

# 第二阶段-面向对象

## 面向对象 - Day01（类、对象）

### 今日学习内容

- 软件的开发方式
- 成员变量和局部变量
- 类的定义和对象的创建
- 对象的操作
- 构造器
- 封装思想
- 访问权限修饰符
- JavaBean规范

### 今日学习目标

- 了解什么是面向过程和面向对象
- 掌握变量的分类有哪些
- 记住成员变量的初始值
- 记住变量的作用域
- 记住变量的生命周期
- **必须掌握类的定义**
- **必须掌握对象的创建和操作方法和字段**
- **必须掌握构造器的定义和作用**
- 理解封装思想的好处
- **必须掌握private和public两个访问权限修饰符**
- **必须掌握JavaBean规范的四个要求**

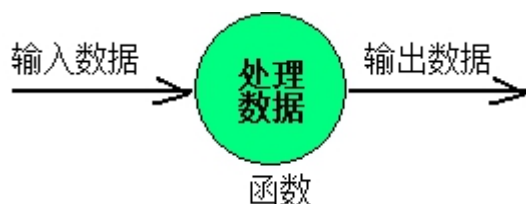
## 第一章、面向对象基础概述

### 1.1 软件开发方式（了解）

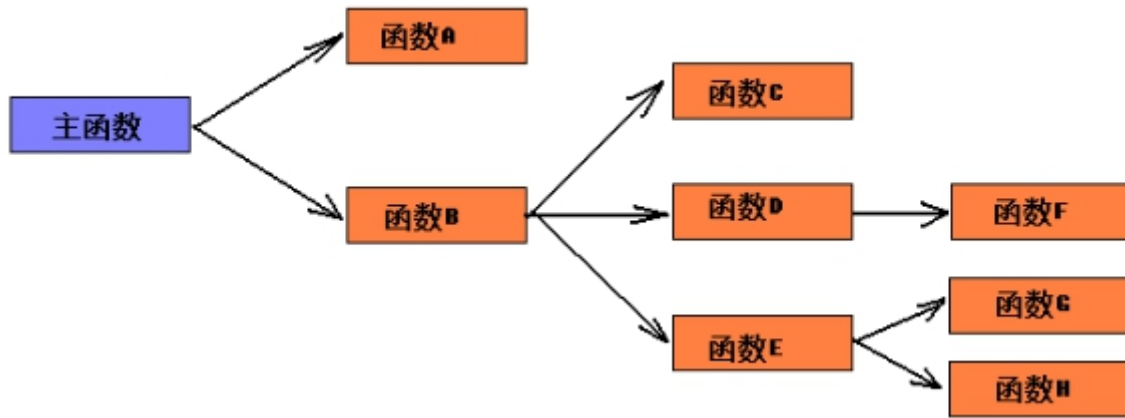
#### 1.1.1 面向过程（了解）

一种较早的编程思想，顾名思义该思想是站在过程的角度思考问题，强调的是**我该怎么去做**。即功能的执行过程，即先干啥，后干啥。

面向过程思想中函数(=>方法)是一等公民，每个方法负责完成某一个功能，用以接受输入数据，函数对输入数据进行处理，然后输出结果数据。



而每一个功能我们都使用函数（类似于方法）把这些步骤一步一步实现，使用的时候依次调用函数就可以了。

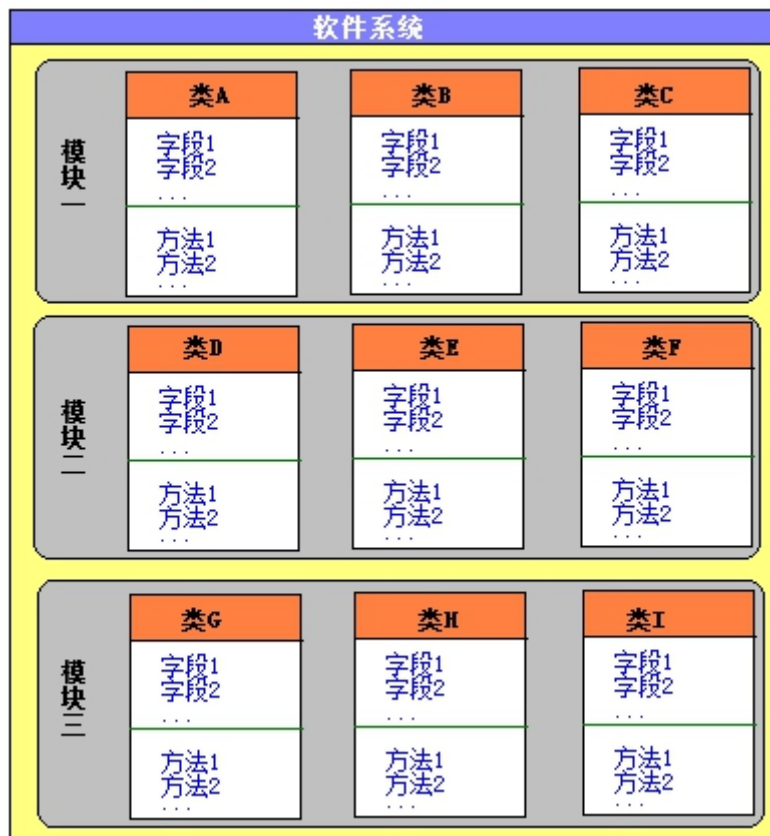


面向过程的设计思想，系统软件适应性差，可拓展性差，维护性低。

### 1.1.2 面向对象（了解）

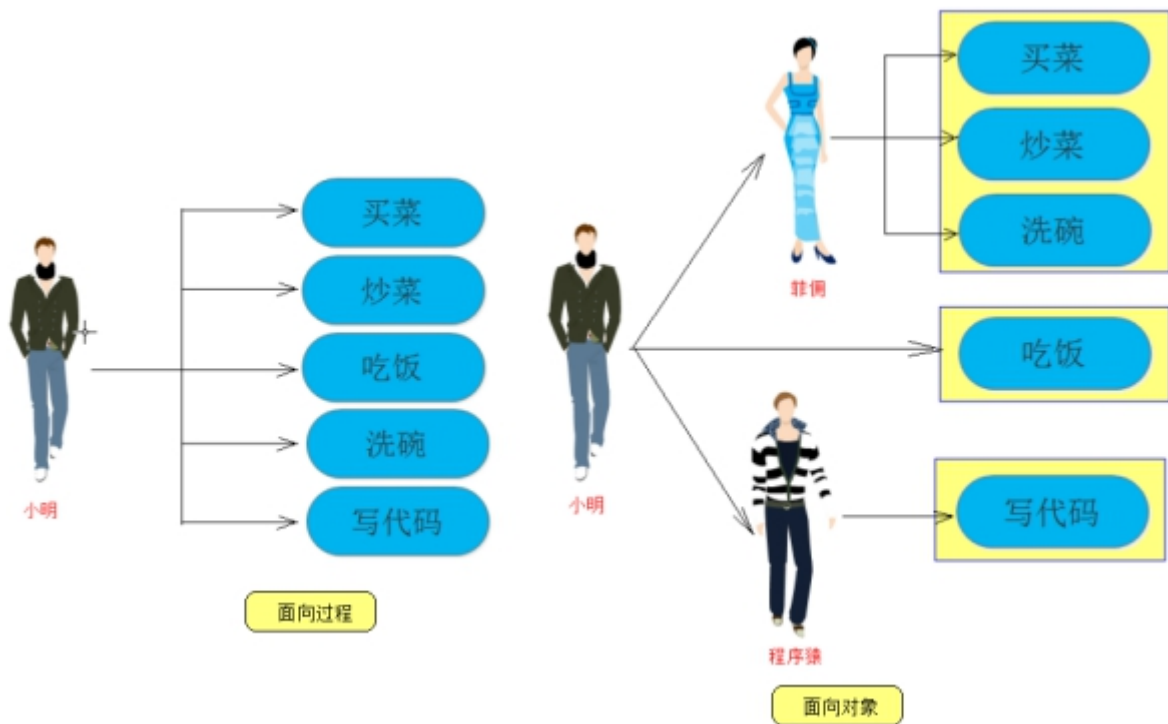
一种基于面向过程的新的编程思想，顾名思义该思想是站在对象的角度思考问题，我们把多个功能合理的放到不同对象里，强调的是**我该让谁来做**。

面向对象最小的程序单元是类，必须先存在类的定义，再有对象，而具备某种功能的实体，称为对象。



举个例子，小明完成买菜，做菜，吃饭，洗碗，写代码功能。

大家一起来看看有对象和没对象的区别：



左图是没有对象的，右图是有对象的。区分面向过程的我该怎么做和面向对象的我该让谁来做的思想。

面向过程和面向对象各有千秋，面向对象更符合我们常规的思维方式，稳定性好，可重用性强，易于开发大型软件产品，有良好的可维护性，它拥有**三大特征**：

- 封装 (Encapsulation)
- 继承 (Inheritance)
- 多态 (Polymorphism)

面向对象的学习，在基础班至少必须掌握**知识点的定义和使用问题**，思想是比较深远的，在大神班我们还会结合很多案例，去学习和巩固面向对象更多的精髓。

## 1.2 成员变量和局部变量（掌握）

回忆变量的定义语法：

```
数据类型 变量名 [= 初始值];
```

### 1.2.1 变量的分类（能区分）

根据变量定义位置的不同，分成两大类：

- **成员变量**：直接定义在类中，方法外面。又称之为字段（Field），先不要称之为属性（错误），后讲。
- **局部变量**：除了成员变量，其他都是局部变量，具体存在于三个地方：
  - 方法内
  - 方法的形参
  - 代码块中（一对花括号）

以下代码中，哪些是成员变量哪些是局部变量？

```
public class App {
```

```

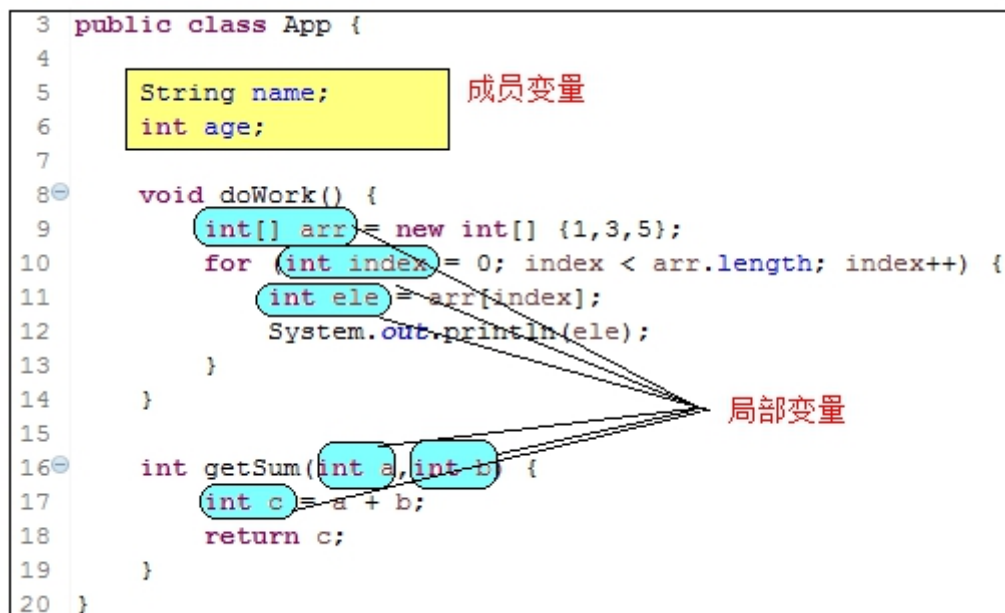
String name;
int age;

void dowork() {
    int[] arr = new int[] {1,3,5};
    for (int index = 0; index < arr.length; index++) {
        int element = arr[index]; // element元素的意思
        System.out.println(ele);
    }
}

int getSum(int a,int b) {
    int c = a + b;
    return c;
}
}

```

你猜对了吗？



### 1.2.2 变量的作用域 (要记住)

变量根据定义的位置不同，也决定了各自的作用域是不同的，**关键看变量所在的那对花括号{ }**

- 局部变量：从开始定义的位置开始，只能在自己所在的花括号内有效
- 成员变量：在所定义的类中都有效

{ } 也常常被称为作用域，作用域可以嵌套，内层作用域可以访问外层作用域的变量，反之不成立。

### 1.2.3 变量的初始值 (要记住)

变量的初始化表示在内存中开辟存储空间，只有初始化之后，才能使用。

- 局部变量：没有初始值，所以必须**先定义后赋值再使用**
- 成员变量：可以不赋值，默认是有初始值的，不同类型的初始值，如下图：

数据类型	默认初始化值
byte、short、int、long	0
float、double	0.0
char	一个空字符（空格），即'\u0000'
boolean	false
引用数据类型	null，表示不引用任何对象

### 1.2.4 变量的生命周期（要记住）

变量的生命周期，表示变量在内存存在能存活多久。

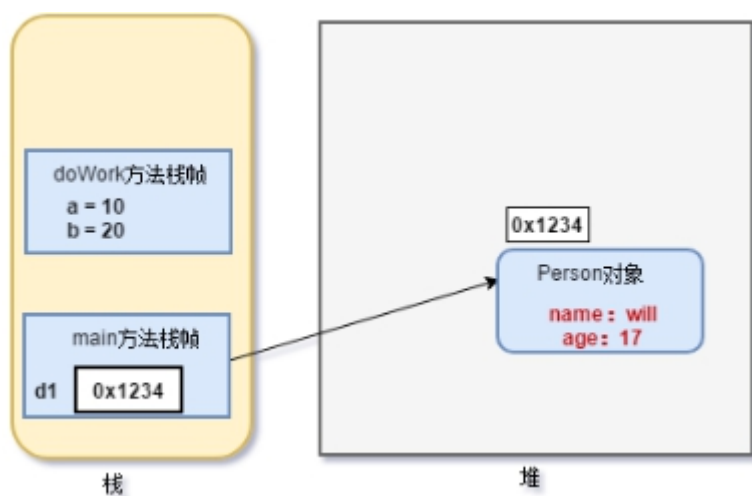
- 成员变量：**存储在堆内存中**，随着对象的销毁而销毁（文档后续对象内存图详解）
- 局部变量：**存储在栈内存中**，随着所定义方法的调用而产生，当方法结束而销毁
  - 局部变量存储在方法中，每次调用方法都会在栈空间开辟一块内存空间——栈帧，方法调用结束，栈帧就被销毁了，内存中的存储的变量数据也销毁了。

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

public class Demo {

    public static void dowork() {
        int a = 10;
        int b = 20;
    }

    public static void main(String[] args) {
        Demo.dowork();           // 调用dowork方法
        Person p = new Person(); // 记住每次new 就表示在堆空间开辟一块空间
    }
}
```



当doWork方法被调用时，在栈顶位置分配doWork方法的内存空间，存储局部变量数据。

当doWork方法调用结束，doWork方法的栈帧就被销毁，main方法重新回到栈顶位置。

堆中存储的都是new出来的对象（数组其实也是对象）。

## 为什么要学习类和对象？

在计算机中存储一个年龄 `int age = 20;`

在计算机中存储多个人的年龄 `int[] ages = {20,21,23...}`

思考 1：复杂的数据如何存储到计算机中？

**门店有售 Apple iPhone 11 (A2223) 128GB 白色 移动联通电信4G手机 双卡双待**

【公开版到手价4899元!】选购iPhone11ProMax, 仅加2000元得6.5英寸超大显示屏手机! 点击抢购! [查看>](#)

京 东 价 **¥ 4899.00** 降价通知

累计评价  
400万+

促 销 **组套商品**  × 1  × 1

**换购** **换购** **满额返券**

展开促销 ∨

增值业务 **3 高价回收, 极速到账**

配 送 至 **广东广州市天河区洗村街道** ∨ 有货 支持 可配送港澳台 | 99元免基础运费

**京东物流** 京准达 | 211限时达 | 京尊达 ∨

由 **京东** 发货, 并提供售后服务. 23:10前下单, 预计明天(03月15日)送达




重 量 0.371kg

选择颜色  黑色  **白色**  红色  黄色  紫色  绿色

选择版本 **64GB** **128GB** 256GB

购买方式 **公开版** 快充套装 换修无忧月付版 换修无忧年付版 老包装-含充电头耳机 AirPods套装

购买方式 **官方标配**

增值保障  换修无忧1年 ¥599.00/年 ∨  2年碎屏保修 ¥289.00 ∨  2年电池换新 ¥39.00 ∨ ?

京东服务  黑科技充电宝 ¥129.00 ∨  服务赠充电宝 ¥68.00 ∨ ?

## 1.3 类(class)的定义（重点掌握）

类是拥有相同特征（状态）和相同行为（功能）的多个事物的抽象(抽出像的部分)描述。

类是用来描述群体的，是对群体的抽象描述（思考如何描述群体）。

例如：白富美：女 白净 富有 漂亮 / 爱学java

在程序中，类就可以作为一种新型的数据类型。可以使用类来声明变量。

描述类或者说定义类，就从特征和行为上分析，那么怎么来表示特征和行为呢？

- 使用**成员变量**来表示特征 (状态)
- 使用**成员方法**来表示行为 (功能)

类定义语法格式：

```

public class 类名 {

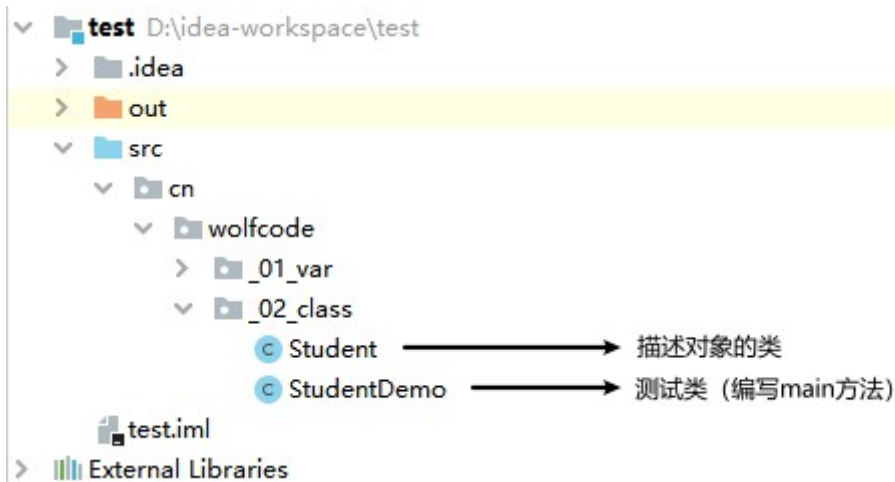
    //可编写0到N个成员变量
    [修饰符] 数据类型 变量名1;
    [修饰符] 数据类型 变量名2;

    //可编写0到N个成员方法
    [修饰符] 返回值类型 方法名称(参数){
        //方法体
    }
}

```

注意:

- 成员变量和方法都 **不能**使用static修饰，修饰符是可选用的，都先不使用任何修饰符
- 在面向对象设计中，描述对象的类和测试类分开来编写
- 在描述对象的类中，不需要定义main方法，专门在测试类中提供main方法。



需求：描述一个学生类Student，学生有名字和年龄两个状态，有自我介绍(控制台打印名字和年龄)的行为。

```

/**
 * 步骤：
 * step1:定义类，取类名(大写的驼峰)，一般使用名词。
 * step2:定义成员变量/字段
 * step3:定义成员方法
 * step4:写测试类，声明一个Student类型的student变量
 */

public class Student {
    // step 2:使用成员变量表示学生类的特性(状态)
    String name;
    int age;

    // step 3: 定义成员方法表示学生的行为(功能)
    void sayHi() {
        System.out.println("我是" + name + ",我今年" + age + "岁");
    }
}

```

注意：此时的Student类前面的public单词是一个修饰符，表示公共访问权限，先用着，后面细讲。

## 1.4 对象操作（重点掌握）

对象：对象是类的一个具体实例（实实在在的例子），它用于强调一个具体的个体。

对象是独立的，唯一的个体。

对象一定属于某一类(型)，具备该类的特性和行为。

例如：白富美中刘亦菲的是具体的对象。

### 1.4.1. 对象基本操作（重点掌握）

- 创建对象

```
类名 对象变量名 = new 类名();
```

1. 直接打印对象的时候，打印的是类似于数组一样的内存地址

```
// 输出格式如：类名@3294e4f4  
System.out.println(对象变量名);
```

2. 匿名对象：创建对象之后没有赋给某一个变量，只能使用一次（先知道）

```
new 类名();
```

- 对象操作字段（成员变量）

1. 给字段设置数据

```
对象变量名.字段名 = 值;
```

2. 获取字段数据

```
数据类型 变量 = 对象变量名.字段名;
```

- 对象调用方法

```
对象变量名.方法(参数);
```

测试代码如下：

```
public class StudentDemo {  
    public static void main(String[] args) {  
        // 1: 创建Student对象,对象的变量名为stu  
        Student stu = new Student();  
  
        // 2: 获取字段的值,先看看字段的初始值  
        String name = stu.name;  
        int age = stu.age;  
        System.out.println("名字=" + name);  
        System.out.println("年龄=" + age);  
  
        // 3: 给字段设置值
```



```

    stu.name = "汤姆";
    stu.age = 5;

    // 4:设置字段值后，再次获取字段的值
    name = stu.name;
    age = stu.age;
    System.out.println("名字=" + name);
    System.out.println("年龄=" + age);

    // 5:调用对象的方法
    stu.sayHi(); // 说话的功能
}

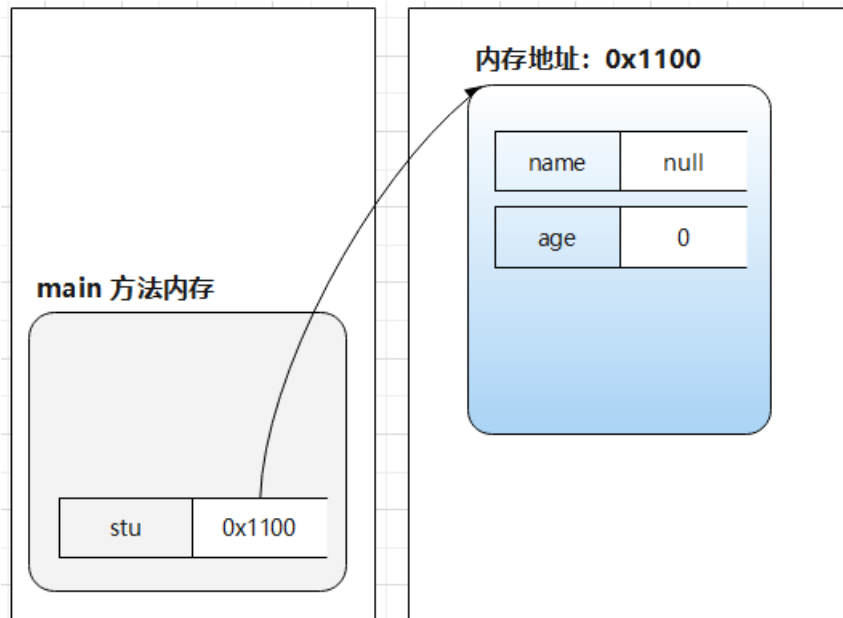
```

#time 15.00

#### 1.4.2. 对象实例化过程-内存分析 (理解即可)

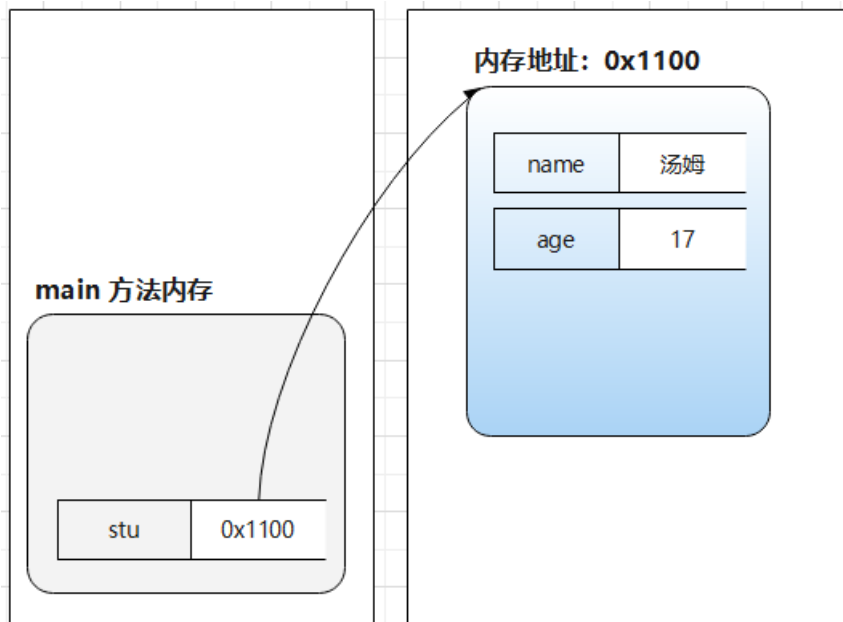
分析内存的目的，是为了更清楚数据在内存中的存储和变化，方便我们理解和分析。

- 创建对象 `Student stu = new Student();`



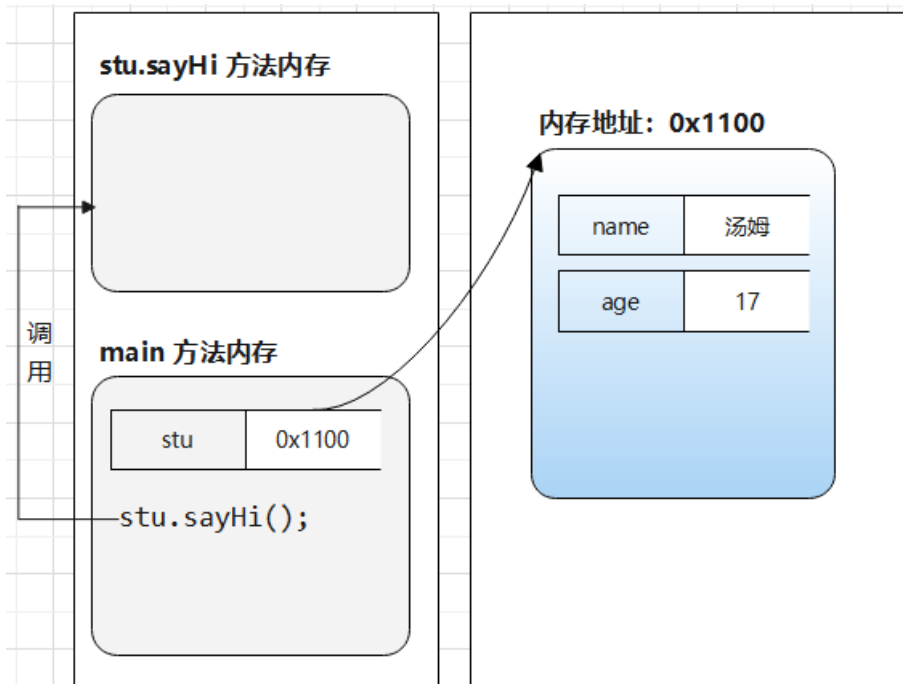
此时通过stu对象查看name和age，初始值分别是null和0。

- 通过对象给字段设置数据 `stu.name = "汤姆"; c.age = 5;`



此时通过stu对象查看name和age的值，分别是汤姆 和17。

- 通过对象调用方法 `stu.sayHi(); System.out.println("我是" + name + ",我今年" + age + "岁");`



- sayHi方法调用结束，也就是`stu.sayHi();`代码执行完成。sayHi()方法栈帧从栈内存销毁（出栈）后如图二。

## 1.5.类和对象的关系（了解）

面向对象思想中有两个非常重要的概念，类和对象，其中：

- 类（class），是对某一类事物的抽象描述（状态和行为），如下图的抽象女人图。
- 对象（object），表示现实生活中该类事物的个体，也称之为实例，如下图的每一个具体的女人。
- 类可以看作是对象的**数据类型**，就好比无论你是谁，只要是女人，那么类型就是女人。



上图，从左往右看是抽象的过程，从右往左看是实例化的过程，所谓实例化就表示根据类这个模板制造出每一个对象。

任何事物存在就有一定的功能，在面向对象中，任何事物都可以看成对象，也就是万物皆对象的概念。

注意：

- 在开发中，必须先有类，再使用new关键字根据类来创建出对象。

## 1.6.构造器/构造方法（重点掌握）

创建学生对象的时候，代码如下：

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

其实这段代码，是在通过调用Student类的构造器，来创建对象。

构造器，也称之为构造方法（Constructor），作用是用来**创建对象和给对象做初始化操作**，专门用于构造一个对象。

构造方法的语法和特点：

```
// 普通方法语法
[修饰符] 返回值类型 方法名(参数列表) {

}

// 构造方法语法
[修饰符] 类名(参数列表){

}
```

对比观察发现：

- 构造器名称必须和类名相同
- 不能定义返回类型
- 构造器中不能使用return语句

### 1.6.1.默认的构造器（掌握）

但是我们现在在Student类中，却找不到该构造器，而在创建对象时却没有报错，这是为什么？如果源文件中没有定义构造器，编译器在编译源文件的时候，会创建一个默认的构造器。

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

    // Student 类的默认构造器
    public Student() {

    }

    public void sayHi() {
        System.out.println("我是" + name + ",今年" + age + "岁");
    }
}
```

默认构造器的特点：无参数、方法体中无任何代码，因为无参数，经常被称为**无参构造器**。

### 1.6.2.通过构造器设置初始值（掌握）

之前，我们是先通过一个默认参数构造器，先创建一个对象，然后再初始化（给字段设置值）

```
Student stu = new Student();
stu.name = "汤姆";
stu.age = 17;
```

有了构造器之后，可以直接通过构造器同时完成对象创建和初始化操作。

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

    public Student(String aName,int aAge) {
        name = aName;
        age = aAge;
    }

    public void sayHi() {
        System.out.println("我是" + name + ",今年" + age + "岁");
    }
}
```

注意：当显式定义出构造器之后，编译器不再创建默认的构造器了。

```
Student stu = new Student("汤姆",17);
stu.sayHi();
```

因为不再有无参数构造器，之前的创建对象的代码报错。

```
Student stu = new Student(); // 此行报错
```

此时，我们可以在Student类中同时存在带参数和不带参数的构造器，他们之间的关系就是重载关系。

```

public class Student {
    String name;
    int age;

    // 无参数构造器
    public Student() { }

    // 带参数构造器
    public Student(String aName,int aAge) {
        name = aName;
        age = aAge;
    }

    // 其他代码
}

```

通过带参数创建对象并给对象初始化（给字段设置值）

```

public class ConstructorDemo2 {
    public static void main(String[] args) {
        // 使用有参构造器创建对象并给对象初始化（给字段设置值）
        Student stu = new Student("汤姆",17);
        stu.sayHi();
    }
}

```

小结：构造器就先了解到这即可，后面再讲，现在需要掌握的是：

**记住构造器的定义语法和功能，手动写出构造器代码，认识同一个类中的多个构造器之间是重载关系。**

#16.30

## 第二章、封装思想(重点掌握)

封装 ( encapsulation ) 是面向对象三大特征之一，其含义有两个（掌握思想）：

- 把对象的字段和方法存放在一个独立的模块（类）中
- 信息隐藏，尽可能隐藏对象的数据和功能的实现细节

封装的好处：

- 提高组件的重用性，把公用功能放到一个类中，谁需要该功能，直接调用即可
- 保证数据的安全性，防止调用者随意修改数据

没有封装带来的困惑：

学生类：

```

public class Student{
    String name;
    int age;

    public void sayHi() {
        System.out.println("我是" + name + ",今年" + age + "岁");
    }
}

```

测试类：

```
public class StudentDemo {
    public static void main(String[] args) {
        Student stu = new Student();
        stu.name = "小明";
        stu.age = -12;
        stu.sayHi();    // 我是小明,今年-12岁
    }
}
```

此时从代码语法上来看，是没有任何问题的，但是从逻辑上来分析人的年龄怎么能是负数呢？造成该问题的根本原因就是：可以随意访问对象中的字段。

那么问题来了，怎样才能限制不能随意访问字段数据呢？

此时，就该访问修饰符登场了！

## 2.1 访问修饰符（必须记住）

车库有一个车位，旁边写着“公共车位”，那么该车位就是公共的，谁都可以访问它。如果我在车位旁边写上“私家车位”，那么该车位就只能是我自己来访问。外界（除我之外）都访问不了，像“公共”、“私有”这种限制外界访问的标记符号，就称之为**访问修饰符**。

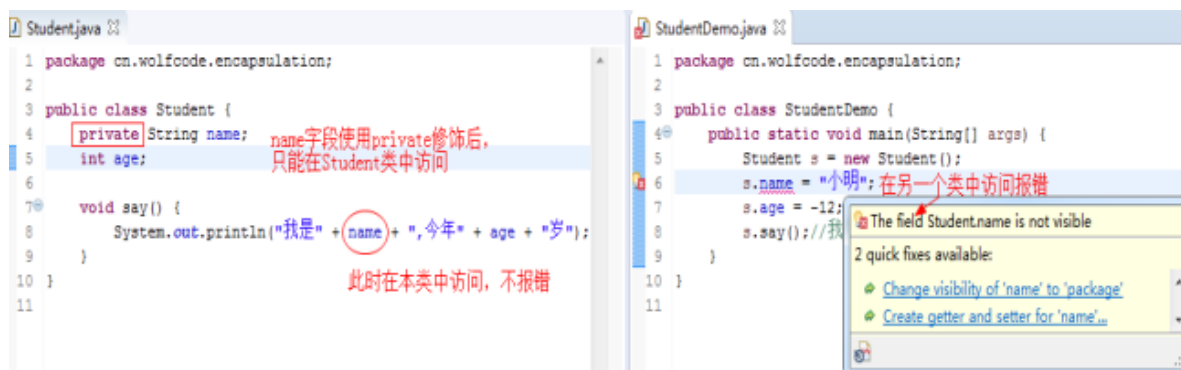
访问修饰符，决定了有没有权限访问某个成员(资源)。

封装其实就是要让有些类看不到另外一些类中定义的字段和方法。Java提供了不同的访问权限修饰符来限定类中的成员让谁可以访问到。

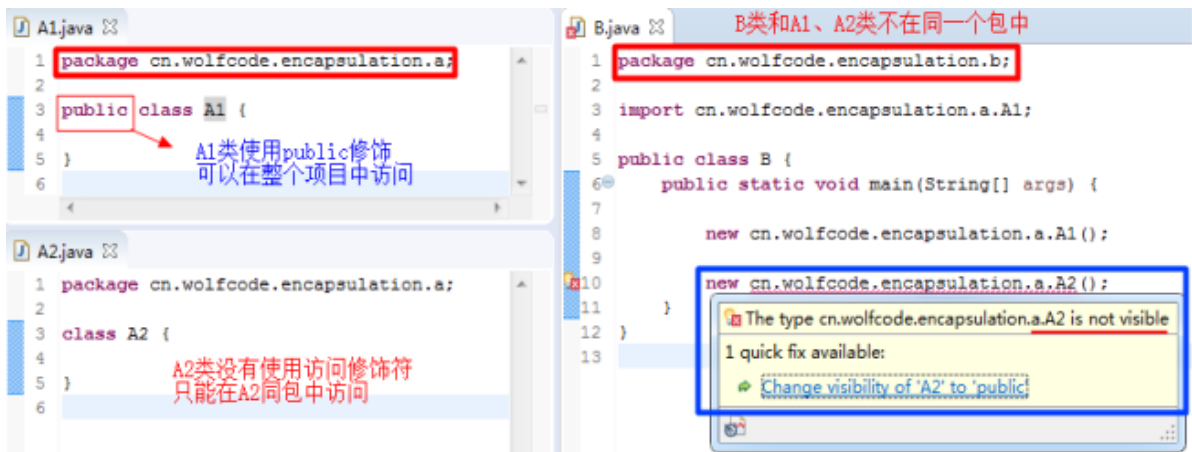
修饰符	类内部	同一个包	子类	任何地方
private	✓			
无	✓	✓		
protected	✓	✓	✓	
public	✓	✓	✓	✓

- **private**：表示当前类私有的，类访问权限，只能在本类中操作，离开本类之后就不能直接访问
- 不写（缺省）：表示当前包私有，包访问权限，定义和调用只能在同一个包中，才能访问
- **protected**：表示子类访问权限，同包中的可以访问，即使不同包但是有继承关系也可以访问
- **public**：表示公共的，可以在当前项目中任何地方访问

private修饰符演示：



缺省和public修饰符演示：



暂时记住：把所有的字段使用private修饰，所有方法使用public修饰。

## 2.2 封装使用（了解）

使用private修饰了Student类中的字段，此时在测试类中访问报错。

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

测试类：

```
public class StudentDemo {  
    public static void main(String[] args) {  
        Student stu = new Student();  
        stu.name = "小明";    // 此行报错，访问权限不足  
        stu.age = -12;        // 此行报错，访问权限不足  
    }  
}
```

此时使用private修饰字段后，在测试类中不能再操作这些字段了，怎么办？我们可以使用JavaBean的规范来解决。

## 2.3 JavaBean规范（重点掌握）

JavaBean是一种某些符合条件的特殊类，但是必须遵循一定的规范：

- 类必须使用public修饰
- 必须保证有公共无参数构造器。即使手动提供了带参数的构造器，也得手动提供无参数构造器
- 字段使用private修饰(私有化)
- 每个字段提供一对getter和setter方法

需求：针对名为name的字段名来举例

**getter方法：**仅仅用于返回某一个字段的值。getter方法约定命名方式：**get+字段首字母大写**。

```
public String getName(){  
    return name;    //返回name字段存储的值  
}
```

如果操作的字段是boolean类型的，此时是is方法，把 getName 变成 isName。

**setter方法：**仅仅用来给某一个字段设置值，setter方法约定命名方式：**set+字段首字母大写**。实际开发过程中，可以融入校验逻辑。

```
public void setName(String aName){
    name = aName;    //把传过来的参数aName的值,存储到name字段中
}
```

注意：每一个字段都得使用private修饰，并提供一对getter/setter方法。

IDEA工具可以自动生成标准的getter/setter，前期必须手写。

代码如下：

```
public class Student {
    private String name;
    private int age;
    private boolean graduate    // 是否毕业

    public String getName() {
        return name;
    }
    public void setName(String aName) {
        name = aName;
    }

    public int getAge() {
        return age;
    }
    public void setAge(int aAge) {
        if(aAge < 0) {
            System.out.println("非法的年龄");
            age = 0;
        } else {
            age = aAge;
        }
    }

    // 如果字段是boolean类型，getter方法可以是isGraduate或getGraduate，从语义上讲，更推荐isGraduate
    public boolean isGraduate() {
        return graduate;
    }

    public void setGraduate(boolean aGraduate) {
        graduate = aGraduate;
    }

    // 其他代码
}
```

测试类：

```
public class StudentDemo {
    public static void main(String[] args) {
        Student stu = new Student();
    }
}
```



```
// 调用setter方法设置字段数据
stu.setName("小明");
stu.setAge(12);

// 调用getter方法获取字段数据
String name = stu.getName();
int age = stu.getAge();

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