# 尚马教育 JAVA 基础课程

# Java面向对象--继承

文档编号：A07

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-7.继承专题课程.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](#_Toc18968)

[Java面向对象--继承 1](#_Toc27798)

[1. 面向对象 2](#_Toc12028)

[1.1. 三大特征 2](#_Toc15130)

[2. 继承(extends) 2](#_Toc27133)

[2.1. 继承的意义 3](#_Toc15378)

[2.1.1. 代码冗余问题 3](#_Toc24648)

[2.1.2. 提取冗余代码 4](#_Toc16435)

[2.1.3. 生活中的继承 4](#_Toc9980)

[2.2. 继承的实现 5](#_Toc23161)

[2.3. 子类对象初始化流程 6](#_Toc22058)

[2.4. Super 8](#_Toc30603)

[2.4.1. Super的作用 8](#_Toc4411)

[2.4.2. 调用父类构造 8](#_Toc22823)

[2.4.3. 调用父类成员 9](#_Toc22298)

[2.4.4. 案例 9](#_Toc24074)

[2.5. 访问权限修饰符 10](#_Toc273)

[2.6. 方法重写(override) 10](#_Toc490)

[3. 作业 12](#_Toc19778)

## 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

## 继承(extends)

* 可以看做是描述了一种代码复用的高效方法。

### 继承的意义

* 继承是能自动传播代码和重用代码的有力工具；
* 继承能够在某些比较一般的类的基础上建造、建立和扩充新类；
* 通过使用继承我们能够非常方便地复用以前的代码，能够大大的提高开发的效率
* 引入继承，实现了递增式的程序设计，并通过增强一致性来减少模块间的接口和界面，从而增强了程序的可维护性
* 能清晰地体现出类与类之间的层次结构关系
* 注意:
  + 继承是单方向的，即派生类可以继承和访问基类中的成员，但基类则无法访问派生类中的成员
  + 在Java中只允许单一继承方式，即一个派生类只能继承于一个基类，而不能象C++中派生类继承于多个基类的多重继承方式

#### 代码冗余问题

* 对比下面这两个类：程序员类和项目经理类；
* 如果再加入一个CEO类或者Sale类呢，是否也有相同的属性？
* 结果是: 代码冗余度太高。

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

#### 提取冗余代码

|  |
| --- |
|  |

#### 生活中的继承

|  |
| --- |
|  |
|  |

### 继承的实现

* 在 java 中，类可以继承自另一个类。衍生的类（子类）继承父类的方法和数据成员
* 子类继承父类，父类派生子类
  + 父类又叫基类
  + 子类又叫派生类
* 继承就是子类继承父类的特征和行为，使得子类对象（实例）具有父类的实例域和方法，或子类从父类继承方法，使得子类具有父类相同的行为

|  |
| --- |
|  |

* 最高层是最普遍的、最一般的情况，往下每一层都比上一层更具体，并包含有高层的特征，通过这样的层次结构使下层的类能自动享用上层类的特点和性质
* 继承其实就是自动地共享基类中成员属性和成员方法的机制.(不包含父类private成员以及父类构造)
* 语法:

|  |
| --- |
| **【访问权限修饰符】【修饰符】子类名 extends 父类名{子类体}** |

* 案例
  + 创建员工基类，
  + 创建程序员和项目经理子类

|  |
| --- |
|  |

### 子类对象初始化流程

|  |
| --- |
| @Getter  @Setter  **public** **class** Employee {  **private** **int** id;// 员工的编号  **private** String name;  **private** String gender;  **private** **double** salary;  **public** **void** show() {  System.***out***.println("员工的信息:" + name + "," + salary + "," + gender);  }  **public** Employee() {  System.***out***.println("Employee父类的构造");}} |
| @Setter  @Getter  **public** **class** SE **extends** Employee {  **private** **int** hot;  **public** SE() {  System.***out***.println("SE子类的构造");  }  } |
| **public** **static** **void** main(String[] args) {  //创建一个SE的对象  SE se = **new** SE();  System.***out***.println(se);  se.setId(1001);  se.setName("jim");  se.setSalary(6000);  se.setHot(3);  se.setGender("男");  se.show();  } |
| 控制台输出内容: |

* 在创建子类对象过程中，优先执行了父类的构造方法。
* 所以一定要确保父类构造里面没有任何问题，否则子类对象是无法创建成功的。
* 子类会默认调用父类的无参构造方法，也就是说子类必须能够访问父类的一个构造方法，并且一定会调用。
* 因为在子类的构造逻辑的第一行中，其实是**使用super默认调用了父类无参构造。**

### Super

#### Super的作用

* super并不表示一个指向对象的引用，它只是一个特殊的关键字，用来告诉编译器，现在要调用的是父类的方法。
* super关键字也有两种意义：
  + 调用父类的成员(但是不能是private修饰)
  + 调用父类的构造方法

#### 调用父类构造

* 在子类对象创建流程中会优先执行父类无参构造，如果父类不具备无参的构造方法怎么办呢？那在子类的构造里面肯定会出现问题(因为子类构造默认调用父类无参构造)
* 如果没有无参的父类构造方法，子类必须要显示的调用父类的构造方法，而且必须是在子类构造器中做的第一件事
  + 通过super关键字可以在子类构造方法中显式调用父类的构造方法，该调用必须位于**子类构造**方法**的第一行**
* 利用super显式调用父类构造方法：

|  |
| --- |
|  |

#### 调用父类成员

* 如果父类和子类中有同名成员，在子类中默认访问是属于自己的那一个成员；
* super关键字可以明确地指定要访问父类中的成员
* 其一般语法是： **super.成员名;**

|  |
| --- |
|  |

#### 案例

* 给定如下Java代码，编译运行后，输出结果是什么？

|  |
| --- |
|  |

### 访问权限修饰符

* 信息隐藏是面向对象程序设计的重要特点之一，它可以：防止类的使用者意外损坏数据；对任何实现细节所作的修改不会影响到使用该类的其它代码；使类更易于使用。
* 在Java中实现信息隐藏的是访问控制权限机制；
* 访问控制权限包括4个访问修饰符：public、protected、private和缺省；
* 可以使用上述访问修饰符修饰类的成员。

|  |
| --- |
|  |
|  |

* 因此，对于继承而言，父类里面的成员如果只是想让子类拥有，可以使用protected进行修饰。

### 方法重写(override)

* 前提:

有层级关系。(有继承关系)

* 子类总是希望使用父类提供的数据和行为吗？不一定。

|  |
| --- |
|  |

* 在上面这个案例中，我们发现所有的动物都会move，但是每个具体的动物move的形式却不一样。
* 在父类Bird类里面有一个move方法，但是每个子类的move的功能都不一样。
* 也就是说，子类希望修改父类的方法的方法体，可以怎么做呢？
* 第一种做法是子类创建一个不同名字的新方法，实现新的逻辑，然而，这种做法会导致子类依然包含父类中的那个方法，却不应该使用，破坏封装性
* 我们希望子类中的方法依然和父类方法的声明形式一样，但是具体方法体却不同，这种做法就叫做方法重写
* 方法重写，必须遵循以下规则:
  + **子类方法名，形式参数，必须与父类保持一致**
  + **子类方法访问权限修饰符>=父类访问权限修饰符**
  + **子类返回值类型<=父类返回值类型**
* 上面的鸟类示例中，企鹅和鸵鸟均需要覆盖鸟类的move方法：

|  |
| --- |
|  |

* 如果在子类覆盖的方法里或其他地方需要明确使用父类声明的方法版本，也可以使用之前遇到的super关键字显示调用

|  |
| --- |
|  |

## 作业

* 通过继承，实现以下需求:

1.

|  |
| --- |
|  |

2.描述动物类、鸟类

* + 动物具备 name、 age 属性，还有移动的行为
  + 鸟具备 name、age、color属性，具备移动和飞的行为

3.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

4.描述动物、狗、鱼 三个类，动物都具备name、 color两个属性、动物都具备移动的行为。

* + 狗的特有行为是咬人
  + 鱼的特有行为是吹泡泡