

程序员书库

初学者的入门宝典，程序员的百科全书



CD-ROM

10小时多媒体视频讲解

本书特色

- ※ 起点低，即使没有任何编程经验，也能通过本书掌握Java
- ※ 避免大段理论讲解，而是通过大量实例进行讲解，有很强的实践性
- ※ 对代码进行了详细注释，阅读起来非常容易，没有任何障碍
- ※ 通过现实中的事物类比Java中的概念，使读者可以很容易理解
- ※ 重点讲解Java语言的基础知识和应用，并对一些设计模式也有所介绍
- ※ 全书提供190个实例和2个综合案例，非常实用

Java

从入门到精通

高宏静 等编著



化学工业出版社

第 8 章 面向对象编程

Java 作为一门完全的面向对象语言有它本身的特性，在前面的讲解中，重点主要放在了知识点的讲解上，一些代码的编写并不符合 Java 的规范。在这一章中主要对 Java 的一些编程规范以及技巧进行简单的讲解。

8.1 封装性

作为一门完全的面向对象语言，封装是 Java 的三大特征之一。面向对象语言的优点之一就是可以写一个类给别人使用，但是在类中并不是所有的信息都能让用户可见的，因此隐藏这些信息就显得很有必要，这就是面向对象对封装的要求。Java 的封装是通过 `private`、`protected` 和 `public` 来实现的。

8.1.1 成员变量的封装

在前面的程序中，成员变量都为 `public` 或是默认修饰符的。实际上这是违反面向对象思想的，在面向对象编程中有一原则，即尽量让数据私有。

也许读者会有疑问如果数据私有的话，怎么访问这些成员变量呢？答案是通过方法来访问。下面是一个简单的实例，用标准的形式实现前面的 `Human` 类，代码如下。

```
class Human
{
    //成员属性都是私有的
    private String name;
    private String sex;
    private int age;
    private String addr;
    //public 的设置和访问方法
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    //省略其他变量的 get 和 set 方法
    public void work() {
        System.out.println("我在工作");
    }
}
```

```
public void eat() {
```

```

        System.out.println("我在吃饭");
    }
}

```

在程序中类 `Human` 的 4 个属性 `name`、`age`、`sex`、`addr` 都被声明为 `private`，对每个属性都提供了一个 `set` 方法和一个 `get` 方法用来设置和获取相应属性的值，并且都声明为 `public`。显然其他的类直接访问这些属性是不行的，而应该使用类提供的方法来取值和赋值。示例如下。

```

public class HumanDemo {
    public static void main(String[] args) {
        Human zhangsan=new Human();
        //zhangsan.name="张三";           这个语句是错误的
        //设置成员变量的值
        zhangsan.setName("张三");
        zhangsan.setAge(25);
        zhangsan.setSex("男");
        zhangsan.setAddr("中国北京");
        System.out.println("张三的个人信息如下: ");
        //打印出对象的信息，用 get 方法获得属性值
        System.out.println("姓名: "+zhangsan.getName());
        System.out.println("性别: "+zhangsan.getSex());
        System.out.println("年龄: "+zhangsan.getAge());
        System.out.println("地址: "+zhangsan.getAddr());
    }
}

```

在上面的程序中首先声明一个 `Human` 对象 `zhangsan`，可以看到是不能直接使用 `zhangsan.name` 的形式访问成员变量的，必须使用类提供的 `set` 和 `get` 方法来进行赋值和取值。程序的运行结果如下。

```

张三的个人信息如下:
姓名: 张三
性别: 男
年龄: 25
地址: 中国北京

```

为许多成员变量编写 `set` 和 `get` 方法是一件繁琐的事情，`eclipse` 提供了一个功能可以自动地对成员变量生成 `set` 和 `get` 方法。下面演示它的使用，首先要在类中写好需要的成员变量。

```

class Human {
    private String name;
    private String sex;
    private int age;
    private String addr;
}

```

在程序编码区右击，选择“源代码”→“生成 Getter 和 Setter”命令，如图 8-1 所示，选择之后弹出属性选择窗口，如图 8-2 所示。选择要生成 `set`、`get` 方法的属性，单击“确定”按钮即可。

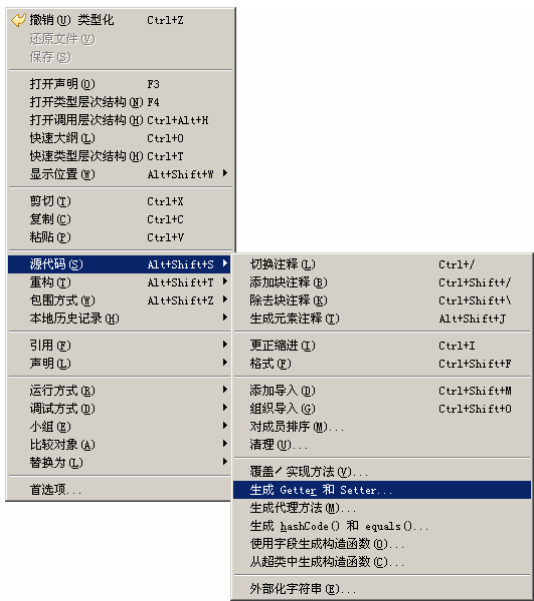


图 8-1 生成 getter 和 setter



图 8-2 属性选择窗口

8.1.2 成员变量的继承

在继承一章中主要讲了方法的继承，实际上成员变量也是可以实现继承的。成员变量能否被继承，完全取决于它的修饰符。

1. public 成员变量

对于 public 的成员变量，它的任何子类都可以继承它。下面是一个演示程序，首先定义一个类 FatherClass，在类中有一个 public 的 String 类型变量，子类 SonClass 继承自 FatherClass 类，另外还包括一个演示类。3 个类的代码如下。

```
public class FatherClass
{
    public String str1="父类的 public 属性";
}
class SonClass extends FatherClass
{
    void print(){
        SonClass son=new SonClass();
        System.out.println("在子类的方法中: ");
        //子类对象继承父类的属性 str1
        System.out.println(son.str1);
    }
}
public class Demo
{
    public static void main(String[ ] args) {
        //创建子类对象
        SonClass son=new SonClass();
    }
}
```

```

        //调用相关方法
        son.print();
    }
}

```

程序的运行结果如下。

在子类的方法中：
父类的 public 属性

2. protected 成员变量

protected 类型的变量它的子类可以访问，不管该子类与父类是否处于同一个包内。下面是演示程序，注意它们在不同的包内。

```

package chapter11;
public class FatherClass
{
    public String str1="父类的 public 属性";
    protected String str2="父类的 protected 属性";
}
//注意程序所在的包是不同的
package anotherPak;
import chapter11.FatherClass;
public class SonClass2 extends FatherClass {
    void print(){
        SonClass2 son=new SonClass2();
        System.out.println("在子类的方法中：");
        System.out.println(son.str1);
    }
}
package anotherPak;
public class Demo2 {
    public static void main(String[ ] args) {
        SonClass2 son2=new SonClass2();
        son2.print();
    }
}

```

程序的运行结果如下。

在子类的方法中：
父类的 protected 属性

可以看到 protected 属性可以被子类继承，而不管该子类是与父类处于同一个包中。

3. 默认修饰符成员变量

对于默认修饰符，包外的子类是不能访问它的，包内类可以继承。示例如下。

```

package chapter11;
public class FatherClass {
    public String str1="父类的 public 属性";
    protected String str2="父类的 protected 属性";
    String str3="父类默认修饰符的属性";
}
//在不同的包是不能访问的
package anotherPak;
import chapter11.FatherClass;
public class SonClass2 extends FatherClass {

```

```

        void print(){
            SonClass2 son=new SonClass2();
            System.out.println("在子类的方法中: ");
            //该语句编译错误
            //System.out.println(son.str3);
        }
    }
}
package anotherPak;
public class Demo2
{
    public static void main(String[ ] args) {
        SonClass2 son2=new SonClass2();
        son2.print();
    }
}

```

4. private 成员变量

对于 private 类型的成员变量，任何子类都是不能继承的，它是类私有的。

8.1.3 成员变量的隐藏

对成员变量的封装和继承了解后，在本节就来学一下成员变量的隐藏。成员变量既然是可以继承的，那么它也应该是被覆盖的，更准确的是可以隐藏。示例如下。

```

class Father
{
    String str="父类的 Str 成员变量";
}
class Son extends Father
{
    String str="子类的 Str 成员变量";
}
public class Hidden
{
    public static void main(String[ ] args) {
        //创建父类对象
        Father father =new Father();
        //创建子类对象
        Son son =new Son();
        //分别访问它们的成员变量
        System.out.println("父类对象访问 Str: ");
        System.out.println(father.str);
        System.out.println("子类对象访问 Str");
        System.out.println(son.str);
    }
}

```

程序的运行结果如下。

```

父类对象访问 Str:
父类的 Str 成员变量
子类对象访问 Str
子类的 Str 成员变量

```

可以看到子类访问同名的父类成员变量时候是调用的子类自己的变量，这样可以实现成员变量的隐藏。被隐藏的父类成员变量是可以访问的，通过 `super` 关键字可以实现：

```
class Father
{
    String str="父类的 Str 成员变量";
}
class Son extends Father
{
    String str="子类的 Str 成员变量";
    void show(){
        System.out.println(super.str);
    }
}
public class Hidden
{
    public static void main(String[] args) {
        //创建子类对象
        Son son =new Son();
        System.out.println("调用隐藏的父类成员变量");
        //调用子类的方法
        son.show();
    }
}
```

在子类中，可以用 “`super.`” 变量名的方式访问父类成员变量。

8.2 合理使用类

类是 Java 的强大之处，但是要适当地使用它，才能发挥它最大的作用，如何设计类的成员、操作以及类之间的关系就成了重点。本节将会介绍在类设计中要注意的问题，当然这只是简单的介绍，如何合理的设计需要在实际编码开发中不断总结。

8.2.1 合理地分解类

在使用类的时候要合理地把类进行分解，而不应该在类中放置过多的基本类型字段。例如在前面的 `Human` 类中有一个 `String` 类，用它来表示地址。但是这样的缺点是只是一个字符串，可能不同的人的地址的格式不同，在设置地址的时候也没有统一的标准。这样可以把地址单独做一个 `Address` 类，用它来存放用户信息。

```
public class Address
{
    private String country;
    private String province;
    private String city;
    private String street;
}
```

给该类加上构造函数和 `get` 以及 `set` 方法，就成为一个完整的类，用它来存放地址信息的时候，地址就有了统一的格式。

8.2.2 让类的名字和方法反映它的作用

在前面讲变量时提到一点，变量名应该尽量能反映它表示的东西。类名和方法也应该这样来设计。在前面的编码中已经尽量做到这一点，如 `Human` 类、`Animal` 类、`Tiger` 类以及 `Fish` 类都是使用了名词来表示相关的类（注意这些类名都是大写字母开头的）。方法应该尽量用动词或者动词与名词的组合来表示，如 `move()`、`setName()` 等。

8.2.3 复用现有的类

面向对象带来重要的改进就是代码的可重用性。`Java` 中提供了许多已有的类，这些类都可以拿来用，不必自己去重新写，况且写出来的代码效率也不一定高，使用已经提供的类，可以极大地提高开发效率。

代码的重用性进一步改进了程序的可维护性，当一个类中有 `bug` 的时候，改进这一个类就可以把整个系统中的 `bug` 改正。而如果没有代码的重用，这段代码出现多少次，恐怕就得修改多少次，这在大系统中实施的困难是难以想象的。下面的程序使用 `Java` 提供 `GregorianCalendar` 类编来写一个日历。

```
import java.util.*;
public class MyCalendar {
    public static void main(String[] args) {
        GregorianCalendar now = new GregorianCalendar();
        //获得一个 Date 对象
        Date date = new Date();
        //打印出 date
        System.out.println(date.toString());
        //用 date 设置 now 的时间
        now.setTime(date);
        //从 now 中取出当前的日期、月份
        int today = now.get(Calendar.DAY_OF_MONTH);
        int month = now.get(Calendar.MONTH);
        //设置 now 的日期设为 1
        now.set(Calendar.DAY_OF_MONTH, 1);
        //得到 now 是一周的第几天
        int week = now.get(Calendar.DAY_OF_WEEK);
        //打印出日历头
        System.out.println("Sun   Mon   Tue   Wed   Thu   Fri   Sat");
        //打印出前面的空格
        for (int i = Calendar.SUNDAY; i < week; i++) {
            System.out.print("    ");
        }
        while (now.get(Calendar.MONTH) == month) {
            int day = now.get(Calendar.DAY_OF_MONTH);
            //为了对齐要对大于 10 和小于 10 的数打印不同空格数
            if (day < 10) {
```



```

        if (day == today)
            System.out.print("-" + day + "- ");
        else
            System.out.print(" "+day+" ");
    }
    else
    {
        if (day==today)
            System.out.print("-" + day + "- ");
        else
            System.out.print(" "+day+" ");
    }

    //周六换行
    if (week == Calendar.SATURDAY) {
        System.out.println();
    }
    //增加一天
    now.add(Calendar.DAY_OF_MONTH, 1);
    week = now.get(Calendar.DAY_OF_WEEK);
}
}
}

```

程序的运行结果如下。

```

Mon Sep 29 15:39:55 CST 2008
Sun  Mon  Tue  Wed  Thu  Fri  Sat
 1    2    3    4    5    6    7
 8    9   10   11   12  -13-  14
15   16   17   18   19   20   21
22   23   24   25   26   27   28
29   30   31

```

8.3 继承与组合的使用

继承是子类与父类之间的关系，这种关系是 is-a 的关系，即子类是父类的一种。例如前面讲到的父类 `Animal`，它的子类 `Tiger`、`Fish` 都是 `Animal`。组合是另一种技术，它是指一个类中基本类型外还有其他的类的对象作为它的成员，它们是 has-a 的关系。

8.3.1 如何使用继承

继承作为面向对象的重要特征之一，自然是面向对象的强大所在，合理的设计继承才能发挥它的强大威力。

在对事物进行抽象的时候要对该事物的属性和操作进行分析，想明白它隶属于哪一类，即它 is-a 什么。下面是继承设计中一些要注意的方面。

把子类都有的操作和属性放在超类中，其他的操作和属性放在子类中。例如前面用到类 `Animal`，它的合理设计如下：

```

class Animal
{
    private String type;
    private String name;
    private int age;
    private int weight;
    Animal(){
        type=null;
        name=null;
        age=0;
        weight=0;
    }
    Animal(String type,String name,int age,int weight){
        this.type=type;
        this.name=name;
        this.age=age;
        this.weight=weight;
    }
    public void eat(){
        System.out.println("animal eat");
    }
    public void breath(){
        System.out.println("animal breath");
    }
    public void sleep(){
        System.out.println("animal sleep");
    }
}

```

它的子类任何动物不管是 Tiger 类还是 Fish 类可以有名字（age）、类型（type）、年龄（age）和体重（weight），都有操作吃饭（eat）、呼吸（breath）和睡觉（sleep）。而对于在子类 Tiger 中可以定义一个操作奔跑（run），由于并不是所有的动物都会奔跑，所以不适合把 run 放入 Animal 中。

把成员变量设为私有的而不使用 protected 和 public 的。把成员变量设为 private，并对每个成员变量添加 set 和 get 方法；否则的话通过继承，子类就可以访问父类的属性，这有违面向对象的封装性。

覆盖方法的时候应该不违背原来的定义。在覆盖父类的方法时应该尽量保持原来的操作，例如在 Animal 类中的 breath 方法，在 Tiger 类中重写的时候就要实现老虎的呼吸动作，而不应该实现它的吃饭操作。

继承的思想是十分复杂的，以上几点只是继承设计需要注意的比较明显的几点，在实际的设计中会更加复杂，继承的合理设计是十分重要的，读者可以在实际的开发中慢慢体会。

8.3.2 组合

组合是指在一个类中除了基本类型外还有其他类的对象作为类的属性存在。其实使用 String 类就可以看做组合，因为 Java 中的 String 是作为类来实现的。

```

public class Address
{
    private String country;
}

```

```

private String province;
private String cicy;
private String street;
Address(String country,String province,String city,String street){
    this.country=country;
    this.province=province;
    this.cicy=city;
    this.street=street;
}
//本类的 toString 信息，返回对象的信息
public String toString() {
    return country+"."+province+"."+cicy+"."+street;
}
}
public class Human {
    private String name;
    private String sex;
    private int age;
    private Address addr;
    Human(String name,String sex,int age,Address addr){
        this.name=name;
        this.sex=sex;
        this.age=age;
        this.addr=addr;
    }
    //本类的 toString 信息，返回对象的信息
    public String toString() {

        return "姓名: "+name+" 性别: "+sex+" 年龄: "+age+" 地址: "+addr.toString();
    }
}

```

在 `Human` 类中，用一个单独的类来描述人的地址信息。示例如下。

```

public class HumanDemo {
    public static void main(String[] args) {
        //产生一个 Address 对象
        Address addr=new Address("中国","山东省","青岛市","XX 街");
        //产生一个 Human 对象
        Human lisi=new Human("李四","男",24,addr);
        System.out.println("李四的个人信息: ");
        System.out.println(lisi.toString());
    }
}

```

程序的运行结果如下。

```

李四的个人信息:
姓名: 李四 性别: 男 年龄: 24 地址: 中国:山东省:青岛市:XX 街

```

8.3.3 继承与组合比较

组合通常用于在新类中使用已有类的功能，而不是它的接口的情形，也就是把对象做为

新类的成员变量，用它来实现所需要的功能。

举个例子，用一个类来表示电脑。电脑都有 CPU、主板、硬盘、内存和显示器等配件。这样可以分别为 CPU、主板、硬盘、内存和显示器等配件设计相应的类，最后有一个电脑类，把前面设计的类的对象作为电脑类的成员变量。

现在又需要一个类来表示联想的某一类型电脑，这个类就可以继承子电脑类，而不是新设计一个类，这个新类由电脑类的组成所组成，并能进一步设计自己的细节部分。

8.4 小结

在本章中主要介绍了面向对象程序设计的一些注意点。面向对象不仅仅是程序设计语言的修饰词，它更是一种思想，这种思想需要在实际的学习中慢慢体会，本章只是抛砖引玉，希望读者在学习中细细体会。另外，Java 提供了一些类，通过它们的源代码学习面向对象设计也是一种很好的方法。