# 尚马教育 JAVA 基础课程

# Java面向对象--封装

文档编号：A06

创建日期： 2017-04-12

最后修改日期：2019-09-23

版 本 号：V3.0

电子版文件名：尚马教育-第一阶段-5.封装专题课程.docx

**文档修改记录：**

|  |  |  |  |
| --- | --- | --- | --- |
| 更新日期 | 更新作者 | 更新说明 | 版本号 |
| 2017-07-30 | 张元林 | 初始版本 | V1.0 |
| 2018-08-01 | 王绍成 | Java基础版本更新 | V2.0 |
| 2019-08-09 | 徐丽莎 | Java基础版本更新 | V3.0 |

**主讲人：**

**徐丽莎**

目录

[尚马教育 JAVA 基础课程 1](#_Toc16526)

[Java面向对象--封装 1](#_Toc1593)

[1. 面向对象 2](#_Toc7416)

[1.1. 三大特征 2](#_Toc27304)

[2. 封装 3](#_Toc12350)

[2.1. 封装的意义 3](#_Toc26173)

[2.2. 封装的实现 3](#_Toc14315)

[2.3. Lombok 6](#_Toc4109)

[2.3.1. Lombok与eclipse集成 6](#_Toc26518)

[2.3.2. Lombok使用 7](#_Toc10975)

[2.3.3. 案例 8](#_Toc1607)

[2.3.4. 优缺点 8](#_Toc15220)

[2.4. 课堂练习 9](#_Toc24624)

[3. this 9](#_Toc5600)

[3.1. this使用场景 10](#_Toc30489)

[4. static 11](#_Toc17093)

[4.1. 静态代码块 11](#_Toc28454)

[4.2. 注意 12](#_Toc25473)

[4.3. 案例 12](#_Toc25320)

[5. 作业 13](#_Toc13436)

## 面向对象

* 重点：不需要亲手去实现一段代码，调用其它类中的方法即可。
* 生活中案例：
  + 想修空调，不需要学习如何修空调，找一个专业的空调维修人员，调用他身上的修空调方法，传入的参数即是你付出的代价，返回值即是你想要的结果。
  + 想听音乐，不需要自己去唱，找到播放器类，调用播放方法，传入参数歌曲名，返回结果音乐。
  + 想吃饭，不需要去学习做饭，找女朋友类，调用女朋友身上做饭的方法，返回结果可口的饭。

### 三大特征

* + 封装

解决了数据的安全性问题。

* + 继承

解决了代码的重用问题

* + 多态

解决了程序的扩展问题

## 封装

* 把客观事物封装成抽象的类，并且类可以把自己的数据和方法只让可信的类或者对象操作，对不可信的进行信息隐藏。在一个对象内部，某些代码或某些数据可以是私有的，不能被外界访问。通过这种方式，现实世界可以被描绘成一系列完全自治、封装的对象，这些对象通过一个受保护的接口访问其他对象

### 封装的意义

* 封装是一种信息隐藏技术，在java中通过控制成员的访问权限实现封装，即使用方法将类的数据隐藏起来，控制用户对类的修改和访问数据的程度。 适当的封装可以让代码更容易理解和维护，也加强了代码的安全性。

### 封装的实现

* 编写一个类，代表教员，要求：
  + 具有属性：姓名、年龄
  + 具有行为：自我介绍
  + 教员的最小年龄要求：22岁

|  |
| --- |
| **public** **class** Teacher {  **public** String name; // 教员姓名  **public** **int** age; // 年龄  /\*\*  \* 返回自我介绍的内容  \*/  **public** String introduction() {  **return** "大家好！我是" + name + "，我今年" + age + "岁";  }  } |

* 编写一个测试类，要求：
  + 实例化一个教员对象，并对其初始化
  + 在控制台输出该教员的自我介绍

|  |
| --- |
| **public** **class** TeacherTest {  **public** **static** **void** main(String[] args) {  Teacher teacher = **new** Teacher();  teacher.name = "李雷";  teacher.age = 10;  System.***out***.println(teacher.introduction());  }  } |
|  |
| 但是并没有满足对年龄的数据需求: |

* 再次定义封装就是:
  + 封装,就是隐藏实现细节
  + 将属性私有化，提供公有方法访问私有属性
  + 通过这些公有方法访问私有属性

|  |
| --- |
|  |

* 封装的实现流程：
  + 修改属性的可见性来限制对属性的访问
  + 为每个属性创建一对赋值(setter)方法和取值(getter) 方法，用于对这些属性的访问
  + 在setter和getter方法中，加入对属性的存取限制

|  |
| --- |
|  |

* 测试类通过调用setter方法，为对象的各个属性赋值

|  |
| --- |
|  |

### Lombok

#### Lombok与IDEA集成

注意:

* 参照阿里开发规范，我们发现在set以及get方法中，其实我们是不推荐写逻辑功能。

|  |
| --- |
|  |

* 因此在开发中，经常会使用一些开发利器帮助我们提高开发效率。
* 由于lombok属于第三方的插件，需要我们下载有关lombok的类库文件，lombok.jar
* 1.下载路径: <https://projectlombok.org/download> 目前最先版本是1.18.10
* 2.下载完毕之后，在项目中，新建lib目录，将类库文件存储在文件夹中
* 3.将类库文件放到编译的路径下。 add as library
* 4.将lombok与IDEA进行集成
  + 1. 在setting里面安装lombok的插件，然后重启。(或者下载离线也可)
  + 2. 开启注解处理器。

#### Lombok使用

* 在开发中，我们是利用lombok提供的注解进行生成set以及get方法。
* 何时生成? 代码编译的时候生成。
* 这也是为什么要与IDEA集成的原因，因为编译源代码的工作IDEA帮我们实现了。
* Lombok插件给我们提供了很多常用的注解
* 这些注解只对非static和非瞬态transient修饰的成员变量有效
  + @Setter
  + @Getter
  + @ToString
  + @AllArgsConstructor
  + @NonArgsConstructor
  + @EqualsAndHashCode
  + @Data == @Setter+@Getter+@ToString+@EqualsAndHashCode
  + @NonNull 非空的声明，可用于校验参数，能帮助避免空指针
  + @Cleanup 帮助我们自动调用close()方法，很大的简化了代码(主要与IO有关)
  + .......

#### 案例

#### 优缺点

* 优点:
  + 能通过注解的形式自动生成构造器、getter/setter、equals、hashcode、toString等方法，提高了一定的开发效率
  + 让代码变得简洁，不用过多的去关注相应的方法
  + 属性做修改时，也简化了维护为这些属性所生成的getter/setter方法等
* 缺点:
  + 不支持多种参数构造器的重载
  + 虽然省去了手动创建getter/setter方法的麻烦，但大大降低了源代码的可读性和完整性，降低了阅读源代码的舒适度
* 总之虽然非常不建议在属性的getter/setter写一些业务代码，但在多年项目的实战中，有时通过给getter/setter加一点点业务代码，能极大的简化某些业务场景的代码。所谓取舍，也许就是这时的舍弃一定的规范，取得极大的方便。
* Lombok有它的得天独厚的优点，也有它避之不及的缺点，熟知其优缺点，在实战中灵活运用才是王道。

### 课堂练习

* 请通过代码封装，实现如下需求：
* 编写一个类Book，代表教材:
  + 具有属性：名称（title）、页数（pageNum），其中页数不能少于200页，否则输出错误信息，并赋予默认值200，
  + 为各属性设置赋值和取值方法
  + 具有方法:detail，用来在控制台输出每本教材的名称和页数
* 编写测试类BookTest进行测试：为Book对象的属性赋予初始值，并调用Book对象的detail方法，看看输出是否正确

## this

* 对象在内存中的存储形式:

|  |
| --- |
|  |

* 每个对象都有自己独立的数据成员，但是所有的对象共享成员方法。
* 既然所有的对象都共用相同的成员方法，那么在不同的对象都调用同一方法时，它是怎么确定要使用哪个对象的数据成员呢？
* 每个成员方法都有一个隐含的this引用，它总是指向调用它的对象；
* 关键字this给出用于调用成员方法的对象的地址；
* 每当调用成员方法时，编译器会向this分配调用该方法的对象的地址；
* 可以像使用任何引用那样使用this
* 可以很简单的认为this就是当前正在运行的对象。

|  |
| --- |
| // 定义学生类  @Setter  @Getter  **public** **class** Student{  **private** String mName; // 姓名  **private** **int** mAge; // 年龄  **public** Student(String name, **int** age) {  // 隐式调用，等同于this.mName = name;  mName = name;  // 显式调用，等同于mAge = age;  **this**.mAge = age;  }  } |

### this使用场景

* 当成员变量与局部变量重名的时候 我们可以使用this进行区分
* this访问类的成员
* this可以充当实际参数
* this调用构造方法(再构造方法) this(参数); 代码的第一行
* this不能再static所修饰的方法运用

## static

* 静态的，修饰符/关键字
* 成员变量前加static关键字，可以将其声明为静态成员变量或者类变量；
* 如果类中成员变量被定义为静态，那么不论有多少个对象，静态成员变量只有一份内存拷贝，即所有对象共享该成员变量；
* 静态成员变量的作用域只在类内部，但其生命周期却贯穿整个程序
* 可以修饰成员变量，成员方法，也可以修饰代码块(静态代码块)
* 加入变量数据要让所有对象共享，那就必须声明成静态变量。

|  |
| --- |
| **class** Dog {  **public** **static** **int** *count* = 0; // 静态成员变量  **public** Dog(){  *count*++;  }  }  **public** **class** Test {  **public** **static** **void** main(String[] args) {  System.***out***.println("当前狗的数量是：" + Dog.*count*);  Dog d1 = **new** Dog();  Dog d2 = **new** Dog();  System.***out***.println("当前狗的数量是：" + Dog.*count*);  }  } |

### 静态代码块

* 与构造方法的功能相似，主要对成员变量做初始化数据的操作。
* 但是有且只走一次
* 执行级别比构造方法还要高

### 注意

* 静态成员变量
  + 和程序同生命周期；
  + 在没有实例化对象时，可以通过类名访问静态成员变量；
  + 也可以通过对象访问静态成员变量，但不论使用的是哪个对象，访问到的都是同一个变量；
  + 静态成员变量在声明时最好初始化，如果不进行初始化，系统会默认初始化为初始值。
* 静态成员方法
  + 在成员方法前加static关键字，可以将其声明为静态成员方法；
  + 静态成员方法只能对类的静态成员变量进行操作；
  + 静态成员方法没有this引用；
  + 在没有实例化对象时，可以通过类名访问静态成员方法。
* 静态成员属于整个类而不是属于某个对象，它被该类的所有对象共享；
* 访问静态成员时一般通过类名访问，也可以通过对象访问；
* 静态成员也受访问权限的约束；
* 静态数据成员在使用之前应尽量初始化；
* 静态成员方法不能操作非静态成员；
* 静态成员方法没有this引用。

### 案例

* 给定如下Java构造函数代码，请指出代码中的错误，并解释原因

|  |
| --- |
| **public** **class** Sample {  **private** **int** x;  **public** Sample() {  x = 1;  }  **public** Sample(**int** i) {  x = i;  }  **public** **int** Sample(**int** i) {  x = i;  **return** x++;  }  **private** Sample(**int** i, String s) {  }  **public** Sample(String s, **int** i) {  }  **private** Sampla(**int** i){  x=i++;  }  **private** **void** Sampla(**int** i) {  x = i++;  }  } |

## 作业

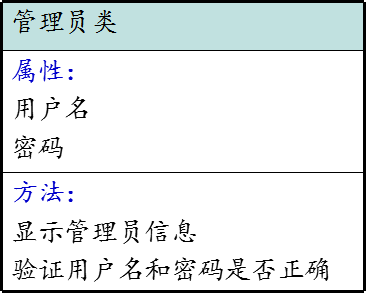
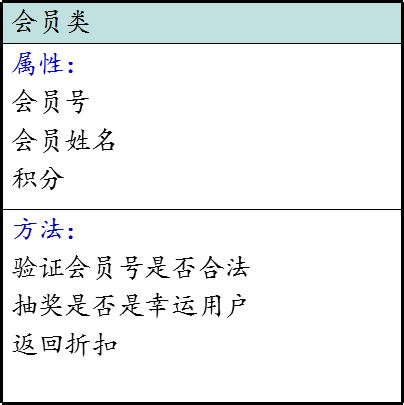
* 作业1
  + 创建一个程序员类和项目经理类
  + 封装两个类的所有属性，生成getter和setter方法；
  + 为两个类分别编写两个构造函数，一个无参构造函数要求为所有属性设置默认值，另一个是带所有属性参数的构造函数。
  + 编写测试类，分别创建两个程序员和两个项目经理，并调用show（）方法显示详细信息。

|  |  |
| --- | --- |
|  |  |

* + 编写计算工资类：
  + 添加两个计算程序员和计算项目经理工资的方法，实现方法的重载，分别输出应付多少工资；
  + 编写测试类：
  + 分别创建一个程序员和一个项目经理对象，再创建一个计算工资对象，调用计算程序员和项目经理工资的方法。

|  |
| --- |
|  |

* 作业2:
  + 使用java描述一个矩形，矩形具备高、宽两个属性，还具备计算面积与周长的方法。
  + 要求：一旦创建矩形对象时，那么该矩形对象就具备了高、宽两个属性值（不能使默认的初始值）
* 作业3
  + 定义一个人类，id ，name , age 三个属性， 还具备一个自己与他人比较年龄的方法。
  + 类中有属性，可以统计创建人类对象的次数
* 综合运用方法实现
* 作业4：
  + 描述一个计算器类 , 计算机有 操作数1 ， 操作数2 ， 运算符 三个属性，还具备运算的功能。
  + 要求：不能直接给 操作数1 操作数2，运算符直接赋值
* 作业5：
  + 定义一个管理员类：
  + 显示管理员信息的方法：返回管理员用户名和密码的信息；
  + 验证用户密码的方法：传入用户名和密码的参数，返回布尔类型，用户名和密码是否正确。
  + 编写一个测试类，创建一个管理员类，初始化设置用户名和密码，控制台接收用户输入的用户名和密码，调用验证的方法，显示是否登录成功。

* 作业6：
  + 编写会员类：
  + 编写测试类：
  + 接收控制台用户输入会员号、会员姓名、积分，创建一个会员对象，验证会员号是否合法，不合法提示退出，合法保存会员数据，同时显示当前用户的(积分)折扣信息，最后进行抽奖，提示是否是(会员号)幸运用户。
* 作业7：
  + 描述音乐类，具有属性：音乐名、播放时长、歌手
  + 描述音乐播放器类，具有属性：播放器名字、播放音乐方法。
  + 控制台录入音乐名、播放时长、歌手名
  + 根据信息创建音乐对象
  + 将音乐对象传入播放器的播放音乐方法
  + 在播放音乐方法中输出音乐信息
  + 正在播放xx歌唱的音乐【xxx】剩余时间xxx