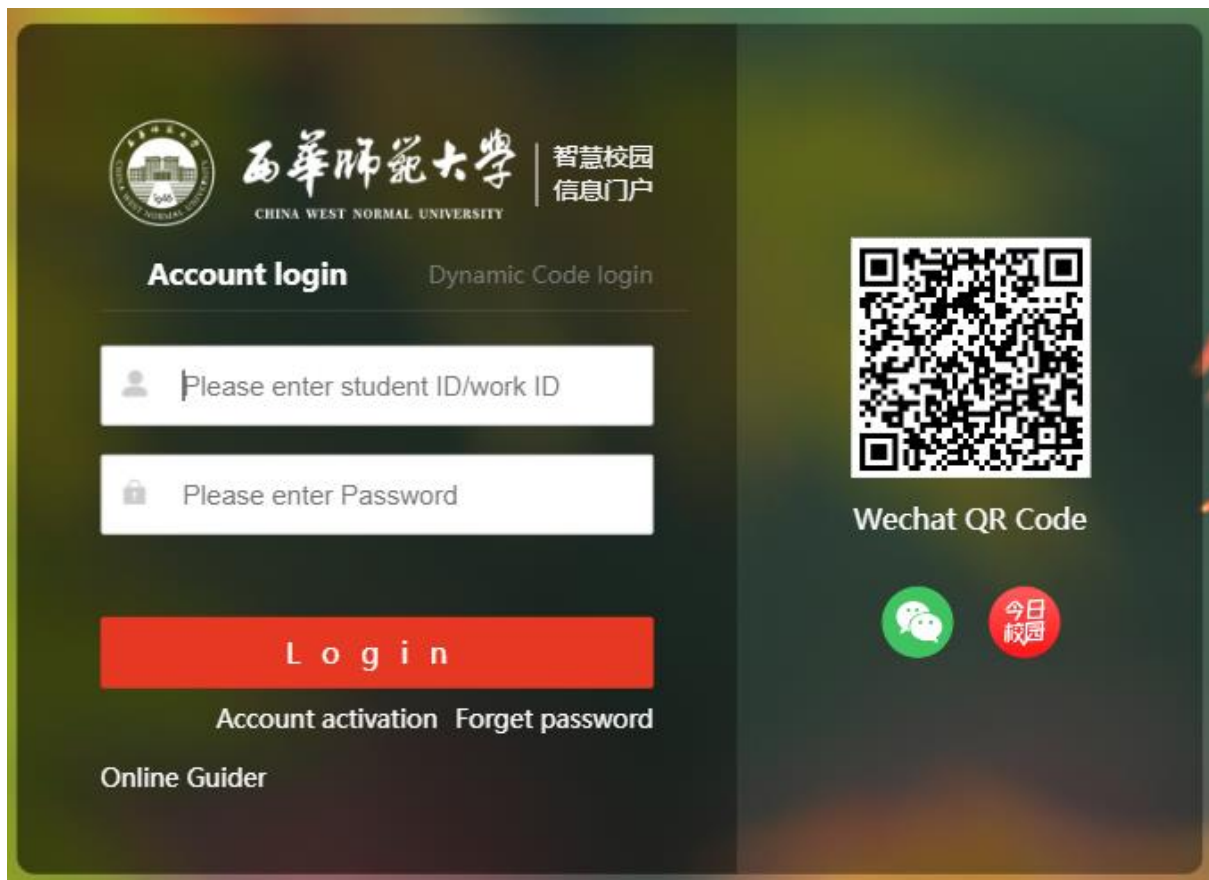
人形态 **对象**

```
Person p = new Student();
```



4.4.1 多态概述

多态的应用场景



Student

Teacher

Admin

4.4.1 多态概述

多态的应用场景

```
public void register(???) {  
    //方法里面就是注册的代码逻辑  
}
```

Student

Teacher

Admin

4.4.1 多态概述

多态的应用场景

```
public void register(Person p) {  
    p.show();  
}
```

根据传递对象的不同，调用不同的show方法

new Student();

new Teacher();

new Admin();

4.4.1 多态概述

什么是多态?

同类型的对象，表现出的不同形态。

多态的表现形式

父类类型 对象名称 = 子类对象;

多态的前提

- 有继承/实现关系;
- 有父类引用指向子类对象
- 方法的重写

Fu f = new Zi();

4.4.1 多态概述

1. 什么是多态?

对象的多种形态。

2. 多态的前提?

- 有继承/实现关系
- 有父类引用指向子类对象
- 有方法的重写

3. 多态的好处?

使用父类型作为参数，可以接收所有子类对象，体现多态的扩展性与便利性。



4.4.2 对象类型转换

多态调用成员的特点

- **变量调用：编译看左边，运行也看左边**

编译看左边：javac编译代码时，会看左边的父类中有没有这个变量，如果有，编译成功，如果没有编译失败。

运行也看左边：java运行代码的时候，实际获取的就是左边父类中成员变量的值。

- **方法调用：编译看左边，运行看右边**

编译看左边：javac编译代码的时候，会看左边父类中有没有这个方法，如果有，编译成功，如果没有编译失败。

运行看右边：java运行代码的时候，实际上运行子类中的方法。

4.4.2 对象类型转换

理解：

Person a = new Student();

现在是用a去调用变量和方法的。

而a是Person类型的，所以默认都会从Person这个类中去找

- **成员变量**：在子类的对象中，会把父类的成员变量也继承下来。父：name 子：name.
- **成员方法**：如果子类对方法进行了重写，那么在虚方法中是会把父类的方法进行覆盖的。

4.4.2 对象类型转换

向上转型:

父类类型 父类对象 = 子类实例

向下转型:

父类类型 父类对象 = 子类实例

子类类型 子类对象 = (子类)父类对象

向下转型可以调用子类类型中所有的成员

4.4.2 对象类型转换

注意:

- Java 编译器允许在具有直接或间接继承关系的类之间进行类型转换。对于向下转型，必须进行强制类型转换；对于向上转型，不必使用强制类型转换。
- 类型强制转换时想运行成功就必须保证父类引用指向的对象一定是该子类对象，最好使用 instanceof 运算符判断后，再强转

```
Person a = new Student();  
if (a instanceof Student) {  
    Student s = (Student)a ; // 向下转型  
}
```

4.4.3 instanceof关键字

instanceof是Java中的**二元运算符**，左边是对象，右边是类。

当对象是右边类或子类所创建对象时，返回true；否则，返回false。

对象 instanceof 类（或接口）

```
Person p1 = new Student();
System.out.println("p1: Person " + (p1 instanceof Person));
System.out.println("p1: Student " + (p1 instanceof Student));
Student p2 = new Student();
System.out.println("p2: Person " + (p2 instanceof Person));
System.out.println("p2: Teacher " + (p2 instanceof Teacher));
```

/4.5

Object类



4.5 Object类

Object概述:

- Object是Java中的顶级父类。所有的类都直接或间接的继承于Object类
- Object类中的方法可以被所有子类访问，所以我们要学习Object类和其中的方法

4.5 Object类

Object的构造方法

方法名	说明
public Object()	空参构造

4.5 Object类

Object的成员方法

方法名	说明
public String toString()	返回对象的字符串表示形式
public Boolean equals(Object obj)	比较两个对象是否对等
public int hashCode()	返回对象的散列码值

4.5 Object类

toString方法的结论:

1. 如果我们打印一个对象，想要看到属性值的话，那么就重写toString方法就可以了。
2. 再重写的方法中，把对象的属性值进行拼接。

4.5 Object类

equals方法的结论:

1. 如果没有重写equals方法，那么默认使用Object中的方法进行
比较，比较的是地址值是否相等。
2. 一般来说比较地址值对于我们意义不大，所以我们会重写，重
写之后比较的就是对象内部的属性值了。