



# Java程序设计

---

## 第5章 继承与多态



# 第5章 继承与多态

---

5.1 继承简介

5.2 子类的继承性

5.3 多态

5.4 抽象类

5.5 final修饰符

5.6 内部类

5.7 匿名类



## 5.1 继承简介

---

在面向对象程序设计中，继承是不可或缺的一部分。通过继承可以实现代码的重用，提高程序的可维护性。



## 5.1.1 继承的概念

---

- 继承一般是指晚辈从父辈那里继承财产，也可以说是子女拥有父母所给予他们的东西。在面向对象程序设计中，继承的含义与此类似，所不同的是，这里继承的实体是类。也就是说继承是子类拥有父类的成员。



## 5.1.2 子类对象的创建

在类的声明中，可以通过使用关键字`extends`来显式地指明其父类。

语法格式为：

```
[修饰符] class 子类名 extends 父类名
```

修饰符：可选，用于指定类的访问权限，可选值为`public`、`abstract`和`final`。

子类名：必选，用于指定子类的名称，类名必须是合法的Java标识符。一般情况下，要求首字母大写。

`extends` 父类名：必选，用于指定要定义的子类继承于哪个父类。



## 5.1.3 继承的使用原则

- 子类可以继承父类中所有可被子类访问的成员变量和成员方法，但必须遵循以下原则：
  - (1) 子类能够继承父类中被声明为public和protected的成员变量和成员方法，但不能继承被声明为private的成员变量和成员方法；
  - (2) 子类能够继承在同一个包中的由默认修饰符修饰的成员变量和成员方法；
  - (3) 如果子类声明了一个与父类的成员变量同名的成员变量，则子类不能继承父类的成员变量，此时称子类的成员变量隐藏了父类的成员变量；
  - (4) 如果子类声明了一个与父类的成员方法同名的成员方法，则子类不能继承父类的成员方法，此时称子类的成员方法覆盖了父类的成员方法。



## 5.1.4 使用super关键字

- super关键字主要有以下两种用途。
- (1) 调用父类的构造方法。
- 子类可以调用父类的构造方法，但是必须在子类的构造方法中使用super关键字来调用。其具体的语法格式如下：
- `super([参数列表]);`
- 如果父类的构造方法中包括参数，则参数列表为必选项，用于指定父类构造方法的入口参数。
- (2) 操作被隐藏的成员变量和被覆盖的成员方法。
- 如果想在子类中操作父类中被隐藏的成员变量和被覆盖的成员方法，也可以使用super关键字。
- 语法格式为：
- `super.成员变量名`
- `super.成员方法名([参数列表])`



# 使用super关键字

---

super关键字主要有以下两种用途。

- (1) 调用父类的构造方法
- (2) 操作被隐藏的成员变量和被覆盖的成员方法





## 5.2 子类的继承

---

子类中的一部分成员是子类自己声明、创建的，另一部分是通过它的父类继承的。在Java中，Object类是所有类的祖先类，也就是说任何类都继承自Object类。除了Object类以外的每个类，有且仅有一个父类，一个类可以有零个或多个子类。

## 5.2.1 同一包中的子类与父类

- 如果子类与父类都在同一包中，那么子类继承父类中非private修饰的成员变量和方法。
- 【例】 有三个类，People类是父类，Student类是继承父类的子类，Teacher类也是继承父类的子类，Example类是测试类。



## 5.2.2 非同一包中的子类与父类

当子类与父类不在同一包中，父类中使用`private`修饰符修饰的成员变量和友好的成员变量不会被继承，也就是子类只能继承父类中使用`public`和`protected`访问修饰符修饰的成员变量作为子类的成员变量，同样，子类也只能继承父类中使用`public`和`protected`访问修饰符修饰的方法作为子类的方法。



## 5.2.3 继承关系的UML图

---

当一个类是另一个类的子类的时候，可以通过UML图使用实线连接两个类来表示二者之间的继承关系。实线的起始端是子类的UML图，实线的终止端是父类的UML图。在实线的终止端使用一个空心三角形表示实线的结束。



## 5.2.4 继承中的Protected

在一个类A中，它所定义的成员变量和方法都被protected所修饰，类A被类B、类C继承，那么在类B与类C中都继承了类A的成员变量和方法。这时，如果在类C中创建一个自身的对象，那么该对象可以访问父类的和自身定义的protected修饰的变量和方法。但是在其他类中，比如Student类，对于子类C自己声明的protected成员变量和方法，只要Student类与C类在同一包中，创建的对象就可以访问这些被protected修饰的成员变量和方法。对于子类C从父类中继承的protected成员变量和方法，只要Student类与C类的父类在同一包中，创建的对象就能够访问继承的protected成员变量和方法。



## 5.3 多 态

---

多态是面向对象程序设计的重要组成部分，是面向对象的3个基本特性之一。在Java语言中，通常使用方法的重载（Overloading）和覆盖（Overriding）实现类的多态性。

## 5.3.1 方法的重载

方法的重载是指在一个类中，出现多个方法名相同，但参数个数或参数类型不同的方法，则称为方法的重载。Java在执行具有重载关系的方法时，将根据调用参数的个数和类型区分具体执行的是哪个方法。

【例】 定义一个名称为Calculate的类，在该类中定义两个名称为getArea()的方法（参数个数不同）和两个名称为draw()的方法（参数类型不同）。

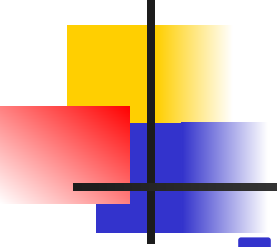


## 5.3.2 避免重载出现的歧义

---

- 方法重载之间必须保证参数不同，但是需要注意，重载方法在被调用时可能出现调用歧义。例如下面Student类中的speak方法就很容易引发歧义。





```
■ public class Student {  
■     static void speak (double a ,int b) {  
■         System.out.println( “我很高兴” );  
■     }  
■     static void speak (int a,double b) {  
■         System.out.println( “I am so Happy”  
■     );  
■     }  
■ }
```

### 5.3.3 方法的覆盖

当子类继承父类中所有可能被子类访问的成员方法时，如果子类的方法名与父类的方法名相同，那么子类就不能继承父类的方法，此时，称子类的方法覆盖了父类的方法。覆盖体现了子类补充或者改变父类方法的能力，通过覆盖，可以使一个方法在不同的子类中表现出不同的行为。



## 5.3.4 向上转型

一个对象可以看做本类类型，也可以看做它的超类类型。取得一个对象的引用并将它看做超类的对象，称为向上转型。

【例】 创建抽象的动物类，在该类中定义一个move() 移动方法，并创建两个子类：鸚鵡和乌龟。在Zoo类中定义free() 放生方法，该方法接收动物类做方法的参数，并调用参数的move() 方法使动物获得自由。



## 5.4 抽象类

---

通常可以说四边形具有4条边，或者更具体一点，平行四边形是具有对边平行且相等特性的特殊四边形，等腰三角形是腰相等的三角形，这些描述都是合乎情理的，但对于图形对象却不能使用具体的语言进行描述，它有几条边，究竟是什么图形，没有人能说清楚，这种类在Java中被定义为抽象类。

## 5.4.1 抽象类和抽象方法

■ 所谓抽象类就是只声明方法的存在而不去具体实现它的类。抽象类不能被实例化，也就是不能创建其对象。在定义抽象类时，要在关键字class前面加上关键字abstract。

■ 语法格式为：

■ abstract class 类名{

■ 类体

■ }

## 5.4.2 抽象类和抽象方法的规则

- 综上所述，抽象类和抽象方法的规则总结如下：
- (1) 抽象类必须使用abstract修饰符来修饰，抽象方法必须使用abstract修饰符来修饰。
- (2) 抽象类不能被实例化，无法使用new关键字来调用抽象类的构造器创建抽象类的实例，即使抽象类里不包含抽象方法，这个抽象类也不能创建实例。
- (3) 抽象类可以包含属性、方法（普通方法和抽象方法）、构造器、初始化块、内部类、枚举类。抽象类的构造器不能用于创建实例，主要是用于被其子类调用。
- (4) 含有抽象方法的类（包括直接定义了一个抽象方法；继承了一个抽象父类，但没有完全实现父类包含的抽象方法；以及实现了一个接口，但没有完全实现接口包含的抽象方法三种情况）只能被定义成

抽象类。

信息科学与工程学院



## 5.4.3 抽象类的作用

■ 抽象类不能被创建实例，只能被继承。从语义角度上看，抽象类是从多个具体类中抽象出来的父类，它具有更高层次的抽象。从多个具有相同特征的类中抽象出一个抽象类，以这个抽象类为模板，从而避免子类的随意设计。

■ 抽象类体现的就是这种模板模式的设计，抽象类作为多个子类的模板，子类在抽象类的基础上进行扩展，但是子类大致保留抽象类的行为。



## 5.5 final修饰符

---

final关键字用来修饰类、变量和方法，final关键字用于表示它修饰的类、方法和变量不可改变。





## 5.5.1 final 变量

### 1. final 修饰成员变量

成员变量是随着类初始化或对象初始化而初始化的。当类初始化时，系统会为该类的类属性分配内存，并分配默认值；当创建对象时，系统会为该对象的实例属性分配内存，并分配默认值。

对于final修饰的成员变量，如果既没有在定义成员变量时指定初始值，也没有在初始化块、构造器中为成员变量指定初始值，那么这些成员变量的值将一直是0、' \u0000'、false或null，这些成员变量也就失去了意义。

因此当定义final变量时，要么指定初值，要么在初始化块、构造器中初始化成员变量。当给成员变量指定默认值之后，则不能在初始化块、构造器中为该属性重新赋值。。



## ■ 2. final修饰局部变量

- 使用final修饰符修饰的局部变量，如果在定义的时候没有指定初始值，则可以在后面的代码中对该final局部变量赋值，但是只能赋一次值，不能重复赋值。如果final修饰的局部变量在定义时已经指定默认值，则后面代码中不能再对该变量赋值。

## ■ 3. final修饰基本类型和引用类型变量的区别

- 当使用final修饰基本类型变量时，不能对基本类型变量重新赋值，因此基本类型变量不能被修改。但是对于引用类型的变量，它保存的仅仅是一个引用，final只保证这个引用所引用的地址不会改变，即一直引用同一对象，这个对象是可以发生改变的。



## 5.5.2 final类

---

使用关键字final修饰的类称为final类，该类不能被继承，即不能有子类。有时为了程序的安全性，可以将一些重要的类声明为final类。例如，Java语言提供的System类和String类都是Final类。

语法格式为：

```
final class 类名{  
    类体  
}
```



## 5.5.3 final 方法

使用final修饰符修饰的方法是不可以被重写的。如果想要不允许子类重写父类的某个方法，可以使用final修饰符修饰该方法。

例如：

```
public class Father {  
    public final void say () {}  
}
```

```
public class Son extends Father {  
    public final void say () {} // 编译错误，不允许重写final方法  
}
```



## 5.6 内部类

Java语言允许在类中定义内部类，内部类就是在其他类内部定义的子类。

一般格式为：

```
public class Zoo{  
    ... ..  
    class Wolf{    // 内部类Wolf  
    }  
}
```



# 1. 成员内部类

---

成员内部类和成员变量一样，属于类的全局成员。

一般格式为：

```
public class Sample {  
    public int id;    // 成员变量  
    class Inner{    // 成员内部类  
    }  
}
```



## 2. 局部内部类

---

局部内部类和局部变量一样，都是在方法内定义的，其有效范围只在方法内部有效。

一般格式为：

```
public void sell() {  
    class Apple {  
    }  
}
```

// 局部内部类



## 3. 静态内部类

静态内部类和静态变量类似，它都使用static关键字修饰。所以在学习静态内部类之前，必须熟悉静态变量的使用。

一般格式为：

```
public class Sample {  
    static class Apple { // 静态内部类  
    }  
}
```





## 5.7 匿名类

匿名类就是没有名称的内部类，它经常被应用于Swing程序设计中的事件监听处理。

匿名类有以下特点：

(1) 匿名类可以继承父类的方法也可以重写父类的方法。

(2) 匿名类可以访问外嵌类中的成员变量和方法，在匿名类中不能声明静态变量和静态方法。

(3) 使用匿名类时，必须在某个类中直接使用匿名类创建对象。

(4) 在使用匿名类创建对象时，要直接使用父类的构造方法。

信息科学与工程学院





## 匿名类的一般格式为：

```
new ClassName() {  
    ...  
}
```

【例】 创建Apple接口和Sample类，在Sample类中编写print()方法，该方法接收一个实现Apple接口的对象做参数，并执行该参数的say()方法打印一条信息。