# 尚马教育 JAVA 基础课程

# Java面向对象--多态

文档编号：A08

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-8.多态专题课程.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

[1. 面向对象](#_Toc3290_WPSOffice_Level1) [2](#_Toc3290_WPSOffice_Level1)

[1.1. 三大特征](#_Toc12124_WPSOffice_Level2) [2](#_Toc12124_WPSOffice_Level2)

[2. 多态](#_Toc12124_WPSOffice_Level1) [2](#_Toc12124_WPSOffice_Level1)

[2.1. 多态的定义](#_Toc27358_WPSOffice_Level2) [3](#_Toc27358_WPSOffice_Level2)

[3. 多态的实现(继承关系下)](#_Toc27358_WPSOffice_Level1) [3](#_Toc27358_WPSOffice_Level1)

[3.1.1. 案例](#_Toc31193_WPSOffice_Level2) [3](#_Toc31193_WPSOffice_Level2)

[3.1.2. 多态体现](#_Toc22929_WPSOffice_Level2) [6](#_Toc22929_WPSOffice_Level2)

[3.1.3. 向下转型](#_Toc24801_WPSOffice_Level2) [6](#_Toc24801_WPSOffice_Level2)

[3.1.4. 课堂练习](#_Toc12859_WPSOffice_Level2) [8](#_Toc12859_WPSOffice_Level2)

[4. 抽象方法](#_Toc31193_WPSOffice_Level1) [8](#_Toc31193_WPSOffice_Level1)

[4.1. 案例](#_Toc14396_WPSOffice_Level2) [9](#_Toc14396_WPSOffice_Level2)

[5. 抽象类](#_Toc22929_WPSOffice_Level1) [9](#_Toc22929_WPSOffice_Level1)

[5.1. 案例](#_Toc26686_WPSOffice_Level2) [9](#_Toc26686_WPSOffice_Level2)

[5.2. 特征](#_Toc16793_WPSOffice_Level2) [12](#_Toc16793_WPSOffice_Level2)

[6. 作业](#_Toc24801_WPSOffice_Level1) [12](#_Toc24801_WPSOffice_Level1)

## 面向对象

### 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

## 多态

* 现在我们已经学会了继承这一类与类之间的关系，并且能够在子类继承父类的基础上进一步对子类的数据及操作进行扩展，增加新的成员变量和方法或者对父类的方法进行重写覆盖。不过重写了父类方法之后，我们仍然按照传统的方法构建子类对象后对方法进行调用，看起来继承仅仅是帮助我们省去了一部分变量、方法声明的代码编写而没有其他特殊的作用，但是如果只有这一点好处，继承也不能被称为是面向对象编程语言的一大基石了，事实上他还作为了其他语言特性的基础支持，例如接下来我们要探讨的更为重要的多态性，那么多态性究竟是如何定义，如何实现的呢，通过本节的学习后我们将对其有较为深入的认知。

### 多态的定义

* Java中多态性指允许不同类的对象对同一消息做出响应。即同一消息可以根据发送对象的不同而采用多种不同的行为方式。
* 发送消息就是方法调用。
* 现实中，关于多态的例子不胜枚举。比方说按下 F1 键这个动作，如果当前在IE界面下弹出的浏览器的帮助文档；如果当前在 Word 下弹出的就是 office帮助；在 Windows 下弹出的就是 Windows 帮助和支持
* 可见，同一个事件发生在不同的对象上会产生不同的结果，因此，**多态的主要作用适用于消除类型之间的耦合关系，提高程序的可扩展性。**
* **简单说多态是具有表现多种形态的能力的特征**

## 多态的实现(继承关系下)

* 前提:**要有层级关系**。(在继承的关系下，或者是在接口的实现关系下)

#### 案例

* 在**继承**的章节中，我们为了提取冗余代码
  + 创建员工基类，
  + 创建程序员和项目经理子类

|  |
| --- |
|  |

* 现在需求增加，我们需要对程序员SE以及项目经理PM进行发工资。
* 因此，我们创建了一个CalSalary.java的类,可以计算SE以及PM的工资

|  |
| --- |
| **public** **class** CalSalary {  // 1.将构造变成私有  **private** CalSalary() {  }  /\*\*  \* 利用方法重载实现对SE以及PM发放工资  \* **@param** pm  \* **@return**  \*/  **public** **static** **double** calSal(PM pm) {  **return** pm.getSalary() + pm.getBonus();  }  **public** **static** **double** calSal(SE se) {  **return** se.getSalary();  }  } |

* 但是在一个公司里面，公司的职位不仅有SE和PM，还有Sale，Manager等。
* 如果想要对这些员工进行发送工资的话，需要在CalSalary.java再新增n个方法来满足对Sale以及Manager工资的发放。
* 方法重载能够实现我们的需求，对于我们发工资的程序，可扩展性以及可维护性极差。
* 因此，为了避免出现以上弊端，我们可以使**用多态进行处理**。
* 因为对于SE,PM,Sale,Manager这些类，有一个共同的特点就是: 父类都是Employee
* 因此，在CalSalary.java类里面，我们就可以改成以下:

|  |
| --- |
| **public** **class** CalSalary {  // 1.将构造变成私有  **private** CalSalary() {  }  /\*\*  \* 利用多态实现对SE以及PM发放工资  \* **@param** pm  \* **@return**  \*/  **public** **static** **double** calSal(Employee employee) {  **return** employee.getSalary();  }  } |
| **public** **class** EmployeeTest {  **public** **static** **void** main(String[] args) {  // 对项目经理 和程序员进行工资的发放  PM pm = **new** PM(1, "tom", "男", 60000, 4, 2000);  SE se = **new** SE(2, "jim", "男", 10000, 4);  Sale sale = **new** Sale(3, "jucy", "男", 30000);  System.***out***.println("pm:" + CalSalary.*calSal*(pm));  System.***out***.println("se:" + CalSalary.*calSal*(se));  System.***out***.println("sale:" + CalSalary.*calSal*(sale));  }  } |
|  |

#### 多态体现

* 以上案例实现了对所有员工进行工资发放，出现了多态。
* 将pm，se，sale对象作为实际参数传到CalSalary.*calSal(Employee employee);*方法中
* 这时候*employee*有多种形态，既可以是se，pm，也可以使sale对象
* 因此得出: 父类对象有多种形态的能力，父类对象可以是任意一个子类的实例。
* 发生多态，我们也称为**子类对象向上转型**，或者说**父类引用指向任意子类对象**。或者认为**编译时数据类型与运行时数据类型不一致**

|  |
| --- |
|  |

* 引用变量在编译阶段只能调用其编译时类型所具有的方法，但运行时则执行它运行时类型所具有的方法
* 因此，编写Java代码时，引用只能调用编译器类型里包含的成员
* 通过**Employee emp =new PM()代码定义一个变量emp ，则这个emp 只能调用Employee 类的成员，而不能调用PM类里定义的成员**。

#### 向下转型

* 虽然上述案例能够实现对所有员工发工资，但是对于PM而言，薪资发放的有问题?没有发放奖金?
* 这也可以认为是多态的弊端，只能访问父类成员，子类独有的成员无法访问。
* 因此，我们需要**向下转型**: 将父类实例转化成子类对象

|  |
| --- |
| **public** **static** **double** calSal(Employee employee) {  **double** salary = employee.getSalary();  PM pm = (PM)employee;//父类对象向下强制转换成子类实例  salary+=pm.getBonus();  **return** salary;  } |
|  |

* 以上代码能够实现对pm的薪资发放，但如果是其他员工，代码出现了问题:

|  |
| --- |
|  |

* 出现了类型转换的异常，因为并不是所有的员工都是pm，如果是se的话，就不能强制转换。
* 所以，需要加条件判断，如果父类引用是pm对象的时候，才进行转换。

|  |
| --- |
| **public** **static** **double** calSal(Employee employee) {  **double** salary = employee.getSalary();  **if**(employee **instanceof** PM) {  PM pm = (PM)employee;//父类对象向下强制转换成子类实例  salary+=pm.getBonus();  }  **return** salary;  } |

* Instanceof 判断一个对象是否是一个类的实例。这样就可以避免出现类型转换的异常了。
* 可以使用一个简单案例解释:人类是父类，男人女人是子类，一个人是男人，一定也是人，所以向上转型不需要强制，但是一个人是人，但不一定是男人，所以需要强制转

|  |
| --- |
| **public** **class** EmployeeTest {  **public** **static** **void** main(String[] args) {  // 对项目经理 和程序员进行工资的发放  // 父类对象执行任何一个子类的实例(多态)  Employee employee = **new** PM(1, "tom", "男", 60000, 4, 2000);  System.***out***.println("pm:" + CalSalary.*calSal*(employee));  employee = **new** SE(2, "jim", "男", 10000, 4);  System.***out***.println("pm:" + CalSalary.*calSal*(employee));  employee = **new** Sale(3, "jucy", "男", 30000);  System.***out***.println("sale:" + CalSalary.*calSal*(employee));  }  } |

#### 课堂练习

* 使用多态实现以下需求:
* 创建教员基类，在教员基类的基础上，再创建两个子类：JAVA教员和WEB教员，
* 开发一个类代表教学管理中心，负责对各教员进行评估，评估内容包括：
  + 教员的自我介绍
  + 教员的授课
* 现在，升级教员类，增加一种新类型：DBTeacher，该类型的教员专门负责教授数据库，要求如下：
  + 属性：姓名、年龄、教学年限
  + 方法：授课（启动Mysql、实施理论课授课）、自我介绍
  + 同时，教学管理中心也负责对这类教员进行评估

## 抽象方法

* 在某些情况下，基类无法（或者没有必要）提供被覆盖方法的具体实现，那么就可以将此方法声明成抽象方法；(我们上节讲的老师授课案例)
* 使用关键字abstract声明抽象方法，一般语法：

|  |
| --- |
|  |

### 案例

|  |
| --- |
| //定义/声明方法  **public** **abstract** **void** introduce();  **public** **abstract** **void** teach() ; |

## 抽象类

* 如果某个类中包含有抽象方法，那么该类就必须定义成抽象类；
* 定义抽象类同样使用关键字abstract，一般语法：

|  |
| --- |
|  |

* 简单的理解，Java的抽象就是只声明行为接口（方法签名）而不完成具体的实现。

### 案例

|  |
| --- |
| **public** **abstract** **class** Teacher {  **private** String name;  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  //规范  //定义/声明方法  **public** **abstract** **void** introduce();  **public** **abstract** **void** teach() ;  **public** Teacher(String name) {  **super**();  **this**.name = name;  }  } |
| **public** **class** JavaTeacher **extends** Teacher {  **public** JavaTeacher(String name) {  **super**(name);  }  **public** **int** num = 100;  //子类重写父类里面定义的抽象方法  @Override  **public** **void** introduce() {  System.***out***.println("JavaTeacher在自我介绍");  }  @Override  **public** **void** teach() {  System.***out***.println("JavaTeacher在授课");  }  } |
| **public** **class** WebTeacher **extends** Teacher {  **public** WebTeacher(String name) {  **super**(name);  }  @Override  **public** **void** introduce() {  System.***out***.println("web老师在自我介绍");  }  @Override  **public** **void** teach() {  System.***out***.println(getName() + "在教授web");  }  } |
| **public** **class** TeacherManager {  **public** **static** **void** evaluate(Teacher teacher) {  System.***out***.println("评估开始...........");  //获得每位老师的授课和自我介绍  // 1、教员的自我介绍  teacher.introduce();  // 2、教员的授课  teacher.teach();  **if**(teacher **instanceof** JavaTeacher) {  JavaTeacher javaTeacher = (JavaTeacher)teacher;  System.***out***.println(javaTeacher.num);  }  System.***out***.println("评估结束...........");  }  **public** **static** **void** main(String[] args) {  //创建老师的对象  JavaTeacher javaTeacher = **new** JavaTeacher("javaTeacher");  WebTeacher webTeacher = **new** WebTeacher("webTeacher");  *evaluate*(javaTeacher);  System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  *evaluate*(webTeacher);  }  } |

### 特征

* 利用抽象的概念，能够在开发项目中创建扩展性很好的架构，优化程序
* 抽象类，抽象方法，在软件开发过程中都是设计层面的概念。也就是说，设计人员会设计出抽象类，抽象方法，程序员都是来继承这些抽象类并覆盖抽象方法，实现具体功能
* 抽象类是抽象方法的容器，如果某个类中包含有抽象方法，那么该类就必须定义成抽象类。抽象类中也可以包含有非抽象的方法甚至抽象类中可以没有抽象方法（只要一个类不应该有具体对象，就应该是抽象类，不一定是有抽象方法的）
* 抽象类不可以直接实例化，只可以用来继承作为其他类的父类存在
* 抽象类的派生子类应该提供对其所有抽象方法的具体实现
* 如果抽象类的派生子类没有实现其中的所有抽象方法，那么该派生子类仍然是抽象类，只能用于继承，而不能实例化，但可以有构造函数（用于帮助子类快速初始化共有属性）

## 作业

* 结合多态，实现以下需求:
* 1.乐器（Instrument）分为：钢琴(Piano)、小提琴(Violin)，各种乐器的弹奏（ play ）方法各不相同。
  + 编写一个测试类InstrumentTest，要求：
  + 编写方法testPlay，对各种乐器进行弹奏测试。要依据乐器的不同，进行相应的弹奏。
  + 在main方法中进行测试
* 2.描述车、宝马、奔驰 三个类，车都具备颜色、名字、轮胎数量，还具备跑的功能
  + 司机类，具有一个开车方法，可以开不同的车
* 3.动物、 蛇、 鱼 、三个类，编写一个方法可以接收任意的动物对象，定义一个方法可以接收任意的动物类型对象，并且要调用动物类型对象的特有方法。
* 4.需求：为地下城勇士游戏编写怪物模块的功能
  + 现怪物的种类有两种：哥布林、猫妖
  + 怪物的共有的属性有：名称、血量、攻击力、防御力
  + 哥布林特有的属性有：狂暴度，
  + 哥布林具有的方法有：攻击（使用石头进行攻击）和移动(缓慢移动)
  + 猫妖具有的方法有：攻击（使用爪子攻击）和移动（跳着移动)
  + 现需要在测试类中进行测试两只哥布林和两只猫妖的攻击和移动方法，要求将创建的怪物对象存放在数组对象中进行统一管理，循环调用每只怪物的攻击和移动的方法。
  + 要求使用多态的思想实现怪物的攻击和移动。
* 5.需求：为画图软件编写图案绘制模块的功能
  + 画图软件可以的绘制图案有：线条、矩形、圆形
  + 各个图案共有属性有：坐标X轴、坐标Y轴，图案颜色
  + 共有的方法有：绘制（抽象方法）
  + 线条独有的属性有：终点X轴坐标，终点Y轴坐标
  + 矩形独有的属性有：长度，宽度
  + 圆形独有的属性有：半径
  + 现需要编写测试类进行测试各个图案的绘制功能是否正常，为了便于管理，需要使用数组对象管理这些图案对象，每种图案各创建一个对象保存到数组中，循环调用绘制的方法。
  + 要求使用抽象方法实现多态。