内 容 简 介

这是一本非常实用的 VB 设计模式专著。书中从使用 VB 进行面向对象编程的角度，讲述如何在设计模式思想指导下用 VB.NET 和 VB6 编写应用程序。

本书首先概述了 VB 面向对象编程的概念和方法，然后讨论了 23 种设计模式，每种模式都通过至少一个完整的 VB 程序来说明，以帮助读者建立设计模式的思想。这种方式使得设计模式的概念容易掌握，也更容易理解设计模式的本质及目的。读过本书，VB 程序员可以迅速提高编程水平，并从设计模式中受益。

本书可以为设计比较复杂的 VB 程序提供指导，既可以作为深入学习 VB 编程的教科书，也可作为探讨软件设计模式研究领域人员的参考书。

EISBN：0-201-70265-7

书　名：VISUAL BASIC 设计模式——VB6.0 和 VB.NET
作　者：James W.Cooper
译　者：赵会群等
出版者：清华大学出版社（北京清华大学学研大厦，邮编 100084）
http://www.tup.com.cn
印刷者：北京市耀华印刷有限公司
发行者：新华书店总店北京发行所
开　本：异 16　　印张：25.625　　字数：559 千字
版　次：2003 年 4 月第 1 版　　2003 年 4 月第 1 次印刷
印　数：0001～4000
盘　号：ISBN 7-89494-071-2
定　价：47.00 元（1CD）

译 者 序

设计模式一词是 20 世纪 70 年代一个叫 *Christopher Alexander* 的建筑工程师首先提出来的。在他编写的两本书中首先提到设计模式的概念，一本书叫《模式语言》，另一本书叫《永恒的建筑风格》。在这两本书中用许多实例阐述了文档模式的机理。

计算机领域中有关模式的概念在 1987 年 OOPSLA 会议上首次正式提出。此后设计模式的概念在一些知名计算机专家（如 Grady Booch，Richard Helm，Erich Gamma 和 Kent Beck）的论文和报告中频频出现。可以说在 1995 年前，设计模式概念还只停留在理论研究阶段，到 1995 年 Erich Gamma，Richard Helm，Ralph Johnson 和 John Vlissides 出版了 *design patterns：software reuse basics* 一书之后，设计模式概念才真正被广泛地应用。

设计模式是一种建议，它提供了程序共享设计的思路。在编写程序过程中，我们可能遇到许多以前已经遇到过的问题，而且今后还会继续发生，我们经常思考如何解决该问题。文本模式的思想就是你可以重用和共享你已经知道解决问题的最佳途径的信息。软件设计模式就是借鉴了这种思想。

软件设计模式作为软件重用的一种方法，对提高程序设计与开发的效率有直接的作用，学习和研究软件设计模式十分重要。软件设计模式作为一个新的研究领域，对它的研究与学习还刚刚起步，有关的理论和方法尚未完全形成。本书作为目前仅有的几本有关软件设计模式的专著，对读者学习和进一步研究软件设计模式的理论和方法有一定的作用。本书的最大特点是结合具体设计示例介绍软件设计模式的基本思想和方法，这对读者理解软件设计模式相关概念和方法十分有利。另外，书中采用比较常用的 VB6 和功能强大的 VB.NET 作为软件设计模式的实现工具，使得读者能够更容易接受软件设计模式的实现方法。

由于有关软件设计模式的专著在国内还刚刚引入，我们对这方面的学习和研究还不够深入，所以在翻译过程中可能有许多地方都没有把原著的精华体现出来，如有不当之处还请读者多多指教。本书主要的翻译工作和统稿工作由赵会群完成。此外，吴洁明、胡景德、吕春和刘娟也参加了翻译工作，15 章到 18 章由吴洁明翻译，19 章到 21 章由胡景德翻译，22 章到 26 章由吕春翻译，27 章到 32 章由刘娟翻译。

前　言

这是一本非常实用的 VB 设计模式专著，讲述如何在设计模式思想指导下用 VB.NET 和 VB6 编写程序，也可以作为 VB.NET 的编程导论。本书以一系列篇幅较短小的章节来讨论设计模式，每种模式一章，并给出一到两个已经用程序实现的例子，并在每一章中用 UML 图来解释类之间的相互作用。

该书不是众所周知的由“四人帮”（Gang of Four）编写的 *Design Patterns* 一书的姊妹篇，而是为那些希望深入学习和应用设计模式的人编写的导学书。在阅读本书之前没有必要一定读过 *Design Patterns*，但在阅读之后你可能急切地想去读或更细致地重读原著 *Design Patterns*。

在本书中，你将了解到设计模式是程序中经常用于组织对象的有效方法，可以使程序设计更简单并且更容易修改。你也将会发现，在逐渐地了解设计模式后，将会增加许多如何进行程序设计的新词汇。

人们开始从纯理论到纯实践的各个方面去认同设计模式，到最终发现它的巨大作用的时候，感叹将脱口而出。此刻也就意味着你的脑海里出现了如何使用设计模式来帮助你工作的设想。

在本书中，我们试图通过各种不同的方法，帮助你建立设计模式的思想。本书由引言、VB.NET 简介、设计模式一般性描述、生成模式、结构模式和行为模式六个部分构成。

对每一个设计模式，我们首先简单介绍所讨论的设计模式，然后设计简单的程序实例。每一个实例都是基于图形用户界面的程序，你可以运行这些程序，以便更具体地了解设计

模式的含义。书中的所有示例程序，以及这些示例的改进程序都放在配套光盘中，你可以从光盘中运行、修改这些程序，并且根据实际工作调整程序中的内容。

我们首先向你展示如何在 VB6 中有效地使用设计模式，然后再在 VB.NET（也被称为 VB7）中使用同样的模式。因为每一个程序中都有许多类文件，所以我们为每一个示例建立了一个 VB 工程文件，并分别存放在各自的子目录中，从而避免混乱，我们把 VB.NET 示例分开存放于该模式的子目录中。本书基于 Beta-2 版的 VB.NET 讨论编程，它与最终的发行版不会有太大的变化。你可以从 Addison-Wesley 的网址获得书中改进的示例程序。

翻开此书，你将会看到我们制作的一些用于解释示例的程序屏幕截图，它提供另一种巩固设计模式学习的方法。另外，你也可以看到另一种用于解释类之间交互的 UML 图。这是一种用框和线来解释类之间的联系和层次结构的示意图。图中的箭头指向父类，而点线箭头指向接口。如果你不是十分了解 UML，可以仔细阅读我们在第 II 部分对 UML 的介绍。

读完此书，你将从中受益颇多，并将在日常的 VB 编程工作中使用设计模式。

James W. Cooper
Nantuchet, MA
Wilton, CT
Maui, Hi

目　　录

第 I 部分　VB 面向对象程序设计

第 1 章　设计模式概述 3

1.1　定义设计模式 4
1.2　学习过程 5
1.3　学习设计模式 6
1.4　评论面向对象方法 6
1.5　VB 设计模式 7
1.6　本书组织 7

第 2 章　UML 图 8

2.1　继承 9
2.2　接口 10
2.3　组合 11
2.4　注释 12
2.5　基于 WithClass 的 UML 图 12
2.6　Visual Basic 工程文件 13

第 3 章　在 VB 中使用类和对象 14

3.1　一个简单的温度换算程序 14
3.2　构建一个温度类 15
3.2.1　换算到开氏温标 17
3.3　在 Temperature 类作决定 17
3.4　在类中进行数据的格式化和值转换 18
3.4.1　处理不合理值 20
3.5　一个字符串的分割类 21
3.6　类对象 23
3.6.1　类包含 24
3.7　类初始化 25
3.8　类和属性 26
3.9　另一个接口示例——伏特计 27
3.10　一个 vbFile 类 28
3.11　Visual Basic 程序设计风格 30
3.12　小结 30

第 4 章　面向对象的程序设计 31

4.1　构建 VB 对象 32
4.2　产生一个对象实例 32
4.3　一个用 VB 实现的测量程序 32
4.4　对象中的方法 33
4.5　变量 33
4.6　参数传值和传址 34
4.7　面向对象程序设计中的术语 34

第 5 章　创建自己的控件 36

5.1　一个激活的文本 36
5.1.1　调整用户控件大小 37
5.2　测试 HiText 控件 38
5.3　在用户控件中增加属性和方法 39
5.4　编译一个用户控件 40
5.5　小结 40
5.6　光盘中的程序 40

第 6 章　继承和接口 41

6.1　接口 41
6.2　一个投资模拟器 42

6.3 编写一个模拟器 43
6.4 用于接口使用的指示器....44
6.5 重新使用共同的方法....47
6.6 隐藏接口 49
6.7 小结 49
6.8 光盘中的程序 49

第 7 章 VB.NET 简介 50

7.1 VB.NET 中新的语法 50
7.1.1 改进的函数语法....51
7.2 变量声明和作用域 52
7.2.1 VB.NET 中的对象 53
7.3 编译选择 53
7.3.1 VB.NET 中的数值型变量....54
7.4 VB6 和 VB.NET 中的属性....54
7.5 快捷等号语法 55
7.6 管理语言和垃圾回收....56
7.7 VB.NET 中的类 56
7.8 构建一个 VB7 应用 58
7.9 VB.NET 最简单的窗口程序 60
7.10 继承 61
7.11 构造函数 63
7.12 VB.NET 中的图画 65
7.13 工具标签和鼠标移动键....65
7.14 重载 66
7.15 继承 66
7.16 名字空间 67
7.16.1 从长方形中产生一个正方形 68
7.17 Public，Private 和 Protected 69
7.18 在导出类中重载方法....69
7.19 重载与隐蔽....70
7.20 重载窗口控件....72
7.21 接口 73
7.22 小结 74
7.23 光盘中的程序....74

第 8 章 VB.NET 中的数组、文件和异常..... 75

8.1 数组 75
8.2 集合对象 76
8.2.1 数组列表....76
8.2.2 Hashtable....77
8.2.3 SortedList 78
8.3 异常 78
8.4 多重异常 79
8.5 抛出异常 80
8.6 文件处理 80
8.6.1 File 对象....80
8.6.2 读一个文本文件....81
8.6.3 写一个文本文件....81
8.7 在文件处理中使用异常....82
8.8 测试文件结束....82
8.9 FileInfo 类....83
8.10 vbFile 类 84
8.11 光盘中的程序....85

第 II 部分 生成模式

第 9 章 简单工厂模式..... 87

9.1 一个简单工厂如何工作....87
9.2 代码片段 87
9.3 两个导出类....88

9.4 构建简单工厂 89
9.4.1 使用工厂.....89
9.5 用 VB.NET 编写工厂模式 91
9.6 使用数学计算的工厂模式.....92
9.7 光盘中的程序 93

第 10 章 工厂方法模式..... 94

10.1 Swimmer 类.....96
10.2 Events 类 97
10.3 直接筛选 99
10.3.1 交叉筛选.....99
10.4 我们的选拔程序 100
10.5 其他的类工厂 101
10.6 用 VB7 实现的选拔程序 101
10.7 什么时候使用工厂方法.....103
10.8 光盘上的程序 104

第 11 章 抽象工厂模式..... 105

11.1 一个花卉工厂.....105
11.2 用户接口如何工作.....108
11.3 用 VB7 实现一个抽象工厂 108
11.3.1 PictureBox 框.....110
11.3.2 处理单选按钮和按钮事件 111
11.4 增加更多的类.....112
11.5 抽象工厂评价.....113
11.6 光盘中的程序.....113

第 12 章 单一类模式 114

12.1 使用静态方法产生单一类.....114
12.2 捕获错误 116
12.3 提供单一类的全局访问点.....116
12.4 MSComm 控件作为单一类 117
12.4.1 可用的串口.....118
12.5 用 VB.NET 实现单一类 119
12.5.1 使用私有的构造函数.....120
12.5.2 VB7 实现的单一类中的错误处理.....120
12.6 一个 VB.NET 实现的 SpoolDemo 程序.....122
12.7 全局访问点.....123
12.8 单一类模式其他结论.....123
12.9 光盘上的程序.....123

第 13 章 构造器模式.....124

13.1 一个投资跟踪程序.....124
13.2 调用构造器.....126
13.3 列表框构造器.....128
13.4 复选框构造器.....128
13.5 用 VB.NET 实现构造器 129
13.5.1 股票类工厂 131
13.5.2 复选框类.....132
13.5.3 列表框类.....133
13.6 在列表框中使用下标集合 134
13.6.1 最终选择.....135
13.7 小结 137
13.8 光盘中的程序.....137

第 14 章 原型模式 138

14.1 VB6 中的克隆 138
14.2 使用原型.....139
14.3 使用原型模式.....142
14.3.1 在子类中增加方法 143
14.3.2 具有相同接口的不同类 144
14.4 原型管理器.....147
14.5 用 VB7 编写原型 147
14.6 小结 150
14.7 光盘中的程序.....151
14.8 生成模式总结.....151

第III部分 结构模式

第 15 章 适配器模式 153

15.1 在列表中移动数据.....153
15.2 使用 MSFlexGrid 154
15.3 使用 TreeView.....157
15.3.1 对象适配器.....157
15.4 在 VB7 中使用适配器.....158
15.5 VB.NET 的 TreeView 适配器.....160
15.6 采用 DataGrid 控件.....161
15.7 类适配器 163
15.8 两路适配器 164
15.9 在 VB.NET 中实现对象和类适配器 164
15.10 可插入的适配器 164
15.11 在 VB 中的适配器 164
15.12 光盘中的程序 165

第 16 章 桥模式..... 166

16.1 visList 类 169
16.2 类关系图 169
16.3 桥模式的扩展 169
16.4 ActiveX 控件作为桥.....172
16.5 用 VB.NET 实现桥模式 172
16.5.1 ListBox 的 visList 类 173
16.5.2 Grid 的 visList 类.....174
16.5.3 导入数据.....174
16.6 改变数据格式 175
16.7 小结 177
16.8 光盘中的程序 177

第 17 章 组合模式..... 178

17.1 一个组合的实现 179
17.2 计算工资 179
17.3 员工类 179
17.4 下标类 182
17.5 Boss 类.....183
17.6 构建员工树.....185
17.7 自提升 187
17.8 双向链表.....187
17.9 小结 188
17.10 一个简单组合.....189
17.11 VB 中的组合 189
17.12 VB.NET 中的组合 189
17.12.1 枚举器.....191
17.12.2 Boss 构造函数多态 191
17.13 其他实现条款.....192
17.14 光盘中的程序.....193

第 18 章 修饰模式 194

18.1 CoolButton 按钮的修饰 194
18.2 使用 Decorator.....197
18.3 将 ActiveX 控件作为 Decorator 使用.....200
18.4 VB.NET 中的 Decorator.....200
18.5 不可见的 Decorator.....202
18.6 修饰、适配和组合模式.....202
18.7 小结 203
18.8 光盘中的程序.....203

第 19 章 伪模式 204

19.1 数据库是什么.....204
19.2 从数据库中获得数据.....205
19.3 数据库的种类.....206
19.4 ODBC 206
19.5 微软的数据库连接策略.....207

19.6 数据库的结构 207
19.6.1 DBase 类....207
19.7 建立 Façade 类 210
19.7.1 Stores 类....212
19.8 建立 Stores 和 Foods 表....214
19.9 建立 Price 表 215
19.9.1 建立价格查询....216
19.10 小结 217
19.11 在 VB6 中使用 ADO 访问数据库 218
19.11.1 使用 ADO 连接数据库 218
19.11.2 在表中添加或查询记录 219
19.11.3 使用 ADO 扩展 220
19.12 ADO 中的 DBase 类 221
19.13 在 VB.NET 中访问数据库 224
19.14 使用 ADO.NET 224
19.14.1 连接数据库....225
19.14.2 从数据库表中读取数据....225
19.14.3 执行查询....226
19.14.4 删除表中的内容....226
19.15 使用 ADO.NET 向数据库表中添加记录 227
19.16 编写 VB.NET ADO 伪模式....228
19.16.1 DBTable 类 229
19.17 为每一个表格创建类....230
19.18 存储价格 232
19.19 装载数据库表....234
19.20 最终的应用程序....235
19.21 Façade 的组成 236
19.22 小结 236
19.23 光盘中的程序....237
第 20 章 轻量模式 238
20.1 讨论 239
20.2 举例 239
20.2.1 类结构图....242
20.2.2 选择一个文件夹....242
20.3 用 VB.NET 实现轻量模式文件夹 243
20.4 VB 中 Flyweight 的使用....247
20.5 可共享对象....247
20.6 Copy-on-write 对象 248
20.7 光盘中的程序....248
第 21 章 代理模式 249
21.1 示例 249
21.2 用 VB.NET 实现代理模式....251
21.3 VB 中的 Proxy....253
21.4 Copy-on-write 253
21.5 相关模式的比较....253
21.6 光盘中的程序....254
21.7 结构模式总结....254

第IV部分　行为模式

第 22 章 响应链.... 256
22.1 适用范围 256
22.2 代码示例 257
22.3 列表框 260
22.4 实现帮助系统 262
22.4.1 获得帮助命令....263
22.5 链还是树....264
22.6 用 VB.NET 实现响应链 266
22.7 请求的种类....268
22.8 VB 中的示例....268

22.9 小结 269
22.10 光盘中的程序 269
第 23 章 命令模式 270
23.1 目的 270
23.2 命令对象 271
23.3 建立 Command 对象 271
23.4 命令数组 272
23.5 命令模式小结 275
23.6 提供撤消 275
23.7 VB.NET 中的命令模式 280
23.8 CommandHolder 接口 282
23.9 在 VB.NET 中处理撤消命令 284
23.10 VB 中的命令模式 288
23.11 光盘中的程序 288
第 24 章 解释模式 289
24.1 目的 289
24.2 适用范围 289
24.3 一个简单的报告示例 290
24.4 解释这种语言 290
24.5 解析时使用的对象 291
24.6 减少分析栈 295
24.7 实现解释模式 296
24.7.1 语法树 296
24.8 用 VB6 实现解释器 300
24.9 解析对象 302
24.10 小结 303
24.11 光盘上的程序 303
第 25 章 迭代模式 304
25.1 目的 304
25.2 VB6 示例代码 305
25.2.1 使用迭代 306
25.3 过滤迭代 307
25.3.1 过滤迭代器 307
25.4 用 VB.NET 实现迭代 309
25.5 小结 311
25.6 光盘中的程序 311
第 26 章 协调模式 312
26.1 一个示例 312
26.2 控件间的交互 313
26.3 代码示例 314
26.3.1 系统的初始化 317
26.4 协调和命令对象 317
26.5 用 VB.NET 实现协调模式 318
26.5.1 初始化 320
26.5.2 处理新控件的事件 320
26.6 小结 321
26.7 单接口协调类 322
26.8 用法的讨论 322
26.9 光盘中的程序 322
第 27 章 记事模式 323
27.1 目的 323
27.2 实现 324
27.3 示例 324
27.3.1 注意事项 330
27.4 用户接口中的命令对象 330
27.5 处理鼠标和画图事件 331
27.6 用 VB.NET 编写 Memento 332
27.7 小结 334
27.8 光盘中的程序 335
第 28 章 观察模式 336
28.1 查看颜色的变化 337
28.2 用 VB.NET 实现观察模式 339
28.3 传递的消息 341
28.4 小结 342
28.5 光盘中的程序 342

第 29 章　状态模式 343

29.1　示例代码 343
29.2　状态间的转换 348
29.3　Mediator 和 StateManager 之间的交互.....349
29.4　处理 Fill 状态.....352
29.5　处理撤消列表 352
29.5.1　用 VB6 实现填充圆形 355
29.6　在 VB.NET 中实现状态模式 355
29.7　Mediator 和 God 类.....360
29.8　小结 361
29.9　状态转换 361
29.10　光盘中的程序 362

第 30 章　策略模式 363

30.1　目的 363
30.2　示例代码 364
30.3　Context 类.....365
30.4　程序命令 365
30.5　线形图和柱形图策略.....366
30.6　用 VB 绘图.....367
30.7　用 VB.NET 实现策略模式 369
30.8　小结 372
30.9　光盘中的程序 373

第 31 章　模板方法模式 374

31.1　目的 374
31.2　Template 类中的方法.....375
31.3　示例 376
31.3.1　画一个标准的三角形 377
31.3.2　画一个等腰三角形 378
31.4　画三角形程序.....379
31.5　模板和回调.....380
31.6　小结 381
31.7　光盘中的程序.....381

第 32 章　访问者模式.....382

32.1　目的 382
32.2　使用访问者模式的时机.....383
32.3　示例 384
32.4　访问类 385
32.5　访问多个类.....386
32.6　经理也是员工.....388
32.7　访问者的全部捕获操作.....388
32.8　两次分派.....389
32.9　为什么要这样做.....389
32.10　遍历一系列的类.....390
32.11　在 VB6 中编写访问者 390
32.12　小结 393
32.13　光盘中的程序.....393

参考文献.....394

第 I 部分

VB 面向对象程序设计

本书的第 I 部分首先介绍面向对象程序设计的基本概念，以及用 VB 进行面向对象程序设计的方法。内容包括如何用 UML 图表示程序及其类对象，如何用 VB6 构建自己的控件；还将介绍对象的继承和接口概念，以及 VB.NET 的基本编程方法。在此基础上，还将进一步学习如何用 VB6 和 VB.NET 这些强大的语言工具进行简化程序设计的技术。

在学完上述导引性内容之后，你就可以深入地学习和使用本书介绍的各种设计模式。

第 1 章　设计模式概述

当坐在计算机前开始设计一个新程序的时候，你直观地知道要实现的功能，用什么对象和什么数据结构来实现这些功能，但如何选择更好的、更通用的编程方法仍然不是十分明确。

事实上，在想清楚程序中包括哪些代码模块以及这些代码模块的接口之前，你可能不会编写一条语句。越是从全局考虑，编写的程序越有效。如果在编写程序代码之前，你还没有形成全局观点，那么最好还是暂时离开计算机，放眼窗外，即使是程序设计的方法比较简单，你也要如此。

在某种意义上，越优雅的程序的可复用性与可维护性也会越好。但在你放心地设计完一个相对优雅的解决方案时，却有可能存在着很多未经显露的内在问题。

设计模式开始引起计算机学者重视的原因之一是，它是实现简单、精巧和可重用解决方案的基础。术语“设计模式”对从未听过的人来说有点太正规，甚至容易引起困惑。但事实上，它是一种方便的重用方法，无论是工程之间，还是程序员之间都是如此。设计模式的思想并不复杂，简单地说，设计模式就是典型的不同对象之间交互的定义和分类。

早期的关于程序框架的文献中经常引用的模式是 Smalltalk（Krasner and Pope 1988）中提出的“模型-视图-控制器”框架（Model-View-Controller）。在该模式下，把用户接口设计问题分成 3 个部分，如图 1-1 所示。这 3 个部分分别是数据模型（Data Model），用于计算；视图（View），表现用户接口；控制器（Controller），实现用户与视图的交互。

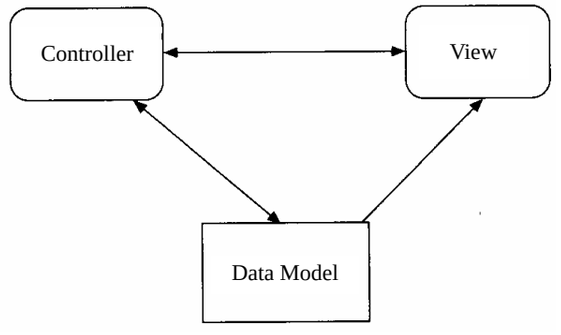


图 1-1　模型-视图-控制器框架

这 3 个部分构成 3 个不同的对象，它们有其各自的数据处理规则。用户、用户界面和

数据之间的通信应被细心地组织，而且它们的功能被有效地实现，它们采用一定的连接约束集进行交互，这就是一个典型的设计模式实例。

另一方面，从面向对象程序设计的角度来看，设计模式描述了如何使得不同对象的方法和数据方法在对象通信中保持相互独立，这正是面向对象程序设计的目标。如果你一直是这样做的，那么你可能正在使用设计模式的基本思想。

设计模式是 20 世纪 90 年代初期由 Erich Gamma（1992 年）正式提出的，他结合 GUI 应用框架 ET++，对设计模式给出描述。之后设计模式引起广泛关注，组织了许多有关的讨论和会议，但最能反映学术水平的是由 Gamma、Helm、Johnson 和 Vlissides 于 1995 年撰写的 Design Patterns — Elements of Reusable Software（《设计模式——可复用面向对象软件的基础》）。该书对设计模式的研究产生了重要的影响，深受读者的好评，读者也风趣的把这本书称为“四人帮”（Gang of Four，GoF）。这本书中介绍了常见的 23 种设计模式，讲述了何时使用和如何使用这些模式。在本书中，我们提到这本里程碑式的巨著时将简称它为 Design Patterns。

目前已出版了一些有关设计模式的专著，比较相关的一本书是 Alpert、Brown 和 Woolf 于 1998 年撰写的 The Design Patterns Smalltalk Companion，该书包括 23 个与 Design Patterns 相同的基于 Smalltalk 的设计模式，所以我们把该书称为 Smalltalk 实现。最近出版的一本关于设计模式的书是《Java Design Patterns: A Tutorial》，书中完全用 Java 来解释设计模式思想。

1.1 定义设计模式

在日常生活中，我们经常可以发现许多事情的处理是按一定的模式进行的。同样，在编写程序时也是如此，并完全接受对象之间以相对独立的方式进行通信的这种有效的编程模式。

下面是在一些文献中出现的设计模式定义。

- “设计模式是对常见设计问题的可重复使用的解决方案”（The Smalltalk Companion）。
- “设计模式构成一个规则集合，这些规则描述如何实现软件开发工作中的特定任务”（Pree 1994）。
- “设计模式更加注重体系结构设计的重用，而框架关心软件设计和实现的细节”（Coplien and Schmidt 1995）。
- “模式阐述了在特定设计环境下出现的设计问题，并给出它的解决方案”（Buschmann et al.1996）。
- “模式是对类、实例或组件的更高层次抽象的标识和说明”（Gamma et al.1993）。

上述定义对理解和分析设计模式的结构、构成和内部逻辑是十分有帮助的，设计模式不仅是有关对象的设计，而且也是关于对象之间交互的设计。从这个角度看，设计模式又可理解成为通信模式。

有的模式不仅仅关心对象之间的通信，而且还关心对象之间的继承和包含关系。实际上，这些设计模式正是因其简单而精巧的交互方法的设计而变得非常重要。

设计模式可以存在于许多不同的设计层次，这些设计层次包括从底层的解决方案到更具有一般意义的高层系统问题。可见到的设计模式有近百种，在一些文章和会议论文集中可以看到不同类型的设计模式，一些是已经被广泛使用的设计模式，而有一些是特殊应用中的设计模式（Kurata 1998）。

显然，设计模式不是不加思考地拷贝一种已有的模式，而是需要认真地思考、发现，这一过程称为“模式挖掘”，这个问题值得用一本书的篇幅来讨论。

本书从原版的《Design Patterns》一书中，选择了 23 个具有一定应用背景并且规模适中的设计模式，作为本书的实例。这些实例会使你更好地理解我们所讨论的对象。

这些实例分为 3 种类型。生成模式、构造模式和行为模式，这 3 种模式的定义如下：

- 生成模式：可以在此类模式基础上生成更灵活的设计模式，而不是让你直接实例化一个对象，你可以在此类模式中增加符合给定条件的新对象。
- 构造模式：可以帮助你把一些对象组合到一个更大的系统结构中，如复杂的用户接口或会计数据。
- 行为模式：可以帮助你定义系统中不同对象之间的通信机制，以及如何实现复杂系统中信息流的控制。

Visual Basic 各种版本的应用是本书另一个需要注意的问题，书中为 23 个模式中的每一个提供了至少一个完整的 VB6 和 VB.NET 程序。在本书的配套光盘中有上述原程序和对应的设计模式，读者可以对原程序做少许编辑和修改之后运行。你能够在讨论模式的每一章的后面看到一个光盘内容的列表。

1.2　学习过程

如果不考虑实现语言，设计模式的学习可以分为以下几个步骤：

1. 接受阶段
2. 共识阶段
3. 深化阶段

也就是说，首先你应接受设计模式对你的工作很重要这一观点；然后你要充分认识到需要学习的设计模式的内容以便知道什么时候可以用它们；最后，结合你工作中遇到的实

际问题，深入学习有关的细节。

对有些人来说是幸运的，因为设计模式是显而易见的工具，他们可以通过阅读一些总结性的材料就能掌握设计模式的基本功能。而对其他人来说，认识过程是一个较缓慢的引导过程，只有当他们看到设计模式在工作中得到卓有成效的应用时，才认识到设计模式的有用。这本书帮助你逐步进入最后阶段，你可以通过书中提供的程序，逐步深化对设计模式的认识。

《Design Patterns》一书中的示例很简洁，有些是用 C++实现，有些是用 Smalltalk 实现。你还可以根据实际工作情况，用其他语言来实现。本书采用 Visual Basic 来实现。

1.3 学习设计模式

可以有多种途径研究设计模式。其一是读这本书之后，再读该书的原著《Design Patterns》，当然也可以改变先后次序。其次，强烈地建议你深入学习《the Design Patterns Smalltalk Companion》一书，该书提供了模式的不同描述。除此之外，在许多网站上有学习和讨论设计模式的相关资料。

1.4 评论面向对象方法

使用设计模式的根本原因是严格区分不同的类，防止不同类之间的互相干预。不仅如此，使用设计模式可以不必一切重新开始，它提供了简单的、其他程序员容易理解的方式描述程序的方法。

有许多实现对象分离的方法，面向对象程序设计中的继承和封装就是程序员经常采用的方法。几乎所有面向对象的程序语言都支持继承。一个从父类继承的子类可以访问父类的所有方法，也可以访问非私有的变量。然而，继承有时是不适合的，你完全没有必要像背着一个大行李一样地继承不需要的类方法和变量。Design Patterns 提倡：使用接口而不是实现。

简单地说，你应该在任何类的顶层定义抽象类或接口，在抽象类或接口中只定义一些简单的、类必须支持的方法，不必具体实现。这样在所有的派生类中，你就有更广泛的自由空间去实现更适合的方法。

另一个你需要接受的设计理念是对象组合。这是一种简单的把几个对象组合在一起的方法，即把几个对象封装到一个大的对象中。对一个面向对象初学者来说，可能过多地使用继承来解决问题，但逐渐地你会体会到使用对象组合的优点。可以通过接口来实现你的任务，而不是继承父类的方法。为此，Design Patterns 提倡的第 2 个设计原则是：对象合成优于继承。

让我们来看看 VB，VB 提供了简便的实现类包含的方法，所以在本书中你会体会到如何用 VB 实现面向对象思想。

1.5　VB 设计模式

本书讨论 23 个设计模式，每一个设计模式至少有一个程序实例，所有实例都有一个可视化的用户界面。所有的设计模式都使用类、接口和对象组合，但是它们十分简明，所以不会因为程序过于复杂而掩盖设计模式的基本特性。

书中分别用 VB6 和 VB.NET 来表述设计模式。多数情况下先用 VB6，然后再说明如何用 VB.NET(VB7)使程序实现更容易一些。

虽然本书用 VB 作为描述语言，但它不是 VB 语言教科书。它包含了 VB 的中心内容，但有许多 VB 特性没有包括在内。你会发现这是一本比较好的用 VB 进行面向对象程序设计，以及用 VB.NET 编程的导学。

1.6　本书组织

书中引入 23 个设计模式，这些模式分为生成模式、构造模式和行为模式 3 种类型。大多数模式是独立的，但在讨论过程中也利用前面已讨论过的模式实例进行解释。例如，在介绍完工厂（Factory）和命令（Command）模式后，又对其进行了扩展；在介绍完协调（Mediator）实例后，又多次地引用它；在状态（State）模式中多次使用记事（Memento）模式；在解释（Interpreter）模式的讨论中用到响应链（Chain of Responsibility）模式，在讨论次轻量（Flyweight）模式中用到单一（Singleton）模式。原则上，在一个模式没有正式引入之前，是不会在其他模式讨论中使用的。

书中用后来介绍的较复杂的模式来展示 VB.NET 的新特性。例如，列表框、数据网格和树形视图在适配器(Adapter)模式和桥(Bridge)模式中介绍 ,在抽象工厂(Abstract Factory)模式中展示画图对象。我们在单一模式中使用异常，并且讨论在伪 (Facade) 模式中的 ADO 数据库连接，最后介绍如何在代理（Proxy）模式中使用 VB.NET 计时器。

第 2 章　UML 图

在第 1 章中已经用 UML（Unified Modeling Language，统一建模语言）图解释了一个设计模式。UML 是由 Grady Booch，James Rumbaugh 和 Ivar Jacobson 提出的一种通用建模语言，它综合了 3 位建模语言专家的思想和风格，并最终被确认为建模语言的标准。有许多书可以供进一步学习 UML，如 Booch et al（1998），Fowler & Scott（1997）和 Grand（1998）撰写的有关 UML 的书。本章简要介绍 UML 的基本方法。

基本 UML 图用框表示类。让我们看下面一个简单的类（该程序的实际功能不是太强）。

```
'Class Person
Private age As Integer
Private personName As String
'-----
Public Sub init(nm As String)
  personName = nm
End Sub
'-----
Public Function makeJob() As String
  makeJob = "hired"
End Function
'-----
Private Sub splitNames()

End Sub
'-----
Public Function getAge() As Integer
 getAge = age
End Function
'-----
Private Function getJob() As String

End Function
```

我们可以用 UML 图示这个类，如图 2-1 所示。

在图 2-1 中，框的上部包含类和包的名称，第二栏列出了类变量，最下面一栏列出类中的所有方法。名称前面的符号含义分别为，“＋”表示 public，“－”表示 private；“＃”表示 protected。静态方法用下划线注明，抽象方法可以用斜体字表示或加“{abstract}”说明，如图 2-1 中的最后一行所示。

你可以在 UML 图中表示所有的有帮助的类型信息，如图 2-2a 所示。

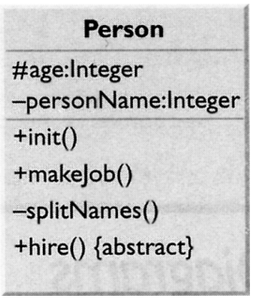
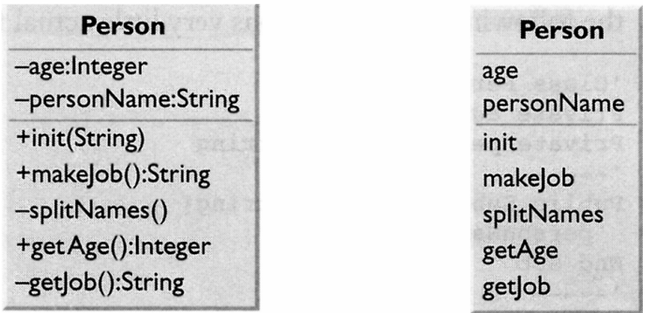


图 2-1　Person 类表示 private，protected 和 public 变量，以及静态和抽象方法



(a)　(b)

图 2-2(a)(b)　Person 类的标有方法类型和没有方法类型的 UML 图

UML 并不要求你把所有的属性都列入其中，可以只把你感兴趣的属性列出。例如，在图 2-2b 中省略了一些方法的细节。

2.1　继承

现在看一下基于 VB7 的由 Person 类派生出来的 Employee 类，其中包含 public，protected 和 private 变量和方法，在 Person 类中把 getJob 方法设置成抽象方法 abstract。这意味着我们要用 MustOverride 关键字。

```
Public MustInherit Class Person
    'Class Person
    Private age As Short
    Protected personName As String
    '-----
    Public Sub init(ByRef nm As String)
        personName = nm
    End Sub
```

```
    '-----
    Public Function makeJob() As String
        makeJob = "hired"
    End Function
    '-----
    Private Sub splitNames()

    End Sub
    '-----
    Public Function getAge() As Short
        getAge = age
    End Function
    '-----
    Public MustOverride Function getJob() As String
End Class
```

我们现在从 Person **类导出** Employee **类，并在** getJob **方法中添加一些代码。**

```
Public Class Employee
    Inherits Person
    Public Overrides Function getJob() As System.String
        Return "Worker"
    End Function
End Class
```

我们用实线和三角箭头表示继承关系。图 2-3 **用** UML **图示** Person **类和** Employee **类的关系，实线和三角箭头表示** Employee **类是** Person **类的子类。**

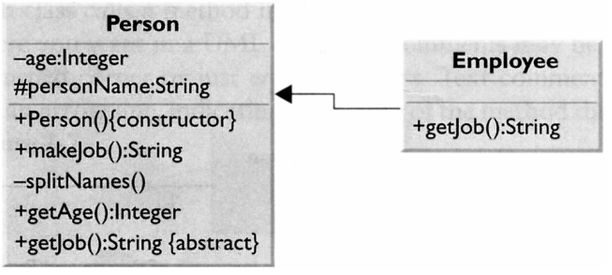


图 2-3 **表示** Employee **类是** Person **类的子类的** UML **图**

注意，在 Employee **类中** Employee **名没有用斜体字，这是因为它现在是一个具体的类，而且在** Employee **类中** getJob **方法已经不再是抽象方法。另外，通常用箭头指向超类，以表明继承关系，但这并不是** UML **所必须的，有时不加任何标识会显得更加简洁。**

2.2 接口

UML **中，接口与继承的表示方法非常相似，只是接口采用虚线箭头，如图** 2-4 **所示。接口名也可采用加双三角括号表示，如**<<interface>>**。**

图 2-4　ExitCommand 实现 Command 接口

2.3　组合

在表示类的层次关系时，经常需要表示类与类之间的包含关系。例如，一个小 Company 可能只有一个 Employee 类和一个 Person 类。

```
Public Class Company
  Private emp as Employee
  Private pers as Person
End Class
```

用 UML 表示这个结构，如图 2-5 所示。

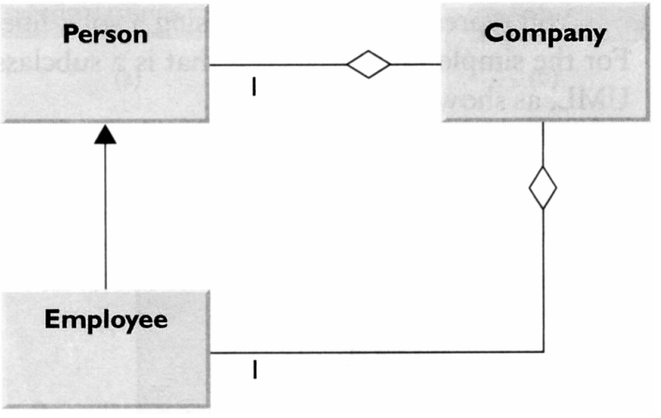


图 2-5　Company 包含 Person 和 Employee 两个类的实例

在图 2-5 中 Person 类与 Company 的连线上表示成 0 到 1，同样 Employee 类与 Company 的连线上也表示成 0 到 1，这种标识表示 Person 类和 Employee 类是 Company 类的构成元素。

如果一个类的许多实例都在其他类中，如一个 Employees 数组的程序如下：

```
Public Class Company
  Private emp() as Employee
  Private pers
End Class
```

本书用单线加“*”或加“0, *”表示对象组成关系，如图 2-6 所示。

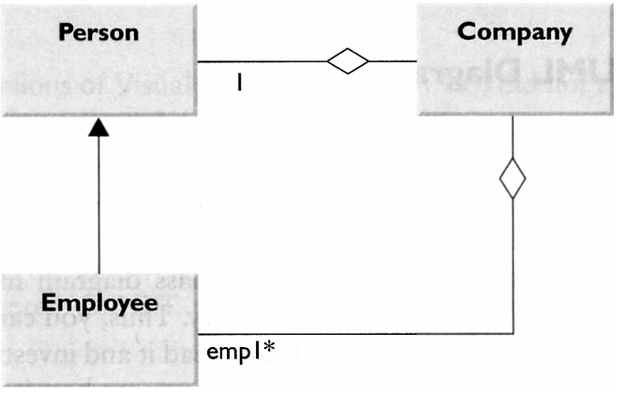


图 2-6 公司中包含许多 Employee 类的实例

有些作者用空心或实心三角箭头表示多元素的组成关系，用圆箭头表示单一对象组合关系，但也不是必须的。

2.4 注释

你可能觉得在 UML 中加一些注释，表示方法的调用关系是非常方便的。UML 允许你在任何地方加注释。通常采用带有翻转角的解释框或在指示箭头上边加说明的方法，如图 2-7 所示。

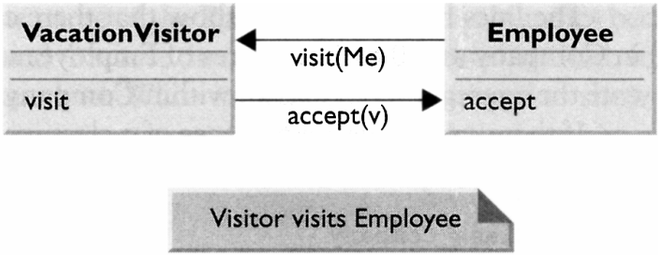


图 2-7 一个加注释的 UML 图

UML 是一种表示程序中对象关系的有力工具，具有丰富的图表说明。但这里我们只简单地介绍一些本书中用到的基本图表。

2.5 基于 WithClass 的 UML 图

本书中所有的 UML 图都是用 MicroGold 版的 WithClass 绘制的，它读取类结构描述，

并生产 UML 类图表。书中只列出部分用此方法生成的重要的方法和类关系图表，完整的设计模式 WithClass 图表文件存放在模式的目录中，这样可以运行光盘中 WithClass 的演示版的拷贝，然后你可以阅读并深入研究，就像在本书看到的各种 UML 图的细节。

2.6　Visual Basic 工程文件

书中所有的程序都是用 Visual Basic 6.0 和 VB.NET 编写的，并使用了工程文件的特征。在光盘中，每一个子目录都包含一个工程文件，你可以编译后运行。

第 3 章　在 VB 中使用类和对象

在早期的 Visual Basic（从 1.0 到 3.0 版）中，不支持面向对象的程序设计风格，一些程序员也已经习惯了这种非面向对象的编程方法。但从 4.0 版开始 Visual Basic 就增加了面向对象的特性，而且可以生成类模块和窗体模块，并把它们作为对象来使用。本章将阐述使用类模块进行程序设计的优点。在以后的章节中，我们将把这一思想用面向对象方法在 VB.NET 下加以实现。

3.1　一个简单的温度换算程序

假定现在编写一个实现从摄氏到华氏换算温度的程序。你知道水在摄氏 0 ° 以下结冰，在 100 ° 以上沸腾；而水在华氏 32 ° 以下结冰，在 212 ° 以上才沸腾。从这些数据你可以导出二者的换算公式。

在两种温度下，沸点到冰点的差分别是 100 ° 和 180 ° ，它们的比是 100/180 或 5/9。因为华氏稳定的冰点是 32，我们把它作为二者的偏移量，这样就有下面的公式：

```
C=(F-32)*5/9
```

或

```
F=9/5*C+32
```

在可视化程序中，允许用户输入温度值，并选择换算的温标，如图 3-1 所示。

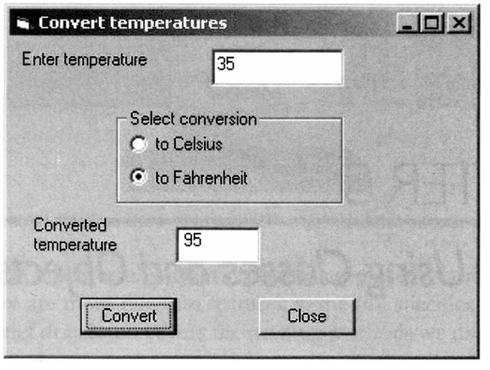


图 3-1　将摄氏 35°换算为华氏 95°的可视化界面

使用 VB 强有力的可视化界面编程工具，你可以几秒钟就完成这个界面的设计工作，并通过两个按钮调用例程来实现计算工作。

```
Private Sub btConvert_Click()
Dim enterTemp As Single, newTemp As Single

 enterTemp = Val(txTemperature.Text)

 If opFahr.Value Then
    newTemp = 9 * (enterTemp / 5) + 32
 Else
    newTemp = 5 * (enterTemp - 32) / 9
 End If

 lbNewtemp.Caption = Str$(newTemp)
End Sub
'------
Private Sub Closit_Click()
 End
End Sub
```

上面是典型的 VB 程序，也是比较简单、便于理解的 VB 程序，但该程序还有许多需要改进的地方。

一个明显的问题是用户界面和数据处理在一个模块中实现，而不是把计算从界面模块中分离出来。一个比较好的思想是，让界面和数据处理在不同的模块中实现，这样当界面修改后不会影响数据处理，反之亦然。

3.2　构建一个温度类

正如上一章提到的那样，在 VB 中一个类就是一个模块，它包含 public 和 private 函数、子例程，并能够存放数据。从逻辑上讲，除了没有可视化的输入窗口之外，类与 Form 相同。类中的函数和子程序是方法的集合。

类模块也与 Basic 语言中的类型或 C 语言中的结构相似，这种相似主要体现在它们都是一组数据的集合，这些数据可以通过 get 和 set 函数获取和创建，这种函数称为访问函数。

你可以在 VB 集成开发环境（IDE）中建立一个类模块，首先使用 Project | Add 命令在已经建立的工程文件中加入类模块，这个类模块称为 clsTemp；然后，你按功能键 F4，打开 Properties 窗口并输入模块名（clsTemp）。

我们想做的就是把所有的计算和温度换算放入这个 clsTemp 类模块中，从而实现计算与交互界面的分离。一个方法是使用类模块重写调用程序，通过产生一个类 clsTemp 的实例来实现计算，代码如下：

```
Private Sub btConvert_Click()
 Dim enterTemp As Single, newTemp As Single
 Dim clTemp As New clsTemp  'create class instance

 If opFahr.Value Then
    clTemp.setCels txTemperature
    lbNewtemp.Caption = Str$(clTemp.getFahr)
 Else
    clTemp.setFahr txTemperature
    lbNewtemp.Caption = Str$(clTemp.getCels)
 End If
```

注意，在你生成一个实例时需要在 Dim **语句中用** new **定义一个新的类名。**

```
Dim clTemp As New clsTemp    'create class instance
```

如果你不用 new **声明变量，如：**

```
Dim clTemp as clsTemp
```

这时你只建立了指向类实例的一个指针，在没有用 new **创建之前，该变量没有初始化。你可以用** Set **命令给指针赋值。**

```
Set clTemp = New clsTemp    'create instance of clsTemp
```

在这个程序中有 setCels **和** setFahr **两个** set **方法，**getCels **和** getFahr **两个** get **方法。这些方法实现数据的设置和获取。程序如下：**

```
Private temperature As Single

Public Sub setFahr(tx As String)
       temperature = 5 * (Val(tx) - 32) / 9
End Sub

Public Sub setCels(tx As String)
       temperature = Val(tx)
End Sub

Public Function getFahr() As Single
       getFahr = 9 * (temperature / 5) + 32
End Function

Public Function getCels() As Single
       getCels = temperature
End Function
```

在上面的程序中，temperature **被声明为私有变量，所以它不能从类的外部访问。对数据的访问只能通过** 4 **种访问器方法进行。该程序重新编写的主要目的是实现了变量的存储和获取细节与外部隔离，这些细节只在类中可知。在这个类中仍然用摄氏度量值存储温度，然后再根据需要实现转换。在程序中还应该对合法字符串的输入进行正确性检查，但因为** Val **函数的返回值为** 0**，并不对非法字符串报错，所以在此不需要检测输入的合法性。**

类的另一重要特性是，它有数据成员。你可以给数据成员赋值，也可以在使用这些数据时访问它们。上面的程序有些简单，类中只包含一个 temperature 数据成员，但在需要时可以设置更多的数据成员。

可以进一步修改这个程序，使其在用户不知道数据存储的细节和换算关系的情况下，实现其他温度度量下的温度换算。

3.2.1　换算到开氏温标

开氏温标或称绝对温标中的 0°定义为摄氏－273.16°。这是最低的温度，因为在－273.16°时分子停止了运动。可以在上面的程序中增加到开氏温度的转换函数。

```
Public Function getKelvin() As Single
    getKelvin = temperature + 273.16
End Function
```

增加了上面的函数后，原来的用户界面不需要改变，这是因为新定义的类已经把计算与界面分离。上面只给出了 getKelvin 方法，你可以自己编写 setKlvin 方法。

3.3　在 Temperature 类作决定

现在我们仍然在 Temperature 类的计算方法的用户界面中做决定，把一些复杂的处理放到 clsTemp 类中是一种好的方法，这样我们可以重写 Conversion 按钮方法如下：

```
Private Sub btConvert_Click()
Dim clTemp As New clsTemp

'put the entered value and conversion request
'into the class
clTemp.setEnterTemp txTemperature.Text, opFahr.Value

'and get out the requested conversion
lbNewtemp.Caption = clTemp.getTempString

End Sub
```

上面调用界面程序的代码减少到只有两行，而把计算方法确定的程序放到 temperatrue 类。

这个类变得有些复杂，但它接受所有的选择，并且做出转换方法的决定。

```
Private temperature As Single    'always in Celsius
Private toFahr As Boolean        'conversion to F requested

Public Sub setEnterTemp(ByVal tx As String, _
       ByVal isCelsius As Boolean)
'convert to Celsius and save
```

```
 If Not isCelsius Then
  makeCel tx               'convert and save
  toFahr = False
 Else
  temperature = Val(tx)    'just save temperature
  toFahr = True
 End If
End Sub
'------
Private Sub makeCel(tx As String)
  temperature = 5 * (Val(tx) - 32) / 9
End Sub
```

在 temperature **类中，布尔变量** isCelsius **用来确定是否需要换算，以及是否保持该温度值。计算结果输出程序如下：**

```
Public Function getTempString() As String
 getTempString = Str$(getTempVal)
End Function
'------
Public Function getTempVal() As Single
 Dim outTemp As Single
 If toFahr Then           'should we convert to F?
  outTemp = makeFahr     'yes
 Else
  outTemp = temperature 'no
 End If
 getTempVal = outTemp     'return temp value
End Function
'------
Private Function makeFahr() As Single
 Dim t As Single
'convert t to Fahrenheit
 t = 9 * (temperature / 5) + 32
 makeFahr = t
End Function
```

在 temperature **类中有** public **和** private **两种方法，分别是** getTempVal **方法和** makeFahr **方法，方法** getTempVal **是外层模块可调用的方法，如在用户界面中调用了该方法。**makeFahr **和** makeCel **只能在类的内部访问，它们用来计算温度值。**

如果对计算结果的输出格式有严格的要求，可以进一步对程序进行修改。可以按字符串输出，还可以按单精度浮点数值输出，当然还可以根据需要对输出的格式做限制。

3.4 在类中进行数据的格式化和值转换

有时在一个类中用一个方法来实现数据不同格式的转换是很方便的，这样做的好处是可以隐蔽转换处理的细节。例如，你可以用带或不带冒号的分钟或秒的形式输入时间：

```
315.20
3:15.20
315.2
```

因为各种可能的输入形式都可以使用，所以你可以建立一个类来实现时间表示的标准化。图 3-2 就是一个实现不同时间输入的解析界面。

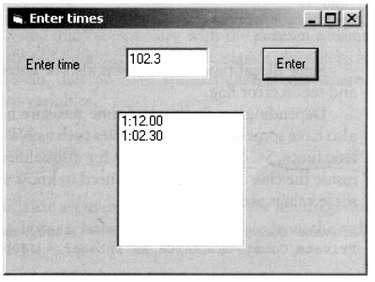


图 3-2　一个简单的使用 Times 类的解析程序

在 Times 类中的访问函数有：

```
setText (tx as String)
setSingle (t as Single)
getSingle as Single
getFormatted as String
getSeconds as Single
```

解析程序十分简单，关键是寻找冒号。如果没有冒号，则当输入值大于 99 时记为分钟。

```
Public Function setText(ByVal tx As String) As Boolean
 Dim i As Integer, mins As Long, secs As Single
  errflag = False
  i = InStr(tx, ":")
  If i > 0 Then
  mins = Val(Left$(tx, i - 1))
   secs = Val(Right$(tx, Len(tx) - i))
   If secs > 59.99 Then
     errflag = True
   End If
   t = mins * 100 + secs
 Else
  mins = Val(tx) \ 100
  secs = Val(tx) - (100 * mins)
  If secs > 59.99 Then
    errflag = True
    t = NT
  Else
   setSingle Val(tx)
  End If
 End If
 setText = errflag
End Function
```

因为任何无效的时间都可能被输入，所以应该建立一个错误标志，这里用 89.22 作为检验数据。

时间格式的转换还与时间计量方法有关，有时可能采用非数字化的时间表示，如 NT 表示 notime，SC 表示 scratch 或 DQ 表示 disqualified。在类中进行这类时间的处理很方便，你无需知道这些非数字时间的内部表示。

```
Private Const tmNT As Integer = 10000, tmDQ As Integer = 20000
Private Const tmSCRATCH As Integer = 30000
```

图 3-3 是实现部分转换处理的窗口

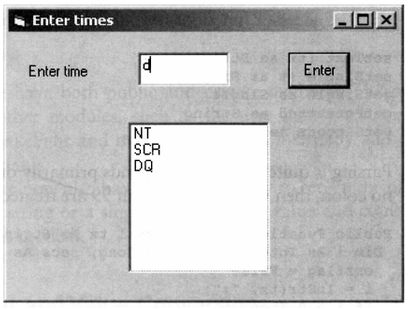


图 3-3 一个时间输入处理界面，显示对符号 NoTime，Scratch 和 Disqualification 的解析

3.4.1 处理不合理值

用类来封装错误处理是一个好方法。例如，输入的时间比某一个时间上限大并且没有小数点，这时可能是一种输入错误，比如输入的时间为 123473，但实际上应该是 12:34.73。这种情况可以采用下面的程序段来处理。

```
Public Sub setSingle(tv As Single)
t = tv
If tv > minVal And tv <> tmNT Then
  t = tv / 100
End If
End Sub
```

时间下限 minVal 是一个变量，它可以在时间域内变化。当类本身没有 Form_Load 事件，那么一定有一个初始化事件，在这可以建立 minVal 的默认值。

```
Private Sub Class_Initialize()
       minVal = 10000
End Sub
```

为了在 IDE 中建立初始化事件，点击左侧文本框的下三角按钮，选择 Class；再在右侧的下拉列表框中选择 Initialize，如图 3-4 所示。

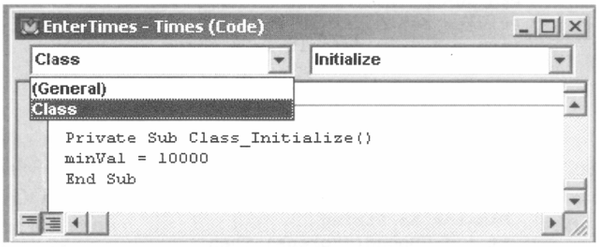


图 3-4　选择类初始化方法

3.5　一个字符串的分割类

多数的程序设计语言提供了简单的字符串的分割方法，但 VB 没有提供这个功能，不过这个功能可以很容易地通过建立一个分解函数来实现。字符串的分割就是把输入的字符串分割成单个的词。例如，现有一个字符串

```
Now is the time
```

分割后返回的 4 个单词为：

```
Now
is
the
time
```

分割算法的关键是保存输入字符，并记住下一个需要返回的单词。

可以用 Instr 函数或 Split 函数实现 Tokenizer 类，与其他语言实现的分割方法不同，这里的 Tokenizer 类返回的是一个单词数组而不是一个类的界面。在 Tokenizer 类中分割处理是用 next-Token 方法来实现的，它返回一个单词，或当搜索到输入字符串的尾部时返回一个空字符串。

一个完整的 Tokenizer 类如下：

```
'String tokenizer class
Private s As String, i As Integer
Private sep As String   'token separator
Private stokens() As String 'array of tokens
Public Sub init(ByVal st As String)
 s = st
 setSeparator " "
End Sub
Private Sub Class_Initialize()
 sep = " "   'default is a space separator
End Sub
```

```
Private Sub Class_Initialize()
 sep = " "    'default is a space separator
End Sub
Public Sub setSeparator(ByVal sp As String)
  sep = sp
  stokens = Split(s, sp)
  i = -1
End Sub
Public Function nextToken() As String
 Dim tok As String
 If i < UBound(stokens) Then
   i = i + 1
   tok = stokens(i)
 Else
   tok = ""
 End If
 nextToken = tok     'return token
End Function
```

图 3-5 是对 Tokenizer 类的解释。

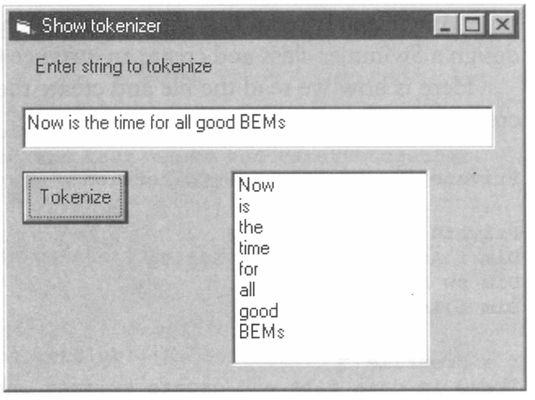


图 3-5 工作中的 Tokenizer 类

下面的代码就是使用 Tokenizer 类的算法。

```
Private Sub Tokenize_Click()
 Dim tok As New Tokenizer
 Dim s As String

 tok.init txString.Text  'set the string from the input
 lsTokens.Clear          'clear the list box
 s = tok.nextToken       'get a token
 While Len(s) > 0        'as long as not of zero length
   lsTokens.AddItem s    'add into the list
   s = tok.nextToken     'and look for next token
 Wend
End Sub
```

3.6　类对象

过程化与面向对象程序设计主要的不同在于，后者引入了类的概念。就像我们前面见到的那样，类就是一个模块，它有 public 和 private 两种方法，并且可以有数据。类可以有多个实例，每一个实例可以有不同的数据。我们通常把这些实例称为对象。下面通过单个实例和多个实例来加以解释。

假设有一个有关游泳数据的文本文件，它的部分数据如下：

```
1 Emily Fenn                17  WRAT        4:59.54
2 Kathryn Miller            16  WYW         5:01.35
3 Melissa Sckolnik          17  WYW         5:01.58
4 Sarah Bowman              16  CDEV        5:02.44
5 Caitlin Klick             17  MBM         5:02.59
6 Caitlin Healey            16  MBM         5:03.62
```

每一列分别表示序号、名字、年龄、所在俱乐部和成绩。如果我们编写一个程序显示游泳者的名字和成绩，必须首先读入这些数据并解析。对每一个游泳运动员，我们需要解析出他的名字、年龄、所在俱乐部和成绩。一个有效的方法是把每个游泳运动员的数据放到一起。为此，我们建立一个游泳运动员类，并为每一个运动员生成一个实例。下面的程序实现读文件和产生类的实例。当实例被产生后，我们把它放入一个 Collection 对象。

```
Private swimmers As New Collection

Private Sub Form_Load()
Dim f As Integer, S As String
Dim sw As Swimmer
Dim i As Integer

f = FreeFile
'read in data file and create swimmer instances
Open App.Path & "\500free.txt" For Input As #f
While Not EOF(f)
  Line Input #f, S
  Set sw = New Swimmer  'create instances
  sw.init S             'load in data
  swimmers.Add sw       'add to collection
Wend
Close #f
'put names of swimmers in list box
For i = 1 To swimmers.Count
  Set sw = swimmers(i)
  lsSwimmers.AddItem sw.getName
Next i
End Sub
```

游泳运动员类 Swimmer 解析每一个数据行，并存储以便 getXXX 访问函数访问。

```
Private frname As String, lname As String
Private club As String
Private age As Integer
Private tms As New Times
Private place As Integer
'------
Public Sub init(dataline As String)
 Dim tok As New Tokenizer

 tok.init dataline            'initilaize string tokenizer
 place = Val(tok.nextToken)   'get lane number
 frname = tok.nextToken       'get first name
 lname = tok.nextToken        'get last name
 age = Val(tok.nextToken)     'get age
 club = tok.nextToken         'get club
 tms.setText tok.nextToken    'get and parse time
End Sub
'------
Public Function getTime() As String
 getTime = tms.getFormatted
End Function
'------
Public Function getName() As String
 'combine first and last names and return together
 getName = frname & " " & lname
End Function
'------
Public Function getAge() As Integer
 getAge = age
End Function
'------
Public Function getClub() As String
 getClub = club
End Function
```

3.6.1 类包含

每一个 Swimmer 类实例，包含一个 Tokenizer 实例和 Times 类的实例，分别用于解析输入字符串和时间，并返回标准格式输出给调用程序。在面向对象程序设计中，类中有类是常见的现象，通过这种方法可以像滚雪球一样地构造更复杂的程序。

显示游泳运动员的程序如图 3-6 所示。

在这个示例中，当点击一个运动员的名字后，在右边的文本框中就会显示他的成绩。显示成绩的代码非常容易实现，因为成绩就在 Swimmer 类中。

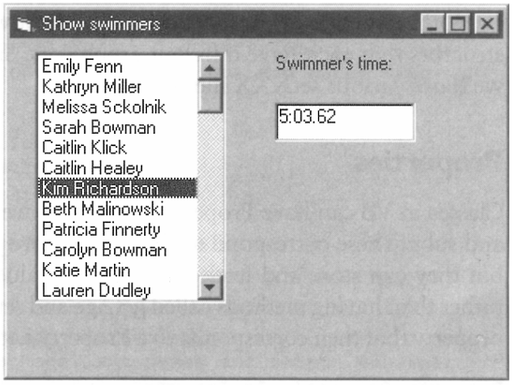


图 3-6　使用类包含建立的一个游泳运动员和他们的成绩的列表

```
Private Sub lsSwimmers_Click()
 Dim i As Integer
 Dim sw As Swimmer
 i = lsSwimmers.ListIndex         'get index of list
 If i >= 0 Then
   Set sw = swimmers(i)           'get that swimmer
   lbTime.Caption = sw.getTime    'display that time
 End If
End Sub
```

3.7　类初始化

正像前面提到的那样，可以使用 Class_Initialize **事件为类变量设置初值。然而，如果你希望为每一个实例（如游泳运动员的名字和成绩）建立初值，就必须有一个标准的方法来实现这样的工作。在其他面向对象程序设计语言中，使用称为构造函数的方法对类实例进行初始化。因为** VB6 **没有提供构造函数，所以这里介绍一种实现类实例初始化工作较常见的方法。**

在 Swimmer **类中，建立一个初始化方法，它依次调用** Tokenizer **类的初始化方法。**

```
Public Sub init(dataline As String)
Dim tok As New Tokenizer

tok.init dataline            'initialize string tokenizer
```

包括 VB7 **在内，其他语言都允许建立带有参数的构造函数，来实现类实例的初始化。因为** VB6 **没有提供构造函数，我们将使用** setXXX **方法来代替构造函数。**

3.8 类和属性

VB 中的类提供了一种称为 Property 方法的方法，这种方法与 public、private 和 sub 函数一样，只是这种方法与窗体 Form 的属性相对应，但是它可以存储、获取各种你需要的属性值。例如，如果有一个属性 Age，则就有相应的属性方法 Property Let 和 Property Get，而不用建立类似于 getAge 和 setAge 一样的方法。

```
Property Get age() As Integer
 age = sAge  'return the current age
End Property
'------
Property Let age(ag As Integer)
 sAge = ag   'save a new age
End Property
```

在使用这些属性时，你可以使用赋值语句，例如：

```
myAge = sw.Age        'Get this swimmer's age
sw.Age = 12           'Set a new age for this swimmer
```

虽然，属性多用于窗体的有关操作，但许多程序员认为它是十分有用的。在没有使用 get 和 set 方法前，属性并未提供任何特性，由 get 和 set 方法及直接设置属性两者产生相同效果的代码。

下面用属性方法对 SwimmerTimes 显示程序进行改进，把所有的 get 和 set 方法用属性方法实现，并允许用户通过输入新的成绩改变已有的成绩。新的 Swimmer 类如下：

```
Option Explicit
Private frname As String, lname As String
Private sClub As String
Private sAge As Integer
Private tms As New Times
Private place As Integer
'------
Public Sub init(dataline As String)
Dim tok As New Tokenizer

 tok.init dataline           'initilaize string tokenizer
 place = Val(tok.nextToken)  'get lane number
 frname = tok.nextToken      'get first name
 lname = tok.nextToken       'get last name
 sAge = Val(tok.nextToken)   'get age
 sClub = tok.nextToken       'get club
 tms.setText tok.nextToken   'get and parse time
End Sub
'------
Property Get time() As String
 time = tms.getFormatted
End Property
```

```
'------
Property Let time(tx As String)
 tms.setText tx
End Property
'-------
Property Get Name() As String
 'combine first and last names and return together
 Name = frname & " " & lname
End Property
'------
Property Get age() As Integer
 age = sAge  'return the current age
End Property
'------
Property Let age(ag As Integer)
 sAge = ag   'save a new age
End Property
'------
Property Get Club() As String
 Club = sClub
End Property
```

当 txTime **文本项字段失去焦点时，我们可以存储一个新时间如下：**

```
Private Sub txTime_Change()
 Dim i As Integer
 Dim sw As Swimmer
 i = lsSwimmers.ListIndex        'get index of list
 If i >= 0 Then
  Set sw = swimmers(i)          'get that swimmer
  sw.time = txTime.Text  'store that time
 End If
End Sub
```

3.9　另一个接口示例——伏特计

假设你需要为计算机加一个数字化伏特计的接口，并假定这个伏特计已经接到串口上，你可以给伏特计发送一个命令，并得到一个电压的返回值。再假定你建立了一个测量范围，如毫伏、伏和 10 **伏，这样访问伏特计的方法如下：**

```
'The Voltmeter class
Public Sub setRange(ByVal maxVal As Single)
  'set maximum voltage to measure
End Sub
'------
Public Function getVoltage() As Single
 'get the voltage and convert it to a Single
End Function
```

你可以使用上面的类框架，编写一个伏特计的仿真程序。

现在让我们认真地思考一下程序接口设计问题。上面的接口设计可以适合任何伏特计

的仿真。你需要做的只是编写实现这样接口的代码。数据收集程序只需知道使用哪种伏特计，相关代码如下：

```
Private Sub OK_Click()
 If opPe.Value Then
    Set vm = New PE2345
 Else
   Set vm = New HP1234
 End If
 vm.getVoltage
End Sub
```

更进一步考虑，你应该扩大数据量以便提供更多的测量值，你可以快速地编写一个实现相同伏特计接口的代码。这就是面向对象程序设计的优点：只有类处理内部的实现细节，接口只处理外部的事务。

3.10 一个 vbFile 类

由于历史的原因，VB 在文件处理上并不十分完美。在 VB 中打开文件的语句为：

```
f = FreeFile
Open "file.txt" for Input as #f
```

从文件中读数据的语句为：

```
Input #f, s
Line Input #f, sLine
```

VB 中没有简单的，用于检测文件是否存在、改名和删除文件的语句。

```
Exists = len(dir$(filename))>0          'file exists
Name file1 as file2                     'Rename file
Kill filename                           'Delete file
```

上述程序段是过程式的处理方式，没有体现面向对象的特点。基于面向对象程序设计风格的文件处理，应该把这些声明包装到一个对象中，屏蔽有关的实现细节。

VB6 用 Scripting.FileSystemObject 来处理文件，这多少有些面向对象的风格。然而，它不是完全面向对象的，并且使用不是很方便。可以建立自己的 VB 文件处理对象，下面就是一个具有面向对象风格的文件处理程序：

```
Public Function OpenForRead(Filename As String) As Boolean
Public Function fEof() As Boolean
Public Function readLine() As String
Public Function readToken() As String
Public Sub closeFile()
Public Function exists() As Boolean
Public Function delete() As Boolean
Public Function OpenForWrite(fname As String) As Boolean
```

```
Public Sub writeText(s As String)
Public Sub writeLine(s As String)
Public Sub setFilename(fname As String)
Public Function getFilename() As String
```

一个典型的实现上面方法的代码如下：

```
Public Function OpenForRead(Filename As String) As Boolean
 'open file for reading
 f = FreeFile                'get a free handle
 File_name = Filename        'save the filename

 On Error GoTo nofile       'trap errors
 Open Filename For Input As #f
    opened = True           'set true if open successful
oexit:
    OpenForRead = opened    'return to caller
Exit Function
'--error handling--
nofile:
  end_file = True           'set end of file flag
  errDesc = Err.Description 'save error messae
  opened = False            'no file open
  Resume oexit              'and resume
End Function
'------
Public Function fEof() As Boolean
 'return end of file
 If opened Then
  fEof = EOF(f)
 Else
  fEof = True    'if not opened then end file is true
 End If
End Function
'------
Public Function readLine() As String
 Dim s As String
 'read one line from a text file
 If opened Then
   Line Input #f, s
   readLine = s
 Else
   readLine = ""
 End If
End Function
```

用这些有用的方法，我们可以实现一个在列表框中读并显示一个文件的简单程序。

```
Dim fl As New vbFile
cDlg.ShowOpen   'use common dialog open

fl.OpenForRead cDlg.Filename
'read in up to end of file
sline = fl.readLine
```

```
While Not fl.fEof
  lsFiles.AddItem sline
  sline = fl.readLine
Wend
fl.closeFile
```

上述代码可以作为原 VB 文件系统的一种改进方法。该方法隐蔽了文件处理的细节，我们可以进一步改进其实现方法，以后的章节中你会看到用 VB.NET 实现的文件处理类。

3.11 Visual Basic 程序设计风格

有许多适合 VB 的程序设计风格可以使用，本书采用的编程风格受到 Microsoft's Hungarian 以它的发明者 Charles Simonyi 命名风格影响，也受到 Java 程序风格的影响。

对 VB 控件，如命令按钮、列表框的命名，采用前缀的方法，这样会使得意义更加明确。只要一个窗体上有多于一个控件时，我们就采用这种命名方法。

控件	前缀	举例
Buttons	bt	btCompute
List boxes	ls	lsSwimmers
Radio(option buttons)	op	opFSex
Combo boxes	cb	cbCountry
Menus	mnu	mnuFile
Text boxes	tx	TxTime

对类的命名时，把类的功能与名字相结合，并在类名前加 clsXXX，以免产生二义性。由于没有在代码中直接引用，所有对 Labels、Frames 和 Form 不重新命名。虽然，VB 对大小写不敏感，但书中类名还是约定以大写字母开头，类实例以小写字母开头。为了使名字更加明确，有时类名和类实例采用大小写混合编码，例如：swimmerTime。

3.12 小结

在这一章中，我们学习了 VB 中有关类的概念，以及如何定义 public、private 等方法和定义类的数据。每个类可以有多个实例，每一个实例的属性值可以不同。类还可以建立属性方法，从而实现对属性的访问。属性方法提供了简单的语法，这一方面优于通常访问方法的命名语法，如 getXXX 和 setXXX，但除此之外，优点不明显。

第 4 章　面向对象的程序设计

面向对象程序设计与早期的过程式程序设计不同。不同之处在于，前者提供了对象的概念。所谓的对象就是过程（方法）与数据的载体。本章介绍面向对象程序的基本思想，以及它在简化程序设计以及提供程序的健壮性方面的优点。

我们所熟悉的过程式的程序，通常包括一些算法、逻辑语句、变量、函数和子程序等程序元素。数据在程序或函数的头部声明，数据通过过程或函数的参数列表导入和导出。

这种过程式的程序设计风格，被采用了相对长的一段时间，但后来发现它有一定的缺点。例如，数据在过程之间通过参数传递过程时，你必要保证数据的格式、类型和内容完全正确。这样当你修改或增加一个新的过程时，过程与过程的调用程序都必须进行反复检测，以免产生错误。

与过程式程序设计不同，面向对象程序设计是把一组过程，以及它们所操作的数据集成到一个对象中。对象通常被设计成一个个程序要处理的实际的物理实体，如客户、订单、帐户和图标等等。

不仅如此，面向对象程序设计有效地实现了对数据操作的封装。在对象内部，数据和对数据的处理是用户不可见的。你可以向对象传递数据，并请求计算，但当你创建并使用对象时，不需要了解对象的内部数据的表现形式和操作细节。

在 VB 中，类只是一个对象的模板。例如，你设计了一个 Customer 对象类，这并不意味着你已经拥有了一个对象。对象是类的实例，要获得对象实例，还要进一步声明。当然，一个类可以产生多个对象实例，产生对象实例的过程称为类的实例化。

由于对象拥有数据，所以可以把对象看成是有状态的。在过程式的程序设计中，一个包含若干函数的模块，即使变量与函数在一个模块内，函数的行为可能也不与模块中的变量相关。然而，当你编写一个类或对象时，你会希望类中的所有方法与变量保持一致，以变量作为类行为的参照，这就是对象具有状态的含义。例如，你可能产生一个文件对象类，它具有打开或关闭文件、判断是否已到文件的结尾等操作（行为），这些行为必须与文件（变量）状态（值）相关。

一旦一个完整的对象被生成，其他人很可能不再修改，只是在此基础上继承。我们在第 5 章讨论派生新对象的概念。

正如我们解释的那样，对象有些像 C 语言中的结构或 Pascal 语言中的记录，只是对象同时拥有函数和数据。然而，对象又是一种结构或数据类型。在编写程序时，我们产生这种类型的变量（对象）。我们把这种变量称之为对象实例。

4.1 构建 VB 对象

让我们看一个简单的示例，设计一个测量距离的对象。现在建立一个子程序来实现具体的测量任务，每次需要计算距离的时候就调用该子程序。

在 VB 中，我们可以用一些对象而不是子程序来实现这样的工作，实现步骤如下：

- 产生一个 TapeMeasure 类。
- 产生这个类的实例，每个实例的大小不同。
- 执行对象实例。

在 VB 中，对象被表示成类模块，每个 VB 类都是可以产生多个实例的对象。在你编写一个 VB 程序的时候，整个程序就是一个或多个类。必须有一个主类来启动程序的运行，这个类必须与程序同名。在这个示例中，程序名为 Measure.cls，主类名为 Measure.frm。

在 VB 中类包含数据和函数（方法）。数据和方法可以有 public 或 private 修饰符，这些修饰限制了外层程序代码的访问权限。通常我们设定变量为私有的，而设定方法为共有的。这样，可以在类模块的外部存储、获取变量，又可以防止在访问变量时无意中改变变量的值。

如果我们希望在类的外部使用类中的方法，就必须把方法设置成 public 类型；否则，就应该设置成 private 类型。一个 VB 程序可以包含任意多个.cls（类）文件和.frm（窗体）文件。

4.2 产生一个对象实例

在 VB 中使用 new 运算符产生一个类实例。例如，为了产生一个类实例 TapeMeasure，编写下列代码：

```
Dim tp as TapeMeasure 'variable of type TapeMeasure

'create instance of TapeMeasure
set tp = new TapeMeasure
```

记住，当你创建一个基础数据类型（如 Integer，Single 等）的对象时，仍然需要使用 new 运算符。这是因为对象在内存中是一块区域。为了预留这块内存区域，必须使用 new 运算符创建一个实例。

4.3 一个用 VB 实现的测量程序

在下面的示例中，有一个完整的 TapeMeasure 类，它包括一个测量程序。

```
'Tape measure class
Private width As Single, factor As Single

Public Sub setUnits(units As String)
'allows units to be cm or feet
 Select Case LCase$(units)
     Case "c":            'centimeters
        factor = 1
     Case "i":            'inches
        factor = 2.54
    Case Else
        factor = 1
     End Select
End Sub

Public Function Measure() As Single
  width = Rnd * 100#
  Measure = width / factor
End Function
Public Function lastMeasure() As Single
  lastMeasure = width / factor
End Function
```

调用程序是 Measurer **窗体，它的代码如下：**

```
Dim tp As New TapeMeasure

Private Sub btMeasure_Click()
txMeasure.Text = Str$(tp.Measure)
End Sub

Private Sub opCM_Click()
tp.setUnits "c"
txMeasure.Text = Str$(tp.lastMeasure)
End Sub

Private Sub opFt_Click()
tp.setUnits "f"
txMeasure.Text = Str$(tp.lastMeasure)
End Sub
```

4.4　对象中的方法

正如前面解释的那样，类中的函数称为方法。这些函数可以是共有的，即可以在类的外部访问；也可以是私有的，即只能在类的内部访问。

4.5　变量

在面向对象程序设计中，通常把类中的变量设定成 private **类型，和我们前面对** width

和 factor 变量的处理一样。然后通过构造函数或专用的 set 和 get 函数实现变量的赋值和访问。这样可以防止类外部的不正确的访问，并且可以在 set 函数中对变量进行完整性检测，从而确保数据的有效性。

当然，可以在 TapeMeasure 类中把变量 factor 设定成 public 类型，并直接对它赋值。

```
tp.factor = 2.54;
```

然而，这样做的缺点是无法防止错误的赋值。如：

```
tp.factor = -50;
```

所以，我们必须使用访问函数，如 setUnits 来保证数据的有效性。

```
tp.setUnits "c"
```

然后，在程序中加上有关访问函数错误检测的代码。

这样，由于 TapeMeasure 类已经存储了最后的测量值，所以你可以用 lastMeasure 方法读取该值。

4.6 参数传值和传址

默认情况下，所有的变量通过参数传递到方法中。换句话说，原始数据可以通过方法被修改和访问。

```
Public Sub setTemp(t As Single)
  t = 5              'changes t in the calling program
End Sub
```

上面的程序是通过传递变量的地址，来改变 t 值。为了避免这种情况发生，可以使用参数声明 ByVal，这使得参数传递只传递变量的值。

```
Public Sub setTemp(ByVal t as Single)
       t = 5  'has no affect on calling program
End Sub
```

4.7 面向对象程序设计中的术语

面向对象的程序具有以下 3 个特性：

- 封装（Encapsulation） 对象中“隐藏”了许多方法。
- 多态（Polymorphism） 许多不同的对象有名字相同的方法，如 Measure 方法。它们可以完成同样的功能，但是实现的方法不同。另外，在一个对象中，有许多方法

可以是同名的，但它们的参数不同。在 VB 中，一个类中不允许存在多个同名的方法带有不同的参数，但在不同的类中允许有多个同名的方法带有不同的参数。

- 继承（Inheritance）　对象可以从其他对象中继承属性和方法，这样你可以从一些简单的对象开始构造更复杂的程序。在 VB6 中，仅仅支持继承性的部分特性，在下一章中，我们将看到，VB6 只通过接口和实现来体现继承性。VB.Net 是一个较完备的面向对象程序语言，在本书的第 7、8 两章将学习 VB.Net。虽然如此，我们仍然可以用 VB6 实现一些复杂的面向对象程序。

第 5 章　创建自己的控件

VB 最强大的功能是它的可视化开发环境（IDE）。利用它可以很容易地实现向窗体拖动控件，并对控件编程。然而，当所需要的控件不存在时，可能认为创建一个适合需要的新控件是相对困难的事情。本章将通过介绍如何方便地从 VB6 的 ActiveX 控件中创建新控件的方法，进一步展示 VB 面向对象的特性。第 7 章，我们将说明如何用 VB.Net 来实现它。

5.1　一个激活的文本

假设我们想建立一个文本实体域，当该文本实体域成为当前焦点时，它的所有的文本被激活。这样当你刚刚按下一个键后，所有的旧文本即刻被新文本取代了。事实上，我们希望在 TextBox 文本框的基础上生成一个新类。然而，VB6 不允许这样做，因为 VB6 不支持继承。正像“四人帮”所言“对象组合优于继承”。

对象组合是封装或包含的同义词。这样，我们可以产生一个包含 TextBox 的新类，使得无论何时控件变为焦点，该类的文本都被激活。

让我们从 VB IDE 开始建立 ActiveX 控件。选择 File|New Project 命令，并且从打开的对话框中选择 ActiveX Control 图标，如图 5-1 所示。

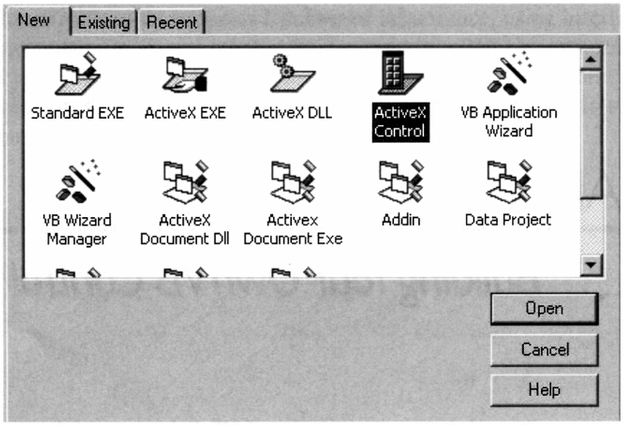


图 5-1　从 VB 工程对话框中选择 ActiveX 控件

这时屏幕上会出现一个称为 UserControl 的没有边框的灰色窗体，它提供了一个画布，

可以在上面拖放控件，如图 5-2 所示。

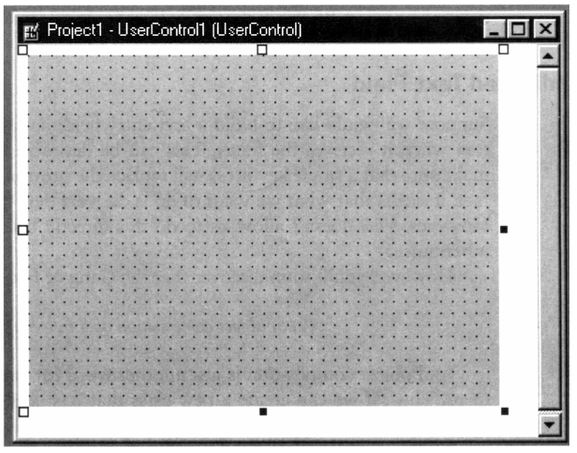


图 5-2　UserControl 画布

首先，按 F4 键并在属性窗口把 UserControl1 改名为 HiText。然后，把 TextBox 放到窗体的左上角，调整背景画布的尺寸，使其与 TextBox 完全重叠，如图 5-3 所示。

图 5-3　在 UserControl 中的文本框

现在，为控件编写一些代码，为 txEntry 框选择 GotFocus 事件，然后加入下列代码。

```
Private Sub txEntry_GotFocus()
Dim s As String
       s = Text1.Text
       txEntry.SelStart = 0          'Start highlight
       txEntry.SelLength = Len(s)    'end highlight
End Sub
```

5.1.1　调整用户控件大小

设计用户控件的另一个重要的问题是，在设计时要考虑调整用户控件的尺寸。选择 HiText 用户控件，并且选择 Resize 事件。输入下面的代码：

```
Private Sub UserControl_Resize()
 txEntry.Width = Width     'design time resize of width
 txEntry.Height = Height   'and height
End Sub
```

5.2 测试 HiText 控件

现在测试一下所设计的控件。关闭设计窗口，然后选择 File|Add project to 命令加入第二个工程用于测试控件。在控件工具栏上，会出现一个新图标表示 HiText 控件，如图 5-4 所示。

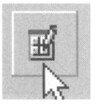


图 5-4 在控件工具栏中的 HiText 控件图标

这个图标仅在设计窗口被关闭后被激活。点击该新图标，并且在刚产生的新工程窗体上放一个控件实例。然后，加入一个名为 Clear 的按钮，如图 5-5 所示。

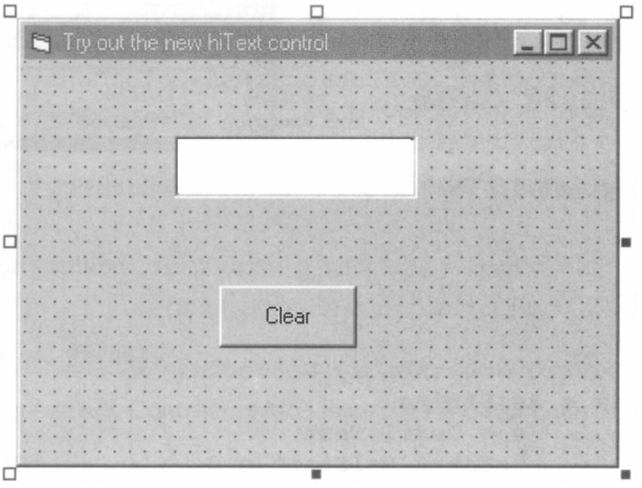


图 5-5 用于测试 HiText 控件的测试工程

如果调整 HiText 控件的大小，可以看到窗体内 TextBox 的尺寸也随之改变。

如果现在运行测试窗体，你可以在 HiText 框中输入字符。如果接着按两次 Tab 键，将把焦点移到 Clear 按钮后，再移回 HiText 控件。当获得焦点后，文本框内的文本被激活，如图 5-6 所示。

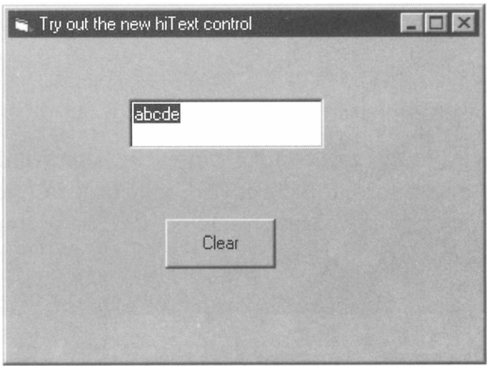


图 5-6　运行状态下的 HiText 控件

5.3　在用户控件中增加属性和方法

你可以为用户控件增加任何属性，如果你增加访问属性，则这些属性将出现在属性框中。例如，你可能希望改变 TextBox 的背景颜色，这时可以加入下面的代码。

```
Property Let Backcolor(c As ColorConstants)
  txEntry.Backcolor = c
  PropertyChanged "BackColor"
End Property
'------
Property Get Backcolor() As ColorConstants
  Backcolor = Text1.Backcolor
End Property
```

你必须增加一个被调用的 PropertyChanged 方法，用于改变用户控件的属性。这个方法传递控制信息到 VB 引擎，从而使屏幕被适当地更新。

用同样的方法，你可以在用户控件中增加方法，这些方法将出现下拉菜单中，并按字母次序排列的，以供控件实例选择。例如，TextBox 控件没有 Clear 方法，但我们可以方便地将 Clear 方法加入到新控件中。

```
Public Sub Clear()
  txEntry.Text = ""
End Sub
```

现在已经加入了一个 Clear 方法，我们可以把这个方法连接到 Clear 按钮上。

```
Private Sub Clearit_Click()
  hiText1.Clear
End Sub
```

点击按钮后文本框被清除。

5.4 编译一个用户控件

在完成了测试程序并确认用户控件工作正常之后，你就可以编译它们。方法是执行 File|Make Project Group 命令进行编译。编译后将产生一个由测试程序生成的.EXE 文件和一个由用户控件生成的.OCX 文件。如果以后还想在程序中使用该控件，你会发现 VB 已经自动执行了注册工作，并且你可以在 Project|Components 菜单下找到该控件，并可将它添加到工程中。

5.5 小结

建立控件的过程帮助你学习面向对象的封装概念，使你了解如何在一个新对象中加入你需要的属性。这种方法的惟一缺点是，你必须手工加入这些属性，而不是自动继承，而在 VB7 中允许自动继承。然而，在许多情况下，封装是十分有效的，因为只有少量的属性从外层控件转移到内层控件。如果一个控件包含多个基础控件，那么本章介绍的方法是惟一的一个途径。

5.6 光盘中的程序

\ActiveX\htx.vbg	为 Dll 和示例激活控件工程组

第 6 章　继承和接口

在开始用类编写程序的时候，你会发现有许多程序代码是相似的。你可能认为重复地拷贝这些相似的代码，并建立各自的对象有些不够巧妙。

在面向对象语言中，如 Java 和 VB.NET，可以采用继承的方法，建立一个新的类，然后再在新类中改变相应的方法或自动继承父类的方法。很遗憾，VB6 没有提供这种类型的继承。但 VB6 提供了接口和实现的方法，也可以方便地实现类似于继承的代码重用。

6.1　接口

在 VB 中，你可以建立一个只有方法定义，而没有方法代码的类，这个类称为接口。然后，你可以另外定义一个类，来实现这个接口中定义的方法。这有些像建立了一个父接口的实例，在这个实例中可以用不同的代码实现与名字相同的方法。

例如，你可以产生一个称为 Command 的接口，它包括下列方法。

```
Public Sub Execute()

End Sub
'------
Public Sub init(nm As String)

End Sub
```

你可以产生许多 Command 对象，如 ExitCommand，来实现 Command 接口。这样你必须在 ExitCommand 中插入下面的一行语句。

```
Implements Command
```

然后，从 VB6 方法设计窗口中左侧的下拉菜单中选择 Command 接口，从右侧下拉菜单中产生一个 Execute 类的实例和 init 方法。现在用下面的代码填充这个方法。

```
Private Sub Command_Execute()
'do something
End Sub
'------
Private Sub Command_init(nm As String)
'initialize something
End Sub
```

这种方法的优点是，ExitCommand 类是 Command 类型，所有实现 Command 接口的类可作为 Command 类的实例。下面的投资模拟器示例将会更好地体现这一优点。

6.2 一个投资模拟器

这个投资模拟程序能够模仿股票、公债按不同时间间隔增长变化的情况。我们假定公债是免税的地方公债，并且股票有正的增长率。

该程序初始阶段有 7 种股票和公债，并且投资底数为$10 000，你可以以任何比例投资到股票和公债中，直到用完所有的储备。最初的程序状态如图 6-1 所示。

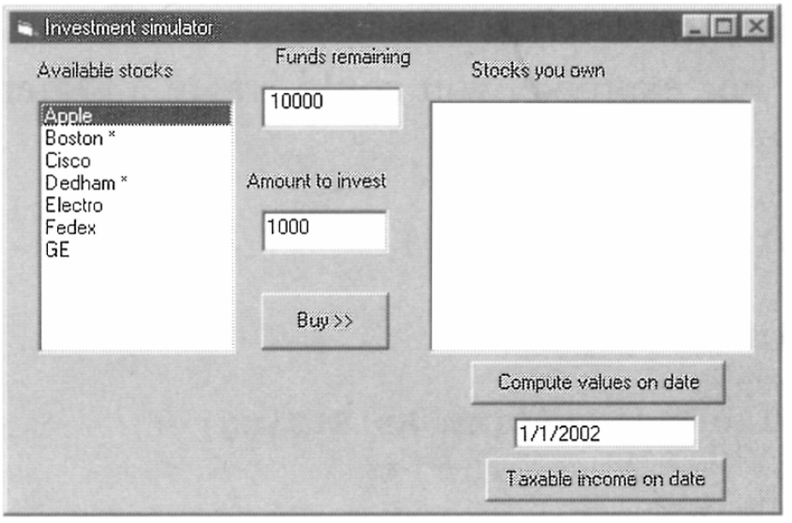


图 6-1 最初状态的投资模拟器

通过点击每一个股票或公债可以选择投资对象。在 Amount to invest 框中输入购买数量后，点击 Buy 按钮确认。

一旦完成了上述操作，你就可以进一步输入数据，并计算总的股票值和总的含税收入。为了简单起见，我们假定股票的收入是含税的，而公债的收入是免税的。一个典型的投资运行结果如图 6-2 所示。

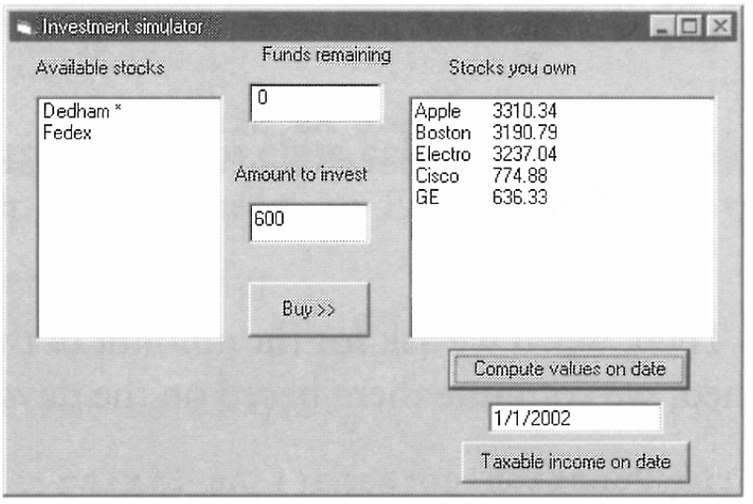


图 6-2　一个典型的投资结果

含税收入如图 6-3 所示。

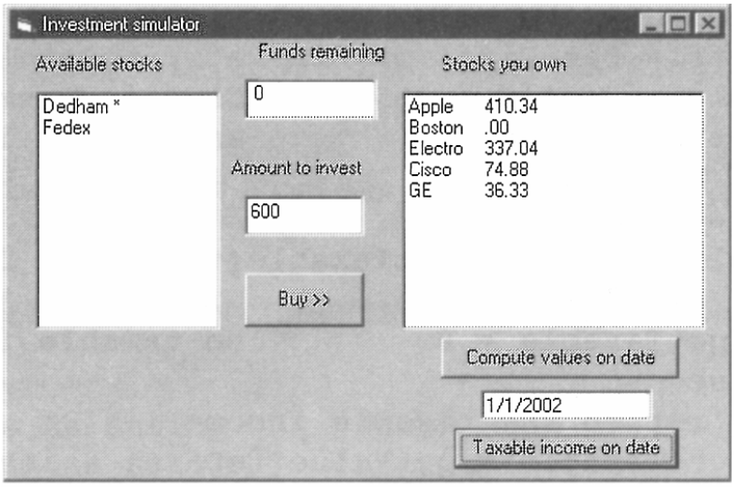


图 6-3　含税收入结果

6.3　编写一个模拟器

在投资模拟程序中，一个重要的类是表现股票的类。该类有一个 init **方法用于设置股票和公债的名称和类型，一个** invest **方法用于确定投资的时间和投资的数量。**

```
Private stockName As String      'name
Private isMuniBond As Boolean    'true of muni bond
Private investment As Single     'amount invested
Private invDate As Date          'date of investment
Private rate As Single           'growth rate
'------
Public Sub init(nm As String, muniBond As Boolean)
 stockName = nm           'save the name
 isMuniBond = muniBond    'and whether a bond
 If isMuniBond Then
   rate = 0.05            'low fixed rate for bonds
 Else
   rate = Rnd / 10        'random rate for stocks
 End If
End Sub
'-----
Public Sub invest(amt As Single)
 invDate = CVDate(Date$) 'remember date
 investment = amt        'and amount invested
End Sub
```

然后，当我们查询投资额和含税利润时，我们可以根据投资的天数计算出查询结果。

```
Public Function getName() As String
 getName = stockName      'return the stock name
End Function
'------
Public Function getValue(toDate As Date) As Single
 Dim diff, value As Single
 'compute the value of the investment
 diff = DateDiff("d", invDate, toDate)
 value = (diff / 365) * rate * investment + investment
 getValue = value         'and return it
End Function
'------
Public Function getTaxable(toDate As Date) As Single
 If isMuniBond Then
   getTaxable = 0         'no taxable income
 Else
   'return the taxable income
   getTaxable = getValue(toDate) - investment
 End If
End Function
```

6.4 用于接口使用的指示器

在前面的程序中有两个地方还需要处理。一个是何时决定投资率，另一个是何时决定返税。在你阅读上面的程序时，发现有处理上述内容的代码，就加一个黄色的标记，以表明这里需要进一步改进。为什么在一个类中要做如此决定？如果每一个类只处理一种类型的投资是否更好呢？如果我们分别建立一个股票类和一个公债类，程序将会变得更复杂。

因为我们已经假定所显示的数据列表都是股票类型。

```
Private Sub Taxable_Click()
 Dim i As Integer, dt As Date
 Dim stk as Stock
 lsOwn.Clear
 dt = CVDate(txDate.Text)
 For i = 1 To stocksOwned.Count
   Set stk = stocksOwned(i)
   lsOwn.AddItem stk.getName & vbTab & _
       Format$(stk.getTaxable(dt), "####.00")
 Next i
End Sub
```

换一个方式，我们可以建立一个称为 Equity **的新类，把** Stock **和** Bond **类作为** Equity **的子类。下面是空的** Equity **接口。**

```
Public Sub init(nm As String)
End Sub
'------
Public Sub invest(amt As Single)
End Sub
'------
Public Function getName() As String
End Function
'------
Public Function getValue(toDate As Date) As Single
End Function
'------
Public Function getTaxable(toDate As Date) As Single
End Function
'------
Public Function isBond() As Boolean
End Function
```

现在我们的股票类变为：

```
Implements Equity
Private stockName As String      'stock name
Private investment As Single     'amount invested
Private invDate As Date          'investment date
Private rate As Single           'rate of return
'------
Private Function Equity_getName() As String
  Equity_getName = stockName     'return the name
End Function
'------
Private Function Equity_getTaxable(toDate As Date) As Single
 'compute the taxable include
 Equity_getTaxable = Equity_getValue(toDate) - investment
End Function
'------
Private Function Equity_getValue(toDate As Date) As Single
 Dim diff, value As Single
```

```
 'compute the total value of the investment to date
 diff = DateDiff("d", invDate, toDate)
 value = (diff / 365) * rate * investment + investment
 Equity_getValue = value
End Function
'------
Private Sub Equity_init(nm As String)
  stockName = nm    'initialize the name
  rate = Rnd / 10   'and a rate
End Sub
'------
Private Sub Equity_invest(amt As Single)
  invDate = CVDate(Date$)   'set the date
  investment = amt          'and the amount
End Sub
'------
Private Function Equity_isBond() As Boolean
  Equity_isBond = False  'is not a bond
End Function
```

公债类除了 getTaxable、init **和** isBond **方法与股票类不同之外，其余基本相似。**

```
Implements Equity
Private stockName As String
Private investment As Single
Private invDate As Date
Private rate As Single
'------
Private Function Equity_getName() As String
 Equity_getName = stockName
End Function
'------
Private Function Equity_getTaxable(toDate As Date) As Single
 Equity_getTaxable = 0
End Function
'------
Private Function Equity_getValue(toDate As Date) As Single
 Dim diff, value As Single

 diff = DateDiff("d", invDate, toDate)
 value = (diff / 365) * rate * investment + investment
 Equity_getValue = value
End Function
'------
Private Sub Equity_init(nm As String)
  stockName = nm
  rate = 0.05
End Sub
'------
Private Sub Equity_invest(amt As Single)
  invDate = CVDate(Date$)
  investment = amt
End Sub
'------
```

```
Private Function Equity_isBond() As Boolean
  Equity_isBond = True
End Function
```

然而，用 Stocks **和** Bonds **来实现** Equity **接口，就像处理所有的** Equity **类一样，因为它们有同样的方法，而不是决定投资的类型。**

```
Private Sub Taxable_Click()
Dim i As Integer, dt As Date
Dim stk as Equity
'Show the list of taxable incomes
 lsOwn.Clear
 dt = CVDate(txDate.Text)
 For i = 1 To stocksOwned.Count
   Set stk = stocksOwned(i)
   lsOwn.AddItem stk.getName & vbTab & _
        Format$(stk.getTaxable(dt), "####.00")
 Next i
End Sub
```

6.5　重新使用共同的方法

再回顾一下前面的程序，不难发现 Stock **和** Bond **类有相同的代码。避免这种情况发生的一种方法是，把共同的方法放到一个基础类中，以便子类调用。然而，最初的想法是在接口模块中只包含一些空的方法。实际上，在** VB6 **中可以不是这样，它也能够间接实现方法的继承。**

例如，我们可以重写基础 Equity **类，并包括一个可以被子类调用的计算利息的方法。**

```
'Class Equity with calcValue added
Public Sub init(nm As String)
End Sub
'------
Public Sub invest(amt As Single)
End Sub
'------
Public Function getName() As String
End Function
'------
Public Function getValue(toDate As Date) As Single
End Function
'------
Public Function getTaxable(toDate As Date) As Single
End Function
'------
Public Function isBond() As Boolean
End Function
```

```
'------
Public Function calcValue(invDate As Date, toDate As Date, _
       rate As Single, investment As Single)
 Dim diff, value As Single

 diff = DateDiff("d", invDate, toDate)
 value = (diff / 365) * rate * investment + investment
 calcValue = value
End Function
```

由于在 VB6 **中没有继承，所以没有办法直接从子类中调用父类方法，但我们可以在** Stock **和** Bond **类中引入一个** Equity **类的实例，然后再调用** calcValue **方法。这样** Stock **类可以简化成下面的程序。**

```
Implements Equity
Private stockName As String      'stock name
Private investment As Single     'amount invested
Private invDate As Date          'date of investment
Private rate As Single           'rate of return
Private eq As New Equity         'instance of base Equity class
'------
Private Function Equity_getName() As String
  Equity_getName = stockName
End Function
'------
Private Function Equity_getTaxable(toDate As Date) As Single
  Equity_getTaxable = Equity_getValue(toDate) - investment
End Function
'------
Private Function Equity_getValue(toDate As Date) As Single
'compute using method in base Equity class
  Equity_getValue = eq.calcValue(invDate, toDate, rate, _
                    investment)
End Function
'------
Private Sub Equity_init(nm As String)
  stockName = nm
  rate = Rnd / 10
End Sub
'------
Private Sub Equity_invest(amt As Single)
  invDate = CVDate(Date$)
  investment = amt
End Sub
'------
Private Function Equity_isBond() As Boolean
  Equity_isBond = False
End Function
```

因为 calcValue **方法是接口的一部分，所以你必须在** Stock **和** Bond **类中定义一个同名的空的方法，以免编译时出错。**

```
Private Function Equity_calcValue(invDate As Date, _
        toDate As Date, rate As Single, _
        investment As Single) As Variant
        'never used in child classes
End Function
```

你也可以建立另一个包括计算方法的辅助类，并在 Stock **和** Bond **类中产生该类的实例，但这不可避免地会导致类关系混乱。**

6.6　隐藏接口

完成上面任务的另一种实现方法是根本不用 Equity **方法而是把** Stock **和** Bond **对象放入一个公共接口中。也就是说，对** Stock **和** Bond **对象不加区分，因为在这个简单的程序中所有的操作通过** Collection **生效，所以可以在不知道** equity **类型的情况下得到一个集合的无素，并且调用它的公共方法。例如，下面是有关集合** stocks **的代码，该集合中包括** Stock **和** Bond **两个对象，代码如下：**

```
For i = 1 To stocks.Count
  sname = stocks(i).getName
 lsStocks.AddItem sname
Next i
```

由于以后不再需要利用这些对象作为特别的类型，所以，这只是一种特殊的情况，在以后的章节中这样的情况将很少出现。

6.7　小结

在这一章中，学习了如何建立一个接口和建立实现接口的类。我们可以把所有的子类看成是接口类的实例，这样可以简化程序设计。

6.8　光盘中的程序

\Inheritance\Basic	**投资模拟器**

第 7 章　VB.NET 简介

VB.NET 也称为 VB7，与低版本的 VB 有相同的基本语法，但它又是一种全新的语言。与早期的 VB 不同，VB7 是完全面向对象的语言。因此，它们在编程方法上有许多地方不同。基于这个原因，VB7 是开发.NET 应用的最好的语言，而不只是一个 VB 新版的编译器。VB.NET 与 VB7 是一个软件的两种称号，在本书的标题中采用 VB.NET，而在段落中使用 VB7，这样既不乏正规，又容易排版。本章我们将看到 VB7 的一些优点，在以后的章节中，我们将通过设计模式，进一步展示 VB7 在面向对象程序设计方面的能力。

7.1　VB.NET 中新的语法

在 VB7 中，你将发现一个最主要的不同是，所有的子程序或类方法的调用必须被放入一对圆括号内。而在 VB6 中，我们可以编写如下代码：

```
Dim myCol As New Collection
MyCol.Add "Mine"
```

然而，在 VB7 中，你必须把 Mine 放入圆括号内。

```
Dim myCol As New ArrayList
MyCol.Add ("Mine")
```

另一个明显的不同是，子程序的参数默认情况下是传数值，而不是传地址。这一点对大多数人来说是一种改进。换句话说，你可以放心地操作子程序中的参数变量的值，这不会改变变量调用程序的原值。这意味着 ByVal 限定符是默认的。实际上，这个限定符是由开发环境自动加入的。如果你真的希望在子程序中改变参数变量的值，你可以在参数变量前加 ByRef 限定符。

从 VB6 中去掉了 3 个关键字，Set，Variant 和 Wend，实际上是开发环境在有 Set 的语句行上去掉了 Set 关键字。

Dim 允许你在声明语句的一行中声明多个相同类型的变量。

```
Dim x, y As Integer
```

也可以在一个声明语句中声明不同类型的变量。

```
Dim X as Integer, Y As Single
```

你也可以在不同的行中声明。

```
Dim X as Integer            'legal in both vb6 and vb7
Dim Y as Single
```

这些变化在表 7-1 中给出总结。

表 7-1　VB6 与 VB7 的语法区别

VB6	VB7
Set q= New Collection	q = New Collection
Dim y as Variant	Dim y as Object
While x<10	While x<10
x=x+1	x=x+1
Wend	End While
Dim x as Integer, y as integer	Dim x, y as Integer
ReDim x(30) As Single	Dim x(30) as Single
	X=New Single(40)
	or ReDim x(40)

另外，字符串函数 Instr，Left 和 Right 有了改进，增加了 indexOf 和字符串类的子字符串方法。在 VB7 中字符串索引从 0 开始计数，如表 7-2 所示。

表 7-2　VB6 与 VB7 字符串函数比较

VB6 或 VB7	VB7
Instr(s, ",")	s.indexof(",")
Left(s,2)	s.substring(0,2)
Right(s,4)	s.substring(s.Length() -4)

7.1.1　改进的函数语法

VB 中有一个特殊的地方就是把函数名作为返回值。

```
Public Function Squarit(x as Single)
  Squarit = x * x
End Function
```

在 VB7 中，这个限制被取消，你可以使用 reture 语句返回计算结果，这与其他许多语句类似。

```
Public Function Squarit(x as Single)
    Return x * x
End Function
```

这样函数更加容易使用。

7.2 变量声明和作用域

正常情况下，VB7 的变量在使用前都用 Dim 语句声明。也可能不这样做，但这是不明智的，这实质上起到了变量错误检测保护的作用。

另外，VB7 允许变量声明与赋值同时进行。

```
Dim age As Integer = 12
```

事实上，这完全改变了程序的风格和变量声明的方法。因为，你可以在使用变量的时候声明，而不是在子程序的头部声明。下面是一个数组元素 a 排序的程序。

```
For i = 1 To UBound(a)
    Dim j As Integer
    For j = i + 1 To UBound(a)
        If (a(i) > a(j)) Then
            Dim temp As Single
            temp = a(i)
            a(i) = a(j)
            a(j) = temp
        End If
    Next j
Next i
'error! temp no longer exists
Console.WriteLine("temp=" + temp)
```

我们在最外层循环内声明变量 j，在内层循环内声明变量 temp。这些变量只存在于被声明的程序块内，在被声明的程序之外，变量不再有效。所以你不能在 j 循环的外层使用变量 j，也不能够在内层循环之外使用变量 temp。如果违背上面的原则，Console.WriteLine 语句将导致一个编译错误，因为 temp 变量不再存在于内层循环之外。

在 VB7 中，不能在两个不同的程序块中声明一个相同的变量。如果在外层循环声明了一个变量，那么你就不能再在该循环的内层循环中声明该变量。

```
Dim temp As Single  'outer declaration

For i = 1 To UBound(a)
    Dim j As Integer
    For j = i + 1 To UBound(a)
        If (a(i) > a(j)) Then
            'error! temp already longer exists
            Dim temp As Single
            temp = a(i)
            a(i) = a(j)
            a(j) = temp
        End If
    Next j
Next i
```

然而，一旦程序从前一个块中退出来，就可以在新的程序块中声明相同的变量。

```
For i = 1 To UBound(a)
            Dim j As Integer 'legal
            For j = i + 1 To UBound(a)
           'some code
            Next j
      Next i
      Dim j As Integer 'legal
```

7.2.1　VB.NET 中的对象

在 VB6 中，对象是类的实例，它包含数据（属性）和对数据处理的方法。而在 VB7 中，所有的变量都看成是对象。

在 VB7 中，字符串变量当然也是对象，该对象有下面一组方法。

```
Substring
ToLowerCase
ToUpperCase
IndexOf
Insert
```

你可能发现，字符串对象没有与数值型转换的函数。实际上，你可以通过 CInt、CSng 和 CDbl 函数来实现转换功能。

整数、单精度和双精度浮点数变量也是对象，它们有下列方法：

```
Dim s as String
Dim x as Single

x =  12.3
s = x.ToString              'convert to String
x = CSng(s)                 'convert to Single
```

注意，字符串和数值型变量的转换最好采用上述 3 种方法，而不要使用 Val 和 Str 函数。如果你希望把数值型变量按照一定格式转换成字符串，可以采用下面的 Format 函数。

```
Dim s as String
Dim x as Single
x = 12.34

s = Single.Format(x, "##.000")
```

7.3　编译选择

如果你对 VB6 比较熟悉，你就会知道编译器 Option Explicit 的设置。这意味着，在使用一个变量之前，必须用 Dim 语句加以说明。VB7 引入了 Option Strict On 声明，它要求对变量进行严格的类型检查，并在不同类型变量之间自动进行类型转换。VB7 中的默认设置是 Option Explicit，但你可以为整个工程设置编译选项 Option Strict 为 On。方法是在工程浏

览器窗口右键点击工程名，选择 Bulid 后再从 Option Strict 下拉菜单中选择 On，如图 7-1 所示。

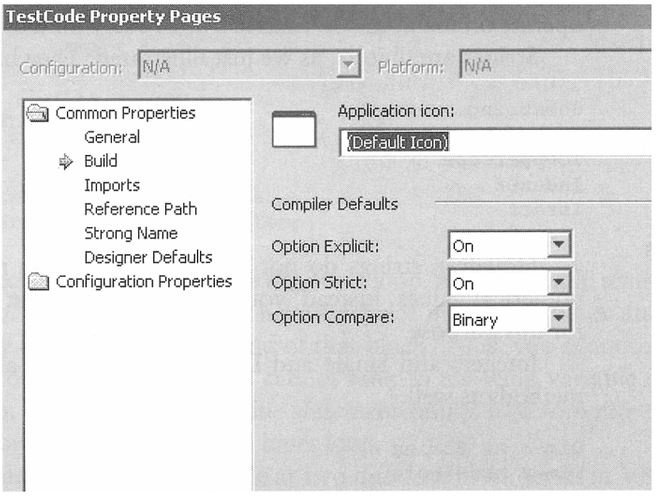


图 7-1 在 Properties Pages（属性页）对话框中设置 Option Strict

7.3.1 VB.NET 中的数值型变量

在 VB7 中，所有没有小数点的数都被假定为整数或 Int32（32 位无符号整数），所有带有小数点的数都被假定为双精度浮点数。

正常情况下，VB.NET 编译器被设置成 Option Explicit，以便阻止没有被声明的类型转换。你可以将整型变量转换为单精度或双精度浮点数。但是如果你想把单精度浮点数转换为整型就必须使用转换函数。这些函数有 CInt，CSng，CDbl，CLng，CDate 和 CStr。

```
Dim k As Integer = CInt(time)
Dim time as Single = CSng("3.3")
```

在 VB7 中，Currency 类型不再是变量。当然，你可以使用小数类型，以便提供精度。VB7 有一个 Convert 类，它提供了许多实现类型转换的共享方法。

```
Dim k As Integer = Convert.ToInt32(s)
```

7.4 VB6 和 VB.NET 中的属性

Visual Basic 提供与其他语言类似的 getXxx 和 setXxx 方法的属性。在 VB6 中，你可以说明一个属性并且定义 Get 和 Let 方法。这两个方法用于设置、读取私有变量的值。

```
Property Get fred() As String
  fred = fredName
End Property

Property Let fred(s As String)
  fredName = s
End Property
```

当然，你可以通过事先处理和事后处理数据的方法来验证它们，或者根据需要从其他窗体导入。

在 VB7 中，属性组合到单一的子程序中。

```
Property Fred() As String
    Get
        Return fredName
    End Get
    Set
        fredName = value
    End Set
End Property
```

请注意一个特殊的关键字 value。在 VB7 中，它用来指示一个传入属性的值。如果我们写下面的语句：

```
Abc.Fred = "dog"
```

当执行该语句后，value 的值为“ dog ”。

无论采用那种方式，这两个系统都是在实现同一目的，就是提供一个简便的接口，以便在类中在不必了解数据的实际存储方式的情况下访问这些数据。你可以在赋值语句中使用属性，如果 Fred 是一个 Boy 类的属性，可以写下面的语句：

```
Dim kid as Boy
Kid.fred = "smart"
```

也可以这样写：

```
Dim brain as string
Brain = kid.fred
```

总之，属性机制提供了一种可以代替 getFred 和 setFred 函数的语法。这使编程更容易一些，但在语法上与其他 VB 版本不同。除了 VB 对象有属性而不是 get 和 set 方法之外，VB 没有明显的优点。在本书中，没有过多地使用属性，因为它在面向对象编程方法没有体现出优点。

7.5　快捷等号语法

VB7 采用 C，C++，C#和 Java 也都使用的快捷等号语法，它允许你在一个变量加、减、

乘和除一个常量时，不用两次写变量。

```
Dim i As Integer = 5
        i += 5  'add 5 to i
        i -= 8  'subtract 8 from i
        i *= 4  'multiply i by 4
```

这是一种节省代码编写量的方法，它与下面的代码等价。

```
i = i + 5
i = i - 8
i = i * 4
```

除法的应用也一样，但由于除法在读起来有一点不上口，所以书中很少使用。

7.6 管理语言和垃圾回收

VB.NET 和 C#都是管理语言，这有两层含义。其一，它们都被编译成一个低级语言代码，而后，使用公共语言运行时（common language runtime，CLR）来执行这个编译好的代码，也可能进一步对它进行编译处理。所以 VB7 和 C＃不仅有相同的运行时库，而且在其他方面有许多相似之处，是同种语言系统的两个方面。它们之间的不同主要体现在，VB7 是 VB 的后续版本，更容易学习和使用；C#与 C++和 Java 类似，对 C++和 Java 程序员更具有吸引力。

其二，管理语言都有垃圾回收机制。垃圾回收语言严格监控无用资源的释放，这样你就不用关心无用资源状态。当一个垃圾回收系统发现一个变量、数组和对象不再活动，即刻把它们所占用的内存返回给系统。因为所分配的内存并未释放，所以你不用担心内存被耗尽。当然，你还能编写内存使用的代码，只是你不用再过多的顾及内存的分配与释放问题。

7.7 VB.NET 中的类

VB7 中类是十分重要的概念。每一个程序都是由一个或多个类组成。VB7 中已经没有窗体和类之分，VB7 程序全部由类组成。正是由于这个原因，有些类对象的名字十分重要。它们包含在函数库中，在使用库中函数之前，必须声明。

每一个类库都是 DLL 文件。可以通过一个 Import 语句来使用类名和类中的函数。

```
Imports System.IO       'Use File namespace classes
```

从逻辑上讲，每一个类库表示不同的名字空间（nameSpace）。每一个名字空间分成类名和方法名两组，它们在导入过程中由编译器加以区分。你可以导入包含相同名字空间的类和方法，但只有在两个完全相同的类或方法被使用时，才提示你发生了冲突。

你可以使用 Property Pages 对话框，说明在工程文件中希望导入到类中的名字空间，如图 7-2 所示。

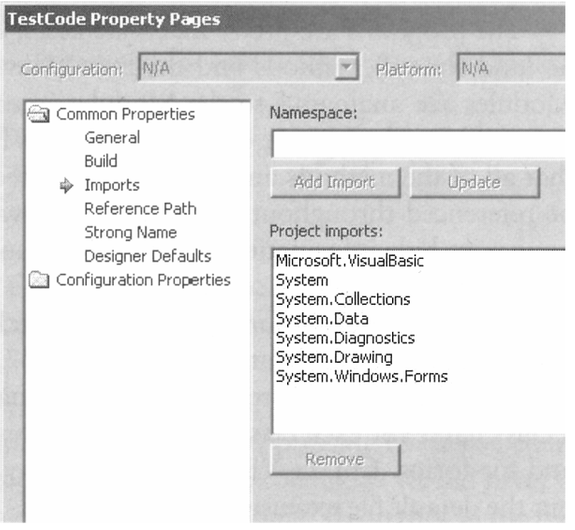


图 7-2　使用 Property Pages 对话框导入名字空间，图中的名字是默认设置

最普通的名字空间是系统名空间，它是被自动导入的，不需要声明。它包括一些 VB7 用来访问基础类的基本类和方法，基础类有应用、数组、控制台、异常、对象，其中标准对象包括字节型、布尔型、单精度和双精度浮点数，以及字符型。一个在控制台上显示 hello 的简单 VB7 程序如下，在这个程序中没有使用窗体。

```
'Simple VB Hello World program
Public Class cMain
   Shared Sub Main()     'entry point is Main
      'Write text to the console
      Console.WriteLine ("Hello VB World")
   End Sub
End Class
```

该程序只写“Hello VB World”到命令行窗口（DOS）。在 VB7 中，任何一个程序的入口必须是 Sub Main 子程序，在类模块中它必须被声明是 Shared 型。在 VB7 中其他模块的类型是 Module 类型，下面是一个实例。

```
'Simple VB Hello World program
Public Module cMain
   Sub Main()     'entry point is Main
      'Write text to the console
      Console.WriteLine ("Hello VB World")
   End Sub
End Module
```

上面的两个程序几乎完全相同，只是第二个程序是 Module 型。这意味着，所有的方法都是共享的（shared），Sub Main 可以不必再加 Shared 说明。Module 类型模块与早期的 VB 版本中的模块说明相似，都以.bas 为扩展名。不同的是用 Module 说明的模块中的变量和方法是共享的，在整个程序中都可访问。但是，它与类（Class）不同，它没有隐蔽信息或算法的功能，所以在本书中不再使用。

在 VB6 中，类名被声明成文件名，当然，你可以改变类名。在 VB7 中，关键字 Class 允许类名与文件名无关，每个类有一个类名。VB6 中类文件的扩展名是.cls，窗体的扩展名是.frm。VB7 中默认情况下，类文件的扩展名是.vb，你可以按自己的想法改变。

7.8 构建一个 VB7 应用

让我们从构建一个简单的控制台应用开始，这个控制台应用没有任何窗口，是直接从命令行运行的程序。运行 Visual Studio.NET 程序，并且执行 File|New Project 命令。从右侧的 Templates 框中，选择 Console Application，如图 7-3 所示。

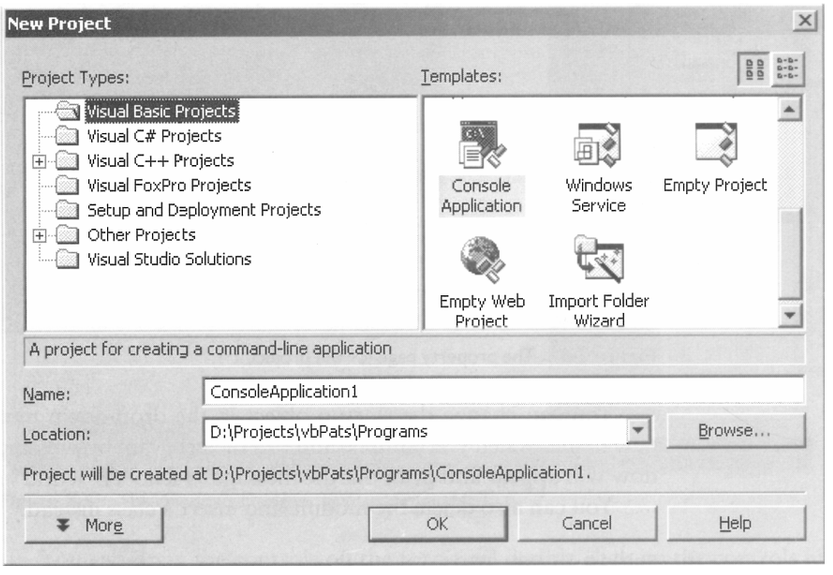


图 7-3 在 New Project 对话框中选择 Console Application

这样就产生了一个模块，该模块已经自动加入一个 Sub Main 语句，你只要键入其他代码即可。

```
Module cMain
    Sub Main()
        'write text to the console
        Console.WriteLine("Hello VB world")
    End Sub
End Module
```

按 F5 键编译并运行该程序。如果你像上面的程序一样，把程序的主模块名从 Module1 改成 cMain，这意味着程序的启动模块名也已经同时被改变。为了改变启动模块名，要在右边的 Solution Explorer 窗口右键点击工程名，并从弹出菜单中选择属性。如图 7-4 所示。

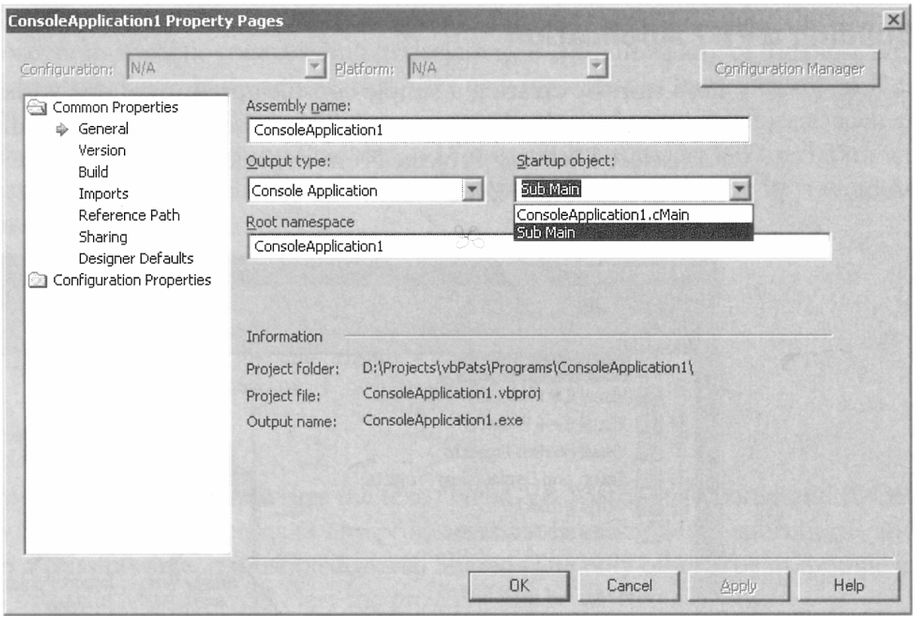


图 7-4　工程属性页

你可以在下拉菜单中选择一个正确的启动对象名。当按 F5 键进行编译并运行该程序时会出现一个 DOS 窗口，并在屏幕上显示“ Hello VB World ”后返回。

你可以把 Module 用 Public Class 替代。

```
Public Class cMain
    Shared Sub Main()
        Console.WriteLine("Hello classy VB world")
    End Sub
End Class
```

这两个程序编译运行的过程是相同的。

7.9 VB.NET 最简单的窗口程序

编写一个窗口程序是十分简单的。事实上，可以使用窗口设计器（Windows Designer）产生许多窗口。为了使用窗口设计器，你首先启动 Visual Studio.NET，并选择 File|New Project 命令，然后选择 Windows Application。默认的工程文件名是 WindowsApplication1，但是你可以在关闭 New 对话框之前改变文件名。WindowApplication1 中包括一个称为 Form1.vb 的窗体。你可以像在 VB6 中一样，利用 Toolbox 在窗体上放入控件。

一个带有两个标签，一个文本字段和一个按钮的简单窗体的 Windows Designer 如图 7-5 所示。

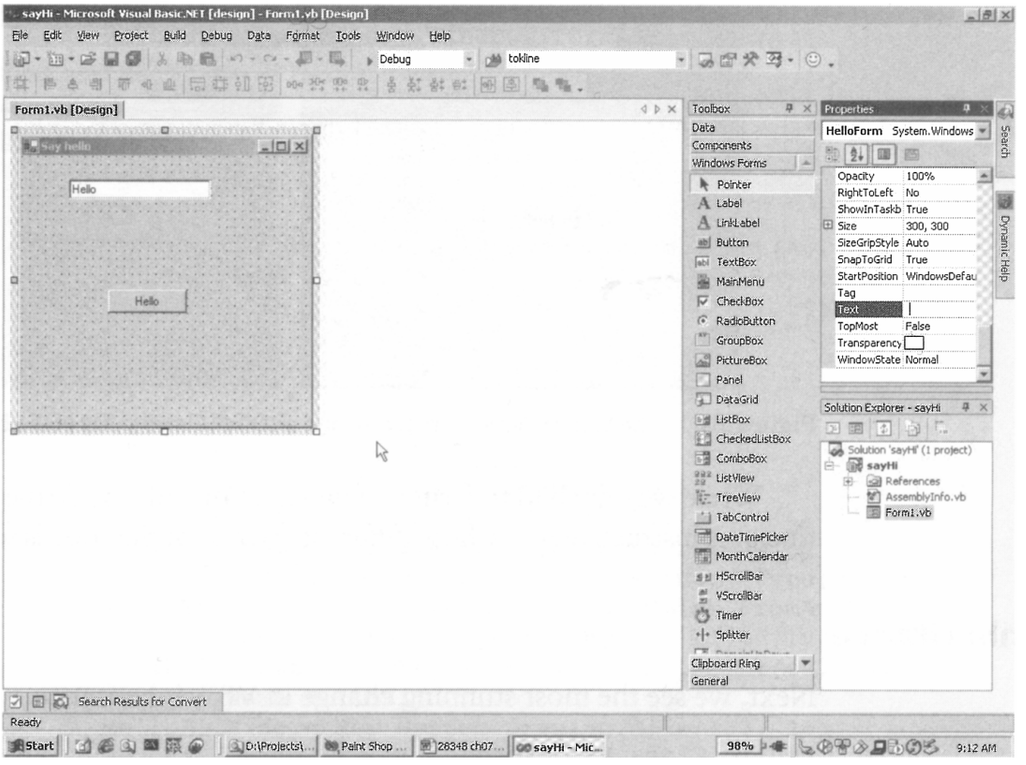


图 7-5 Visual Studio.NET 窗口设计器

你可以在窗体上放入控件后，双击该控件进入控件编程环境。在本例中，点击“Say hello”按钮，它从文本字段拷贝相应的文本到空的标签，该标签我们命名为 1bHi 并清除文本字段。

```
Protected Sub SayHello_Click(ByVal sender As Object, _
                             ByVal e As System.EventArgs)
        lbhi.Text() = txhi.Text
        txhi.Text = ""
End Sub
```

在该程序中，Click 子程序传入一个 sender 对象和一个事件对象，它们可以获得进一步的信息。程序运行如图 7-6 所示。

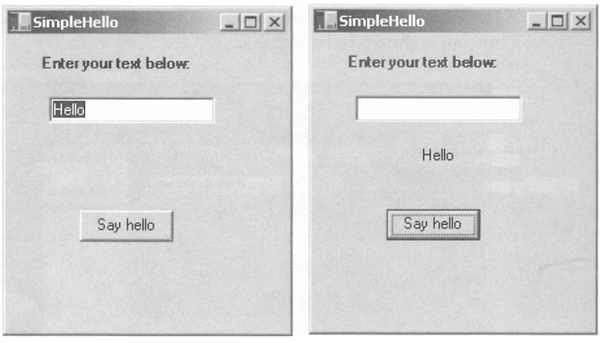


图 7-6　点击 Say hello 按钮前后 SimpleHello 窗体变化情况

在上面的子程序中，我们只列出两行代码，读一下该程序的其他代码是有帮助的。

7.10　继承

现在来学习 VB7 中改进最明显的地方——继承。

```
Public Class HelloForm
    Inherits System.Windows.Forms.Form
```

这个语句产生了一个 Form 类的子类，而不是像在早期的 VB 版本那样产生一个实例。这一点有十分重要的意义。你可以产生一个可视化的对象，并重载其中的属性，使得这些对象有不同的行为。我们将在下面的内容中见到此类示例。

使用设计工具产生一个控件的代码对初学者是有好处的。在 VB7 中，不再采用这种方式，因为该方式下产生的代码不容易修改。VB7 中采用更开放的代码编写方式，它可以手工编写程序而不是使用属性页，这时窗口设计工具将不起作用。这就是为什么该部分最初通过“ [+] ”爆炸框来显示代码。

采用手工编程时，每一个控件被声明为一个变量，这些控件被加入到一个容器中。下面是控件声明的代码。

```
Friend WithEvents txHi As System.Windows.Forms.TextBox
Friend WithEvents lbHi As System.Windows.Forms.Label
Friend WithEvents Button1 As System.Windows.Forms.Button
```

```
    'Required by the Windows Form Designer
    Private components As System.ComponentModel.Container
    <System.Diagnostics.DebuggerStepThrough()>
Private Sub InitializeComponent()
        Me.txHi = New System.Windows.Forms.TextBox()
        Me.lbHi = New System.Windows.Forms.Label()
        Me.Button1 = New System.Windows.Forms.Button()
        Me.SuspendLayout()
        '
        'txHi
        '
        Me.txHi.Location = New System.Drawing.Point(48, 24)
        Me.txHi.Name = "txHi"
        Me.txHi.Size = New System.Drawing.Size(144, 20)
        Me.txHi.TabIndex = 0
        Me.txHi.Text = "Hello"
        '
        'lbHi
        '
        Me.lbHi.Font = New System.Drawing.Font(
              "Microsoft Sans Serif", 14.25!,
              System.Drawing.FontStyle.Regular,
              System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.lbHi.ForeColor = System.Drawing.Color.FromArgb(
              CType(0, Byte), CType(0, Byte), CType(192, Byte))
        Me.lbHi.Location = New System.Drawing.Point(48, 64)
        Me.lbHi.Name = "lbHi"
        Me.lbHi.Size = New System.Drawing.Size(152, 24)
        Me.lbHi.TabIndex = 1
        '
        'SayHello
        '
        Me.SayHello.Location =
              New System.Drawing.Point(88, 136)
        Me.SayHello.Name = "SayHello"
        Me.SayHello.Size = New System.Drawing.Size(80, 24)
        Me.SayHello.TabIndex = 2
        Me.SayHello.Text = "Hello"
        '
        'Form1
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(292, 273)
        Me.Controls.AddRange(
        New System.Windows.Forms.Control()
                      {Me.Button1, Me.lbHi, Me.txHi})
        Me.Name = "Form1"
        Me.Text = "Say hello"
        Me.ResumeLayout(False)

    End Sub
```

在上面的程序中，用 WithEvents 修饰符说明 Say hello 按钮，这意味着建立了按钮与 SayHello_Click 子程序的连接。

7.11 构造函数

所有的类都有一个构造函数，该构造函数被用来产生了一个类的实例。无论是窗体类还是非可视化的类，构造函数有一个专用的名字 New。下面是一个 hello 窗口的构造函数。

```
Public Sub New()
  MyBase.New()
 'This call is required by the Windows Form Designer.
  InitializeComponent()
 'Add any initialization after the InitializeComponent() call
End Sub
```

注意 MyBase.New 方法的调用，它调用父类的构造函数进行实例的初始化。即使没有 MyBase.New 语句，该调用也是自动完成的。

在启动应用程序时，经常需要对一些变量和其他类进行初始化的工作。自然应该把初始化代码放在 New 方法的下面。然而，New 方法经常被隐蔽在一个“+ -”区域，所以我们采用一种变通的方法。New 方法总是调用 init()，并把 init()函数的调用语句放到 New 方法的下面。然后把 init()代码放到更显著的地方。下面是它的示例程序。

```
    Private Sub init()
        lbHi.Text = "..."
    End Sub
#Region " Windows Form Designer generated code "

Public Sub New()
    MyBase.New()
        'This call is required by the Windows Form Designer.
        InitializeComponent()
'Add any initialization after the InitializeComponent() call
        init()
End Sub
```

当建立了一个新类，你就可以定义一个 New 方法，并通过参数对类中的属性初始化。假设你想建立一个在第 3 章中定义过的 StringTokenizer 类。在 VB7 中有一个与 VB6 相似的 Split 函数，它可以分解一个字符串，返回的结果是一个子字符串数组，所不同的是 VB7 要求输入的参数是字符数组，而不是一个字符串。

```
sarray = s.Split(sep.ToCharArray)
```

在 VB7 实现的 Tokenizer 类中，将使用这个方法。

下面使用真正的构造函数，而不是 init（）方法。这个构造函数将拷贝字符串到内部变量，并为字符分离器产生一个默认的值。

```
Public Class StringTokenizer
    Private s As String
    Private i As Integer
    Private sep As String   'token separator
    Private sArray() As String
    '------
    Public Sub New(ByVal st As String)
        s = st          'copy in string
        set_Separator(" ")        'default separator
    End Sub
    '------
    Public Sub New(ByVal st As String, _
            ByVal sepr As String)
        s = st
        set_Separator(sepr) 'creates the array
    End Sub
    '------
    Private Sub set_separator(ByVal sp As String)
        sep = sp       'copy separator
        'and create the array
        sarray = s.Split(sep.ToCharArray)
        i = 0
    End Sub
    '------
    Public Sub setSeparator(ByVal sp As String)
        set_separator(sp)
    End Sub
    '------
    Public Function nextToken() As String
        Dim j As Integer
        j = i
        i = i + 1
        If j < sarray.Length Then
            Return sarray(j)
        Else
            Return ""
        End If
    End Function
End Class
```

下面的调用程序简单地产生一个 Tokenizer **类的实例，并在控制台屏幕上输出单词。**

```
'illustrates use of tokenizer
Public Class TokTest
    Shared Sub Main()
        Dim s As String
        Dim tok As New StringTokenizer("Hello VB World")
        s = tok.nextToken()
        While (s <> "")
            Console.writeLine(s)
            s = tok.nextToken()
        End While
    End Sub
End Class
```

在上面的程序中，可以看到 VB7 允许在声明一个变量的同时初始化该变量。

```
Dim tok As New StringTokenizer("Hello VB World")
```

7.12　VB.NET 中的图画

在 VB7 中，控件是被窗口系统画到屏幕上去的，你可以重载 OnPaint 事件，重新画图。原理是把 PaintEventArgs 对象传递到子程序中，这样就可以获得画图接口。为了作画，你必须产生一个 Pen 对象的实例，并定义它的颜色和它的宽度。下面是一个宽度的磅值为 1 的黑画笔的示例。

```
Protected Overrides Sub OnPaint(ByVal e as PaintEventArgs)
  Dim g as Graphics = e.Graphics
  Dim rpen As new Pen(Color.Black)
  g.drawLine(rpen, 10,20,70,80)
End Sub
```

重载关键字 Override 是 VB 继承性中十分关键的，通过使用关键字，编译器才获知父类中的方法将要被重载。

7.13　工具标签和鼠标移动键

工具标签（Tooltips）是一个解释帮助信息，该信息在把鼠标放在一个控件上移动时出现。在 VB6 中每一个控件都有 Tooltips 属性。在 VB7 中，首先要为窗体产生一个 Tooltip 对象的实例，然后再在窗体上加入带有解释信息的控件。

```
Dim tips As New ToolTip()
tips.SetToolTip(Button1, "Click to Execute")
```

鼠标移动键（Cursor）是一个鼠标位置的可视化指示器。它的默认图标是一个箭头。只有一个图标显示在屏幕上，但你可以在 Cursor 对象的 Current 属性中重新设置。你可以在 Cursors 对象中选择鼠标移动键的类型。

```
Cursor.Current = Cursors.WaitCursor
Cursor.Current = Cursors.Default
```

常见的鼠标移动键名如下：

```
Cursors.Arrow
Cursors.Cross
Cursors.Default
Cursors.Ibeam
Cursors.No
Cursors.SizeAll
```

```
Cursors.UpArrow
Cursors.WaitCursor
```

7.14 重载

像其他的面向对象语言一样，在 VB7 中可以定义同名的类方法，只要它们的调用参数不同。例如，可以产生一个 StringTokenizer 的实例，该实例定义了字符串和分隔符两个参数。

```
tok = New StringTokenizer("apples, pears", ",")
```

如果想实现这个构造函数，我们可以重载该构造函数，这时编译器就会知道有两个方法是同名的，但有不同的参数。下面就是这两个构造函数。

```
Public Sub New(st as String, sepr as String)
  s = st
  sep = sepr
End Sub
'-------
Public Sub New(st As String)
 s = st          'copy in string
 sep = " "       'default separator
End Sub
```

VB 允许重载任何方法，只要提供编译程序得以区分不同重载（或多态）方法的参数。在定义子程序或函数时，你可以使用 OverLaods 关键字说明有两个方法同名，但参数不同。如果你没有使用 Overlaods 关键字，编译系统会报错。

7.15 继承

在 VB7 中，一个最鲜明的特点是，它可以在一个类基础上产生新类。在新类中我们只要说明新方法和发生改变的方法，其他特性可以从它的基类继承。为了更好地理解继承，让我们编写一个简单的长方形类，该类在屏幕上画出它自己的形状。该类仅有两个方法，constuctor（构造）方法和 draw（画图）方法。

```
Namespace VBPatterns
Public Class Rectangle
    Private x, y, h, w As Integer
    Protected rpen As Pen
    '-------
Public Sub New(ByVal x_ As Integer, _
                ByVal y_ As Integer, _
                ByVal h_ As Integer, _
                ByVal w_ As Integer)
```

```
        x = x_
        y = y_
        h = h_
        w = w_
        rpen = New Pen(Color.Black)
    End Sub
    '--------
    Public Sub draw(ByVal g As Graphics)
        g.DrawRectangle(rpen, x, y, w, h)
    End Sub
  End Class
End Namespace
```

7.16　名字空间

我们曾经提到过系统名字空间。VB7 为每一个工程产生一个与工程名相同的名字空间。你可以参见图 7-4 生成的一个默认的名字空间，你也可以在属性页中改变名字空间，或者将它设置为空，使得工程不在名字空间内。你还可以为自己产生一个名字空间，Rectangle 类是一个好的示例。现在 Rectangle 类已经在程序引入的 System.Drawing 名字空间中。我们可以通过把类包装在名字空间的办法，把 Rectangle 类放入名字空间，而不是用改名的方法来避免名字冲突。

当我们在主窗体中声明一个变量时，就把该变量声明为名字空间的一个成员。

```
Public Class Rect_Form
    Inherits System.Windows.Forms.Form

    Private rect As VBPatterns.Rectangle
```

就像以前一样，我们调用 New 方法对主窗体初始化，我们可以产生一个 Rectangle 类的实例。

```
Private Sub init()
        rect = New VBPatterns.Rectangle(20, 50, 100, 200)
End Sub
```

然后，我们启动 OnPaint 事件绘画，并把图形接口导入 Rectangle 实例。

```
Protected Overrides Sub OnPaint( _
            ByVal e As PaintEventArgs)
        Dim g As Graphics
        g = e.Graphics
        rect.draw(g)
End Sub
```

该程序给我们一个如图 7-7 所示的显示窗口。

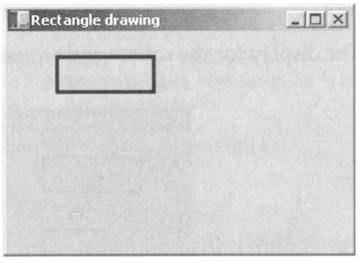


图 7-7 画长方形程序

7.16.1 从长方形中产生一个正方形

正方形是四边形的一个特例，我们可以从 Rectangle 类中导出 Square 类而不用编写代码，下面是一个完整的类。

```
Namespace VBPatterns
    Public Class Square
        Inherits Rectangle
        Public Sub New(ByVal x As Integer, _
            ByVal y As Integer, ByVal w As Integer)
            MyBase.New(x, y, w, w)
        End Sub
    End Class
End Namespace
```

Square 类只包含一个构造函数，它通过调用父类（Rectangle 类）的构造函数，向父类传递正方形边的长度。然后 Rectangle 类产生一个画笔，并画出正方形。实际上，Square 类中根本没有 draw 方法。如果你没有说明一个新的方法，父类的方法就会自动执行。

在一个既画长方形，又画正方形的程序中有一个简单的构造函数，这个构造函数创建了两个对象实例，然后调用 init（初始化）方法。

```
Private Sub init()
        rect = New VBPatterns.Rectangle(20, 50, 100, 200)
        sq = New VBPatterns.Square(70, 80, 50)
    End Sub
```

该程序也有一个 OnPaint 子程序，负责具体的画画的动作。

```
Protected Overrides Sub OnPaint( _
            ByVal e As PaintEventArgs)
   Dim g As Graphics
   g = e.Graphics
   rect.draw(g)
   sq.draw(g)
End Sub
```

显示正方形和长方形的屏幕如图 7-8 所示。

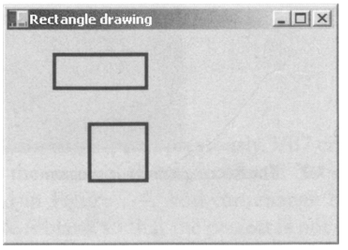


图 7-8　Rectangle 类和导出的 Square 类

7.17　Public，Private 和 Protected

在 VB6 中，你可以把变量和类方法声明成 public 和 private 类型。一个 public 型的方法是可以在类的外部访问的，而 private 型的方法只可以在类的内部访问。一般情况下，把变量设置成 private，并通过 getXXX 和 setXXX 访问函数设置或获得变量的值。因为对象的封装性，所以把变量设置成 public 是违背面向对象思想的。换句话说，类是表示实际数据的惟一所在，并且可以在类的内部改变算法而不需要考虑类之外的事情。

在 VB7 中引入了 protected 关键字，它可以用来修饰变量和方法。protected 型的变量可以在类的内部和在子类中被访问。与变量相似，protected 方法可以被类的内部成员访问，也可以被类的子类访问，其他类无法访问到 protected 方法和变量。

7.18　在导出类中重载方法

假设我们想从 Rectangle 类中导出一个 DoubleRect 类，该类用两种颜色画一个长方形。在构造函数中我们产生一个红色画笔。

```
Namespace VBPatterns
    Public Class DoubleRect
        Inherits Rectangle
        Private redPen As Pen
        '-----
        Public Sub New(ByVal x As Integer, _
                ByVal y As Integer, ByVal w As Integer, _
                ByVal h As Integer)
         MyBase.New(x, y, w, h)
         redPen = New Pen(Color.FromARGB(255, _
                  Color.Red), 2)
        End Sub
```

这意味着 DoubleRect 类必须有自己的 draw 方法。现在基类有 draw 方法，因为希望所有的类作用相同，所以我们产生一个与基类方法名字相同的方法。然而，draw 方法还是使

用父类的 draw 方法，但增加了许多它自己的参数。换句话说，draw 方法将被重载，所以我们必须特殊声明，以便满足 VB 编译器。

```
Public Overrides Sub draw(ByVal g As Graphics)
    MyBase.draw(g)
    g.drawRectangle(redPen, x + 4, y + 4, w, h)
End Sub
```

如果我们想使用在构造函数中定义的坐标和长方形的尺寸，就应该在 DoubleRect 类中保留这些参数的拷贝，或者在基类 Rectangle 中把变量的 private 型改成 protected 型。

```
Protected x, y, h, w As Integer
```

我们也必须告诉编译器，我们允许 Rectangle 的 draw 方法，通过声明它是可重载的以使其可重载。

```
Public Overridable Sub draw(ByVal g As Graphics)
     g.DrawRectangle(rpen, x, y, w, h)
End Sub
```

最后的画长方形的屏幕如图 7-9 所示。

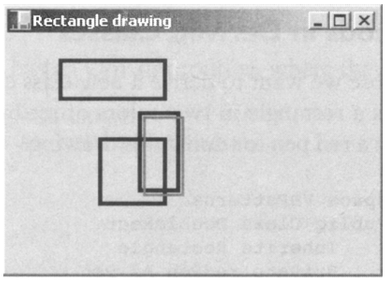


图 7-9 Rectangle 中 Sqare 和 DoubleRect 类

7.19 重载与隐蔽

VB7 除了提供重载之外，还提供了 Shadows 关键字。一个方法或属性重载了基类的成员后，重载方法和属性替换原有的方法和属性。如果你有一个方法的多个重载版本，则只有被重载的方法可被替换。

再考虑下面这个简单的 BaseClass 类。

```
Public Class BaseClass
    Private data As Single
    '.....
    Public Sub setData(ByVal x As Single)
        data = x
    End Sub
```

```
    '------
    Public Sub setData()
        data = 12.2
    End Sub
End Class
```

在这个类中，有两个 setData **方法的重载版本，一个需要参数，而另一个使用内部默认值。你也可以使用** Overloads **关键字写相同的类。**

```
Public Class BaseClass
    Private data As Single
    Private Const defaultv As Single = 12.2
    '------
    Public Overloads Sub setData(ByVal x As Single)
        data = x
    End Sub
    '------
    Public Overloads Sub setData()
        data = defaultv
    End Sub
End Class
```

它们有相同的含义，并且编译产生相同的代码。

现在让我们考虑一个从拥有新版 SetData **方法类导出的类。**

```
Public Class DerivedClass
    Inherits BaseClass
    '------
    Public Overloads Sub setData(ByVal x As Single)
        MyBase.setData(x + 12)
    End Sub
End Class
```

这个类有两个 setData **方法：一个被声明成上面的形式，另一个从** BaseClass **类继承。所以我们可以写成：**

```
Dim cl As New DerivedClass()
        cl.setData()
        cl.setData(14)
```

另一方面，如果我们可以使用 Shadows **关键字，那么在基类中命令的所有其他被重载的方法在子类中就不可见。**

```
Public Class DerivedClass
    Inherits BaseClass
    '------
    Public Shadows Sub setData(ByVal x As Single)
        MyBase.setData(x + 12)
    End Sub
End Class
```

仅在 DerivedClass **类中的** setData **方法版本是可见的，并且必须有参数说明。**

```
Dim cl As New DerivedClass()
        cl.setData()   'now is illegal
        cl.setData(14) 'is legal
```

7.20 重载窗口控件

在 VB7 中，可以使用继承在已有的控件基础上创建新的 Windows 控件。以前我们用产生 TextBox（文本框）控件来激活其中的文本。在 VB6 中，可以编写一个新的 DLL 程序把一个文本框嵌入到用户控件中，并把所有的有用事件传递到内嵌的文本框中。在 VB7 中，可以用继承的方法在 TextBox 类中导出新的控件。

下面我们还是从 Windows Designer 开始，首先产生一个有两个文本框的窗体。然后打开 Project|Add 用户控件菜单，加入一个称为 HiTextBox.vb 的对象。我们将使用继承从 TextBox 类中得到这个对象，而不是使用 UserControl。

```
public class HTextBox
        Inherits Textbox
```

然后，在做任何改变之前编译该程序，新的 HiTextBox 控件出现在 Toolbox 控件的下面，如图 7-10 所示。你也可以在任何窗体中产生可视化的 HiTextBox 实例。

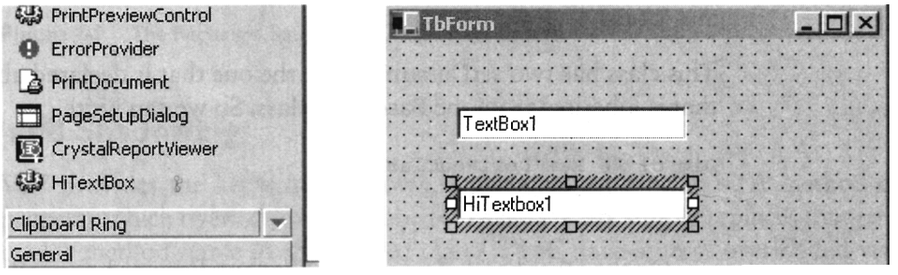


图 7-10 在一个新的窗体中，工具框 Toolbox 显示我们产生的新控件，如一个 HiTextBox 实例

现在我们可以修改这个类，并插入使文本被激活的代码。

```
'A text box that highlights when you tab into it

 Public Class HiTextBox
    Inherits System.windows.forms.TextBox
    Private Sub init()
        AddHandler Enter, _
            New System.EventHandler(AddressOf Me.HT_Enter)
    End Sub

    '-------------
    'Enter event handler is inside the class
```

```
    Protected Sub HT_Enter(ByVal sender As System.Object, _
            ByVal e As System.EventArgs)
        Me.SelectionStart = 0
        Me.SelectionLength = Me.Text.Length
    End Sub

End Class
```

上面的程序就是导出一个新的 Windows 控件的整个过程，可以在图 7-11 中看到此程序的执行结果。如果运行这个程序，你可能发现这两个文本框相同，原因是当用 Tab 键在它们之间切换时，它们都被激活，这是 VB7 中文本框自动激活的特性。然而，当你点击这两个文本框内的文本，并按 Tab 键在它们之间切换时，你就会看到在图 7-11 中只有新创建的 HiTextBox 文本框继续被激活。

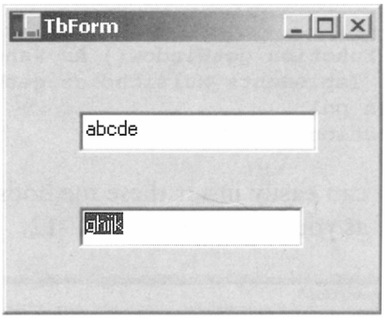


图 7-11　一个新导出的 HiTextBox 控件和通常的 TextBox 控件

7.21　接口

就像 VB6 一样，VB7 继续支持接口概念。然而，语法有一些变化。VB7 中的接口是一个特殊的类。

```
Public Interface MultiChoice
    'an interface to any group of components
    'that can return zero or more selected items
    'the names are returned in an Arraylist
    Function getSelected() As ArrayList
    Sub clear()              'clear all selected
    Function getWindow() As Panel
End Interface
```

当你在固定的类中实现接口的方法时，你必须声明此类实现指定的接口。

```
Public Class ListChoice
  Implements MultiChoice
```

并且你必须声明实现接口类中的每一个方法。

```
Public Function getSelected() As ArrayList _
         Implements MultiChoice.getSelected

 End Function
'--------
'clear all selected items
Public Sub clear()  implements MultiChoice.clear
End Sub
'--------
Public Function getWindow() As Panel _
        Implements MultiChoice.getWindow
  return pnl
End Function
```

你事先从 Visual Studio Code Designer **中提取这些方法，如图** 7-12 **所示。**

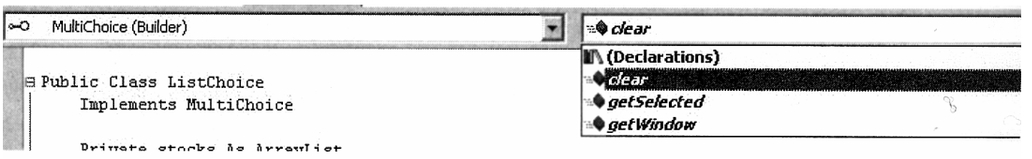


图 7-12 **使用** IDE **创建方法来实现** MultiChoice **接口**

注意，所有的接口必须被声明为 Public **类型，否则你将得到错误编译。我们将在** Builder **模式中继续讨论接口应用技术。**

7.22 小结

在这一章中已经体会到了 VB7 **的新特性、语法上的变化、继承、构造函数以及提供不同版本的重载方法。这些新的特性增强了** VB7 **在生成新控件时的能力。在下面的各章中将介绍如何用** VB6 **和** VB7 **编写设计模式。**

7.23 光盘中的程序

\IntroVBNet\Hellol	**使用** Module **方法的控制台“** Hello **”应用程序**
\IntroVBNet\Hello2	**使用** Class **方法的控制台“** Hello **”应用程序**
\IntroVBNet\HiText	**一个子类化的文本框**
\IntroVBNet\SayHello	**一个** Simple Hello **程序**
\IntroVBNet\Tokenizer	**一个字符串的分离器**

第 8 章　VB.NET 中的数组、文件和异常

在 VB7 中对数组、文件和错误处理有了很大的改进，这些改进使得编程更加容易。

8.1　数组

在 VB7 中，数组下标是从 0 开始计数的，这一点与其他 VB 版本相同。例如在 VB6 中，如果你定义了下面这个 x 数组

```
Dim x(10) As Single
```

然后假定 x 数组的元素从 1~10，但实际上 x 数组下标仍然从 0 开始计数。也就是说，x 数组元素个数是 11，而不是 10 个。在 VB7 中，这一规定仍然有效。

然而，在 VB7 中，数组的使用与其他高级语言，尤其与 C，C++，C#和 Java 等程序设计语言的风格更加接近。

```
Dim Max as Integer
Max = 10
Dim x(Max)

For j = 0 to Max-1
 x(j) = j
Next j
```

你也可以定义下面的语句

```
For j = 1 to Max
```

数组变量有长度属性，可以通过访问该属性获得数组的长度。

```
Dim z(20) As Single
Dim j As Integer

For j = 0  To z.Length – 1
       z(j) = j
Next j
```

注意，循环语句

```
For j = 0  To z.Length -- 1
```

实际上遍历了 21 个元素，如果你声明

```
Dim z(20) As Single
```

它包含 21 个元素，数组下标从 0~20。所以 z.Length 的值是 21，从 0 到 z.Length-1 遍历了 21 个元素。如果你定义循环语句

```
For j = 0  To z.Length
```

将出现越界错误，一个安全的方法是定义下面的循环语句

```
For j = 0 to Ubound(z) - 1
```

在 VB7 中，数组是动态的。也就是说，你可以在任意时候重新分配数组空间。为了在类中为一个数组重新分配空间，可以采用下面的语法定义数组下标。

```
'Declare at the class level
    Dim x() As Single
'allocate within any method
    x = New Single(20) {}
```

注意在语句后面两个大括号的使用，这意味着你可以产生一个新的数组对象。采用这种语法的优点是，你可以在大括号中声明数组元素。

```
Dim a() As Single
a = New Single() {1, 3, 4, 5}
```

你也可以使用带有或不带有 Preserve 的 ReDim 语句改变已经声明的数组的长度。请注意，ReDim 语句不再有 As 子句。这是因为 ReDim 语句不能改变数组元素的类型。

```
ReDim x(40)
ReDim Preserve x(50)
```

8.2 集合对象

在 System.Collections 名字空间中包含有一些变长数组对象，你可以使用这些对象操作数组元素。

8.2.1 数组列表

因为数组从 0 开始计数，所以在 VB7 中引进了 ArrayList 对象来代替 Collection 对象，ArrayList 对象总是从 1 开始计数。ArrayList 是一个基本的变长数组，你可以在需要时增加新元素。ArrayList 中的基本方法与 Collection 相同，并增加了一些新方法。

```
Dim i, j As Integer
 'create ArrayList
Dim arl As New ArrayList() 'constructor
 'add to it
```

```
For j = 0 To 9
    Arl.Add(j)
Next j
```

与 Collection 对象相同，ArrayList 有 Count 属性和 Item 属性，这些属性允许你根据下标访问数组元素。像 Collection 一样，这两个属性可以省略，这时再对 ArrayList 的处理，就像数组一样。

```
'print out contents
For i = 0 To arl.Count - 1
    Console.writeLine(arl.Item(i))
    Console.writeLine(arl(i))
Next i
```

你还可以使用其他 ArrayList 方法，这些方法如表 8-1 所示。

表 8-1　ArrayList 方法

方法	说明
Clear	清除数组内容
Contains(object)	如果数组有值，则返回 true
CopyTo(array)	拷贝 ArrayList 到一维数组
IndexOf(object)	返回值的第一个下标
Insert(index,object)	在指定的下标位置插入一个值
Remove(object)	从表中删除元素
RemoveAt(index)	从指定位置删除元素
Sort	ArrayList 排序

8.2.2　Hashtable

Hashtable 是一个可变长的数组，它的每一项对应一个关键字。典型的关键字是一个字符串，但也可以是一个对象。关键字对每一个元素是惟一的，即使元素本身不需要是惟一的。Hashtable 通常被用于快速访问一个大的、未排序的记录集。它可以通过改变关键字的值和入口值来产生一个确保入口值是惟一的列表。Hashtable 最重要的方法是 Add 和 Item。

```
Dim hash As New Hashtable()
  Dim fredObject As New Object()
  Dim obj As Object
  hash.Add("Fred", fredObject)
  obj = hash.Item("Fred")
```

Hashtable 也具有计数属性，你可以获得一个关键字或一个值的枚举。

8.2.3 SortedList

SortedList 类与 VB6 中的 Collection 类相似，它包含两个内部数组，所以你可以通过以 0 开始计数的下标或以字符关键字下标读取一个元素。

```
Dim sList As New SortedList()
  slist.Add("Fred", fredObject)
  slist.Add("Sam", obj)
  Dim newObj As Object
  newObj = slist.GetByIndex(0)     'by index
  newObj = slist.Item("Sam")       'by key
```

你还可以在 SortedList 名字空间中发现栈和 Queue（队列）对象。这些对象具有通常意义下的特性，你可以在系统帮助文档中找到它们的方法。

8.3 异常

在 VB7 中，错误处理通过异常（Exception）而不是采用蹩脚的 On Error Goto 语法。异常处理是通过把产生错误的语句放入一个用 Try 标示的语句体内，并用一个 Catch 语句捕获错误。

```
Try
  'Statements
 Catch e as Exception
   'do these if an error occurs
 Finally
   'do these anyway
End Try
```

通常使用这种方法处理文件错误、数组下标越界等其他出现错误的情况。该方法的工作原理是，让需要判断的语句在 Try 语句块内执行。如果没有错误，则执行 Finally 语句，并继续执行 End Try 以后的语句；如果有错误，则执行 Catch 语句，在该语句体内处理错误，然后执行 Finally 语句及以后的语句。

下面是一个异常处理的示例。该示例中数组列表 ArrayList 有下标越界发生。

```
Try
    For i = 0 To ar.Count                'NOTE: one too many
        Console.write(ar.Item(i))
    Next i
Catch e As Exception
    Console.writeLine(e.Message)
    Console.writeLine(e.stackTrace)
End Try

Console.writeline("end of loop")
```

上面的代码输出错误信息和调用程序入口，然后继续执行。

```
0123456789Index is out of range.  Must be non-negative and less
than size.
Parameter name: index
   at System.Collections.ArrayList.get_Item(Int32)
   at ArrayTest.Main()
end of loop
```

对比一下，如果没有捕获异常，我们将从系统环境中得到一个错误信息，而且程序将退出，而不是继续执行。

```
Exception occurred: System.ArgumentOutOfRangeException: Index
is out of range.
Must be non-negative and less than size.
Parameter name: index
   at System.Collections.ArrayList.get_Item(Int32)
   at ArrayTest.Main()
   at _vbProject._main(System.String[])
```

表 8-2 列出了一些普通的异常。

表 8-2　VB7 的异常

异常	异常说明
AccessException	在访问一个类中的方法和属性时出错的异常
ArgumentException	对一个方法的声明无效
ArgumentNullException	声明为空
ArithmeticException	上溢或下溢
DivideByZeroException	除数为 0
IndexOutOfRangeException	数组下标越界
FileNotFoundException	文件未找到
EndOfStreamException	输入流（文件）越界访问
DirectoryNotFoundException	目录没找到
NullReferenceException	对象变量没有被初始化

8.4　多重异常

你可以在一个 Try 语句体内捕获和处理多个异常。

```
Try
    For i = 0 To ar.Count
       Dim k As Integer = CType(ar(i), Integer)
       Console.writeLine(i.toString & " " & k / i)
```

```
     Next i
Catch e As DivideByZeroException
   Console.writeLine(e.Message)
   Console.writeLine(e.stackTrace)
Catch e As IndexOutOfRangeException
   Console.writeLine(e.Message)
   Console.writeLine(e.stackTrace)
Catch e As Exception
   Console.writeLine("general exception" + e.Message)
   Console.writeLine(e.stackTrace)
End Try
```

这种方式给你提供了从各种错误恢复的多种途径。

8.5 抛出异常

你不一定要在错误发生的地方处理异常，你可以使用 Throw 语句把错误返回到调用程序中去处理。该语句是把异常返回到调用程序。

```
Try
  'some code
Catch e as Exception
   Throw e   'pass on to calling routine
End Try
```

8.6 文件处理

可以使用 VB6 中的文件处理函数进行文件处理，由于所有的声明必需放到括号内，所以这些函数的语法比较复杂。这样你处理文件过程中必须做相应的转换。

```
Input #f, s   'read a string from a file in VB6
```

转换成

```
Input(f, s)   'vb6 compatible string read from file
```

在 VB7 中，已经没有文件的 Line Input（线性输入）语句。取而代之的是，采用更容易读、写数据的文件和流方法。

8.6.1 File 对象

File 类提供了一些有用的文件处理方法，这方法包括判断文件是否存在、文件改名和删除文件等等。File 对象仅包含共享方法，这样就可以在不产生实例的情况下，使用 File 类中的方法。共享方法是 VB 中的叫法，在其他语言中也叫静态方法（static）。之所以这样

称呼是因为这些方法的使用不用定义对象只用定义表。表 8-3 列出了一些常见的文件操作方法，更全面的说明参考帮助文档。

表 8-3　文件方法

共享（静态）方法	解释
File.Exists(filename)	如果文件存在，返回 true
File.Delete(fileename)	删除文件
File.AppendText(String)	在文件中追加文本
File.getAttributes(String)	返回 FileAttribute（文件属性）对象
File.Copy(fromFile, toFile)	拷贝文件
File.Move(fromFile,toFile)	用一个新文件覆盖一个老文件
File.OpenText(filename)	为读打开一个文本文件
File.OpenWrite(filename)	为读打开任意一个文件

例如，如果想用 File 类删除一个文件，首先要判断文件是否存在，然后用如下语句删除该文件：

```
If File.Exists("foo.txt") Then  'test existence
    File.Delete("foo.txt")      'delete it
End If
```

你也可以使用 File 对象获得 StreamReader 或 FileStream 实现文件的读写操作。

```
Dim ts As StreamReader
Dim fs As FileStream

ts = File.OpenText("foo.txt")  'open a text file for reading
fs = File.OpenRead("foo.data") 'open any file for reading
```

8.6.2　读一个文本文件

为了读文本文件，需要使用 File 对象获得一个 StreamReader 对象，然后使用文本流的读方法。

```
Dim ts As StreamReader
ts = File.OpenText("foo.txt")  'open a text file

Dim s As String
s = ts.ReadLine()
```

8.6.3　写一个文本文件

为了建立一个文本文件并写入数据，使用 CreateText 方法获得一个 StreamWriter 对象。

```
Dim sw As StreamWriter
sw = File.CreateText("foo.txt")
sw.WriteLine("Hello there")
```

如果想在一个已存在的文件中追加数据，可以创建一个 SreamWriter 对象，并在参数中使用 true。

```
'appending
 sw = New StreamWriter("foo.txt", True)
```

8.7 在文件处理中使用异常

许多异常都发生在文件的输入/输出环境。你可以进行非法文件名、文件不存在、目录不存在、非法的文件名声明和文件保护错误等异常处理。所以最好的方法是在处理文件输入/输出时把相关代码放到 Try 语句体内，从而确保文件处理不发生致命错误。在帮助文档中列出了所有的 file 类的方法。如果你捕获一个一般的异常对象。你可以确信捕获到所有的异常，但在不同异常的处理上，还需要分别进行。

例如，可以用下面的语句打开文件：

```
Try
      ts = File.OpenText("foo.txt")
Catch e As FileNotFoundException
     'print out any error
      Console.WriteLine(e.Message)
      errFlag = True
End Try
```

8.8 测试文件结束

有两个方法可以使你确信当前的操作没有超过文本文件结束符：一个是发现 null 异常，另一个是发现数据流的结束符。当你越界读文件时，没有错误发生，也没有文件结束异常被抛出。然而，如果在文件结束后读一个字符串，它将返回一个空值 null。在 VB7 中，没有提供 IsNull 方法，但你可以通过计算字符串长度的方法，强制产生一个 Null Reference 异常。如果你对一个空字符串求长度，系统就会抛出一个 Null Reference 异常，这样你就可以使用这个方法检测文件是否结束。

```
Public Function readLine() As String
'Read one line from the file
 Dim s As String
 Try
    s = ts.readLine         'read line from file
    lineLength = s.length  'use to catch null exception
 Catch e As Exception
  end_file = True           'set EOF flag if null
```

```
  s = ""                    'and return zero length string
 Finally
  readLine = s
 End Try
End Function
```

另一个确信没有越界读文件的方法是使用流的 Peek 方法。该方法返回下一个字符的 ASCII 码值，如果没有剩下字符，则返回-1。

```
'example of alternate approach to detecting end of file
 Public Function readLineE() As String
    'Read one line from the file
    Dim s As String
    If ts.peek >= 0 Then       'look ahead
         s = ts.readLine       'read if more chars
         Return s
    Else
         end_file = True       'Set EOF flag if none left
         Return ""
    End If
 End Function
```

8.9　FileInfo 类

FileInfo 有许多用于操作一个特殊文件名的非共享方法，如表 8-4 所示。你可以使用这些方法得到特殊文件的信息。

表 8-4　FileInfo 中的文件操作方法

方法	解释
Attributes	获得 FileAttribute 对象
DirectoryName	获得文件路径
Extension	获得文件的扩展名
Length	文件长度
Name	获得文件名
Exists	如果文件存在，则返回 true
Delete()	删除文件
OpenText()	为读文本文件产生一个 StreamReader
AppendText()	为追加文本文件产生一个 StreamWriter
OpenWrite()	为输出文本文件产生一个 StreamWriter

8.10 vbFile 类

在以前的 VB 版本中，我们为读写一个文本文件而建立一个 vbFile 类。我们可以用 VB7 重新实现 vbFile 类，它们有完全相同的方法，只是在 VB7 中使用了 VB7 的文件处理类。实际上我们正是在用 VB7 实现 vbFile 接口。由于语法是相同的，所以我们使用相同的接口，但在语法上还是有一点不同。我们将使用相同的接口编写一个新的类。

主要不同是我们可以在构造函数内包含文件名和路径。在这个示例中，使用静态 File 类来获得 StreamReader 和 StreamWriter 对象。你也可以类似地使用一个 FileInfo 类的实例，该示例在本书的配套光盘中。

```
Public Sub New(ByVal filename As String)
    'Create new file instance
     File_name = filename      'save file name
     tokLine = ""              'initialize tokenizer
     sep = ","                 'and separator
End Sub
```

我们可以用下面两个方法中的任意一个方法打开一个文件：一个方法是包含文件名，而另一个是在参数声明中使用文件名。

```
Public Overloads Function OpenForRead() As Boolean
          Return OpenForRead(file_name)
End Function
'-----------------
Public Overloads Function OpenForRead(_
ByVal Filename As String) As Boolean
    'opens specified file
    file_name = Filename     'save file name
    errFlag = False          'clear errors
    end_File = False         'and end of file
    Try
        ts = fl.Opentext()     'open the file
    Catch e As Exception
        errDesc = e.Message    'save error message
        errFlag = True         'and flag
    End Try
    Return Not errFlag   'false if error
End Function
```

这样你可以从文本文件中读数据。

同样，下面的方法允许你为写打开文件，并写一行到文件中。

```
Public Overloads Function OpenForWrite( _
            ByVal fname As String) As Boolean
     errFlag = False
     Try
         File_name = fname
         sw = File.CreateText(File_name) 'get StreamWriter
```

```
        Catch e As Exception
           errDesc = e.Message
           errFlag = True
        End Try
        OpenForWrite = Not errFlag
    End Function
    '--------------
    Public Overloads Function OpenForWrite() As Boolean
        OpenForWrite = OpenForWrite(file_name)
    End Function
    '--------------
    Public Sub writeText(ByVal s As String)
        sw.writeLine(s)              'write text to stream
    End Sub
```

由于我们已经在 vbFile **类中实现了与** VB6 **相同的方法，所以我们可以在** VB7 **程序中使用这些新的方法，而根本不用改变其他相关程序。**

8.11　光盘中的程序

\FilesArraysExceptions\Array	**使用数组和** ArrayList **的示例**
\FilesArraysExceptions\Exceptions	**解释如何使用异常**
\FilesArraysExceptions\FileObject	**在** VB7 **中使用** File **对象实现** vbFile **类**
\FilesArraysExceptions\FileStream	**在** VB7 **中使用** FileInfo **对象实现** vbFile **类**

第 II 部分

生 成 模 式

在前面系统学习了 VB7 中的对象、继承和接口等概念之后，现在开始讨论设计模式。所谓的设计模式就是写好面向对象程序的方法，这一点我们已经十分明确。在讨论中把设计模式分成 3 个部分：生成模式、结构模式和行为模式。我们首先讨论生成模式。

生成模式讨论生成对象实例的方法。这一点尤其重要，因为一个程序不能依赖于对象是如何被创建和排列的。在 VB6 中，创建一个对象实例的最简单方法是使用 new 操作符。

```
set fred1 = new Fred                'instance of Fred class
```

但这种代码依赖于如何产生一个对象。在许多情况下，需要根据实际情况的变化产生一个对象，把对象生成过程抽象成一个“creator（生成器）”类，这样会使你的程序更灵活和通用。

工厂方法（Factory Method）模式提供了一个简单的决策类，它根据提供的数据返回抽象基类的几个可能的子类。我们先使用简单工厂（Simple Factory）模式介绍工厂的背景，然后介绍工厂方法模式。

抽象工厂（Abstract Factory）模式提供了一个产生并返回相关对象集合的接口。

构造器（Builder）模式把一个复杂对象的构造函数从它的实现分开，这样可以根据程序的需要产生几个不同的实现。

原型（Prototype）模式从一个实例化类开始，然后复制或克隆成一个新实例。这些实例的公共方法可以进一步被使用。

单一类（Singleton）模式是一个没有更多实例的类。它提供了一个实例的单一全局访问入口。

第 9 章　简单工厂模式

在面向对象程序中我们多次看到的一种模式就是简单工厂模式。一个简单工厂模式返回一个实例，这个实例是根据实际数据，在几个可能的类的实例中确定的。通常它返回的所有类中有一个普通的父类和普通的方法，但每一个执行不同的任务，并被不同类型的数据优化。实际上，这个简单工厂与 GoF 第 23 个模式不同，这里的简单工厂模式仅作为将要讨论的 Factory Method GoF 模式的简介。

9.1　一个简单工厂如何工作

为了理解简单工厂模式，让我们分析一下图 9-1。

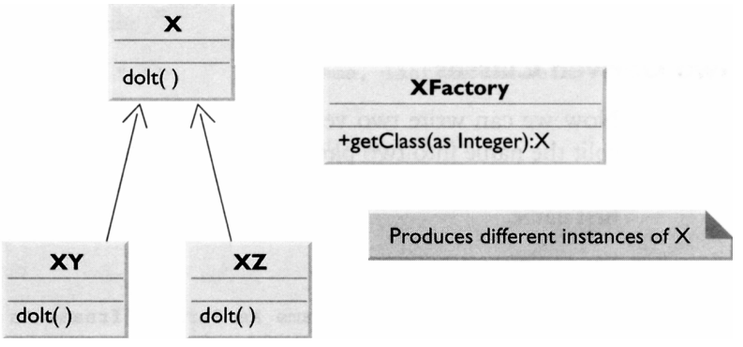


图 9-1　一个简单工厂模式

在图 9-1 中，X 是一个基础类，XY 和 XZ 是它的导出类。由 Xfactory 类根据你给定的参数确定返回哪一个导出类。在右侧的图中我们定义了一个 getClass 方法，并根据输入参数值返回 x 类的一些实例。返回的实例不受程序员的控制，这是因为虽然它们有相同的方法，但它们的实现是不同的。它决定哪一个任务上传给 Xfactory 类。该程序看似很复杂，但实际上十分简单。

9.2　代码片段

让我们再用 VB 来解释 Factory 类。假设我们已经建立了一个实体数据表，并允许用户

使用不同的次序输入名字，如“名姓”或“姓，名”。为了简化处理，进一步假定它们的次序可以用是否在名和姓之间加“，”来决定。

这是一个十分简单的判断，只要你在类中使用一个简单的 if 语句就可以实现，但我们的目的是想通过这个示例来解释类工厂是如何工作的，它的输出是什么。我们还是从定义一个简单的接口开始，该接口的输入参数是一个名字字符串，你也可以获得名字。

```
Public Sub init(ByVal s As String)
End Sub
'-----
Public Function getFrName() As String
End Function
'-----
Public Function getLname() As String
End Function
```

9.3 两个导出类

现在我们写两个十分简单的类来实现上面定义的接口，并在构造函数中实现名字的分解。在 FNamer 类中，我们假设最后一个空格前都是人的名。

```
'Class FNamer
Implements Namer
Private nm As String, lname As String, frname As String
'-----
Private Function Namer_getFrname() As String
Namer_getFrname = frname
End Function
'-----
Private Function Namer_getLname() As String
Namer_getLname = lname
End Function
'-----
Private Sub Namer_init(ByVal s As String)
Dim i As Integer
nm = s
i = InStr(nm, " ")               'look for space
If i > 0 Then
  frname = Left$(nm, i - 1)      'separate names
  lname = Trim$(Right$(nm, Len(nm) - i))
Else
  lname = nm             'or put all in last name
  frname = ""
End If
End Sub
```

在 LNamer 类中，我们假定逗号是姓的结束符。在这两个示例中，我们还提供了对空格和逗号不存在时的错误处理。

```
'Class LNamer
Implements Namer
Private nm As String, lname As String, frname As String
'-----
Private Function Namer_getFrname() As String
Namer_getFrname = frname
End Function
'-----
Private Function Namer_getLname() As String
Namer_getLname = lname
End Function
'-----
Private Sub Namer_init(ByVal s As String)
Dim i As Integer
nm = s                    'save whole name
i = InStr(nm, ",")        'if comma, last is to left
If i > 0 Then
  lname = Left$(nm, i - 1)
  frname = Trim$(Right$(nm, Len(nm) - i))
Else
  lname = nm              'or put all in last name
  frname = ""
End If
End Sub
```

9.4　构建简单工厂

现在我们的 Simple Factory **十分容易实现，并且它还是用户的接口。通过测试逗号是否存在，返回一个类的实例。**

```
Private nmer As Namer   'will be one kind or the other
'-----
Private Sub getName_Click()
Dim st As String, i As Integer
st = txNames.Text         'get the name from the entry field
i = InStr(st, ",")        'look for a comma
If i > 0 Then
  Set nmer = New lNamer   'create last name class
Else
  Set nmer = New Frnamer 'or fist name class
End If
nmer.init st
'put results in display fields
txFrName.Text = nmer.getFrName
txlName.Text = nmer.getLname
End Sub
```

9.4.1　使用工厂

现在我们把上面的程序段组合成一个完整的程序，图 9-2 **是一个完整的类说明图。**

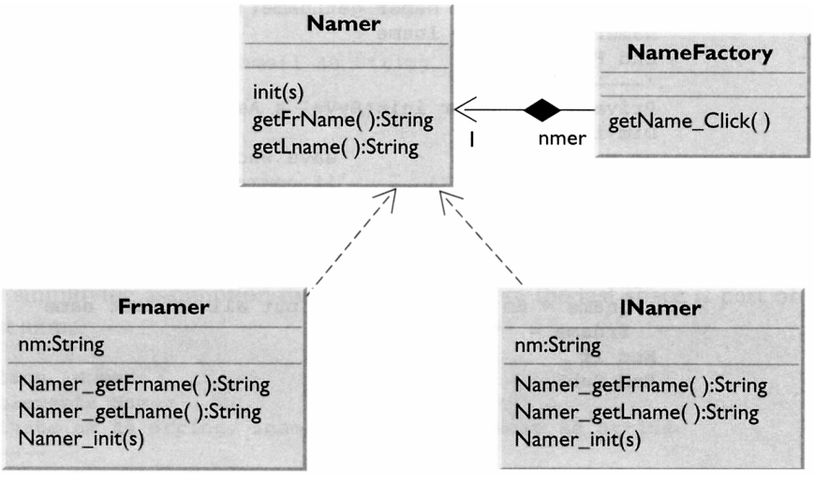


图 9-2 Namer 工厂程序

我们设计了一个简单的窗体，该窗体提供了一个输入框，你可以按任何次序在此框内输入名字。在该窗体上还可以显示分解后的名字。窗体如图 9-3 所示。

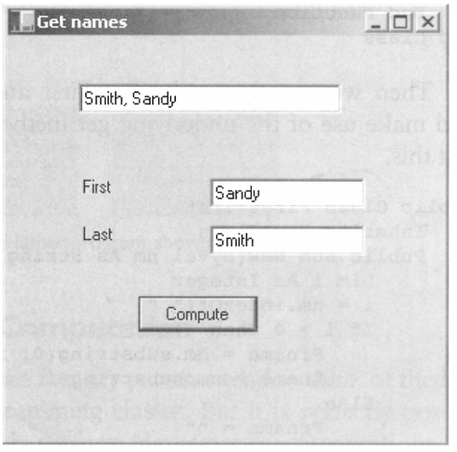


图 9-3 Namer 程序执行窗体

当你在输入框内输入一个名字后，点击 Compute 按钮，这个被分离的名字就分别显示在下面的 First 和 Last 文本框内。该程序的核心是计算方法，它接收输入文本、产生一个 Namer 类的实例和显示结果。

简单工厂模式的基本原则是：产生一个用于确定返回类的抽象类，并实现一个返回。然后调用类实例中的方法，哪个子类被调用无需知晓。这一原则保证了对象方法与数据的依赖性。

9.5　用 VB.NET 编写工厂模式

在 VB7 中，可以广泛使用继承来实现工厂。我们首先定义一个称为 NameClass 的基类，它用两个 protected 变量存放名和姓，并定义了两个访问名和姓的访问函数。

```
Public Class NameClass
    Protected Lname, Frname As String
    Public Function getFirst() As String
        Return Frname
    End Function

    Public Function getLast() As String
        Return Lname
    End Function
End Class
```

然后我们在 NameClass 类的基础上导出 FirstFist 和 LastFirst 类，并继承父类的方法。完整的 FirstFirst 类程序如下：

```
Public Class FirstFirst
    Inherits NameClass
    Public Sub New(ByVal nm As String)
        Dim i As Integer
        i = nm.indexOf(" ")
        If i > 0 Then
            Frname = nm.substring(0, i).trim()
            Lname = nm.substring(i + 1).trim()
        Else
            Frname = ""
            LName = nm
        End If
    End Sub
End Class
```

LastFirst 类与 FirstFirst 类相似。Factory 类也基本相同，只是 Factory 类中使用了构造函数。

```
Public Class NameFactory

    Public Function getNamer( _
        ByVal nm As String) As NameClass
        Dim i As Integer

        i = nm.indexOf(",")
        If i > 0 Then
            Return New LastFirst(nm)
        Else
            Return New FirstFirst(nm)
        End If
    End Function
End Class
```

在图 9-4 中可以看到它们的不同。

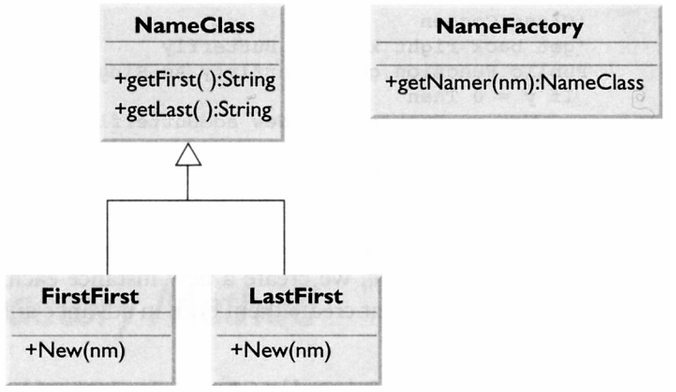


图 9-4 用 VB7 实现的带有继承性的 Namer 程序

9.6 使用数学计算的工厂模式

也许很多人希望使用工厂模式作为一个简单的三角运算的程序类。不仅如此，工厂模式完全可以用于简化数学计算。例如，在快速富里埃变换（Fast Fourier Transform，FFT）中，你需要重复给下面的 4 个方程赋值。这样有许多点对通过数组存放并导入。由于这些计算结果的图形被画出，所以下面的 4 个方程构成 FFT“蝴蝶”的一个实例，如方程（1）~（4）所示。

$$R_1 = R_1 + R_2\cos(y) - I_2\sin(y) \quad (1)$$
$$R_2 = R_1 - R_2\cos(y) + I_2\sin(y) \quad (2)$$
$$I_1 = I_1 + R_2\sin(y) - I_2\cos(y) \quad (3)$$
$$I_2 = I_1 - R_2\sin(y) - I_2\cos(y) \quad (4)$$

然而，在许多情况下，上面的 y 值为 0，所以可以有下面的方程组成立。

$$R_1 = R_1 + R_2 \quad (5)$$
$$R_2 = R_1 - R_2 \quad (6)$$
$$I_1 = I_1 + I_2 \quad (7)$$
$$I_2 = I_1 - I_2 \quad (8)$$

下面我们用一个 Factory 类来确定返回哪个类实例。因为方程（1）～（4）称为蝴蝶方程，所以我们称 Factory 为 Cocoon。

```
'Class Cocoon
'get back right kind of Butterfly
Public Function getButterfly(y As Single) As Butterfly
 If y = 0 Then
    Set getButterfly = New addButterfly
 Else
    Set getButterfly = New trigButterfly
 End If
End Function
```

在这个示例中每一次产生一个新实例。由于只有两种类型，所以先产生两个实例，然后再根据需要使用。

```
'Class Cocoon1
Private addB As Butterfly, trigB As Butterfly
'-----
'create instances in advance
Private Sub Class_Initialize()
 Set addB = New addButterfly
 Set trigB = New trigButterfly
End Sub
'-----
'get back right kind of Butterfly
Public Function getButterfly(y As Single) As Butterfly
 If y = 0 Then
    Set getButterfly = addB
 Else
    Set getButterfly = trigB
 End If
End Function
```

思考题：

1. **分析一下类似于** Quicken **的个人帐户管理程序。它可以管理几个银行的帐户和投资，并为你支付帐单。试设计一个工厂模式加以实现？**
2. **假设你正在为一个房主设计一个实现房屋装修设计的程序，那么使用工厂可以产生哪些对象呢？**

9.7　光盘中的程序

\Factory\Namer	VB6 **实现的** Namer **工厂**
\Factory\Namer\vbNetNamer	VB7 **实现的** Namer **工厂**
\Factory\FFT	VB6 **实现的** FFT **示例**

第 10 章　工厂方法模式

我们已经接触了两个最简单的工厂的示例。工厂再现了面向对象思想，这种思想在其他设计模式（如建设者模式）和 VB 程序中也得以体现。在这些示例中，一个类扮演一个交通警，由他来决定哪个继承的子类被初始化。

工厂方法模式对上述思想有了一点改进。其中不再存在一个类来决定哪个子类被初始化，而是由子类自己决定。实际上该模式已经不再设计一种机制来决定被选择的子类，而是定义一个用于产生对象的抽象类，但还是由子类决定哪个对象被产生。

现在给出一个简单的示例说明工厂方法模式。在游泳比赛中，通常需要经过一些预赛来选择一些种子选手进入决赛，参赛的运动员按预赛的成绩排序。比赛赛道的分配也是按照预赛的成绩分配的，成绩好的运动员被分配到中心赛道。这种方法称为直接筛选法（Straight Seeding）。

现在正式开始比赛，过程分两个阶段，预赛和决赛。每一个运动员都参加预赛，并在预赛中选择 12 个或 16 个运动员参加决赛。为了使预赛更加公平，成绩较好的运动员被交叉分配到各组。做法是，成绩最好的 3 个运动员被分配到中心赛道，次之的 3 个运动员被分配到余下赛道的中心赛道，依此类推。

如何用工厂方法实现上述种子选手的选择过程呢？首先让我们设计一个抽象的 Events 类。

```
Option Explicit
'Class Events
Private numLanes As Integer
Private swimmers As New Collection  'list of swimmers
'-----
Public Sub init(Filename As String, lanes As Integer)
End Sub
'-----
Public Function getSwimmers() As Collection
 Set getSwimmers = swimmers
End Function
'-----
Public Function isPrelim() As Boolean
End Function
'-----
Public Function isFinal() As Boolean
End Function
'-----
Public Function isTimedFinal() As Boolean
End Function
```

```
'-----
Public Function getSeeding() As Seeding
End Function
```

从语法上讲，该类最好命名为 Event 类，但 Event 是 VB6 的保留字。

现在简单定义其他的方法，暂时不必实现这些方法的细节。接下来在 Events 类基础上生成一些具体的类，这些类称为 PrelimEvent 和 TimedFinalEvent。它们的不同之处是返回不同的种子选手。

再用下面的方法定义一个抽象的 Seeding 类。

```
'Class Seeding
Private numLanes As Integer
Private laneOrder As Collection
Dim asw() As Swimmer
'-----
Public Function getSeeding() As Collection
End Function
'-----
Public Function getHeat() As Integer
End Function
'-----
Public Function getCount() As Integer
End Function
'-----
Public Sub seed()
End Sub
'-----
Public Function getSwimmers() As Collection
End Function
'-----
Public Function getHeats() As Integer
End Function
'-----
Private Function odd(n As Integer) As Boolean
  odd = (n \ 2) * 2 <> n
End Function
'-----
Public Function calcLaneOrder(lns As Integer) As Collection
  numLanes = lns
 'This function is implemented but not shown here
  ReDim lanes(numLanes) As Integer
End Function
'-----
Public Sub init(swmrs As Collection, lanes As Integer)
End Sub
'-----
Public Function sort(sw As Collection) As Collection
  ReDim asw(sw.count) As Swimmer
  'This function is implemented but not shown here
End Function
```

在上面的程序中，实际上还应该包含 calcLaneOrder 和排序函数的代码，但为了简化程序而省略了。每一个导出类产生一个 Seeding 类的实例，以便调用这些函数。

我们接着可以十分方便地创建两个具体的 Seeding 类的子类：StraightSeeding 和 CircleSeeding。PrelimEvent 类将返回一个 CircleSeeding 实例，TimedFinalEvent 类将返回一个 StraightSeeding 实例。这样我们可以看到有两个层次结构：一个是 Events；另一个是 Seeding。我们可以通过图 10-1 和 10-2 理解这两个层次结构。

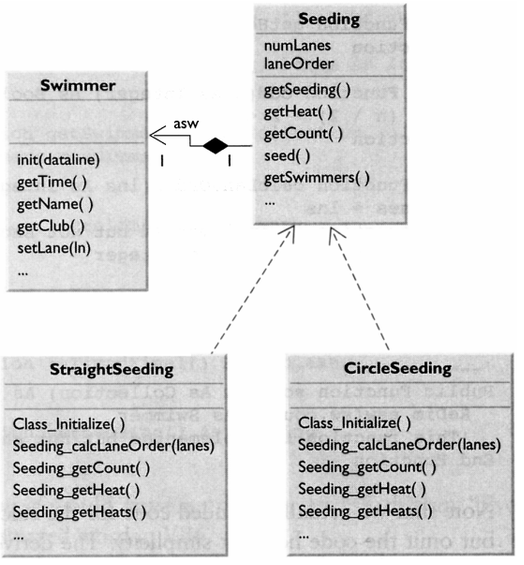


图 10-1 Seeding 类之间的类关系

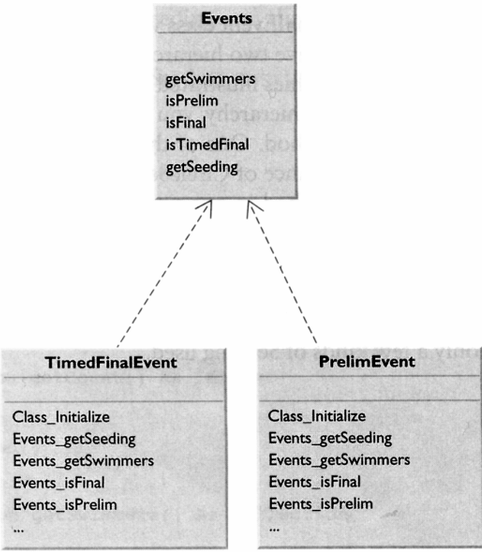


图 10-2 Events 类

在 Events 层次结构中，你将看到两个导出的 Events 类都包含一个 getSeeding 方法。一个返回 StraightSeeding 实例，而另一个返回 CircleSeeding 实例。因此你会发现，这里没有一个像简单工厂示例中的决策点。的确，像 Events 类初始化一样，决定就是确定哪一个 Seeding 类将被初始化。

这两个类的层次结构看上去有一个一一对应关系，这是不需要的。可能有多个 Events 类而仅有几个 Seeding 类被使用。

10.1 Swimmer 类

现在让我们了解一下 Swimmer 类，它包括如下属性：运动员的名字、注册时间、成绩、参加预赛的地点和进入决赛后的道次。当你从 Events 类中间接地调用 Seeding 类中的 getSeeding 方法时，Events 类从数据库中（这里用的是文件）读出 Swimmer 记录后放入

Seeding 类的集合 Collection 中。

10.2　Events 类

我们已经了解到前面的抽象 Events 类。在实际的使用中，用该类读入运动员的数据，并把读入的数据导入 Swimmer 类的实例去解析。

```
Option Explicit
'Class Events
Private numLanes As Integer
Private swimmers As New Collection  'list of swimmers
'-----
Public Sub init(Filename As String, lanes As Integer)
Dim f As Integer, s As String
Dim sw As Swimmer
Dim fl As New vbFile
'read in the data file in the constructor
f = FreeFile
numLanes = lanes
Set swimmers = New Collection
  'read in swimmers from file
  Filename = App.Path & "\" & Filename
  fl.OpenForRead Filename
  s = fl.readLine

  While (Not fl.fEof)
    Set sw = New Swimmer     'create each swimmer
    sw.init s                'and initialize it
    swimmers.Add sw          'add to list
    s = fl.readLine               'read another
  Wend
 Close #f
End Sub
'-----
Public Function getSwimmers() As Collection
 Set getSwimmers = swimmers
End Function
'-----
Public Function isPrelim() As Boolean
End Function
'-----
Public Function isFinal() As Boolean
End Function
'-----
Public Function isTimedFinal() As Boolean
End Function
'-----
Public Function getSeeding() As Seeding
End Function
```

在基类 Events 中，isPrelim 方法、isFinal 方法和 isTimedFinal 方法是空的，这些方法将

在导出类实现。下面的 PrelimEvent **类返回一个** CircleSeeding **类的实例。**

```
'Class PrelimEvent
Implements Events
Private numLanes As Integer
Private swimmers As Collection
Private evnts As New Events
Private sd As Seeding
Private Sub Class_Initialize()
 Set evnts = New Events
End Sub
'-----
Private Function Events_getSeeding() As Seeding
 Set sd = New CircleSeeding
 sd.init swimmers, numLanes
 Set Events_getSeeding = sd
End Function
'-----
Private Function Events_getSwimmers() As Collection
 Set Events_getSwimmers = swimmers
End Function
'-----
Private Function Events_isFinal() As Boolean
 Events_isFinal = False
End Function
'-----
Private Function Events_isPrelim() As Boolean
 Events_isPrelim = True
End Function
'-----
Private Function Events_isTimedFinal() As Boolean
 Events_isTimedFinal = False
End Function
'-----
Private Sub Events_init(Filename As String, lanes As Integer)
 evnts.init Filename, lanes
 numLanes = lanes
 Set swimmers = evnts.getSwimmers
End Sub
```

TimedFinalEvent **返回一个** StraightSeeding **类的实例。**

```
'Class PrelimEvent
Implements Events
Private numLanes As Integer
Private swimmers As Collection
Private evnts As New Events
Private sd As Seeding
Private Sub Class_Initialize()
 Set evnts = New Events
End Sub
'-----
Private Function Events_getSeeding() As Seeding
 Set sd = New CircleSeeding
```

```
 sd.init swimmers, numLanes
 Set Events_getSeeding = sd
End Function
```

在两种情况下我们的事件类包含基 Events **类的实例，该实例读入数据文件。**

10.3　直接筛选

在实际实现该程序时，会发现大量的工作是在 StraightSeeding **类中完成，而** CircleSeeding **类的处理工作要少一些。所以我们对** StraightSeeding **类实例化，并拷入运动员集合和比赛的道次。**

```
Private Sub Seeding_seed()
Dim lastHeat As Integer, lastlanes As Integer
Dim heats As Integer, i As Integer, j As Integer
Dim swmr As Swimmer

Set sw = sd.sort(sw)
Set laneOrder = sd.calcLaneOrder(numLanes)
count = sw.count
lastHeat = count Mod numLanes
If (lastHeat < 3) And lastHeat > 0 Then
     lastHeat = 3    'last heat must have 3 or more
End If
count = sw.count
lastlanes = count - lastHeat
numheats = lastlanes / numLanes
If (lastHeat > 0) Then
    numheats = numheats + 1
End If
heats = numheats

'place heat and lane in each swimmer's object
  'details on CDROM
'Add in last partial heat
  'details on CDROM
End Sub
```

在上面的子程序中，当调用 getSwimmers **方法时，整个** Swimmers **的数组变成可用的。**

10.3.1　交叉筛选

由于 CircleSeeding **类的实现细节出自于** StraightSeeding **类，所以它拷入相同的数据。**

```
'Circle seeding method
Private Sub Seeding_seed()
Dim i As Integer, j As Integer, k As Integer
'get the lane order]
Set laneOrder = sd.calcLaneOrder(numLanes)
```

```
Set sw = sd.sort(sw)     'sort the swimmers
strSd.init sw, numLanes 'create Straight Seeding object
strSd.seed              'seed into straight order
numheats = strSd.getHeats   'get the total number of heats
  If (numheats >= 2) Then
      If (numheats >= 3) Then
          circlesd = 3       'seed either 3
      Else
          circlesd = 2       'or 2
      End If
      i = 1
      'copy seeding info into swimmers data
      'details on CDROM
  End If
End Sub
```

因为 CircleSeeding **调用了** StraightSeeding **类的实例，所以它拷入运动员集合对象和道次数据。之后我们调用** strSd.seed **进行直接筛选。由于我们总是在优胜者中直接筛选，所以我们简化了处理过程。为了简便，这里仅给出了最后两三次预赛的选拔处理细节，在此基础上进行讨论。**

10.4 我们的选拔程序

在这个示例中，我们在 Web **上采集了一组参加** 500 **公尺自由泳、**100 **公尺自由泳比赛的运动员数据，并使用这些数据构建** TimedFinalEvent **和** PrelimEvent **类。你可以在图** 10-3 **中看到选拔的结果。**

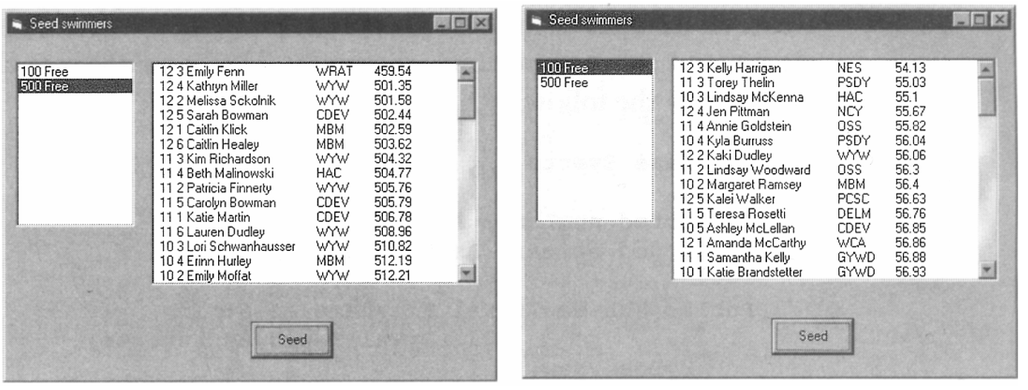


图 10-3 500 **公尺自由泳比赛直接筛选和** 100 **公尺自由泳比赛交叉筛选的窗口**

10.5　其他的类工厂

现在我们开始讨论在程序中如何读入运动员数据，这些数据决定了哪种类型的 Events 类将被产生。在读入数据时，根据数据产生正确的 Events 类的类型。代码在 Form_Load 事件中。

```
Dim ev As Events

Set ev = New PrelimEvent     'create a Prelim/final event
ev.init "100free.txt", 6     'read in the data
Seedings.Add ev.getSeeding   'get the seeding and add to collection
lsEvents.AddItem "100 Free"

Set ev = New TimedFinalEvent 'create a new Timed final event
ev.init "500free.txt", 6      'read in the data
Seedings.Add ev.getSeeding    'get the seeeding
lsEvents.AddItem "500 Free"   'and add to collection
```

显然，上面的程序片段是 EventFactory 类工厂的实例，由类工厂来确定哪个 Events 类应该产生。这与本专题开始时讨论的类工厂思想是相似的。

10.6　用 VB7 实现的选拔程序

在 VB7 中可以充分利用继承性，使得选拔程序简化。例如，定义 Events 类是一个抽象类，在它的子类 TimedFinalEvent 和 PrelimEvent 中我们实现其方法。在 VB7 中，我们在 Seeding 类中实现文件读方法，并且在子类中使用文件。与 VB7 不同，在 VB6 中我们不得不在 TimedFinalEvent 和 PrelimEvent 产生一个基类 Events 的实例，然后调用它的函数。抽象基类 Events 代码如下：

```
Public Class Events

    Protected numLanes As Integer
    Protected swmmers As Swimmers
    '------
    Public Sub New(ByVal Filename As String, _
                   ByVal lanes As Integer)
        MyBase.New()
        Dim s As String
        Dim sw As Swimmer
        Dim fl As vbFile

        fl = New vbFile(filename)   'Open the file
        fl.OpenForRead()

        numLanes = lanes            'Remember lane number
        swmmers = New Swimmers()    'list of kids
```

```
        'read in swimmers from file
        s = fl.ReadLine

        While Not fl.feof
            sw = New Swimmer(s)     'create each swimmer
            swmmers.Add(sw)         'add to list
            s = fl.ReadLine         'read another
        End While
        fl.closeFile()
    End Sub
    '-----
    Public Function getSwimmers() As ArrayList
        Return swmmers
    End Function
    '-----
    Public Overridable Function isPrelim() As Boolean
    End Function
    '-----
    Public Overridable Function isFinal() As Boolean
    End Function
    '-----
    Public Overridable Function isTimedFinal() As Boolean
    End Function
    '-----
    Public Overridable Function getSeeding() As Seeding
    End Function
End Class
```

TimedFinalEvent 是 Events 的导出类，它产生了一个 StraightSeeding 类的实例。

```
Public Class TimedFinalEvent
    Inherits Events

    Public Sub New(ByVal Filename As String, _
              ByVal lanes As Integer)
        MyBase.New(Filename, lanes)
    End Sub
    '-----
    Public Overrides Function getSeeding() As Seeding
        Dim sd As Seeding
        'create seeding and execute it
        sd = New StraightSeeding(swmmers, numLanes)
        sd.seed()
        Return sd
    End Function
    '-----
    Public Overrides Function isFinal() As Boolean
        Return False
    End Function
    '-----
    Public Overrides Function isPrelim() As Boolean
        Return False
    End Function
```

```
    '------
    Public Overrides Function isTimedFinal() As Boolean
        Return True
    End Function
End Class
```

除了产生一个 CircleSeeding 类的实例和建立预赛和决赛标志之外，PrelimEvent 类与以前没有什么不同。下面是 getSeeding 方法。

```
Public Overrides Function getSeeding() As Seeding
        Return New CircleSeeding(swmmers, numLanes)
End Function
```

两种实现方法的风格是相似的，基类 Seeding 包含 sort 函数和 getLaneOrder 函数，并且导出类 StraightSeeding 和 CircleSeeding 仅在筛选方法有所变化。图 10-4 中画出了 VB7 版的类层次结构。

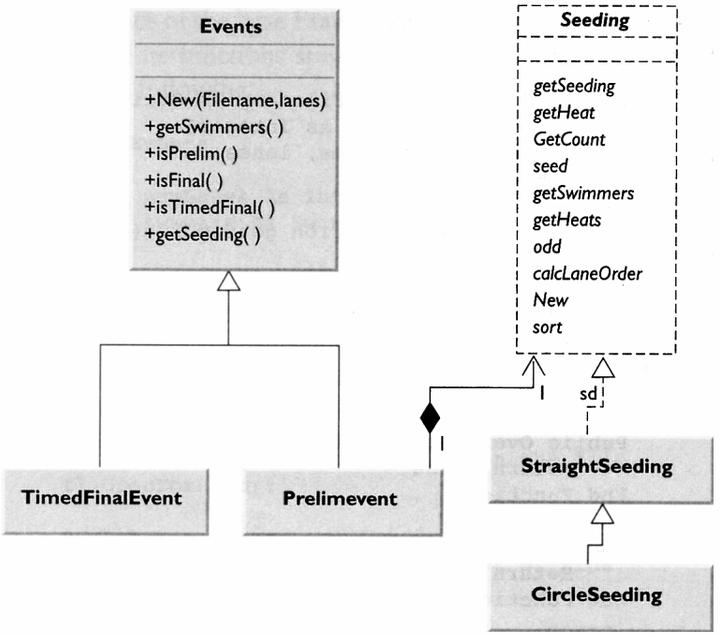


图 10-4　用 VB7 实现的筛选程序

10.7　什么时候使用工厂方法

你可以考虑在下面的情况下使用工厂方法

- 一个类无法预见何种对象类被产生

- 一个类使用它的子类来指定哪个对象被产生
- 你希望把产生类的选择局部化

下面是几种工厂模式的变形

1. 基类是一个抽象类，并且模式必须返回一个完整的工作类。
2. 基类包含默认的方法，这些方法被调用，除非默认的方法无效。
3. 通过参数传递的方式告诉工厂哪种类型的类被产生。在这种情况下类可以使用相同的方法名，但实现的内容完全不同。

思考题

假设种子选手从内道到外道排列。你应该建立何种类实现该筛选方法呢？

10.8 光盘上的程序

\Factory\Seeder	VB6 实现的筛选程序
\Factory\Seeder\vbNetSeeder	VB7 实现的筛选程序

第 11 章　抽象工厂模式

抽象工厂（Abstract Factory）模式是对工厂模式的更高一级的抽象。它可以在几个相关的类对象中返回一个类对象，每一个抽象工厂模式可以按需求返回其中的一组对象类。也就是说，抽象工厂是工厂的一个对象，它可以返回一组对象类。你可以使用简单工厂，决定在由抽象工厂返回的一组对象中哪一个被返回。

轿车生产工厂就是一个带有实际背景的示例。你希望丰田轿车工厂使用丰田轿车的零件，而伏特轿车工厂使用伏特轿车的零件。你可以把轿车工厂看成一个抽象工厂类，并且是一组相关类的一部分。

11.1　一个花卉工厂

现在考虑一个实际的示例，在该例中你会体会到抽象工厂的用处。假设你正在着手实现一个花园设计的程序。设计中的花园可能需要年生、季生和常年生的不同类型的植物。然而，无论你想设计什么样的花园，都要考虑下面的问题：

- 什么样的植物靠边种植
- 什么样的植物在中间种植
- 什么样的植物需要阴凉
- 其他的问题

我们用一个 VB6 的 Garden 类中的方法来回答上面的问题。

```
Public Function getCenter() As Plant
End Function
'-----
Public Function getBorder() As Plant
End Function
'-----
Public Function getShade() As Plant
End Function
```

下面的 Plant 对象返回植物名。

```
'Class Plant
Private plantName As String
'-----
Public Sub init(nm As String)
  plantName = nm        'save the plant name
End Sub
'-----
Public Function getName() As String
  getName = plantName   'return the plant name
End Function
```

从设计模式的观点看，Garden 接口就是一个抽象工厂。它定义了一个具体类的方法，并可以返回一个具体的类。下面用抽象工厂的方法分别返回上面问题中提出的几种类型植物类。该抽象工厂也可以返回一些专门的信息，如土壤的 pH 值或土壤湿度测试值。

在实际的系统中，每一种植物的信息可能都存放在数据库中。在这个示例中，将要返回一个类型的植物，例如季生植物，下面是具体的程序：

```
'Class VeggieGarden
Implements Garden
Private pltShade as Plant, pltBorder as Plant
Private pltCenter As Plant
'-----
Private Sub Class_Initialize()
 Set pltShade = New Plant
 pltShade.init "Broccoli"
 Set pltBorder = New Plant
 pltBorder.init "Peas"
 Set pltCenter = New Plant
 pltCenter.init "Corn"
End Sub
'-----
Private Function Garden_getBorder() As Plant
  Set Garden_getBorder = pltBorder
End Function
'-----
Private Function Garden_getCenter() As Plant
  Set Garden_getCenter = pltCenter
End Function
'-----
Private Function Garden_getShade() As Plant
  Set Garden_getShade = pltShade
End Function
```

同样，我们可以创建一个 PerennialGarden 类和 AnnualGarden 类。因为它实现了父类中列出的方法，所以把它们称为具体工厂。现在我们有一系列的 Garden 对象，每一个对象返回一个 Plant 对象。图 11-1 是类的层次结构图。

可以十分容易地构建我们的抽象工厂程序，返回这些 Garden 对象。图 11-2 是一个程序界面，当用户选择相应的单选按钮时，将产生对应的花卉。

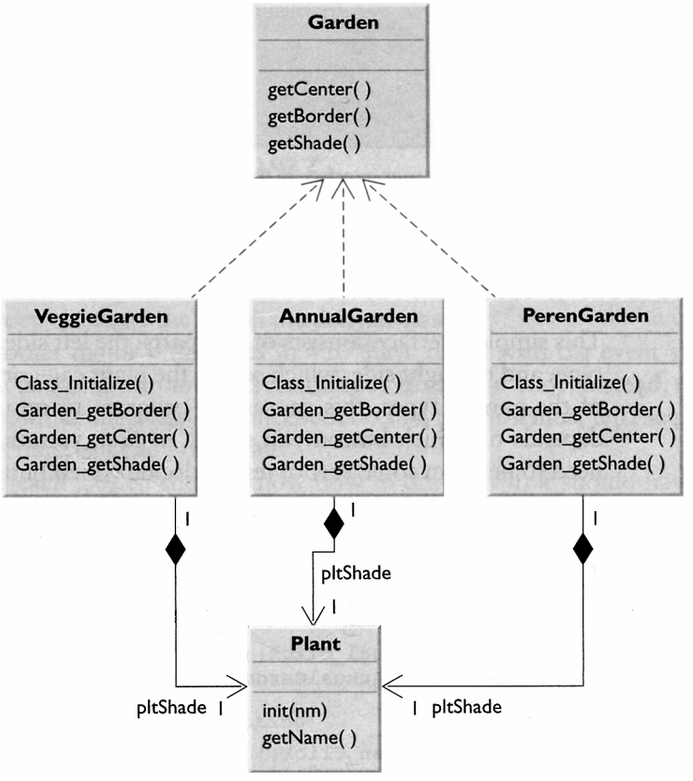


图 11-1　在 Gardener 程序中的主要对象

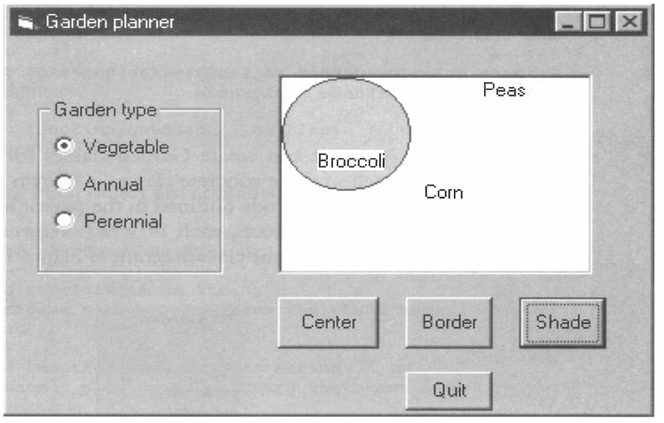


图 11-2　Gardener 程序的用户界面

11.2 用户接口如何工作

该程序的用户接口包含两个部分：左边一部分用于选择花卉类型；而右边一部分用于选择植物的种类。这两部分构成一个花卉植物的选择。首先，你可能希望测试一下各种按钮的选择效果，然后实列化 Concrete Factory 类。你可以先在左侧选择一个花卉的类型，即产生一个具体类工厂。你也可以先按下右侧的按钮，选择花卉的种类，然后选择改变当前花卉的类型，这时将返回一个花卉的名字，并把名字显示在屏幕上。

```
Private Sub opAnnual_Click()
  Set gden = New AnnualGarden   'select Annual garden
End Sub
'-----
Private Sub opPeren_Click()
  Set gden = New PerenGarden    'select Perennial garden
End Sub
'-----
Private Sub opVeggie_Click()
  Set gden = New VeggieGarden   'select vegetable garden
End Sub
```

下面是一个实现屏幕显示的程序片段，该片段实现了花卉名字在屏幕上刷新显示。然后，当要显示植物名称的时候，我们用 XORing 删去旧名字，并从当前的 Garden 中获得一个正确的名称后显示在指定位置上。

```
'-----
Private Sub btCenter_Click()
  Set plt = gden.getCenter  'get the center plant
  drawCenter plt.getName    'and draw it's name
End Sub
'-----
Private Sub drawCenter(st As String)
  pcGarden.PSet (1200, 1000)
  pcGarden.Print oldCenter   'XOR out old name
  pcGarden.PSet (1200, 1000)
  pcGarden.Print st          'draw in new name
  oldCenter = st             'remember this name so we can erase
End Sub
```

11.3 用 VB7 实现一个抽象工厂

用 VB7 实现一个花园设计程序有些不同，在 VB7 中可以定义一个花卉接口，然后再定义具体的类来实现接口中定义的方法。在 Garden 类中可以极容易地实现接口中的一些方法，其他的方法在导出点中实现。

另一些不同点是 VB7 需要建立事件模型。在 VB7 中，你不能从代码直接写屏幕。实际上屏幕的刷新是用激发 OnPaint 事件来完成的，你必须告诉画笔程序哪个对象将被执行。

因为花卉应该知道如何显示它们自己的属性，所以它应该有一个方法来实现具体的操作。我们是通过提供一个按钮来启动显示方法的，所以需要设置一个布尔变量来指示显示开始。

我们首先建立一下简单的 Plant 类，该类中通过构造函数导入植物名。

```
Public Class Plant
    Private plantName As String
    '-----
    Public Sub New(ByVal nm As String)
        MyBase.New()
        plantName = nm          'save the plant name
    End Sub
    '-----
    Public Function getName() As String
        getName = plantName    'return the plant name
    End Function
End Class
```

然后，我们产生一个基类 Garden，该类中包含 getShade、getCenter 和 getBorder 方法。由于 Garden 类本身实现了具有绘画功能的方法，所以我们不需要实现上述 3 种方法。

```
Public Class Garden
    'protected objects are accessed by derived classes
    Protected pltShade, pltBorder, pltCenter As Plant
    Protected center, shade, border As Boolean

    'These are created in the constructor
    Private gbrush As SolidBrush
    Private gdFont As Font

    'Constructor creates brush and font fro drawing
    Public Sub New()
        MyBase.New()
        gBrush = New SolidBrush(Color.Black)
        gdFont = New Font("Arial", 10)
End Sub
'-----
```

显示部分通过一个简单方法实现，它首先检测布尔变量的值，如果是真，则显示各种植物名。

```
Public Sub draw(ByVal g As Graphics)
        If border Then
            g.DrawString(pltBorder.getName, gdFont, _
                gbrush, 50, 150)
        End If
        If center Then
            g.DrawString(pltCenter.getName, gdFont, _
                gbrush, 100, 100)
        End If
        If shade Then
```

```
            g.DrawString(pltShade.getName, gdFont,_
                gbrush, 10, 50)
        End If
End Sub
```

然后，我们加入 3 **个具体实现显示功能的方法。**

```
    Public Sub showCenter()
        center = True
    End Sub
    '-----
    Public Sub showBorder()
        border = True
    End Sub
    '-----
    Public Sub showShade()
        shade = True
    End Sub
    '-----
    Public Sub clear()
        center = False
        border = False
        shade = False
    End Sub
```

现在，用于 3 **个花园的** 3 **个导出类都非常简单，它们只包括** 3 **个花卉的构造函数。下面是一个完整的** AnnualGarden **类。**

```
Public Class AnnualGarden
    Inherits Garden
    Public Sub New()
        MyBase.New()
        pltShade = New Plant("Coleus")
        pltBorder = New Plant("Alyssum")
        pltCenter = New Plant("Marigold")
    End Sub
End Class
```

注意，植物的名字是在构造函数中设置的，这 3 **个植物变量是** Garden **类的一部分。**

11.3.1 PictureBox 框

我们在 PictureBox **内画一个圆用于表示阴凉区域，并在其中显示植物的名字。这样需要在** PictureBox **类中加入一个** OnPaint **方法，而不是在** GardenMaker **中增加该方法。一种加入的方法是建立** PictureBox **的子类，在子类中增加** OnPaint **方法，实现画圆和显示植物名字的功能。**

```
Public Class GdPic
    Inherits System.Windows.Forms.PictureBox
    Private gden As Garden
    Private loaded As Boolean
```

```
    Private br As SolidBrush
    '------
    Private Sub init()
        br = New SolidBrush(Color.Gray)
    End Sub
    '------
    Public Sub setGarden(ByVal gd As Garden)
        gden = gd
        gden.clear()
        Refresh()
        loaded = True
    End Sub
    '------
    Protected Overrides Sub OnPaint( _
       ByVal pe As System.Windows.Forms.PaintEventArgs)
        Dim g As Graphics = pe.Graphics
        'draw the circle
        g.FillEllipse(br, 5, 5, 100, 100)
        If loaded Then
            'have the garden draw itself
            gden.draw(g)
        End If
    End Sub
End Class
```

注意，我们不用每一次都删除植物名字，这是由于 Paint **方法仅当图片需要重新显示时才被调用。**

11.3.2　处理单选按钮和按钮事件

当点击 3 **个单选按钮中的一个时，就可以产生一个新的花卉，并把所产生的花卉名字传递给** PictureBox **类。**

```
Private Sub opPerennial_CheckedChanged( _
            ByVal sender As System.Object, _
            ByVal e As System.EventArgs) _
            Handles opPerennial.CheckedChanged
        gden = New PerennialGarden()
        pbox.setGarden(gden)
End Sub
```

当点击一个按钮显示植物名字时，你实际上在调用 Garden **方法显示植物名，该方法通过调用** PictureBox **类的** Refresh **方法实现屏幕刷新。**

```
Protected Sub ckBorder_CheckedChanged( _
              ByVal sender As Object, _
              ByVal e As System.EventArgs)
        gden.showBorder()
        pBox.refresh()
End Sub
```

用 VB7 实现的 Gardener 类和相应的 UML 图标分别如图 11-3 和图 11-4 所示。

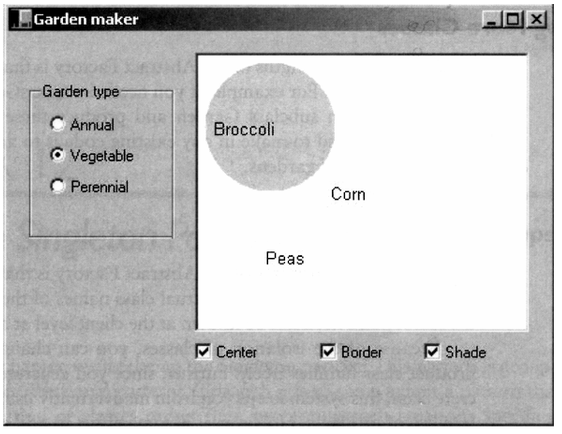


图 11-3 用 VB7 实现的 Gardener 程序

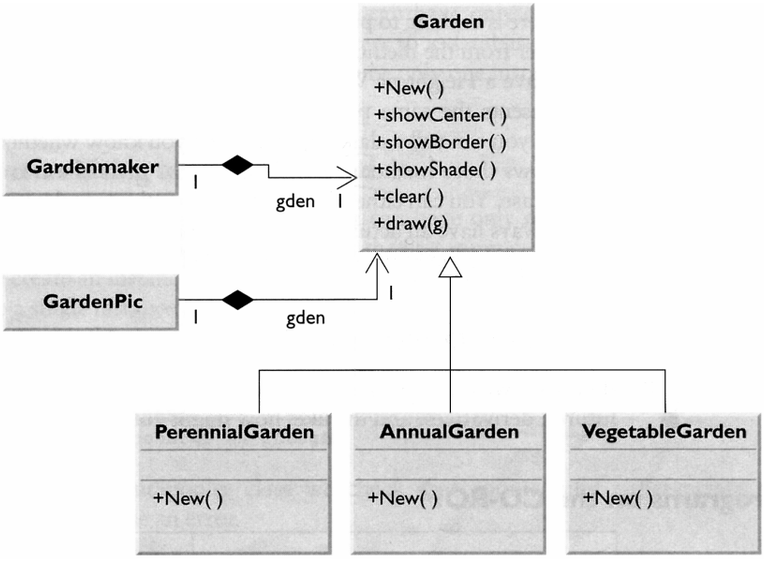


图 11-4 用 VB7 实现的 Gardener 程序的 UML 图表

11.4 增加更多的类

抽象类工厂的更强大的功能是可以方便地产生新的子类。例如，如果需要产生 GrassGarden 或 WildFlowerGarden，你可以通过创建 Garden 的子类来实现你的想法。你需

要真正改变的只是在现有的代码基础上，增加选择新类型花卉的选择语句。

11.5　抽象工厂评价

使用抽象工厂的一个主要目的是它可以实现抽象类与实例类的分离。实例类的类名被隐藏在类工厂中，客户端根本不需要知道。

由于抽象类具有的这种特性，你可以自由地改变具体类的内容。也就是说，由于你仅产生一种类型的具体类，系统就会避免与其他类成员产生冲突。当然，在你增加新的类时，需要定义新的没有二义性的条件，这些条件使得新的类被返回。

所有的抽象工厂产生的子类只有一个基类，而且这些子类可以有自己的方法，不同子类中的方法可以不同。例如 BonsaiGarden 类中有 Height 方法和 WateringFrequency 方法，这些方法在其他类中是不存在的。抽象类的这种隔离性引发一个问题：你不知道哪些类方法可供调用，除非你知道是否子类中的方法可以被调用。有两种相似的解决此问题的方法。其一是你在基类中定义所有的方法，即使在具体类中不一定有实际的函数；其二是定义一个新的基类的接口，该接口包括所有的方法及所有花园类型的子类。

思考题

如果你正在编写一个投资分析程序，如股票、公债等，你如何用抽象工厂实现呢？

11.6　光盘中的程序

\AbstractFactory\GardenPlanner	VB6 实现的 Gardener 程序
\AbstractFactory\VBNet\Gardenmaker	VB7 实现的 Gardener 程序

第 12 章　单一类模式

这一章将学习单一类模式。把单一类模式放到“生成模式”部分里来学习，似乎有些不恰当。但从单一类模式仅生成一个单一实例角度考虑，这样组织有一定的道理。单一类的示例很多，例如你只需要一个窗口管理程序或打印池，再如你可能只需建立一个串行口 COM1 的实例。

12.1　使用静态方法产生单一类

一种比较容易的方法是使用外部静态变量来产生单一类实例，在该方法中，我们在第一个实例中设立一个静态变量，并在每一次创建类的实例时检查。然后定义一个 Class_Initialize 方法创建一个单一类实例或引起错误。

在第一个示例中，我们创建一个 public 型的 spool_counter 变量，以便可以全局访问。

```
'Singleton PrintSpooler static constant
Public spool_counter As Integer
```

在 PrintSpooler 类中，我们检查计数器并且产生一个实例或者引发错误。

```
'------
Private Sub Class_Initialize()
 If spool_counter > 0 Then
    Err.Raise vbObjectError + 1            'raise error
 End If
 spool_counter = spool_counter + 1
End Sub
```

虽然上面的程序仅产生一个实例，但它没有提供全局访问点。为了做到这样，我们必须允许类能创建一个可全局访问的实例，并使用 getSpooler 方法返回实例。

```
'Class PrintSpooler
'------
Private Sub Class_Initialize()
 If spool_counter = 0 Then
    Set glbSpooler = Me                    'save legal instance
    spool_counter = spool_counter + 1      'count it
 End If
End Sub
'------
Public Function GetSpooler() As PrintSpooler
```

```
    Set GetSpooler = glbSpooler     'return legal instance
End Function
```

在存放实例的模块级代码上，我们仍然使用两个全局变量。

```
'Singleton PrintSpooler static constants
Public spool_counter As Integer
Public glbSpooler As PrintSpooler
```

使用上述方法的一个主要优点是，你不用担心在单一类已经存在的情况下会产生错误。即总是获得相同的 Spooler **实例。然而，如果你在** PrintSpooler **类中创建一个实例，并选择不使用错误处理时，你必须在** Spooler**（存储池）中建立一个标志，用于指示仅一个实例是合法的。这里使用** legalInstance **变量来确定只有一个打印输出在合法的实例中发生。**

```
Option Explicit
Private legalInstance As Boolean     'true for only one
'Class PrintSpooler
'-----
Private Sub Class_Initialize()
  If spool_counter = 0 Then             'create and save one instance
    legalInstance = True                'flag it
    Set glbSpooler = Me                 'save it
    spool_counter = spool_counter + 1   'count it
  Else
    legalInstance = False               'not the legal one
    Err.Description = "Illegal spooler instance"
    Err.Raise vbObjectError + 1     'could raise an error
  End If
End Sub
'-----
Public Function GetSpooler() As PrintSpooler
    Set GetSpooler = glbSpooler  'return the legal one
End Function
'-----
Public Sub Printit(str As String)
  If legalInstance Then
   'test to make sure this isn't called directly
    MsgBox str
  End If
End Sub
'-----
Private Sub Class_Terminate()
 'terminate legal class
 If legalInstance Then
   Set glbSpooler = Nothing
   spooler_cnt = 0
 End If
End Sub
```

如果需要改变程序，允许产生两个或多个实例，你可以十分方便地在模块代码建立一个实例数组来实现多个实例的产生。

12.2 捕获错误

一类错误是用户自己定义的，并通过定义常量加入到 vbObjectError 错误常量中。你可以捕获在类模块中出现的错误，并在调用程序中处理。方法是在 Tools|Option 下的 General 标签中选择“ Break on unhandled errors ”；否则，错误在类中被激发，而不是上传到调用程序中处理。

```
'PrintSpooler Driver form
Dim prSp As PrintSpooler
'-----
Private Sub GetSpooler_Click()
On Local Error GoTo nospool
 'create a spooler
    Set prSp = New PrintSpooler 'create class
    Set prSp = prSp.GetSpooler  'get legal instance
    errText.Text = "Spooler created"
spexit:
    Exit Sub
'if the spooler causes an error we will get this message
nospool:
  errText.Text = "Spooler already exists"
  Resume spexit
End Sub
```

12.3 提供单一类的全局访问点

因为每一个单一类提供一个类的单一全局访问点，所以在程序中必须设计一个贯穿程序的单一类参考。

一个方法是在程序首部产生一个所有单一类的集合，并把这些单一类集合作为参数提供给主类。

```
Dim Singletons as New Collection
Singletons.add prSpl, "PrintSpooler"
```

这种方法的一个缺点是主程序可能不需要知道所有的单一类，这样有时会产生不可预知的后果。

好一点的解决方法是为所有的单一类产生一个注册项。单一类每次被初始化时，相应的注册项被标记，在程序的任何地方可以使用标识串访问单一类的实例并获得实例变量。

当然，注册项本身也可能是单一类对象，这时你可以通过全局变量、初始化函数或各种函数把注册项访问点提供给主程序。

12.4　MSComm 控件作为单一类

MSComm 控件提供了方便的访问 PC 串口的方法。你可以设置端口号、波特率、奇偶效验位、数据位和暂停位，还可以打开串口、发送或接收数据等。通常 PC 机的基本输入输出系统（BIOS）允许有 4 个串口，分别称为 COM1、COM2、COM3 和 COM4。实际上配置的串口往往要少于 4 个，它们分别被映射到 MSComm 控件。另外，在同一时刻只有一个串口实例被初始化，这是因为两个程序或者设备不能同时使用同一个串口。

串口是展现一种资源作为单一类的十分恰当的示例，这是因为在同一时刻只能有一个程序访问串口，即使是在一个程序内，也只有一个模块或一组模块通过串口进行通信。

串口应用中需要建立两个单一类：一个单一类用于管理串口的集合，并判断是否只有一个串口实例被初始化；另一个单一类作为单一类实例的串口对象。

在示例程序中，我们在一个非可视化窗体中创建一个 MSComm 控件的数组，并提供访问控件的方法。非可视化窗体如图 12-1 所示。

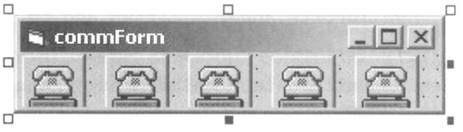


图 12-1　非可视化窗体模块

接下来可以创建一个公共方法，用于列出哪个串口是可用的，并打开此串口。

```
Dim coms As New Collection
Dim loaded As Boolean
Dim validPorts As New Collection
Dim availablePorts As New Collection
'------
Private Sub Form_Load()
 loaded = False
 loadComms   'load coms into collection
End Sub
'------
Private Sub loadComms()
 'create array of 5 MSComm controls
 If Not loaded Then
    coms.Add comm(0)
    coms.Add comm(1)
    coms.Add comm(2)
    coms.Add comm(3)
    coms.Add comm(4)
    loaded = True   'collection is now loaded
 End If
End Sub
```

你可能要询问哪个串口是可用的，哪个不可用。对每一种情况你只需要遍历串口列表，并通过打开每个串口来检验。如果某一个串口已经被打开，你将得到“Port already open”（串口已被打开）错误；如果此串口不存在，你将得到“Invalid port number”（不可用的串口号）错误。下面是实现上述功能的代码。

```
Public Function getValidPorts() As Collection
 Dim cm As MSComm, i As Integer
 Dim valid As Integer
 loadComms                'make sure ports list has been loaded
 On Local Error GoTo cmsb
 For i = 1 To 5
   Set cm = coms(1)       'get any MSComm control
   valid = True           'assume it is valid
   cm.CommPort = i        'set the port number
   cm.PortOpen = True     'try to open it
   cm.PortOpen = False    'if it opens, close it
   If valid Then          'if not negated by error
     validPorts.Add i     'then add to list
   End If
 Next i
 Set getValidPorts = validPorts  'return list
Exit Function

'error handling for opening ports
cmsb:
  Select Case Err.Number
    Case 8002    'invalid port
      valid = False
    Case 8005    'port already open
      valid = True
  End Select
 Resume Next
End Function
```

12.4.1 可用的串口

我们可以构造一个相似的方法来返回一个可用串口列表。该方法是在“Port already open”和“Invalid port number”错误都没有发生时，把串口加入到列表中。

图 12-2 展示了一个实现依次打开每一个串口，并检查是否有错误发生的程序。它同时展示了 getValidPorts 和 getAvailablePorts 方法。

在图 12-2 中的 Com 选择按钮列表，由 comClick 方法的调用构成，该方法在列表框中显示错误信息或串口的状态。

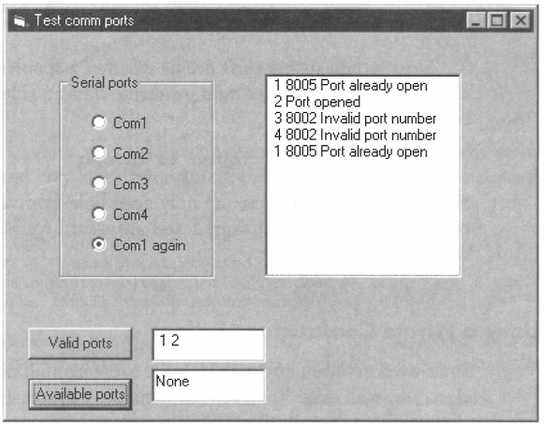


图 12-2　可用的串口和使用的串口

```
Private Sub comClick(i As Integer)
Dim com As MSComm

On Error GoTo nocom
Set com = comFrm.getComm(i)              'Get a port from the array
com.CommPort = i
comFrm.getComm(i).PortOpen = True        'try to open it
'display status of port
List1.AddItem Str$(i) & " Port opened"
pexit:
Exit Sub

nocom:
'or display the error message
     List1.AddItem Str$(i) & Str$(Err.Number) & " " & _
                   Err.Description
     Resume pexit
End Sub
```

在上面的程序中，哪一个是单一类呢？每一个 MSComm 控件就像打开的书一样，一旦一个串口被打开，该串口就变成了一个单一类。你不可以用同一串口号打开另一个 MSComm 控件。类似地，可以把非可视化的 commForm 控件视为单一类。在一个程序中有多个串口，但是没有任何理由同时打开两个以上的串口。图 12-2 实现了管理并显示每一个串口的结果。

12.5　用 VB.NET 实现单一类

在 VB7 中提供了一组特殊的创建和使用单一类的方法，这说明了 VB7 的许多强大特性。我们希望系统中能够分配一个而且是仅有的一个打印机存储池，并希望知道它是否被

分配。在 VB7 中，我们可以产生一个拥有共享方法的类，这些共享方法可以通过类名来调用，而不是通过创建类的实例。在其他语言中，共享方法也称静态方法（static methods）。例如，可以直接使用 VB7 中的 File 类中的共享方法来获得文件中的信息，而不用在使用时创建该文件的实例。

```
If File.Exists("foo.txt")
```

MessageBox（信息框）类是另一个有共享方法的示例，该类中惟一的方法就是共享方法。

```
MessageBox.Show("Error in program!")
```

你也可以创建自己的具有共享方法的类，做法是在方法定义时在方法名前加 Shared 修饰符。共享方法的优点是可以使用类名进行调用。当然，在实例中也可以使用类名调用共享方法。

在单一类示例中，可以定义一个方法用于得到一个合法的 spooler 实例。

```
Public Shared Function getSpooler() As Spooler
```

那么如何确定一个打印机存储池是否已被分配呢？你可以在类中定义一个共享的计数变量，用该变量来确定一个打印机存储池是否已被分配。

```
Private Shared Spool_counter as Integer
```

上面的共享变量只能通过共享方法来访问。一个方法如果不是共享方法，并且是通过类实例调用的就不能访问这些共享变量。

12.5.1 使用私有的构造函数

因为我们不希望每个人都能创建多个 Spooler 类的实例，所以我们将定义一个新的私有的构造函数。这一改变的结果是 Spooler 类产生的实例就是它本身。这看起来是循环定义，但我们发现可以利用 Spooler 类的共享方法来产生一个 Spooler 的实例。代码如下：

```
Public Shared Function getSpooler() As Spooler
    getSpooler = New Spooler
End Function
```

然后我们可以通过共享方法返回的 Spooler 实例调用非共享方法，如 Print 方法。

```
spl = Spooler.getSpooler
spl.Print("Hi there")
```

12.5.2 VB7 实现的单一类中的错误处理

现在我们需要做的是，在没有分配一个打印机存储池或试图分配更多打印机存储池时

指出错误所在。较理想的方法是使用异常。我们可以产生一个 SpoolerException **类，该类为单一类提供一个相应的错误信息，代码如下：**

```
Public Class SpoolerException
    Inherits Exception
    Private mesg As String
    '-----
    Public Sub New()
        MyBase.New()
        mesg = "Only one spooler instance allowed"
    End Sub
    '-----
    Public Overrides ReadOnly Property Message() As String
        Get
            Message = mesg
        End Get
    End Property
End Class
```

然后我们需要做的就是，在 Spooler **类的构造函数中检验是否是第一个并且是合法的实例被产生。如果不是这样，就创建一个合法的实例，如果是这样，就抛出一个异常。**

```
Public Class Spooler
    Private Shared Spool_counter As Integer
    Private Shared glbSpooler As Spooler
    Private legalInstance As Boolean
    '-----
    Private Sub New()
        MyBase.New()

        If spool_counter = 0 Then    'create one instance
            glbSpooler = Me          'save it
            spool_counter = spool_counter + 1    'count it
            legalInstance = True
        Else
            legalInstance = False
            Throw New SpoolerException()
        End If
    End Sub
```

使用相似的风格扩展 getSpooler **共享方法，把抛出的异常上传到调用程序。**

```
Public Shared Function getSpooler() As Spooler
        Try
            glbSpooler = New Spooler()
        Catch e As Exception
            Throw e              'pass on to calling program
        Finally
            getSpooler = glbSpooler  'or return legal one
        End Try
    End Function
```

12.6 一个 VB.NET 实现的 SpoolDemo 程序

现在重新编写简单的 SpoolDemo 程序，该演示程序显示是否有一个 Spooler 可用。它在两个按钮点击事件中实现错误的捕获和处理。

```
Protected Sub Print_Click(ByVal sender As Object, _
                          ByVal e As System.EventArgs)
        Try
            spl.Print("Hi there")
        Catch ex As Exception
            ErrorBox("No spooler allocated")
        End Try
    End Sub
    '------
    Private Sub ErrorBox(ByVal mesg As String)
        MessageBox.Show(mesg, "Spooler Error", _
                        MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Sub
    '------
       Private Sub btGetSpooler_Click( _
                   ByVal sender As System.Object, _
                   ByVal e As System.EventArgs) _
                   Handles btGetSpooler.Click
        Try
            Spl = Spooler.getSpooler
            TextBox1.Text = "Got spooler"
        Catch ex As Exception
            ErrorBox("Spooler already allocated")
        End Try
    End Sub
```

上面的程序产生一个简单的窗口，如图 12-3 所示，它提供的两个错误消息框如图 12-4 所示。

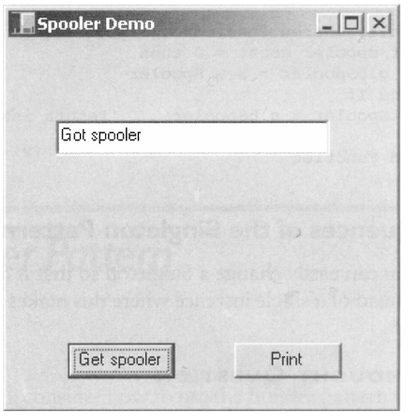


图 12-3 VB7 的 SpoolDemo 程序

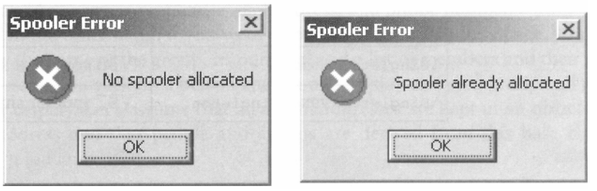


图 12-4　VB7 的 SpoolDemo 程序产生的错误消息

12.7　全局访问点

为了解释如何使用异常并把异常上传到调用程序去处理，我们不提供全局访问点。一旦我们调用共享 getSpooler 函数，就有一个附加调用用于抛出异常。这是一个处理异常的十分简化的方法。

```
Public Shared Function getSpooler() As Spooler
 If spooler_count = 0 then
   glbSpooler = New Spooler
 End If
GetSpooler = glbSpooler     'return instance

End Function
```

12.8　单一类模式其他结论

有时在你开发的系统中，希望对单一类多增加几个实例，这也是十分正常的。这种改进实现起来十分方便。

思考题

考虑一个 ATM 信用卡使用的示例。现有一个小偷得到了一个 ATM 信用卡的卡号，并用它窃取卡内的现金。你如何利用单一类实现一个程序来降低 ATM 信用卡使用的风险呢？

12.9　光盘上的程序

\Singleton\	VB6 实现的存储池单一类
\Singleton\comms	VB6 实现的串口单一类
\Singleton\VBNetSingleton	VB7 实现的存储池单一类

第 13 章　构造器模式

本章学习如何使用构造器模式来构造对象。在我们已经学习的类工厂模式中，可以通过参数传递控制信息给创建的方法，确定类工厂模式需要产生的子类。类工厂提供了共享算法的途径，但有时不仅需要共享算法，而且还希望提供不同的用户界面来显示数据。一个典型的示例是 e-mail 地址簿。在你的地址簿中可能有两类地址。一类是个人地址；另一类是单位地址。你可能希望对地址簿的显示方式进行调整，对个人地址按名、姓、公司、e-mail 地址和电话号格式显示。

另一方面，如果你正在显示一个单位地址页，你可能希望浏览单位名称、单位类别、单位成员和他们的 e-mail 地址。你点击 Person 按钮得到一种显示；点击 Group 按钮得到另一种显示。让我们假定所有的 e-mail 地址放在一个称为 Address 的对象内，而 Person 和 Group 是 Address 基类的子类，如图 13-1 所示。

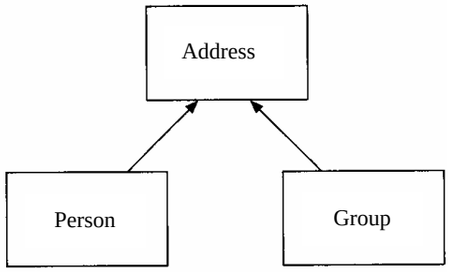


图 13-1　Person 和 Group 是 Address 的子类

我们将看到对象特性的不同显示，这些不同取决于我们点击的 Address 对象的类型。与类工厂模式不同，我们不仅需要产生不同的对象，而且需要产生不同的显示界面对象。构造器模式可以构造多个对象，比如依赖不同数据类型确定的不同方式的显示控件，这样用类表示数据，用窗体表示显示，你就可以把数据和显示方法分成简单的对象。

13.1　一个投资跟踪程序

现在分析一个较为简单的示例，它对用类建立用户界面是十分有用的。假设我们正在编写一个投资效益跟踪程序。程序中包括股票、公债和共有基金等投资项目。我们希望显示每种投资项目的持有金额，并由此确定投资的种类和效益分析。

即使我们不能预测某一时刻自己所拥有的每种投资金额，但我们希望屏幕的显示方便于投资金额很多的情况（如股票），也有利于投资金额很少的情况（如共有基金）。在每一种情况中，我们都希望有多重选择显示，以便可以策划不同的投资项目。如果是多个投资项目（如股票），则使用多种选择列表框；如果有 3 个或更少的投资项目，则使用一组复选框。我们希望 Builder 类根据显示内容产生一个界面，并有相同的方法来返回结果。

输出屏幕如图 13-2 所示。左边的图包括股票投资，右边包括公债投资。

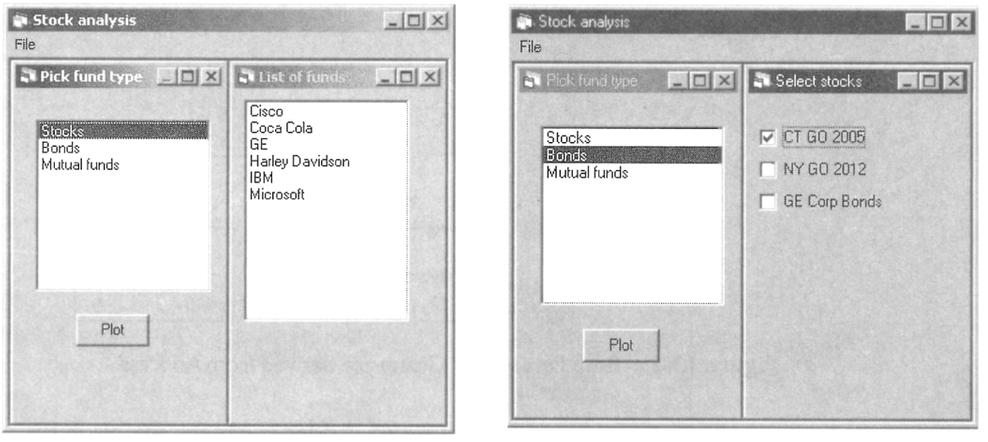


图 13-2　带有列表框的股票投资项窗体和带有复选框的公债投资项窗体

现在考虑如何用程序实现上面的的窗口界面。首先使用 MultiChoice 接口定义需要实现的方法。

```
'Interface MultiChoice
'This is the interface for the multi-Select windows
'-----
'get collection of all selected stocks
Public Function getSelected() As Collection
End Function
'-----
'get window containing multichoice controls
Public Function getWindow() As Form
End Function
'-----
'store list of stocks
Public Sub init(stocks As Collection)
End Sub
```

方法 getWindow 返回一个带有多重选择显示的窗口。这里使用了两种显示屏幕，一个用复选框，另一个用列表框。

```
'checkBox form
Implements MultiChoice
```

或

```
'Listbox form
Implements MultiChoice
```

接下来创建一个简单的 Factory 类，确定上述两种类哪一种被返回。

```
'Class StockFactory
'gets correct window for number of stocks presented
Public Function getBuilder(stocks As Collection)
 Dim mult As MultiChoice
 If stocks.Count <= 3 Then
   Set mult = New checkForm  'get check box form
 Else
   Set mult = New listForm   'get list box form
 End If
 mult.init stocks            'initialize it
 Set getBuilder = mult
End Function
```

在设计模式术语中，简单的 Factory 类被称为指挥者，而 MultiChoice 实际导出的类称为构造器。

13.2 调用构造器

在基于 VB6 实现的程序中，进行动态的生成和改变不是十分方便，所以我们将通过生成各种类型的多重选择窗口的实例构成 MDI 窗体，然后我们再逐一的初始化。

MDI 窗体组成的用户界面由位于窗体左侧的基金选择列表框和右侧一个空白的窗体构成。这个空白的窗体会逐渐地被填充。如图 13-3 所示。

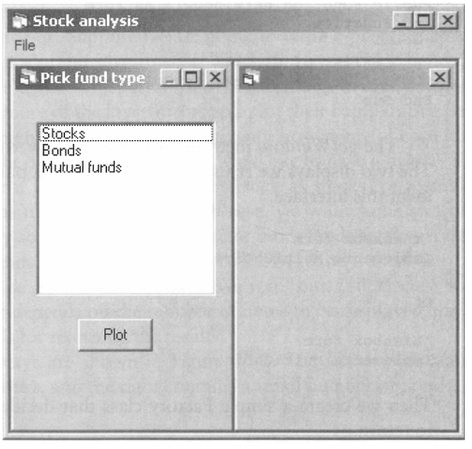


图 13-3 在选择投资项目之前的窗体

我们首先定义 Equities**（投资净值）接口。**

```
'Interface to Equities class
Public Function getFunds() As Collection
  'returns list of fund names
End Function
```

然后在 3 **个类中保存** 3 **种投资列表，它们分别称为** Stocks **类、**Bonds **类和** Mutuals **类，每一个类是** Equities **接口的不同实现。在** Funds **类中分别创建这** 3 **个类的实例，并根据所选择的投资类型确定返回相应的类实例。**

```
'Class funds
Private fundList As New Collection
Private eq As Equities
Private Sub Class_Initialize()
 'Creates a collection of equities
 fundList.Add New stocks
 fundList.Add New Bonds
 fundList.Add New Mutuals
End Sub
'-----
Public Function getFund(i As Integer) As Equities
 'return selected equity from collection
 If i > 0 And i <= fundList.Count Then
   Set getFund = fundList(i)
 End If
End Function
```

在程序初始化时，各种投资项目对象的初始值（数据）是任意设定的。

```
'Class stocks
Implements Equities
Private stockList As New Collection

Private Sub Class_Initialize()
 'add in arbitrary list of stocks
    stockList.Add "Cisco"
    stockList.Add "Coca Cola"
    stockList.Add "GE"
    stockList.Add "Harley Davidson"
    stockList.Add "IBM"
    stockList.Add "Microsoft"
End Sub
'-----
Private Function Equities_getFunds() As Collection
 'return collection of stocks
    Set Equities_getFunds = stockList
End Function
```

对 Bonds **类和** Mutuals **类也是如此。在实际的系统中，我们可能从文件或数据库中读入上述投资项目对象的数据。当用户在左侧的列表框中选择一个投资项目时，我们就可以从** Funds **类中得到该项目的投资净值，并由此确定右侧框的显示内容。**

```
Private Sub fundList_Click()
 Dim i As Integer, selStocks As Collection, eq As Equities

 'catch list box click selecting fund type
 i = fundList.ListIndex + 1
 If (i > 0) Then
   'get one type of fund
   Set eq = fnds.getFund(i)       'get equit from funds class
   Set selStocks = eq.getFunds    'get list from equity
   'get a multiChoice form from the factory
   Set mchoice = sfact.getBuilder(selStocks)
   'tell the parent MDI to show it
   mParent.setShowForm mchoice
 End If
End Sub
```

13.3 列表框构造器

最简单的两种构造器之一是列表框构造器，该构造器返回一个包含投资项目净值列表框的窗体。

```
'Listbox form
Implements MultiChoice
Private sels As Collection
'-----
Private Sub MultiChoice_init(stocks As Collection)
 Private i As Integer
 For i = 1 To stocks.Count
   List1.AddItem stocks(i)
 Next i
End Sub
```

另一个重要的方法是 getSelected **方法，该方法返回用户选择的投资项目的字符串集合。**

```
Private Function MultiChoice_getSelected() As Collection
Dim i As Integer
Set sels = New Collection
For i = 0 To List1.ListCount - 1
  If List1.Selected(i) Then
    sels.Add List1.List(i)
  End If
Next i
Set MultiChoice_getSelected = sels
End Function
```

13.4 复选框构造器

复选框构造器是另一个简单的构造器。该例中复选框构造器用来确定有多少元素需要

显示。本例中需要显示的元素只能为 0~3，所以我们先定义再适时显示。

```
Private Sub MultiChoice_init(stocks As Collection)
 Dim i As Integer
 'set captions for the check boxes we are using
 For i = 1 To stocks.Count
   ckFunds(i - 1).Caption = stocks(i)
 Next i
 'make the rest invisible
 For i = stocks.Count + 1 To 3
   ckFunds(i).Visible = False
 Next i
End Sub
```

getSelected 方法与前面的方法类似，代码如下：

```
Implements MultiChoice
Private sels As Collection
'-----
Private Function MultiChoice_getSelected() As Collection
 Dim i As Integer
 'create collection of checked stock names
 Set sels = New Collection
 For i = 1 To 3
   If ckFunds(i - 1).Value = 1 Then
    sels.Add ckFunds(i - 1).Caption
  End If
 Next i
 'return collection to caller
 Set MultiChoice_getSelected = sels
End Function
```

完整的 UML 类图如图 13-4 所示。

13.5　用 VB.NET 实现构造器

VB7 为实现 Builder 类提供了相对大的灵活性，这是因为我们可以访问允许从基础组件构造窗体的方法。例如，我们可以设计一个构造器，该构造器构造一个包括所需构件的显示板（Panel）。接下来我们把该显示板加到窗体中的适当位置。当显示发生变化时，新的显示板会取代旧的显示板。VB6 没有 Panel 类。而在 VB7 中，显示板就是一个无边界的容器，它可以容纳若干个窗口构件。就像我们在前面实现的那样，两个显示板将满足多重选择界面。

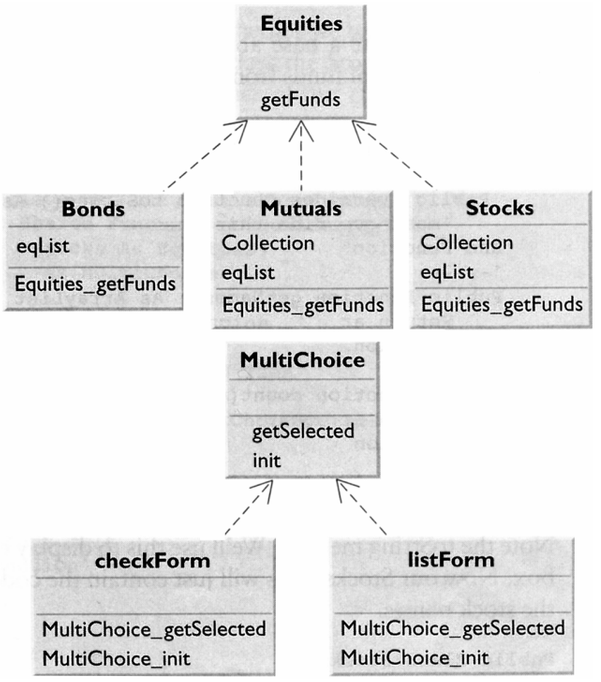


图 13-4 构造器的 UML 图

```
Public Interface MultiChoice
    'an interface to any group of components
    'that can return zero or more selected items
    'the names are returned in an Arraylist
    Function getSelected() As ArrayList
    Sub clear()              'clear all selected
    Function getWindow() As Panel
End Interface
```

我们将产生一个称为 Equities 的基础抽象类，并从该抽象类导出 Stocks、Bonds 和 Mutuals 三个子类。

```
Public MustInherit Class Equities
    Protected ar As Arraylist
    '-----
     Public Overrides Function toString() As String
        'must override this one
    End Function
    '-----
    Public Function getNames() As ArrayList
        Return ar
    End Function
    '-----
    Public Function count() As Integer
```

```
        Return ar.count
    End Function
    '-----
End Class
```

上面的程序段使用 toString 方法显示每一种类型的投资项目的列表框。现在 Stocks 类中包括调入股票名数组 ArrayList 的代码。

```
Public Class Stocks
    Inherits Equities
    '-----
    Public Sub New()
        MyBase.New()
        ar = New ArrayList()
        ar.Add("Cisco")
        ar.Add("Coca Cola")
        ar.Add("GE")
        ar.Add("Harley Davidson")
        ar.Add("IBM")
        ar.Add("Microsoft")
    End Sub
    '-----
    Public Overrides Function toString() As String
        Return "Stocks"
    End Function
End Class
```

其他所有代码（getNames 和 count）在 Equities 类中实现。Bonds 和 Mutuals 类与 Stocks 类一样简单。

13.5.1　股票类工厂

我们需要建立一个类来确定返回复选框还是列表框，这个类称为 StockFactory。但是我们仅需要该类的一个实例，所以创建带有一个共享方法的类。

```
Public Class StockFactory
    'This class has only one shared method
    Public Shared Function getBuilder(ByVal _
            stocks As Equities) _
             As MultiChoice

        Dim mult As MultiChoice

        If stocks.Count <= 3 Then
            'get check boxes
            mult = New checkChoice(stocks)
        Else
            'get a list box
            mult = New listChoice(stocks)
        End If
```

```
        Return mult
    End Function
End Class
```

13.5.2 复选框类

我们的复选框构造器构造一个显示板，包括 0~3 **个复选框，并把生成的显示板返回给调用程序。**

```
Public Class CheckChoice
    Implements MultiChoice

    Private stocks As ArrayList
    Private pnl As Panel
    Private boxes As ArrayList
    '------
    'create a Panel containing
    '0 to 3 check boxes
    Public Sub New(ByVal stks As Equities)
        MyBase.New()
        stocks = stks.getNames
        pnl = New Panel()
        boxes = New Arraylist() 'in an ArrayList
        Dim i As Integer
        For i = 0 To stocks.count - 1
            Dim Ck As New Checkbox()
            Ck.Location = _
               New System.Drawing.Point(8, 16 + i * 32)
            Ck.Text = stocks(i).toString
            Ck.Size = New Size(112, 24)
            Ck.TabIndex = 0
            Ck.TextAlign = _
               ContentAlignment.MiddleLeft
            boxes.add(ck)               'internal array
            pnl.Controls.add(ck)        'add into panel
        Next i

    End Sub

End Class
```

用于返回窗口和选定类型的列表的方法显示如下。这里使用 Ctype **函数，实现由一个** ArrayList **数组返回的对象类型到实际需要的复选框类型之间的转换。**

```
    'clear all selected check boxes
    Public Sub clear() Implements MultiChoice.clear
        Dim i As Integer
        Dim ck As Checkbox
        For i = 0 To boxes.count - 1
            ck = CType(boxes(i), Checkbox)
            ck.Checked = False
```

```
        Next i
    End Sub
'--------
'gets list of selected names
Public Function getSelected() As ArrayList _
         Implements MultiChoice.getSelected
    Dim ar As New ArrayList()
    Dim i As Integer
    Dim ck As Checkbox

    For i = 0 To Boxes.count - 1
        ck = CType(boxes(i), Checkbox)
        If ck.Checked Then
            ar.add(ck.Text)
        End If
    Next i
    Return ar
End Function
'--------
'gets the Panel containing the check boxes
Public Function getWindow() As Panel _
        Implements MultiChoice.getWindow
    Return pnl
End Function
```

13.5.3　列表框类

列表框类创建一个多重选择列表框，将它插入到显示板，把名字装入列表。

```
Public Class ListChoice
    Implements MultiChoice

    Private stocks As ArrayList
    Private pnl As Panel
    Private lst As ListBox
    '--------
    'create a panel containing a
    'multiselectable list box
    Public Sub New(ByVal stks As Equities)
        MyBase.New()
        stocks = stks.getNames 'get the names
        pnl = New Panel()
        'create the list box
        lst = New ListBox()
        lst.Location = New Point(16, 0)
        lst.Size = New Size(120, 160)
        lst.SelectionMode = _
              SelectionMode.MultiExtended
        lst.TabIndex = 0
        'add it into the panel
        pnl.Controls.Add(lst)
        'add the names into the list
```

```
        Dim i As Integer
        For i = 0 To stks.count - 1
            lst.items.add(stocks(i))
        Next i
    End Sub
```

由于这是一个多重列表框，所以可以在 SelectedIndices **集合中得到所有的供选择的内容。但该方法仅使用于多重选择列表框。该方法返回-**1**，代表一个单一选择列表框。我们使用该列表框装入被选中的内容，代码如下：**

```
Public Function getSelected() As ArrayList _
            Implements MultiChoice.getSelected
        Dim i As Integer
        Dim item As String
        Dim arl As New ArrayList()

        'get items and put in ArrayList
        For i = 0 To lst.SelectedIndices.count - 1
            item = lst.Items( _
                lst.SelectedIndices(i)).toString
            arl.add(item)
        Next i
        Return arl    'return the ArrayList
    End Function
    '-----
    'clear all selected items
    Public Sub clear() Implements MultiChoice.clear
        lst.Items.clear()
    End Sub
    '-----
    'return the constructed panel
    Public Function getWindow() As Panel _
            Implements MultiChoice.getWindow
        Return pnl
    End Function
```

13.6 在列表框中使用下标集合

VB7 **的列表框中不限制所列内容。当你在下标集合中加入数据时，它可能是带有** toString **方法的任何对象。在** VB6 **中对列表框的** Itemdata **属性有更多的限制。**

因为创建了 3 **个带有** toString **方法的** Equities **类，所以在主程序的构造函数中可以直接把它们加入到列表框内。**

```
Public Class wBuilder
    Inherits System.Windows.Forms.Form
    Private Pnl As Panel
    Private mchoice As MultiChoice
    Private eq As Equities
    '-----
    Private Sub init()
        lsEqTypes.Items.Add(New Stocks())
        lsEqTypes.Items.Add(New Bonds())
        lsEqTypes.Items.Add(New Mutuals())
    End Sub
```

无论何时点击列表框中的一行，都会返回给 MultiChoice **类工厂一个** Equities **类的实例，该类工厂依次产生** 3 **个显示板，其中包括所选投资项目中内容，而显示板会依次更新。**

```
Private Sub lsEqTypes_SelectedIndexChanged( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles lsEqTypes.SelectedIndexChanged

    Dim i As Integer
    i = lsEqTypes.SelectedIndex

    'get the Equity from the list box
    eq = CType(lsEqTypes.Items(i), Equities)

    'get the right Builder
    mchoice = StockFactory.getBuilder(eq)

    'remove the old panel
    Me.Controls.Remove(Pnl)

    'get the new one and add it to the window
    Pnl = mchoice.getWindow
    setPanel()
End Sub
```

13.6.1　最终选择

现在已经创建了所有程序运行所需要的类，可以运行程序了。在程序启动时，在屏幕右侧有一个空白的显示板，所以将总有移动的显示板，每次我们点击一个 Equities **名字，显示板被移动，并且一个新的名字被加入。如图** 13-5 **所示。**

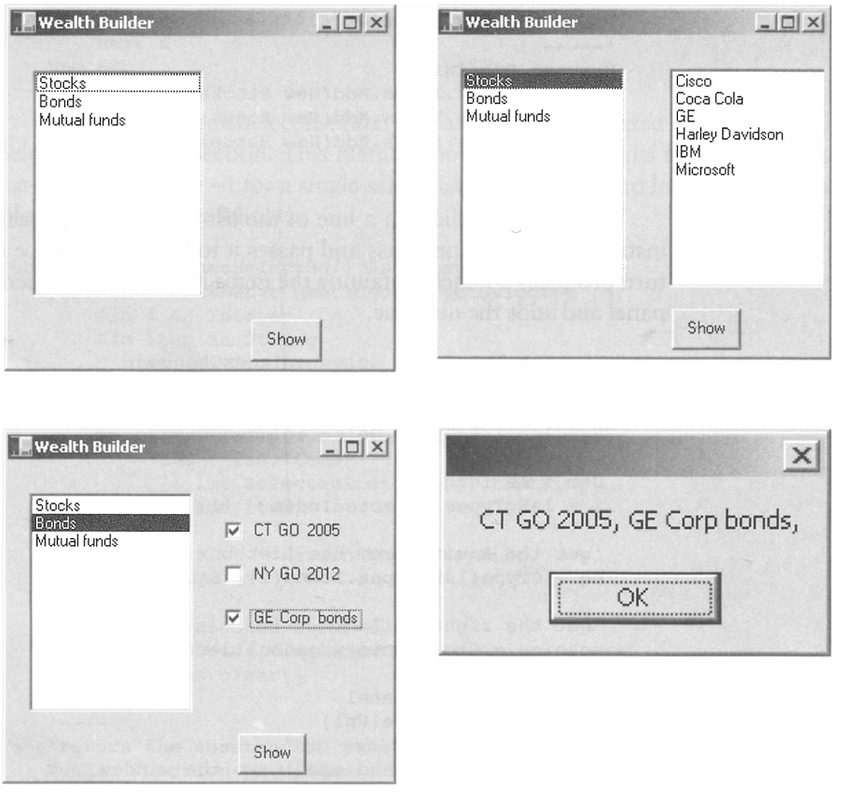


图 13-5 VB7 实现的 Wealth Builder 程序

用 UML 图表示的类之间的关系如图 13-6 所示。

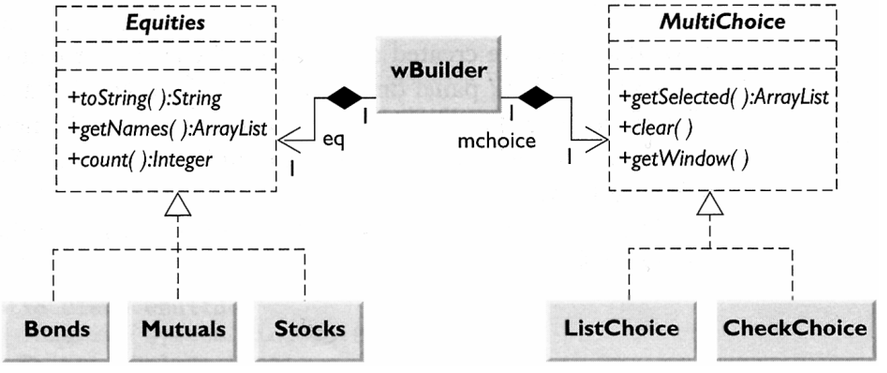


图 13-6 Builder 模式的继承关系

13.7　小结

1. 一个构造器提供了改变内部表现形式的实现方法，并隐藏了实现细节。
2. 每一个构造器之间和程序的其他部分都是独立的，这不但实现了程序的模块化，也使得添加其他构造器相对简化。
3. 由于每一个构造器所构造的内容是根据数据逐步实现的，所以你可以对构造过程进行控制。

构造器模式有些像抽象工厂模式，它们都生成由方法和对象组成的类。它们主要的区别是抽象工厂返回相关的类，而构造器是根据显示的内容实现一个复杂的对象。

思考题

1. 一些字处理软件和图形处理软件，根据所显示内容的上下文动态改变系统菜单。这种情况下如何有效地使用构造器呢？
2. 并不是所有的构造器都产生可视化的对象。如何用构造器构造一个个人的投资公司？假设你正在为一个会议记录，需要记录 5 个或 6 个不同的事件。你如何使用构造器？

13.8　光盘中的程序

\Builders\SimpleBuilder	VB6 实现的基础 Equities 构造器
\Builders\VBNetBuilder	VB7 实现的 Equities 构造器

第 14 章　原型模式

原型模式是另一种创建类工具。与其他类型的创建类相似，它首先创建一个一般类，而后在程序运行时，再创建具体的类。这一点与构造器模式十分相似，不同的是所产生的结果类是通过克隆一个或更多的原型类，然后再根据要求填充细节。

原型模式在你需要只有处理内容不同的类时被使用。例如，解析以不同基数表示数字的字数串等。在这种意义上，它与 Coplien（1992）描述的 Examplar 模式相似。

让我们考虑一个数据库应用的示例。应用中需要根据查询构造答案，在查询以一个表或 RecordSet 形成查询结果后，其他的查询可以通过以上查询结果间接地导出。

基于上述假设，我们将考虑一个游泳俱乐部中运动员的数据库。每一个运动员在一个赛季都有参赛的时间和游程的记录，并且最好的成绩按年龄分组。这样就会出现一个问题，在一个赛季（四个月）中所有的运动员都可能随着年龄的增长，而需要调整分组。所以一个查找小组中成绩最好的运动员的查询，取决于分组时的程序和运动员的生日。如果希望保持数据的最新状态，需要的代价会十分高。

一旦我们对数据库中记录按性别排序，而又希望按运动员的成绩和年龄检索，其代价是可以想象的。我们不想改变原有数据字段的次序，且重新计算这些数据意义不大，所以拷贝数据对象是一种理想的选择。

14.1　VB6 中的克隆

克隆一个类，即完全拷贝，不是 VB6 的内嵌设计特性。但在 VB6 中你也可以实现这样的拷贝。VB 中的 Clone 方法仅在数据库操纵中使用。如在应用中形成了一个查询结果集 Recordset，并一次遍历一个记录。如果你需要操作两个当前记录，一个简单的方法是使用 VB6 中的 Clone 方法，拷贝 Recordset 记录集。

```
Public Sub cloneRec(Query As String)
 Dim db As Database
 Dim rec As Recordset, crec As Recordset

 'open database recordset
 Set rec = db.OpenRecordset(Query, dbOpenDynaset)

 'clone a copy
 Set crec = rec.Clone
End Sub
```

这种方法并没有产生 Recordset 的两个拷贝，它仅产生了两个行指针的集合。这样就可以相互独立地遍历记录集。对 Recordset 的一个克隆的改变，将会在另一个克隆中立刻发生相应的变化，这是因为它们实际上操作的是一个数据表。在下面的示例中，我们讨论一个相似的问题。

14.2　使用原型

现在编写一个简单的程序，该程序从数据库中读数据，并克隆结果对象。在这个示例程序中，我们还是从文件中读入数据，而这些数据库来自于大型数据库。就像我们在前面讨论的那样，文件的格式如下：

```
Kristen Frost, 9, CAT, 26.31, F
Kimberly Watcke, 10, CDEV,27.37, F
Jaclyn Carey, 10, ARAC, 27.53, F
Megan Crapster, 10, LEHY, 27.68, F
```

我们将使用 vbFile 类编程。

首先，创建一个称为 Swimmer 的类，类中定义了运动员的名字、俱乐部名称、运动员性别和成绩变量，并使用 File 类读入这些变量。

```
Option Explicit
'Class Swimmer
Private ssex As String
Private sage As Integer
Private stime As Single
Private sclub As String
Private sfrname As String, slname As String
'-----
Public Sub init(Fl As vbFile)
 Dim i As Integer
 Dim nm As String

 nm = Fl.readToken    'read in name
 i = InStr(nm, " ")
 If i > 0 Then        'separate into first and last
   sfrname = Left(nm, i - 1)
   slname = Right$(nm, Len(nm) - i)
 Else
   sfrname = ""
   slname = nm        'or just use one
 End If

 sage = Val(Fl.readToken)     'get age
 sclub = Fl.readToken         'get club
 stime = Val(Fl.readToken)    'get time
 ssex = Fl.readToken          'get sex
End Sub
```

```
'-----
Public Function getTime() As Single
 getTime = stime
End Function
'-----
Public Function getSex() As String
 getSex = ssex
End Function
'-----
Public Function getName() As String
 getName = sfrname & " " & slname
End Function
'-----
Public Function getClub() As String
 getClub = sclub
End Function
'-----
Public Function getAge() As Integer
 getAge = sage
End Function
```

在 SwimData **类中提供了** getSwimmer **方法，在** Swimmer **类中提供** getName、getAge **和** getTime **方法。在把数据读入到** SwimInfo **类之后，就可以在列表框中显示这些数据。**

接下来创建一个称为 SwimData **的接口类，在该类中维护一个从数据库中读入的运动员集合。**

```
'Interface SwimData
Public Sub init(Filename As String)
End Sub
'-----
Public Sub Clone(swd As SwimData)
End Sub
'-----
Public Sub setData(swcol As Collection)
End Sub
'-----
Public Sub sort()
End Sub
'-----
Public Sub MoveFirst()
End Sub
'-----
Public Function hasMoreElements() As Boolean
End Function
'-----
Public Function getNextSwimmer() As Swimmer
End Function
```

当用户点击 Clone **按钮时，就开始克隆一个类，并在新类中对数据进行排序。由于产生一个新的类实例会很慢，所以我们克隆数据，但我们还是希望保持两种形式的数据。**

```
Private Sub SwimData_Clone(swd As SwimData)
    swd.setData swimmers    'copy data into new class
End Sub
```

在原类中，运动员的名字依次按性别和成绩排序，而在克隆类中它们仅按成绩排序。在图 14-1 中，我们看到一简单的用户界面，该界面允许我们在左侧显示原始数据，在右侧的克隆类中存放数据。

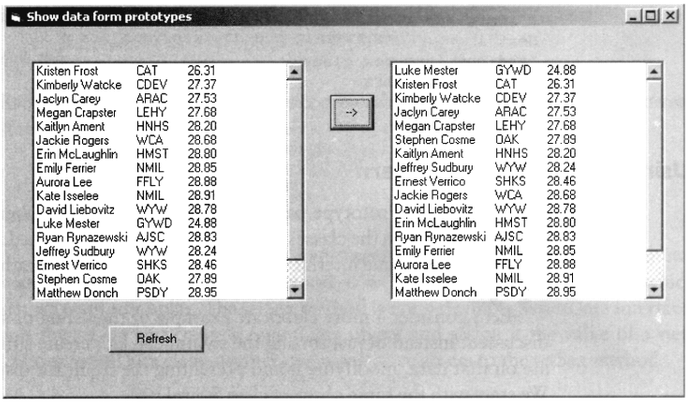


图 14-1　原型模式示例，左边列表框是程序最初的数据，右边是点击 Clone 后的数据

现在点击 Refresh 按钮，重新从数据库中调入左边的列表框的数据。其结果略微有一点不同，如图 14-2 所示。

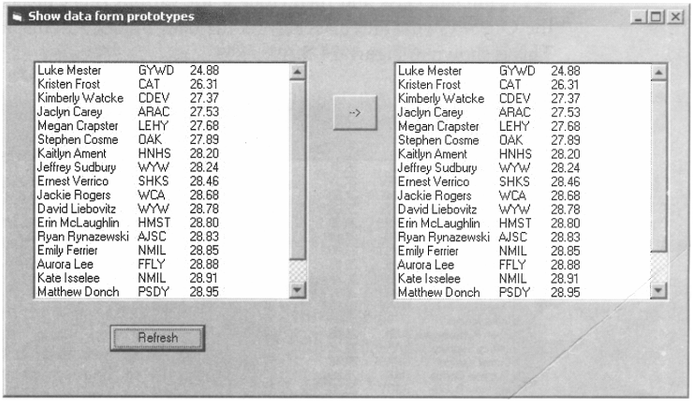


图 14-2　原型模式示例，点击 Clone 和 Refresh 后的显示结果

为什么左边的列表框中的人名会被重新排序？这是因为克隆是一种对原始类的“浅”拷贝，也就是说，数据对象的地址被拷贝，但是它们共享一个基础数据对象。这样对克隆数据的操作也会作用到原型数据。

在一些情况下，“浅”拷贝是可以接受的，但如果想创建一个数据的“深”拷贝，你必须自己编写一个“深”拷贝程序，该程序作为你想克隆的类的一部分。在简单的类中，你可以定义一个集合，并拷贝一个老集合中的数据到新集合中。

```
Private Sub SwimData_Clone(swd As SwimData)
 Dim swmrs As New Collection
 Dim i As Integer
 'copy data from one collection
 ' to another
 For i = 1 To swimmers.Count
   swmrs.Add swimmers(i)
 Next i
 'and put into new class
 swd.setData swmrs
End Sub
```

14.3 使用原型模式

无论在什么情况下，当需要生成多个类或对已经生成的类进行修改时，你就可以使用原型模式。在使用原型模式时，只要所有的类有相同的接口，它们就可以在同一接口下执行不同的操作。

让我们重新考虑曾经讨论过的一个示例，在这个示例中列出了所有游泳运动员的名单。这里采用的方法不是对运动员排序后再显示，而是建立一个子类，在子类中对数据进行必要的处理，并在列表框中显示运动员的名单。这里还是使用抽象类 SwimData 作为父类。

接下来就是根据不同的需求，编写实现具体 SwimData 类的程序，这里还是使用 SexSwimData 基类，然后为各种其他显示克隆此类。例如，程序中的 OneSexSwimData 类对数据按性别进行重新排序并显示，如图 14-3 所示。

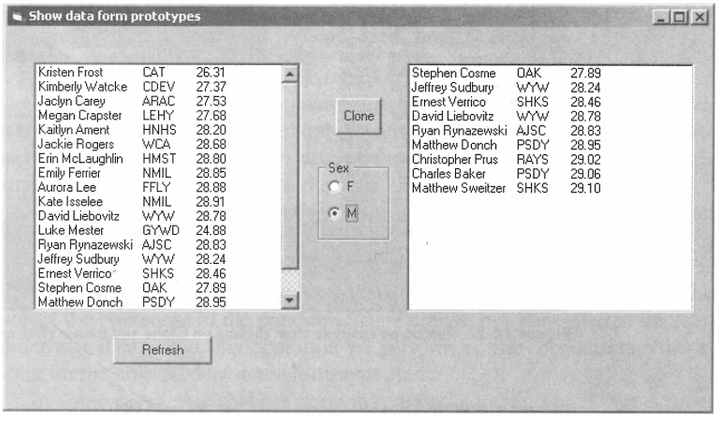


图 14-3 OneSexSwimData 类在右侧仅显示一个性别

在 OneSexSwimData 类中，将数据按运动员的成绩排列，然后再按所选择的运动员的性别显示数据。相应的方法如下：

```
Public Sub setSex(sx As String)
  Sex = sx 'copy current sex preference
End Sub
```

当选择一个性别按钮后，这个类就按当前的状态显示数据。

14.3.1　在子类中增加方法

OneSexSwimData 类实现了 SwimData 接口，但还需要在该类中增加一个新的方法，该方法允许我们按指定的运动员性别显示数据。setSex 方法就是实现该功能的新方法，它并不是 SwimData 接口中的方法。这样，如果创建一个 SwimData 类型的变量，并把一个 OneSexSwimData 的新实例分给该变量（对象），在这个对象中将不能访问 setSex 方法。

```
Private swd As SwimData
Private tsd As SwimData
'------
Private Sub Clone_Click()
    Set tsd = New OneSexSwimData
    swd.Clone tsd    'clone into any type
    tsd.sort         'call interface method
```

另一方面，如果我们创建另一个 OneSexSwimData 类的实例，在这个实例中将不能访问 SwimData 接口中的方法。

```
Private swd As SwimData
Private osd As OneSexSwimData
'------
Private Sub Clone_Click()
Set osd = tsd    'copy to specific type
```

我们可以通过为每一个类型产生一个变量，并且使用 SwimData 和 OneSexSwimData 变量来访问相同类的方法，解决这个问题。

```
Private swd As SwimData
Private tsd As SwimData
Private osd As OneSexSwimData
'------
Private Sub Clone_Click()
    Set tsd = New OneSexSwimData
    swd.Clone tsd    'clone into any type
    tsd.sort         'call interface method

    Set osd = tsd    'copy to specific type
    osd.setSex "F"   'call derived class method
    SexFrame.Enabled = True 'enable sex selection
    loadRightList
End Sub
```

这里只有在对象变量拷贝执行后，SexFrame 窗体中才包括 F 和 M 选择项。这可以有效地防止还没有进行初始化就执行 setSex 方法。

```
Private Sub Sex_Click(Index As Integer)
'sets the sex of the class to either F or M
  osd.setSex Sex(Index).Caption
  loadRightList
End Sub
```

14.3.2 具有相同接口的不同类

类当然可以是各具特色的，AgeSwimData 类按所获得的数据排列，按年龄显示直方图。如果选择 F 单选按钮，就可以看到女运动员的分布情况；如果选择 M 单选按钮，就可以看到男运动员的分布情况，如图 14-4 所示。

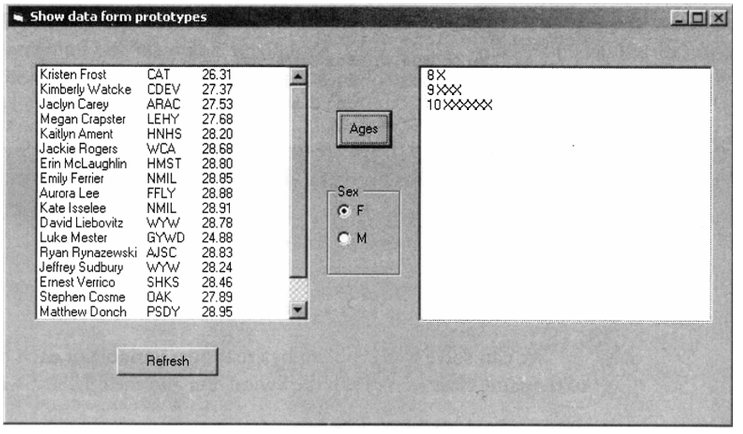


图 14-4 AgeSwimData 类显示年龄分布

十分有趣的是，AgeSwimData 类使用了 SwimData 类的所有接口方法，同时又使用了 OneSexSwimData 类中的 setSex 方法。我们在 AgeSwimData 类中把 setSex 方法设定为 public 型，或者我们声明 AgeSwimData 实现上面的两种接口。

```
'Class AgeSwimData
Implements OneSexSwimData
Implements SwimData
```

在这种情况下，在它们之间几乎无法选择，这是因为仅有一个额外的方法，setSex 存在于 OneSexSwimData 类中。但在数据的显示方面却有很大的不同。

```
Private Sub SwimData_sort()
 Dim i As Integer, j As Integer
 Dim sw As Swimmer, age As Integer
 Dim ageString As String
```

```
 'Sort the data inbto increasing age order
 max = swimmers.Count
 ReDim sws(max) As Swimmer
 'copy the data into an array
 For i = 1 To max
   Set sws(i) = swimmers(i)
 Next i
 'sort by increasing age
 For i = 1 To max
   For j = i To max
     If sws(i).getAge > sws(j).getAge Then
       Set sw = sws(i)
       Set sws(i) = sws(j)
       Set sws(j) = sw
     End If
   Next j
 Next i
 'empty the collection
 For i = max To 1 Step -1
   swimmers.Remove i
 Next i
 'fill it with the sorted data
 For i = 1 To max
   swimmers.Add sws(i)
 Next i
 'create the histogram
 countAgeSex
End Sub
'----
Private Sub countAgeSex()
 Dim i As Integer, j As Integer
 Dim sw As Swimmer, age As Integer
 Dim ageString As String

 'now count number in each age
 Set ageList = New Collection
 age = swimmers(1).getAge
 ageString = ""
 i = 1
 While i <= max
   'add to histogram if in age and sex
   If age = swimmers(i).getAge And Sex = swimmers(i).getSex Then
     ageString = ageString & "X"
   End If
   If age <> swimmers(i).getAge And Sex = swimmers(i).getSex Then
     'create new swimmer if age changes
      Set sw = New Swimmer
      sw.setFirst Str$(age) 'put string of age in 1st name
      sw.setLast ageString  'put histogram in last name
      ageList.Add sw        'add to collection
      age = swimmers(i).getAge
      ageString = "X"       'start new age histogram
   End If
   i = i + 1
```

```
 Wend
 'copy last one in
  Set sw = New Swimmer
  sw.setFirst Str$(age)
  sw.setLast ageString
  ageList.Add sw
  amax = ageList.Count

End Sub
```

最初的类显示了选择的运动员的名和姓。现在我们实现了相同的显示，返回一个带有多个名和姓的 Swimmer 对象。

图 14-5 的 UML 图清晰地解释了原型模式所构成的系统的内部关系。其中，SwimInfo 类是主 GUI 类，它包含了两个 SwimData 类的实例，但没有指定用哪一个。TimeSwimData 类和 SexSwimData 类是由抽象类 SwimData 导出的两个具体的类。AgeSwimData 类是由 SexSwimData 导出的类，它用直方图显示运动员的名单。

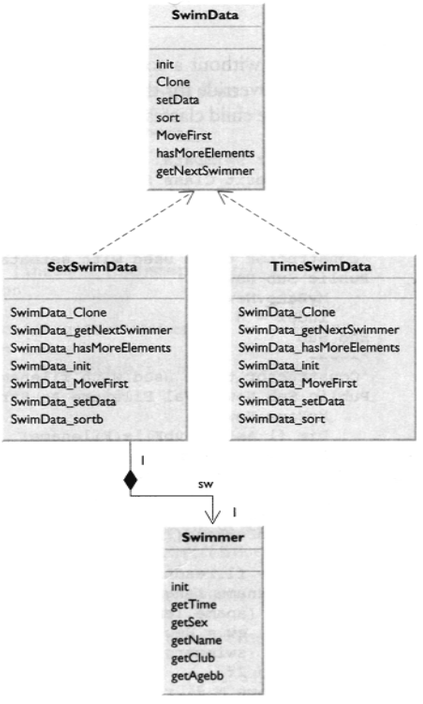


图 14-5 各种 SwimData 类的 UML 图

在本例中没有限制创建具体类的个数。用适当的代码实现一个具体的类并注册该类是十分简单的事情。另一个需要提醒你的是，用户是程序的决策点（deciding point）或 factory，哪一个按钮被选择，是用户来确定的。在下面的内容中，你会见到另一个示例，我们称之

为原型管理器（prototype manager）。这是一个实现对类管理的程序，它可以和工厂模式相结合，创建和管理不同的类。

14.4　原型管理器

一个原型管理器类可以用来确定哪几个具体的类返回给用户，它同时管理几个原型的集合。例如，除了返回几个游泳运动员类中的一个类外，它还可以返回不同游泳项目和距离的运动员的分组。同时还管理哪几种列表框被返回，这些列表框用于显示运动员数据，表格包括单栏表、多栏表和图形显示。原型管理器适合于确定类的返回需求，但不适合于产生新类的程序。也就是说，在原型管理器中，父类或抽象类中包含有多个方法的定义，而客户并不需要知道哪个子类需要被处理。

14.5　用 VB7 编写原型

VB7 可以用相似的代码实现一个原型模式。不同的是使用 ArrayLists 和 zero-based 数组，并且建立了一个 SwimData 类，在此基础上可以继承其中有用的方法。在 SwimData 类中没有定义 sort 方法，并指定 MustInherit 用于类，MustOverride 用于方法，而且该方法你必须在子类中实现。

```
'Base class for SwimData
Public MustInherit Class SwimData
    Protected Swimmers As ArrayList
    Private index As Integer
    '-------
    'constructor to be used with setData
    Public Sub New()
        MyBase.New()
        index = 0
    End Sub
    '-----
    'Constructor to be used with filename
    Public Sub New(ByVal Filename As String)
        MyBase.New()
        Dim fl As New vbFile(Filename)
        Dim sw As Swimmer
        Dim sname As String

        swimmers = New ArrayList()
        Fl.OpenForRead(Filename)

        sname = fl.readLine
        While sname.length > 0
            If (sname.length > 0) Then
                sw = New Swimmer(sname)
```

```
                swimmers.Add(sw)
            End If
            sname = fl.readLine
        End While
        sort()
        index = 0
    End Sub
    '-------
    Public Sub setData(ByVal swcol As ArrayList)
        swimmers = swcol
        movefirst()
    End Sub
    '-------
    'Clone dataset from other swimdata object
    Public Sub Clone(ByVal swd As SwimData)
        Dim swmrs As New ArrayList()
        Dim i As Integer
        'copy data from one collection
        ' to another
        For i = 0 To swimmers.Count - 1
            swmrs.Add(swimmers(i))
        Next i
        'and put into new class
        swd.setData(swmrs)
    End Sub
    '------
    'sorting method must be specified
    'in the child classes
    Public MustOverride Sub sort()
    '------
    Public Sub MoveFirst()
        index = -1
    End Sub
    '------
    Public Function hasMoreElements() As Boolean
        Return (index < (Swimmers.count - 1))
    End Function
    '------
    Public Function getNextSwimmer() As Swimmer
        index = index + 1
        Return CType(swimmers(index), Swimmer)
    End Function

End Class
```

这里我们仍然使用前面的 vbFile **类从文件中读一行数据。当数据读入后，**Swimmer **类中的相应方法就可以进行解析。在** VB7 **中采用不同的数据转换方法。我们使用** CInt **函数转换整型数，而不是使用** Val **函数。**

```
sage = CInt(tok.nextToken)  'get age
```

我们使用 toSinge **方法转换时间值。**

```
stime = CSng(tok.nextToken)      'get time
```

下面是一个完整的 Swimmer **类的构造函数。**

```
Public Class Swimmer
    Private ssex As String
    Private sage As Integer
    Private stime As Single
    Private sclub As String
    Private sfrname, slname As String
    '-----
    Public Sub New(ByVal nm As String)
        MyBase.New()
        Dim i As Integer
        Dim s As String
        Dim t As Single
        Dim tok As StringTokenizer

        tok = New StringTokenizer(nm, ",")
        nm = tok.nextToken
        i = nm.indexOf(" ")
        If i > 0 Then        'separate into first and last
            sfrname = nm.substring(0, i)
            slname = nm.substring(i + 1)
        Else
            sfrname = ""
            slname = nm       'or just use one
        End If
        sage = CInt(tok.nextToken)       'get age
        sclub = tok.nextToken            'get club
        stime = CSng(tok.nextToken)      'get time
        ssex = tok.nextToken             'get sex
    End Sub
```

最后运行 VB7 **实现的原型模式程序如图** 14-6 **所示。**

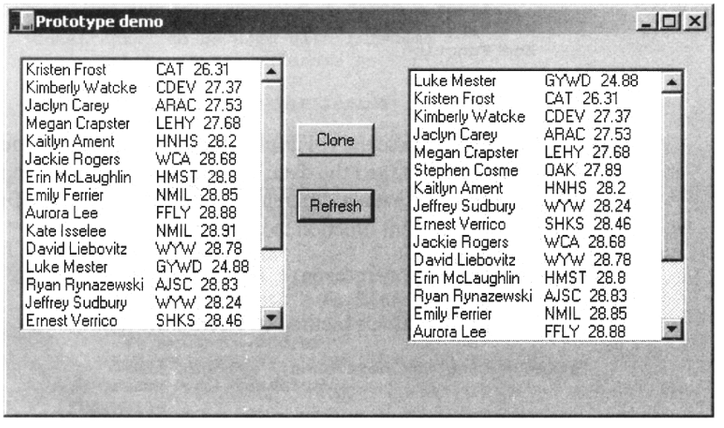


图 14-6　**点击** Clone **按钮后** VB7 **实现的原型模式**

TimeSwimData 类十分简单，它仅由 New 方法和 sort 方法构成。

```
Public Class TimeSwimData
    Inherits SwimData
    '----------
    Public Sub New(ByVal filename As String)
        MyBase.New(filename)
    End Sub
    '----------
    Public Sub New()
        MyBase.New()
    End Sub
    '----------
    'Required sort method
    Public Overrides Sub sort()
        Dim i, j, max As Integer
        Dim sw As Swimmer
        max = swimmers.Count
        'copy into array
        Dim sws(max) As Swimmer
        swimmers.CopyTo(sws)
        'sort by time
        For i = 0 To max - 1
            For j = i To max - 1
                If sws(i).getTime > sws(j).getTime Then
                    sw = sws(i)
                    sws(i) = sws(j)
                    sws(j) = sw
                End If
            Next j
        Next i
        'copy back into new ArrayList
        swimmers = New Arraylist()
        For i = 0 To max - 1
            swimmers.Add(sws(i))
        Next i
    End Sub

End Class
```

14.6 小结

使用原型模式，在程序运行过程中通过克隆的方法可以增加和删除一个类；可以在运行期间根据条件改变类的内部数据的表现形式；还可以说明新的对象，而不用增加新的类定义。

用 VB 实现原型模式比较困难的一个方面是，当一个类已经存在时，你就不可能增加新的克隆方法来改变原有的类。另外，一个被其他类循环引用的类不能被克隆。

像在单一类中的注册一样，你可以创建一个原型类的注册，它可以为一些可能的原型

列表克隆和检索注册对象。你也可以从一个已有的类克隆一个新类，而不是重新编写它。

每一个你想用作原型的类必须被实例化，以便为了使用 Prototype Registy。这可能是一种性能退化。

总而言之，原型类的思想是建立一个可使用的克隆的类，其中有丰富的数据和方法可供其他类进行修改。这意味着你必须在原型类中增加许多方法，以便其他类拷贝并修改。

思考题

在一个标题游戏中，程序在不同的地方，不同的时间，用不同的字体和大小显示一个标题。试用原型模式设计一个程序实现上述功能。

14.7　光盘中的程序

\Prototype\Ageplot	VB6 age plot
\Prototype\DeepProto	VB6 deep prototype
\Prototype\OneSex	VB6 display by sex
\Prototype\SimpleProto	VB6 shallow copy
\Prototype\TwoclassAgePlot	VB6 age and sex display
\Prototype\VBNet\DeepProt	VB6 deep prototype

14.8　生成模式总结

- 工厂模式被用于基于你提供给工厂的数据，从许多相似的类中选择并返回一个类的实例。
- 抽象工厂模式用于返回几组类，它实际上为每一个组返回一个工厂。
- 构造模式基于数据重新把一些对象组合成为一个新的对象，通常使用工厂模式来组合这组对象。
- 原型模式拷贝或克隆一个已经存在的类，而不是创建一个代价昂贵的新类。
- 单一模式是一个这样的模式，它有且仅有一个对象实例，并且该对象实例是全局可访问的。

第III部分

结 构 模 式

结构模式讨论如何根据类和对象构造一个大的结构。类模式和对象模式有所不同，类模式是描述如何使用类的继承性提供有效的程序接口；而对象模式描述如何通过对象之间的组合构成更大的结构，还描述了其他对象中包含的对象。

在本部分中将结合实例讨论结构模式的设计。示例有 Adapter 模式、Composite 模式、Proxy 模式、Flyweight 模式、Façade 模式、Bridge 模式、Decorator 模式。

适配器（Adapter）模式可以用来实现不同类之间的接口，以便使编程更容易。我们将讨论其他结构模式。在这些模式中通过组合不同的对象来提供新的功能。

组合（Composite）模式通过对象构造增加的功能，它可以组合简单的对象，也可以组合构造（由简单对象组合而成的对象）对象。

代理（Proxy）模式是一种用简单模式代替复杂模式的示例，在网络环境中经常使用代理模式。

轻量（Flyweight）模式是一种共享对象的模式。该模式中每个实例不包含它自己的声明，而另外存放这些声明。在 Flyweight 模式中，所有对象实例共享一个公共的数据源。当一些相同类型的多个实例共享一个数据源时，使用该模式可以有效地节省存储空间。

伪（Façade）模式是利用一个类实现一个完整的子系统。

桥（Bridge）模式是把对象的接口与其实现相分离的模式，这种分离可以对接口和实现分别修改。

修饰（Decorator）模式，该模式运行动态的增加对象的功能。

在讨论结构模式时一些内容会重复，在本部分的最后我们将对上面的模式进行总结。

第 15 章　适配器模式

适配器（Adapter）模式用于实现一个类程序的接口到另一个类程序接口的转换，它把两个没有关系的类放在一个程序中工作。该模式与通常意义的适配器有相同的含义。即 Adapter 是一个类，该类提供了一个多功能接口，它可以实现与不同类型接口的类进行通信。

有两种实现适配器的方法：继承和组合。第一种方法是通过从一个基类导出一个新类，并增加满足新接口的方法。第二种方法是在新类中包含一个原始类，并在新类中产生一个方法来传递调用。这两种方法分别称为类适配器和对象适配器，该方法通过对象调用来实现适配器，这在面向对象的程序设计语言中十分容易实现。在 VB7 发布之前，用 VB 来实现适配器模式时只能用第二种方法，因为在 VB7 之前的 VB 版本中还没有继承功能。

15.1　在列表中移动数据

现在分析一个简单的程序，该程序允许你把运动员的名字从一个列表移动到另一个列表中，以便细化显示内容。第一个列表包括队员的花名册，第二个列表包括运动员的名字和成绩。

在这个简单的程序中，初始化时从花名册文件中读入人名，如图 15-1 所示。为了把人名移到右边的列表框中，首先点击人名并点击右移箭头按钮。也可以把人名从右边的列框移到左边的列表框中，方法是点击人名后再点击左移箭头按钮。

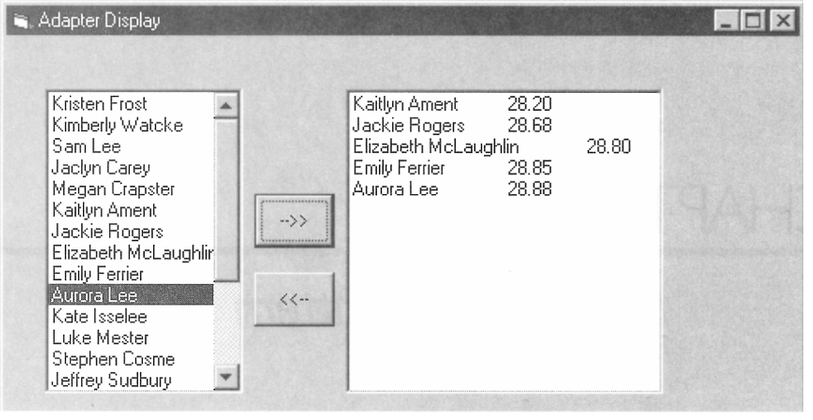


图 15-1　选择显示人名的简单程序

这是一个十分容易实现的 VB 程序，该程序包含可视化的窗口和点击按钮后的执行程序。但把人名从花名册中读出之后，把所有运动员的名字和成绩存储在 Swimmer 对象中，并把所有的对象存储在集合中。当你在左边的列表中选择了一个运动员，你就可以获得该运动员的数据，这些数据就是要显示到右边列表框的数据。

```
Private Sub Moveit_Click()
Dim i As Integer
i = lsKids.ListIndex + 1
 If i > 0 And i <= swmrs.Count Then
    Set sw = swmrs(i)
    lsTimes.AddItem sw.getName & vbTab & str$(sw.getTime)
 End If
End Sub
```

同样，你可以使用相似的方法把人名从右边列表框移到左边的列表框。

```
Private Sub putback_Click()
 Dim i As Integer
 i = lsTimes.ListIndex
 If i >= 0 Then
   lsTimes.RemoveItem i
 End If
End Sub
```

不同列的排列是用 tab 键（制表位）来获得段前空格，只要人名的长度基本相同，该方法简单有效。但是在一个人名比其他人名长得较多或短得较多时，使用制表位就不太适宜，如第 3 行。

15.2 使用 MSFlexGrid

为了更好地解决这个问题，我们可以使用网格显示。VB 中一种简单的网格称为 MSFlexGrid。它是可以从第三方供应商购买到的控件集合的一个子集。MSFlexGrid 控件有行和列属性，你可以用这些属性来获得当前的表格大小。当然你也可以用这些属性来改变表格的行和列，也可以用文本属性来改变表格中的文本。

```
Private Sub Movetogrid_Click()
Dim i As Integer, row As Integer

i = lsKids.ListIndex + 1
 If i > 0 And i <= swmrs.Count Then
    Set sw = swmrs(i)
    grdTimes.AddItem ""
    row = grdTimes.Rows
    grdTimes.row = row - 1
    grdTimes.Col = 0
    grdTimes.Text = sw.getName
    grdTimes.Col = 1
```

```
    grdTimes.Text = Str$(sw.getTime)
 End If
End Sub
```

然而，我们希望在使用网格时不改变原来处理列表框的程序代码。显然，我们可以做到这一点，这是因为 MSFlexGrid 中的 AddItem 方法采用与列表框相似的风格处理制表位。

下面的语句

```
grdTimes.AddItem sw.getName & vbTab & Str$(sw.getTime)
```

与前面示例中的第 7 行代码是等价的，并且把运动员的人名显示在第 1 列，而成绩显示在第 2 列上，如图 15-2 所示。

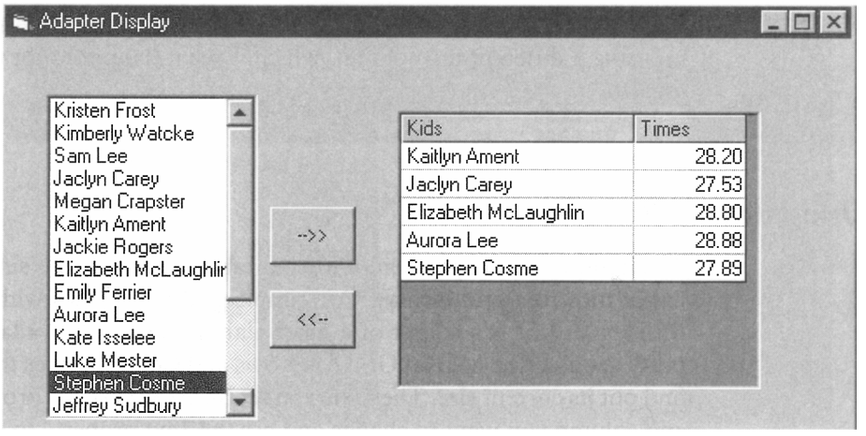


图 15-2　在 MSFlexGrid 控件中选择显示用的人名

从另一个角度，MSFlexGrid 控件提供一种编程接口，它是它本身到列表框的适配器。

事实上，因为列表框和网格有相同的编程接口，所以编写一个加入数据的子程序是十分容易的。

```
Private Sub addText(ctl As Control, sw As Swimmer)
    ctl.AddItem sw.getName & vbTab & str$(sw.getTime)
End Sub
```

然后编写一个按钮点击的处理程序，以便在每一次调用时使用不同的列表作为参数。

```
Private Sub Moveit_Click()
 Dim i As Integer
 i = lsKids.ListIndex + 1
 If i > 0 And i <= swmrs.Count Then
    Set sw = swmrs(i)
```

```
    addText lsTimes, sw
 End If
End Sub
'------
Private Sub Movetogrid_Click()
 Dim i As Integer, row As Integer
 i = lsKids.ListIndex + 1
 If i > 0 And i <= swmrs.Count Then
    Set sw = swmrs(i)
    addText grdTimes, sw
 End If
End Sub
```

但是上面的方法不是完全的面向对象的。实际上 addText **方法是所用的类的一部分，我们不必把一个列表或网格实例传递到同一个类的方法中。在** VB6 **以及更低的版本中，无法在控件中加入一个方法。现在编写一个简单的** ControlAdapter **类来处理网格和列表，并且增加一个用子程序实现的** addText **方法。该类的程序如下：**

```
'Class ControlAdapter
Private ctrl As Control

Public Sub init(ctl As Control)
 Set ctrl = ctl  'copy control into class
End Sub
'-----
Public Sub addText(sw As Swimmer)
'add new line to list or grid
 ctrl.AddItem sw.getName & vbTab & str$(sw.getTime)
End Sub
```

我们在 Form_Load **事件中对类中的列表和网格进行初始化：**

```
Private grdAdapt As New ControlAdapter

'pass grid into Control Adapter
grdAdapt.init grdTimes
```

然后就可以点击右箭头键来调用该类的 addText **方法，不用考虑使用哪种显示控件。**

```
Private Sub Moveit_Click()
 Dim i As Integer
 i = lsKids.ListIndex + 1
 If i > 0 And i <= swmrs.Count Then
    Set sw = swmrs(i)
    grdAdapt.addText sw
 End If
End Sub
```

15.3　使用 TreeView

如果使用 TreeView 控件显示选择的数据，你就会发现一个不改变程序结构的适配器接口是不存在的。这样 ControlAdapter 类就无法适用于 TreeView 的显示方式，我们必须编写一个新的 TreeAdapter 类，在不改变接口的情况下实现在显示窗口中的树型结构中加入一个横线。

TreeView 类包含一个 Nodes 集合，你可以通过加入节点、建立节点的文本和定义是否是一个子节点等方法加入数据。子节点与父节点是有层次关系的，下面是一个先加入父节点后再加入子节点的程序。

```
'Class TreeAdapter
Private Tree As TreeView
Public Sub init(tr As TreeView)
  Set Tree = tr
End Sub
Public Sub addText(sw As Swimmer)
Dim scnt As String, nod As Node
    scnt = Str$(Tree.Nodes.Count)
    Set nod = Tree.Nodes.Add(, tvwNext, "r" & _
        sw.getName, sw.getName)
    Tree.Nodes.Add "r" & sw.getName, tvwChild, , Str$(sw.getTime)
    nod.Expanded = True
End Sub
```

我们用图 15-3 来解释 TreeView 程序。

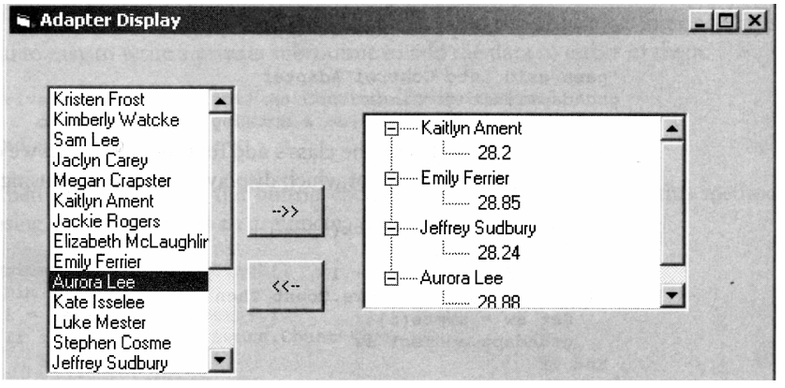


图 15-3　TreeView 适配器程序

15.3.1　对象适配器

在对象适配器方法（如图 15-4 所示）中，创建一个包含列表框（ListBox）的类，该类实现 ControlAdapter 接口中的方法。我们在前面的示例中也采用这个方法。

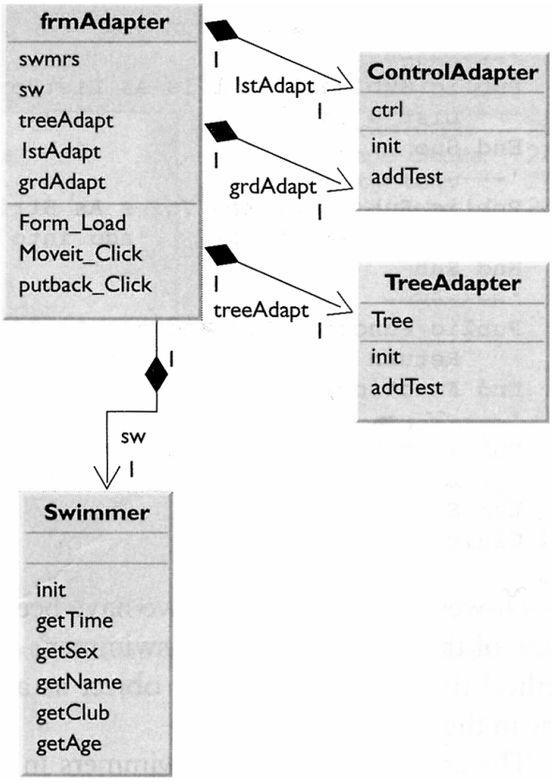


图 15-4 一个实现列表适配器的对象适配器

15.4 在 VB7 中使用适配器

用 VB7 实现适配器更适合一些。首先让我们考虑 VB7 下的 ListBox 控件，它与 VB6 中的 ListBox 控件拥有截然不同的方法，但是我们可以很好地屏蔽这些不同之处。

使用 VB7，你可以通过在列表框的 Items 集合中加入一个字符串的方法，在 ListBox 中增加一行。

```
list.Items.Add(s)
```

其中 ListIndex 属性被 ListBox 方法的 SelectedIndex 属性取代。所有可以容易地编写一个简单的包装类，该类实现上述方法到 VB7 中的 ListBox 方法的转换。ListAdaper 类中的构造函数中包含并存储一个 ListBox 实例，所采用的方法是封装或对象组合。

```
Public Class ListAdapter
    'Adapter for ListBox emulating some of
    'the methods of the VB6 list box.
    Private List As ListBox          'instance of list box
    '--------
    Public Sub New(ByVal ls As ListBox)
        List = ls
    End Sub
    '--------
    Public Sub addItem(ByVal s As String)
        list.Items.Add(s)    'add into list box
    End Sub
    '--------
    Public Function ListIndex() As Integer
        Return list.SelectedIndex     'get list index
    End Function
    '--------
    Public Sub addText(ByVal sw As Swimmer)
        List.Items.Add(sw.getName & vbTab & sw.getTime.ToString)
    End Sub
End Class
```

然而，就像在前面的程序中讨论的那样，我们希望显示运动员的名字和成绩。用 VB7 可以方便地实现把游泳运动员对象作为参数的方法，从而把运动员的名字和成绩放入列表框中。

从文件中读入运动员的名字，并把它们放入到左边的列表框的程序仍然使用一个 ListAdaper 类的实例。下面是它的声明和初始化的代码。

```
Private lsAdapter, ksAdapter As ListAdapter

        lsAdapter = New ListAdapter(lsNames)
        ksAdapter = New ListAdapter(lsKids)
```

下面是简单的读入数据的代码。

```
Private Sub ReadFile()
        Dim s As String
        Dim sw As Swimmer
        Dim fl As New vbFile("swimmers.txt")
        fl.openForRead()
        s = fl.readLine
        While Not fl.fEof
            sw = New Swimmer(s)
            swimmers.add(sw)
            ksAdapter.addItem(sw.getName)
            s = fl.readLine
        End While
    End Sub
```

程序的执行结果如图 15-5 所示。

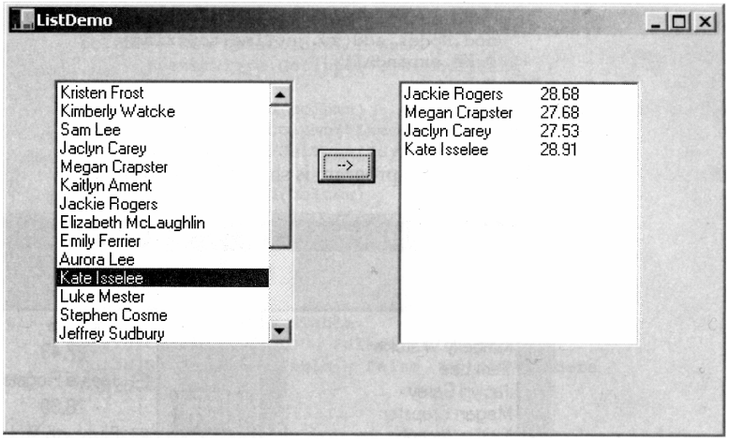


图 15-5 使用两个 ListAdaper 类的实例调入的两个列表框

15.5 VB.NET 的 TreeView 适配器

用 VB7 实现的 TreeView 类与 VB6 略有不同。用 VB7 实现时，如果想产生一个新的节点，你就可以创建一个 TreeNode 类的实例，并把该实例对应的根节点集合加入到另一个节点集合中。在我们的示例中使用 TreeView 类，并将游泳运动员的名字加入到根节点的集合，而将运动员的成绩作为子节点。下面是一个完整的 TreeAdapter 类，这里只需实现 addText 方法。

```
Public Class TreeAdapter
    'An adapter to use TreeView
    'instead of list boxes
    Private Tree As TreeView    'instance of tree

    Public Sub New(ByVal tr As TreeView)
        Tree = tr
    End Sub
    '-------------
    Public Sub addText(ByVal sw As Swimmer)
        Dim scnt As String
        Dim nod As TreeNode
        'add a root node
        nod = Tree.Nodes.add(sw.getName)
        'add a child node to it
        nod.Nodes.add(sw.getTime.toString)
        Tree.expandAll()
    End Sub

End Class
```

TreeDemo 程序对应的屏幕如图 15-6 所示。

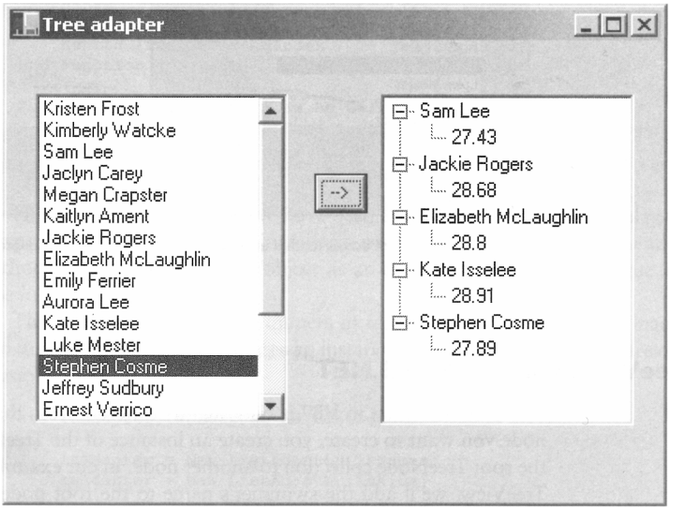


图 15-6　使用 TreeView 适配器的运动员选择程序

15.6　采用 DataGrid 控件

VB7 的 DataGrid 控件比 VB6 中 MSFlexGrid 控件更精美，它可以与数据库或一个内存数组绑定。为了在没有数据库时使用 DataGrid 控件，可以创建一个 DataTable 类的实例，并把 DataColumns 加入其中。DataColumns 的默认数据类型是字符串类型，但是你可以在创建该对象时把它定义成任何的数据类型。下面是一个展示如何使用 DataTable 创建 DataGrid 控件的程序框架。

```
dTable = New DataTable("Kids")
dTable.MinimumCapacity = 100
dTable.CaseSensitive = False
       Dim column As DataColumn
       column = New DataColumn("Frname", _
             System.Type.GetType("System.String"))

       dTable.Columns.Add(column)
       column = New DataColumn("Lname", _
             System.Type.GetType("System.String"))

       dTable.Columns.Add(column)
       column = New DataColumn("Age", _
             System.Type.GetType("System.Int16"))

       dTable.Columns.Add(column)
```

```
        DGrid.DataSource = dTable
        DGrid.CaptionVisible = False     'no caption
        DGrid.RowHeadersVisible = False 'no row headers
        DGrid.EndInit()
```

为了在 DataTable 中加入文本，你首先通过行对象询问该表，然后为该行设置行对象元素。如果所有的字段都是字符串类型，则可以进行拷贝。但如果有的字段不是字符串类型，例如是整数类型，这时你应确信 DataGrid 中的该字段也设定成同样的类型。

下面是使用上面风格的 GridAdapter 类，实现完全填充各行的程序。

```
Public Class GridAdapter
    Private dtable As DataTable
    Private Dgrid As DataGrid
    '-----
    Public Sub New(ByVal grid As DataGrid)
        dtable = CType(grid.DataSource, DataTable)
        dgrid = grid
    End Sub
    '------
    Public Sub addText(ByVal sw As Swimmer)
        Dim scnt As String
        Dim row As DataRow

        row = dtable.NewRow
        row("Frname") = sw.getFirstName
        row(1) = sw.getLastName
        row(2) = sw.getAge  'This one is an integer
        dtable.Rows.Add(row)
        dtable.AcceptChanges()
    End Sub
End Class
```

注意，你可以通过数值位置或者通过名称来访问每列。程序的运行如图 15-7 所示。

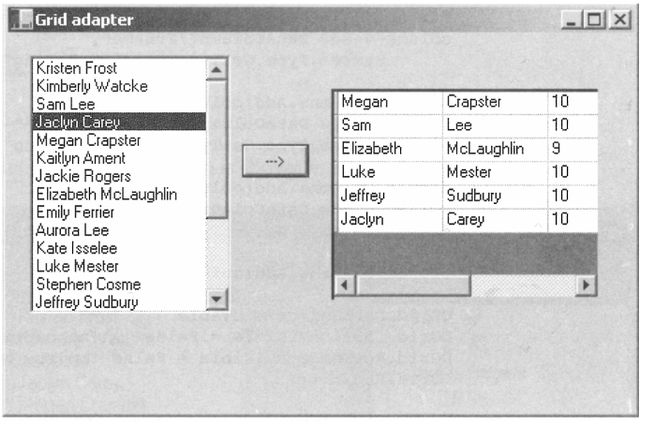


图 15-7 GridAdapter 程序

15.7　类适配器

我们可以用类适配器从 ListBox、网络或树型控件导出一个新类，并在类中增加必要的方法。这在 VB7 中是可行的，但在其他低版本的 VB 中是做不到的。在光盘的这个类适配器示例中，我们从 ListBox 类中导出一个称为 OurList 的类，该类实现一个如下接口：

```
Public Interface ListAdapter
    'Interface Adapter for ListBox emulating some of
    'the methods of the VB6 list box.
    Sub addItem(ByVal s As String)
    Function ListIndex() As Integer
    Sub addText(ByVal sw As Swimmer)
End Interface
```

所产生类的示意图如图 15-8 所示，其他代码与对象适配器相同。

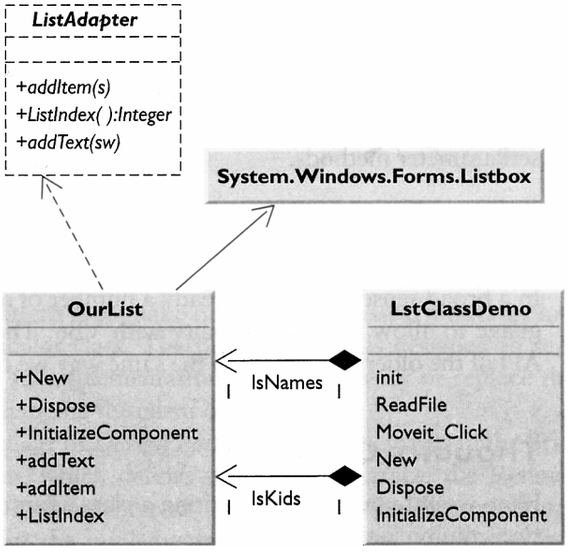


图 15-8　用类适配器实现的 List 适配器

虽然它们的区别不像在 C++中对象和类的区别那样明确。在 VB7 中类适配器与对象适配器方法还是有一些不同。

在类适配器中：

- 由于导出类是在定义时才从它的父类中导出，所以无法实现导出类与所有的子类进行适配。
- 允许类适配器改变一些被适配的类的方法，但原有的方法仍然可以使用。

一个对象适配类

- **可以通过构造函数来实现子类的适配**
- **可以把被适配对象的所有方法变成可用**

15.8 两路适配器

两路适配器是一种允许一个对象被两个不同的类观察的适配器，如一个对象可同时被 ListBox **或** MSFlexGrid **观察。由于所有基类中的方法在子类中被自动继承，所以用类适配器很容易实现两路适配器。当然这只能在子类没有重载基类的方法时有效。**

15.9 在 VB.NET 中实现对象和类适配器

VB.NET **中的** List，Tree **和** Grid **都属于对象适配器，这是因为它们都是包含了我们所要适配的可视化组件的类。然而，在此基础之上我们很容易进一步实现包含** addText **方法的** List **或** Tree **导出类。**

对 DataGrid，**因为我们不得不产生一个** DataTable **的实例，并在** DataGrid **类中产生** Columns。**这样该类是一个十分复杂的类，必须了解允许此类的工作细节，所以该方法不是一个好方法。**

15.10 可插入的适配器

一个可插入的适配器对一个类而言是动态的适配器。当然一个适配器仅能与一个可识别的类进行适配，通常这是由构造函数和 setParameter **方法决定的。**

15.11 在 VB 中的适配器

从更广泛的意义来说，在 VB7 **中已经建立了许多适配器，这些适配器与** VB6 **具有兼容性。它们是通过包装旧版的** API **函数来实现新的适配器，这就像列表框一样。**

思考题

你怎样写一个类适配器，使得 Grid **表看上去像一个两栏的列表框。**

15.12　光盘中的程序

\ Adapter\ TreeAdapter	VB6 **树适配器**
\ Adapter\ VBNet\LstAdapter	VB7 **列表适配器**
\ Adapter\ VBNet\GrdAdapter	VB7 **网格适配器**
\ Adapter\ VBNet\TreAdapter	VB7 **树型视图适配器**
\ Adapter\ VBNet\ClassAdapter	VB7 **基于类的列表适配器**