面向对象进阶部分学习方法：

特点：

逻辑性没有那么强，但是概念会比较多。

记忆部分重要的概念，理解课堂上讲解的需要大家掌握的概念，多多练习代码。

day13

今日内容

- 复习回顾
- static关键字
- 继承

教学目标

- √ 能够掌握static关键字修饰的变量调用方式
- √ 能够掌握static关键字修饰的方法调用方式
- √ 知道静态代码块的格式和应用场景
- √ 能够写出类的继承格式
- √ 能够说出继承的特点
- √ 能够区分this和super的作用
- √ 能够说出方法重写的概念
- √ 能够说出方法重写的注意事项

第一章 复习回顾

1.1 如何定义类

类的定义格式如下:

```
修饰符 class 类名 {
    // 1.成员变量（属性）
    // 2.成员方法 (行为)
    // 3.构造方法 （初始化类的对象数据的）
}
```

例如:

```
public class Student {
    // 1.成员变量
    public String name ;
    public char sex ; // '男'  '女'
    public int age;
}
```

1.2 如何通过类创建对象

```
类名 对象名称 = new 类名();
```

例如:

```
Student stu = new Student();
```

1.3 封装

1.3.1 封装的步骤

1.使用 `private` 关键字来修饰成员变量。

2.使用`public`修饰getter和setter方法。

1.3.2 封装的步骤实现

1. private修饰成员变量

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

2. public修饰getter和setter方法

```
public class Student {
    private String name;
    private int age;

    public void setName(String n) {
        name = n;
    }

    public String getName() {
        return name;
    }

    public void setAge(int a) {
        if (a > 0 && a <200) {
            age = a;
        } else {
            System.out.println("年龄非法！");
        }
    }

    public int getAge() {
        return age;
    }
}
```

1.4 构造方法

1.4.1 构造方法的作用

在创建对象的时候，给成员变量进行初始化。

初始化即赋值的意思。

1.4.2 构造方法的格式

```
修饰符 类名(形参列表) {
    // 构造体代码，执行代码
}
```

1.4.3 构造方法的应用

首先定义一个学生类，代码如下：

```
public class Student {
    // 1.成员变量
    public String name;
    public int age;

    // 2.构造方法
    public Student() {
        System.out.println("无参数构造方法被调用");
    }
}
```

接下来通过调用构造方法得到两个学生对象。

```
public class CreateStu02 {
    public static void main(String[] args) {
        // 创建一个学生对象
        // 类名 变量名称 = new 类名();
        Student s1 = new Student();
        // 使用对象访问成员变量，赋值
        s1.name = "张三";
        s1.age = 20 ;

```

```
        // 使用对象访问成员变量 输出值
        System.out.println(s1.name);
        System.out.println(s1.age);

        Student s2 = new Student();
        // 使用对象访问成员变量 赋值
        s2.name = "李四";
        s2.age = 18 ;
        System.out.println(s2.name);
        System.out.println(s2.age);
    }
}
```

1.5 this关键字的作用

1.5.1 this关键字的作用

this代表所在类的当前对象的引用（地址值），即代表当前对象。

1.5.2 this关键字的应用

1.5.2.1 用于普通的gettter与setter方法

this出现在实例方法中，谁调用这个方法（哪个对象调用这个方法），this就代表谁（this就代表哪个对象）。

```
public class Student {
    private String name;
    private int age;

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setAge(int age) {
```

```
        if (age > 0 && age < 200) {
            this.age = age;
        } else {
            System.out.println("年龄非法! ");
        }
    }

    public int getAge() {
        return age;
    }
}
```

1.5.2.2 用于构造方法中

this出现在构造方法中，代表构造方法正在初始化的那个对象。

```
public class Student {
    private String name;
    private int age;

    // 无参数构造方法
    public Student() {}

    // 有参数构造方法
    public Student(String name,int age) {
        this.name = name;
        this.age = age;
    }
}
```

第二章 static关键字

2.1 概述

以前我们定义过如下类：

```
public class Student {
```

```
    // 成员变量
    public String name;
    public char sex; // '男'  '女'
    public int age;

    // 无参数构造方法
    public Student() {

    }

    // 有参数构造方法
    public Student(String  a) {

    }
}
```

我们已经知道面向对象中，存在类和对象的概念，我们在类中定义了一些成员变量，例如name,age,sex ,结果发现这些成员变量，每个对象都存在（因为每个对象都可以访问）。

而像name ,age , sex确实是每个学生对象都应该有的属性，应该属于每个对象。

所以Java中成员（**变量和方法**）等是存在所属性的，Java是通过static关键字来区分的。**static关键字在Java开发非常的重要，对于理解面向对象非常关键。**

关于 `static` 关键字的使用，它可以用来修饰的成员变量和成员方法，被static修饰的成员是**属于类**的是放在静态区中，没有static修饰的成员变量和方法则是**属于对象**的。我们上面案例中的成员变量都是没有static修饰的，所以属于每个对象。

2.2 定义格式和使用

static是静态的意思。 static可以修饰成员变量或者修饰方法。

2.2.1 静态变量及其访问

有static修饰成员变量，说明这个成员变量是属于类的，这个成员变量称为**类变量**或者**静态成员变量**。 直接用 类名访问即可。因为类只有一个，所以静态成员变量在内存区域中也只存在一份。所有的对象都可以共享这个变量。

如何使用呢

例如现在我们需要定义传智全部的学生类，那么这些学生类的对象的学校属性应该都是“传智”，这个时候我们可以把这个属性定义成static修饰的静态成员变量。

定义格式

```
修饰符 static 数据类型 变量名 = 初始值;
```

举例

```
public class Student {
    public static String schoolName = "传智播客";  // 属于类，只有一份。
    // .....
}
```

静态成员变量的访问：

格式：类名.静态变量

```
public static void  main(String[] args){
    System.out.println(Student.schoolName); // 传智播客
    Student.schoolName = "黑马程序员";
    System.out.println(Student.schoolName); // 黑马程序员
}
```

2.2.2 实例变量及其访问

无static修饰的成员变量属于每个对象的， 这个成员变量叫**实例变量**，之前我们写成员变量就是实例成员变量。

需要注意的是：实例成员变量属于每个对象，必须创建类的对象才可以访问。

格式：对象.实例成员变量

2.2.3 静态方法及其访问

有static修饰成员方法，说明这个成员方法是属于类的，这个成员方法称为**类方法或者**静态方法**。 直接用 类名访问即可。因为类只有一个，所以静态方法在内存区域中也只存在一份。所有的对象都可以共享这个方法。

与静态成员变量一样，静态方法也是直接通过**类名.方法名称**即可访问。

举例

```
public class Student{
    public static String schoolName = "传智播客";  // 属于类，只有一份。
    // .....
    public static void study(){
        System.out.println("我们都在黑马程序员学习");
    }
}
```

静态成员变量的访问：

格式：类名.静态方法

```
public static void  main(String[] args){
    Student.study();
}
```

2.2.4 实例方法及其访问

无static修饰的成员方法属于每个对象的，这个成员方法也叫做**实例方法**。

需要注意的是：实例方法是属于每个对象，必须创建类的对象才可以访问。

格式：对象.实例方法

示例：

```
public class Student {
    // 实例变量
    private String name ;
    // 2.方法：行为
    // 无 static修饰，实例方法。属于每个对象，必须创建对象调用
    public void run(){
        System.out.println("学生可以跑步");
    }
    // 无 static修饰，实例方法
    public  void sleep(){
```

```
        System.out.println("学生睡觉");
    }
    public static void study(){

    }
}
```

```
public static void main(String[] args){
    // 创建对象
    Student stu = new Student ;
    stu.name = "徐干";
    // Student.sleep();// 报错，必须用对象访问。
    stu.sleep();
    stu.run();
}
```

2.3 小结

1.当 `static` 修饰成员变量或者成员方法时，该变量称为**静态变量**，该方法称为**静态方法**。该类的每个对象都**共享**同一个类的静态变量和静态方法。任何对象都可以更改该静态变量的值或者访问静态方法。但是不推荐这种方式去访问。因为静态变量或者静态方法直接通过类名访问即可，完全没有必要用对象去访问。

2.无static修饰的成员变量或者成员方法，称为**实例变量，实例方法**，实例变量和实例方法必须创建类的对象，然后通过对象来访问。

3.static修饰的成员属于类，会存储在静态区，是随着类的加载而加载的，且只加载一次，所以只有一份，节省内存。存储于一块固定的内存区域（静态区），所以，可以直接被类名调用。它优先于对象存在，所以，可以被所有对象共享。

4.无static修饰的成员，是属于对象，对象有多少个，他们就会出现多少份。所以必须由对象调用。

第三章 继承

3.1 概述

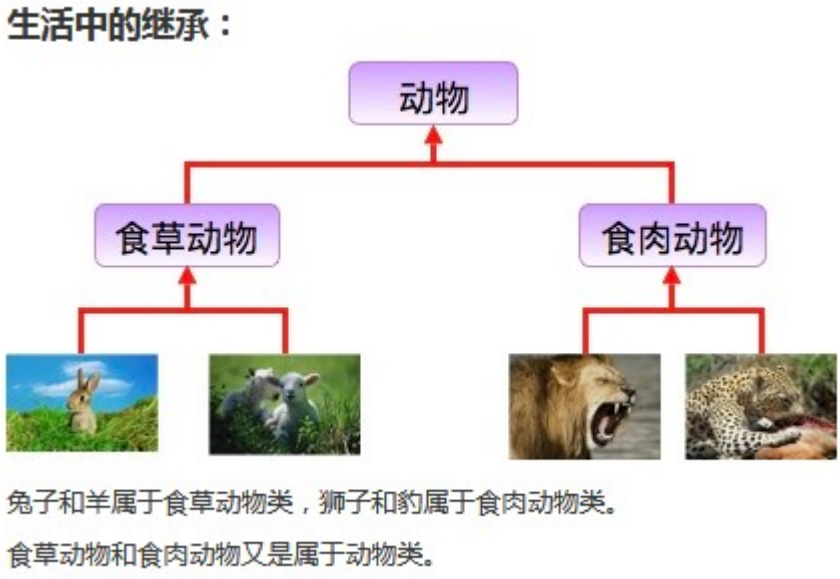
3.1.1 引入

假如我们要定义如下类:
学生类,老师类和工人类，分析如下。

1. 学生类
   属性:姓名,年龄
   行为:吃饭,睡觉
2. 老师类
   属性:姓名,年龄，薪水
   行为:吃饭,睡觉，教书
3. 班主任
   属性:姓名,年龄，薪水
   行为:吃饭,睡觉，管理

如果我们定义了这三个类去开发一个系统，那么这三个类中就存在大量重复的信息（属性:姓名，年龄。行为：吃饭，睡觉）。这样就导致了相同代码大量重复，代码显得很臃肿和冗余，那么如何解决呢？

假如多个类中存在相同属性和行为时，我们可以将这些内容抽取到单独一个类中，那么多个类无需再定义这些属性和行为，只要**继承**那一个类即可。如图所示：



其中，多个类可以称为**子类**，单独被继承的那一个类称为**父类、超类（superclass）**或者**基类**。

3.1.2 继承的含义

继承描述的是事物之间的所属关系，这种关系是：`is-a` 的关系。例如，兔子属于食草动物，食草动物属于动物。可见，父类更通用，子类更具体。我们通过继承，可以使多种事物之间形成一种关系体系。

继承：就是子类继承父类的**属性**和**行为**，使得子类对象可以直接具有与父类相同的属性、相同的行为。子类可以直接访问父类中的**非私有**的属性和行为。

3.1.3 继承的好处

1. 提高**代码的复用性**（减少代码冗余，相同代码重复利用）。
2. 使类与类之间产生了关系。

3.2 继承的格式

通过 `extends` 关键字，可以声明一个子类继承另外一个父类，定义格式如下：

```
class 父类 {
    ...
}

class 子类 extends 父类 {
    ...
}
```

需要注意：Java是单继承的，一个类只能继承一个直接父类，跟现实世界很像，但是Java中的子类是更加强大的。

3.3 继承案例

3.3.1 案例

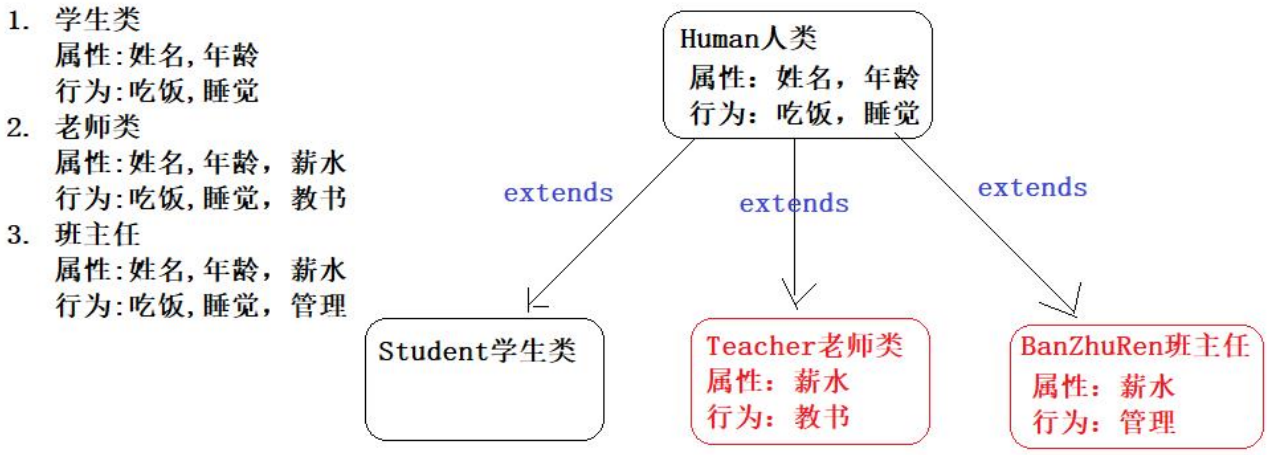
请使用继承定义以下类:

1. 学生类
   属性:姓名,年龄
   行为:吃饭,睡觉
2. 老师类
   属性:姓名,年龄，薪水
   行为:吃饭,睡觉，教书
3. 班主任
   属性:姓名,年龄，薪水
   行为:吃饭,睡觉，管理

3.3.2 案例图解分析

老师类，学生类，还有班主任类，实际上都是属于人类的，我们可以定义一个人类，把他们相同的属性和行为都定义在人类中，然后继承人类即可，子类特有的属性和行为就定义在子类中了。

如下图所示。



3.3.3 案例代码实现

1.父类Human类

```
public class Human {
    // 合理隐藏
```

```
    private String name ;
    private int age ;

    // 合理暴露
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
 }
```

2.子类Teacher类

```
public class Teacher extends Human {
    // 工资
    private double salary ;

    // 特有方法
    public void teach(){
        System.out.println("老师在认真教技术！");
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }
}
```

3.子类Student类

```
public class Student extends Human{

}
```

4.子类BanZhuren类

```
public class Teacher extends Human {
    // 工资
    private double salary ;

       // 特有方法
    public void admin(){
        System.out.println("班主任强调纪律问题！");
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }
}
```

5.测试类

```
  public class Test {
      public static void main(String[] args) {
          Teacher dlei = new Teacher();
          dlei.setName("播仔");
          dlei.setAge("31");
          dlei.setSalary(1000.99);
          System.out.println(dlei.getName());
          System.out.println(dlei.getAge());
          System.out.println(dlei.getSalary());
          dlei.teach();

          BanZhuRen linTao = new BanZhuRen();
          linTao.setName("灵涛");
```

```
14            linTao.setAge("28");
15            linTao.setSalary(1000.99);
16            System.out.println(linTao.getName());
17            System.out.println(linTao.getAge());
18            System.out.println(linTao.getSalary());
19            linTao.admin();
20
21            Student xugan = new Student();
22            xugan.setName("播仔");
23            xugan.setAge("31");
24            //xugan.setSalary(1000.99); // xugan没有薪水属性，报错!
25            System.out.println(xugan.getName());
26            System.out.println(xugan.getAge());
27
28
29
30        }
31    }
```

3.3.4 小结

1.继承实际上是子类相同的属性和行为可以定义在父类中，子类特有的属性和行为由自己定义，这样就实现了相同属性和行为的重复利用，从而提高了代码复用。

2.子类继承父类，就可以直接得到父类的成员变量和方法。是否可以继承所有成分呢？请看下节！

3.4 子类不能继承的内容

3.4.1 引入

并不是父类的所有内容都可以给子类继承的：

子类不能继承父类的构造方法。

值得注意的是子类可以继承父类的私有成员（成员变量，方法），只是子类无法直接访问而已，可以通过**getter/setter**方法访问父类的**private**成员变量。

3.4.1 演示代码

```
public class Demo03 {
    public static void main(String[] args) {
        Zi z = new Zi();
        System.out.println(z.num1);
//      System.out.println(z.num2); // 私有的子类无法使用
        // 通过getter/setter方法访问父类的private成员变量
        System.out.println(z.getNum2());

        z.show1();
        // z.show2(); // 私有的子类无法使用
    }
}

class Fu {
    public int num1 = 10;
    private int num2 = 20;

    public void show1() {
        System.out.println("show1");
    }

    private void show2() {
        System.out.println("show2");
    }

    public int getNum2() {
        return num2;
    }

    public void setNum2(int num2) {
        this.num2 = num2;
    }
}

class Zi extends Fu {
}
```

3.5 继承后的特点—成员变量

当类之间产生了继承关系后，其中各类中的成员变量，又产生了哪些影响呢？

3.5.1 成员变量不重名

如果子类父类中出现**不重名**的成员变量，这时的访问是**没有影响的**。代码如下：

```
class Fu {
    // Fu中的成员变量
    int num = 5;
}
class Zi extends Fu {
    // Zi中的成员变量
    int num2 = 6;

    // Zi中的成员方法
    public void show() {
        // 访问父类中的num
        System.out.println("Fu num="+num); // 继承而来，所以直接访问。
        // 访问子类中的num2
        System.out.println("Zi num2="+num2);
    }
}
class Demo04 {
    public static void main(String[] args) {
        // 创建子类对象
        Zi z = new Zi();
        // 调用子类中的show方法
        z.show();
    }
}

演示结果：
Fu num = 5
Zi num2 = 6
```

3.5.2 成员变量重名

如果子类父类中出现**重名**的成员变量，这时的访问是**有影响的**。代码如下：

```
class Fu1 {
    // Fu中的成员变量。
    int num = 5;
}
class Zi1 extends Fu1 {
    // Zi中的成员变量
    int num = 6;

    public void show() {
        // 访问父类中的num
        System.out.println("Fu num=" + num);
        // 访问子类中的num
        System.out.println("Zi num=" + num);
    }
}
class Demo04 {
    public static void main(String[] args) {
        // 创建子类对象
        Zi1 z = new Zi1();
        // 调用子类中的show方法
        z1.show();
    }
}
演示结果：
Fu num = 6
Zi num = 6
```

子父类中出现了同名的成员变量时，子类会优先访问自己对象中的成员变量。如果此时想访问父类成员变量如何解决呢？我们可以使用super关键字。

3.5.3 super访问父类成员变量

子父类中出现了同名的成员变量时，在子类中需要访问父类中非私有成员变量时，需要使用`super`关键字，修饰父类成员变量，类似于之前学过的`this`。

需要注意的是：**super代表的是父类对象的引用，this代表的是当前对象的引用。**

使用格式：

```
super.父类成员变量名
```

子类方法需要修改，代码如下：

```
class Fu {
    // Fu中的成员变量。
    int num = 5;
}

class Zi extends Fu {
    // Zi中的成员变量
    int num = 6;

    public void show() {
        int num = 1;

        // 访问方法中的num
        System.out.println("method num=" + num);
        // 访问子类中的num
        System.out.println("Zi num=" + this.num);
        // 访问父类中的num
        System.out.println("Fu num=" + super.num);
    }
}

class Demo04 {
    public static void main(String[] args) {
        // 创建子类对象
        Zi1 z = new Zi1();
        // 调用子类中的show方法
        z1.show();
    }
}

演示结果：
method num=1
Zi num=6
Fu num=5
```

> *小贴士：Fu 类中的成员变量是非私有的，子类中可以直接访问。若Fu 类中的成员变量私有了，子类是不能直接访问的。通常编码时，我们遵循封装的原则，使用 private修饰成员变量，那么如何访问父类的私有成员变量呢？对！可以在父类中提供公共的getXxx方法和setXxx方法。*

3.6 继承后的特点—成员方法

当类之间产生了关系，其中各类中的成员方法，又产生了哪些影响呢？

3.6.1 成员方法不重名

如果子类父类中出现**不重名**的成员方法，这时的调用是**没有影响的**。对象调用方法时，会先在子类中查找有没有对应的方法，若子类中存在就会执行子类中的方法，若子类中不存在就会执行父类中相应的方法。代码如下：

```
class Fu {
    public void show() {
        System.out.println("Fu类中的show方法执行");
    }
}
class Zi extends Fu {
    public void show2() {
        System.out.println("Zi类中的show2方法执行");
    }
}
public  class Demo05 {
    public static void main(String[] args) {
        Zi z = new Zi();
        //子类中没有show方法，但是可以找到父类方法去执行
        z.show();
        z.show2();
    }
}
```

3.6.2 成员方法重名

如果子类父类中出现**重名**的成员方法，则创建子类对象调用该方法的时候，子类对象会优先调用自己的方法。

代码如下：

```java
class Fu {
    public void show() {
        System.out.println("Fu show");
    }
}
class Zi extends Fu {
    //子类重写了父类的show方法
    public void show() {
        System.out.println("Zi show");
    }
}
public class ExtendsDemo05{
    public static void main(String[] args) {
        Zi z = new Zi();
        // 子类中有show方法，只执行重写后的show方法
        z.show();  // Zi show
    }
}
```

3.7 方法重写

3.7.1 概念

方法重写：子类中出现与父类一模一样的方法时（返回值类型，方法名和参数列表都相同），会出现覆盖效果，也称为重写或者复写。**声明不变，重新实现**。

3.7.2 使用场景与案例

发生在子父类之间的关系。
子类继承了父类的方法，但是子类觉得父类的这方法不足以满足自己的需求，子类重新写了一个与父类同名的方法，以便覆盖父类的该方 法。

例如：我们定义了一个动物类代码如下：

```java
public class Animal  {
    public void run(){
        System.out.println("动物跑的很快！");
    }
    public void cry(){
        System.out.println("动物都可以叫~~~");
    }
}
```

然后定义一个猫类，猫可能认为父类cry()方法不能满足自己的需求

代码如下：

```java
public class Cat extends Animal {
    public void cry(){
        System.out.println("我们一起学猫叫，喵喵喵！喵的非常好听！");
    }
}

public class Test {
    public static void main(String[] args) {
        // 创建子类对象
        Cat ddm = new Cat();
        // 调用父类继承而来的方法
        ddm.run();
        // 调用子类重写的方法
        ddm.cry();
    }
}
```

3.7.2 @Override重写注解

- @Override:注解，重写注解校验！
- 这个注解标记的方法，就说明这个方法必须是重写父类的方法，否则编译阶段报错。
- 建议重写都加上这个注解，一方面可以提高代码的可读性，一方面可以防止重写出错！

  加上后的子类代码形式如下：

```java
public class Cat extends Animal {
     // 声明不变，重新实现
    // 方法名称与父类全部一样，只是方法体中的功能重写写了！
    @Override
    public void cry(){
        System.out.println("我们一起学猫叫，喵喵喵！喵的非常好听！");
    }
}
```

3.7.3 注意事项

1. 方法重写是发生在子父类之间的关系。
2. 子类方法覆盖父类方法，必须要保证权限大于等于父类权限。
3. 子类方法覆盖父类方法，返回值类型、函数名和参数列表都要一模一样。

3.8 继承后的特点—构造方法

3.8.1 引入

当类之间产生了关系，其中各类中的构造方法，又产生了哪些影响呢？
首先我们要回忆两个事情，构造方法的定义格式和作用。

1. 构造方法的名字是与类名一致的。所以子类是无法继承父类构造方法的。
2. 构造方法的作用是初始化对象成员变量数据的。所以子类的初始化过程中，必须先执行父类的初始化动作。子类的构造方法中默认有一个`super()`，表示调用父类的构造方法，父类成员变量初始化后，才可以给子类使用。（**先有爸爸，才能有儿子**）

继承后子类构方法器特点:子类所有构造方法的第一行都会默认先调用父类的无参构造方法

3.8.2 案例演示

按如下需求定义类:

1. 人类
   成员变量: 姓名,年龄
   成员方法: 吃饭

2. 学生类
   成员变量: 姓名,年龄,成绩
   成员方法: 吃饭

代码如下：

```
class Person {
    private String name;
    private int age;

    public Person() {
        System.out.println("父类无参");
    }

    // getter/setter省略
}

class Student extends Person {
    private double score;

    public Student() {
        //super(); // 调用父类无参,默认就存在，可以不写，必须再第一行
        System.out.println("子类无参");
    }

     public Student(double score) {
        //super();  // 调用父类无参,默认就存在，可以不写，必须再第一行
        this.score = score;
        System.out.println("子类有参");
     }

}

public class Demo07 {
    public static void main(String[] args) {
        Student s1 = new Student();
        System.out.println("----------");
        Student s2 = new Student(99.9);
    }
}

输出结果：
```

```
父类无参
子类无参
----------
父类无参
子类有参
```

3.8.3 小结

- 子类构造方法执行的时候，都会在第一行默认先调用父类无参数构造方法一次。
- 子类构造方法的第一行都隐含了一个**super()**去调用父类无参数构造方法，**super()**可以省略不写。

3.9 super(...)和this(...)

3.9.1 引入

请看上节中的如下案例：

```
class Person {
    private String name;
    private int age;

    public Person() {
        System.out.println("父类无参");
    }

    // getter/setter省略
}

class Student extends Person {
    private double score;

    public Student() {
        //super(); // 调用父类无参构造方法,默认就存在，可以不写，必须再第一行
        System.out.println("子类无参");
    }

     public Student(double score) {
```

```
        //super();  // 调用父类无参构造方法,默认就存在，可以不写，必须再第一行
        this.score = score;
        System.out.println("子类有参");
    }
     // getter/setter省略
}

public class Demo07 {
    public static void main(String[] args) {
        // 调用子类有参数构造方法
        Student s2 = new Student(99.9);
        System.out.println(s2.getScore()); // 99.9
        System.out.println(s2.getName()); // 输出 null
        System.out.println(s2.getAge()); // 输出 0
    }
}
```

我们发现，子类有参数构造方法只是初始化了自己对象中的成员变量score，而父类中的成员变量name和age依然是没有数据的，怎么解决这个问题呢，我们可以借助与super(...)去调用父类构造方法，以便初始化继承自父类对象的name和age.

3.9.2 super和this的用法格式

super和this完整的用法如下，其中this，super访问成员我们已经接触过了。

```
this.成员变量        --    本类的
super.成员变量       --    父类的

this.成员方法名()     --    本类的
super.成员方法名()    --    父类的
```

接下来我们使用调用构造方法格式：

```
super(...) -- 调用父类的构造方法，根据参数匹配确认
this(...) -- 调用本类的其他构造方法，根据参数匹配确认
```

3.9.3 super(....)用法演示

代码如下：

```
class Person {
    private String name ="凤姐";
    private int age = 20;

    public Person() {
        System.out.println("父类无参");
    }

    public Person(String name , int age){
        this.name = name ;
        this.age = age ;
    }

    // getter/setter省略
}

class Student extends Person {
    private double score = 100;

    public Student() {
        //super(); // 调用父类无参构造方法,默认就存在，可以不写，必须再第一行
        System.out.println("子类无参");
    }

     public Student(String name ,  int age, double score) {
        super(name ,age);// 调用父类有参构造方法Person(String name , int age)初始化name和age
        this.score = score;
        System.out.println("子类有参");
     }
      // getter/setter省略
}

public class Demo07 {
    public static void main(String[] args) {
        // 调用子类有参数构造方法
        Student s2 = new Student("张三", 20, 99);
```

```
37          System.out.println(s2.getScore()); // 99
38          System.out.println(s2.getName()); // 输出 张三
39          System.out.println(s2.getAge()); // 输出 20
40      }
41  }
```

注意：

子类的每个构造方法中均有默认的**super()**，调用父类的空参构造。手动调用父类构造会覆盖默认的**super()**。

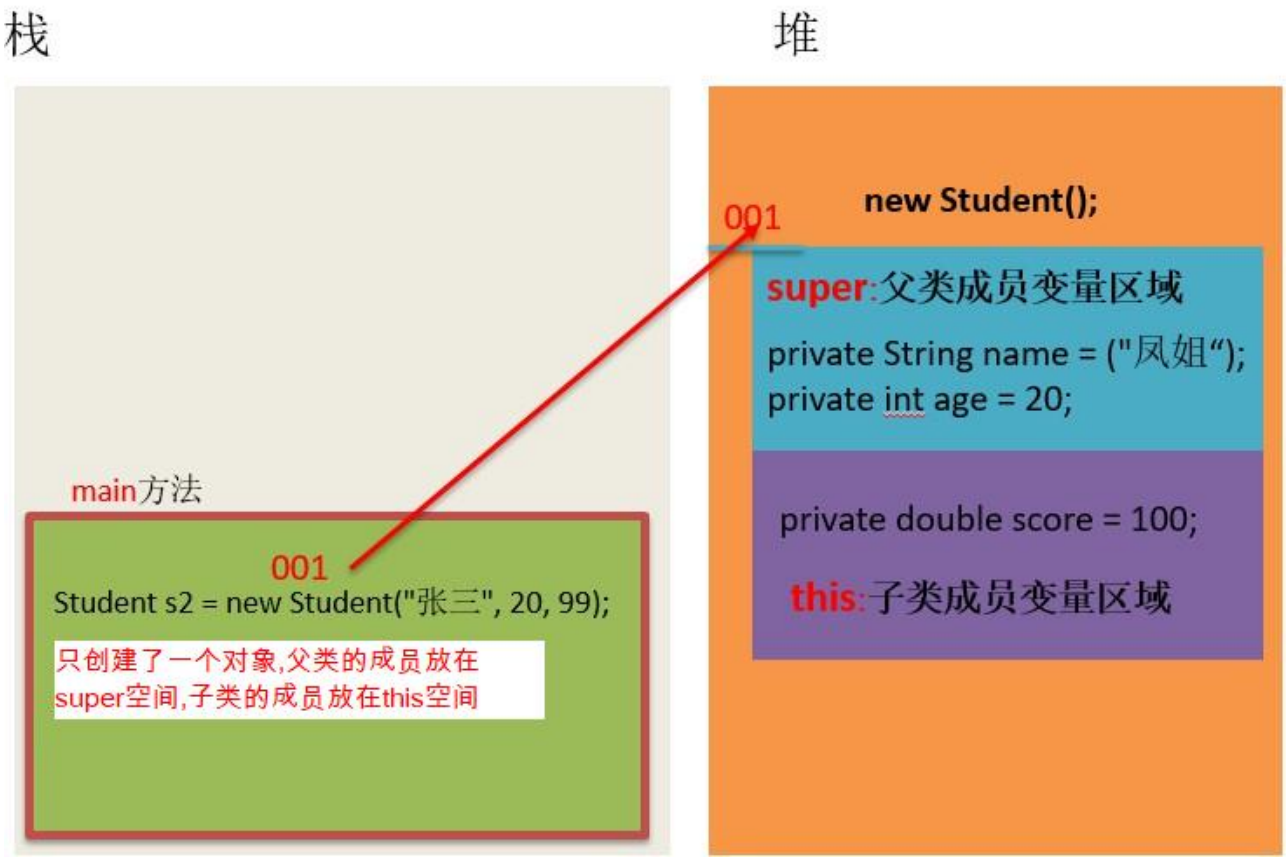
super() 和 **this()** 都必须是在构造方法的第一行，所以不能同时出现。

super(..)是根据参数去确定调用父类哪个构造方法的。

3.9.4 super(...)案例图解

父类空间优先于子类对象产生

在每次创建子类对象时，先初始化父类空间，再创建其子类对象本身。目的在于子类对象中包含了其对应的父类空间，便可以包含其父类的成员，如果父类成员非private修饰，则子类可以随意使用父类成员。代码体现在子类的构造七调用时，一定先调用父类的构造方法。理解图解如下：



3.9.5 this(...)用法演示

this(...)

- 默认是去找本类中的其他构造方法，根据参数来确定具体调用哪一个构造方法。
- 为了借用其他构造方法的功能。

```
package com.itheima._08this和super调用构造方法;
/**
 * this(...):
 *     默认是去找本类中的其他构造方法，根据参数来确定具体调用哪一个构造方法。
 *     为了借用其他构造方法的功能。
 *
 */
public class ThisDemo01 {
    public static void main(String[] args) {
        Student xuGan = new Student();
        System.out.println(xuGan.getName()); // 输出:徐干
        System.out.println(xuGan.getAge());// 输出:21
        System.out.println(xuGan.getSex());// 输出:  男
    }
}

class Student{
    private String name ;
    private int age ;
    private char sex ;

    public Student() {
  // 很弱，我的兄弟很牛逼啊，我可以调用其他构造方法：Student(String name, int age, char sex)
        this("徐干",21,'男');
    }

    public Student(String name, int age, char sex) {
        this.name = name ;
        this.age = age   ;
        this.sex = sex   ;
    }
```

```
32
33      public String getName() {
34          return name;
35      }
36
37      public void setName(String name) {
38          this.name = name;
39      }
40
41      public int getAge() {
42          return age;
43      }
44
45      public void setAge(int age) {
46          this.age = age;
47      }
48
49      public char getSex() {
50          return sex;
51      }
52
53      public void setSex(char sex) {
54          this.sex = sex;
55      }
56  }
```

3.9.6 小结

- 子类的每个构造方法中均有默认的**super()**，调用父类的空参构造。手动调用父类构造会覆盖默认的**super()**。
- **super()** 和 **this()** 都必须是在构造方法的第一行，所以不能同时出现。
- **super(..)**和**this(...)**是根据参数去确定调用父类哪个构造方法的。
- super(..)可以调用父类构造方法初始化继承自父类的成员变量的数据。
- this(..)可以调用本类中的其他构造方法。

3.10 继承的特点

1. Java只支持单继承，不支持多继承。

```
// 一个类只能有一个父类，不可以有多个父类。
class A {}
class B {}
class C1 extends A {} // ok
// class C2 extends A, B {} // error
```

2. 一个类可以有多个子类。

```
// A可以有多个子类
class A {}
class C1 extends A {}
class C2 extends  A {}
```

3. 可以多层继承。

```
class A {}
class C1 extends A {}
class D extends C1 {}
```

> *顶层父类是Object类。所有的类默认继承Object，作为父类。*

4. 关于今天知识的小结：

会写一个继承结构下的标准Javabean即可

需求：

猫：属性，姓名，年龄，颜色

狗：属性，姓名，年龄，颜色，吼叫

分享书写技巧：

1.在大脑中要区分谁是父，谁是子

2.把共性写到父类中，独有的东西写在子类中

3.开始编写标准Javabean（从上往下写）

4.在测试类中，创建对象并赋值调用

代码示例：

```
package com.itheima.test4;

public class Animal {
    //姓名，年龄，颜色
    private String name;
    private int age;
    private String color;


    public Animal() {
    }

    public Animal(String name, int age, String color) {
        this.name = name;
        this.age = age;
        this.color = color;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

```

```
35      public String getColor() {
36          return color;
37      }
38
39      public void setColor(String color) {
40          this.color = color;
41      }
42  }
43
44
45  public class Cat extends Animal{
46      //因为猫类中没有独有的属性。
47      //所以此时不需要写私有的成员变量
48
49      //空参
50      public Cat() {
51      }
52
53      //需要带子类和父类中所有的属性
54      public Cat(String name, int age, String color) {
55          super(name,age,color);
56      }
57  }
58
59
60  public class Dog extends Animal{
61      //Dog : 吼叫
62      private String wang;
63
64      //构造
65      public Dog() {
66      }
67
68      //带参构造：带子类加父类所有的属性
69      public Dog(String name, int age, String color,String wang)
    {
70          //共性的属性交给父类赋值
71          super(name,age,color);
72          //独有的属性自己赋值
73          this.wang = wang;
74      }
75
```

```
76      public String getWang() {
77          return wang;
78      }
79
80      public void setWang(String wang) {
81          this.wang = wang;
82      }
83  }
84
85  public class Demo {
86      public static void main(String[] args) {
87          //Animal ：  姓名，年龄，颜色
88          //Cat ：
89          //Dog ：吼叫
90
91          //创建狗的对象
92          Dog d = new Dog("旺财",2,"黑色","嗷呜~~");
93          System.out.println(d.getName()+", " + d.getAge() + ", "
    + d.getColor() + ", " + d.getWang());
94
95          //创建猫的对象
96          Cat c = new Cat("中华田园猫",3,"黄色");
97          System.out.println(c.getName() + ", " + c.getAge() + ",
    " + c.getColor());
98      }
99  }
100
101
```