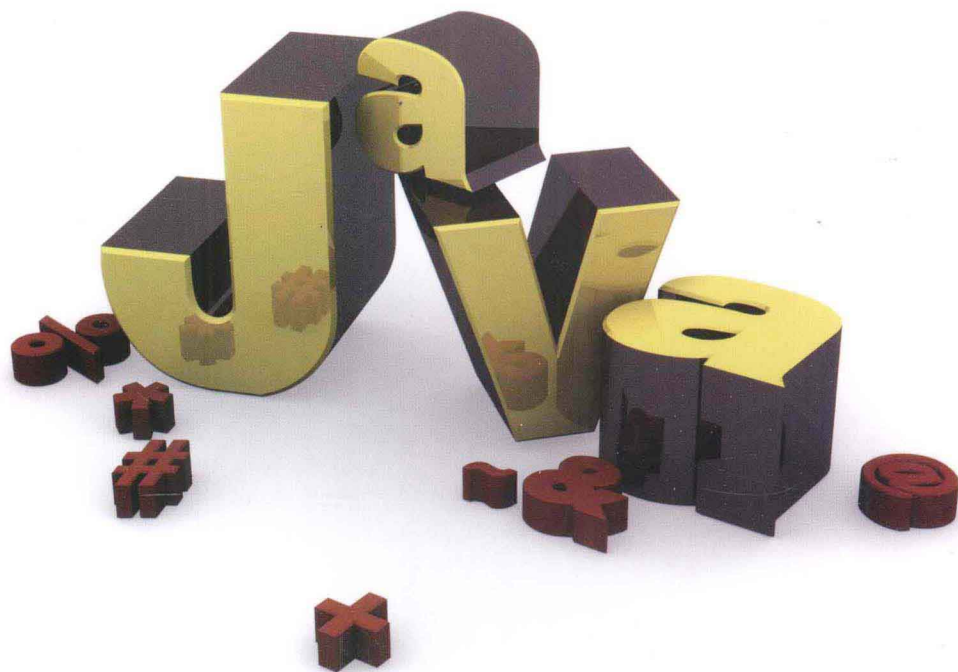


Java 高等学校Java课程系列教材

面向对象与设计模式

耿祥义 张跃平 著



清华大学出版社

高等学校 Java 课程系列教材

面向对象与设计模式

耿祥义 张跃平 著

清华大学出版社
北 京

内 容 简 介

本书是面向有一定 Java 语言基础和一定编程经验的读者, 重点介绍了面向对象的核心内容以及作者在面向对象研究中的一些新思想, 全面探讨在 Java 程序设计中怎样使用一些重要的设计模式。作者编写本书的目的是让读者不仅学习怎样在软件设计中使用好设计模式, 更重要的是让读者通过学习使用设计模式深刻地理解面向对象的设计思想, 以便更好地使用面向对象语言解决设计中的诸多问题。

本书可以作为计算机相关专业研究生或高年级学生的教材, 也可以作为软件项目管理人员、软件开发工程师等专业人员的参考用书。可登录 www.tup.com.cn 下载书中的示例代码。

本书封面贴有清华大学出版社防伪标签, 无标签者不得销售。
版权所有, 侵权必究。侵权举报电话: 010-62782989 13701121933

图书在版编目(CIP)数据

面向对象与设计模式 / 耿祥义, 张跃平著. —北京: 清华大学出版社, 2013.6

高等学校 Java 课程系列教材

ISBN 978-7-302-30823-2

I. ①面… II. ①耿… ②张… III. ①JAVA 语言-程序设计 IV. ①TP312

中国版本图书馆 CIP 数据核字 (2012) 第 287864 号

责任编辑: 魏江江 王冰飞

封面设计: 杨 兮

责任校对: 李建庄

责任印制: 宋 林

出版发行: 清华大学出版社

网 址: <http://www.tup.com.cn>, <http://www.wqbook.com>

地 址: 北京清华大学学研大厦 A 座 邮 编: 100084

社 总 机: 010-62770175 邮 购: 010-62786544

投稿与读者服务: 010-62776969, c-service@tup.tsinghua.edu.cn

质 量 反 馈: 010-62772015, zhiliang@tup.tsinghua.edu.cn

印 装 者: 北京鑫海金澳胶印有限公司

经 销: 全国新华书店

开 本: 185mm×260mm 印 张: 18.25 字 数: 457 千字

版 次: 2013 年 6 月第 1 版 印 次: 2013 年 6 月第 1 次印刷

印 数: 1~3000

定 价: 29.50 元

产品编号: 050593-01

前言

目前，面向对象程序设计已经成为软件设计开发领域的主流，而学习使用设计模式无疑非常有助于软件开发人员使用面向对象语言开发出易维护、易扩展、易复用的代码，其原因是设计模式是从许多优秀的软件系统中总结出的成功的可复用的设计方案，已被成功应用于许多系统的设计中。本书是面向有一定 Java 语言基础和一定编程经验的读者，重点介绍了面向对象的核心内容，全面探讨在程序设计中怎样使用一些重要的设计模式。作者编写本书的目的是让读者不仅学习怎样在软件设计过程中使用好设计模式，更重要的是让读者通过学习使用设计模式深刻地理解面向对象的设计思想，以便更好地使用面向对象语言解决设计中的诸多问题。

本书共 26 章，前 6 章重点介绍了面向对象的核心内容，以及作者在面向对象研究中的一些新思想；第 7 章～第 26 章探讨在程序设计中怎样使用一些重要的设计模式。为了说明一个模式的核心实质，本书精心研究了针对每个模式的示例，以便让读者结合这样的示例更好地理解和使用模式。本书的全部示例由作者编写完成，本书示例代码及相关内容仅供学习设计模式使用。希望本书能对读者学习和使用设计模式有所帮助，并请读者批评指正。

作 者

2013 年 4 月

目 录

第 1 章 面向对象入门	1
1.1 编程语言的几个发展阶段	1
1.1.1 面向机器语言	1
1.1.2 面向过程语言	1
1.1.3 面向对象语言	2
1.1.4 使用 Java 的必要性	3
1.2 从抽象到类	3
1.3 类与程序的基本结构	5
1.4 使用的开发工具	6
第 2 章 封装	7
2.1 从类的角度看封装性	7
2.1.1 封装数据及操作	7
2.1.2 类的 UML 图	8
2.2 从对象的角度看封装性	9
2.2.1 创建对象所体现的封装性	9
2.2.2 对象访问变量和调用方法所体现的封装性	11
2.2.3 实例变量和类变量体现的封装性	12
2.2.4 关于实例方法和类方法	13
2.3 从使用者角度看封装性	14
2.4 有理数的类封装	15
2.4.1 Rational 类	15
2.4.2 用 Rational 对象做运算	18
2.5 从访问权限看封装性	19
2.5.1 访问限制修饰符	19
2.5.2 加强封装性	21
2.6 包与类的封装	22
2.6.1 包封装	22
2.6.2 引入类库中的类	24
2.6.3 引入自定义包中的类	24

第3章 继承、接口与多态	26
3.1 子类与父类	26
3.2 子类的继承性	27
3.3 关于 protected 的进一步说明	27
3.4 子类对象的特点	28
3.5 隐藏继承的成员	29
3.6 通过重写实现多态	30
3.7 上转型对象体现多态	31
3.8 通过 final 禁止多态	33
3.9 通过 super 解决多态带来的问题	33
3.10 接口	34
3.11 接口回调体现的多态	36
3.12 重载体现的多态	38
第4章 组合	41
4.1 引用的重要作用	41
4.2 对象的组合	44
4.3 组合关系和依赖关系	46
4.4 组合关系是弱耦合关系	47
4.5 基于组合的流水线	48
4.6 汽车动态更换驾驶员	50
第5章 面向对象的几个基本原则	53
5.1 抽象类与接口	53
5.1.1 抽象类	53
5.1.2 接口	54
5.2 面向抽象原则	55
5.3 “开-闭”原则	57
5.3.1 什么是“开-闭”原则	57
5.3.2 标准、标准构件和面向标准的产品	58
5.4 “多用组合，少用继承”原则	61
5.4.1 继承与复用	61
5.4.2 组合与复用	62
5.4.3 多用组合，少用继承	62
5.5 “高内聚-弱耦合”原则	63
第6章 设计模式简介	64
6.1 什么是设计模式	64

6.2	设计模式的起源	65
6.3	GOF 之著作	65
6.4	学习设计模式的重要性	65
6.5	合理使用模式	66
6.6	什么是框架	67
6.7	模式分类	68
6.8	设计模式资源	69
第 7 章	策略模式	70
7.1	策略模式的结构与使用	70
7.1.1	策略模式的结构	70
7.1.2	策略模式的使用	74
7.2	策略模式的优点	75
7.3	适合使用策略模式的情景	75
7.4	策略模式相对继承机制的优势	76
7.5	举例——加密、解密文件	76
7.5.1	设计要求	76
7.5.2	设计实现	76
第 8 章	状态模式	82
8.1	状态模式的结构与使用	82
8.1.1	状态模式的结构	82
8.1.2	状态模式的使用	85
8.2	状态切换	86
8.3	状态模式的优点	89
8.4	适合使用状态模式的情景	89
8.5	举例——模拟咖啡自动售货机	89
8.5.1	设计要求	89
8.5.2	设计实现	90
第 9 章	命令模式	93
9.1	命令模式的结构与使用	93
9.1.1	命令模式的结构	93
9.1.2	命令模式的使用	97
9.2	命令接口中的撤销方法	98
9.3	命令模式的优点	101
9.4	适合使用命令模式的情景	101
9.5	举例——控制电灯	101
9.5.1	设计要求	101

9.5.2 设计实现	101
第 10 章 中介者模式	105
10.1 中介者模式的结构与使用	105
10.1.1 中介者模式的结构	105
10.1.2 中介者模式的使用	108
10.2 中介者模式的优点	110
10.3 适合使用中介者模式的情景	110
10.4 举例——组件交互	110
10.4.1 设计要求	110
10.4.2 设计实现	111
第 11 章 责任链模式	114
11.1 责任链模式的结构与使用	114
11.1.1 责任链模式的结构	114
11.1.2 责任链模式的使用	118
11.2 责任链模式的优点	118
11.3 适合使用责任链模式的情景	119
11.4 举例——计算阶乘	119
11.4.1 设计要求	119
11.4.2 设计实现	119
第 12 章 模板方法模式	123
12.1 模板方法模式的结构与使用	123
12.1.1 模板方法模式的结构	123
12.1.2 模板方法模式的使用	126
12.2 钩子方法	127
12.3 模板方法模式的优点	130
12.4 适合使用模板方法模式的情景	130
12.5 举例——考试与成绩录入	130
12.5.1 设计要求	130
12.5.2 设计实现	131
第 13 章 观察者模式	135
13.1 观察者模式的结构与使用	135
13.1.1 观察者模式的结构	135
13.1.2 观察者模式的使用	140
13.2 观察者模式中的“推”数据与“拉”数据	141
13.3 观察者模式的优点	144

13.4	适合使用观察者模式的情景	145
13.5	举例——关注天气和旅游信息	145
13.5.1	设计要求	145
13.5.2	设计实现	145
第 14 章	访问者模式	150
14.1	访问者模式的结构与使用	150
14.1.1	访问者模式的结构	150
14.1.2	访问者模式的使用	153
14.2	双重分派	154
14.3	访问者模式的优点	154
14.4	适合使用访问模式的情景	155
14.5	举例——评价体检表	155
14.5.1	设计要求	155
14.5.2	设计实现	155
第 15 章	装饰模式	160
15.1	装饰模式的结构与使用	160
15.1.1	装饰模式的结构	160
15.1.2	装饰模式的使用	163
15.2	使用多个装饰者	164
15.3	装饰模式相对继承机制的优势	165
15.4	装饰模式的优点	166
15.5	适合使用装饰模式的情景	166
15.6	举例——读取单词表	166
15.6.1	设计要求	166
15.6.2	设计实现	167
第 16 章	组合模式	171
16.1	组合模式的结构与使用	171
16.1.1	组合模式的结构	171
16.1.2	组合模式的使用	174
16.2	组合模式的优点	176
16.3	适合使用组合模式的情景	176
16.4	举例——苹果树的重量及苹果的价值	176
16.4.1	设计要求	176
16.4.2	设计实现	177

第 17 章 适配器模式	180
17.1 适配器模式的结构与使用	180
17.1.1 适配器模式的结构	180
17.1.2 适配器模式的使用	182
17.1.3 适配器的适配程度	183
17.2 适配器模式的优点	184
17.3 适合使用适配器模式的情景	184
17.4 单接口适配器	184
17.5 举例——Iterator 接口与 Enumeration 接口	185
17.5.1 设计要求	185
17.5.2 设计实现	185
第 18 章 外观模式	188
18.1 外观模式的结构与使用	188
18.1.1 外观模式的结构	188
18.1.2 外观模式的使用	190
18.2 外观模式的优点	191
18.3 适合使用外观模式的情景	191
18.4 举例——解析文件	191
18.4.1 设计要求	191
18.4.2 设计实现	192
第 19 章 代理模式	195
19.1 代理模式的结构与使用	195
19.1.1 代理模式的结构	195
19.1.2 代理模式的使用	197
19.2 远程代理	198
19.2.1 RMI 与代理模式	198
19.2.2 RMI 的设计细节	199
19.3 代理模式的优点	203
19.4 适合使用代理模式的情景	203
19.5 举例——使用远程窗口阅读文件	203
19.5.1 设计要求	203
19.5.2 设计实现	203
第 20 章 享元模式	207
20.1 享元模式的结构与使用	207
20.1.1 享元模式的结构	207

20.1.2	享元模式的使用	212
20.2	享元模式的优点	213
20.3	适合使用享元模式的情景	213
20.4	举例——化合物	213
20.4.1	设计要求	213
20.4.2	设计实现	214
第 21 章	桥接模式	218
21.1	桥接模式的结构与使用	218
21.1.1	桥接模式的结构	218
21.1.2	桥接模式的使用	221
21.2	桥接模式的优点	222
21.3	适合使用桥接模式的情景	222
21.4	举例——模拟电视节目	222
21.4.1	设计要求	222
21.4.2	设计实现	222
第 22 章	工厂方法模式	226
22.1	工厂方法模式的结构与使用	226
22.1.1	工厂方法模式的结构	226
22.1.2	工厂方法模式的使用	229
22.2	工厂方法模式的优点	230
22.3	适合使用工厂方法模式的情景	231
22.4	举例——药品	231
22.4.1	设计要求	231
22.4.2	设计实现	231
第 23 章	抽象工厂模式	235
23.1	抽象工厂模式的结构与使用	235
23.1.1	抽象工厂模式的结构	235
23.1.2	抽象工厂模式的使用	240
23.2	工厂方法模式的优点	241
23.3	适合使用抽象工厂模式的情景	242
23.4	举例——商店与西服、牛仔服	242
23.4.1	设计要求	242
23.4.2	设计实现	242
23.4.3	模式的使用	245
第 24 章	生成器模式	247
24.1	生成器模式的结构与使用	247

24.1.1	生成器模式的结构	247
24.1.2	生成器模式的使用	250
24.2	生成器模式的优点	251
24.3	适合使用生成器模式的情景	252
24.4	举例——日历牌	252
24.4.1	设计要求	252
24.4.2	设计实现	252
第 25 章	原型模式	259
25.1	java.lang.Object 类的 clone 方法	259
25.2	Serializable 接口与克隆对象	263
25.3	原型模式的结构与使用	263
25.3.1	原型模式的结构	263
25.3.2	原型模式的使用	265
25.4	原型模式的优点	266
25.5	适合使用原型模式的情景	266
25.6	举例——克隆容器	267
25.6.1	设计要求	267
25.6.2	设计实现	267
第 26 章	单件模式	270
26.1	单件模式的结构与使用	270
26.1.1	单件模式的结构	270
26.1.2	单件模式的使用	272
26.2	单件模式的优点	273
26.3	适合使用单件模式的情景	273
26.4	举例——冠军	273
26.4.1	设计要求	273
26.4.2	设计实现	273
参考文献		278

本书是面向有一定 Java 语言基础和一定编程经验的读者，因此读者在学习本书之前应当有一定的 Java 语言基础，熟悉一种 Java 开发工具（如 Java SE 提供的 JDK）。

1.1 编程语言的几个发展阶段

1.1.1 面向机器语言

每种计算机都有自己独特的机器指令，比如，某种型号的计算机用 8 位二进制信息 10001010 表示加法指令，用 00010011 表示减法指令，等等。这些指令的执行由计算机的线路来保证，计算机在设计之初，事先就要确定好每一条指令对应的线路逻辑操作。计算机处理信息的早期语言是所谓的机器语言，使用机器语言进行程序设计需要面向机器来编写代码，即需要针对不同的机器编写诸如 01011100 这样的指令序列。用机器语言进行程序设计是一项累人的工作，代码难以阅读和理解，一个简单的任务往往蕴含着编写大量的代码，而且同样的任务需要针对不同型号的计算机分别进行编写指令，因为一种型号的计算机用 10001010 表示加法指令，而另一种型号的计算机可能用 11110000 表示加法指令。因此，使用机器语言编程也称为面向机器编程。20 世纪 50 年代出现了汇编语言，在编写指令时，用一些简单的容易记忆的符号代替二进制指令，但汇编语言仍是面向机器语言，需针对不同的机器编写不同的代码。习惯上称机器语言、汇编语言是低级语言。

1.1.2 面向过程语言

随着计算机硬件功能的提高，在 20 世纪 60 年代出现了过程设计语言，如 C 语言、FORTRAN 语言等。用这些语言编程也称为面向过程编程，语言把代码组成叫做过程或函数的块。每个块的目标是完成某个任务，例如，一个 C 的源程序就是由若干个书写形式互相独立的函数组成。使用这些语言编写代码指令时，不必再去考虑机器指令的细节，只需按照具体语言的语法要求去编写“源文件”。所谓源文件，就是按照编程语言的语法编写具有一定扩展名的文本文件，比如，C 语言编写的源文件的扩展名是 c，FORTRAN 语言编写的源文件的扩展名是 for 等。过程语言的源文件的一个特点是更接近人的“自然语言”，比如，C 语言源程序中的一个函数：

```
int max(int a,int b) {  
    if(a>b)  
        return a;  
    else
```

```
        return b;  
    }
```

该函数负责计算两个整数的最大值。过程语言的语法更接近人们的自然语言，人们只需按照自己的意图编写各个函数，习惯上称过程语言为高级语言。

随着软件规模的扩大，过程语言在解决实际问题时逐渐显露出力不从心。对于许多应用型问题，人们希望编写出易维护、易扩展和易复用的程序代码，而使用过程语言很难做到这一点。面向过程语言的核心是编写解决某个问题的代码块，比如 C 语言中的函数，代码块是程序执行时产生的一种行为，但是面向过程语言却没有为这种行为指定“主体”，即在程序运行期间，无法说明到底是“谁”具有这个行为，并负责执行了这个行为。比如，C 语言编写了一个“刹车”函数，却无法指定是哪个“实体”具有这样的行为。也就是说，面向过程语言缺少了一个概念，那就是“对象”。现实生活中，“行为”往往归结为某个具体的“主体”所拥有，即某个对象所拥有，并且该对象负责产生这样的行为。和面向过程语言不同的是，在面向对象语言中，核心的内容就是“对象”，一切围绕着对象，比如，编写一个“刹车”方法（面向过程称之为函数），那么一定会指定该方法的“主体”，比如，某个汽车拥有这样的“刹车”方法，该汽车负责执行“刹车”方法产生相应的行为。

1.1.3 面向对象语言

随着计算机硬件设备功能的进一步提高，使得基于对象的编程成为可能（面向对象语言编写的程序需要消耗更多的内存，需要更快的 CPU 保证其运行速度）。基于对象的编程更加符合人的思维模式，使用面向对象语言可以编写易维护、易扩展和易复用的程序代码，更重要的是，面向对象编程鼓励创造性的程序设计。

面向对象编程主要体现下列 3 个特性。

1. 封装性

面向对象编程的核心思想之一就是将数据和对数据的操作封装在一起。通过抽象，即从具体的实例中抽取共同的性质形成一般的概念，比如，类的概念。在实际生活中，人们每时每刻都与具体的实物在打交道，例如，我们用的钢笔、骑的自行车、乘的公共汽车等。人们经常见到的卡车、公共汽车、轿车等都会涉及以下几个重要的属性：可承载的人数、运行速度、发动机的功率、耗油量、自重、轮子数目等。另外，还有几个重要的行为（功能）：加速、减速、刹车、转弯等。可以把这些行为称做是它们具有的方法，而属性是它们的状态描述，仅仅用属性或行为不能很好地描述它们。在现实生活中，用这些共有的属性和行为给出一个概念——机动车类。也就是说，人们经常谈到的机动车类就是从具体的实例中抽取共同的属性和行为形成的一个概念，那么一个具体的轿车就是机动车类的一个实例，即对象。一个对象将自己的数据和对这些数据的操作合理有效地封装在一起，例如，每辆轿车调用“减速”行为改变的都是自己的运行速度。

2. 继承

继承体现了一种先进的编程模式（见第 3 章）。子类可以继承父类的属性和行为，即继承父类所具有的数据和数据上的操作，同时又可以增添子类独有的数据和数据上的操作。比如，“人类”自然继承了“哺乳类”的属性和行为，同时又增添了人类独有的属性和行为。

3. 多态

多态是面向对象编程的又一重要特征。有两种意义的多态。一种是操作名称的多态，即有多个操作具有相同的名字，但这些操作所接收的消息类型必须不同。例如，让一个人执行“求面积”操作时，他可能会问你求什么面积。所谓操作名称的多态性，是指可以向操作传递不同消息，以便让对象根据相应的消息来产生相应的行为。另一种是和继承有关的多态，是指同一个操作被不同类型对象调用时可能产生不同的行为。例如，狗和猫都具有哺乳类的功能：“喊叫”。但是，狗操作“喊叫”产生的声音是“汪汪……”；而猫操作“喊叫”产生的声音是“喵喵……”。

1.1.4 使用 Java 的必要性

本书是面向有一定 Java 语言基础和一定编程经验的读者，因此读者在学习本书之前应当有一定的 Java 语言基础，熟悉一种 Java 开发工具（如 Java SE 提供的 JDK）。和 C++ 不同，Java 语言是纯面向对象编程的语言，不支持面向过程，并且更适合用来实现面向对象的程序设计。GOF 所著作的《设计模式》一书无疑是经典之作，但是该书中的示例代码相当简练，而且是采用 C++ 描述的，另外 C++ 中提到的接口就是指类的方法，但是在 Java 中，类和接口是两个不同的概念。本书采用 Java 语言讲解面向对象和设计模式的另一个原因是希望通过本书的学习能让读者更加熟悉怎样用 Java 语言来体现面向对象的设计思想。

1.2 从抽象到类

面向对象语言有 3 个重要特性：封装、继承和多态，其中，封装性是最基本的特性之一。首先观察下列简单的能输出矩形面积的 Java 应用程序的源文件：

ComputerRectArea.java

```
public class ComputerRectArea {
    public static void main(String args[]) {
        double height;      //高
        double width;        //宽
        double area;         //面积
        height=23.89;
        width=108.87;
        area=height*width;   //计算面积
        System.out.println(area);
    }
}
```

上述 Java 应用程序输出宽为 108.87、高为 23.89 的矩形面积。

通过上述 Java 应用程序注意到这样一个事实：

如果其他 Java 应用程序也想计算矩形面积，同样需要知道使用矩形的宽和高来计算矩形面积的算法，即也需要编写和这里同样多的代码。现在提出如下问题：

能否将和矩形有关的数据以及计算矩形面积的算法进行封装，使得需要计算矩形面积

的 Java 应用程序无须编写计算面积的代码就可以计算出矩形面积呢?

面向对象的一个重要思想就是通过抽象得到类,即将某些数据以及针对这些数据上的操作封装在一个类中。也就是说,抽象的关键点有两点:一是数据;二是数据上的操作。

这里对所观察的矩形做如下抽象:

- 矩形具有宽和高的属性。
- 可以使用矩形的宽和高计算矩形面积。

现在根据如上的抽象,定义出如下的 Rect 类。

Rect.java

```
public class Rect
{
    double width;    //矩形的宽
    double height;   //矩形的高
    double getArea() //计算面积的方法
    {
        double area=width*height;
        return area;
    }
}
```

1. 类声明

在上述代码中(第一行), class Rect 称做类声明, Rect 是类名。

2. 类体

类声明之后的一对大括号{、}以及它们之间的内容称做类体,大括号之间的内容称做类体的内容。

上述 Rect 类的类体的内容由两部分构成:一部分是变量的声明,称做域变量或成员变量,用来刻画矩形的属性,如 Rect 类中的 width 和 height;另一部分是方法的定义(在 C 语言中称做函数),用来刻画行为功能,如 Rect 类中的 double getArea()方法。

Rect 类好比是生活中电器设备需要的一个电阻,如果没有电器设备使用它,电阻将无法体现其作用。

以下的 Java 应用程序的主类(含有 main 方法的类,Java 应用程序从主类开始运行)中使用 Rect 类创建对象,该对象可以完成计算矩形面积的任务,而使用该对象的 Java 应用程序的主类无须知道计算面积的算法就可以计算出矩形面积。

Application.java

```
public class Application {
    public static void main(String args[]) {
        Rect rectangle1,rectangle2;    //声明两个对象
        rectangle1 = new Rect();        //创建对象
        rectangle2 = new Rect();
        rectangle1.width=128;
        rectangle1.height=69;
        rectangle2.width=18.9;
```

```

        rectangle2.height=59.8;
        double area=rectangle1.getArea();
        System.out.println("rectangle1的面积:"+area);
        area=rectangle2.getArea();
        System.out.println("rectangle2的面积:"+area);
    }
}

```

1.3 类与程序的基本结构

一个应用程序（也称为一个工程）是由若干个类所构成，这些类可以在一个源文件中，也可以分布在若干个源文件中，如图 1.1 所示。

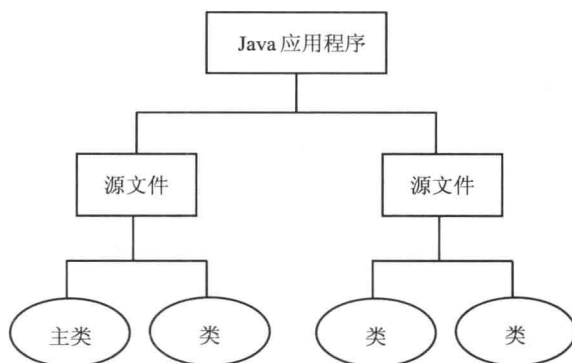


图 1.1 程序的结构

Java 程序以类为“基本单位”，即一个 Java 程序就是由若干个类所构成。一个 Java 程序可以将它使用的各个类分别存放在不同的源文件中，也可以将它使用的类存放在一个源文件中。一个源文件中的类可以被多个 Java 程序使用，从编译角度看，每个源文件都是一个独立的编译单位，当程序需要修改某个类时，只需要重新编译该类所在的源文件即可，不必重新编译其他类所在的源文件，这非常有利于系统的维护。从软件设计角度看，Java 语言中的类是可复用代码，编写具有一定功能的可复用代码是软件设计中非常重要的工作。

在下面的应用程序共有 3 个 Java 源文件，其中，Application.java 是主类。

Rect.java

```

public class Rect {
    double width;           //矩形的宽
    double height;          //矩形的高
    double getArea() {
        double area = width*height;
        return area;
    }
}

```