

# 第五章 面向对象基础

## 1 面向对象

OOP，面向对象编程

OOP (object oriented programming) ， 面向对象编程

- 是一种以对象为中心的编程思想，通过借助对象实现具体的功能
- 将大问题拆分成小问题，然后借助不同对象分别解决，最终实现功能

POP (procedure oriented Programming) ， 面向过程编程

- 是一种以过程为中心的编程思想，靠自己一步一步去实现功能，需要对每个步骤精确控制
- 强调按步骤实现功能，先分析解决问题所需步骤，再自定义方法实现每个步骤功能，然后依次调用方法，最终实现功能

### 生活案例：

小明早上起床，想要吃鸡蛋饼。分别用两种不同思想去解决该问题，具体如下：

- 面向过程

小明先准备面、鸡蛋等原材料，再和面，然后开火烙饼，最后成功吃到鸡蛋饼

- 面向对象

小明到街上找到摊煎饼的大娘，让大娘做个鸡蛋饼，最后吃到了鸡蛋饼

### 面向对象特点：

- 更符合人类思想习惯的思想
- 利用对象去实现功能
- 将复杂事情简单化

针对要解决问题的用户而言，可以把大问题拆解成小问题，分别指挥不同的对象去解决小问题

- 程序员的角色由执行者变成了指挥者

大问题：

造出一辆新能源汽车，可以自动驾驶

拆解成多个小问题：

合作伙伴1：专门研发 电池

合作伙伴2：专门研发 底盘

合作伙伴3：专门研发 电池管理系统

合作伙伴4：专门研发 自动驾驶系统

...

## 面向对象开发：

- 就是不断创建对象，使用对象，指挥对象做事情实现功能
- 原则：如果有对象，就指挥对象实现功能；如果没有，就创建对象，然后再指挥

## 面向对象语言特征：

- 封装（encapsulation） **信息隐蔽**
- 继承（inheritance） **代码重用**

- 多态 (polymorphism) **灵活、接口统一**

Java语言、C++、Python等都是面向对象程序设计语言中的一种，所以都具有这三种特征

后面的课程中会对每个特征进行详细的学习

## 2 对象理解

自然界中客观存在的事物皆为对象，万物皆对象

放眼望去，我们身边的能看到的東西，都是对象，例如桌子、话筒、笔记本电脑、电脑键盘、你、你的鞋子、眼睛等等。

观察下图，列出图上存在的对象：



## 理解对象

- 任何事物都是一个对象 (object)
- 对象由对象组成
- 每个对象都有**属性（静态的描述信息）**、**行为（动态的功能描述）**
- 具有相似属性和行为的对象可以**归为一类**

## 3 类的定义

具有相同属性和行为的对象可以抽象为类（数据类型的一种）

### 类的组成：

- 属性：指事物的特征，静态描述，例如：手机有品牌，价格，尺寸
- 行为：指事物所具有的功能，动态描述，例如：手机可以打电话，也可以发短信

### 类的理解：

- 类是对现实生活中一类具有共同属性和行为的事物的抽象
- 类是对象的数据类型，类是具有相同属性和行为的一组对象的集合
- 简单理解：类就是对现实事物的一种描述
- 类是引用数据类型中的一种

现实生活中，一切皆为对象，其中有些对象的属性和行为是类似的，比如大家桌子上放的那些对象，它们的属性类似，都包含屏幕、键盘、触摸板、内存条等，也具有相似的功能，能够开机、安装运行软件、写代码、看视频等，这些具有相同属性和行为的对象，我们可以抽象为笔记本类。

## 结论：类是对象的抽象，对象是类的实例

### 类定义格式：

```
[public] class 类名 {  
    //属性，可以包含多个  
    [权限修饰符] 数据类型 成员变量名；  
  
    //行为，可以包含多个  
    [权限修饰符] 返回值类型 成员方法名(形参列表) {  
        具体功能实现  
    }  
  
    //构造器，可有可无，暂时不用写，后续章节讨论  
    [权限修饰符] 类名(形参列表) {  
        初始化语句  
    }  
}
```

### 类定义步骤：

- 定义类
- 编写类的成员变量
- 编写类的成员方法

### 具体案例：

```
package com.briup.chap05.test;  
  
public class Student {  
    // 属性：姓名，年龄
```

// 成员变量：跟之前定义变量的格式一样，只不过位置发生了改变，放到了类中的方法之外

```
String name;
```

```
int age;
```

// 行为 ：打招呼

// 成员方法：跟之前定义方法的格式一样，只不过去掉了static关键字

```
public void sayHello() {  
    System.out.println("hello");  
}  
}
```

上述案例中，我们定义了学生类Student;

注意，Student是一种数据类型，是自定义类类型，属于引用类型;

类类型使用时跟int、double、String等类型类似，都必须先定义变量并赋值，才能使用;

类类型和基本数据类型定义变量的方式不同，描述方式也不同，具体如下：

//基本数据类型定义变量

```
int a = 10;           //定义一个int类型变量a
```

```
double d = 2.3;       //定义一个double类型变量d
```

//类类型定义变量

```
Student s1 = new Student(); //实例化一个Student类对象s1
```

```
Student s2 = new Student(); //实例化一个Student类对象s2
```

具体的对象创建，以及对象属性、方法的调用，我们看下一个章节具体描述。

---

## 4 对象使用

---

创建对象格式：

```
类名 对象名 = new 类名();
```

## 调用成员格式:

对象名.成员变量

对象名.成员方法();

## 案例演示:

```
package com.briup.chap05.test;

public class Test04_Student {
    /*
        创建对象的格式:
            类名 对象名 = new 类名();
        调用成员变量的格式:
            对象名.变量名
        调用成员方法的格式:
            对象名.方法名();
    */
    public static void main(String[] args) {
        // 实例化对象
        // 固定格式: 类名 对象名 = new 类名();
        Student stu = new Student();

        // 输出对象属性的默认值
        // 对象属性访问格式: 对象名.变量名
        System.out.println(stu.name); // null
        System.out.println(stu.age);  // 0

        // 给对象属性赋值
        stu.name = "张三";
        stu.age = 23;

        // 再次输出对象属性值
        System.out.println(stu.name); // 张三
        System.out.println(stu.age);  // 23
    }
}
```

```
// 对象成员方法调用
// 固定格式：对象名.方法名(实际参数列表);
stu.sayHello();

// 输出结果：全类名@对象内存地址
// 全类名：包名.类名
System.out.println(stu);
//com.briup.chap05.test.Student@7852e922
    }
}
```

### 类使用总结：

- 一般Java程序会写两个类：基础类，测试类
- 基础类就是我们要实现封装出来的那个类
- 测试类就是包含main方法的类
- 注意：只能在一个类中定义main方法，其是程序的入口，必须唯一

### 数据类型理解：

Java中对数据类型的描述和定义，都是抽象的，每一种数据类型，都是对一类数据的抽象描述，描述这种数据的基本特点。

比如 `int` 类型（系统提供），是对计算机中一个32位数据的定义，描述这种数据的基本特点和表达的含义。又比如 `String` 类型（系统提供），是对程序中的字符串这种数据的定义，描述了作为一个字符串数据，应该具有哪些属性、行为和特点。

刚才操作的 `Student` 类型（程序员自定义），对学生这类数据进行描述，学生具有 `name` 和 `age` 静态特征，也具有 `study` 这样的动态功能。



`int`、`String` 和 `Student` 都是对数据的抽象描述，不能当做具体的数据使用，如果想使用的这些数据的话，必须使用 `int` 或 `String` 类型定义变量，使用 `Student` 实例化对象，然后使用变量或对象来参与运算。

## 类和对象的关系：

通过前面的学习可知，类是一组相关属性和行为的集合，它是对某一种具体事物的抽象描述。

也可以把类看作一个模板，我们使用的对象，就是按照这个模板中的定义，来进行创建的。

- 类是对一类事物的描述，是抽象的
- 对象是一类事物的实例，是具体的
- 类是对象的模板，对象是类的实体

## 案例理解：

抽象出来猫类：（这就相当于定义了一个类）

- 属性：名字、体重、年龄、颜色
- 行为：走、跑、叫

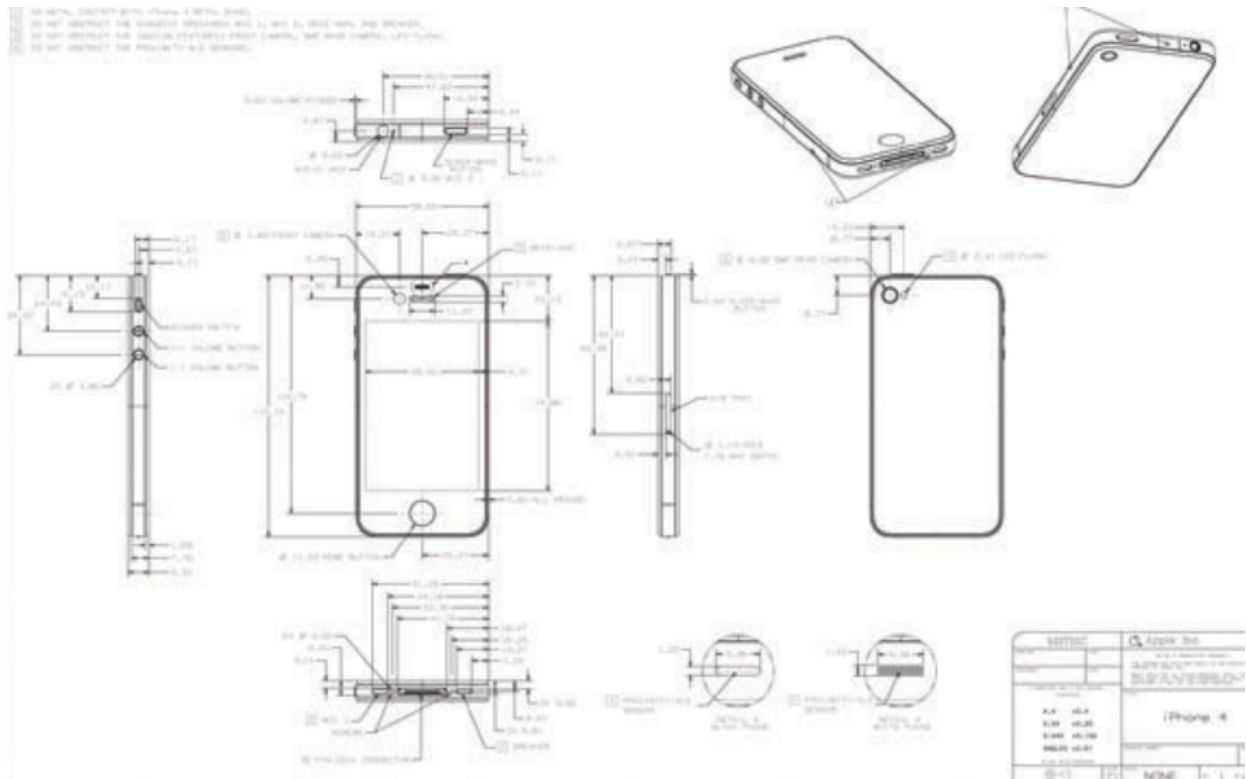
我家的小猫皮皮：（这就是一个具体的对象）

- 名字：皮皮，体重：3KG，年龄：3岁，颜色：白色
- 皮皮可以走，也可以跑，也可以喵喵的叫

可以看出，皮皮这个小猫（对象），完全符合上面对猫的定义的一切属性和行为。

## 案例理解：

下图是手机图纸，定义了手机一些特征（相当于定义了类）



按着上面的图纸，制造出来的两台具体的手机（相当于创建了手机类的两个不同的对象）



一个类的对象，也就是这个类的具体实例。如果有需要，可以根据手机类（图纸），创建出无数个具体的对象（手机）。

**结论：类是一种抽象的数据描述，对象是类的一个具体的实例。**

## 案例实现：

定义一个手机类，属性含品牌、价格，具备打电话、发短信的功能。

再定义一个手机测试类，在手机测试类中通过对象完成成员变量和成员方法的使用。

## 基础类定义：

```
package com.briup.chap05.test;

public class Phone {
    // 品牌，价格
    String brand;
    int price;

    // 打电话，发短信
    public void call(){
        System.out.println("打电话");
    }

    public void sendMessage(String name, String msg){
        System.out.println(" 给 " + name + " 发短信: " + msg );
    }
}
```

## 测试类定义：

```
package com.briup.chap05.test;

public class Test04_Phone {
    public static void main(String[] args) {
        // 1. 创建对象
        Phone p = new Phone();

        // 2. 给成员变量进行赋值
        p.brand = "小米";
    }
}
```

```

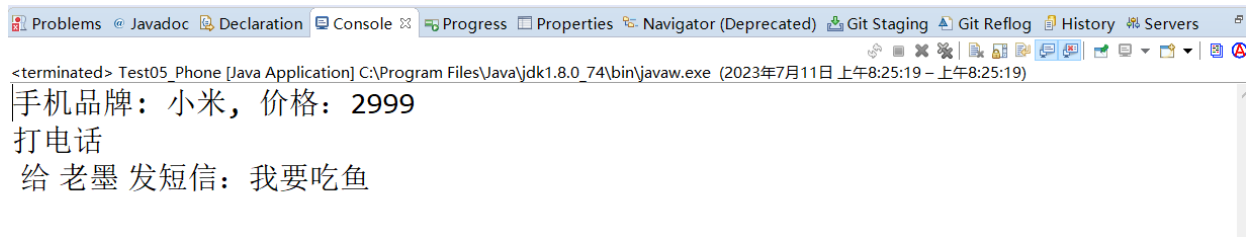
        p.price = 2999;

        // 3. 打印赋值后的成员变量
        System.out.println("手机品牌: " + p.brand + ", 价格: " +
p.price);

        // 4. 调用成员方法
        p.call();
        p.sendMessage("老墨", "我要吃鱼");
    }
}

```

运行效果:



```

<terminated> Test05_Phone [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年7月11日 上午8:25:19 - 上午8:25:19)
手机品牌: 小米, 价格: 2999
打电话
给 老墨 发短信: 我要吃鱼

```

## 5 对象内存

观察下面案例, 理解stu对象的内存构成

```

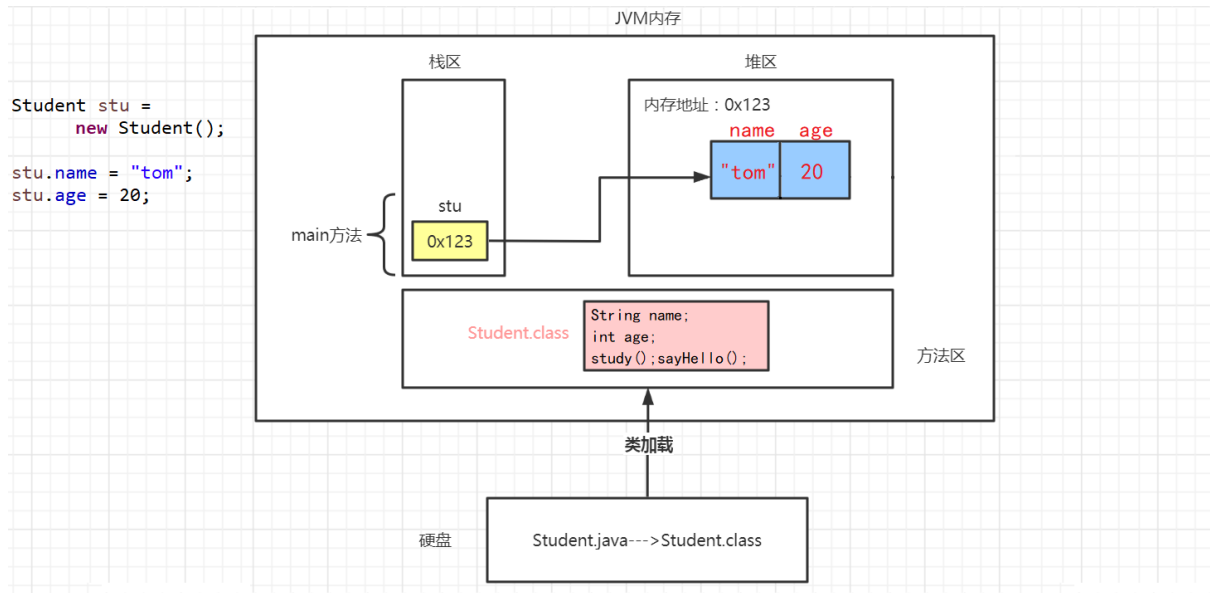
package com.briup.chap05.test;

public class Test05_Memory {
    public static void main(String[] args) {
        //实例化一个对象
        Student stu = new Student();

        stu.name = "tom";
        stu.sayHello();
    }
}

```

## 单个对象内存图：



## 程序运行过程：

1. 加载Student类：把Student.class文件内容加载到方法区中
2. 加载main方法并运行：整个main方法的代码都被加载到栈区中
3. 创建引用类型变量：在栈空间中开辟一块内存空间，用stu标识
4. 在堆中开辟内存创建对象，并给属性赋上默认初始值
5. 将堆空间对象内存地址值放入stu标识的内存区域中
6. 对象属性赋值：将"tom"和20放入堆空间对象内存区域内
7. 对象方法调用：找到方法区sayHello方法对应的代码，执行
8. main方法继续执行，遇到 `}`，程序执行结束

### 引用类型理解：

上述案例中，stu是一个引用类型变量，其对应栈区的一块内存区域，其中放的是一个引用值（地址值），通过这个引用值，系统可以找到对象实际开辟的内存空间（堆区），进而进行对象属性操作或方法调用。

思考：如果创建出多个对象，其内存构成是什么样子的？

## 多个对象案例：

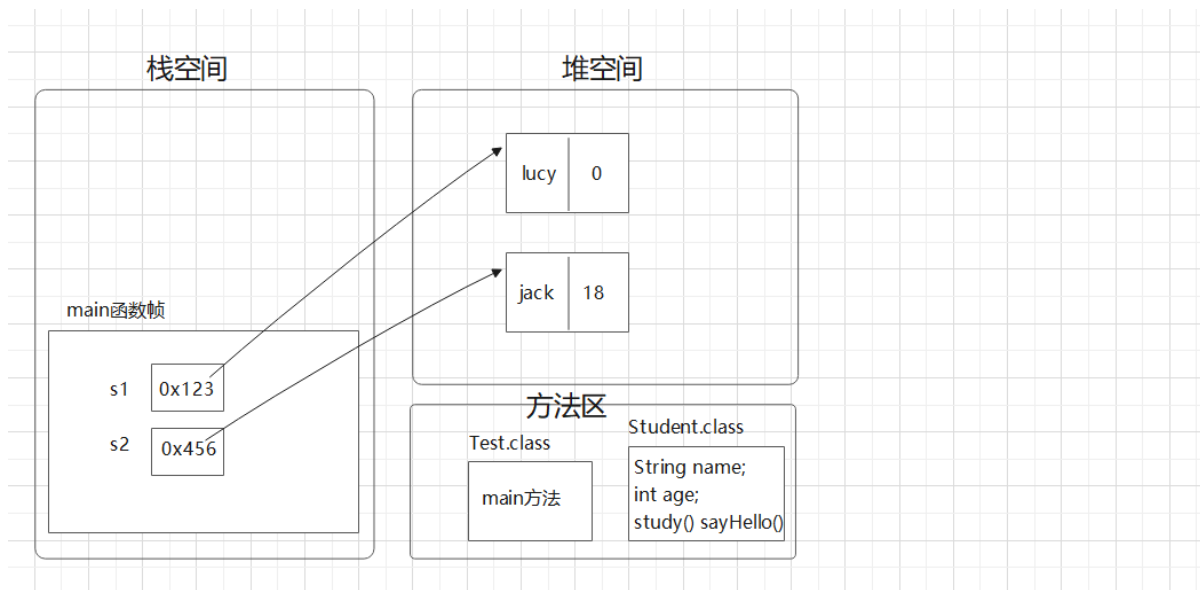
```
package com.briup.chap05.test;

//创建多个对象
public class Test05_TwoObject {
    public static void main(String[] args) {
        //创建对象s1
        Student s1 = new Student();
        //给对象name属性赋值
        s1.name = "lucy";

        Student s2 = new Student();
        s2.name = "jack";
        s2.age = 20;

        s1.sayHello();
        s2.study();
    }
}
```

## 多个对象内存构成：



## 对象内存结论:

系统会为每个**对象**开辟单独的**内存空间（堆空间）**，用来**存储对象的属性**。

类的**成员方法**存储在**方法区**，只保留一份，只要是该类的对象，都可以调用。

## 6 变量对比

到目前为止，我们学习了两种变量：**成员变量**、**局部变量**。

观察下面代码，指出哪些是成员变量，哪些是局部变量。

```
package com.briup.chap05.test;

//导入类
import java.util.Scanner;

public class Test06_Variate {
    int num;

    public void setNum(int n) {
```

```

        num = n;
    }

    public static void main(String[] args) {
        //1.从键盘录入一个整形数
        Scanner sc = new Scanner(System.in);
        System.out.println("input a num: ");
        int number = sc.nextInt();

        //2.实例化对象，然后用上述整形数给其属性赋值
        Test06_Variate t = new Test06_Variate();
        t.setNum(number);

        //3.输出对象属性值
        System.out.println(t.num);
    }
}

```

## 成员、局部变量区别：

- **定义位置不同**

成员变量：类中方法外

局部变量：方法内部或方法声明上（形参列表）

- **内存中位置不同**

成员变量：堆内存

局部变量：栈内存

- **生命周期不同**

成员变量：随着对象的存在而存在，随着对象的消失而消失

局部变量：随着方法的调用而存在，随着方法的调用完毕而消失

- **初始化值不同**

成员变量：有默认初始化值



局部变量：没有默认初始化值，必须先定义，赋值才能使用

## 7 封装特性

---

### 1) 概念理解

封装是面向对象三大特征之一，另外两个是继承，多态

封装是指隐藏对象的属性和实现细节，仅对外提供公共访问方式。

其优点如下：

- 通过方法来控制成员变量的操作，提高了代码的安全性
- 把代码用方法进行封装，提高了代码的复用性
- 隐藏代码实现细节，提供公共访问方式，简化操作

生活案例：

电源插座，如果不对电源插座封装，平时用电将会很不安全，容易触电。

笔记本开关按钮，用户不需要关心开机关机底层那些复杂的实现细节，只需要知道按按钮可以开关机即可。

**封装原则：**

- 把不需要对外提供的内容隐藏起来
- 把属性隐藏，提供公共方法对其访问

## 2) private

private、public都是权限修饰符，可以用来修饰成员变量、成员方法和构造方法；

private表示私有，用它修饰类的成员（含成员变量、成员方法），则这些成员只能在**类内（类的成员函数内部）**去使用，其他地方不可以操作；

public表示公有，用它修饰类的成员（含成员变量、成员方法），则这些成员在类内、类外都可以操作。

## 3) 封装实现

封装步骤：

1. 使用private修饰成员变量
2. 提供对应的setXxx()、getXxx()方法，用public修饰
3. 具体使用时，借助对象的setXxx方法给属性赋值，getXxx方法获取属性值

案例展示：

```
package com.briup.chap05.test;

//银行账户类
class Account {
    //步骤1: private修饰数据成员
    private String id;        //卡号
    private double balance;   //金额

    //步骤2: 定义get|set方法，使用public修饰
    public void setBalance(double money) {
        balance = money;
    }

    public double getBalance() {
```

```

        return balance;
    }

    public void setId(String accountId) {
        id = accountId;
    }

    public String getId() {
        return id;
    }
}

public class Test07_Private {
    public static void main(String[] args) {
        //1.实例化对象，使用系统提供的无参构造器
        Account acc = new Account();

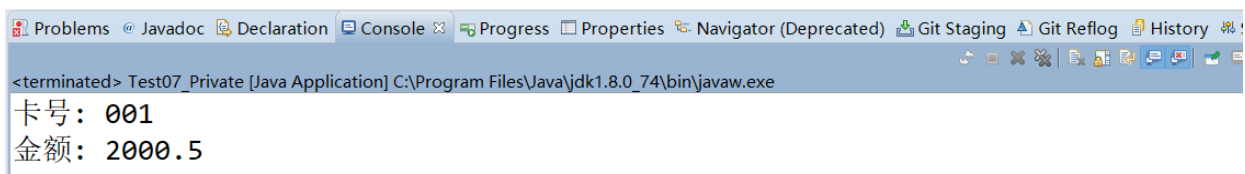
        //2.在类外通过对象访问私有数据成员，编译报错
        //acc.id = "001";
        //acc.balance = 2000.5;

        //步骤3：具体使用时，借助对象的setXxx方法给属性赋值，getXxx方法
        获取属性值
        //3.借助set方法赋值
        acc.setId("001");
        acc.setBalance(2000.5);

        //4.借助get方法获取属性值
        System.out.println("卡号： " + acc.getId());
        System.out.println("金额： " + acc.getBalance());
    }
}

```

运行效果：



```

<terminated> Test07_Private [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe
卡号： 001
金额： 2000.5

```

**思考1：上述set方法中，如果方法的形式参数名跟类数据成员名相同，如何进行赋值？**

```
public void setId(String accountId) {  
    id = accountId;  
}
```

解决方案：借助this解决，下一个章节讨论。

**思考2：属性过多，如何赋初值？**

上述案例中，我们借助set方法给对象的属性赋值，如果对象的属性特别多，有20个，我们难道每次要调用20个set方法赋初值吗，有没有更好的初始化属性值方案？

解决方案：借助构造方法解决，章节9中具体讨论。

## 8 this关键字

在类中的普通成员方法中，可以使用this关键字，其表示调用当前方法的对象引用，即哪个对象调用该方法，this就代表哪一个对象。

**this关键字用法：**

- 对成员变量和局部变量进行区分

固定格式： **this.数据成员**；

- 调用类中的成员方法

固定格式： **this.成员方法(实际参数列表)**；

- 调用类中的其他构造器（后面章节补充）

## 成员变量与局部变量的区分：

方法的形参如果与成员变量同名

- 不带this修饰的变量指的是形参
- 如果要表示成员变量，则必须加this修饰

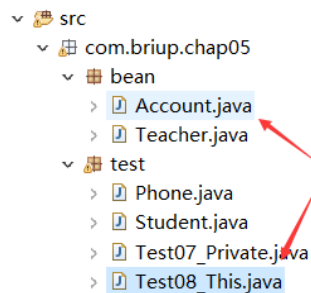
方法的形参与成员变量不同名

- 则不带this修饰的变量指的就是成员变量

## 案例展示：

重新实现Account类，要求使用this关键字，要求成员方法中形式参数名跟类数据成员名相同。

基础类Account：



两个类不在同一个包中

```
package com.briup.chap05.bean;
```

```
public class Account {  
    private String id;        //卡号  
    private double balance; //金额
```

```
    public void setId(String id) {
```

//方法的形参如果与成员变量同名，不带this修饰的变量指的是形参，而不是成员变量

//如果要表示成员变量，则必须加this修饰

```

        System.out.println("参数id: " + id);
        System.out.println("this.id: " + this.id);

        this.id = id;
    }

    public String getId() {
        //如果形参没有与成员变量同名，不带this修饰的变量指的是成员变量
        return id;
        //return this.id;
    }

    public void setBalance(double balance) {
        this.balance = balance;
    }

    public double getBalance() {
        return balance;
    }

    public void show() {
        System.out.println("id: " + this.id);

        //使用this去调用普通成员方法
        System.out.println("balance: " + this.getBalance());
    }
}

```

测试类:

```

package com.briup.chap05.test;

import com.briup.chap05.bean.Account;

public class Test08_This {
    public static void main(String[] args) {
        //1.实例化对象
        Account acc = new Account();
    }
}

```

```

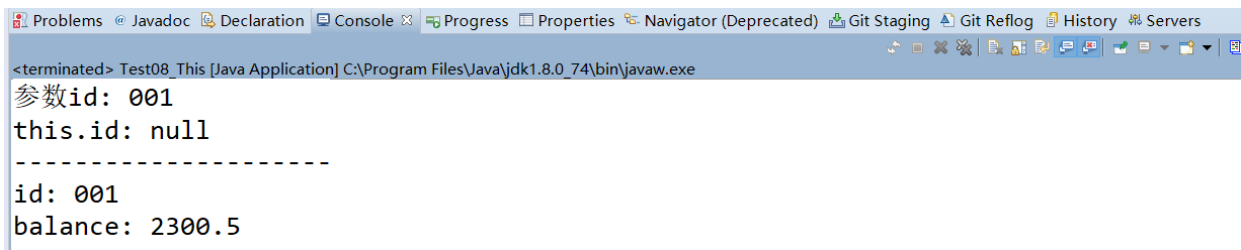
//2.借助set方法赋值
acc.setId("001");
acc.setBalance(2300.5);

System.out.println("-----");

acc.show();
}
}

```

## 运行效果：



```

<terminated> Test08_This [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe
参数id: 001
this.id: null
-----
id: 001
balance: 2300.5

```

测试代码较为简单，注意观察注释描述。

## this内存构成理解：

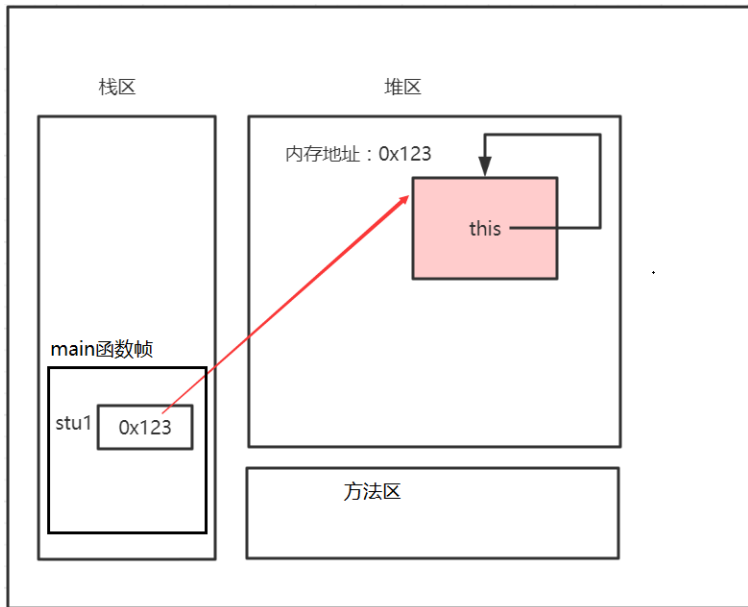
```

public static void main(String[] args) {
    Student stu1 = new Student("zs",21);
    stu1.sayHello();

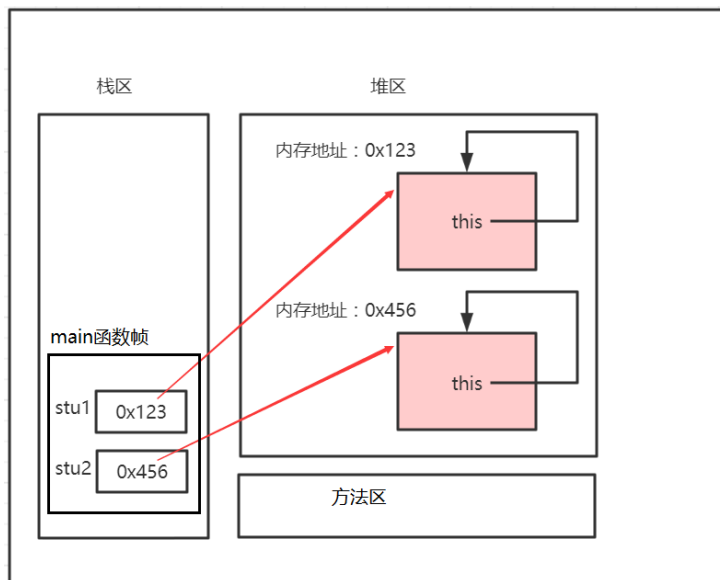
    Student stu2 = new Student("tom",19);
    stu2.sayHello();
}

```

- 单个对象内存图



- 多个对象内存图



观察上图可知：每一个对象中，都有自己的this，和其他对象中的互不影响。

当前执行 `stu1.sayHello()` 代码的时候，`this`代表的就是`stu1`

当前执行 `stu2.sayHello()` 代码的时候，`this`代表的就是`stu2`

**结论：成员方法被哪个对象调用，方法中的this就代表那个对象。即谁调用，this就代表谁。**



思考：在生活中，我们每一个人心中的this，指的是哪一个汉字？

## 9 构造方法

构造方法可以对对象进行初始化操作，即为对象开辟内存空间的时候，给对象的成员成员赋初值。

### 构造方法格式：

```
[修饰符] 类名(参数列表) {  
    初始化语句s;  
}
```

### 注意事项：

- 构造方法一般使用 `public` 修饰
- 构造方法没有返回值类型，连 `void` 都没有
- 构造方法名和类名相同（区分大小写）
- 构造方法可以重载

### 案例演示：

```
package com.briup.chap05.bean;  
  
public class Teacher {  
    private String name;  
    private double salary;  
  
    //无参构造方法
```

```

public Teacher() {
    //构造器中一般只写赋值语句，不放输出语句，此处写出是为了方便学生
    理解构造方法过程
    System.out.println("Teacher() ...");
}

//有参构造方法，注意：构造方法可以重载
public Teacher(String name, double salary) {
    System.out.println("Teacher(String,double) ...");

    //赋值语句
    this.name = name;
    this.salary = salary;
}

public void show() {
    System.out.println("name: " + name);
    System.out.println("salary: " + salary);
}
}

```

## 执行时机：

- 创建对象的时候调用，每创建一次对象，就会执行一次构造方法
- 不能手动调用构造方法

## 功能测试：

```

package com.briup.chap05.test;

import com.briup.chap05.bean.Teacher;

public class Test09_Constructor {
    public static void main(String[] args) {
        //借助无参构造器 实例化对象
    }
}

```

```

Teacher t1 = new Teacher();
t1.show();

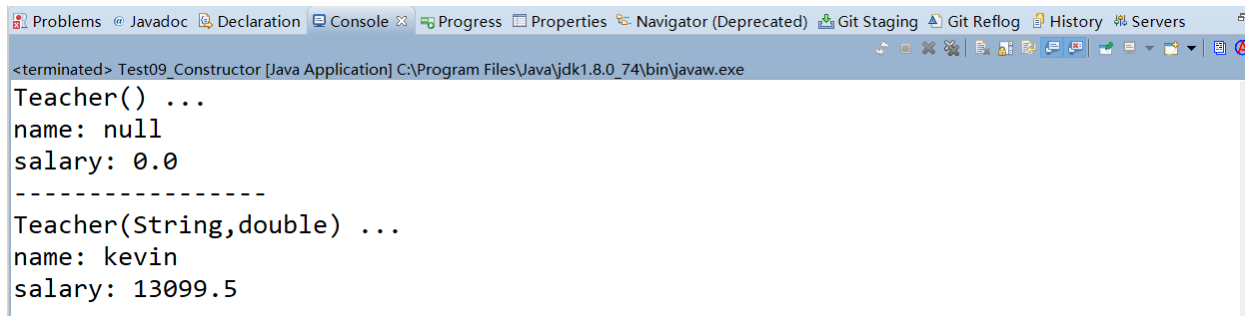
System.out.println("-----");

//借助有参构造器 实例化对象
Teacher t2 = new Teacher("kevin", 13099.5);
t2.show();

//手动调用构造方法，错误用法，无法通过编译
//t2.Teacher();
}
}

```

## 运行效果：



```

<terminated> Test09_Constructor [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe
Teacher() ...
name: null
salary: 0.0
-----
Teacher(String,double) ...
name: kevin
salary: 13099.5

```

观察运行效果可知，创建对象时构造器会被自动调用（实例化对象时指定的那个构造方法），其完成了对象属性的初始化。

## 注意事项补充：

- 用户不定义构造方法，系统会提供一个无参构造方法

```

public 类名() {
    //什么都不干
}

```

- 用户定义构造方法，系统则不再提供无参构造方法

- 用户不需要也不可以主动调用构造方法，系统会自动调用

### 对象创建步骤（理解即可）：

```
public static void main(String[] args) {  
    Student s = new Student("zhang", 23);  
}
```

1. 将Student.class文件加载到内存方法区
2. 在main栈帧开辟一块内存，用s标识
3. 在堆中开辟内存创建对象
4. 给属性以默认初始值（null，0）
5. 属性进行显式初始化（如果存在的话，比如 private int age = 10;）
6. 调用构造方法，用（"zhang", 23）给属性赋值
7. 将堆中对象的内存地址赋值给s，对象创建完成

## 10 this补充

### this特殊用法：

在构造方法中，可以借助this关键字调用其他构造方法

具体格式为：`this(实际参数列表);`

### 案例展示：

### 基础类补充：

```
package com.briup.chap05.bean;
```

```

public class Teacher {
    //...省略

    //该类已经成功定义2参构造器，定义过程省略
    //public Teacher(String name, double salary);

    //this特殊用法：在构造方法中，调用其他构造方法
    public Teacher(String name) {
        //必须是构造方法的第一行有效代码
        this(name, 0);
    }
}

```

测试类代码：

```

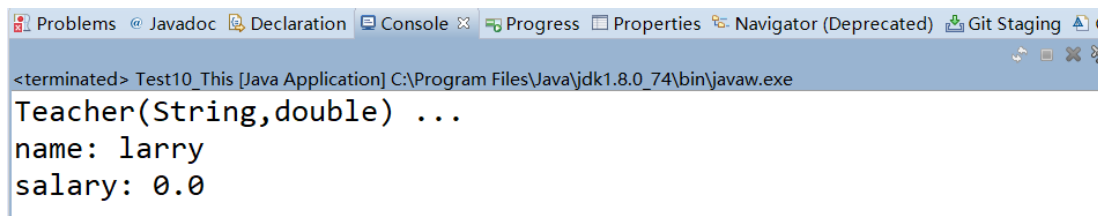
package com.briup.chap05.test;

import com.briup.chap05.bean.Teacher;

public class Test10_This {
    public static void main(String[] args) {
        //调用Teacher(String)构造方法实例化对象
        //底层借助Teacher(String, double)实现
        Teacher t = new Teacher("larry");
        t.show();
    }
}

```

运行效果：



```

<terminated> Test10_This [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe
Teacher(String,double) ...
name: larry
salary: 0.0

```

观察上图可知，构造方法Teacher(String)底层借助Teacher(String, double)实现了功能。

**注意：** `this(实际参数列表)` 必须是构造方法中的第一行有效代码。