day01【类与对象、成员变量、封装】

今日内容

- 面向对象思想
- 类与对象的使用
- 类与对象的内存图
- 成员变量和局部变量区别
- 封装

教学目标

- [ ] 能够知道类和对象的关系
- [ ] 能够完成类的定义及使用
- [ ] 能够知道对象在内存中的初始化过程
- [ ] 能够知道局部变量和成员变量的区别
- [ ] 能够知道private关键字的特点
- [ ] 能够知道this关键字的作用

第一章 复习回顾

1.1 if格式和流程

```
if（布尔表达式1）{

    语句体1;
} else if（布尔表达式2）{

    语句体2;
} else if（布尔表达式3）{

    语句体3;
} else {

    语句体n;
}
```

流程

1. 如果布尔表达式为ture,执行后面{}中的代码
2. 如果布尔表达式为false,看下一个布尔表达式

1.2 switch格式和流程

```
switch (表达式) {
    case 值1:
        语句体1;
        break;

    case 值2:
        语句体2;
        break;

    default:
        语句体n;
        break;
}
```

执行流程

拿表达式的值和case后面的匹配.匹配上哪个就执行哪个

1.3 for循环格式和流程

```
for (初始化表达式; 条件判断语句; 条件控制语句) {
    循环体;
}
```

执行流程

1. 初始化语句只会执行一次
2. 条件为ture继续循环
3. 条件为false结束循环

1.4 while循环格式和流程

```
初始化语句;
while (条件判断语句) {

    循环体;

    条件控制语句;
}
```

执行流程

1. 初始化语句执行一次
2. 条件判断语句为ture执行循环体
3. 条件判断语句为false,结束循环

1.5 数组的定义和使用

动态初始化数组的格式:

```
数据类型[] 数组名 = new 数据类型[长度];
```

静态初始化格式:

```
数据类型[] 数组名 = new 数据类型[] {数据1, 数据2, 数据3};
```

数组的访问:

```
获取元素:
    数组名[索引]

修改元素:
    数组名[索引] = 新的值;
```

1.6 方法的定义和使用

方法定义格式:

```
修饰符 返回值类型 方法名(参数列表) {
    方法体;
    return 结果;
}
```

方法调用格式:

```
直接调用: 方法名(); 没有处理返回的结果
赋值调用: 数据类型 变量名 = 方法名();  保存返回值,方便后续处理
输出调用: System.out.println(方法名()) 打印返回值
```

第二章 面向对象思想

2.1 面向对象思想概述

概述

Java语言是一种面向对象的程序设计语言，而面向对象思想是一种程序设计思想，我们在面向对象思维方式下，使用Java语言去设计、开发计算机程序。软件是模拟现实世界的，面向对象设计思想就是通过代码去高度模拟现实世界事物的。面向对象是非常接近现实世界的思想，面向对象是几乎所有高级语言都支持的设计，是现今最先进的软件设计思想。面向对象的语言是高级语言。 这里的**对象**泛指现实中一切具体存在的事物，每种事物都具备自己的**属性**和**行为**，例如我们每个人就是一个对象。

特点

面向对象思想是一种更符合我们对现实世界事物思考习惯的思想，它可以将复杂的事情简单化。面向对象的语言中最重要的两个概念是：类和对象。

2.2 类和对象

面向对象的语言中最重要的两个概念是：**类和对象。**类是一类具体事物的统称，是一个抽象的概念，对象是类具体存在的实例。

什么是类

- **类**：是相同事物共同特征（**行为，属性**）的描述。类是用来描述一类事物的：比如人类，老师类，学生类，动物类等都是**类。**
- **属性**：就是该事物的特征的信息。例如：人类都有年龄，名字，性别等特征。
- **行为**：就是该事物能够做什么。例如：人类唱歌，跳舞，计算机可以上网，学生学习等都是行为。

  举例：猫类 属性：名字、体重、年龄、颜色。 行为：走、跑、叫。 **小结**：类是一个抽象的概念，类是学术上的一个描述，就像人类只是一个概念，只是为了理解什么是人类。真实存在的是每个具体的人。

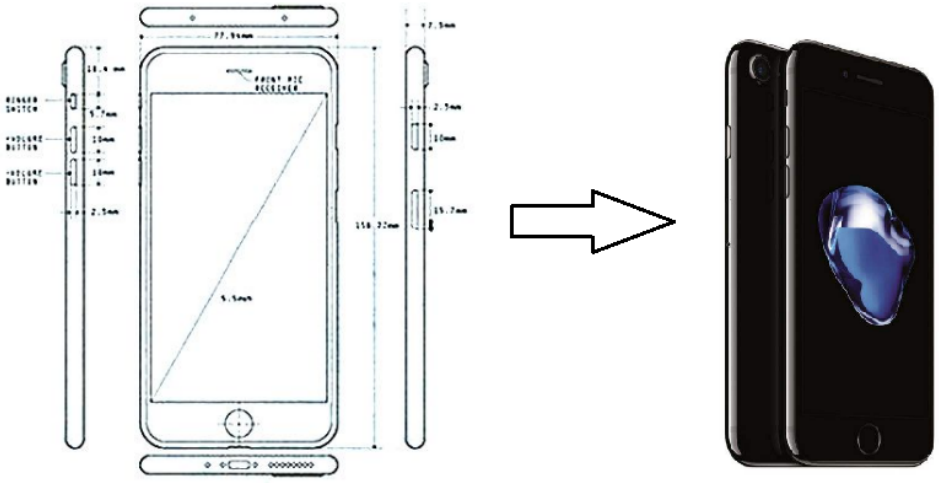
什么是对象

- **对象**：是一类事物的具体体现。对象是类真实存在的一个**实例。**通常我们可以认为对象就是实例的意思。

> 例如：我们说学生张三是学生类的一个对象，正在给我们上课的这位老师是老师类的一个对象。

类与对象的关系小结

- 类是对一类事物的描述，是**抽象的**的概念，并不是真实存在的事物对象。
- 对象是一类事物的具体实例，是**真实存在的。**
- **类是对象的模板，对象是类的实体**，例如我们每个人是人类的实体（也就是真实存在的个体的意思）。
- 类定义一个出来以后，例如学生类，而学生类的对象可以有千千万个。
- **在面向对象的开发中：必须先有类，才能创建类具体的对象。类与对象即可组成万千事物，软件就可以以此模拟现实世界的业务功能了。**



手机的设计图（抽象的）

真正的手机（具体的）

第三章 类与对象的使用

3.1 类的定义

面向对象是通过类和对象去描述和代表万千事物对象的，首先我们需要知道如何去定义一个类。

类的组成是由属性和行为两部分组成

- 属性：在类中通过成员变量来体现（类中方法外的变量）
- 行为：在类中通过成员方法来体现（和前面的方法相比去掉static关键字即可）

类的定义格式

类的定义步骤：

①定义类

②编写类的成员变量

③编写类的成员方法

```
public class 类名 {
    // 成员变量
    数据类型 变量1;
    数据类型 变量2;
    …
    // 成员方法
    方法1;
    方法2;
}
```

示例代码：

```
/*
    手机类：
        类名：
        手机(Phone)

        成员变量：
        品牌(brand)
        价格(price)

        成员方法：
        打电话(call)
        发短信(sendMessage)
 */
public class Phone {
    //成员变量
    String brand;
    int price;

    //成员方法
    public void call() {
        System.out.println("打电话");
    }
```

```
    public void sendMessage() {
        System.out.println("发短信");
    }
}
```

3.3 对象的创建和使用

如何得到对象

有了类后是不行的，我们必须创建类的对象，类的对象是可以有千千万万个的，以便用对象代表现实世界具体存在的一个事物。那么如何创建对象呢？创建对象的格式如下：

```
类名 对象名 = new 类名();
```

如何得使用对象

当我们创建对象后需要使用对象的属性和行为：格式如下：

```
使用对象的成员变量：
    对象名.成员变量

使用对象的成员方法：
    对象名.成员方法();
```

示例代码：

```
/*
    创建对象
        格式：类名 对象名 = new 类名();
        范例：Phone p = new Phone();

    使用对象
        1：使用成员变量
            格式：对象名.变量名
            范例：p.brand
        2：使用成员方法
            格式：对象名.方法名()
            范例：p.call()
 */
public class PhoneDemo {
    public static void main(String[] args) {
        //创建对象
        Phone p = new Phone();

        //使用成员变量
        System.out.println(p.brand);
        System.out.println(p.price);

        p.brand = "小米";
        p.price = 2999;

        System.out.println(p.brand);
        System.out.println(p.price);
```

```
        //使用成员方法
        p.call();
        p.sendMessage();
    }
}
```

3.4 学生对象-练习

- 需求：首先定义一个学生类，然后定义一个学生测试类，在学生测试类中通过对象完成成员变量和成员方法的使用
- 分析：
    - 成员变量：姓名，年龄...
    - 成员方法：学习，做作业...
- 示例代码：

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

    //成员方法
    public void study() {
        System.out.println("好好学习，天天向上");
    }

    public void doHomework() {
        System.out.println("键盘敲烂，月薪过万");
    }
}

/*
    学生测试类
 */
public class StudentDemo {
    public static void main(String[] args) {
        //创建对象
        Student s = new Student();

        //使用对象
        System.out.println(s.name + "," + s.age);

        s.name = "林青霞";
        s.age = 30;

        System.out.println(s.name + "," + s.age);

        s.study();
        s.doHomework();
    }
}
```

3.5 成员变量的默认值

从上面对象访问成员变量属性（email）可以看出，成员变量可以不给初始值的，成员变量实际上是存在默认值的，默认值的规则如下：

	数据类型	默认值
基本类型	整数（byte，short，int，long）	0
	浮点数（float，double）	0.0
	字符（char）	'\u0000'
	布尔（boolean）	false
引用类型	数组，类，接口	null

第四章 类与对象的内存图

4.1 概述

在之前的开发中我们只是知其然，而不知其所以然，本章将告诉大家代码运行的内存图，这将使我们非常直观理解面向对象的程序执行过程。

4.2 一个对象内存图

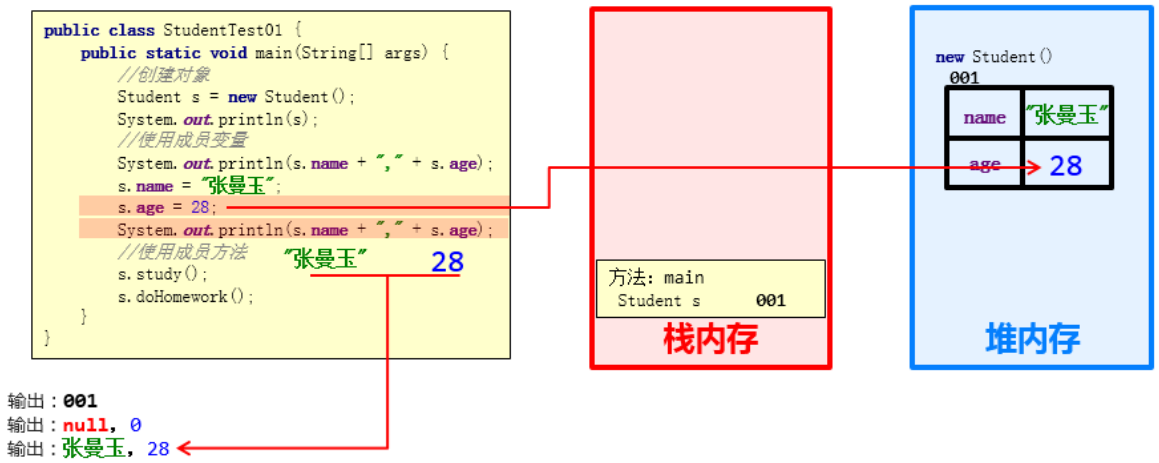
我们使用前面的学生类来介绍对象的内存图

代码

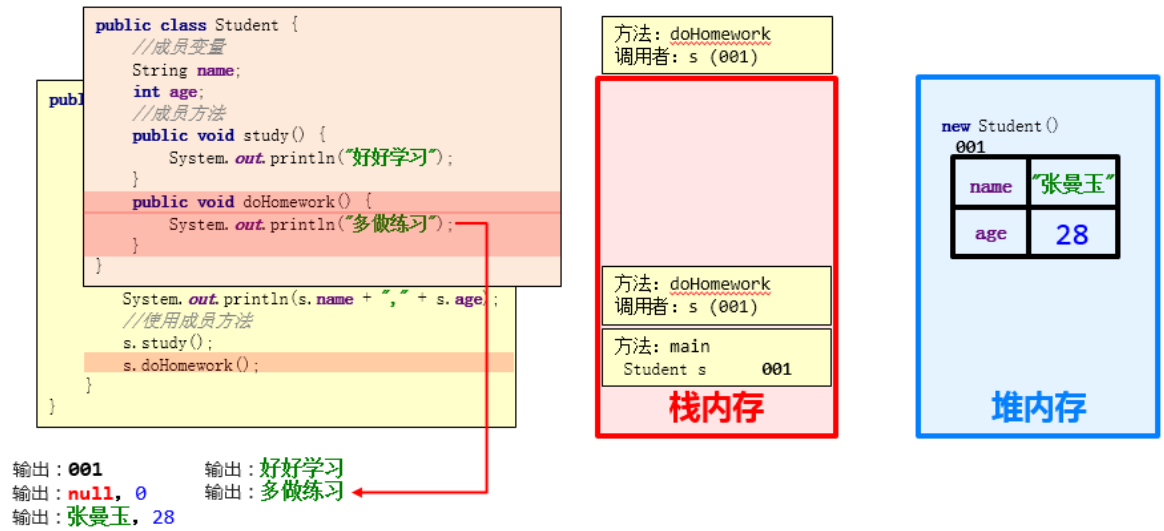
```
/*
    学生测试类
 */
public class StudentDemo {
    public static void main(String[] args) {
        //创建对象
        Student s = new Student();

        //使用对象
        System.out.println(s.name + "," + s.age);

        s.name = "张曼玉";
        s.age = 28;

        System.out.println(s.name + "," + s.age);

        s.study();
        s.doHomework();
    }
}
```

内存图

成员变量使用过程



成员方法调用过程



4.3 两个对象内存图

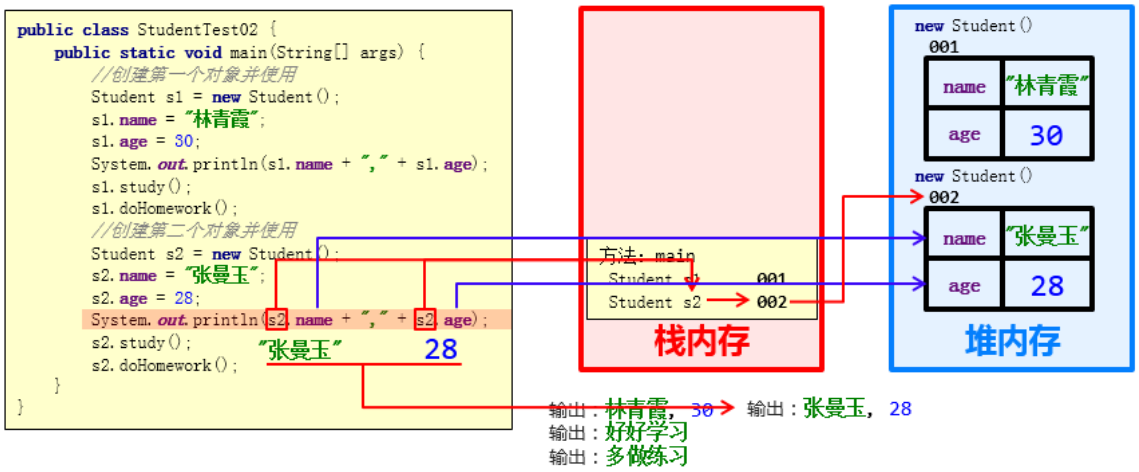
代码

```
/*
    学生测试类
 */
public class StudentDemo2 {
    public static void main(String[] args) {
        //创建对象
        Student s1 = new Student();
        s1.name = "林青霞";
        s1.age = 30;
        System.out.println(s1.name + "," + s1.age);
        s1.study();
        s1.doHomework();

        Student s2 = new Student();
        s2.name = "张曼玉";
```

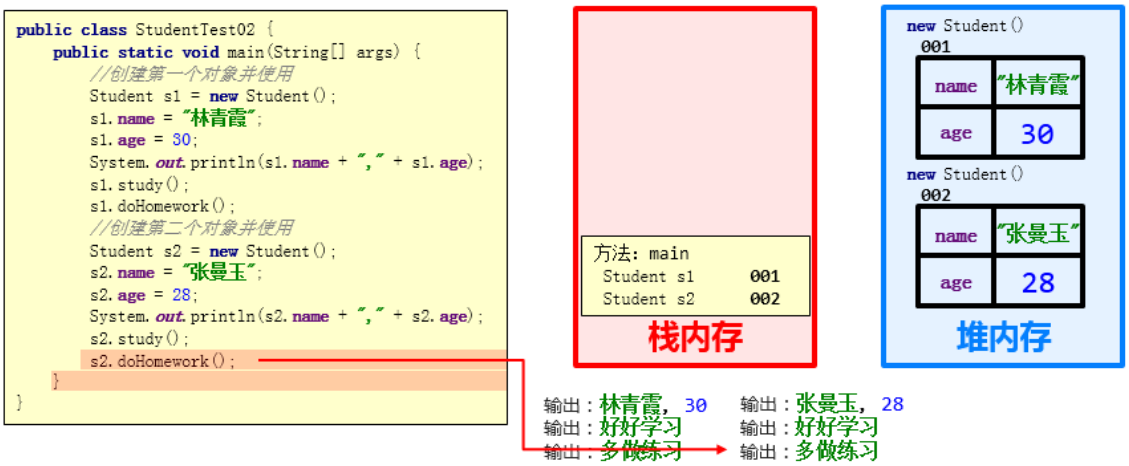
```
        s2.age = 28;
        System.out.println(s2.name + "," + s2.age);
        s2.study();
        s2.doHomework();
    }
}
```

内存图

成员变量使用过程



成员方法调用过程



总结：

多个对象在堆内存中，都有不同的内存划分，成员变量存储在各自的内存区域中，成员方法多个对象共用的一份

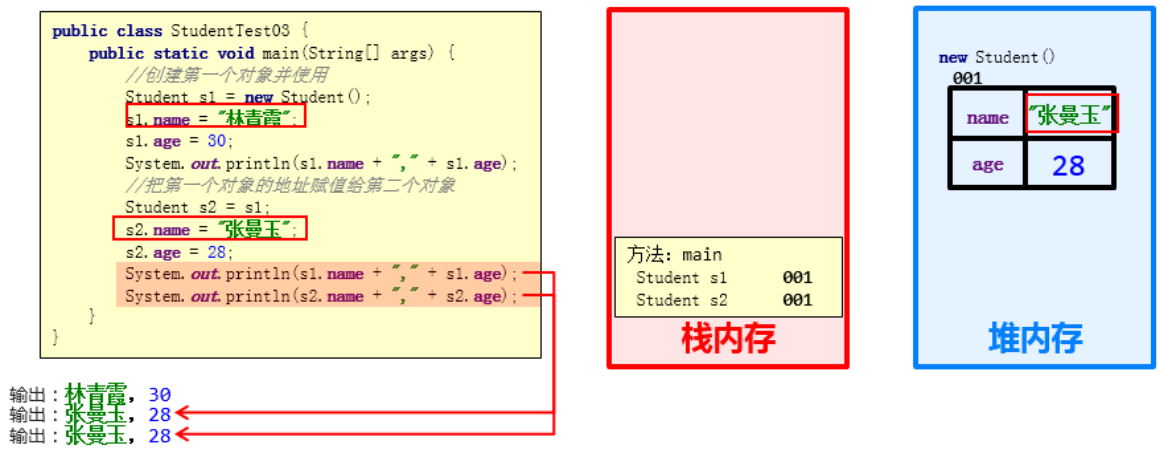
4.4 多个对象指向相同内存图

代码

```
/*
    学生测试类
 */
public class StudentDemo3 {
    public static void main(String[] args) {
        //创建对象
```

```
        Student s1 = new Student();
        s1.name = "林青霞";
        s1.age = 30;
        System.out.println(s1.name + "," + s1.age);

        Student s2 = s1;
        s2.name = "张曼玉";
        s2.age = 28;
        System.out.println(s.name + "," + s.age);
        System.out.println(s2.name + "," + s2.age);
    }
}
```

内存图



总结

当多个对象的引用指向同一个内存空间（变量所记录的地址值是一样的）

只要有任何一个对象修改了内存中的数据，随后，无论使用哪一个对象进行数据获取，都是修改后的数据。

第五章 成员变量和局部变量区别

变量根据定义**位置的不同**，我们给变量起了不同的名字。如下图所示：

- 在类中的位置不同 重点

  成员变量：类中，方法外 局部变量：方法中或者方法声明上(形式参数)

- 作用范围不一样 重点

  成员变量：类中 局部变量：方法中

- 初始化值的不同 重点

  成员变量：有默认值 局部变量：没有默认值。必须先定义，赋值，最后使用

- 在内存中的位置不同 了解

  成员变量：堆内存 局部变量：栈内存

- 生命周期不同 了解

  成员变量：随着对象的创建而存在，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

第六章 封装

6.1 封装概述

面向对象有很多设计上的语法是需要大家掌握的，接下来我们需要学习**面向对象的三大特征中的封装。**

面向对象编程语言是对客观世界的模拟，客观世界里成员变量都是隐藏在对象内部的，外界无法直接操作和修改。封装可以被认为是一个保护屏障，防止该类的代码和数据被其他类随意访问。要访问该类的数据，必须通过指定的方式。适当的封装可以让代码更容易理解与维护，也加强了代码的安全性。

例如：之前我们定义的方法，实际上也是一种封装，我们是把功能代码封装在了方法中，然后必须通过调用该方法来执行功能。

6.2 为什么要封装

假如我们现在定义一个People类。代码如下：

```
public class People {
    // 实例成员变量,属于对象的，无static修饰
    // 合理隐藏
    public String name ;
    public int age  ;
}
```

之前我们已经知道对象可以直接访问这些实例成员变量。那么就会出现以下情况：

```
public class TestPeople {
    public static void main(String[] args){
        // 创建一个人对象
        People p = new People();
        p.name = "古力娜扎";
        p.age = -100; // 很显然这个数据注入是存在问题的，年龄不可能是负数
    }
}
```

上面年龄age直接注入数据是不合理的，说明age不应该直接就被访问，应该把age封装起来，以便不能被直接访问。那么如何进行封装呢?

6.3 封装原则

封装是为了让代码的功能更加的安全，让细节被进一步的隐藏起来。原则上应该**合理隐藏，合理暴露。**

按照规范:我们需要将**成员变量(属性)隐藏**起来，若需要访问某个成员变量，**暴露公共方法**对其访问。

6.4 封装的步骤

1.使用 `private` 关键字来修饰成员变量。表示成员变量不能被直接访问。

2.使用public修饰方法，提供对应的一对用public修饰的 `getXxx方法 、setXxx` 方法来暴露成员变量的访问。

注意：public关键字是公开访问权限，private关键字是隐藏访问权限。

6.5 封装的操作

private修饰成员变量

private的含义

1. private是一个权限修饰符，代表最小权限，是**私有**的含义。
2. 可以修饰成员变量和成员方法。
3. 被private修饰后的成员变量和成员方法，只在本类中才能访问。

private的使用格式

```
private 数据类型 变量名;
```

使用 `private` 修饰成员变量，代码如下：

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

public修饰getter和setter方法

public的含义

1. public是一个权限修饰符，代表最大权限，是**公开权限**的含义。
2. 可以修饰成员变量和成员方法。
3. 被public修饰后的成员变量和成员方法，可以在任何类中访问。

private的使用格式

```
private 数据类型 变量名;
```

使用public修饰成员方法，代码如下：

1.提供 `getXxx` 方法 / `setXxx` 方法（我们也称呼 `getXxx方法` / `setXxx` 方法是getter和setter方法），可以通过getter和setter方法来暴露成员变量的访问，然后可以在setAge方法中过滤掉非法的年龄，代码如下：

```
public class People {
    private String name;
    private int age;
    private String addr;

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

    public String getName() {
        return name;
    }

    public String getAddr() {
        return addr;
    }

    public void setAddr(String ar) {
        addr = ar;
    }

    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;
    }
}
```

接下来我们通过代码进行测试

```
public class TestPeople {
    public static void main(String[] args){
        // 创建一个人对象
        People p = new People();
        // p.age = -100;//这里代码会出现编译错误，因为age被private修饰了，只能通过setAge
访问修改
        // p.setAge(-10); // 提示年龄非法
        p.setAge(19);
        p.setName("张曼玉");
        p.setAddr("香港")
        System.out.pritnln(p.getAge());// 输出年龄：19
        System.out.pritnln(p.getName());// 输出：张曼玉
        System.out.pritnln(p.getAddr());// 输出：香港
    }
}
```

从上面代码可以看出：

成员变量age一旦私有以后，其他类就不能直接访问成员变量了，必须通过setAge来修改，通过getAge来访问。这样就实现了安全性，值得注意的是开发的时候，我们并不一定会在setAge中做参数的校验，因为可以在界面上提前校验用户输入的数据是否合法，但是即便如此，**成员变量私有和提供配套的getter和setter方法已经成为Java设计的一种规范**，建议大家都这样设计自己的代码。

6.6 封装小结

1.从上述代码中可以看出，成员变量使用private修饰以后，就不可以在其他类中访问了。

2.提供成套的被public修饰的setter和方getter法可以暴露对私有成员变量的修改以及获取值。

3.封装就是合理的进行隐藏，合理的进行暴露，按照规范我们会对成员变量进行私有，而对方法进行暴露，当然在一些极特殊的情况下，成员变量也会功能，方法也可能会私有。

6.7 this关键字

this修饰的变量用于指代成员变量，其主要作用是（区分局部变量和成员变量的重名问题）

- 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
- 方法的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

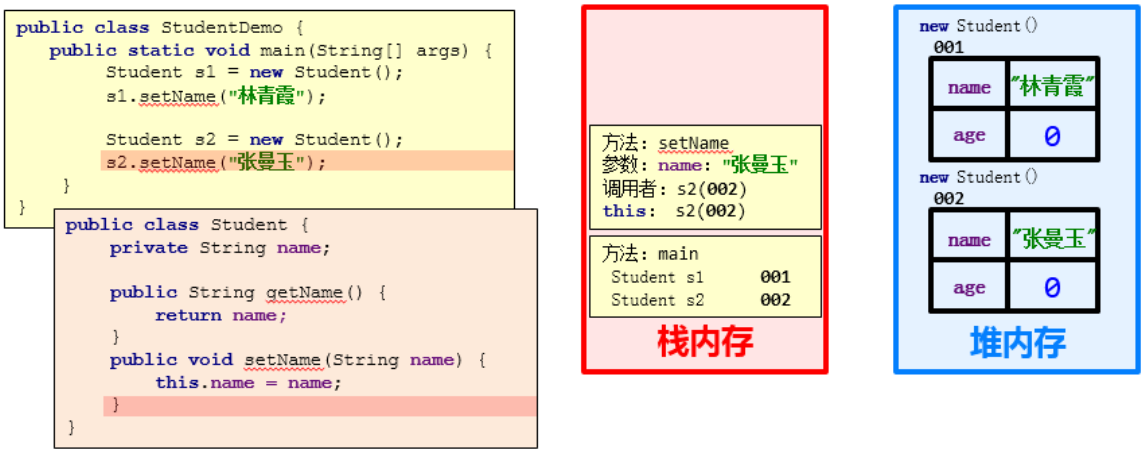
```
public class Student {
    private String name;
    private int age;

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

    public String getName() {
        return name;
    }

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

    public int getAge() {
        return age;
    }

    public void show() {
        System.out.println(name + "," + age);
    }
}
```

6.8 this内存原理

- this代表当前调用方法的引用，哪个对象调用的方法，this就代表哪一个对象
- 示例代码：

```
public class StudentDemo {
    public static void main(String[] args) {
        Student s1 = new Student();
        s1.setName("林青霞");
        Student s2 = new Student();
        s2.setName("张曼玉");
    }
}
```

图解：



6.9 封装思想

1. 封装概述 是面向对象三大特征之一（封装，继承，多态） 是面向对象编程语言对客观世界的模拟，客观世界里成员变量都是隐藏在对象内部的，外界是无法直接操作的
2. 封装原则 将类的某些信息隐藏在类内部，不允许外部程序直接访问，而是通过该类提供的方法来实现对隐藏信息的操作和访问 成员变量private，提供对应的getXxx()/setXxx()方法
3. 封装好处 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