Java程序设计

计算机科学与技术

贡正仙

zhxgong@suda.edu.cn

第5章 继承

导读

难点

- 成员变量的隐藏和方法重写
- 向上转型
- 继承与多态

§5.1 子类与父类

利用继承，可以先编写一个共有属性的一般类，根据该一般类再编写具有特殊属性的新类，新类继承一般类的状态和行为，并根据需要增加它自己的新的状态和行为。

通过继承可以实现代码的重用，提高程序的可维护性

§5.1 子类与父类

声明一个类的子类的格式如下：

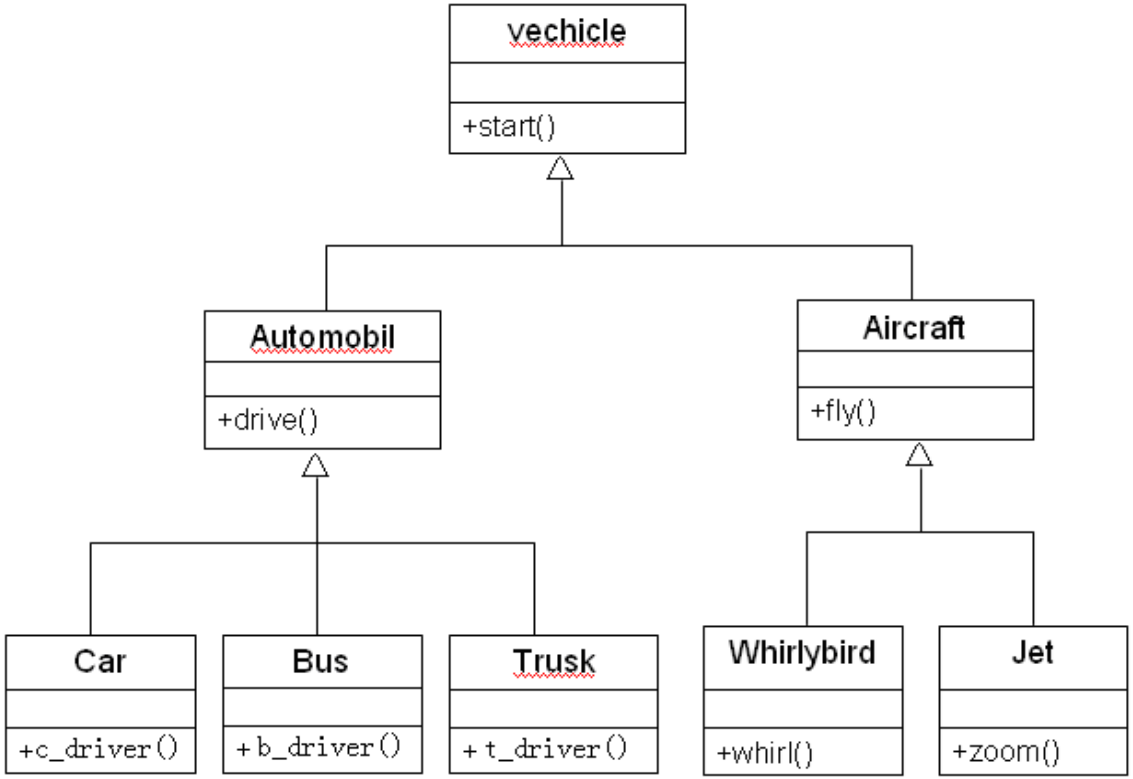
```
(public, abstract,final) class 子类名 extends 父类名 {
    …
}
```

由继承而得到的类称为子类，被继承的类称为父类（超类）。

§5.1 子类与父类

继承的层次结构

在Java中，不允许多重继承，但却允许多层继承，即一个子类又可以是其它类的超类，从而可以形成类的多级继承层次结构。



§5.1 子类与父类

所谓子类继承父类的成员变量作为自己的一个成员变量，就好象它们是在子类中直接声明一样，可以被子类中自己定义的任何实例方法操作。

所谓子类继承父类的方法作为子类中的一个方法，就象它们是在子类中直接定义了一样，可以被子类中自己定义的任何实例方法调用。

§5.1 子类与父类

5.1 子类的继承性

子类能继承超类的成员变量和成员方法，在继承过程中，需要注意的是，类的每一个成员都被赋予了一定的访问权限，成员访问权限不同，子类对它的继承性也不同。子类对超类的继承性主要有以下三种情况：

1．超类的private变量和private方法不能被子类继承。

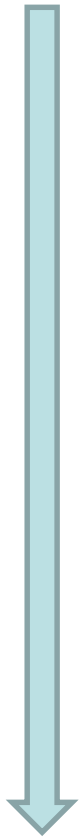
2．在同一个包中，子类能继承超类的所有非private成员（public，protected和缺省）。

3．在不同包中，子类只能继承超类的public和protected成员。

§5.1 子类与父类

访问修饰符

修饰符	使用范围	位置		
		同一类	同一包	不同包
private	方法、变量	Y		
缺省	类、接口、方法、变量	Y	Y	
protected	方法、变量	Y	Y	Y(只有子类)
public	类、接口、方法、变量	Y	Y	Y（任何类）
访问控制表				



访问级别升高

§5.1 子类与父类

5.1 子类的继承性

子类能继承超类的成员变量和成员方法，在继承过程中，除了访问权限还需要注意的是:

1．如果子类声明了一个与父类同名的成员变量，则子类不能继承超类的该成员变量，此时称子类的成员变量隐藏（覆盖）了父类的成员变量。

2．如果子类声明了一个与父类同名的成员方法，则子类不能继承超类的该成员方法，此时称子类的成员变量隐藏（覆盖）了父类的成员方法。

§5.2 子类对象的构造过程

构造函数：

构造函数不能继承——不符合与类同名

当用子类的构造方法创建一个子类的对象时，子类的构造方法**总是先调用父类的某个方法，**在多层继承中，子类构造方法的执行顺序是按创建时的顺序，**从上往下**执行。

§5.2 子类对象的构造过程

子类中构造函数初始化各成员的次序：

- 父类属性初始化成默认值
- 调用父类的构造函数之一完成指定父类属性的初始化
- 子类属性初始化成默认值
- 调用子类构造函数之一完成指定子类属性的初始化

§5.2 子类对象的构造过程

多级继承的构造函数执行顺序

说明：

1.如果子类的构造函数中显式地调用父类构造函数，super必须是第一条语句

2.如果子类构造函数没有显式地调用父类构造函数，则将自动调用父类默认（即无参）的构造函数。如果父类没有无参构造函数，则Java编译器报错。

§5.2 子类对象的构造过程

用子类创建对象时，不仅子类中声明的成员变量被分配了内存，而且父类的成员变量也都分配了内存空间，但只将其中一部分（子类继承的那部分）作为分配给子类对象的变量。

父类未被继承的变量，并不是成为垃圾，因为通过某些继承的方法，可以操作这部分未被继承的变量

看下面例子

§5.2 子类对象的构造过程

```
public class A {
    private int x;
    public void setX(int x) {
       this.x=x;
    }
    public int getX() {
       return x;
    }
}
public class B extends A {
    double y=12;
    public void setY(int y)
    {
       //this.y=y+x; 非法，子类没有继承x
    }
    public double getY() {
       return y;
    }
}
public class Example5_2 {
  public static void main(String args[]) {
     B b=new B();
     b.setX(888);
     System.out.println("子类对象未继承的x的值是:"+b.getX());
     b.y=12.678;
     System.out.println("子类对象的实例变量y的值是:"+b.getY());
  }
}
```

5.3 成员变量的隐藏和方法重写/覆盖

子类继承超类后，自动继承超类的**非私有**成员变量和成员方法，但如果在子类中定义了与超类**同名的成员变量**，且这些成员变量在超类中是非私有的，则超类的这些成员变量不能被子类继承，此时称子类的**成员变量隐藏了超类的成员变量**。

另一方面，如果在子类中定义了一个方法，这个方法的**名字、返回类型和参数声明与超类的某个方法完全相同**，并且超类的这个方法是非私有的，此时超类的这个方法被子类隐藏而不能被子类继承，称这时子类的这个**方法覆盖(override)**或**重写**了超类的同名方法。

覆盖与重载：

重载：签名不同的同名函数间

子类中继承了父类的方法，同时可增加新方法与之重载

覆盖：子类中的方法原型和父类中的方法原型一致

说明：在JDK5中，允许子类将覆盖方法的返回类型定义成原返回类型的子类型

例如：Employee类中有

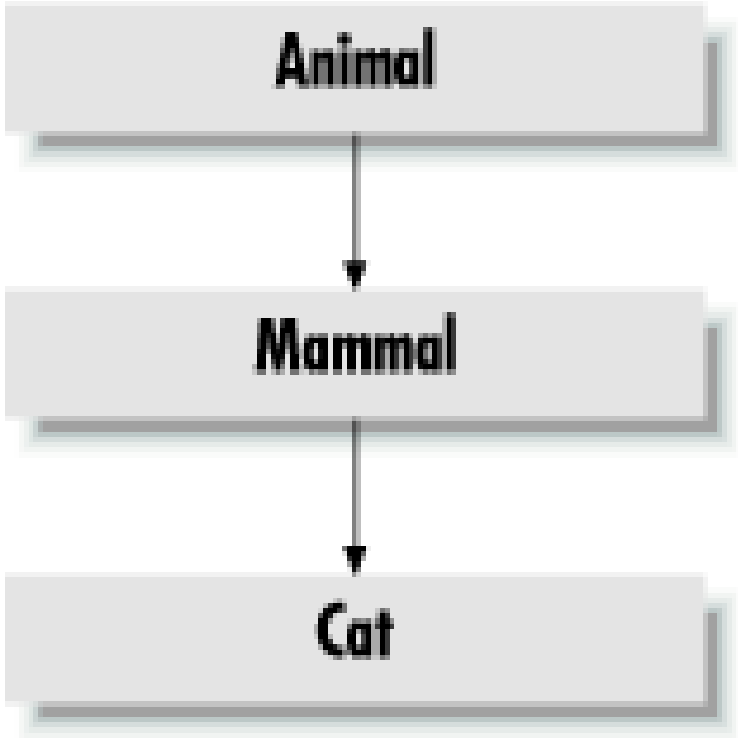
```
public void getBuddy(){……}
```

在其子类Manager中，方法

```
public void getBuddy(){……}
```

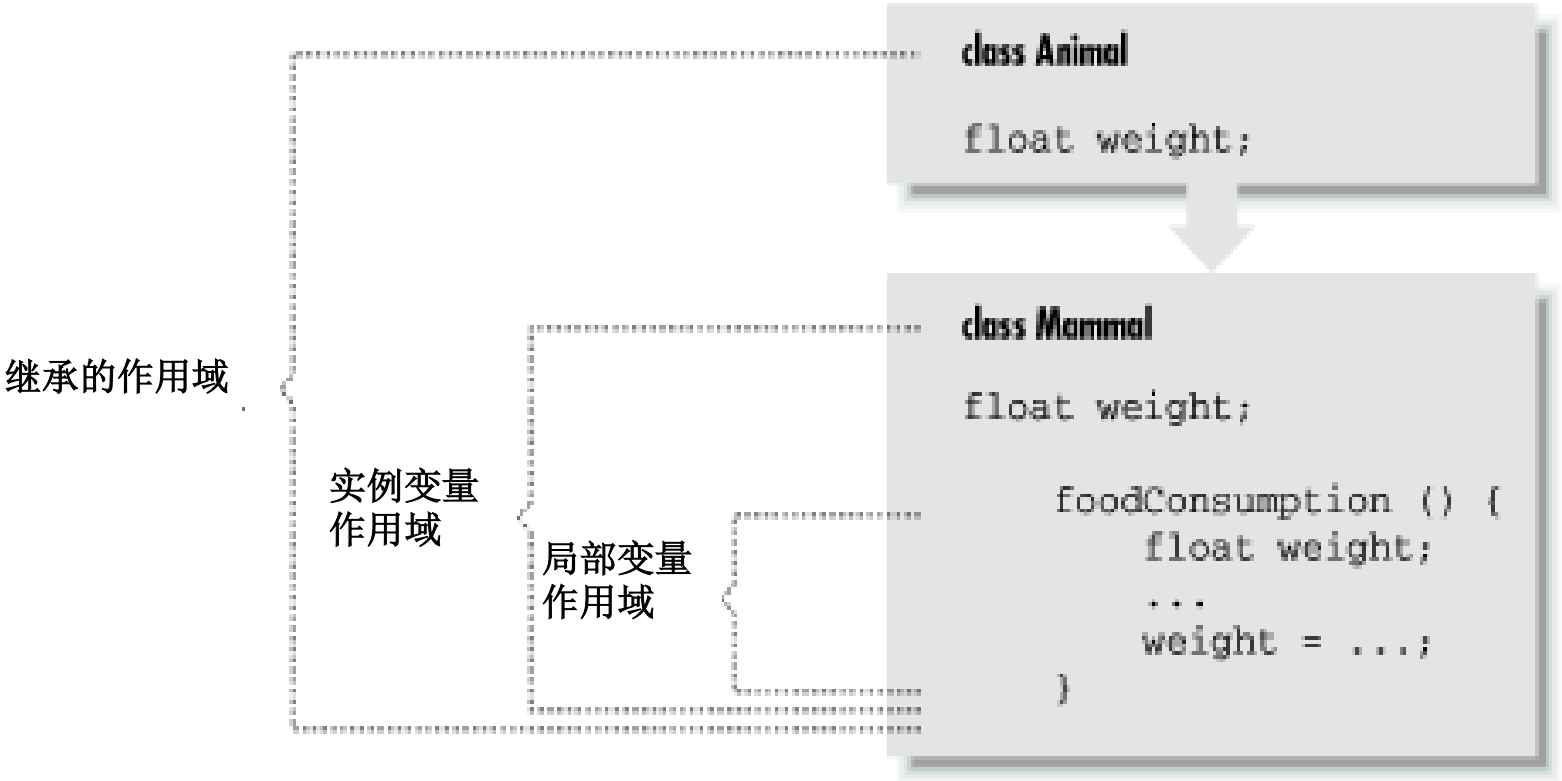
覆盖了父类中的该方法

示例



示例（续）

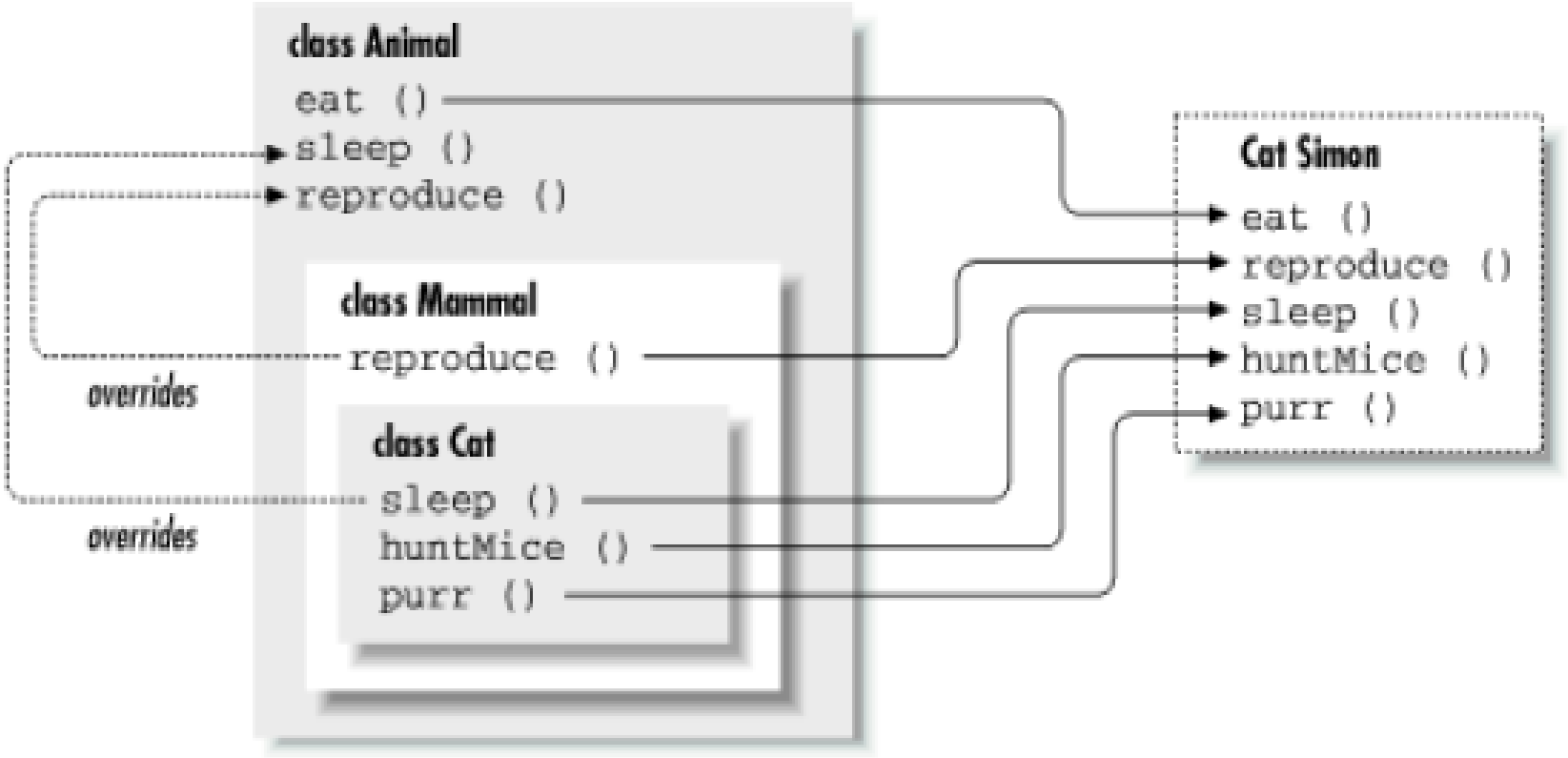
- 屏蔽变量的作用域



示例（续）

- 方法覆盖

参见ExtendClass.java



5.3 成员变量的隐藏和方法重写

子类通过成员变量的隐藏和方法的重写可以把超类的状态和行为改变为自身的状态和行为。如果子类重写了超类的方法，则运行时系统调用子类重写的方法，否则调用继承的方法。在重写超类方法时应**注意以下两点**：

1. 重写超类的方法时，可以**保持或提升访问级别**，但不允许降低方法的访问级别。即父类中该方法为public，子类必须也是public；而父类中是protected，子类可以为public或protected。

2. 在子类中，如果要访问被子类隐藏的超类的成员变量和被重写的超类的方法，可以使用关键字super。

super关键字

在Java中，super关键字可在子类中用来表示对直接超类的引用，所以当我们想在子类中使用被隐藏了的超类的成员变量和被重写的超类方法时，可以使用super关键字。具体应用时，super关键字的使用有两种形式：

一是使用super调用超类的构造方法；

二是使用super访问已被子类成员隐藏掉的超类成员（变量或方法）。

super关键字

1．使用super访问被隐藏的超类的成员变量和被重写的超类方法

在子类中使用super访问被隐藏的成员变量和被重写的超类方法的格式如下：

访问被隐藏的成员变量：super.成员变量名

访问被隐藏的方法：super.方法名

2．使用super调用超类的构造方法

在子类继承超类时，除了超类的私有成员不能被子类继承外，**超类的构造方法也不能被子类继承**。如果想在子类中使用超类的构造方法，需要使用super关键字。

思考：

this和super的用法区别

5.4 多态性

实现对象的多态性有两种途径，分别是编译时多态性和运行时多态性。

1．编译时多态性

也称静态多态性，表现为**方法重载**和变量的隐藏。

2．运行时多态性

也称动态多态性，表现为继承机制中的**方法覆盖**（重写）和**向上转型**来实现。即动态多态性的实现可以通过向上转型对象调用各子类重写的方法，使得运行后，各子类对象可以得到彼此不同的功能行为。

§5.4 继承与多态

多态性就是指父类的某个方法被其子类重写时，可以各自产生自己的功能行为。

```
class  动物 {
   void cry() {

   }
}
class 狗 extends 动物 {
   void cry() {
      System.out.println("这是狗的叫声：
汪汪...汪汪");
   }
}
```

```
class 猫 extends 动物 {
   void cry() {
      System.out.println("这是猫的叫声：
          喵喵...喵喵...");
   }
}
public class Example5_10 {
   public static void main(String args[]) {
      动物 animal=new 狗();
      animal.cry();
      animal=new 猫();
      animal.cry();
   }
}
```

5.6 多态性

5.4.1 向上转型对象

对象的向上转型对象的**实体是子类负责创建的**

```
class A{
}
Class B extends A{
}
```

向上转型

```
A a =new B()
```

5.4 多态性

5.4.1 向上转型对象

使用向上转型对象时，**需特别注意**以下几点：

1．可以将对象的向上转型对象强制转换到一个子类对象，这时该子类对象又具备了子类的所有属性和功能，如：

```
A a=new B();
B b=(B)a;
```

2．不可以将超类创建的对象的引用赋值给子类对象(即不能说动物是狗)。如：

```
A a=new A();
B b=a;//非法
```

5.4 多态性

对象的**向上转型**对象的实体是子类负责创建的，但向上转型对象会失去原对象的一些属性和功能，主要表现如下：

1．向上转型对象**不能操作**子类**新增的成员变量**(失掉了一些属性)；不能使用子类**新增的方法**(失掉了一些功能)。

2．向上转型对象**可以操作**子类继承或重写的**成员方法**。

如果子类重写了超类的某个方法后，向上转型对象调用这个方法时，一定是调用子类这个重写的方法。

3.向上转型对象将使子类继承或重定义的**成员变量失效**

(下例)

```
class Parent{
int x=100;
void m(){
System.out.println("parent:"+x);
}
}

class Child extends Parent{
int x=200;
void m(){
System.out.println("child:"+x);
}
}

  Parent b=new Child();
  b.m(); // which m
  System.out.println(b.x);//which x?
```

```
class Parent{
static int x=100;
static void m(){
System.out.println("parent:"+x);
}
}

class Child extends Parent{
static int x=200;
static void m(){
System.out.println("child:"+x);
}
}
```

左边	右边
Child:200	Parent:100
100	100

5.4 多态性

在**向下转型**过程中，分为两种情况：

情况一：如果父类引用的对象如果引用的是指向的子类对象，那么在向下转型的过程中是安全的。

```
A a=new B();//向上转型
B b=(B)a;    //向下转型
```

这时该子类对象又具备了子类的所有属性和功能

情况二：如果父类引用的对象是父类本身，那么在向下转型的过程中是不安全的，编译不会出错，但是运行时会出现java.lang.ClassCastException错误。

```
A a=new A();
B b=(B)a;  //wrong
```

A a=new A();

a=new B();

……

B b=(B)a;

如何区分？

--instanceof

5.8.2 instanceof运算符

使用instancoeof运算符可以判断对象是否是某个类的实例，常常用在**类层次的强制类型转换**的情况下。

instanceof运算符使用了一般格式如下：

对象 instanceof 类类型

如果对象的类型是指定的类类型或能强制转换成指定的类型，则instanceof运算符得到值为true，否则为false。

- 子类及多态：
  - Instanceof

```
public class Employee extends Object
public class Manager extends Employee
public class Contractor extends Employee

public void method(Employee e) {
  if (e instanceof Manager) {
    // Get benefits and options along with salary
  }else if (e instanceof Contractor) {
    // Get hourly rates

  }else {
    // temporary employee
  }
}
```

参见LowConverter1.java

5.5 Object类

在Java中，所有类都默认继承自java.lang.Object类，编程人员创建的任何类都均是Object类的直接或间接子类。Java在Object类中提供了一些对所有类均可用的方法，其中以下三种方法在其它类中是比较常用的方法。

1．equals()方法。

2．hashCode()方法。

3．toString()方法。

5.5 Object类

5.5.1 equals()方法

equals()方法用于比较两个对象的引用是否相同，相同时返回true，否则返回false。

下面以继承Object类的StringBuffer类为例来演示equals()方法的使用。

```
StringBuffer s1=new StringBuffer("hello");
StringBuffer s2=new StringBuffer("hello");
if (s1.equals(s2))
        System.out.println(true);
else
        System.out.println(false);
```

5.5 Object类

5.5.1 equals()方法

在String中的equals()方法是用来比较当前字符串对象的实体是否与参数所指定的字符串对象的实体相等，而不是比较两个对象的引用。

```
String s1=new String("hello");
String s2=new String("hello");
if (s1.equals(s2))
        System.out.println(true);
else
        System.out.println(false);
```

5.5 Object类

5.5.3 toString()方法

使用toString()方法可以获取有关对象的文本信息。Object类中的toString()方法的实现给出了由类全限定性名称和@以及对象的十六进制的哈希代码所组成的一串文本信息。

Object类的子类可以重写此方法，以获得我们所需的任何方式来提供对象的有关文本信息，如，Date类就对toString()方法进行了重写，从而可以使用“星期 月 日 时：分：秒 时间标准 年”的方式来获取有关Date对象的文本信息。

5.6 final关键字

到目前为止，我们已经知道，在声明类、成员变量和方法时，都可以使用final关键字。根据final关键字出现的位置的不同，final关键字分别具有以下三种功能：

1．阻止类的继承。

2．阻止方法的重写。

3．创建常量。

5.6 final关键字

5.6.1 使用final阻止继承

在定义类时，如果使用final关键字声明类，那么这个类将不能被子类继承。例如：

```
final class A {
    public void f(){
        System.out.println("使用final声明A类");
                }
}/*编译出错,不能创建A的子类*/
class B extends A{
    public void f(){
        System.out.println("创建A的子类");
                }
}
```

5.6 final关键字

5.6.2 使用final阻止方法的重写

在某些情况下，我们可能不希望某个方法被子类重写，这时，可以在定义方法时，将该方法声明为final型即可。例如：

```
class A {
    public final void f(){
        System.out.println("使用final声明f方法");
                }
}//编译出错,不能重写f方法
class B extends A{
    public void f(){
        System.out.println("重写f方法");
                }
}
```

5.6 final关键字

5.6.3 使用final创建常量

声明成员变量时，可以使用final，这样成员变量将转变为一个具有固定值的常量。需要注意事的是，使用final声明变量时，需要同时给变量赋值，此后，该变量在整个程序执行期间将保持声明时所赋的值，不能更改该值。例如：

```
public class FinalVariableDemo {
    final int NO=10;        //声明常量
        public void f(){
            NO=20;          //非法,修改常量的值
            System.out.println("创建常量");
                }
}
```

§5.7 abstract类和abstract方法

用关键字abstract修饰的类称为abstract类（抽象类）。如：

```
abstract class A {
 ...
}
```

用关键字abstract修饰的方法称为abstract方法（抽象方法），例如：

```
abstract int min(int x,int y);
```

例子：AbstractClass1.java

§5.7.1 abstract类的特点

1. abstract类中可以有abstract方法

abstract类可以有abstract方法（抽象方法）也可以有非abstract方法。

2. abstract类不能用new运算创建对象

对于abstract类，我们不能使用new运算符创建该类的对象。

§5.7.2 abstract类与多态

abstract类只关心操作，但不关心这些操作具体实现的细节，可以使程序的设计者把主要精力放在程序的设计上，而不必拘泥于细节的实现上。

使用多态进行程序设计的核心技术之一是使用上转型对象，即将abstract类声明对象作为其子类的上转型对象，那么这个上转型对象就可以调用子类重写的方法。

例子：AbstractClass2.java

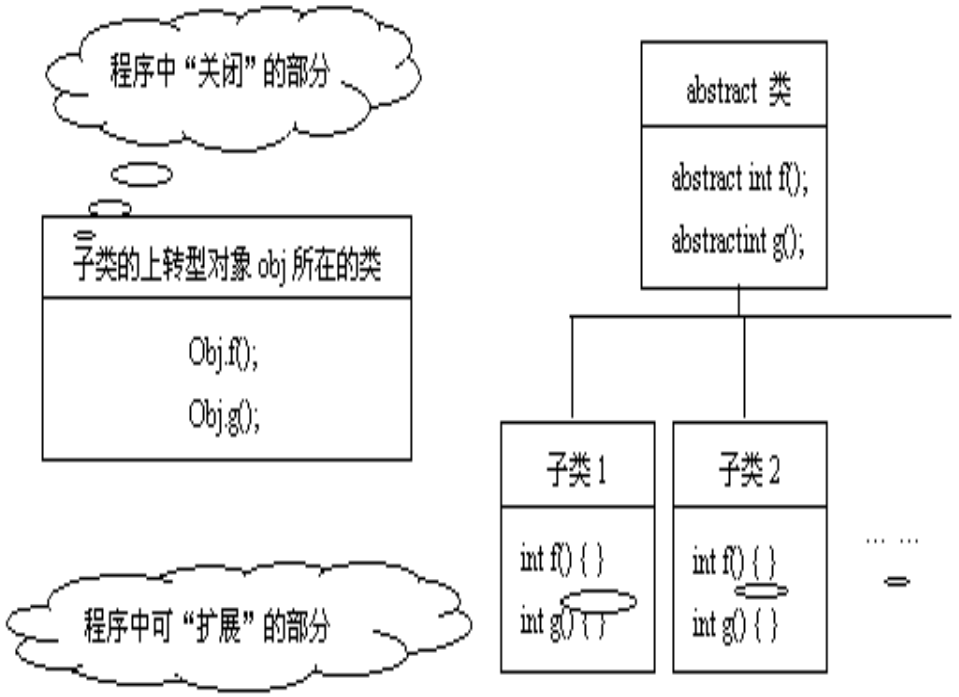


图 5.15 abstract 类与多态的使用

阶段小回顾

- Object
  - Equals（==）
  - toString
- final
  - final类不允许继承
  - Final方法防止篡改

  学过哪些final核心类?

  包装类
  System
  Math
  String, StringBuffer
  Scanner
  ……..
- abstract
  - abstract方法（也可以不包含）
  - 不可以new

本章小结

在本章中，我们主要介绍了有关继承和多态性方面的内容。

继承是面向对象程序设计的一个主要特征，是一种由已有的类创建新类的机制。由继承而得到的类称为子类或派生类，被继承的通用类称为父类、超类或基类，子类继承超类是通过在子类的声明语句后面使用关键字“extends”来体现。

本章小结

使用public修饰符声明的类为公有类，使用缺省修饰符(即没有任何修饰符)声明的类为友好类。公有类可以被包内和包外的任意类访问，即在任意类中，public类都是可见的；友好类只能被同一个包中的类访问，对同一个包中的类中是可见的。声明类成员变量和方法时，可使用的访问修饰符有四种：public、proteced、缺省和private，使用private声明的成员称为私有成员，只能在声明它们的类中使用，在类外不可见；使用public声明的成员称为公有成员，在所有可见该公有成员所属类的类中都是可以通过对象或类名直接访问；使用protected声明的成员称为受保护的成员，受保护的成员能够被同一个包的任何类访问或通过继承访问；不使用private、protected及public声明的成员称为友好成员，能够在同一个包的其它类中被所属类的对象访问或类名直接访问，而不能被任何包外类对象或类名访问。

本章小结

子类的继承性需要由类成员访问修饰符来决定，在同一个包中，子类能继承超类的所有非private成员，在不同包中，子类只能继承超类的public和protected成员。

is-a表示的是一种属于关系，是“一般和具体”的关系；而has-a表示的则是一种包含关系，是一种“整体和部件”的关系。在Java中，继承就是一种is-a关系，而聚合(组合)则是一种has-a关系。

成员变量的隐藏是指在子类中定义了与超类同名的成员变量，且这些成员变量在超类中是非私有的，此时子类的成员变量隐藏了超类的成员变量，超类的这些成员变量不能被子类继承。方法重写是指在子类中定义了一个方法，这个方法的名字、返回类型和参数声明与超类的某个方法完全相同，并且超类的这个方法是非私有的，此时超类的这个方法被子类隐藏而不能被子类继承，称这时子类的这个方法覆盖(override)或重写了超类的同名方法。

本章小结

super关键字可在子类中用来表示对直接超类的引用，可以使用它来访问在子类中被隐藏了的超类的成员变量和被重写的超类方法以及使用super调用超类的构造方法。super访问被隐藏的成员：super.成员名；调用超类构造方法的格式是：super(参数列表)。在子类的构造方法中，如果没有显式使用super关键字调用超类的某个构造方法，则系统会默认地在子类中执行super()语句；在子类中super通过参数来匹配调用超类的构造方法，所以，使用super调用超类构造方法时，必须保证超类中定义了相对应的构造方法。

在Java中，允许多层继承，即一个子类又可以是其它类的超类，从而可以形成类的多级继承层次结构。在多级继承层次结构中，构造方法的执行顺序是：首先按创建时的顺序，从上往下执行超类的构造方法对继承来的成员变量赋值，然后由子类的构造方法对自己定义的成员变量赋值。

本章小结

可以使用final关键字声明类、成员变量和方法，根据final关键字出现的位置的不同，final关键字分别具有阻止类的继承、阻止方法的重写和创建常量三种功能。使用final声明的类不能具有子类；使用final声明的方法不能被子类重写；使用final声明变量时，需要同时给变量赋值，此后，不能再更改该值。

本章小结

实现对象的多态性有编译时多态性和运行时多态性两种途径。编译时多态性也称静态多态性，表现为方法重载和变量的隐藏；运行时多态性也称动态多态性，表现为方法的重写，动态多态性的实现可以通过向上转型对象调用各子类重写的方法，使得运行后，各子类对象可以得到彼此不同的功能行为。

在Java中，所有类都默认继承自java.lang.Object类，编程人员创建的任何类都均是Object类的直接或间接子类。equals()、hashCode()和toString()三种方法是Object类提供的在其它类中比较常用的方法。equals()方法用于比较两个对象的引用是否相同，可以重写它来检查对象中存储的值。Java中创建的每一个对象都有一个对应的哈希代码，这个代码是作为对象在内存中的唯一标识，可以使用hashCode()方法来查找对象的哈希代码。使用toString()方法可以获取有关对象的文本信息，Object类中的toString()方法的实现给出了由类全限定性名称和@以及对象的十六进制的哈希代码所组成的一串文本信息，可以重写此方法，以获得我们所需的任何方式来提供对象的有关文本信息。