# 第 5 章　继承与多态

**知识要点：**

1. 继承
2. 引用类型的转换
3. 多态
4. final 修饰符
5. Object 类

**学习目标：**

继承和多态是面向对象编程的两大核心特征。通过继承可以更有效地组织程序结构，最大限度地达到代码的重复利用和优化；通过多态使用同样的方法调用形式，却可以实现不一样的行为和结果。

通过本章的学习，读者可以理解继承的概念；掌握子类的定义、子类对象的创建过程和继承关系中的内存分配；掌握方法重写与方法重载的应用以及两者的区别；理解引用类型转换中的上转型和下转型；掌握多态的两种形式；掌握 final 修饰符的用法；理解 Object 类。

## 5.1　继　　承


继承

### 5.1.1　继承概述

Java 语言中使用类描述现实世界中的事物。类源自分类学的概念，就像生物可以分为植物和动物，动物又可以分为人类和猫类等。分类层次图如图 5-1 所示。

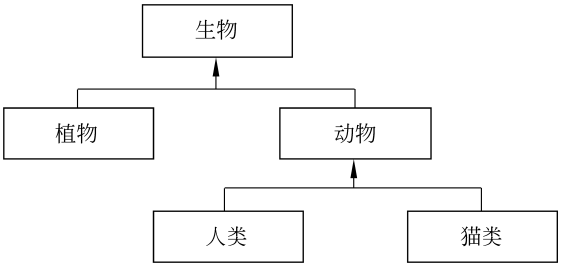


图 5-1　分类层次图

下面以“人类”为例讲解继承关系的必要性。假设“人类”按照职业可以分为“教师”“学生”和“职员”等，没有继承关系时，为了描述“教师”这一类事物时，会抽象出教师编号、名字、年龄、身高、职称等基本特征，以及说话、教学等行为特征；为了描述“学生”这一类事物时，会抽象出学号、名字、年龄、身高等基本特征以及说话、学习等行为特征。这两类事物中有很多相同或者相似的基本特征以及行为特征，是否有一种方式或者机制，可把多个不同类事物之间相同的部分提取出来，共同管理？这就是继承由来的原因。

读者会发现,“教师”和“学生”都是“人类”的1种,“人类”可以拥有所有人具有的基本特征(如名字、年龄和身高)和行为(如说话),而“教师”继承自“人类”,自动拥有“人类”的特征和行为,并且还可以拥有“教师”独特的特征(如教师编号和职称)和行为(如教学),“学生”继承自“人类”,也会自动拥有“人类”的特征和行为,并且可以拥有“学生”独特的特征(如学号)和行为(如学习)等。

通常把分类层次中处于上层(大类)的类称为父类或者超类,把处于下层并由该父类派生出的小类称为子类或者派生类。在整个类继承层次中处于最顶层的类是 Object 类,它是所有类的父类,也称为根类。除了 Object 类之外,所有的类都有父类。

继承是面向对象程序设计中的重要机制之一,它使编程人员可以在原有类的基础上快速设计出功能更强大的新类,而不必从头开始定义,避免了很多重复性的工作。在继承关系中,子类会自动拥有父类的属性和方法,同时也可以加入自己的一些特性,使得子类更具体,功能更丰富。

继承分为单继承和多继承两种方式。在单继承中,每一个类只能有一个父类(如人的亲生父亲只能有一个),而多继承则每一个类可以有多个父类。单继承是最常见的继承方式,因为其条理清晰,语法规则简单,更接近现实世界,所以 Java 语言中采用的是单继承方式。

### 5.1.2 子类的继承规则

在子类定义中,使用关键字 extends 表示继承关系,格式如下:

```
[修饰符] class  子类名  extends  父类名{
        子类体
}
```

例如:

```
public class Person [extends Object]{父类体}
public class Student extends Person{子类体}
```

说明:

① 修饰符,通常是访问权限修饰符 public 和默认修饰符。

② 需要先创建父类,之后才可以创建子类。子类名和父类名都需要满足标识符规则。

③ Student 类是 Person 类的子类,Person 类是 Student 类的父类。

④ 每个类都有父类,如果类在定义时没有使用 extends 关键字显式声明父类,那么系统默认其父类是 Object 类,中括号[]部分为可选。

子类可以继承父类中除了构造方法以外的所有属性和方法,同时也可以在父类的基础上增加新的属性或者新的方法。

**【例 5-1】** 继承关系的使用。

Person 类

```
1  public class Person {
2      String name="zhangsan";
3      int age=18;
4      double height=1.73;
```

```
5      public void say() {
6          System.out.println("Person类中的say()方法");
7      }
8  }
```

Student 类

```
1  public class Student extends Person {
2      String sno="s01";
3      public void study() {
4          System.out.println("Student类中的study()方法");
5      }
6  }
```

Test 类

```
1  public class Test {
2      public static void main(String[] args) {
3          Student s1=new Student();
4          System.out.println(s1.name+" "+s1.age+" "+s1.height+" "+s1.sno);
5          s1.say();
6          s1.study();
7      }
8  }
```

运行结果：

```
zhangsan 18 1.73 s01
Person类中的say()方法
Student类中的study()方法
```

代码解释：

第 1 行 Person 类后省略了 extends Object 语句，为了简化代码的调用，在第 2～4 直接给 3 个实例变量赋初值。

Student 类是 Person 类的子类，在 Student 类中定义了实例变量 sno 并对其赋初值，以及定义了 study()方法。

Test 类中第 3 行创建了 Student 对象 s1，第 4 行使用对象名 s1 访问了从 Person 类中继承过来的 3 个实例变量以及子类中定义的 sno，第 5 行调用 Person 类中继承过来的实例方法 say()，第 6 行调用子类中定义的实例方法 study()。

例 5-1 中 UML 所绘制的继承关系示意图如图 5-2 所示。

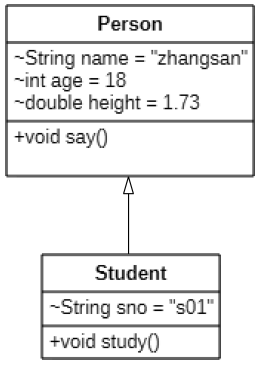

图 5-2　继承关系示意图

说明：在图 5-2 中空心箭头表示继承关系（又被称为泛化关系），箭头由子类指向父类。

### 5.1.3　子类对象的创建和 super

在创建子类对象的过程中，首先会调用子类的构造方法，但是在子类构造方法中总会默

认调用父类的构造方法，先完成父类实例变量的初始化，再完成子类实例变量的初始化，之后才可以创建出子类对象。

子类调用父类构造方法的规则如下：

```
super([参数列表]);
```

说明：

① super()必须在子类构造方法的第 1 行使用，注释语句除外。

② 参数列表由父类构造方法的参数决定。

当例 5-1 中的 Person 类换成如下代码时，Student 类会出现编译错误。

```
1  public class Person {
2      String name="zhangsan";
3      int age=18;
4      double height=1.73;
5      public Person(String name) {
6          this.name=name;
7      }
8      public void say() {
9          System.out.println("Person 类中 say()方法");
10     }
11 }
```

Student 类会出现的错误信息："在默认构造方法中无法调用 Person 类中的 Person()空构造方法"。

按照构造方法的特点，每个类中都有构造方法，当类中没有定义任何构造方法时，系统会为该类中增加一个空构造方法，而在子类的构造方法的第 1 行中，总会使用 super()调用父类的空构造方法。

在 Student 类中补充完整的代码如下：

```
1  public class Student extends Person {
2      String sno="s01";
3      public Student() {
4          super();
5      }
6      public void study() {
7          System.out.println("Student 类中的 study()方法");
8      }
9  }
```

代码解释：

第 3～5 行为系统按照类创建的规则自动添加的内容，这部分往往是初学者容易忽略的地方。第 4 行调用 Person 类中的空构造方法，因为父类中有带参数的构造方法，所以系统就不会为 Person 类提供空构造方法，调用失败，出现编译错误。

该类问题的解决办法是在父类中增加一个空构造方法，如下列所示。

【例 5-2】 在 Person 类中增加空构造方法。

```
1  public class Person {
2      String name="zhangsan";
3      int age=18;
4      double height=1.73;
5      public Person() {}
6      public Person(String name) {
7          this.name=name;
8      }
9      public void say() {
10         System.out.println("Person 类中 say()方法");
11     }
12 }
```

说明：第 5、6 行 Person 类中的 2 个构造方法是方法重载关系。

### 5.1.4 继承关系中的内存分配

当创建子类对象时，如何为子类对象分配内存空间？子类对象的内存分配主要分为栈内存、堆内存和方法区 3 部分。栈内存用来保存对象名，即该对象在堆内存的首地址。堆内存用来保存子类对象的实例变量，以及从父类继承过来的实例变量，同时在子类对象里保留对方法区中实例方法区和类方法区的引用。

例 5-1 子类对象内存分配示意图如图 5-3 所示。

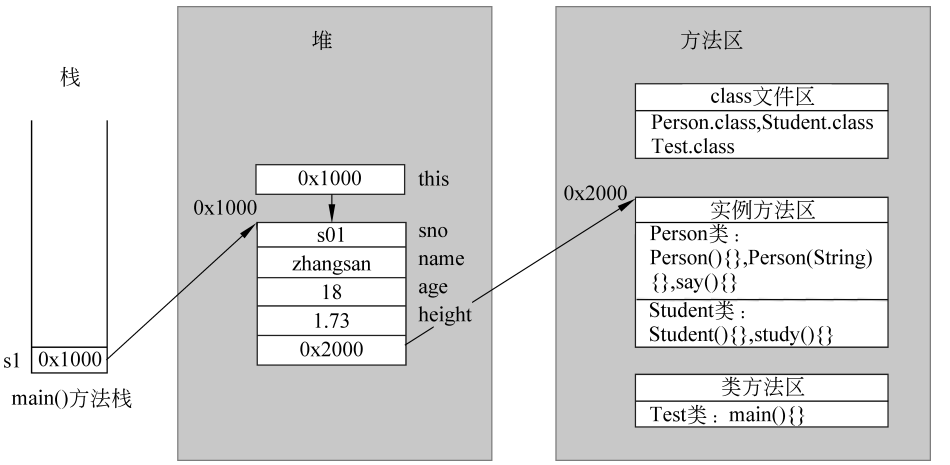


图 5-3 例 5-1 子类对象内存分配示意图

说明：this 是堆内存中的引用，为了访问当前对象。sno 为 Student 类中定义的实例变量，name、age、height 为从 Person 类继承过来的实例变量。另外，因为 Person 类和 Student 类中只有实例方法，所以本图中只有对实例方法区访问的引用。

### 5.1.5 实例变量的隐藏

实例变量的隐藏是指当子类和父类都有同名的实例变量时，子类的实例变量把父类的实例变量隐藏起来，子类对象访问时，优先访问子类中定义的实例变量。就像我们将一张大小完全相同的扑克牌放在另外一张扑克牌前一样，这两张扑克牌都是真实存在的，前一张扑克牌把后一张扑克牌隐藏起来。初学者很容易将其错误地理解为替代或覆盖关系。

为了能够访问被子类实例变量隐藏的父类中的实例变量，需要使用“super.实例变量名”。“super.”表示对父类的引用，和“this.”表示对子类的引用应区别理解。

**【例 5-3】** 实例变量的隐藏。

Person 类

```
1  public class Person {
2      String name="zhangsan";
3      int age=18;
4      double height=1.73;
5  }
```

Student 类

```
1  public class Student extends Person {
2      String name="lisi";
3      String sno="s01";
4      public void showName() {
5          System.out.println(this.name+" "+super.name);
6      }
7  }
```

Test 类

```
1  public class Test {
2      public static void main(String[] args) {
3          Student s1=new Student();
4          System.out.println(s1.name);
5          s1.showName();
6      }
7  }
```

运行结果：

```
lisi
lisi zhangsan
```

代码解释：

子类 Student 和父类 Person 都有同名的实例变量 name。在 Test 类中的第 3 行创建 Student 对象。

第 4 行访问子类的实例变量 name。

第 5 行调用 showName()方法，分别使用 this.和 super.访问了子类的实例变量和父类

的实例变量 name。

例 5-3 对应的内存分配示意图如图 5-4 所示。

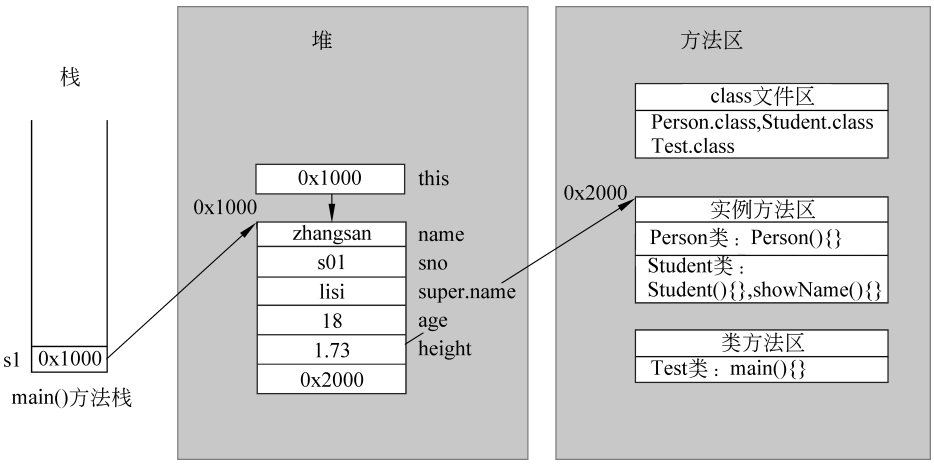


图 5-4　父、子类同名实例变量的内存分配示意图

### 5.1.6　方法重写和方法重载

方法重写(或方法覆盖)是指子类和父类中有同样的实例方法,即方法的访问权限修饰符、返回数据类型、方法名相同。子类对象调用方法时,优先使用子类自己定义的实例方法,为了能够访问到父类的实例方法,需要使用“super.实例方法名()”。

子类和父类中的方法也存在方法重载的情况,即方法名相同、参数列表不同。重载方法的调用由方法名和参数列表决定。

**【例 5-4】** 方法重写和方法重载。

Person 类

```
1  public class Person {
2      String name="zhangsan";
3      int age=18;
4      double height=1.73;
5      public void say() {
6          System.out.println("Person 类中的 say()");
7      }
8  }
```

Student 类

```
1  public class Student extends Person {
2      String name="lisi";
3      String sno="s01";
4      public void say() {
5          super.say();
6          System.out.println("Student 类中的 say()");
```

```
 7      }
 8      public void say(String name) {
 9          System.out.println(this.name+" say hello to "+name);
10      }
11  }
```

Test 类

```
 1  public class Test {
 2      public static void main(String[] args) {
 3          Student s1=new Student();
 4          s1.say();
 5          s1.say("wangwu");
 6      }
 7  }
```

运行结果：

```
Person 类中的 say()
Student 类中的 say()
lisi say hello to wangwu
```

代码解释：

Person 类中第 5 行定义了 say()实例方法，Student 类中第 4 行也定义了同样的 say()实例方法，这两个方法是方法重写关系。Student 类中第 8 行的 say(String name)方法和其他两个 say()方法是方法重载关系。

Test 类中第 3 行创建了子类对象 s1，第 4 行优先调用 Student 类中的 say()方法，Student 类中第 5 行使用 super.say()调用了父类的 say()方法，并执行输出语句。

Test 类中第 5 行调用了 Student 类中第 8 行定义的有参数的 say()方法，因为方法参数是局部变量和类中定义的实例变量重名，所以第 9 行需要用 this.区分，this.name 表示实例变量。

**注意**：*在继承关系中出现的同名方法，要么是方法重写，要么是方法重载，否则会出现编译错误。*

错误的写法，例如：

Person 类

```
public String say(String name) {
    return n;
}
```

Student 类

```
public void say(String name) {}
```

这两个方法既不是方法重写，也不是方法重载，Student 类中的方法会出现编译错误。

### 5.1.7 子类对父类类成员的访问

父类中定义的类成员包括类变量和类方法，对其访问的方式需要使用“父类名.类变量”

或“父类名.类方法名()”。在类方法中不能使用 super.和 super(),因为 super 属于对象的引用,而类成员属于类的引用。

**注意**:子类和父类中相同的类方法之间不叫方法重写,类方法的调用只看引用变量的类型。

**【例 5-5】** 子类对父类类成员的访问。

Person 类

```
1  public class Person {
2      String name="zhangsan";
3      int age=18;
4      double height=1.73;
5      static String country="china";
6      public static void show() {
7          System.out.println("Person 类中的类变量:"+country);
8      }
9  }
```

Student 类

```
1  public class Student extends Person {
2      String name="lisi";
3      String sno="s01";
4      static String country="中国";
5      public static void show() {
6          Person.show();
7          //super.show();
8          System.out.println("Student 类中的类变量:"+country);
9      }
10 }
```

Test 类

```
1  public class Test {
2      public static void main(String[] args) {
3          Student s1=new Student();
4          s1.show();
5          Student.show();
6      }
7  }
```

运行结果:

```
Person 类中的类变量:china
Student 类中的类变量:中国
Person 类中的类变量:china
Student 类中的类变量:中国
```

代码解释:

Test 类中第 3 行创建了子类 Student 对象 s1，第 4 行通过“对象名.类方法名()”的方式调用 Student 类中定义的类方法 show()。

Student 类中 show()方法调用父类中的类方法 show()时，只可以用 Person 类名调用，因为在类方法中不可以使用“this.”或者“super.”。第 6 行调用 Person 类中的类方法 show()。

Test 类中第 5 行使用“类名.类方法名()”的方式调用 Student 类中的类方法 show()。

## 课堂练习 1

1. Java 语言中类间的继承关系是(　　)。

A. 多重的　　B. 单重的

C. 线程的　　D. 不能继承

2. 下列程序的运行结果是(　　)。

```
class Parent {
    void printMe() {
        System.out.print("parent ");
    }
}
class Child extends Parent {
    void printMe() {
        System.out.print("child ");
    }
    void printAll() {
        super.printMe();
        this.printMe();
        printMe();
    }
}
public class TestThis {
    public static void main(String args[]) {
        Child baby =new Child();
        baby.printAll();
    }
}
```

A. child child child　　B. child parent child

C. parent child child　　D. parent parent child

3. 下列程序的运行结果是(　　)。

```
class First {
    public First() {
        speak();
    }
    public void speak() {
        System.out.println("in First class");
```