北京希望电脑公司

Microsoft Press

使用美国微软出版社授权的中文版系列书
编程利器·知识迸发

Learn Microsoft Visual J++ 6.0 Now
即 学 即 用

〔美〕 Kvin Ingalls Dan Jinguji 著
希望图书创作室 译
陈河南 审校

本书配套光盘包括三部分内容:
1. 软件 Microsoft Visual J++ 6.0 完全版和
Microsoft Internet Explorer 4.01
2 本书原版附带光盘中的例子代码及项目文件
3. 与本书配套的中文版电子书

北京希望电脑公司出品
1999

内 容 简 介

本书是美国微软出版社授权的系列中文版图书之一。全书共分为十三章，分别介绍 Java 的入门知识，从头编写 Java 应用程序，类与窗体，菜单、类型和方法，继承性，创建小应用程序，增强小程序，保存信息，小程序中的动画，Java 中的软件包，控制台应用程序，使用 MFC 中的组件，可移植性。书中还有四个附录，分别列出了快速格式对比、Java 格式参考及介绍了程序设计的趋势和时间驱动的程序设计。

本书的特点是浅显易懂、图文并茂，在介绍了基础知识之后马上用实例说明，并且还有实验安排，读者很快就能进入 J++编程的角色。本书附带的 CD-ROM 光盘中有 J++ 6.0 软件和 IE 4.01 软件，而且书中所涉及的实例程序代码和项目文件都在光盘中。读者可以直接使用这些项目文件，按照光盘中提供的代码来完成实验，在实际编程环境中学习 Visual J++，使用方便，省时省力。

本书适用于学习 Java 语言的各类编程人员，读者即使没有 Windows 环境的编程经验，也可以按照本书的说明很快学会。同时本书还可作为大专院校相关专业的师生的自学、教学参考用书。

本书配套光盘包括三部分内容：1. 软件 Microsoft Visual J++ 6.0 完全版和 Microsoft Internet Explorer 4.01；2 本书原版附带光盘中的例子代码及项目文件；3. 与本书配套的中文版电子书。

需要本书和配套光盘及需技术支持的读者请与北京海淀 8721 信箱书刊

部联系，邮政编码：100080，联系电话：010-62562329，62541992，62531267，传真：010-62579874。

版 权 声 明

Learn Microsoft Visual J++ 6.0 Now 即学即用

[美] Kvin Ingalls　Dan Jinguji　著

希望图书创作室　译

陈河南　审校

责任编辑　刘晓融　陈河南

北京希望电脑公司

北京希望电子出版社　出品

北京海淀路 82 号（100080）

新华书店、新华书店音像发行所发行、各地书店及软件专卖店经销

*　*　*　*　*

1999 年 1 月第 1 版　　1999 年 1 月第 1 次印刷

开本：787×1092　　印张：21.25

字 数：483 千 字　　印 数：1-5000
新 出 音 管[1998]210 号
ISBN 7-980023-12-9/TP・12
定 价：75.00 元（1CD，含 配 套 书）

献给我的家人和朋友们，没有他们的帮助，就没有本书的问世；献给我的父母，没有他们，也就没有我的存在。

Kevin Ingalls

献给我的朋友和我的家人，由于他们一直默默地支持我的工作，才使本书得以面世。

Dan Jinguji

致　　谢

应该感谢研究院……大多数书刊前的致谢都好象是作者正在接受学院奖。本书似乎也不例外。如果没有优秀的导演、制片以及强大的演员阵容和剧组人员，是不可能得到奥斯卡金像奖的。那个获得奥斯卡金像奖的家伙确实是天才中的佼佼者。感谢 Dan Jinguji 让我参与这个项目，他总是能把握机会，而且总是成功。感谢技术初稿编辑 Lisa Camire 和 Jennifer Jurcik，特别感谢他们在本项目第一阶段的研究中作出的贡献。他们所做的工作（而这些工作是作者本人难以做到的）使我的研究得以顺利进行。感谢 Beryl Doane 和 Pm Weizenbaum，他们是负责本书第二部分的初稿编辑。他们一开始便以惊人的高效率工作着。本书之所以能够取得成功，正是这四位优秀编辑杰出工作的结果（而我本人应对本书编写中出现的一些疏漏之处向他们致歉)。

Wayne Beardsley 先生是多年来我所在的波音公司的经理，他是人们所梦想的好上司（尽管有时令人畏惧）。Wayne 先生是为数不多的能够身先士卒的经理之一，他始终是一个好榜样。虽然他并不是本书的创作者之一，但是，如果没有他，本书也难以问世。感谢 Wayne 先生。

另外，我应该感谢我的妻子 Misa，我的女儿 Linnea 以及我们的小狗 Megan，感谢他们对我的工作的支持和帮助（甚至包括精神上的支持）。Misa 是我的第一个编辑，也是她的建议使得本书更加完美。在我编写这本书的时候，我六岁的女儿 Linnea 正在自己写一些东西，她把写好的东西用打印机打印出来，然后再将它们装订在一起，俨然一个出版商。我料想她将来必将会成为一位了不起的出版商的。至于我的小狗 Megan，每当我工作到深夜的时，她总是静静地蜷

缩在我的脚边。她从不问我在做些什么，但是，我想她一定很想知道，因为她会坚持几个小时地伸开脚爪，瞪大眼睛，惊奇地望着我身边那个发光的显示屏幕。

Kevin Ingalls
1998 年 7 月

我有那么多的朋友需要感谢。Kevin Ingalls 在我十分困难的时候加入了进来。就像做其他事情一样，有平坦顺利，也有崎岖波折，有了朋友的陪伴，困苦的旅途就变得快乐无比，而且令人回味。感谢 Lisa Camire, Jennifer Jurcik, Beryl Doane 和 Pm Weizenbaum 编辑，是他们将我引上正途。与他们杰出的工作相比，我所做的竟显得有些微不足道了。

感谢 Brian Engler, Niall McDonnell, Lars Mohr 和 Fay Tanagi，他们一直十分关心本书的编写进度。还有 Jordan, James 和 Philip Fleming，在枯燥漫长的编写和校对的日子里，他们总是能带给我温馨和欢乐。感谢我的支持者 St. James Cathedral，他总是让我充满希望，另外，特别感谢 Mrianne Cote, the McCabeSchmeltzers, Alison Warp, David Brazier 和 Joe Adam.

特别感谢我的家人：Dorothy, Beth, Theresa, Todd, Mayumi, Rebecca, Tom 和 LeeAnn。在任何时候，你们总是与我同在。

Dan Jinguji
1998 年 7 月

本书中文版的问世渗透了参与策划、翻译、录排、审校和出版人员的大量心血。本书由贺军、贺民、王雷、陈武、刘传凯、高胜友、佟大双、程宏筠等人翻译，邓蛟龙、陈代川、龚亚萍、王晓娟、吴安定、孟丽艳、王学龙、张定军等人也做了大量的工作，希望图书创作室技术人员对该书进行了仔细审校，在此深表感谢。

作 者 简 介

Kevin L.Ingalls 是西雅图波音公司的程序设计员和软件工程师，他在那里的培训部工作，开课讲授 Ada, C/C++和面向对象技术。他住在华盛顿州肯特郡，同妻子 Misa，两个女儿，Linnea 和 Megan，以及 Amelia 切萨皮克湾拾物犬（一种马里兰州培育的狩猎犬，由拾物犬和纽芬兰犬杂交而成，译者注）住在一起。他同妻子很恩爱，都喜欢去国外进行热带 SCUBA 潜水度假。

Daniel Jinguji 是微软公司研究 Visual J++的开发人员。他有 15 年教育和培训计算机程序员的经验，经常参与计算机界的重要事情。在二进制兼容性和软件组件的领域之外，他还忙碌于早期音乐、罗马礼拜仪式和圣礼传教等领域。

目　　录

设置属性
添加事件处理程序
存储并运行应用程序
应用程序的执行进度
设计应用程序
设计窗体
事件处理
实验 2-1：修正 Hello 代码
增强应用程序的功能：
添加逻辑决策（Decision Logic）
使用颜色
显示图片
插入注释
如何在 Java 当中创建注释内容
TODO 注释
JavaDoc 注释
实验 2-2：输入保密口令
调试代码
断点
在源代码中单步运行
查看变量
Immediate 窗口

使用数组
- 声明数组
- 创建数组
- 访问数组元素
- 数组作为对象
- 实验 8-1：改进电话簿

使用文件
- 文件 I/O
- File 类
- 打开文件
- 写文件
- 读文件
- 关闭文件
- 重复动作：一个简单的循环
- 实验 8-2：更新保存的电话号码簿

使用列表
- 创建列表
- 向列表中添加条目
- 查看列表中的条目
- 删除列表中的条目
- 实验 8-3：创建动态的电话号码簿

接口

简介

欢迎学习 Visual J++语言。Visual J++语言是计算机语言发展的一个重要分支。近几年来，越来越多的人开始学习 Visual J++语言，并使用 Visual J++语言实现 Windows 环境下的编程。这主要是因为随着计算机网络的发展，万维网（World Wide Web）风靡世界，万维网系统向地球电子村的实现迈进了一大步。而 Visual J++6.0 语言正是帮助我们程序员迈向未来的有力工具。使用 Visual J++语言，可以同时方便快捷地开发 Windows 环境下的应用程序和 Web 小程序。

本书是针对程序员编写的。Java 语言功能强大，而且表达能力强，对于初学者有相当的难度。尽管读者不必是使用 Java 语言的专门程序员，但是，我建议结合计算机编程环境进行学习。

若要从本书中学习尽可能多的知识，应把自己视为 Windows 系统的用户，尽管并不需要有 Windows 环境的编程经验。未曾使用过微软（Microsoft）编程软件的用户，很快就会发现，在 Windows 环境下使用 Visual Studio 和 Visual J++是非常高效便利的。

本书将一步步由浅入深地使读者熟悉 Visual J++环境。首先从 Java 语言的基础 Windows 应用程序开始，而后转向潜入 Web 网页的 Java 小程序。接下来，将学习创建和使用 Windows 环境下可重用的 Java 组件。随着对本书内容的深入了解，将进一步学习利用 Java 环境创建基于项目文件（projects）的 Windows 应用程序和 Web 小程序。关于项目文件，将在以后章节的正文和各章节后的实

验课热身练习中见到。本书附带一张 CD-ROM 光盘，本书中所涉及的实例程序代码和项目文件都录入其中。读者可以直接使用这些项目文件按照光盘中提供的代码来完成习题学习 Visual J++。

本书不是 Java 语言方法的目录。单单是罗列 Java 的标准库文件，就要占据数百乃至数千页的篇幅。要学习 Java 语言，最重要的是找到有规可寻的巨大库文件的规律。在编写程序时，非常忌讳重复性的工作，对于 Java 语言更是如此。在介绍语言时，本书引用了大量的 Java 库文件，而并没有包括全部的库文件。但是，本书仍是一个良好的完备体系，因此，读者在阅读实例和做习题时，不必使用另外的 Java 语言参考手册。当然，在使用 Java 语言时，手边有一本好的参考手册是可取的。

本书描述了 Visual J++标准版的基本功能，并未包括区别 Visual J++企业版和专业版的特征。

以下是各章节的目录：

章	内 容
第一章	本章概述 Java 语言和 Visual J++环境，描述 Java 语言和 Visual J++环境的主要特征，并利用应用程序向导创建我们第一个 Java 应用程序，一个记事本风格的编辑器
第二章	本章讨论 Windows 应用程序的一些基本要素：窗体（forms）、事件（events）和事件处理程序（event handlers）。Visual J++应用 WFC 控件来实现与用户的交互。至于 Java 语言，我们创建使用 Java 句法的方法，来处理用户事件。Java 程序组件和 JavaDoc 文档也在本章中阐述

第三章　类（Classes）是编程的核心，窗体（Forms）是应用 Visual J++环境创建 Windows 环境应用程序的核心。本章对这两部分内容均有介绍。本章还将定义和使用类、对象（objects）、引用（references）和构造器（constructor）的方法。同时还将介绍方法（methods）和成员变量

第四章　本章将详细介绍 Java 语言的语法，以及 Visual J++的菜单。同时还将讨论 Java 语言的内建数据类型及使用 Java 类中基本建立模块的方法。在学习了第三章中的成员变量的基础上，本章将介绍如何使用 Java 语言的语法来改变成员变量的行为

第五章　继承性是面向对象计算机语言的主要特点，Java 语言为继承性提供了大量的支持。在本章中，首先通过实例来检验 Java 的继承性，并进一步学习 Java 的类和方法。在此，读者将了解到类修改符（modifier）是如何影响类的行为及继承性的。Visual J++类在线窗口会及时为用户提供帮助，以建立程序代码，并得到使用的处理程序

第六章　本章中将第一次由用户与 Windows 环境应用程序对话，转向用户与 Java Web 小程序的对话。同时学习利用 Java 内置库函数方法创建在 Internet 网络浏览器上运行的小程序

另外，本章还将讨论 Web 网页，HTML， Visual J++ 小程序模板，以及 Java Abstract Window Toolkit（AWT，抽象窗口工具箱）绘图软件

第七章　本章将介绍如何向小程序中添加声音和图像。我们提供了两种方

	法，来让其他人自定义小程序：<PAPAM>标记和 JScript 函数的调用。同时，使读者了解如何将 HTML 控件和 Jscript 代码绑定在一起，以便与小程序通信
第八章	到目前为止，应用程序处理和存储信息的能力相对来说仍是有限的。本章着重介绍 Java 应用程序几种不同的信息存储方式：数组（arrays）、文件（files）和列表（lists）.另外，还以部分篇幅介绍排序（sorting）和接口，在 Java 中，这是类似类（class-like）的构造
第九章	卡通制作是 Java 在小程序上最初的用途，本章将介绍如何在小程序上创建卡通影像。为此，应首先让读者了解 Java 的两大强有力的功能特征：异常（exceptions）和多线程
第十章	Java 中拥有成千上万的类，其中包含的方法数是十分庞大的。Java 的软件包通过整理有逻辑联系的类，而实现对类的有效管理。在本章中，将介绍标准 Java 软件包和 WFC 软件包中类的管理，同时介绍用户如何创建和使用自己的软件包
第十一章	在本章中，介绍如何使 Java 类成为应用程序，同时介绍怎样运行 Java 程序，以及在应用程序中参数是如何传递的。另外,本章还介绍了如何阅读和书写 MS-DOS 环境下的提示信息
第十二章	功能强大齐备的可重用的组件技术，无疑会给软件开发者带来极大的方便。本章介绍 ActiveX 控件和一个新的 WFC 组件的使用方法，以及应用 J/Direct 实现对 Win32 应用编程接口的直接访问
第十三章	本章将介绍从应用程序访问 Internet 网络。标准的 Java 库文件另

外包含了一些软件包，这些软件包连接到 URL，并可以访问其源文件

附录 A　语法对照表：分别用 Java, C,C++语言编写的源代码程序的对照

附录 B　Java 语法说明：包括语法规则及语言的全部关键字

附录 C　编程近况：简单介绍软件项目开发的生存期

附录 D　事件驱动编程：介绍事件驱动和连续编程的不同之处，并对此提出自己的看法

Java 语言有着自己独特的优点，并在它自身的环境中得以迅速发展。Java 是一种非常好的语言工具，而 Visual J++正是一个绝佳的 Java 语言开发环境。相信读者在使用本书学习和应用 Java 语言时，在有所收获的同时，也会感到轻松愉快。

下面的一些信息为读者进一步的学习和提高提供参考：

如果想进一步学习 Visual J++环境下的编程，请参考《Programming Microsoft Visual J++6.0》(Microsoft 出版, 1998)。

要学习使用和扩展网络应用程序接口，使用 Microsoft Visual J++6.0, 请参阅《Network Programming with Visual J++ 6.0》(Microsoft 出版, 1998)。

对于中级程序员，要从事基于 Windows 应用程序的面向任务的程序开发，请参阅《Microsoft Visual J++6.0 Developer's Workshop》(Microsoft 出版, 1998)。

想要了解 Windows 基础类库文件，请参阅《Microsoft Visual J++6.0 Reference Library》(Microsoft 出版, 1998)。

关于配套光盘

本书包括一张 CD-ROM 光盘，本书中所涉及的实例程序代码和项目文件都录入其中。读者可以直接使用这些项目文件，按照光盘中提供的代码来完成习题，以便学习 Visual J++。

在计算机上装载项目文件实例

装载程序将自动在计算机上建立一个名为 JV 的子目录，并将所用的文件拷贝到该子目录下。

装载过程

1. 关闭所有正在运行的程序。
2. 将 Learn Visual J++ Now 光盘放入光驱。
3. 在屏幕底部任务栏单击 Start 按钮，然后单击 Run 按钮，屏幕上将会出现 Run 对话框。
4. 在 Open 对话框中输入 D:\setup.（如果计算机光驱盘符是其他字母，则应输入其他字母，比如光驱是 e 盘，则输入 E:\setup 即可)。
5. 单击 OK 后，就可以按照屏幕上的提示往下进行。

Visual J++代码项目文件（Code Projects）

本书的每个章节包括后面的练习题，都将反复强化读者对书中提到的基本

概念的理解。同时，在篇后的每个习题中都给出提示，以便顺利地开始 Visual J++，并打开合适的文档。Visual J++将 Java 源代码以及其他源程序组织到“解决方案”（solution）文档（扩展名为.sln）及项目文件文档（扩展名为.vjp）中。Java 的项目文件与 Windows 应用程序相类似，每一个解决方案可以包括一个或者多个这样的项目文件。也就是说，任何一个解决方案都可能包括多个项目文件，不过，在本书给出的例子中，美国方案中只包括一个项目文件。每个解决方案/项目文件都会从光盘安装到硬盘上指定的 Windows 目录下。

配套光盘上有示例项目文件、实验课项目文件和解决方案项目文件。以上所有文件的命名遵循以下规则：

项目文件类型	文件名
示例项目文件	Chapter04\BuiltIn\BuiltIn.sln
实验课项目文件	Chapter03\Lab3-1\BidMaker.sln
解决方案项目文件	Chapter02\Sol2-2\Hello.sln

示例项目文件（Example projects）

每个示例项目文件都着重说明一个 Java 编程语言的基本概念。示例项目文件在本书的正文中有所阐述，并可以随时打开项目文件进行验证，而不必对 Java 代码作任何改动。另外,要说明一点，不是所有的章节都附有示例项目文件。

实验课项目文件（Lab projects）

实验课项目文件是每个试验习题的开始。它给出一些只需在某方面作出改动的 java 代码，而使用户方便快捷地开始练习。在练习之前的段落中，实验课

中用到的基本概念已经有所讨论，习题不是必须一个启动器（starter）项目文件，因而可以重新创建一些项目文件。

解决方案项目文件

解决方案项目文件为试验课习题提供了可行的解决方案。每个试验课习题都有一个解决方案项目文件，利用这些项目文件来考核练习情况。

安装Visual J++

Learn Visual J++ Now 光盘带有 Microsoft Visual J++出版版本，这是一套包含了 90 天训练课程的学习 Microsoft Visual J++的标准版软件。应用该软件，可以创立、组建、运行、调试及编辑 Java 程序。

如果计算机上已经安装了其他版本的 Microsoft Visual J++，那就没有必要再安装出版版本了。Visual J++的 setup 文件在光盘中的 VJ98 目录下。

Visual J++的安装和运行需要在 Internet Explorer 4.01 Service Pack1 和 Microsoft Windows 95, Microsoft Windows 98, 或者 Microsoft Windows NT 系统环境下进行。

安装 Visual J++

1. 关闭所有正在运行的程序。
2. 将 Learn Visual J++ Now 配套光盘放入驱动器。
3. 在屏幕底部任务栏单击 Start 按钮，然后单击 Run 按钮，屏幕上将会出现 Run 对话框。

4. 在 Open 对话框中输入 D:\VJ98\setup.exe（如果计算机光驱盘符是其他字母，则应输入其他字母，比如光驱是 e 盘，则输入 E:\setup 即可)。
5. 单击 OK 后，就可以按照屏幕上的提示往下进行。

系统要求

计算机/处理器	**奔腾处理器 PC 机，Pentium90 或更高**
内存 RAM	在 Windows 95 或更新的 Windows 系统，24MB 可行（建议 48MB）
	在 Windows NT 系统下，32MB 内存可行（建议 48MB）
硬盘	典型: 107MB.
	最大: 157MB.
	IE 4.01: 43 MB (典型)
	MSDN: 57MB(典型)
	NT 4.0 Option Pack: 20 MB(Windows 95+)
	200MB(Windows NT 4.0)
光驱	CD-ROM 驱动器
显示器	VGA 或更高；建议 Super VGA
操作系统	Microsoft Windows 95 或者更新的操作系统，或者带有 Service Pack3 的 Microsoft Windows NT 操作系统 4.0 版，或更新的版本
外设/其他	Microsoft 鼠标或其他定点设备

要求安装好 Internet Explorer 4.01 SP1。

第一章　Java 入门

欢迎学习 Visual J++！本书旨在介绍如何应用 Java 编程语言和 Microsoft Visual J++开发系统创建应用程序。

在本章中，将介绍以下内容：

- Java 语言
- Microsoft Visual J++ 6.0 版

你将能够：

- 利用应用程序向导（Application Wizard）建立一个简单的文本编辑器应用程序。

Java 语言

Java 编程语言非常引人注目。其中一个主要的原因是因为 Java 语言是在 WWW（World Wide Web，也被说成 Web 或 Internet，以后均简称为 WWW）上编程最明智的选择。人们谈论他们在使用 Java 语言的一些事情，涉及最多的就是这种语言与众不同的设计方法。在本节中，首先介绍一些关于 Java 语言的

基本信息。虽然这些信息对于学习 Java 语言的编程并不是必不可少的，但这也是极好的背景知识。

Java 首先是一种编程语言，它具有以下几方面的主要特点：

- 它是一种面向对象的编程语言（OOP）
- 它的语法结构与 C 和 C++语言极为相近
- Java 所用的语法十分简单
- 具有良好的可移植性

我们来深入看看每一种特征。

面向对象（OOP）

我们知道，如果将一个大的计算机源程序作为整体来编写，工作会十分复杂和繁重。因此，往往将一个程序划分成若干小的模块，各个模块采用不同的方案，这样，就可以使各个模块之间保持相对的独立性，以便不同的人可以同时各自编写一个模块。

程序员在维护程序时，需要对其作适当的修改，使得这些模块的组成结构显得极为重要。在运行维护过程中，往往希望程序可以实现更为理想的功能，或者希望利用以下操作系统的新特性，或者希望检验程序的某项功能。如果组成应用程序的各个模块之间不能保证相对独立的话，程序员在维护时可能要改动几个模块，而这些改动就很可能引起其他更多的模块要作出相应的改动，新的改动又会引起必要的相应改动，依此类推。如果这样的话，软件的维护工作简直无法进行。

许多资料都阐明了，面向对象编程语言是解决应用程序的模块化及维护问题的有效技术。在面向对象语言编程中，各个模块以应用程序处理的对象（objects）为基础，就像应用功能分解技术创建子模块一样。在功能分解中，各个子模块又以应用程序本身提供的函数为基础。

下面，我们来看一个示例，存货清单控制应用程序。在OOP中，对象包括有：存货清单条款、部件数、停放明细表、存储位置等。在功能分解过程中，可以有接收、加标签、包装及运输等诸如此类的功能。如果部件数的格式发生变化，比如说从6位数字变为6个字符的字母数字，那么，对于一个优秀的OOP解决方案，这些改动就应该可以在一个部件数模块中完成，而不必对其他模块进行改动。当然，其他模块对部件数模块会有调用，只是在这些模块中，把部件数模块作为一个“黑箱”实体（entity）。也就是说，在这些调用部件数的模块当中，不包含任何关于部件数模块的组建和安装启用信息。通常，在标准的功能分解方案中，像部件数那样的基本实体，在应用程序中，可能直接对每个模块操作，也就是说，如果部件数的格式发生了变动，也就意味着，所有引用部件数的模块都要更新和调整。所有这些都进一步说明了模块化设计的好处。任何良好的面向对象的编程，都应比单单基于功能分解的方法更加容易编写和维护。

目前，已有很多关于对象及OOP的书籍，本书并没有对OOP进行详尽完备的讨论，而只是在以后的章节中对对象与编程知识进行简单的介绍。

类似C和C++的语法结构

一个编程语言的语法结构，是指它是如何将语言的各个部分有机地组合起

来而形成程序的。刚一接触，你就会发现，Java 程序的确与 C 和 C++程序极为相似（附录 A 中给出了一些示例，以便比较 C, C++, Java 和 Visual Basic 的语法）。C 语言是一种效率很高、灵活性很强的计算机语言。相当一部分商业软件是用 C 语言编写的，然而，它没有内建的 OOP 支持，C++的语法同 C 类似，但它有了内建的 OOP 支持。C 和 C++都是非常流行的计算机语言，目前上市的大部分软件都是由 C 或 C++编写的。由于 Java 的语法与 C 和 C++语言的语法类似，因此，用户可以很容易地将 C 或 C++程序改写成 Java 程序。

简化的语法结构

Java 语言的发明者拥有丰富的 C 和 C++语言的经验，他们在吸取 C++ 优秀特征的同时，舍弃了 C++ 中一些容易引起问题但不是绝对必要的部分（遗憾的是，他们并没有完全解决这些问题，不过也解决了绝大多数)。

C 和 C++语言中都设有指针。我们知道，指针是 C 和 C++语言完成一些高级程序设计的基础，然而，也正是指针容易引起问题，有时甚至会带来致命的麻烦。可以认为，指针是一把双刃剑，如果能够正确使用，它能实现许多功能，而一旦使用失误，就有可能给用户带来灾难。如果不是为 OOP 设计的语言，也很难向其中加入 OOP 元素。Java 取消了指针，而使用引用（references）。这样，通过使用引用，不但保留了指针的灵活性，而且去除了许多由于指针的存在而潜在的隐患。C 语言的基本组建模块是函数，这极适合功能分解。Java 将类作为它的基本组建模块。Java 类非常适于处理对象，因此，也自然适于面向对象编程。这些转变去除了 C++语法中许多含糊不清的部分。

以上都说明了 Java 语言对语法作了大量的简化，它比 C 和 C++语言都要容

易学习。

可移植性

良好的可移植性也是 Java 语言倍受青睐的一个原因。具有良好的可移植性，就可以灵活地从 Web 站点上下载并使用应用程序。

用户如何取得程序：传统方案

一般用户所创建的程序，都是针对某一台专门的计算机和操作系统的，程序只能在专门的机器和操作系统上运行。因此，PC 机上创建的程序只能在 PC 机上运行,而不能在其他机器如使用 UNIX 操作系统的工作站或大型机上运行。这样，用户只能自己到计算机商店去购买软件，而且只能购买自己的计算机专用的软件。

用户如何取得程序：Web 方案

在 Web 网页，就与上述情况不同了。用户可以从网络服务器上将程序下载到自己的计算机上。HTML、图像和声音等都可以下载。网络浏览器会将这些片段组织到 Web 网页上，以供用户查看。

当需要在网络上运行一个程序时，根本无法保证所有的网络用户都使用同一型号的计算机，这就需要在 Web 上运行的程序可以保证在任何计算机上运行，而 Java 语言恰恰是为了实现这一目的而设计的。

编译器

目前流行的计算机语言都采用英语，来实现便于程序员记忆和开发应用程序的目的。但是，我们知道，一台计算机是无论如何也不会懂得诸如“if”或“public”以及“return”的含义的，这样，就需要编译器，将这些代码翻译成计算机能懂的机器代码。不同类型的计算机使用不同的机器代码，Java 语言设计旨在解决建立不同的计算机使用的应用程序的问题。

Java 的字节代码和虚拟机

Java 编译器将 Java 源代码翻译成机器码。然而，这些机器码并不是特地为 PC 机或工作站或大型机而生成的，而是为一台并不存在的“虚拟机”生成的。也就是说，Java 语言为虚拟机（Virtual Machine，VM）生成机器码。这种虚拟机的 Java 机器码也称作字节码（bytecode）。那么，为一个不存在的机器生成机器码，究竟有什么好处呢？尽管虚拟机并不是以晶体管、二极管和各种线路的形式存在，但它的确存在。Java 的虚拟机实际上是一个程序，这个程序实现字节码向实际机器代码的转换。如果没有考虑到在 Web 上编程的难度，以上做法听起来似乎有一些奇怪。在网络上，会有很多用户可能访问你的网址，而谁也不可能知道这些用户正在或将要使用什么类型的计算机。使用传统的编程语言，用户不得不为每一种可能的机器类型编写一种版本的应用程序，然后，还要确保这些程序要为对应机型的用户所下载。如果网络访问者并未告知他的机器类型，那就不得不应付层出不穷的对话框，例如：“单击此处查看此页”、“如果您在西班牙，请单击此按钮”等等，显然，这对一般用户来说，是难以容忍的。因此，应用了虚拟机的字节码也是 Java 语言的又一个主要的魅力所在。

良好的移植性

Java语言良好的移植性，使得它可以创建一些精巧的小程序，在Web上完成动画设计和人机交互，但这也有些不利的地方。

首先，影响速度。使用其他编程语言，机器码在程序运行时已经就绪，而使用Java语言，首先要生成对应虚拟机的字节码，然后进一步生成真机代码。

第二，Java语言依赖Java虚拟机。用户一定会经常遇到，在原来的操作系统上更新软件版本的问题，版本的更新是十分重要的。Java虚拟机也有自己的版本，单是Windows环境下的就有12个版本。同样的计算机，使用不同的虚拟机版本，其运行情况会有相当大的差别，有时运行非常良好，有时可能会无法运行。

第三，移植性本身就是有争议的。在一个系统上，可以安装有一个、两个、三个乃至五个鼠标健的鼠标（X Windows System支持5个鼠标键的鼠标器）。为了保证程序的可移植性，在设计一个程序时，就必须考虑支持单鼠标健的鼠标，现在，大家都在使用有两个鼠标键的鼠标，于是支持单键鼠标的Java程序就不免显得过于蹩脚。

第四，一些技术问题使得Web网络尚且不能成为人们传递信息的基本工具。其中一个最主要的原因是由于下载复杂软件所消耗的大量时间。在有些情况下Java程序良好的移植性的吸引力被其巨大的尺寸所抹煞了。

第五，许多人往往只是为自己所在的公司的小网络编写程序，而不是编写Internet软件，这些小网络有时称为企业网或内部网。一般来说，在一个公司的内部所用的计算机类型和计算机软件的版本都是标准化的，因而也就很少追求一种完全可移植的软件。内部网一般都有足够的带宽，以保证下载信息的速度。

因此，程序的可移植性并不总是人们所最关注的。尽管如此，下载技术的发展仍为我们提供了令人激动的前景：自动更新。

自动更新

我们知道，用户每次访问 Web 站点时，都可以从某个网址得到新信息内容的拷贝，如果有人每天都将网址的内容更新一次，那么，访问者就每天都可以得到新的内容。对于 Java 字节码也是如此，如果有人将新的 Java 字节码载入 Web 站点，以后就会有某个访问者在相应的 Web 站点，将这些新的 Java 字节码下载到自己的计算机上。这是相当明智的做法，你可以毫不费力地将软件维护与更新的工作分配给网友，可以通过对网络的访问下载最新版本的应用软件。

Java 软件包

最初学习计算机语言的经验表明，一个应用程序所能完成的功能，要么是语言本身提供的，要么是我们自己编写的。Fortran 就是这样的。从磁盘或磁带等存储器读写数据的命令在 Fortran 语言的内部。大多数新的编程语言都带有一个拥有许多实用子程序的函数库。这些子程序处理函数就同从存储器上读写数据类似。函数库具有以下两点好处：

- 库函数对于语言来说是外部的，使得语言更加简洁。
- 库可以独立于语言而不断地更新。

Java 也带有这样的库。库中的子程序按功能可以分为：绘图、处理存储和创建小程序等。这些库文件整理归类到软件包中。这些软件包也构成了 Java 语言应用的一个相对独立的部分。事实上，几乎所有实现

可移植性的代码都在库当中。

有些软件包与 Java 语言本身的联系相当紧密。例如：如果不使用 java.lang 软件包，就不能写最简单的 Java 程序。java.lang 软件包包括 String 和 Object 类，如果不使用 String 类，编写 Java 程序就会十分困难，如果不使用 Object 类，就绝对不可能编写 Java 程序。其他软件包，例如 AWT，它的用途非常专一，这个软件包处理图形用户界面（GUI）元素，比如显示窗口、菜单和按钮等。AWT 用以支持 Java 语言 GUI 的可移植性。WFC (Windows Foundation Class)库包含 com.ms.wfc.ui 软件包，它是伴随 Visual J++6.0 而诞生的，可以代替 AWT 实现窗口、菜单和按钮等的显示。使用 WFC 的主要目的在于从功能强大而又灵活有效的 Win32 图形库中得到帮助（这部分内容将在以后的章节当中进一步阐述)。

外部库文件的使用使得程序员能够为自己 GUI 软件包选择 AWT 或者 WFC。

Java 语言的开发系统 Visual J++

Microsoft Visual J++是开发 Java 语言创建 Java 应用程序的工具。Microsoft Visual J++是 Microsoft Visual Studio 套件中的一个成员，适合于创建应用程序。Microsoft Visual Studio 套件中的其他成员还包括 Microsoft Visual C++和 Visual Basic 开发系统。Microsoft Visual C++为创建强有力的应用程序提供了良好的开发环境。其中包括一个高级调试器、一个灵活的项目文件组建系统，以及一个

高度优化的编译器。Microsoft Visual Basic 是一个快速的应用程序开发环境（Rapid Application Development，RAD），该语言因为可以支持创建基于 Windows 系统的应用程序而倍受青睐。Visual J++继承了上述两种语言各自的优点。

Visual J++的早期版本拥有同 Visual C++相类似的开发环境，而 Visual J++6.0 版本引入了 Visual Basic 语言开发系统的许多特征，包括可视窗体设计和属性窗口。Visual J++继承了许多使 Visual Basic 语言得以成功的特点，比如拥有强有力的编程环境，综合了高质量的应用程序设计、代码生成、组建和调试功能等。这便是一体化的编程环境（Integrated Development Environment 或 IDE）.

解决方案和项目文件

IDE 的工作贯穿于整个项目文件和解决方案系统之中。一个解决方案由一个或者一系列为了解决某个问题的项目文件所组成。每个项目文件都有专门的元素，对要待解决的问题的全部或部分进行访问。下面的示例可以更加明确地说明这个问题。比如要创建一个应用程序，来追踪在电视购物网上人们打电话购物的顺序。首先，将应用程序分解为若干比较简单的组件，比如，分解为一个处理日期的控件和一个处理货币流通的控件，并创立使用这两个控件的窗体。

在 Visual J++中，这个应用程序就以三个独立的项目文件的形式而创立：

- 用来创建处理日期控件的项目文件。
- 用来创建处理货币控件的项目文件。
- 用来创建使用以上两个控件的窗体的项目文件。

以这样的方式分别创建三个项目文件，就可以独立而且更加灵活地应用它们。例如，如果想要创建一个日记（diary）应用程序，这时，可以从头做起，也可以直接利用前面所创建的处理日期的控件。另外,，这种相对独立性，也可以使用户及时对控件作出必要的改动。当然，最简单的情形就是，一个解决方案中只包含一个项目文件。在本书中，大部分的试验习题和示例中，都是每个解决方案中只包含一个项目文件的情形，在讨论创建和使用新组件的章节中，出现了单方案多项目文件的情况。

Java 和 Windows: WFC 和 J/Direct

目前，许多计算机都在使用 Windows 操作系统。应用 Visual J++，可以方便地利用和发挥 Windows 系统的优点，这是因为 Visual J++有了用于 Java 的 WFC。

WFC

在前面我们提到，Java 依赖于软件包来处理大多数函数。为了清楚地比较 WFC 和标准 Java 软件包的区别，我们专门举出下面的示例：视窗。

标准 Java 视窗软件包是 AWT。AWT 软件包包含在 Java.awt 软件包中，支持简单的控件。它由一些绘图代码组成，并支持可移植性。

WFC 视窗库在 com.ms.wfc.ui 软件包中，其中的绘图代码进行了优化设计，更适合于 Win32 操作系统以及 Windows 95，Windows 98，Windows NT 操作系统的用户编程界面。它支持包括树视图、列表视

图和多文本编辑等所有的 Win32 控件。

AWT 控件	WFC 控件
Button	Animation
Checkbox	Button
Choice	CheckBox
Label	ComboBox
List	Edit
Menu	ImageList
Scrollbar	Label
TextArea	ListBox
TextField	ListView
	PictureBox
	RichEdit
	StatusBar
	TabStrip
	ToolBar
	TrackBar
	TreeView

WFC 还为用户提供了操作动态 HTML（DHTML）的方法，DHTML 允许 Internet 开发者创建 Web 网页和服务器应用程序，在用户从网上下载后，便对用户的 HTML 作相应的改动。

J/Direct

J/Direct 是 Java 为本机方法设计的应用程序编程接口。当 WFC 不能提供足够的功能函数支持时，可以利用 J/Direct API 访问 Microsoft Windows 动态链接库（DLLs），包括 Win32 应用程序编程接口。

本机方法属于非 Java 方法。它之所以被称为本机方法，是因为它访问的是本地计算机，而不是 Java 虚拟机。当 Java 虚拟机无法提供对给定的某个操作的支持时，就需要使用本机方法。例如，可以使用 J/Direct 调用 Win32 API，以访问 Windows 注册表。

另外，使用 Visual J++ 6.0 中的 J/Direct Call Builder，可以充分发挥包括 J/Direct 本身在内的优点，便可以利用并快速建立包括 J/Direct 的应用程序。J/Direct Call Builder 大大简化了 Windows 环境下 Java 程序的编写。而在过去，这些程序可能是用 C 或 C++语言编写的。

Java 的可移植性和 Visual J++

Visual J++支持可移植性 Java 程序的创建。编译器完全支持 Java 语言 1.1 版本，包括 Java Beans，因此，任何一个可移植性 Java 程序，无论是 Java 小程序，Java Beans，还是 Java 应用程序，都可以在 Visual J++上编译和运行。

根据本章前面对 Java 语言可移植性的介绍，我们知道，Java 的移植性是以不依赖于机器的字节码为基础的。只要提供“合适”的虚拟机，Visual J++所产生的字节码就可以在任何类型的计算机上运行。这里的“合适”是指支持同一的语言版本。有些虚拟机支持 1.0 版本，而另外一些虚拟机支持 1.1 版本，值得

时刻注意的是，基于 1.1 版本的代码，不可能在支持 1.0 版本的虚拟机上运行。另外要注意，使用了 WFC 的 Java 应用程序是不可移植的。这是因为 WFC 是专门为 Win32 API 而设计的。使用了 WFC 的 Java 应用程序，只可以在 Windows 95， Windows 98 和 Windows NT 等以 Win32 为基础的操作系统上运行。

Visual J++的不同版本

Microsoft Visual J++开发系统 6.0 版有以下三种不同的版本：

- 标准版
- 专业版
- 企业版

标准版的 Visual J++是独立的产品。它为学习 Java 语言及编写一般程序的用户所设计。标准版完全支持 Java 语言和 J/Direct。应用这套软件，可以创建基于 Windows 的应用程序、Web 上的小程序以及新的 WFC 组件；它还可以为应用程序创建 EXE 文件；并且包含了所有 WFC 库文件。另外，标准版 Visual J++不包含支持数据库访问和、数据提供者的安装和数据库驱动程序。本书以标准版 Visual J++文件为基础。

专业版 Visual J++是为专业的 Java 程序员开发设计的。它包含了标准版的所有内容，并在此基础上添加了建立、封装和配置 COM 以及 MTS 服务器组件、DLLs、CABs 和 ZIPs, 另外还有直接访问数据等功能。专业版的 Visual J++同 Visual Basic, Visual C++, Visual InterDev, Visual FoxPro 及 Microsoft Developer Network（MSDN）库一起组成了 Microsoft Visual Studio。

企业版 Visual J++是 Microsoft Visual Studio 企业版的一个组成部分。Microsoft Visual Studio 企业版是为编程小组所设计的，在编程小组中，会需要多种语言和工具来创建解决方案。企业版 Visual J++除了具有 Visual J++专业版所有的内容外，还加入了 Visual SourceSafe 和 Microsoft BackOffice 工具，比如 SQL Server、Microsoft Transaction Server, Internet Information Server 和 SNA Server。

Visual J++概述

下面的内容概述了 Visual J++,如果此时您的计算机上还没有安装 Visual J++的话，那现在就是安装 Visual J++的最好时机。

启动 Visual J++

- 从 Start 菜单选取 Programs 项，然后选择 Microsoft Visual J++6.0 即可。

  屏幕上会闪现 Visual J++的图像，接下来就是 Visual J++ IDE，然后就是典型的 New Project 对话框。可以用这个对话框来创建新的 Visual J++项目文件，也可以打开已有的项目文件或者解决方案。此时，单击 Cancel 按钮，跳过该对话框，屏幕上将出现 Visual J++的 IDE，下面是窗口的初始配置情况。

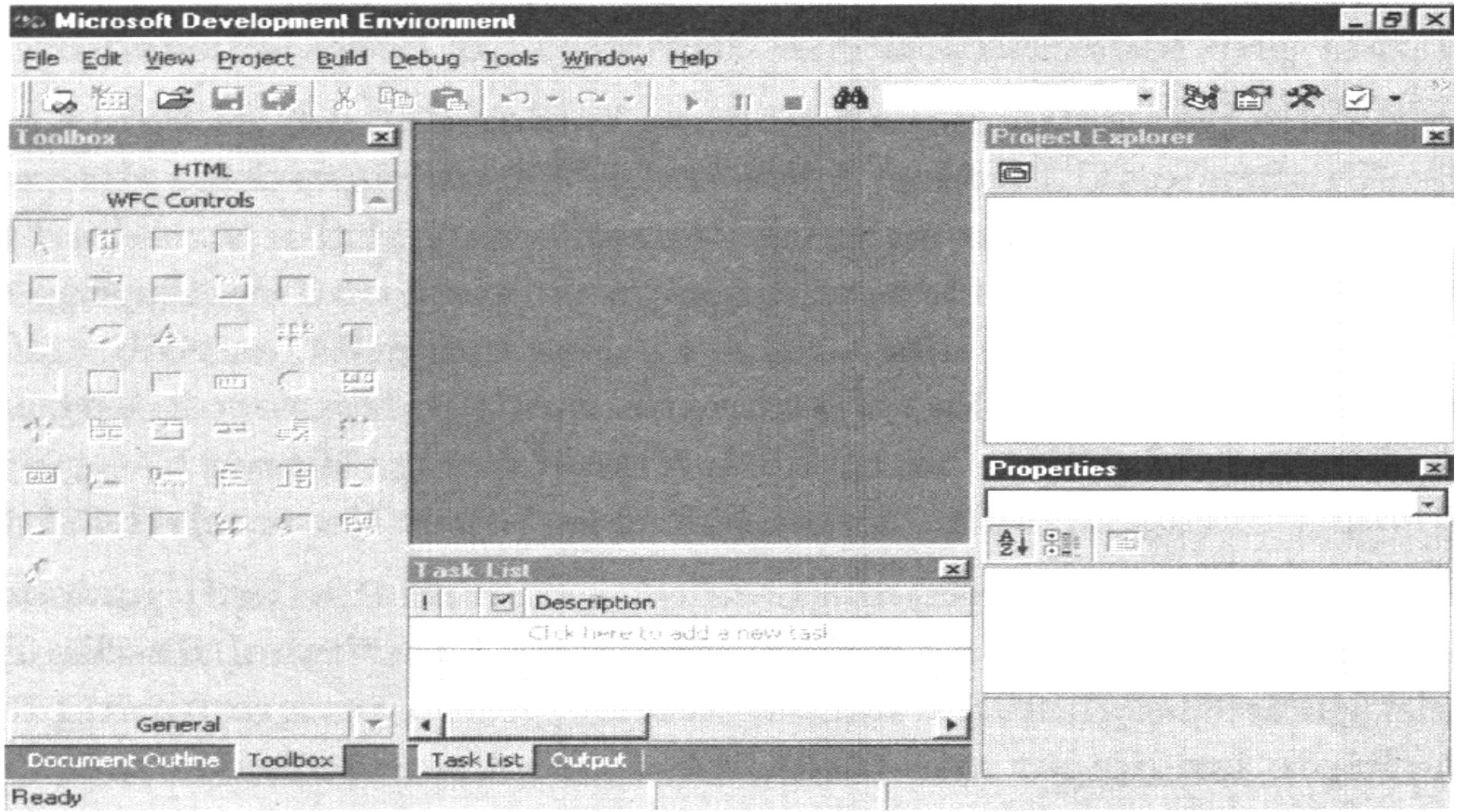


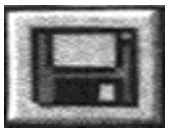
图 1-1　Visual J++ IDE

下面，是IDE处理任务时，顶端工具栏中的图标说明：

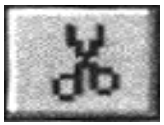

新建项目文件


新建项目文件项


打开项目文件

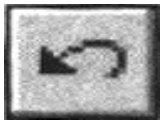

存盘


全部存盘

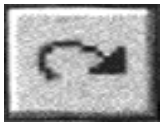

剪切


复制


粘贴

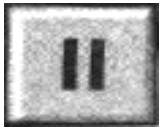


撤消

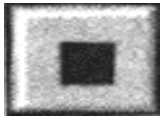


重做

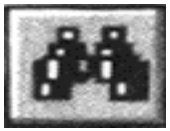
开始（调试）



中断



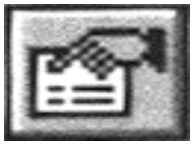
结束



查找

查找文本

开发项目文件（Project Explorer）



属性窗口

工具箱

任务列表

查看其他窗口

在 IDE 窗口的右上部分，可以看到 Project Explorer 窗口，其中显示了当前项目文件中的各个文档。用户可以应用 Project Explorer 在当前项目文件中增加或删除文件。

在 Project Explorer 窗口的下面，就是 Properties 窗口，Properties 窗口显示了在窗体设计中选取的关于窗体及控件的信息。

在 IDE 窗口的左边是带有两个选项卡的窗口，可以以多层的模式显示，单击选项卡就可以看到相应的内容。文档大纲（Document Outline）显示活动的 Java 或者 HTML 文件信息。工具箱（Toolbox）使用户可以方便地访问可以放置在窗体上的控件。

IDE 窗口底部的中央是任务列表窗口，显示各种不同的信息。这些信息包括语法错误、编译错误、TODO 注释和用户定义的任务等（Output 选项卡将在下一章讨论)。以上这六个窗口都嵌在一个主窗口上。用户可以随意调整各个窗口的位置和大小。同时也可以随时将工具停靠在主窗口中或隐藏它。

调整工具窗口的尺寸

- 用鼠标拖动边界到理想的位置即可。

改变工具窗口的位置

- 拖动工具窗口的标题栏到理想的位置即可。在拖动的同时，会有一个轮廓显示当前窗口的位置。

停靠一个工具窗口

- 拖动工具窗口，使其靠近主窗口的边界。轮廓显示工具窗口将停靠的位置。在拖动的过程中，只能改变工具窗口的位置，在完成停靠后，可以改变窗口的大小。

避免工具窗口停靠

- 在拖动工具窗口时，按着 Ctrl 键。

  从图 1-1 中可以看出，Document Outline 窗口和工具箱窗口同时出现在屏幕上，在 Visual J++中，可以为任何两个工具窗口一同设置选项卡。

两个工具窗口一同设置选项卡

1. 首先将其中一个定位。
2. 然后，将第二个窗口拖动到第一个窗口的标题栏位置。窗口轮廓将由原来的矩形变为选项卡的形状。

3. 放开鼠标键即可。

重新排列工具窗口选项卡

- 用鼠标拖动选项卡，将工具窗口移动到理想的位置，选项卡会在鼠标的拖动时，自动重新排列工具窗口的位置。

移动选项卡窗口

- 用鼠标拖动选项卡，将工具窗口移动到理想的位置。

隐藏工具窗口

- 单击工具窗口右上角的关闭按钮即可。

显示工具窗口

- 从 View 菜单中选择要显示的工具窗口项。另外，还可以对工具窗口作一些特殊的配置，可以定义一个窗口布局（Window layout），以便可以方便地返回到原来的窗口配置。

定义窗口布局

1. 从 View 菜单中选择 Define Window Layout 项。
2. 在对 Define Window Layout 话框中输入窗口布局的名称。
3. 单击 Add 按钮定义布局。
4. 单击 Close 按钮关闭对话框。

应用已经定义的窗口布局

1. 从 View 菜单中选择 Define Window Layout 项。
2. 在 Define Window Layout 对话框中选择想要的窗口布局。
3. 单击 Apply 按钮使用定义的窗口布局。
4. 单击 Close 按钮关闭对话框。

当然，还有许多的工具窗口和工具栏可以使用。

浏览工具窗口和工具栏列表

- 浏览 View 菜单

  在以后章节的学习当中，将更多地用到工具窗口，各个工具窗口在本书中第一次出现时，我们会作出说明。

实验 1-1：使用 Visual J++的应用程序向导创建应用程序

下面的实验学习如何利用 Visual J++创建一个应用程序。

实验概述

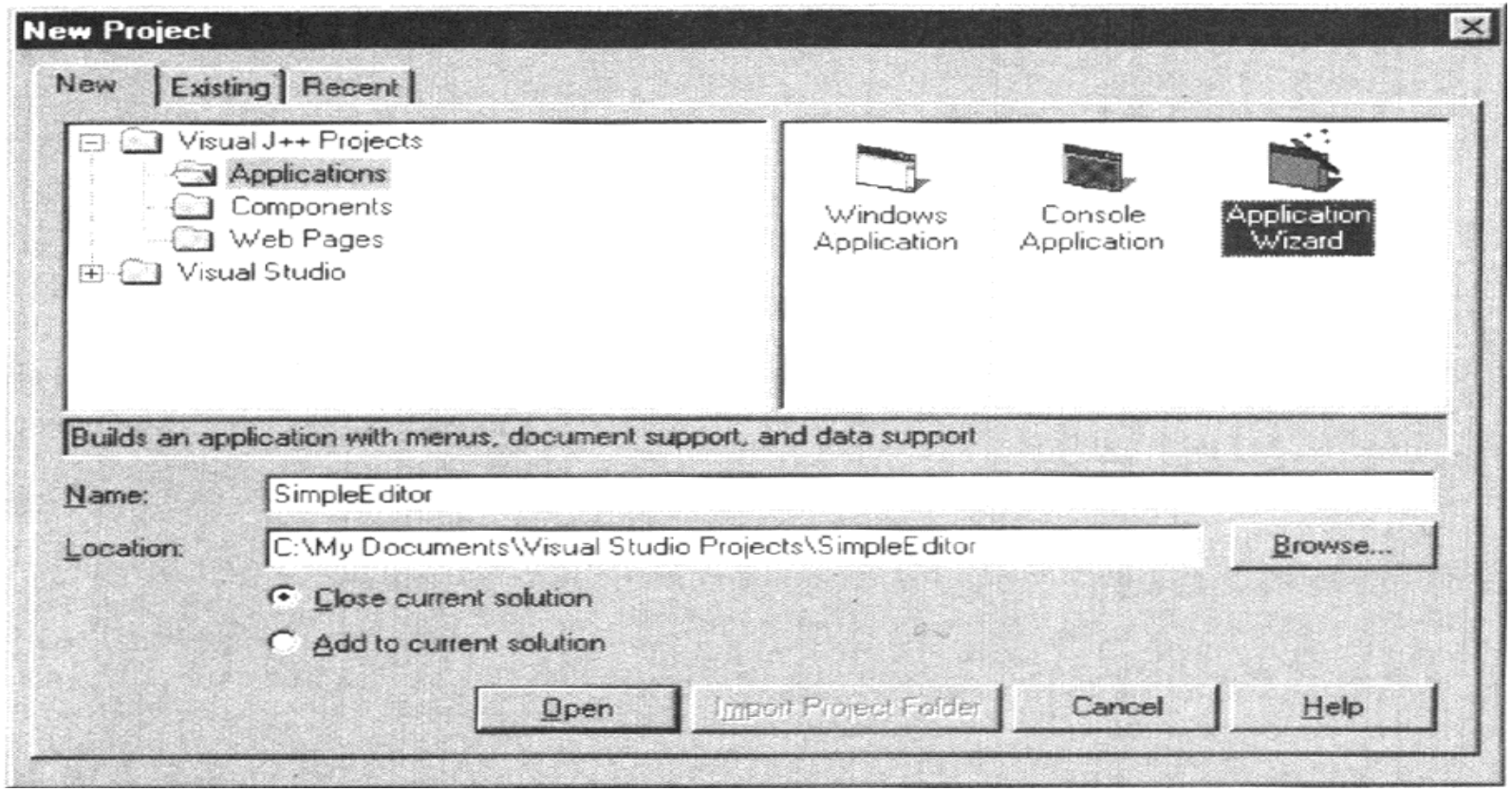
在本次实验中，我们将学习：

- 在一个新的解决方案中创建 Visual J++项目文件。
- 使用应用程序向导（Application Wizard）。
- 在有或没有调试（debugging）支持的情况下，在 IDE 上运行项目文件。

为了帮助用户开始创建应用程序，Visual J++提供了模板（Template）和向导。模板是一个项目文件的梗概或条款，向导可以引导用户一步步完成工作。在本次实验中，我们要使用应用程序向导，来创建我们的第一个应用程序，即一个简单的文本编辑器。该应用程序会用到本书中曾讲到的许多 WFC 库的特点。

试验准备

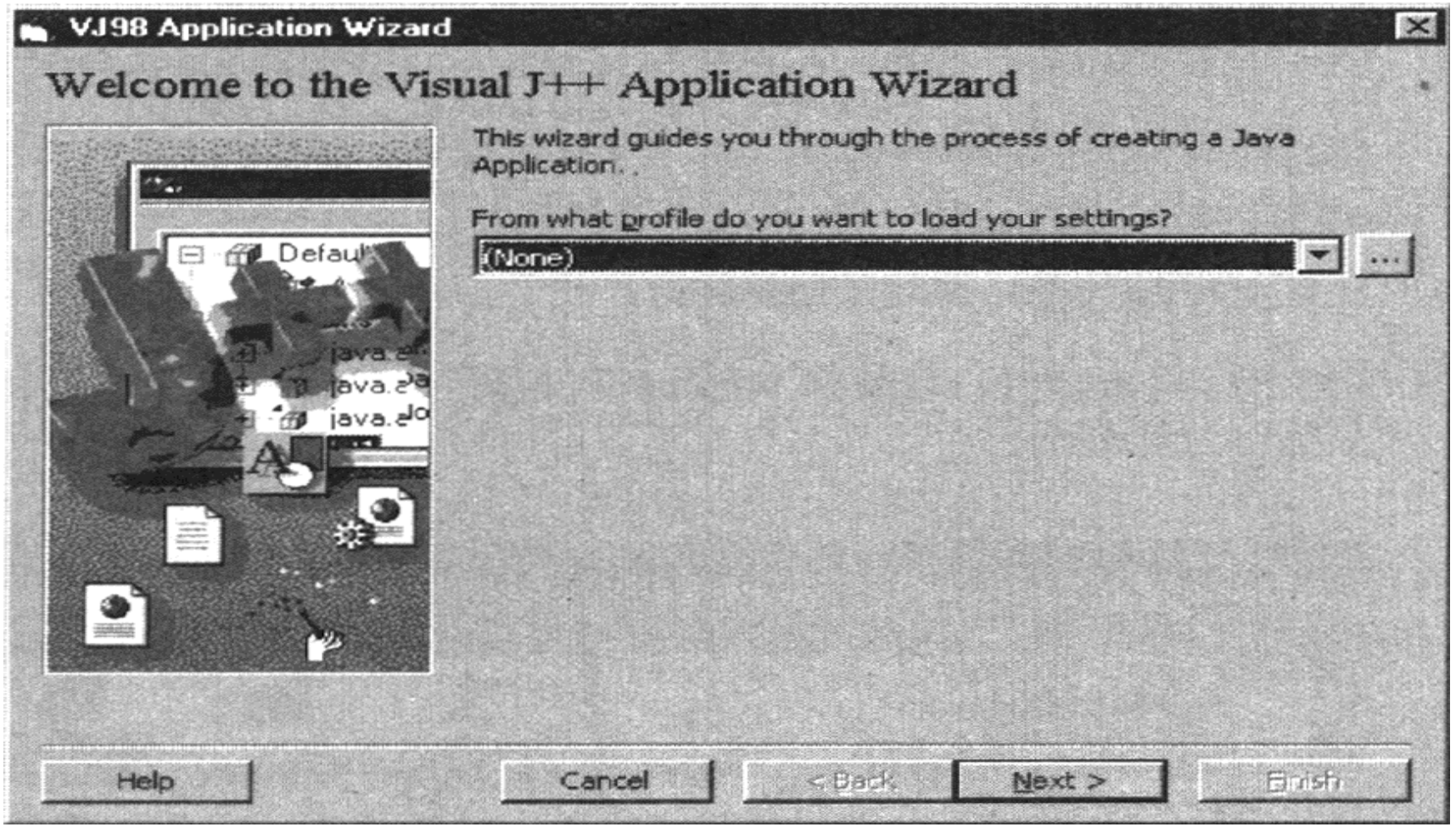
1. 在 File 菜单选择 New Project 项，屏幕上就会出现 New Project 对话框。
2. 在 New Project 对话框中选择 New 选项卡。
3. 在这个 New 页面上，选择 Visual J++ Projects/Applications 项。在窗口的右侧窗格中显示有应用程序的模板和向导。
4. 选择 Application Wizard。
5. 输入新的项目文件名称。对于本次练习，请输入“SimpleEditor”。



6. 单击 Open 按钮启动应用程序向导。

   应用程序向导会提供一个引导屏，在引导屏上可以选择一个配置文件，在配置文件中存储了应用程序向导提问的答案选项。在应用程序向导运行的最后，会提问是否存储设定的配置文件（profile）。既然本次练习是第一次使用应用程序向导，也就没有现成的配置文件。

7. 若要使用配置文件，可以在下拉文件列表中选择一个配置文件，也可以单击带有椭圆形点符的按钮，找到所要的配置文件。



8. 单击 Next 按钮进行到应用程序向导的下一步操作。

试验步骤

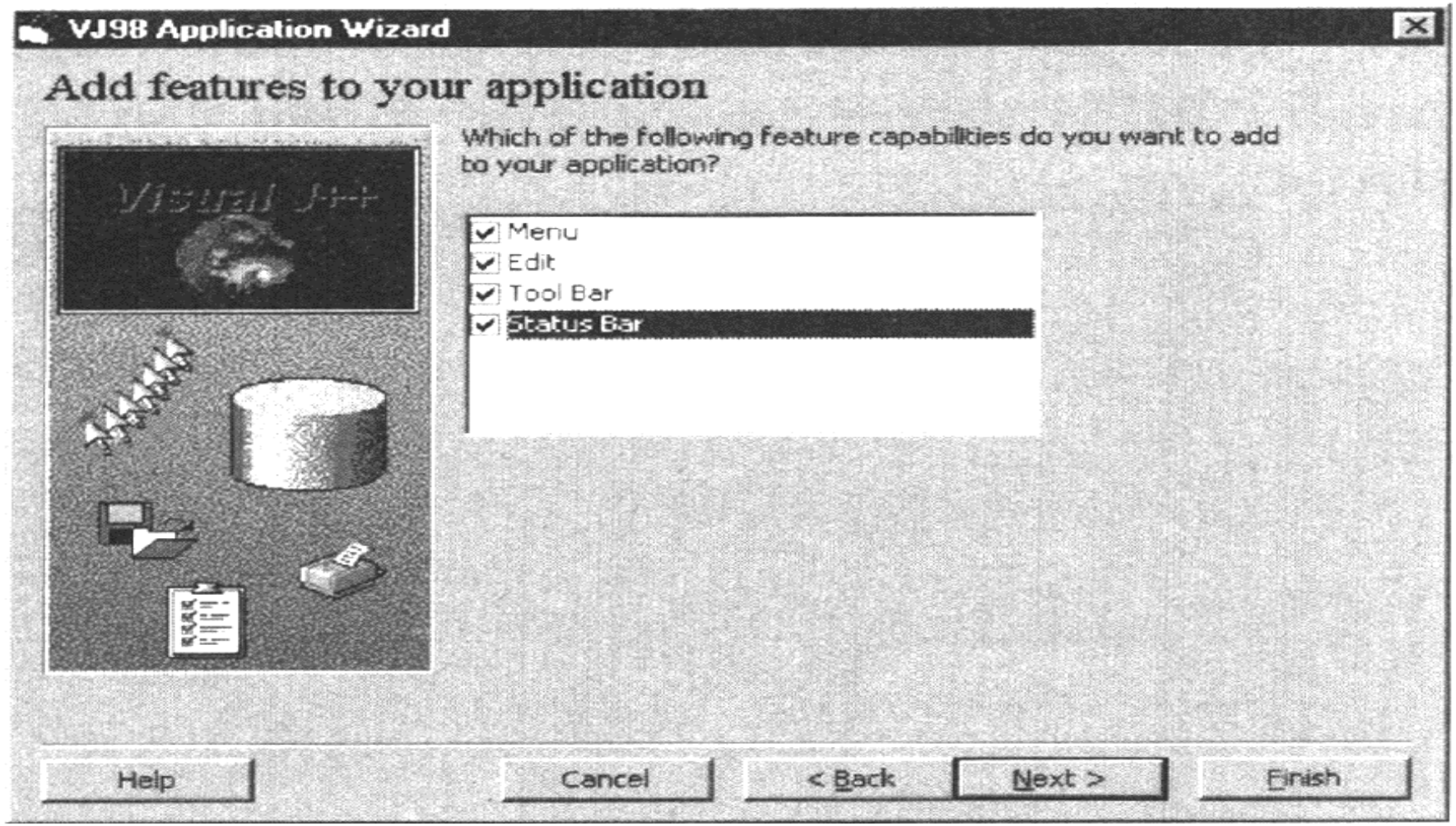
1. 为应用程序设定特征参数。

应用程序向导的下一步将询问有关应用程序的特征参数。

注意：如果使用的是专业版或企业版的 Visual J++，应用程序向导将首先询问应用程序的类型：Form-based 或 Form-Based with Data.标准版的 Visual J++没有数据库支持，因此在此处，对于标准版的 Visual J++，只能选择 Form-Based Application。单击 Next 按钮进入下一步：设定应用程序的特征参数。

需要特别指出的是，为应用程序设定特征参数有以下几个选项：

- Menu（菜单）：当此项被选中时，应用程序就会包括一个菜单。
- Edit（编辑）：选中此项，应用程序中就会有一个与 Notepad 类似的编辑控件。
- ToolBar（工具栏）：选中此项，应用程序中就会包括一个工具栏。
- Status Bar（状态栏）：选中此项，应用程序中就会包括一个窗格，此窗格随时显示 Num Lock 键或 Caps Lock 键是否设定生效。

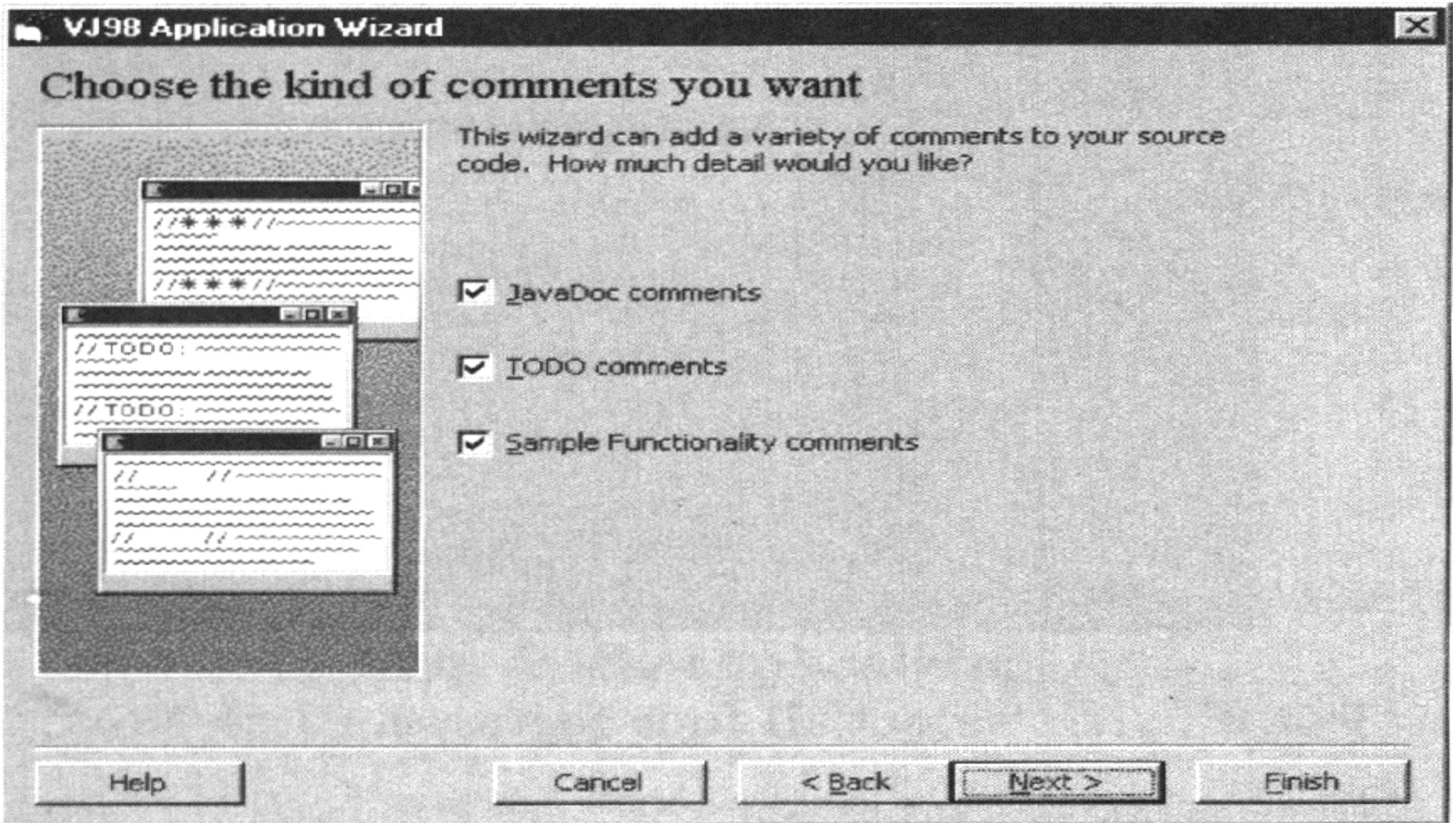


对于该练习，将以上四项全部选中，单击 Next 按钮进入下一步。

2. 选择注释类型。

在下一个步骤中，应用程序向导将询问应用程序在生成的代码中所要选择的注释类型。特别要指出的是，所供选择的注释类型包括以下几项：

- JavaDoc comments：选择此项时，生成的代码中包括 JavaDoc 注释，来说明应用程序中的类和方法。
- TODO comments：选择该项时，生成的代码中包括 TODO 注释，该注释出现在任务列表当中。
- Sample Functionality comments：选择该项时，生成的代码中包括其他类型的注释说明代码产生的各项功能。



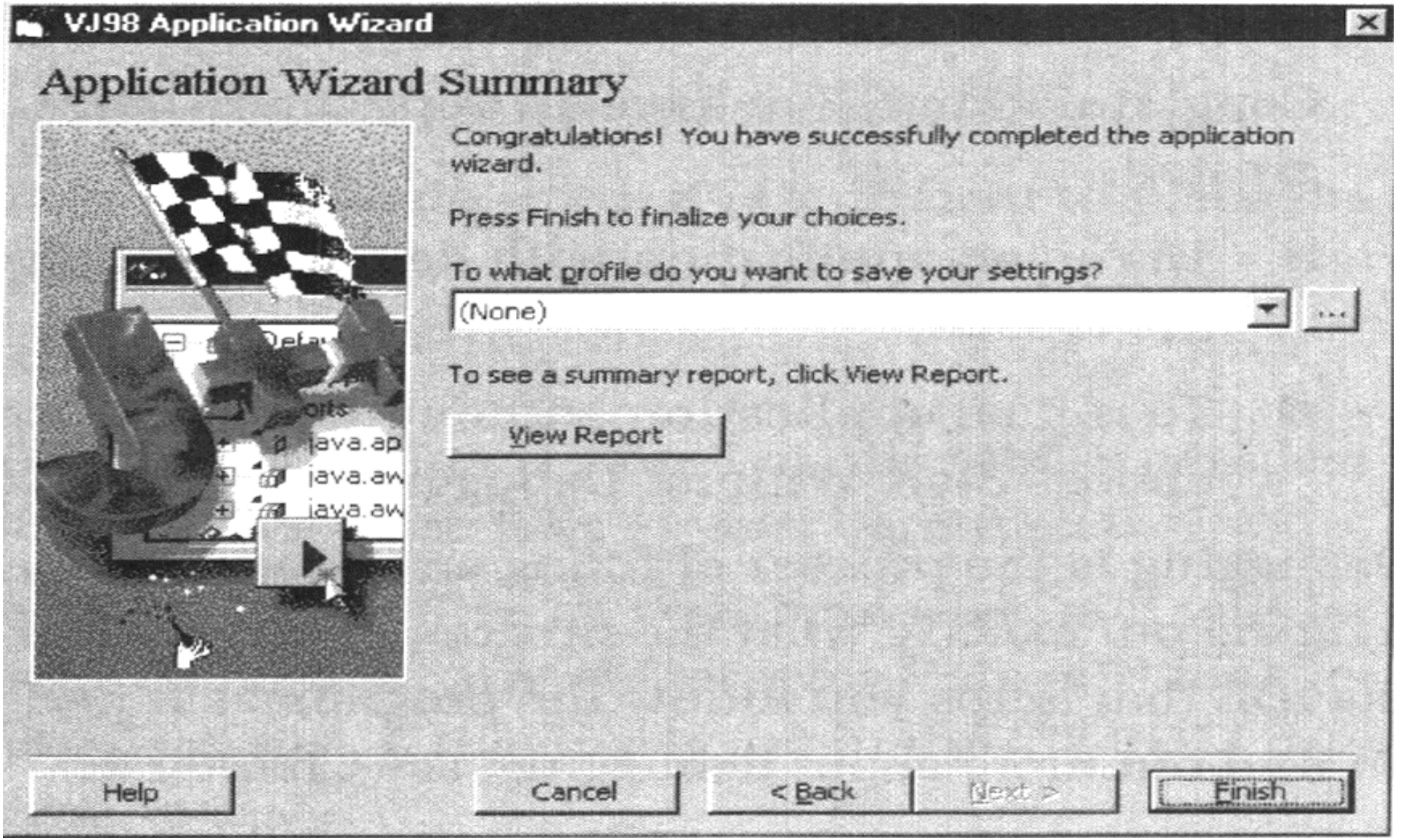
对于该练习，将以上四项全部选中，单击 Next 按钮进入下一步。

3. 查看设定的内容，然后，将所作的设定存储到 Application Wizard Summary，并创建应用程序项目文件。

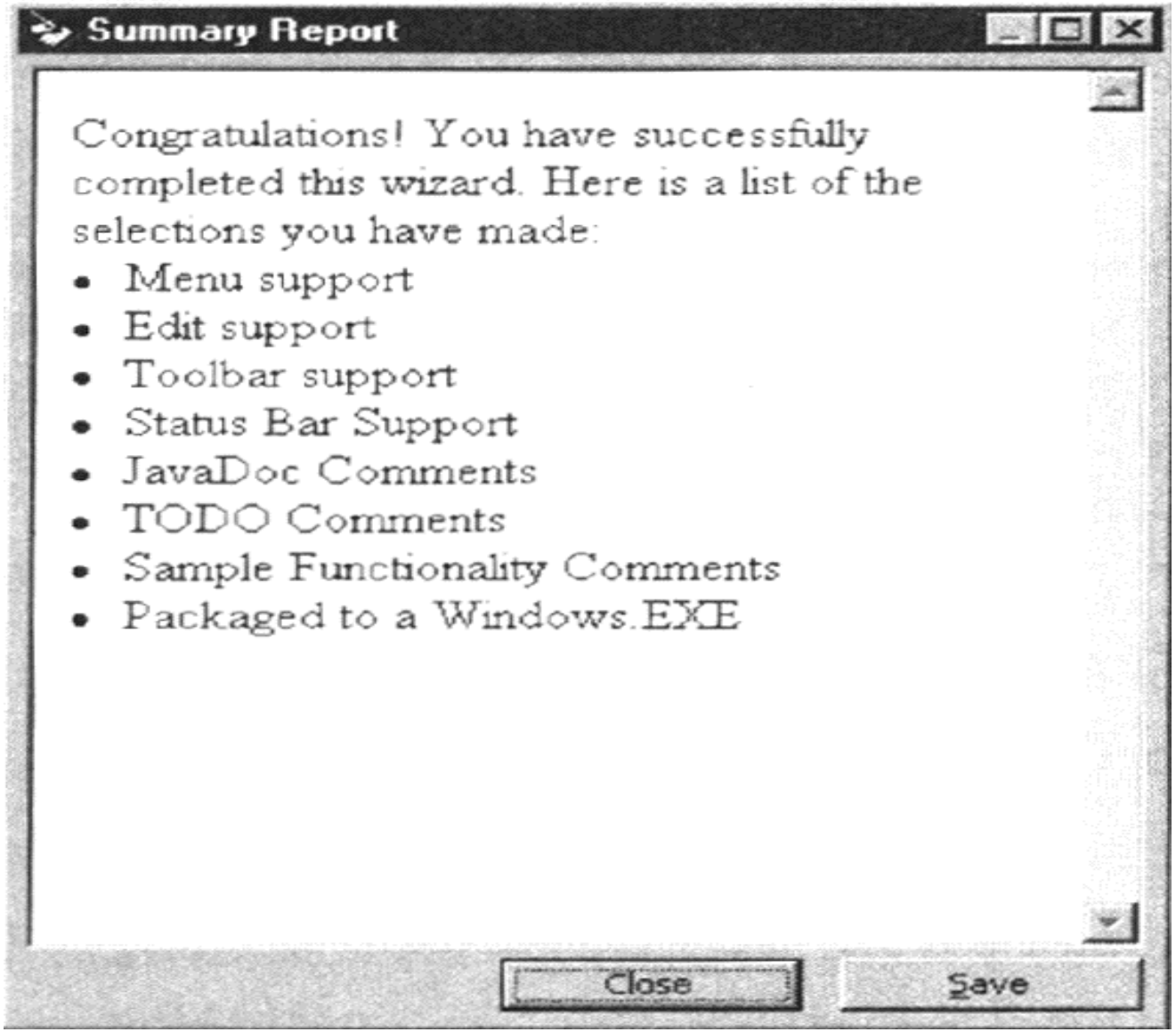
下面是应用程序引导用户要做的下一步内容。

注意：对于专业版和企业版的 Visual J++，此时应用程序向导就程序封装问题提问。标准版 Visual J++不支持程序封装，此时，可以单击 Next 按钮进入下一步。

- 若要将设定内容存储到已存在的配置文件中，可以从下拉式文件列表中选定配置文件。
- 若想将设定内容存储到一个新的配置文件中，可以单击带有椭圆形点符的按钮，并输入所要的配置文件名称。在本练习中不要采用这种方式。
- 查看所作的设定，单击 View Report 按钮。



可以将报表作为一个 HTML 文件存储在项目文件中。



- 要创建应用程序，单击 Finish。

在本次练习中，查看报表，然后创建应用程序。

4. 运行应用程序。

祝贺你！做了以上工作，你就拥有一个 Java 应用程序了。

- 若想在有调试支持的状态下运行一个应用程序，单击工具栏上的 Start 按钮。
- 若要在非调试支持的状态下运行一个应用程序，从 Debug 菜单选择 Start Without Debugging 项。

调试是编程工作中检查和修改错误的过程。在调试的过程中，往往需要清楚程序正在做什么工作，进行到了哪一步。调试器是 IDE 的一部分，它在调试的过程中逐行跟踪程序的运行。

关于调试的更详细的内容，将在本书的第二章中介绍。调试过程需要程序的许多信息，因此，在调试支持状态下运行，有时会比在非调试支持状态下运行速度慢一些。Debug 工具栏上的按钮是在非调试支持状态下运行应用程序。

- 要显示 Debug 工具栏，在 View 菜单中将鼠标指向 Toolbars 项，然后在弹出菜单中选择 Debug 项。或者，先右击菜单或工具栏，然后，在弹出菜单中选择 Debug 项。

Start Without Debugging 按钮呈暗红色，表示操作生效。其他 Debug 工具栏按钮将在第二章和帮助信息中详细介绍。

获取帮助

Visual J++拥有综合帮助信息。这些帮助信息可以归纳为以下几方

面的内容：

- 关于 IDE 的帮助信息。
- 关于使用 Visual J++编程的概念性资料。
- 使用 Java 语言和 WFC 的参考资料。

在对话框中访问帮助信息

- 在对话框中单击 Help 按钮即可。

访问关于 Java 关键字或 WFC 构造的帮助信息的方法

1. 选择想要获取相关帮助的名词。
   如果此词是一个词，将鼠标光标放在这个词内就可以了。
2. 按下 F1 键。

所有的帮助信息都存储在配套光盘的 MSDN 目录下，用户可以将这些信息拷贝到计算机的硬盘上，以便随时访问帮助信息。否则，每当访问帮助信息时，系统会自动访问光驱。如果配套光盘不在光驱中，系统会给出提示，提示用户放入配套光盘。

将 Help 文件安装到计算机硬盘上

1. 在 Control Panel 上选择 Add/Remove Programs 项。
2. 在程序列表中选择 MSDN Library-Visual Studio 6.0 项。

3. 单击 Add/Remove.
4. 在 MSDN Library-Visual Studio 6.0 的 Setup 中选 Add/Remove 项。
5. 在 MSDN Library-Visual Studio 6.0-Cusltom 中选 Master Index File,

VJ++6.0 Documentation 和 VJ++6.0 Product Samples.

6. 单击 Continue 按钮安装帮助文件。

7. 单击 OK 退出 Add/Remove Programs。

第二章　从头编写应用程序

前面，我们介绍了 Java 语言的开发环境和一些关键术语的定义。通过前面的学习，你应该对 6.0 版的 Visual J++ Java 语言开发系统有了初步的认识，甚至可以使用 Visual J++为 Windows 操作系统创建简单的应用程序。我们说 Visual J++是一种强有力的而且易于掌握的语言开发工具，那么，究竟是什么使得 Visual J++如此方便快捷，它具有哪些具体特征呢？我们将在本章的内容中介绍。

本章的具体内容包括：

- 设计和创建应用程序。
- 处理事件并编写事件过程。
- 在应用程序中显示图片。
- 在应用程序中添加颜色。
- 在 Visual J++ Java 程序中使用注释。

本章的练习涉及以下的内容：

- 属性。
- 事件处理程序。
- if 语句。
- 注释和 TODO 标记。
- Visual J++调试器。

创建简单的窗体

在前面的章节中，我们利用应用程序向导创建了一个简单的应用程序。下面，我们将学习使用窗体模板创建一个基本窗体（base form）。与应用程序相比，窗体似乎具有可以由用户控制的更多要素，我们只考虑应用程序所需要的部分就可以了。

现在，打开计算机，按照上一章所讲述的办法启动 Visual J++。

使用窗体模板

如果是第一次启动 Visual J++，屏幕上会显示如图 2-1 所示的 New Project 对话框。若是屏幕上没有显示此对话框，只需在 File 菜单上选择 New Project 项即可。

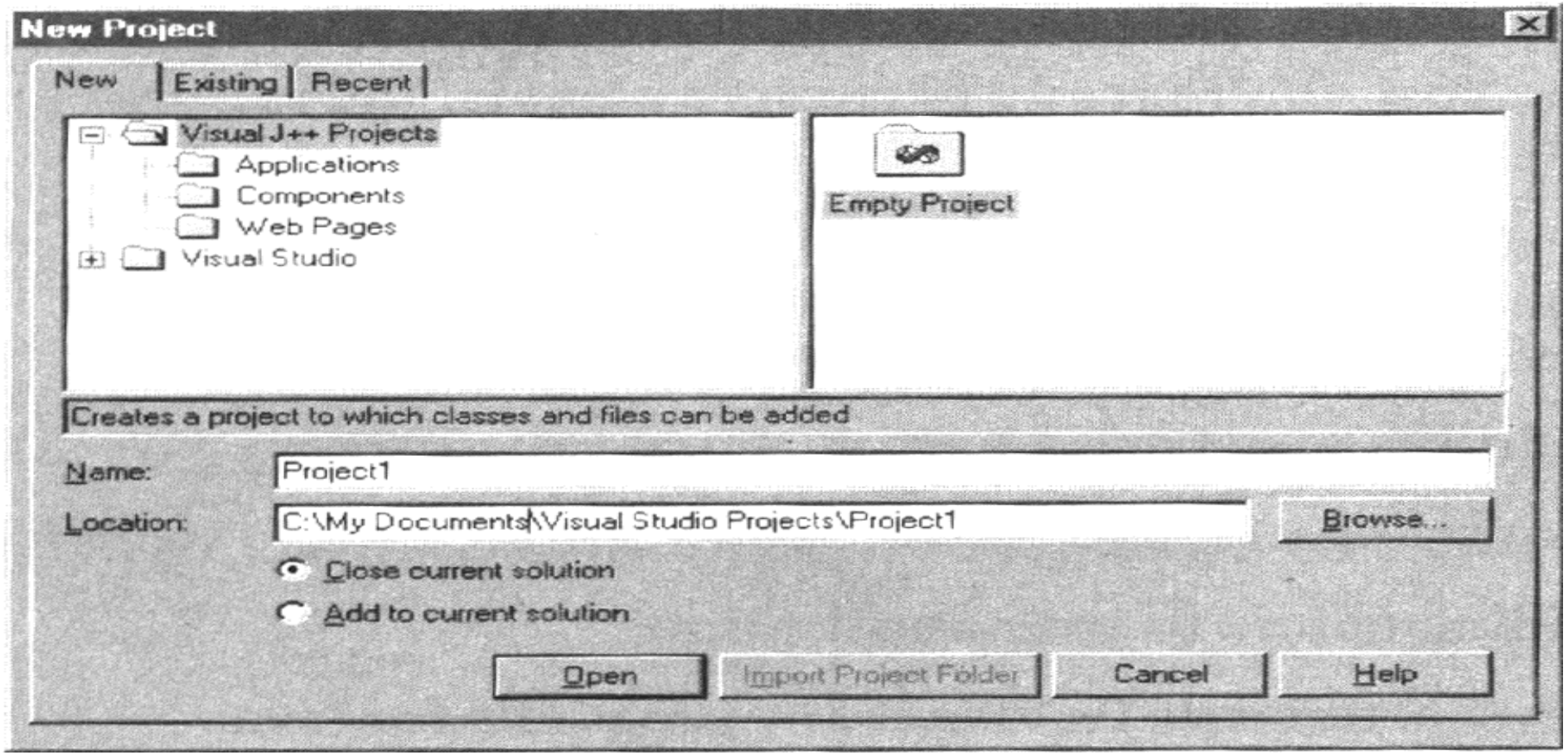


图 2-1　启动 Visual J++时屏幕上显示的 New Project 对话框

如图 2-1 所示，在左侧的窗格中有 Visual J++ Project 文件夹，在它的下面是应用程序 Applications，Components 和 Web Pages 子文件夹（如果此时窗格中并没有显示出这些文件夹的话，可以单击 Visual J++ Project 文件夹左侧(+)标记，展开文件列表)。选择 Application 子文件夹。

在右侧的窗格里有三个图标，其标签分别为：Windows Application，Console

Application 和 Application Wizard。在第一章的学习中，我们选择了应用程序向导（Application Wizard），这次，我们选择 Windows Application。当选择了 Windows Application 之后，在窗口中间的状态栏中会显示如下信息：

Creates an application which uses the Win32 user-interface and hosts controls（创建使用 Win32 用户界面和主控件的应用程序）

可以在标有 Name 的框中输入合法的 Windows 文件名，以此作为项目文件的名称，这里，将该项目命名为“Hello”（无引号标记）。在标有 Location 的框中输入合法的 Windows 路径名，该项目文件就将保存到此路径。用户可以将项目文件存储在磁盘上的任意位置。这里，将项目文件存储在 My Documents 文件夹中。输入了项目文件名称和位置以后，就可以按下 Open 按钮向下进行。

注意：此处按下 Open 按钮，将在硬盘上创建一个文件夹和三个文件。这个文件夹的名称与项目文件的名称相同（均为 Hello），在 Hello 文件夹中的三个文件分别为：项目文件本身（Hello.vjp）、缺省的 Java 源代码文件（Form1.Java）和 Visual J++ Java 内部使用的数据文件（codebase.dat）。其中，Java 源代码文件中保存 Visual J++ Form Designer 的代码，这些代码用以支持将要创建的窗体。

按下 OK 按钮后，屏幕上会出现如图 2-2 所示的画面。

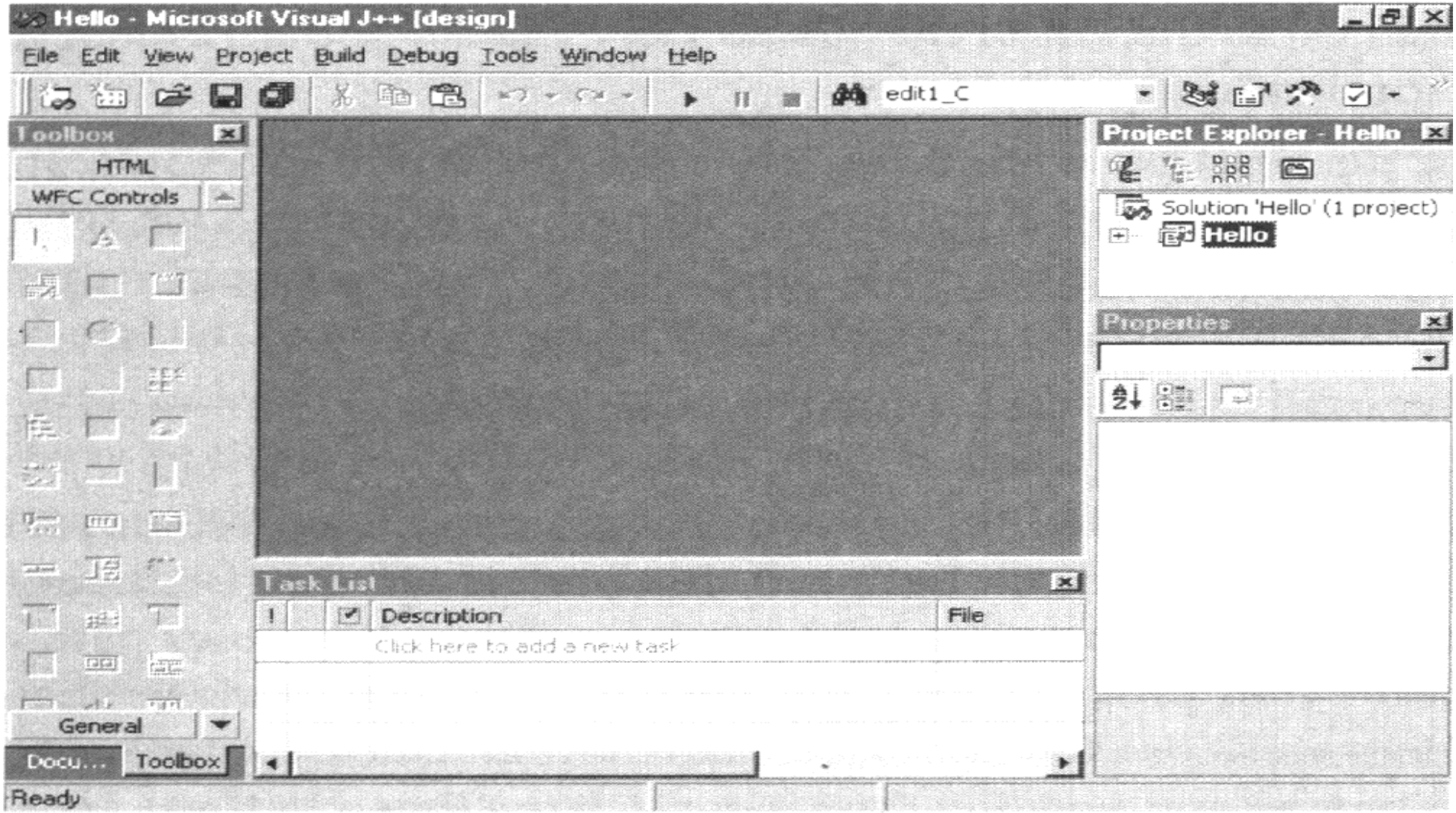


图 2-2　Visual J++准备好可以用来创建基于 Windows 的应用程序

在窗口 Project Explorer 中，可以看到一个解决方案和一个项目文件，它们的名称均为 Hello，当用户保存自己的工作时，解决方案会保存在一个名为

Hello.sln 的文件中，此文件与项目文件（项目文件的扩展名为.vjp）一起放在同一文件夹中。在 Project Explorer 窗口中，这些文件并不显示出扩展名。

展开 Hello 项目文件列表。选择名为 Form1.java 的源代码文件。在 Project Explorer 下端的工具栏上，有一系列的按钮，这些按钮可以实现查看 Project Explorer 中列出的项目信息。单击左数第二个按钮，即 View Designer 按钮（注意，当将鼠标指向某一个按钮时，会出现一个工具提示文本，其中显示有关该按钮功能的信息)。View Designer 会显示由 Visual J++创建的空白窗体（该窗体应用的是缺省窗体模板），这时，就可以选择 Windows Application 来组建当前应用程序（见图 2-3）。

如果现在单击 View Code 按钮，就会出现一个窗口，在该窗口中，显示了 Form1.java 中为创建以上的空白窗体而产生的 Java 代码。可以看到，只是创建一个窗体这样的简单工作，Visual J++就要处理相当数量的 Java 代码。

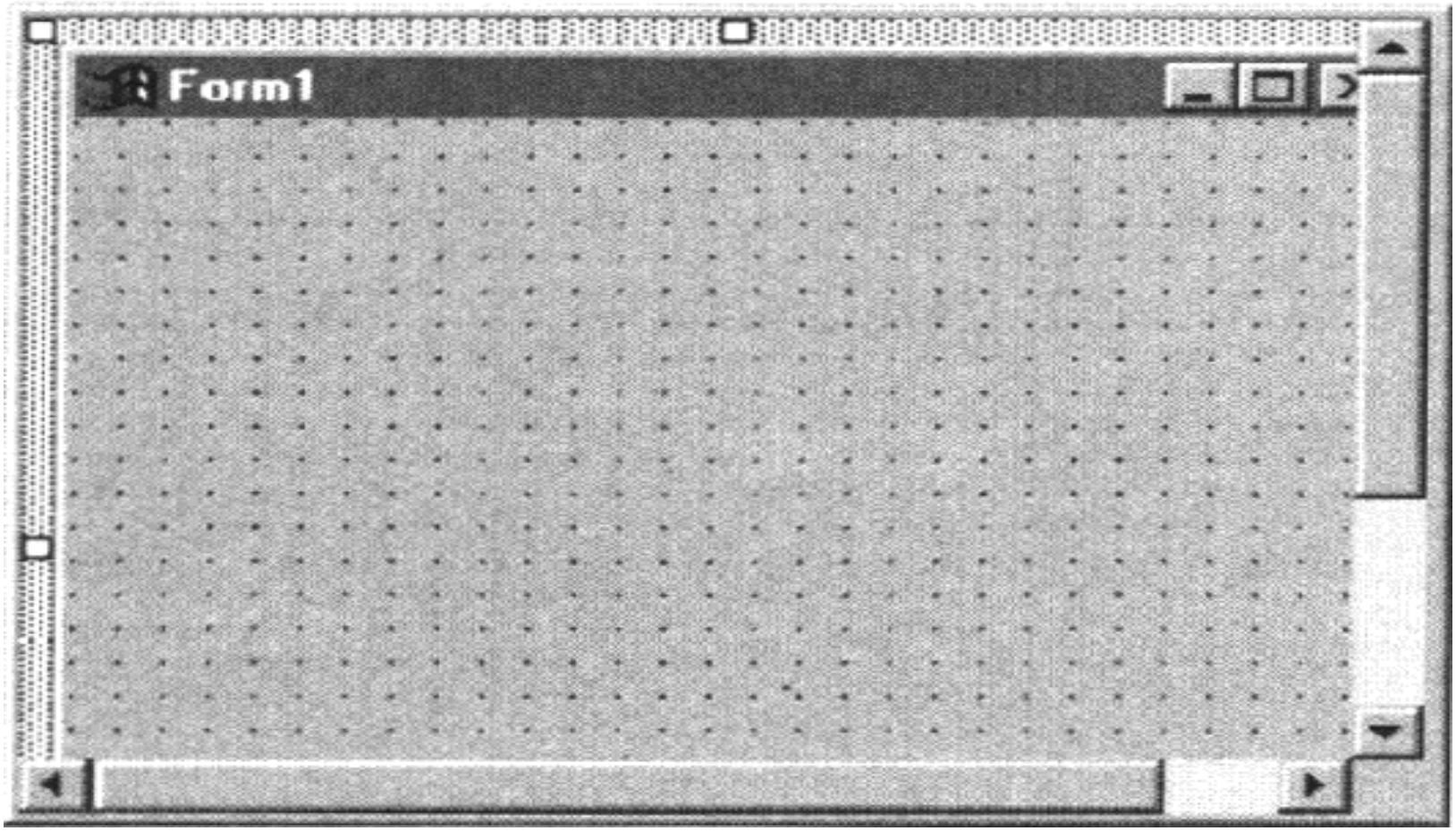


图 2-3　Form1.java 视窗

为窗体添加控件

窗体创建工作完成以后，就可以使用 Visual J++来修改窗体了。

在向窗体中添加控件之前，应确认当前屏幕上是否显示了开发应用程序所需的一些重要窗口。检查开发环境，确认以下窗口可视。

- Project Explorer（项目文件开发程序）
- Properties（属性）
- Toolbox（工具箱）
- TaskList（任务列表）

如果以上四个窗口有看不见的，则应设法使其变为可视的窗口。具体的做法很简单，只需打开 View 菜单，选择相应窗口的名字即可（任务列表窗口可以在其他窗口的子菜单中找到)。

单击 View Designer。注意 Properties 窗口，该窗口中显示了窗体 Form1 的属性或设置。一旦某个窗体成为活动窗口，就可以在此处看到它的属性。

在 Toolbox 窗口中单击 WFC Controls 按钮，屏幕上会出现一个控件列表，可以将列表中的各个控件添加到窗体中（见图 2-4）。使用上下箭头键滚动列表。

我们将在窗体中加入两个标签、一个编辑框和一个按钮。向窗体中添加标签的具体操作为，在工具箱选中标签，用鼠标将其拖动到窗体中，此时，一个具有轮廓的标签就会出现在窗体中。注意 Properties 窗口中的变化。这时，Properties 窗口中已经不再显示 Form1 的属性，而是显示标签 Lable1 的属性。若用户不作改动，所有添加到窗体的控件都将以这种方式命名（即控件名开头，其后接数字)。现在，我们就使用 Visual J++所提供的缺省名称而不作改动。再拖动一个标签置于 Lable1 的上方。这个标签自动命名为 Lable2。然后，再向窗体中添加一个 Edit 控件和一个 Button 控件，这时窗体就如图 2-5 所示。注意，Properties 窗口中的信息总是与最后添加的 WFC 控件相匹配。

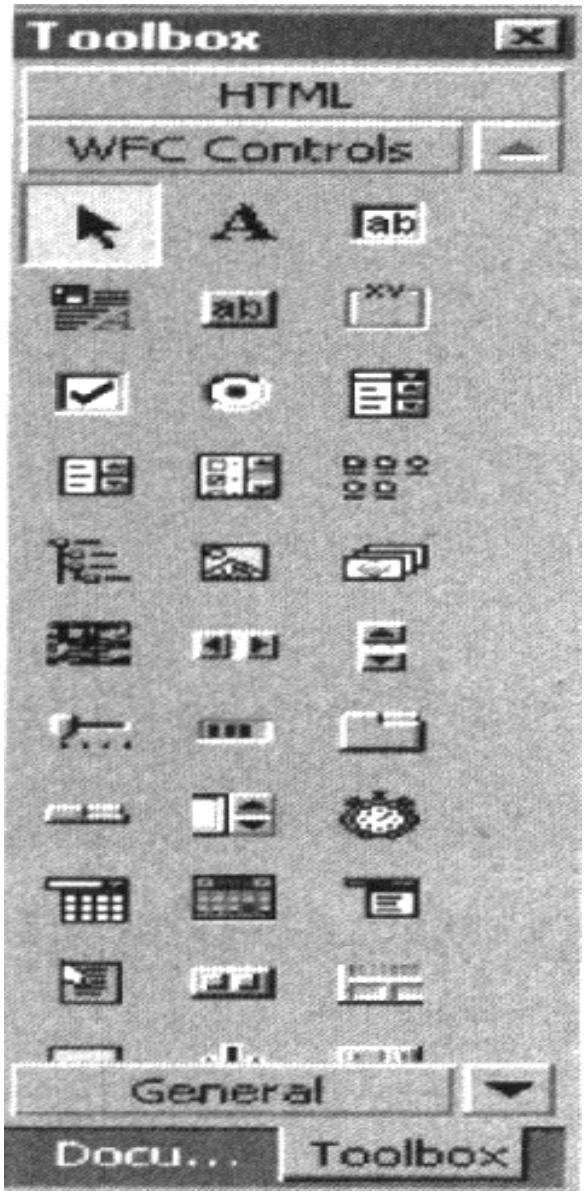


图 2-4　带有 WFC 控件选择的工具箱

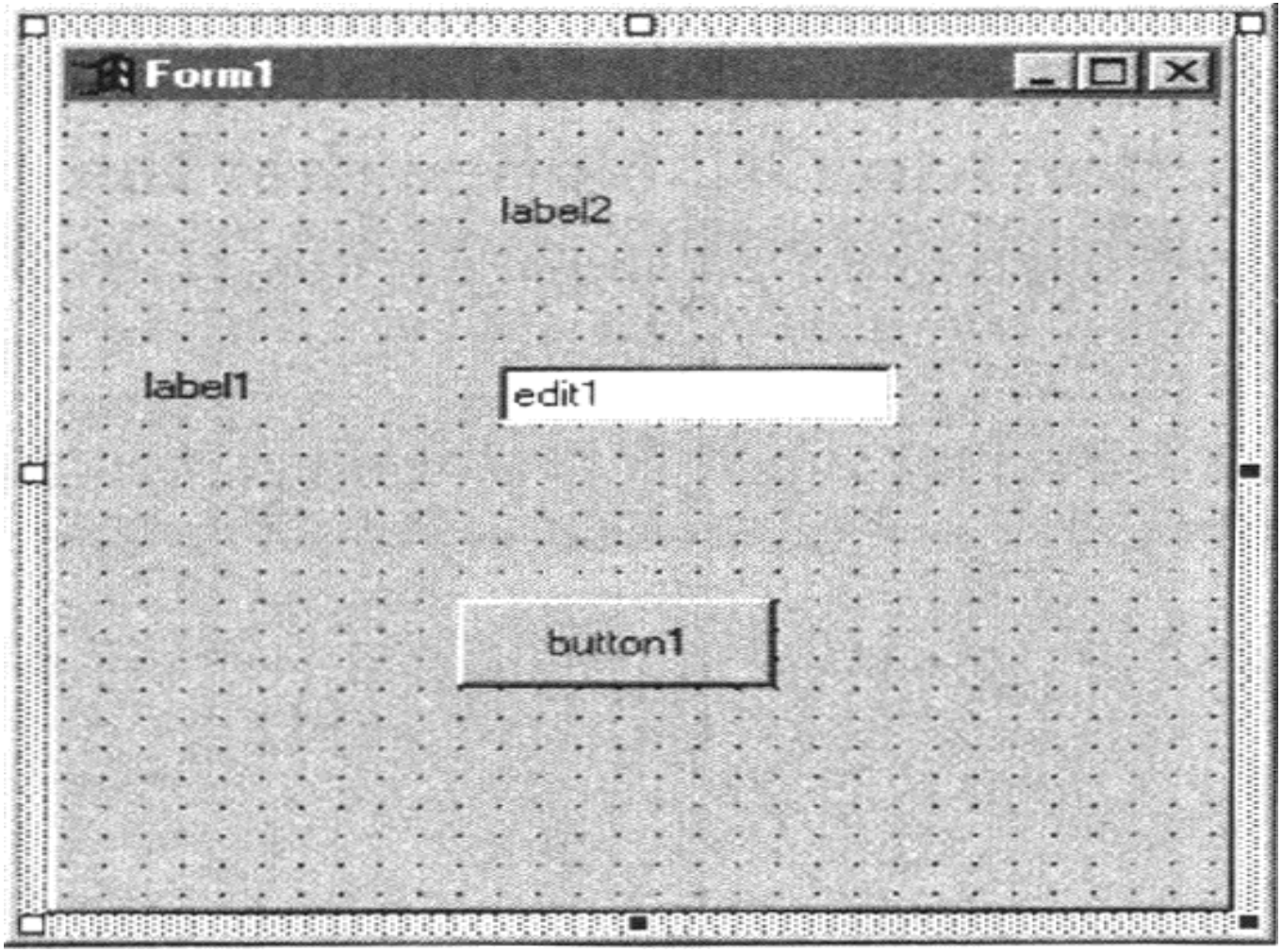


图 2-5　添加了控件的窗体

设置属性

我们已经了解到，当每次向窗体中添加新的控件时，Properties 窗口是如何

反应的，即窗口中显示了控件的属性、特征以及与项目文件中所有其他组件的联系。如果最后向窗体中添加的控件是 Edit 控件，那么，此时 Properties 窗口中显示的信息将是该控件的属性（edit1 的属性）。

Button1

在 View Designer 窗口（当前标题为 Form1.java）中单击（不可双击）名为 button1 的按钮。使用 Properties 窗口的滚动条查看关于该按钮的所有属性。所有按钮的属性都是同样的，当然，每个按钮都有自己特有的按钮值，在该窗体中，就不可能再有其他按钮以 button1 命名。

这个按钮名字的属性值为 Name。用户也可以为 Button1 取一个新名字。此时，我们还是采用系统提供的缺省名称，而对按钮的标题（caption）作一些改动.，按钮控件的标题属性称为 Text。要改变一个控件的属性，可以单击属性名称右侧的区域。将 button1 的 Text 属性改为 OK，可以看到，窗体中按钮的标题立刻就发生了变化。

提示：如果用户想要了解有关某个属性的信息，可以选定 Properties 窗口，在 Properties 窗口的底部，可以看到关于该属性的描述。

Label1 和 edit1

Properties 窗口显示了窗体内所有控件的属性特征，显示的方式为每次一个控件。如果想要查看其他控件的属性，只需单击 Properties 窗口顶部的向下箭头键，就会弹出一个下拉菜单，菜单上显示了用户在创建该窗体时的所有控件的名称，窗体的名称也在菜单上。

选择 label1。可以发现称为文本（Text）的属性，然后，将它的属性值由 label1 改为 Name:，再返回到 Properties 窗口的下拉菜单选择 edit1，选定该区域的值，然后按下 Delete 键，这样，edit1 的文本属性就被改为空。

Label2

单击窗体 Form1 上的 label2 控件。选择文本（Text）属性，在下面输入您的名字，然后，单击加号标记展开字体（Font）属性。在字体（Font）属性的下方会出现一个格式选择的列表。将字号大小（Size）设定为 24。单击 Weight 区域的下拉箭头，来查看该区域的值。将 Weight 设定为 Extrabold。

既然 label2 的文本比原来大了，而且加长了，那么，原来的位置就不能容纳它，此时应在 View Designer 窗口拖动标签轮廓的控制柄，直到标签中的文本内容在窗体上全部显现出来为止。此时用户所创建的窗体如图 2-6 所示。

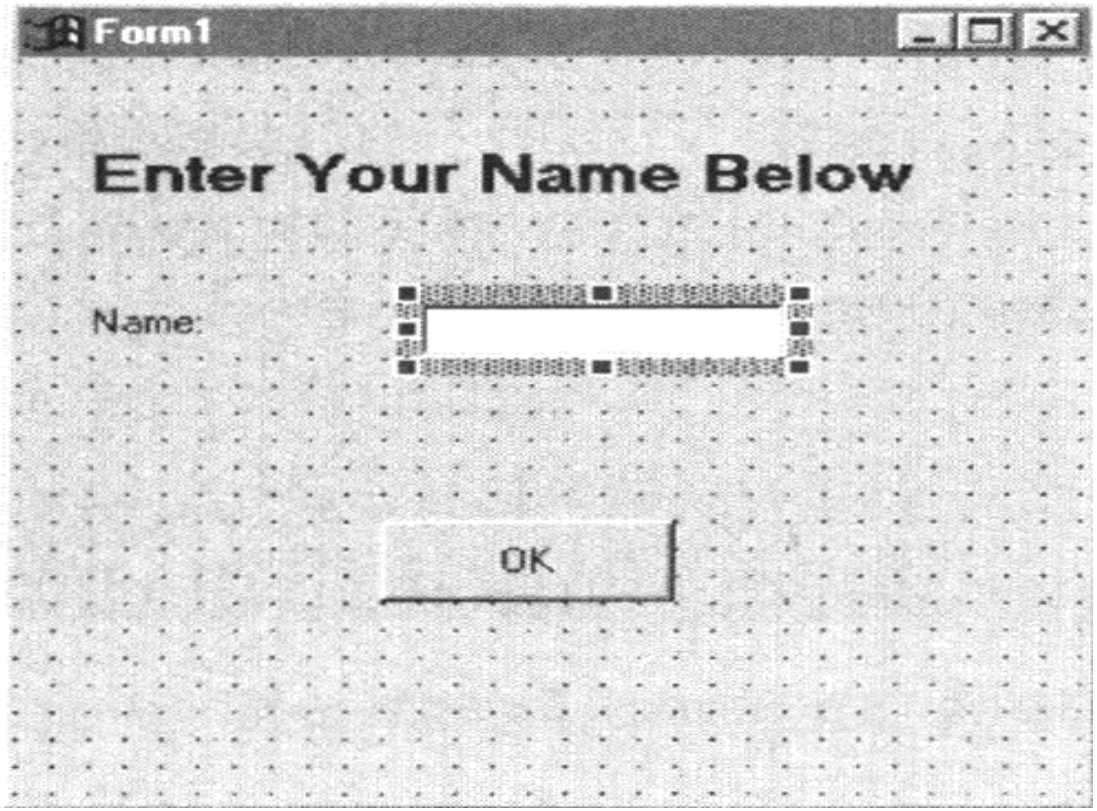


图 2-6 完成了控件的位置设定和属性设定的窗体

添加事件处理程序

现在，我们已经建立了一个窗体，下面可以在其中添加一些操作。正如每个窗体都具有事先设定的可修改的一系列属性一样，窗体中也具有事先设定的窗体可响应的一系列事件。当一个控件第一次安放在窗体中时，该控件对任何事件都是没有响应的。我们必须确定当程序在运行时某一个事件发生，窗体、控件应作出的响应是怎样的。

在一个窗体中，为了实现控件对某个事件的响应，例如，用户移动一下鼠标，我们需要在程序代码中加入一个事件处理程序（或者称为事件过程）。对于大部分一般的事件与控件的联系，Visual J++实现起来都是十分容易的。

Button1

在 View Designer 窗口中，用鼠标左键双击 button1 按钮，就可以为其添加一个事件处理程序。双击 button1 按钮，将进入 Visual J++窗体的代码窗口，以及针对该控件的事件处理程序的梗概代码。代码如下：

```
private void button1_click(Object source, Event e)
{

}
```

在大括号内({, })放置插入点，并在此处输入程序代码，在应用程序运行过程中，用户单击鼠标键时将执行这些代码。当用户用鼠标单击 Button1 按钮时，被执行的事件处理程序的名称为 button1_click。当用户建立一个事件处理程序的梗概时，Visual J++总是以这种命名方案为其命名：首先是控件的名字，后面紧接符号“_”，最后是事件的名称。

当用户单击 Button1 按钮时，会发生什么呢？我们可以添加一些代码，这些代码能够读入用户输入的名字，并将这个名字显示在窗体上。代码如下：

```
private void button1_click(Object source, Event e)
{
```

```
String name = edit1.getText();
Label2.setText(″ Hello″  + name);
 }
```

提示：当用户输入这些代码时，屏幕上会出现 Java 语句和列表，这是 Visual J++环境下的语句补全特征，为用户学习 Java 语言的语法以及一些特殊语句的选择提供建议。这样，用户就可以不必事先去阅读大量的资料，从而可以节约很多时间。在输入代码的过程中，如果发现代码下端有红色的波浪线，就说明该代码有错误，要及时修改。

存储并运行应用程序

存储项目文件

现在，用户可以将项目文件存盘了。单击工具栏上的 Save 按钮，在 File 菜单中选择 Save Form1.java 即可。另外，也可以使用 Save All 命令，该命令将为用户存盘所有与该项目有关的信息。如果用户想要将窗体另存为其他不同的名称，那么，就在 File 菜单中选择 Save As（另存为）选项，在对话框中输入想要的名称即可。下面将继续学习如何使用 Form1.java 项目文件。

运行应用程序

完成了以上的工作，现在就已经具备了运行应用程序的条件了。单击工具

栏上的 Start 按钮，或者在 Debug 菜单中选择 Start Without Debugging 选项。图 2-7 显示了该应用程序运行过程当中屏幕上出现的 Hello 程序窗口。

在标有 Name 的方框中输入名字，然后单击 OK 按钮，就会发现窗体顶部的标题变为“Hello *yourname*”。这样，就从头完成了第一个 Java 应用程序的创建。如果想要定制应用程序，只需单击程序窗口标题栏上的 Close 按钮，系统会返回 Visual J++环境。

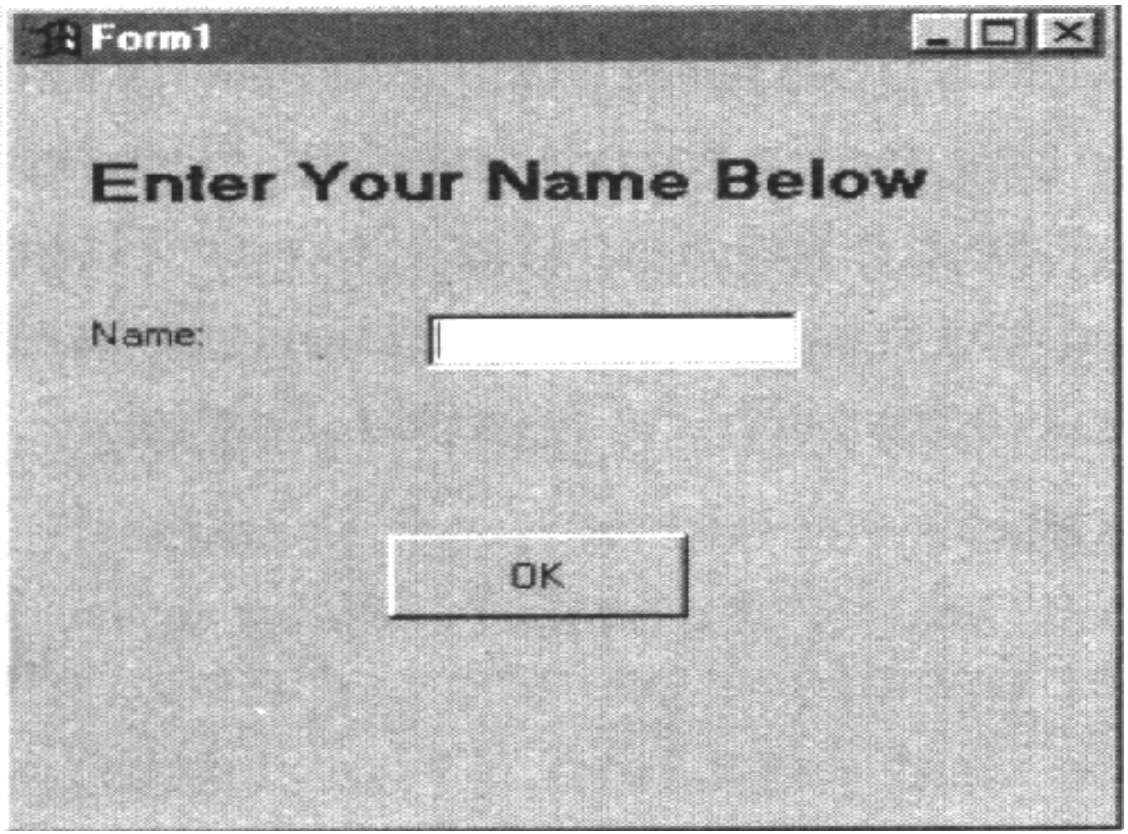


图 2-7　Hello 应用程序在运行中，等待输入

应用程序的执行进度

对用户来说，编写计算机程序就是解决一系列问题的工作。解决问题，对于我们来说，就是要回答两个基本的问题：首先，要清楚要解决的是什么问题，然后，要弄清我们如何从用户那里得到解决该问题所必须的相关信息。也就是说，我们设计的应用程序要符合用户的需求，创建的窗体要与用户交互信息。一旦设计好了应用程序和窗体，就必须添加一些针对控件和窗体的事件处理程序，以与之相匹配，实现用户与应用程序之间的交互。

设计应用程序

首先，应该弄清要解决的问题。对于本书中的示例，应用程序要用用户的名字向用户致意。为了做到这一点，当然，先要设法得到用户的名字，并使用该名字向用户致意，然后，在屏幕上显示结果。自然，我们决定使用 Visual J++窗体，该窗体当中具有一些控件，以实现应用程序与用户的交互。窗体是设计和创建用户界面与应用程序功能的可视化方法。在这些可视的窗体的背后，是一系列源代码，因此，不创建窗体，而通过编写源代码建立应用程序，这也是可行的。但是，能够通过创建窗体来建立应用程序，是 Visual J++开发环境的一个主要特征。

设计窗体

现在，我们已经确定了要待解决的问题，然后，就可以将注意力转移到如

何解决这个问题上了。我们需要从用户那里得到什么信息，怎样使用户知道我们需要哪些信息，我们如何向用户作出回答，以及在解决该问题的各个控制过程当中，又需要哪些信息。这都是我们需要完成的，下面是实现上述的功能而可以使用的基本控件：

- 告诉用户所需哪些信息的方法（可以使用标签（label）控件）。
- 从用户那里得到所需文本的方法（可以使用编辑（edit）控件）。
- 识别用户输入信息所在区域的方法（使用另一个标签（label）控件）。
- 用户输入信息已经完成的判断方法（使用一个按钮（button）控件）。

当将这些控件拖到窗体，并设定好位置以后，就要对这些控件设定属性了。为各个控件设定属性的目的在于，使它们在程序的运行过程中，在指定的时候以指定的方式出现。然后，就可以通过添加事件处理程序的方法，在应用程序的执行过程中操纵各个控件。

事件处理

到目前为止，我们只添加了一个事件处理程序 button1_click。一旦事件“单击按钮 button1”发生时，就将执行这个事件处理程序。下面，让我们了解一下事件的全过程：每当用户单击按钮 button1 时，事件“Click”的信息就会传输到程序（因为用户已经事先编写了针对“Click”事件的事件处理程序）和代码，这时，事件处理程序“button1_click”就会运行。

下面是“button1_click”事件处理程序的代码：

```
private void button1_click(Object source, Event e)
{
```

```
String name = edit1.getText();
Label2.setText(″ Hello″  + name);
 }
```

到目前为止，用户已经了解了应用程序的运行情况，下面，让我们看看程序源代码中发生了什么事情。“button1_click”事件处理程序修改了标签控件“label2”。控件 edit1 的文本内容是用户输入的字符串。由 Java 方法 getText 取得控件 edit1 的文本值，并将其存储到局部字符串当中，变量名为“name”。然后，让我们使用 Java 方法 setText 来设定 label2 的文本属性，将“Hello”加入名为 name 的字符串中去。当我们的学习向下进行时，我们会接触到更多关于 Java 方法的示例，在第四章中有更为详尽的叙述。

实验 2-1：修正 Hello 代码

在本次实验中，我们将要把代码放入另外的事件处理程序中，看看事件处理程序是如何影响应用程序的运行的。

实验概述

在本次实验中，用户将实践使用 View Designer 来创建事件处理程序。在前面所讲的 Hello 应用程序中，用户必须按下 OK 按钮，才可以看到窗体上端的欢迎词。我们可以向 Edit 控件添加一个事件处理程序，该程序可以使应用程序实时更新欢迎词的内容，而不必等待用户更多的输入以及按下 OK 按钮。

试验准备

1. 如果已经按本章前面示例中的内容进行了有关操作，那么，实际上已经完成了安装，可以跳过这部分内容向下进行。
2. 否则，首先启动 Visusl J++。
3. 打开实验 2-1 的项目文件（Chapter02\Lab2-1\Hello.sln）。
4. 查看 Project Explorer 窗口，如果在项目文件图标下没有发现任何东西，则要单击加号标记，以显示与该项目文件相联系的文件。
5. 选择 Form1.java 项，并且在 Project Explorer 窗口的顶部单击 View Code 按钮。

实验步骤

1. 在 View Designer 窗口中，双击 Edit 控件。这样就可以打开该窗体的代码窗口，在此，可以为 Edit 控件创建一个事件处理程序，其名称为 edit1_textChanged。代码显示如下：

```
private void edit1_textChanged（Object source，Event e）
{
}
```

该事件处理程序在每次改变控件 edit1 的属性都将被调用。再向 edit1_textChanged 中加入以下代码：

```
String name =edit1.getTedt();
Label2.setText(“”Hello”+name);
```

2. 再次运行应用程序。试试看，是否仍需要按下 OK 按钮。
3. 将本次实验的工作与本章中的示例“Chapter02\Sol2-1\Hello1.”比较一下，看看有什么不同。

下一步

到目前为止，我们已经学习了基本应用程序的设计方法。并且已经设计了一个应用程序来解决问题；创建了窗体，以实现与用户的交互；并为在应用程序中发生的事件配置相应的事件处理程序。这样，一个应用程序的基础就已经完成了，下面就可以使用逻辑决策、颜色和图片，对其进行适当的修改。

增强应用程序的功能

到目前为止，我们已经在 Visual J++环境下从头编写了一个基本的应用程序。但是，这样的应用程序仍有许多地方需要改进。下面的内容就将介绍如何改进应用程序，我们从引入 if 语句开始，以使应用程序更为“聪明”，来学习这部分内容。

添加逻辑决策（Decision Logic）

对于前面所设计的应用程序，当我们事先并未输入名字而直接按下 OK 按钮时，会怎么样呢？如果这样的话，在欢迎信息中就只会显示“Hello”。这样，如果应用程序可以检测出用户是否输入了名字，然后，对检测的结果采取适当

的行动，这是十分有用的。在应用程序中，实现以上所说功能的代码具有一个 if 语句，具体代码如下：

```
 private void button1_click(Object source, Event e)
 {
String name = edit1.getText();
If (name.equals(″ ″ ))
{
    label2.setText(″ You Must Enter a Name″ );
}
else
{
    label2.setText(″ Hello ″  + name);
}
 }
```

以上这些新的代码，若将其翻译过来，大意是这样的：“如果用户没有输入任何信息，那么，提示用户输入名字；否则，更改标签 label2 的文本属性值”。下面就应该修改 button1_click 事件处理程序，使其包括以上这些新的代码。下面介绍新添加代码中的语句。

If 语句和 Boolean（布尔逻辑表达式）

利用 Java 中的 If 语句，程序员便能够检测条件和布尔逻辑表达式。布尔表达式的值只有两种可能：真“true” 或者假“false”。如果表达式的值为真，

则在第一对大括号（{}）中的代码将执行；如果表达式的值为假，则位于“else”之后的在第二对大括号（{}）中的代码将执行。在下一章中，将详细介绍布尔数的有关知识。

在这个示例当中，布尔表达式为：

```
name.equals（”  ”）
```

这个布尔表达式比较字符串变量 name 与“空”字符串的值。“空”字符串即为不含有任何字符，而只是一系列空的引用标记的字符串，也称作“空串”。布尔表达式中的等号是实现字符串 name 与空字符串比较的方法。下面的一行语句可以实现以下的功能：如果用户在 Edit 控件中没有输入任何信息，程序可以通过改变标签控件 label2 的文本属性来警告用户，提示用户输入名字。查看附录中 Java 语法的简要总结，其中包括 if 语句。

使用颜色

使用颜色是在制作用户界面时的一个强有力的工具，它可以在很大程度上改变一个用户界面的外观。大多数 Visual J++都具有颜色属性，设计者可以使用 Properties 窗口，在应用程序组建时（设计时），或者是在程序运行时，更改这些属性，以改变显示的颜色。在下面的示例中，将介绍如何在设计时和运行时改变窗体背景的颜色。

打开或切换到 Properties 窗口，选择窗体 Form1，找到“backColor”属性。单击当前属性右端的下拉箭头。屏幕上会出现一个具有两个选项卡（系统“System”和调色板“Palette”）的窗口。单击 Palette 选项卡，并选中一种颜

色（比如暗绿色）。这时，你就会发现，窗口背景的颜色会随着不同的选择而改变。

切换到代码窗口，向原来的代码中加入“setBackColor”部分的内容如下：

```
        private void button1_click(Object source,Event e)
 {
String name = edit1.getText();
 If (name.equals(″ ″ ))
{
    label2.setText(″ You Must Enter a Name″ );
}
else
{
    label2.setText(″ Hello ″  + name);
    setBackColor(Color.BLUE);
}
 }
```

提示：如果想要查看所有的颜色名称，只需在 Color 之后键入点（.），再等待片刻，屏幕上就会补全语句，出现关于颜色选择的列表。如果此时选择 BLUE，并按下 Tab 键，该颜色值就会自动加入到源代码中。

此时运行应用程序，一开始时，窗体呈现灰色，若在 edit1 编辑框中输入信

息，然后按下 OK 按钮，窗体的背景颜色就会变成蓝色。

显示图片

PictureBox 控件具有将图形放置在应用程序窗体中的功能。与前面讲述的其他控件的引入一样，只要把所要的控件拖动到窗体中适当的位置即可。若要改变图片的大小，只需拖动图片控件的控制柄便可以实现。

> **提示：** 若要删除不需要的控件，则应先选中该控件，然后按 Delete 键。

经过适当的调整以后，当所引入图片的位置和大小都符合要求以后，切换到 Properties 窗口，找到“pictureBox1”的图像属性（Image property）。此时，显示图像值的区域应该是空的。用鼠标单击该空白区域右侧的上有椭圆形点（…）的按钮。在打开的 Open 对话框中定位，或选择想要引入的图片文件。然后，回到 Properties 窗口的 PictureBox 属性中的 sizeMode 状态，用鼠标单击下拉箭头键，选择 StretchImage。

当完成了上述操作以后，屏幕上会显示出如图 2-8 所示的内容。

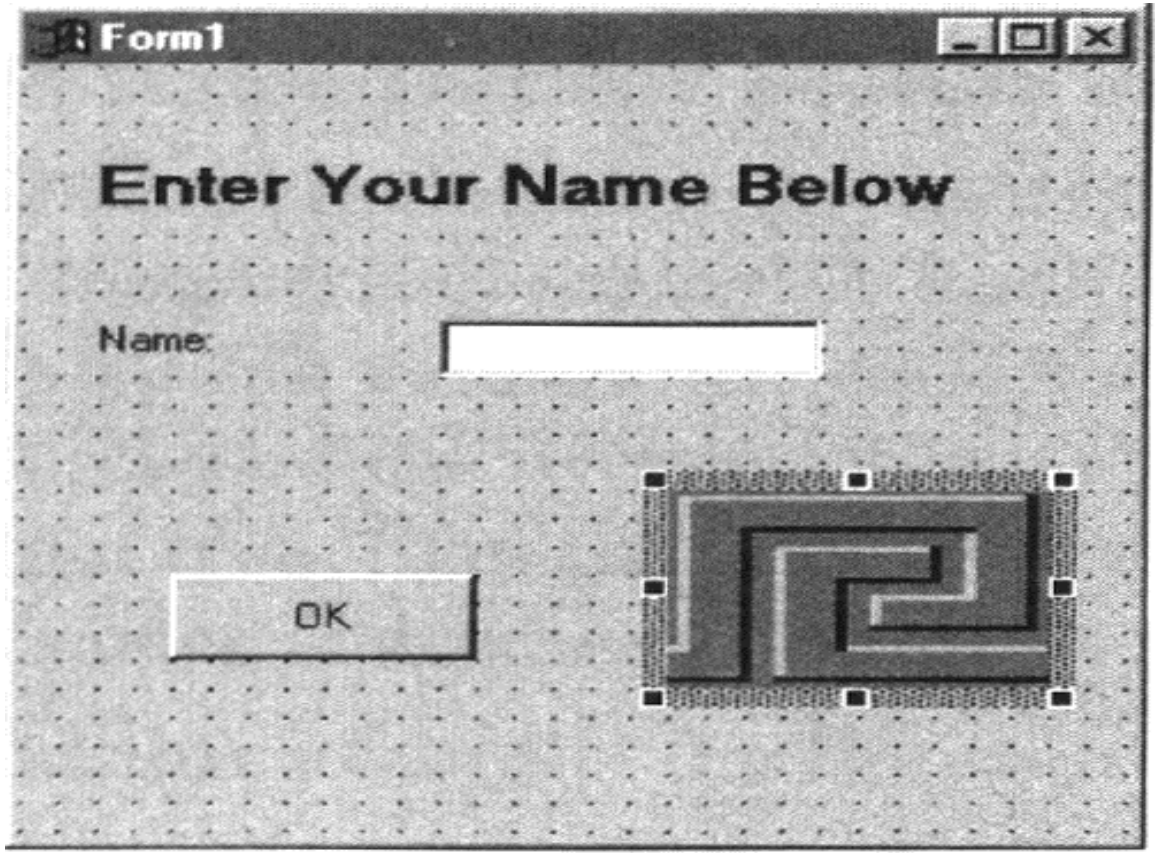


图 2-8　具有 PictureBox 控件的窗体

插入注释

注释是 Java 源代码文件（a .java 文件）中的说明部分，这部分内容在编译过程中被 Java 编译器所忽略。但是，它对于程序员以及应用程序的用户来说，

是十分重要的。注释可以使用户容易读懂源代码，从而更加容易地了解应用程序是如何由源代码实现的。到目前为止，任何编程语言自己都不能产生自己的自述文档代码。所以，良好的注释，可以帮助用户了解应用程序的编写工作进程。有过编程经验的用户往往会很惊讶地发现，当自己在 Java 程序中添加或删除几行代码以后，很快就会忘得一干二净。在以下这部分内容的学习当中，我们将继续研究 Java 语言，并详细地介绍 Visual J++的注释。

如何在 Java 当中创建注释内容

在 Visual J++ Code View 窗口中，所有的注释都会呈现为绿色的文本。

- 单行注释（Single-line comments）以两个正斜杠标记（//）起始，直至该行的代码结束为止。单行注释是最常用的一种注释方法，有时，这种注释方法又称为“C++”型注释。下面是关于这种注释方法的两个示例：

```
int count=0;          //represents the number of entries
    count =count+1;   //increment the entry count
```

- 多行注释以一个斜杠和一个星号（/*）为起始，以一个星号和一个正斜杠（*/）为结束.。由于多行注释的起始和结束与单行注释有着明显的不同（单行注释是在代码行终止处结束），多行注释往往可以很方便地用来解释和说明一整段的代码。然而，多行注释看上去与 JavaDoc 注释十分相似，容易使读者产生混淆。多行注释有时也称为“C 型”注释。下面是一个示例：

```
private void button1 click(Object source.Event e)
{
/* this event procedure will blah, blah, blah,
Notice that C-style comments can continue for
Any number of lines. */
}
```

- JavaDoc 注释与前面讲述的多行注释很相似，它在文档管理过程中是非常有用的。JavaDoc 注释以一个斜杠和两个星号（/**）为起始，以一个星号为和一个正斜杠（*/）为结束.。稍候，我们将详细地介绍 JavaDoc 注释。

TODO 注释

TODO 是 Microsoft Visual Studio 的一个特殊的标志（也就是说，它是一种指示器，由 Visual J++环境所识别，并对其作出反应)。该标志可以应用于上面讲述的三种类型的注释（单行注释、多行注释和 javaDoc 注释）当中。任何具有 TODO 标志的 Java 程序注释，都会自动显示在任务列表（Task List）窗口中，如果用户双击具有 TODO 标志的任务列表条目，便可以直接访问到那一行代码。利用 TODO 标志，用户可以了解到，一段代码是必须的，并且，应用程序最终所用的应该是完整的代码。Microsoft 是在代码模板（例如：在本章的一开始介绍的窗体模板）中应用 TODO 标志的。但是，应该注意的是，TODO 标志必须紧紧跟在注释标记之后，而且必须为大写字母，否则，它便不能显示在任务列表当中。

下面就是关于 TODO 注释的一段示例，这些 TODO 标志将会出现在任务列表窗口当中：

```
private void button2_click(Object source,Event e)
{
// TODO: button2's event procedure still not done
}
```

JavaDoc 注释

JavaDoc 注释是一种比较特别的注释设计，它是由源代码直接产生的说明文档类型。因为 JavaDoc 注释方法不能算是 Visual J++的一部分，所以在这里本文不详细介绍这部分内容。尽管如此，适当了解它也是十分必要的。用户在使用 Visual J++时，有时，产生的源代码就会使用这种注释，如果对 JavaDoc 注释没有深刻全面的了解，就不要随意删除这些注释文件。

当 Visual J++用户要开发和使用 Class Outline 特征时，就会用到 JavaDoc 注释的另一个特殊功能。若在某个类、方法或者数据项前添加了 JavaDoc 注释，当用户选择 Class Outline 中的类、方法或者数据项时，就会看到 JavaDoc 注释的第一行。下面是一个简单的示例：

```
/**
 * This class can take a variable number of parameters on the command
        * line. Program execution begins with the main() method. The class
        * constructor is not invoked unless an object of type 'Application1'
        * is created in the main() method.
```

```
     */
    public class Form1 extends Form
    {

    /** "OK"  button event procedure
   (user-added JavaDoc comment)
*/
    private void button1_click(Object source,Event e)
    {
        String name = edit1.getText();
        // the code continues ...
```

在上面的代码当中，出现了两个 JavaDoc 注释，第一个 JavaDoc 注释是窗体模板代码的一部分，第二个则表示了要添加的 JavaDoc 注释内容。让 Class Outline 窗口变成可视的，然后，注意查看代码段的第一行。如果此时不能在屏幕上看到，则应该在 View 菜单选择 Other Windows 选项，然后，再选择 Document Outline 选项。以上操作就会使 Class Outline 窗口在最前端。如果此时屏幕上出现诸如“所选择的文档没有要显示的项目”的消息，就要切换到 Project Explorer 窗口，并单击 View Code 按钮。在 Class Outline 窗口中，找到窗体 Form1，并将其选中。在此窗口的底部，显示了与窗体 Form1 相联系的 JavaDoc 注释的第一行。展开窗体 Form1 左侧的列表。找到事件 Button1_click 的条目，并将其选中。此时，在该窗口的底部，用户可以再一次看到 JavaDoc 注释的第一行。如图 2-9 所示：

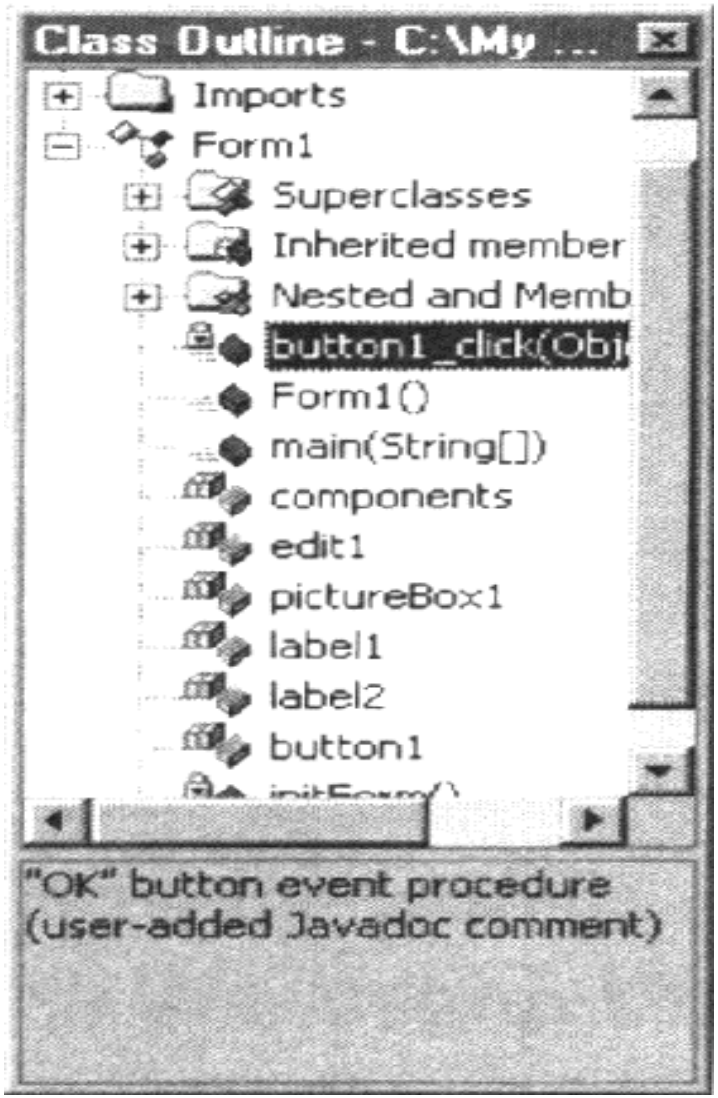


图 2-9　在 Class Outline 窗口中查看 JavaDoc 注释

Class Outline 是一个强有力的工具，我们将在第 5 章中详细介绍。

实验 2-2：输入保密口令

下面，让我们为自己的应用程序增加一些根据用户的输入而作出反应的行为。

实验概述

在本次实验练习当中，我们将学习使用：

- if 语句。
- 颜色（Colors）。
- PictureBox 的可见性。

在这部分的学习中，我们将使用过去讨论的一些工具，为应用程序提供一个保密口令。如果用户输入这个保密口令，应用程序就会被激活，从而使得图片箱可视，并更改标签的文本和标签文本的颜色。

实验准备

1. 如果已经按本章前面示例中的内容进行了有关操作，那么实际上已经完成了安装，可以跳过这部分内容向下进行，阅读实验指导。
2. 否则，首先启动 Visual J++。
3. 打开实验 2-2 的项目文件（Chapter02\Lab2-2\Hello.sln）。
4. 查看 Project Explorer 窗口，如果在项目文件图标（名为 Hello）下没有发现任何东西，则要单击 plus 标记，以显示与该项目文件相联系的文件。
5. 选择 Form1.java 项，并且在 Project Explorer 窗口的顶部单击 View

Code 按钮。

实验步骤

1. 首先找到事件 edit1_textChanged 的事件处理程序，然后在其中加入一个 if 语句，该语句可以用来检验 edit1 的文本属性的当前值。
2. 如果 edit1_textChannged 的文本属性值是保密的，加入其他代码使得应用程序能够：
   - 将 label2 控件的 foreColor 属性改为 Color.RED。
   - 将 label2 控件的文本属性改为“That's the Secret Word!”。
   - 将 PictureBox1 控件的可视属性改为假。
3. 添加布尔表达式，来检验 Edit 控件的文本属性：

```
edit1.getText（）.equals（“secret”）
```

不论何时，在想要确定一个控件的属性值时，都要同时使用属性名和“get”。想要改变一个控件的属性值时，也都要同时使用属性名和“get”。

4. 现在，可以使用 if 语句中的 else 部分，来完成在此改动前事件处理程序所要做的工作。

```
else
{
String name = edit1.getText();
Label2.setText(″ Hello ″  + name);
```

```
}
```

5. 运行应用程序，输入保护口令，看看会出现什么情况。
6. 将本次实验与第二章中 Chapter02\Sol2-2\Hello.sln 的解决方案相比较，看看有什么不同。

下一步

到目前为止，我们已经学习了如何使用窗体模板，了解了窗体模板可以实现用户与窗体交互的优点。我们所编写的应用程序，可以对用户的指令作出适当的反应，同时也可以使用户了解程序的执行情况。但是，在有些时候我们会发现，所编写的代码并不能够如期精确地执行。在这种情况下，调试器就成了我们最好的朋友。让我们进入下一阶段的学习。

调试代码

对于那些复杂庞大的代码，我们就应该拿出程序员的法宝：调试器。利用Visual J++调试器，用户可以在程序运行的过程当中观察代码。使用这一功能，就可以使程序在代码段当中逐行运行，或者，在代码段当中设置断点（breakpoints），观察变量。下面就介绍这些功能是如何实现的。

首先，我们要进行一些设定操作，以使 Visual J++环境准备好调试功能。为了达到这一目的，需要使用 Debug 工具栏。如果此时 Debug 工具栏处于不可视的状态，只需在任何可视的工具栏按钮区域按下鼠标右键，然后从弹出菜单当

中选择“Debug”即可（另外,在 View 菜单中使用 Toolbars 菜单项，也可以得到同样的工具栏列表)。与 Debug 工具栏按钮相联系的命令和 View 菜单中的命令是相对应的。图 2-10 显示了 Debug 工具栏。

图 2-10　Debug 工具栏

到目前为止，我们已经运行过了应用程序，但是，这种运行是单击了“Start Without Debugging”按钮而实现的。现在，我们使用 Debug 工具栏中的 Start 按钮，来实现应用程序的运行，这样，就可以使应用程序在调试模式下运行，也就是说，在这种运行模式之下，调试器的所有功能都可以使用了。

断点

为了在程序的运行过程当中检查代码，就要能够在中途停止一些代码。为了这个目的，就要设置断点。断点是程序调试人员在代码中某行所添加的标记，当程序运行到该处时，编译器会将程序暂停在此处。

设置断点的方法有以下四种：

- 选择要设置断点的代码行，按下 Insert Breakpoint（插入断点）按钮（该按钮上绘有一个小手标记）
- 选择要设置断点的代码行，然后在 Debug 菜单中选择 Insert Breakpoint

项。

- 右击要设定断点的行，在弹出菜单中选择 Insert Breakpoint 项。
- 单击代码窗口边缘的灰色区域。

设置了断点以后，当应用程序开始运行时，程序会一直运行到设置了断点的代码处为止。在代码窗口中，设置了断点的代码行在窗口左侧边缘会有一个停止的标记，用户可以很容易地判断断点即程序运行到的位置。如图 2-11 所示。

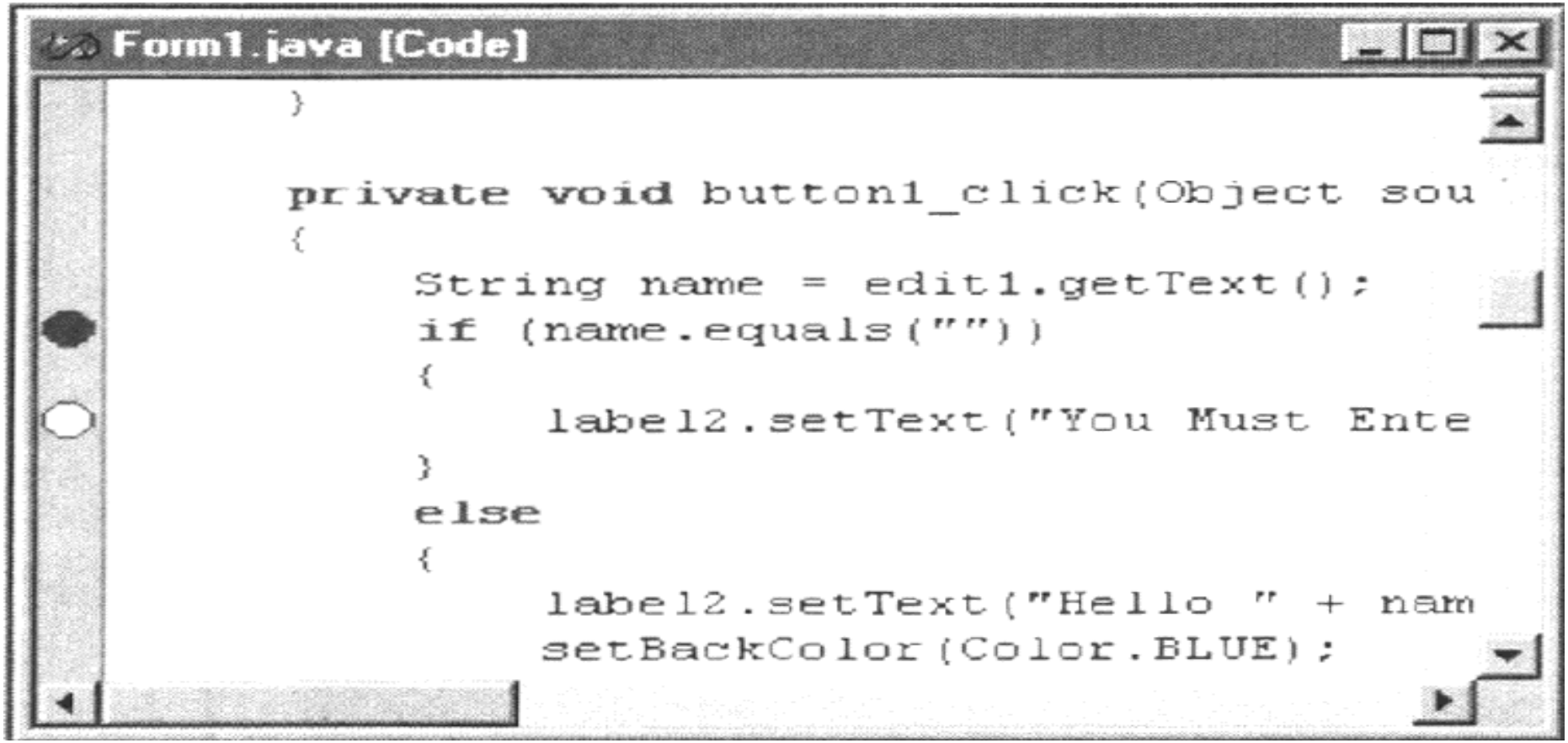


图 2-11　代码窗口左侧边缘的停止标记（图标）示意断点所在的位置

清除和移动断点与设置断点的过程相反。若要清除或移动断点，可以将鼠标指向设置了断点的行，单击鼠标右键，在弹出菜单中选择“Remove Breakpoint”，或者在 Debug 任务栏当中按下“Remove Breakpoint”按钮。另外，清除断点最简捷的方法应是单击代码窗口左侧边缘的停止标记。使用“Clear All Breakpoints”按钮，只要按一下键，便可以清除所有的断点。

除此之外，Visusl J++还提供了完整保留断点而只是在调试运行时忽略断点的功能。这也被称为禁用断点。在任何具有“Remove Breakpoint”选项的菜单或工具栏中，也都具有“Disabling Breakpoint”的选项。从图 2-11 中可以看到，一个禁用断点的标记在代码窗口左侧为空白的停止标记。

如果想要一览所有的断点，可以使用“Breakpoints”按钮。此时，屏幕上就会显示如图 2-12 所示的用户界面。

使用该窗口，可以看到项目文件中的所有断点。这个窗口中包括有断点所在的源代码文件名、所在行以及方法名称和方法所在的行。如果选中了某个断点的复选框，该断点就会被激活（也就是说，程序运行到该断点处，调试器会停止程序)。如果断点的复选框没有被选中，那么，该断点在调试过程当中将不起作用。另外，用户也可以在此窗口中选中要删除的断点，然后使用 Remove 按钮，将断点彻底清除掉。用户在该窗口还可以添加断点，但是，在源代码窗口中添加断点更为容易。

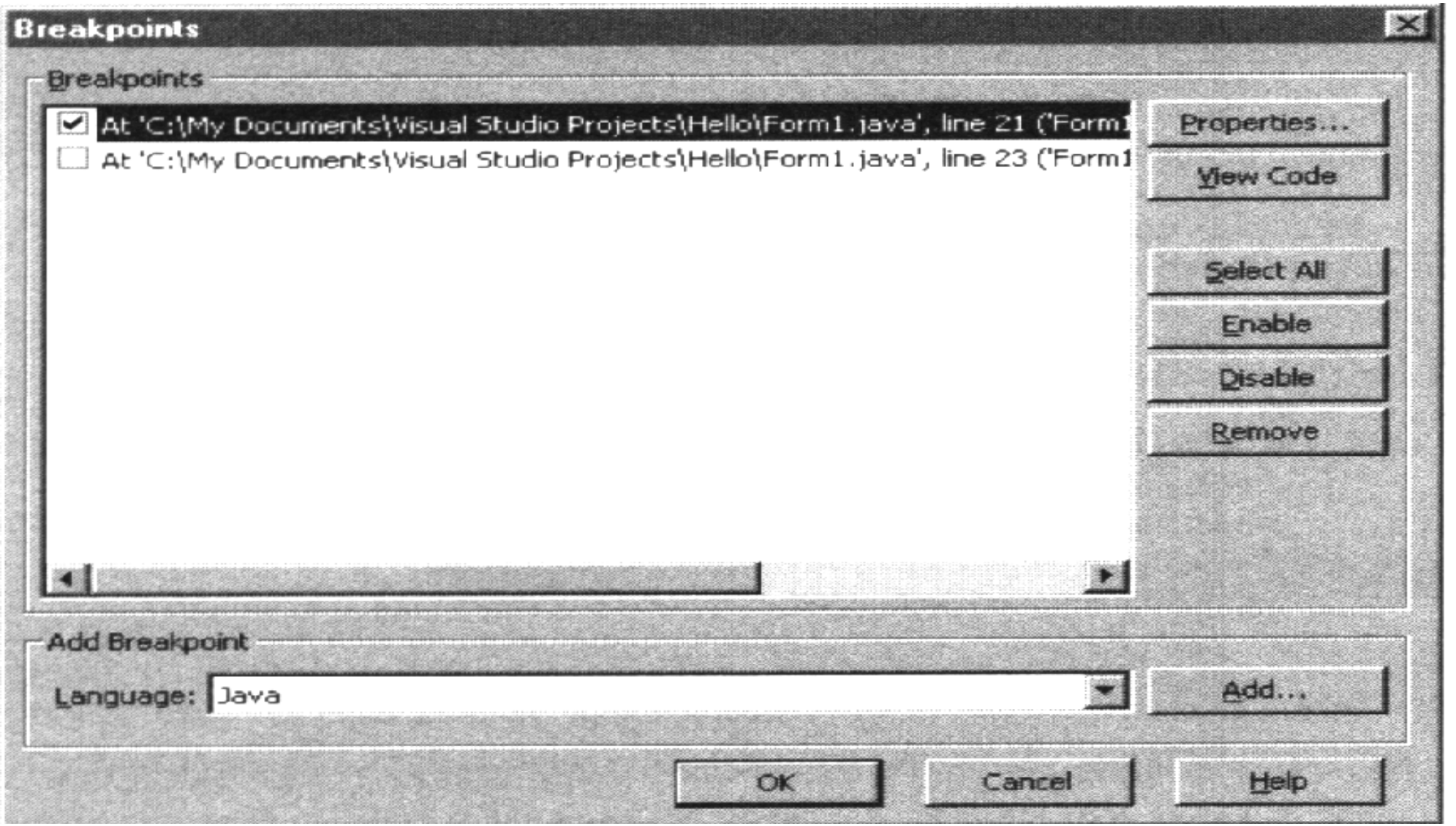

Breakpoints
Breakpoints
At 'C:\My Documents\Visual Studio Projects\Hello\Form1.java', line 21 ('Form1
At 'C:\My Documents\Visual Studio Projects\Hello\Form1.java', line 23 ('Form1
Properties...
View Code
Select All
Enable
Disable
Remove
Add Breakpoint
Language:
Java
Add...
OK
Cancel
Help

图 2-12　断点窗口

在源代码中单步运行

在源代码中添加了断点之后，就可以按下 Start 按钮，开始运行应用程序。在此之后，屏幕上就会出现如图 2-13 所示的“Watch”窗口，与此同时，调试器开始运行代码。用户可以及时查看“Watch”窗口。首先我们介绍如何实现程序调试的进程在代码当中进行。

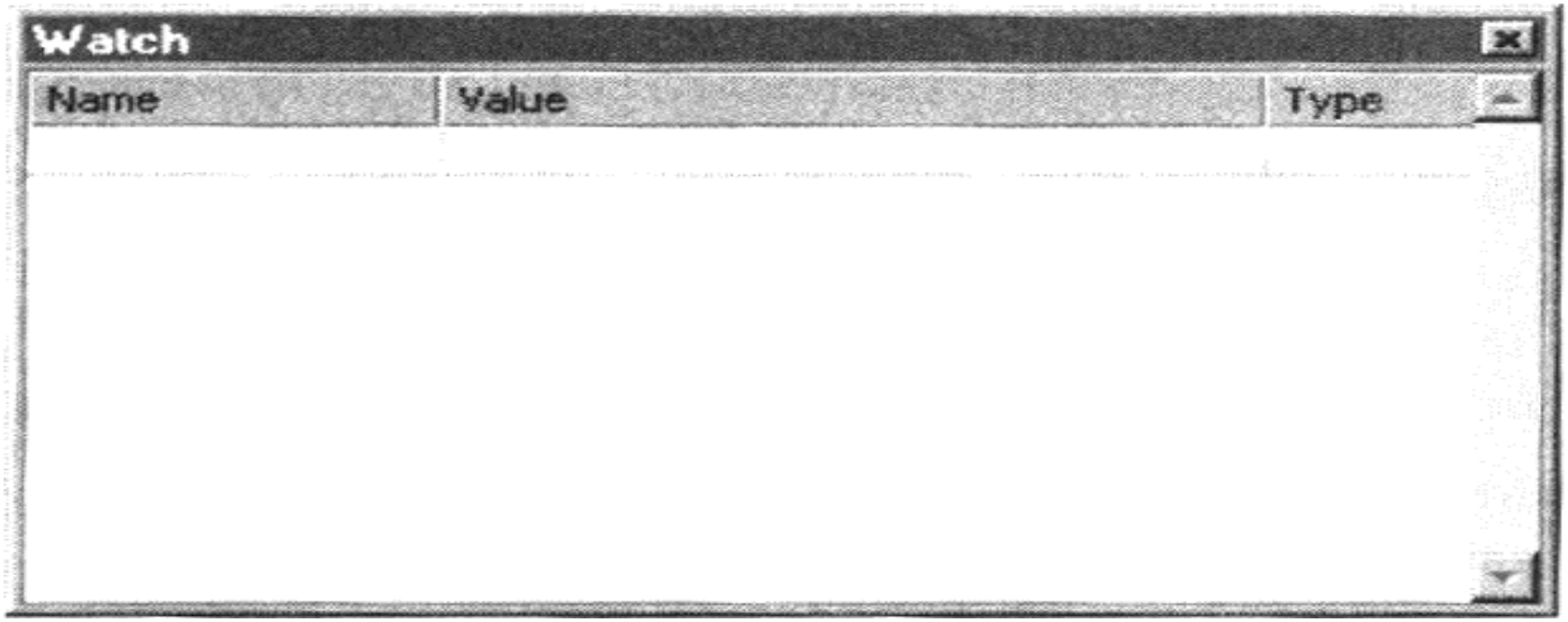


图 2-13　Watch 窗口

注意：对于 Visual J++环境，在调试模式和设计模式下，屏幕上所显示的窗口往往是不同的。比如说，当用户处于程序设计工作模式下，在屏幕上显示“Watch”窗口就没有必要了。所以，当屏幕上窗口的状态发生变化时，你所熟

悉的窗口没有出现，也不必大惊小怪的。完全可以在 View 菜单当中选择合适的选项，以显示所希望的窗口。但是，无论怎样都要确保所选择的窗口是与工作模式相对应的。

应用程序在停止之前究竟运行多少，这完全取决于第一个断点所设定的位置。例如：如果将断点设置在一个按钮控件的事件处理程序上，那么，就必须按下这个按钮，才得以使应用程序停止在该代码行。当程序运行设置了断点的代码行，代码窗口左侧边缘对应该行的位置上，就会出现一个箭头作为标记，表示程序运行到此处。如图 2-14 所示。

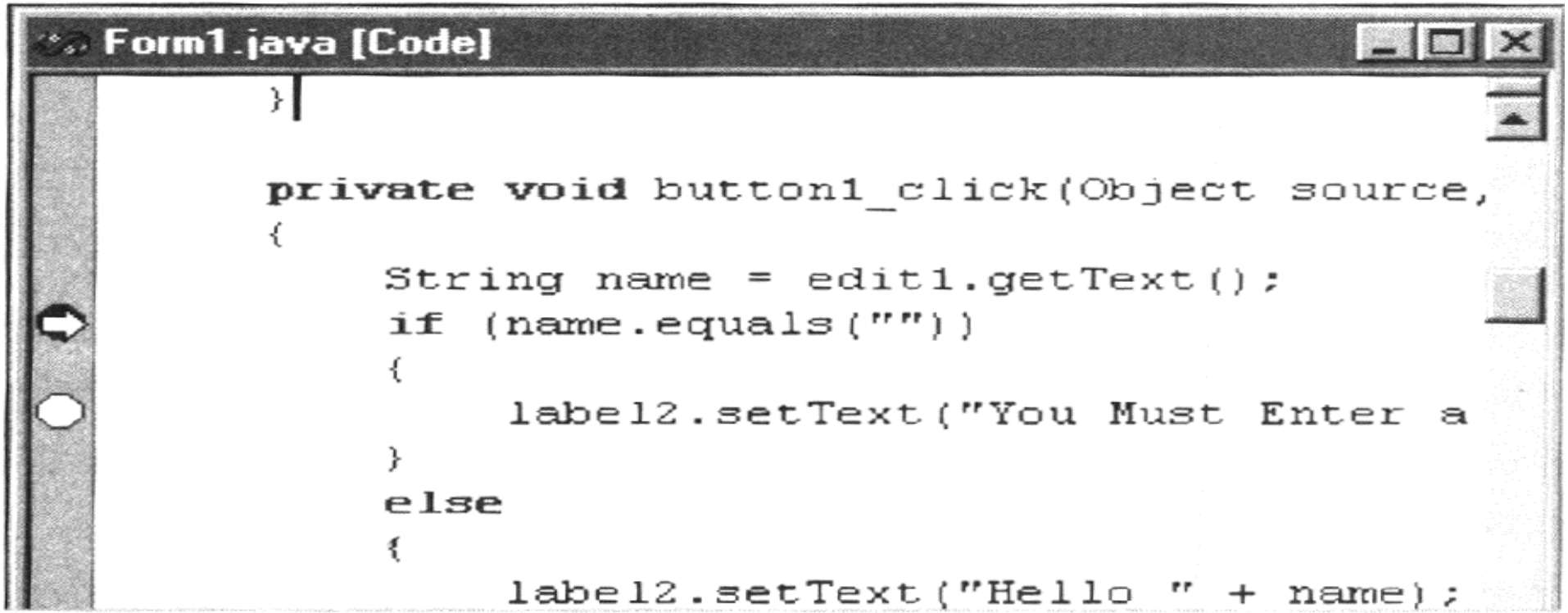


图 2-14　调试运行模式下的代码窗口，在某断点处停止

现在，用户可以查看应用程序的当前状态，或者通过该代码行向下运行。在 Debug 工具栏当中，所有按钮都具有专门的功能对应于不同的调试方案。

若要使调试运行工作往下进行，在 Debug 工具栏当中，可以有几种按钮选择，如果当前行所包含的方法中有需要访问的源代码，可以选择以下几项：

- Step Into：选择此按钮，调试器将直接运行到那个方法处，并且将当前行设定为该方法的第一行。
- Step Over：选择此按钮，调试器将不显示代码的任何信息，而直接运行该方法的代码。
- Step Out：选择该选项时，调试器将运行当前方法的其他有关代码，直至方法调用结束。

如果当前行不是一个方法，或者该行中没有所要访问的源代码，那么“Step Into”和“Step Over”按钮的作用就是相同的。另外，要注意的是，即便选择“Step Out”，断点仍然是有效的。如果选择跳过一个方法，断点将在该方法结束前生效，程序仍然要在断点处停止。

注意：当逐步调试运行程序代码时，总会有这样的时候，应用程序不显示任何东西。此时也不必担心，对于用户感兴趣的值，调试器会随着程序的执行而及时地显示出来。用户可以使用应用程序和 Visusl J++项目文件环境之间的任务栏上的前进和后退按钮反复查看，直到对调试过程有了全面的了解为止。

查看变量

当使用调试器开始运行一个应用程序以后，屏幕上就会出现一个 Watch 窗口，其中有许多空白区域。这个窗口中的区域正是为用户所感兴趣的变量而准备的。如果想将某一变量移动到查看窗口当中去，首先右击该变量，在弹出菜单中选择 Add Watch 项，这样，系统就会将所选择的变量放到 Watch 窗口中标有 Name 的栏目当中，与此同时，窗口中还会显示出，变量的值以及变量类型的简要说明。如果程序继续运行到某行代码，使得变量的值发生变化时，变量值就会变成红色。当程序运行到下一行代码时，变量值又会变为原来的黑色。

例如，下面的三个变量：

```
int count=5;
int maximum=10;
boolean maximumReached=false;
```

图 2-15 显示了这些变量在 Watch 窗口中的情形。

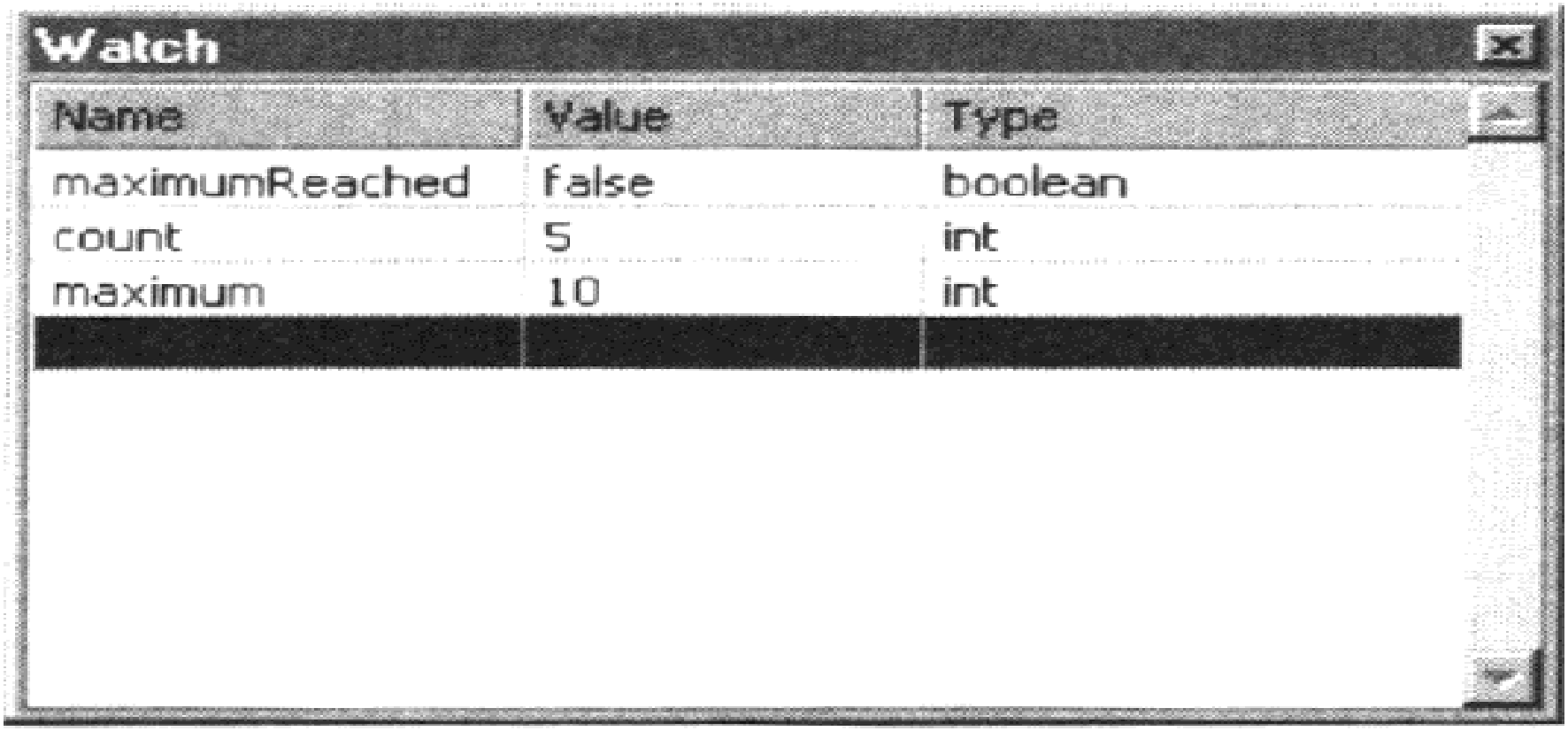


图 2-15　Watch 窗口及窗口中的几个变量

如果此时右击 Watch 窗口中的变量，就会在弹出菜单中得到几个选项，你可以移动该变量在 Watch 窗口中的位置，以及继续往下进行。下面将进一步讨论这个问题。

Immediate 窗口

在程序调试的过程中，往往会碰到这样的情况，就是在对程序进行单步调

试时，需要及时知道某个变量的值。下面，将介绍 Watch 窗口是如何实现这一功能的。当然，其他窗口几经周折，也可以实现这种功能。另外,这里先交代一下，所有的调试窗口都可以通过“Debug”工具栏或者“View”菜单获得。在所有调试窗口中，只有 Immediate 窗口可以通过向变量中存储数值的办法，来实现在运行的过程中改变变量的值，同时，也只有 Immediate 窗口，才允许用户在程序运行时改变变量的值。如果在 Immediate 窗口中输入一个变量名，并按下回车键，窗口中就会显示出变量的值。此时若要改变变量的值，在 Immediate 窗口中输入赋值表达式即可。

就上文讲到的三个变量“count”，“maximum”和“maximumReached”为例，图 2-16 显示了改变这三个变量的过程。用户在窗口中输入变量“count”，并按下回车键，在此之后，屏幕上显示出该变量的值“5”。如此对另外两个变量重复操作。下一行表达式（count<maximum）的值为真。最后，用户将值“true”存储到变量“maximumReached”，同时，所赋的值返回，并显示在窗口中。

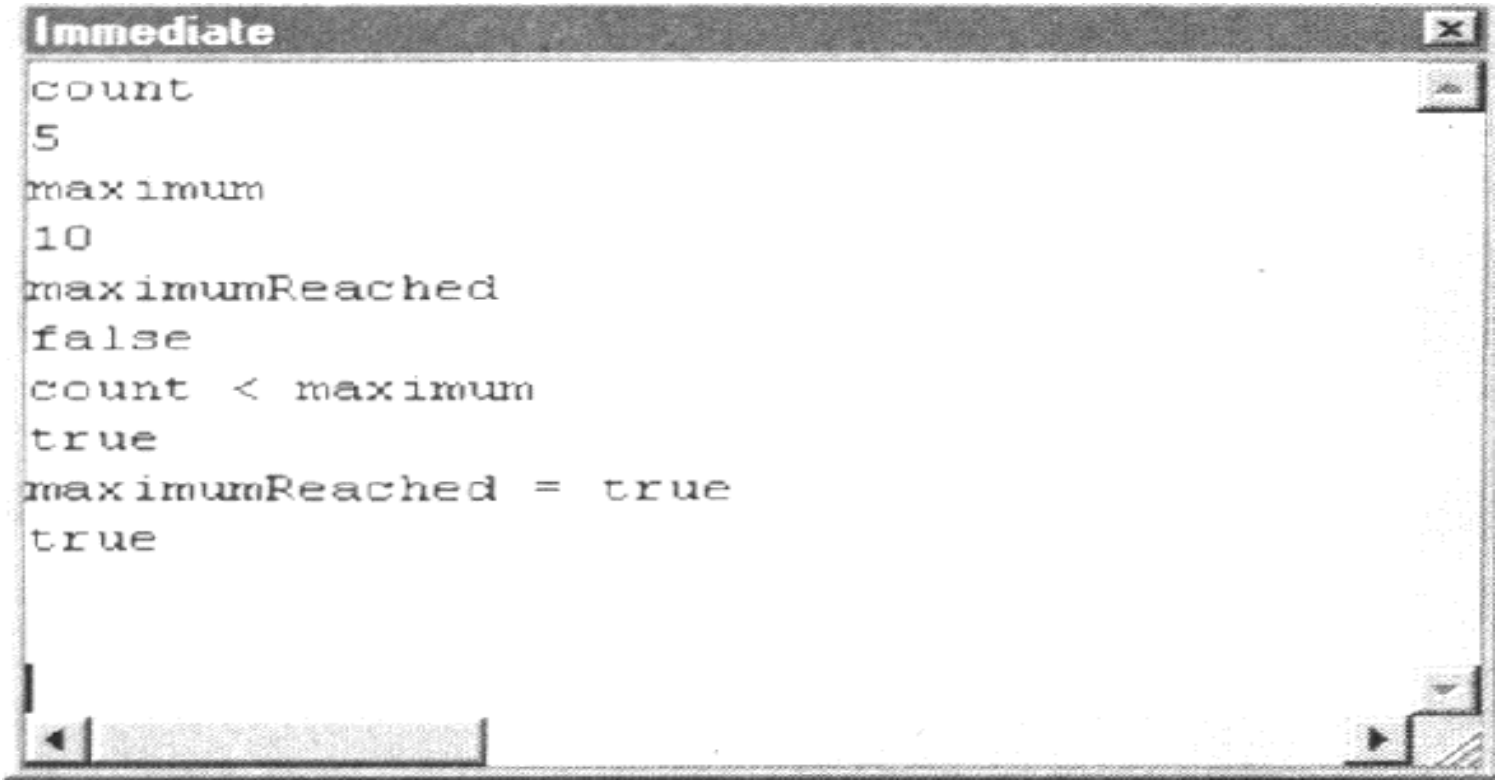


图 2-16　使用 Immediate 窗口改变变量的值

Output 窗口

当 Java 在命令行界面模式的操作系统（例如，MS-DOS 或 UNIX 操作系统）下运行时，程序的输出将写到控制台。另外，写入控制台的数据是以字符为基础的（character-based，也就是说，没有窗口支持）。即使没有命令行界面，Visual J++也可以将程序输出写到控制台。Visual J++是在一个 Output 窗口中显示输出控制台的。为了说明这一点，我们以“Hello”应用程序的另一个形式为例：

```
public class Hello
{
public static void main (String[ ] args)
{
        System.out.println(″ Hello World″ );
        // writes to the Output window
}
}
```

该程序在命令行操作系统中（command-line operating systems）实现在屏幕上输出“Hello World”。在 Visual J++中，“Hello World”将在 Output 窗口中显示。在调试的过程中，用户就可以方便地使用 Visual J++的这一功能，在跟踪程序的运行进程时，可以通过在程序中任何位置放置如下代码行：

System.out.println（”Now executing the XYZ method”）。

从而将所需要的信息写到 Output 窗口。在程序中添加了这样的代码行以后，当程序运行到此处时，系统就会将括号中的字符串写入 Output 窗口，同时又不影响应用程序的其他部分。打开 Output 窗口的方法为，先打开“View”菜单，在 Other Windows（其他窗口）子菜单中选择“Output”项。如果在实际操作时，Output 窗口没有显示出上面所说的内容，此时，应该检查 Output 窗口的顶部下拉框中是否有“Debug”（而不是“Solution Builder”）。如果此处选择的是“Solution Builder”项，那么，在 Output 窗口中就只会显示在组建项目文件时的一些信息（例如，编译错误等等）。然而，我们现在不是在讨论组建项目文件，而是在调试运行应用程序，所以应该确认“Debug”项是被选中的。

注意：在应用这个“Hello”程序时，用户往往会希望使用一个不包含窗体的项目文件。创建不包含窗体的项目文件的方法前面并未介绍，在第 11 章中，将详细讨论这一问题。

自然，当用户经过调试运行，确保自己的程序满足要求时，就一定希望，在运行时不要对系统的 System.out.println 库文件进行任何调用。

再次运行

在调试运行的过程中，一旦检查了自己所感兴趣的代码，就可以准备继续运行程序，直到下一个断点，此时应该再一次单击 Start 按钮（Start 按钮的提示为 Continue）。如果在调试时想要程序运行到某个位置停止，但是又不想在那里设置断点的话，可以使用运行至光标处（Run To Cursor）按钮。单击该按钮，程序就会由当前位置一直运行到光标所指定的代码行（如果此时尚没有开始调试，那系统就自然从第一行开始运行到所选定的行）。但是，要注意的是，如果在程序运行的当前行与光标所选定的代码行之间存在断点，并且这些断点处在激活状态时，调试器会使程序首先在这些断点处停止。如果此时想忽略某个断点，要么将其删除，要么就将其禁用。

重新开始调试运行的另一个方法是，重心运行应用程序。单击 Restart 按钮，就可以重新运行应用程序。

如果想停止应用程序，可以单击 Close 按钮或者选择 Debug 菜单中的 Stop 项。

第三章　类与窗体

到目前为止，我们已经简单介绍了 Visual J++开发环境，以及如何应用该环境创建一些简单的应用程序，及如何实现系统与用户之间的交互。掌握了这些知识以后，就应该进一步研究 Java 语言本身了。

对一个 Java 程序员来说，Java 类的创建和操作是一项主要任务。如果一个程序员未曾深入地了解 Java 类，他就不可能使用 Java 语言来进行编程。Java 语言提供了许多功能强大而且适应性很强的类，供程序员在组建自己的程序时使用，当然，用户也可以针对各种具体问题采用自己的解决方案，以创建自己的类。

本章不仅阐述了一般的 Java 类，同时，也详细地讨论了 Visual J++ Form 类。本书在第二章当中介绍的窗体就是 Form 类的成员。

在本章中，将介绍以下几方面的内容：

- 类与对象
- 引用
- new 关键字
- 构造器
- 特殊的 Visual J++对话类

另外，在本章中，还将实践以下各个 Java 类的使用：

- MessageBox 类
- ColorDialog 类
- FontDialog 类

类与对象

类与对象是使用 Java 语言编程时的两个最重要的组织工具。在组建一个 Java 程序时，对于以上两个概念的应用是必不可少的。一旦用户开始使用 Visual J++ 开发系统就会发现，正是由于类与对象在系统中的应用，才使得 Java 程序的创建变得方便快捷。

Java 类

所有编程语言都有自己的内置数据类型应用在创建变量的过程当中。内置数据类型为系统提供了在程序员创建变量时自动定义变量类型的功能。比如说，Fortran 和 C 语言，程序员在使用这两种语言编程时，就会使用内置数据类型，如 int, float, REAL 和 INTEGER 来创建几乎所有的变量。Java 语言也有自己的一套内置数据类型（或者称为原始数据类型），它以 C++的内置数据类型为基础（关于 Java 语言原始数据类型更详细的内容，将在第四章中介绍)。与 C++语言相类似，Java 语言同样也有自己的类。但是，Java 语言在这方面比 C++更进了一步，就是对于 Java 语言，要求所有的代码和数据都要归属于某个类。

与系统的基本类型不同，类属于用户自定义的类型。基本类型对于许多应

用程序来说都是非常合适的，但是，随着软件发展，复杂程度的不断增加，要想完全使用内置数据类型就变得越来越困难。基本类型所带来的麻烦是，它或多或少地反映了计算机内部的硬件设备：整型数、位和浮点数等。而用户自定义类型（也就是前面所说的类）允许程序员定义自己的变量类型，使其更能反映所要处理的问题。事实上，任何一个程序都是要解决一些实际的问题。对一个程序员来说，能够和自己所要处理的问题的解决方案联系得更为紧密是十分重要的。除此之外，用户自定义类型的引入，大大增加了程序的可读性，同时也使得程序修改起来更为方便。另外，在用户所设计的程序内部环境中，程序员在创建自己的模型时，类的引入也为其提供了表现复杂结构的有力方法。在一个类当中，用户可以放置任意多的数据，同时，还可以设计专门的方法实现对那些数据的处理。一个 Java 类中包含的操作在 Java 语言中是以方法（methods）的形式出现的。在本章当中，将简单地介绍以下有关 Java 方法的基本内容，关于 Java 方法的详细内容，将在本书的第四章当中介绍。

类同时也包含了便于软件扩充的机制，也就是软件的继承性。继承性对于软件开发具有及其重要的意义，它可以使程序员直接应用以前的简单程序组合，来解决更为复杂的问题，避免了重复编写代码，减少了软件开发的复杂程度。在本书的第五章中，将详细介绍有关 Java 语言的继承性的内容。

使用 Java 类的核心是，用它来表现现实世界当中的概念。下面，我们将讨论用 Java 语言编写一个基于 Windows 的应用程序，该程序要求用户递交一个房屋的出价单（标书）。一旦标书递交之后，程序就会计算索要价与标价之间的差别。也就是说，假设我们是一个房地产开发的业主：城市建筑规划公司，我们要负责城市房屋建筑的规划，使得建筑能改善整个城市的面貌，同时，要将

这些房屋建筑出售给合适的买主。为了编写比较索要价与标价的程序，首先，就要求能够用代码来代表房屋。为了做到这一点，我们使用一个多用途 Java 类，其名称为：CityViewHouse。

```
Public class CityViewHouse
{
// methods and member variables for the class CityViewHouse
}
```

可以在代码示例中看出，除了注释内容（代码段中以//开头的部分）以外，这个 Java 类是空的。类的一个重要特点是，它包含方法和变量，这一点很快就会在本书中介绍。

对 象

在 Java 类中，一个对象就是一个简单的变量。通常，程序员在使用“变量”一词时，他们往往指的是基本变量。但是，就像我们前面所讨论的，Java 类是一个特别的类型，也就是说，Java 类的变量与内置类型变量也存在着不同之处，名词 Objects 也就有着不同的概念。尽管基本变量通常是极为简单的，但是，作为类的产物，对象在带来了复杂性的同时，也带来了更为丰富的功能。从本质上讲，示范一个类，就是举出一个类的示例，而一个类的示例也就是对象。

我们可以使用以下代码创建一个类的对象 CityViewHouse:

```
public class BidMaker extends Form
{
private CityViewHouse myHouse = new CityViewHouse();
```

```
// other declarations and code
}
```

要想读懂这些代码，首先必须弄清楚以下几个概念：引用、成员变量、new关键字、构造器等，这些概念将在以后的章节中介绍。

创建新的 Java 类

在本节内容里，将介绍如何在 Visual J++环境中创建一个新的项目文件，同时，再向其中加入一个多功能 Java 类。到目前为止，我们所讨论的项目文件还只是含有窗体。窗体是可以包含 WFC 控件的一种特殊的 Java 类。与第二章中所讲过的窗体模板相类似，Visual J++的类模板也会帮助用户，将自己的类添加到项目文件中去。从现在开始，我们不再使用项目文件向导来组建我们的项目文件，而是从创建一个空的项目文件开始，然后在此基础上，组建我们的应用程序。这样，系统就允许在创建的项目文件上有更多的控件，例如，用户可以自己命名所创建的窗体，而不是使用 Visual J++的缺省名称。

下面，我们开始实践创建一个 Java 类。首先启动 Visual J++。如果此时正处在 Visual J++环境中，则应该首先保存有用信息，然后在 File 菜单中选择 Close All 项，以关闭所有文件。此后，可以在 File 菜单中选择 New Projects 项，在屏幕上出现的 New Projects 对话框中，选中 New 选项卡，在 Visual J++项目文件列表中选择 Empty Project（空项目文件）。然后，在对话框底部的文本框当中，输入 BidMaker，按下 Open 按钮。如图 3-1 所示，New Project 对话框其中有被

选中的 Empty Project 文件夹。

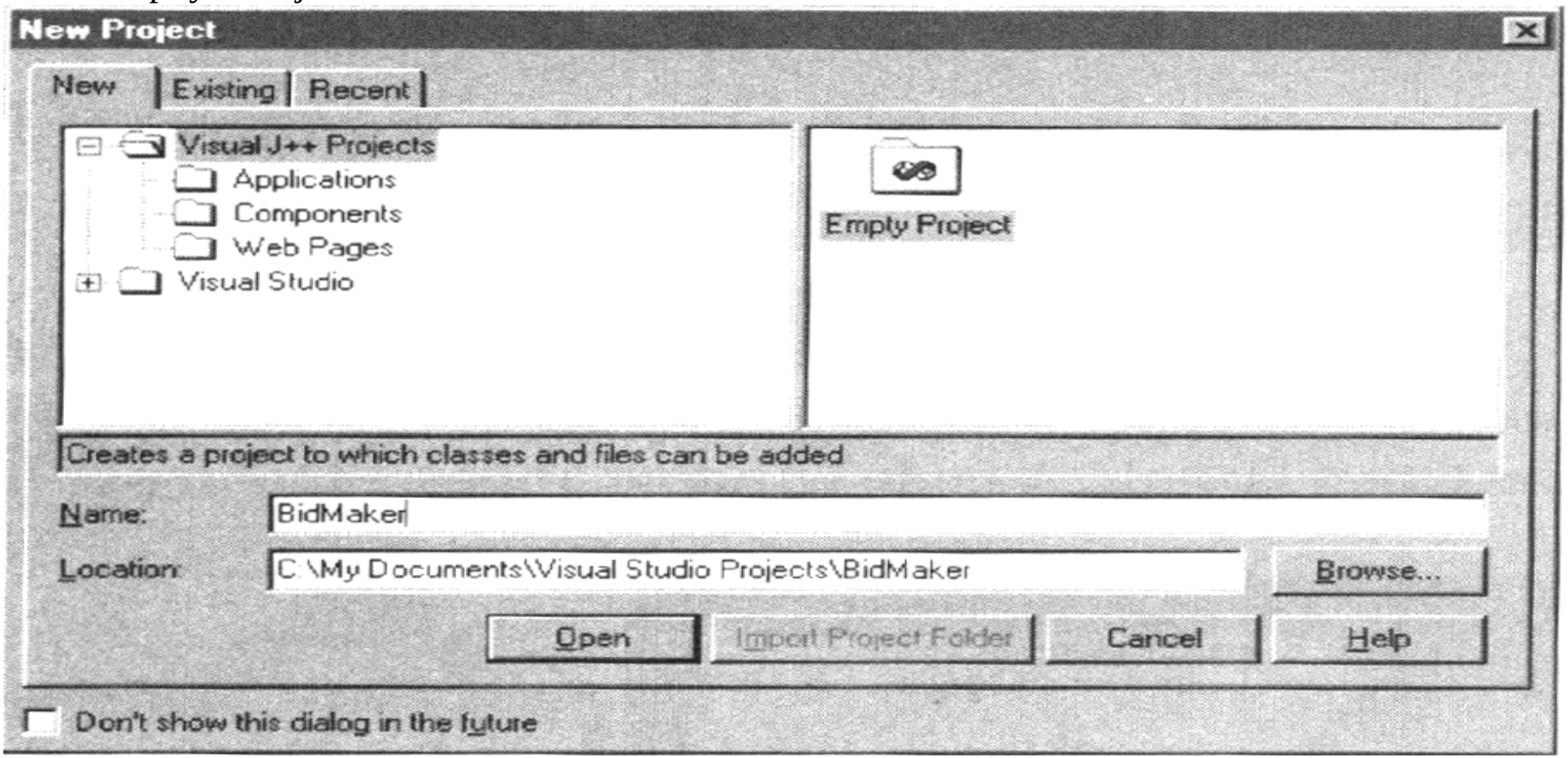


图 3-1　创建一个空的项目文件

当屏幕上出现窗口之后，将鼠标指向 Project Explorer 项目文件图标，右击在弹出菜单中选择 Add Form 项。在添加项目 Add Item 对话框的左侧窗格中，有预先选择的窗体。选择右侧窗格中的窗体，并将其命名为 BidMaker.java。按下 Open 按钮，该窗体就被添加到我们的项目文件中。就像我们前面所讨论的一样，在这个窗体中，可以添加一系列的 WFC 控件，同时，该窗体可以作为

新的应用程序的界面。现在，我们已经在项目文件中创建了自己的窗体，可以向其中添加新的多功能 Java 类了。再一次右击 BidMaker 图标，然后在弹出菜单当中选择 Add Class 项，这时，屏幕上出现 Add Item 对话框，该框体左侧窗格中，有预先选中的类。图 3-2 示意了此对话框的图形。

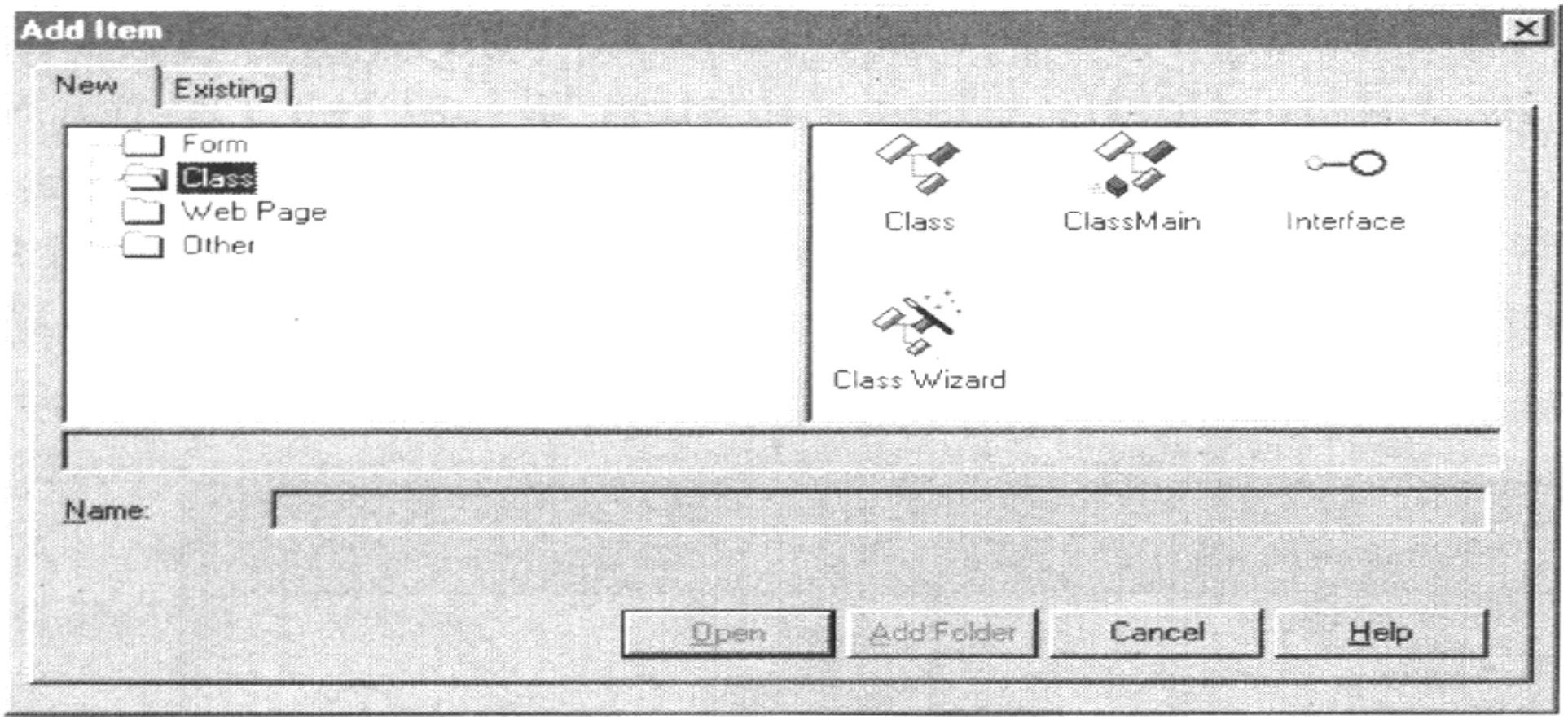


图 3-2　具有事先选中的类的 Add Item 对话框

单击右侧窗格中的类，并将其类文件命名为 CityViewHouse.java，然后按下 Open 按钮。屏幕上此时就会出现如图 3-3 所示的代码窗口。

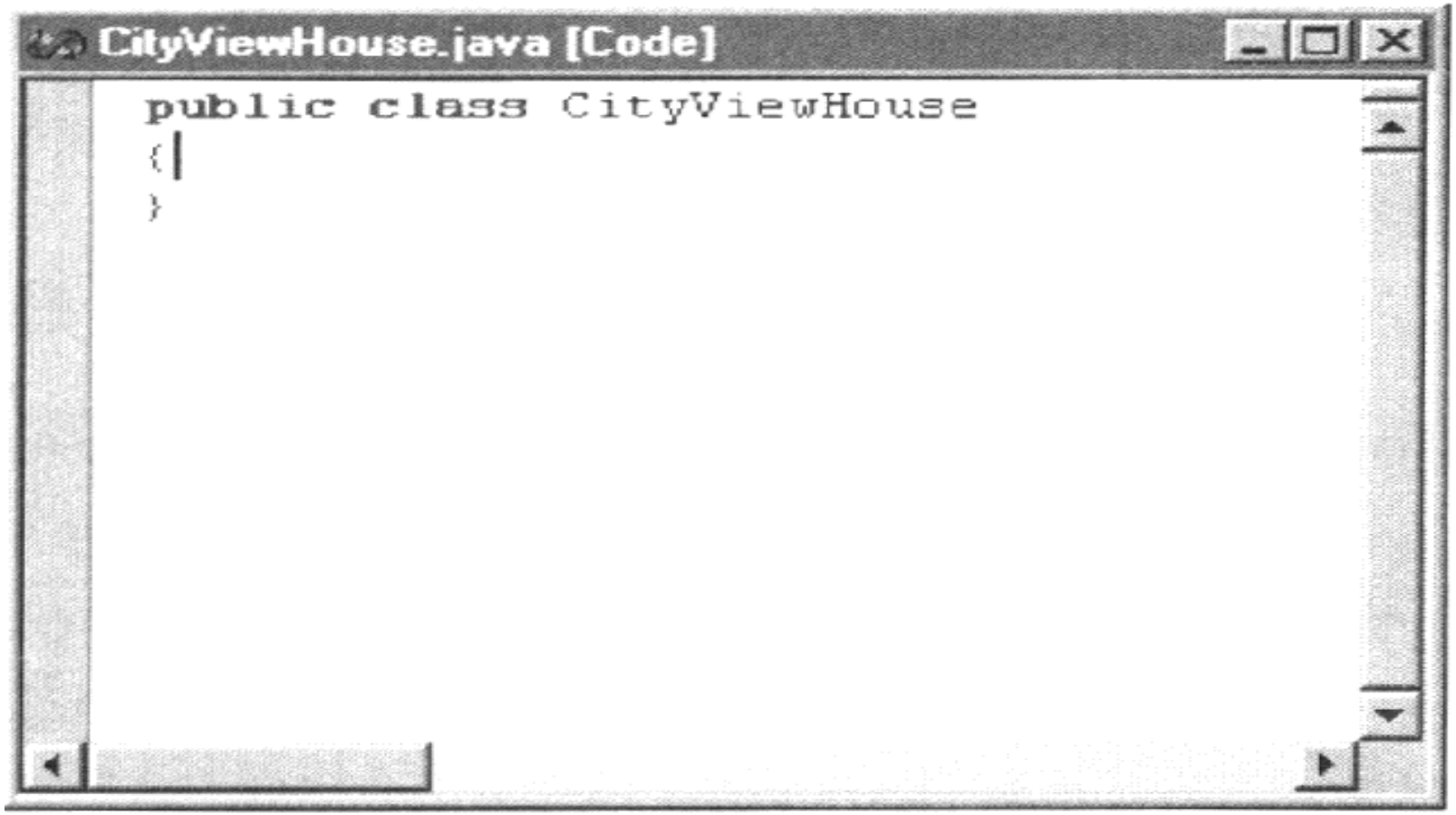


图 3-3　使用类模板新创建的类

要点： Java 类的源代码必须保存在与类同名的文件当中。在我们所讨论的示例中，这个文件名为 CityViewHouse.java，而类的名称为 CityViewHouse。

注意 Project Explorer 窗口，新创建的类是作为项目文件的一个成员出现的（如图 3-4 所示）。

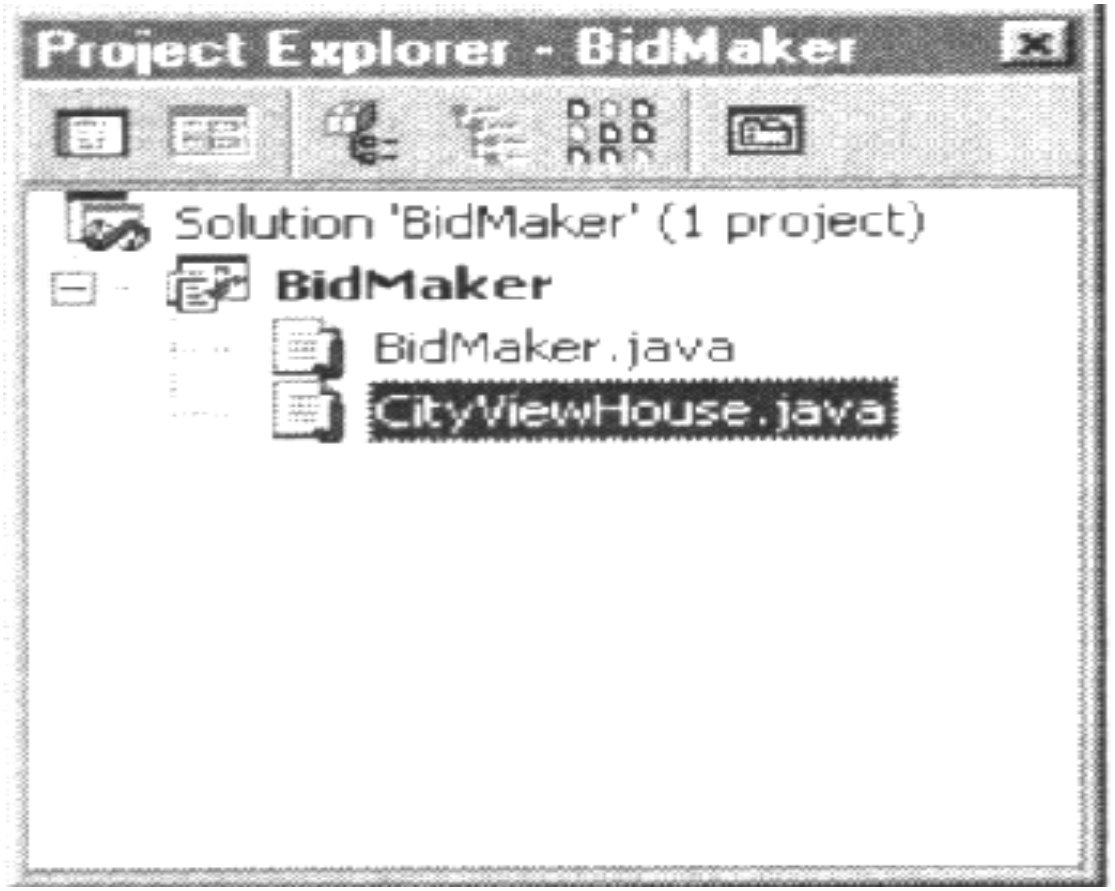


图 3-4　在 Project Explorer 窗口中的新创建的 Java 类 CityViewHouse

创建对象

到目前为止，我们对类与对象的概念以及它们之间的相互关系已经有了基本的了解。图 3-5 示意了两者之间的相互关系。现在，我们进一步讨论 Java 语

言是如何实现 Java 类的实例化的，也就是说，我们将介绍在 Java 语言中是如何创建对象的。

图 3-5　实例化是在一个 Java 类的设计中组建对象的操作

前面反复提到的“创建对象”、“为 Java 类创建实例”，这些都说明了对象是 Java 类设计的代表。要想了解在 Java 语言中是如何创建对象的，还应理解引用和关键字的概念。

对象的引用

由于 Java 对象没有自己的名字，因此，只有通过引用，才使得在 Java 代码中访问我们的对象成为可能。也就是说，Java 对象只能是通过使用标识对象的引用而间接地被访问。这种对对象的访问方式，看起来似乎有些古怪，但是，

当随着进一步的学习，使用这种方式的次数更多时，用户就会发现这种方法的好处。图 3-6 示意了引用的概念。注意名为 CityViewHouse 类上方的箭头。图中表示了引用本身并不是类的对象，而只是指出了访问那一对象的路径而已。

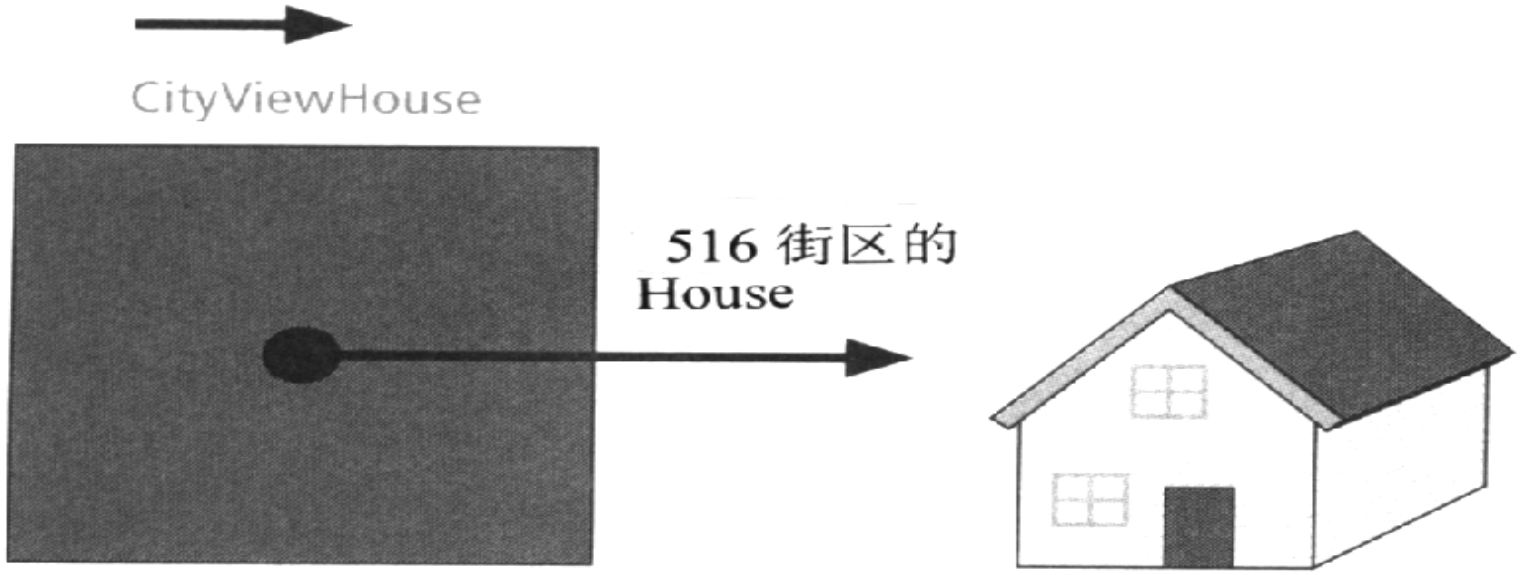


图 3-6　引用及引用与对象的关系图

要想操作某个对象，首先需要对象的引用，要想得到引用，首先就要对其进行说明。这就是我们将在下面的章节中要讨论的内容。

向类中添加成员变量

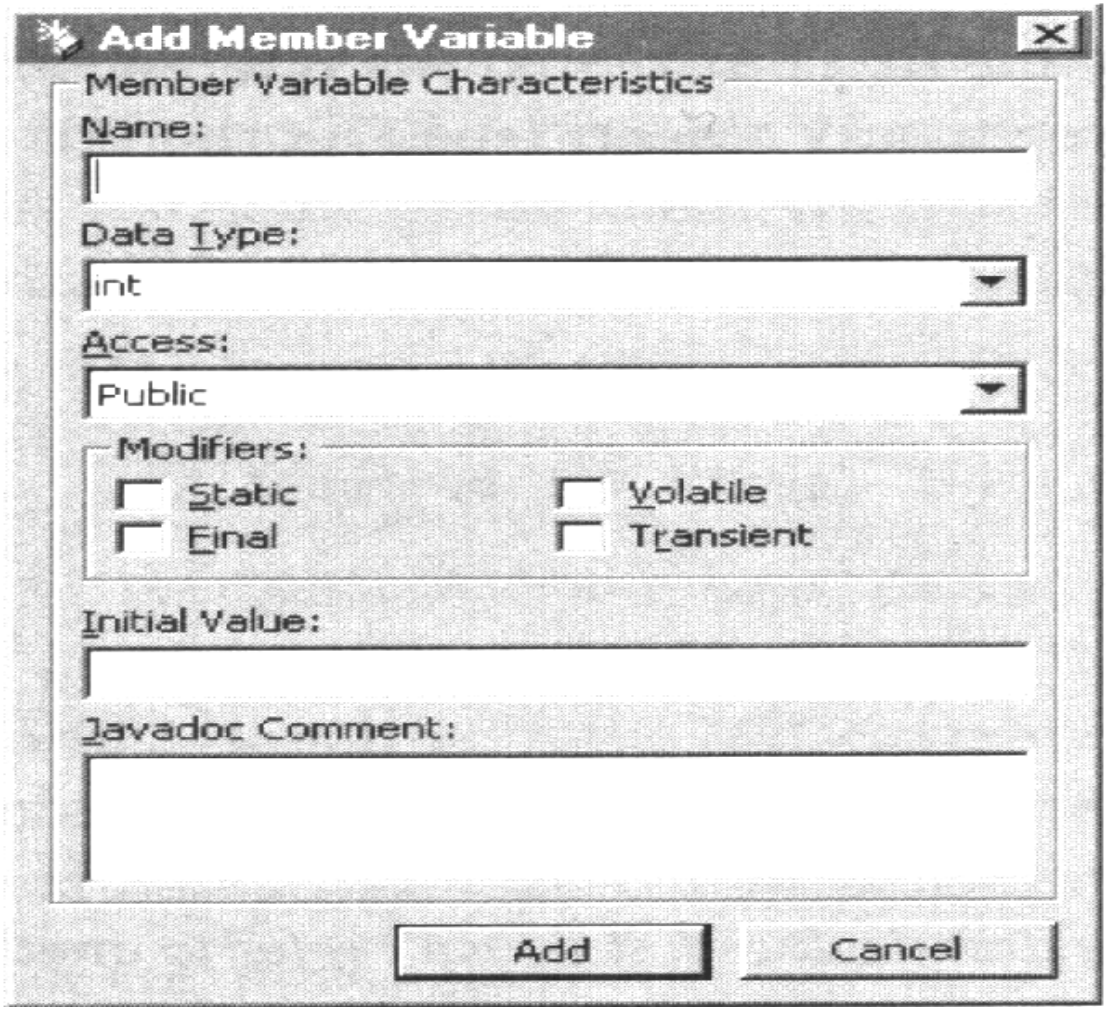


图 3-7　向类当中添加成员变量的 Add Member Variable 对话框

成员变量是在 Java 类中而不是在方法中说明的变量（在后面介绍了 new 关键字以后，我们将介绍方法的概念)。当对象的类创建之后，我们就可以访问其中的变量了，在使用这些对象期间，其变量均处于可访问状态。Visual J++提供了向类当中添加变量的工具。在 Class Outline 窗口中单击 CityHouseView 选项，

然后选择 Add Member Variable 选项，此时屏幕上就会出现一个 Add Member Variable 对话框，如图 3-7 所示。

在 Add Member Variable 对话框中输入如下信息：

Name（名称）	myHouse
Data Type（数据类型）	CityViewHouse
Access（访问方式）	Private

此时，可以在文本框中为该变量预留出空间。然后按下 Add 按钮，这样 myHouse 即由 CityViewHouse 所引用的成员变量就会加入到 BidMaker 的代码中。

后面，我们将进一步讨论有关成员变量的另一些选项。此时，就可以以私有方式访问各个成员变量，也可以不变动修改符中的复选框。与 JavaDoc 注释项类似，变量的初始值是任意的。另外应该注意，当添加 JavaDoc 注释时，实际上并不包括（/**和*/）。因为 Visual J++会自动加入这些符号，如果用户添加了这些符号，编译时反而会出错。

在 Add Member Variable 对话框的右上角有一个 Close 按钮，单击此按钮便可以关闭该对话框。打开代码窗口，查看 BidMaker 类，会看到如下的代码：

```
Private CityViewHouse myHouse;
```

因为一开始时我们在 Initial Value 文本框中没有输入任何信息，所以，引用 myHouse 开始也并没有指定任何对象（如图 3-8 所示）：

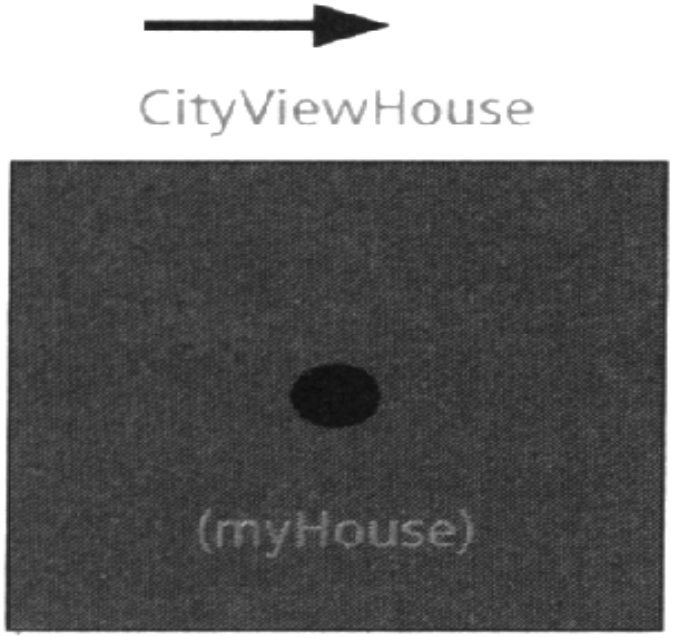


图 3-8　当引用变量第一次说明时，并未指定任何对象

myHouse 框体正中间的黑点代表空值。与其他编程语言一样，空值的意思就是指引用 myHouse 没有指定任何对象。注意，此时我们还没有对象 CityViewHouse，而只拥有对象引用的标签而已。如果想要真正创建一个 CityViewHouse 对象，则需要 new 关键字。

new 关键字

使用 new 关键字，我们可以利用类定义所提供的蓝图真正组建一个对象。如果使用 new 关键字，就会实例化或实现该类的一个对象。用计算机专业的术语来说，该操作将分配计算机的内存，以供所要创建的对象使用和存储。在对

象创建工作完成之后，系统还会为 Java 程序提供一个对象引用的返回值。这个返回的引用值可以存储到前面已经说明了的引用变量中去。

```
CityViewHouse myHouse;
MyHouse = new CityViewHouse;
```

综上所述，创建一个对象分为以下两个步骤：

1. 声明对象的引用变量。
2. 调用 new 关键字功能，将返回值存储到对象变量当中。

前面，我们已经声明了变量 myHouse。上述的第二步是在调用 BidMaker 的方法中完成的。注意方法的名称与类的名称是相同的。在以后对构造器的讨论中，我们将体会到为什么要特别注意这一点。方法 BidMaker 的启动如下列代码所示：

```
public BidMaker()
{
// Required for Visual J++ Forms Designer support initForm();

// TODO: Add any constructor code after initForm call
}
```

下面，就可以在方法 BidMaker 中加入新的代码行，使用 new 关键字创建一个新的 CityViewHouse 对象，并将其引用值赋给对象变量 myHouse。此时，方法 BidMaker 的代码就会处于如下状态：

```
public BidMaker()
{
// Required for Visual J++ Forms Designer support initForm();
```

```
        // TODO: Add any constructor code after initForm call
                    myHouse = new CityViewHouse();
                }
```

现在，我们就完成了 CityViewHouse 对象的创建工作，该对象是由对象变量 myHouse 所指定的。

注意：说一个引用值是存储在引用变量中，是为了便于声明问题，实际上，程序员通常只是使用引用来指某个值（例如 new 关键字的返回值），或者是赋予了值以后的变量。

方法

方法描述了 Java 中的各种操作。在 Java 语言中使用方法，就像在其他计算机语言中使用函数、过程或子程序一样。然而，Java 的方法又与其他语言的函数有所不同，在 Java 中，方法总是属于 Java 类的一个组成部分。也就是说，在 Java 中，方法必须放在一个类中，如果事先没有已经创建的类，方法也就无处可寻了。

前面已经介绍了一些关于 Java 中方法的示例。WFC Label 类有一个 setText 方法和 getText 方法；同样，WFC Edit 类中也有这两个方法。与此同时，所有的事件处理程序都是方法（例如，事件 button1_click）。

另外，方法也可以访问修改符（例如 public 或者 private ）、返回类型（例如：Label.getText 返回一个字符串的值）、名称、参数列表和主体。

因为方法是存在于一个 Java 类的上下文当中，所以，方法（同属性或其他成员变量一样）在一个 Java 类的各个层次中是可继承的。这些继承下来的方法

在其他的类当中也可以使用。本书将在第五章中详细介绍有关 Java 语言继承性的详细内容。

提示：为了简单地考察一下继承性究竟对 Java 程序的组建有多大的作用，首先打开 Class Outline 窗口（在 View 窗口中选择 Other Windows 项，然后再选择 Document Outline 项即可）。在第五章中，我们将介绍如何在 Class Outline 窗口中建立方法。

下面，我们将向 CityViewHouse 类当中添加两个方法（setAskingPrice 和 getAskingPrice），同时也加入一个成员变量 askingPrice。首先，在 Class Outline 窗口中使用 Add Member Variable 对话框，向 CityViewHouse 类中加入一个名为 askingjPrice 的成员变量（这些操作与前面讲述的向 BidMaker 类中添加 myHouse 对象的过程相类似）。

若要向 CityViewHouse 类当中添加方法，右击 Class Outline 中的 CityViewHouse，在弹出菜单中选择 Add 项。Add Method 对话框如图 3-9 所示。

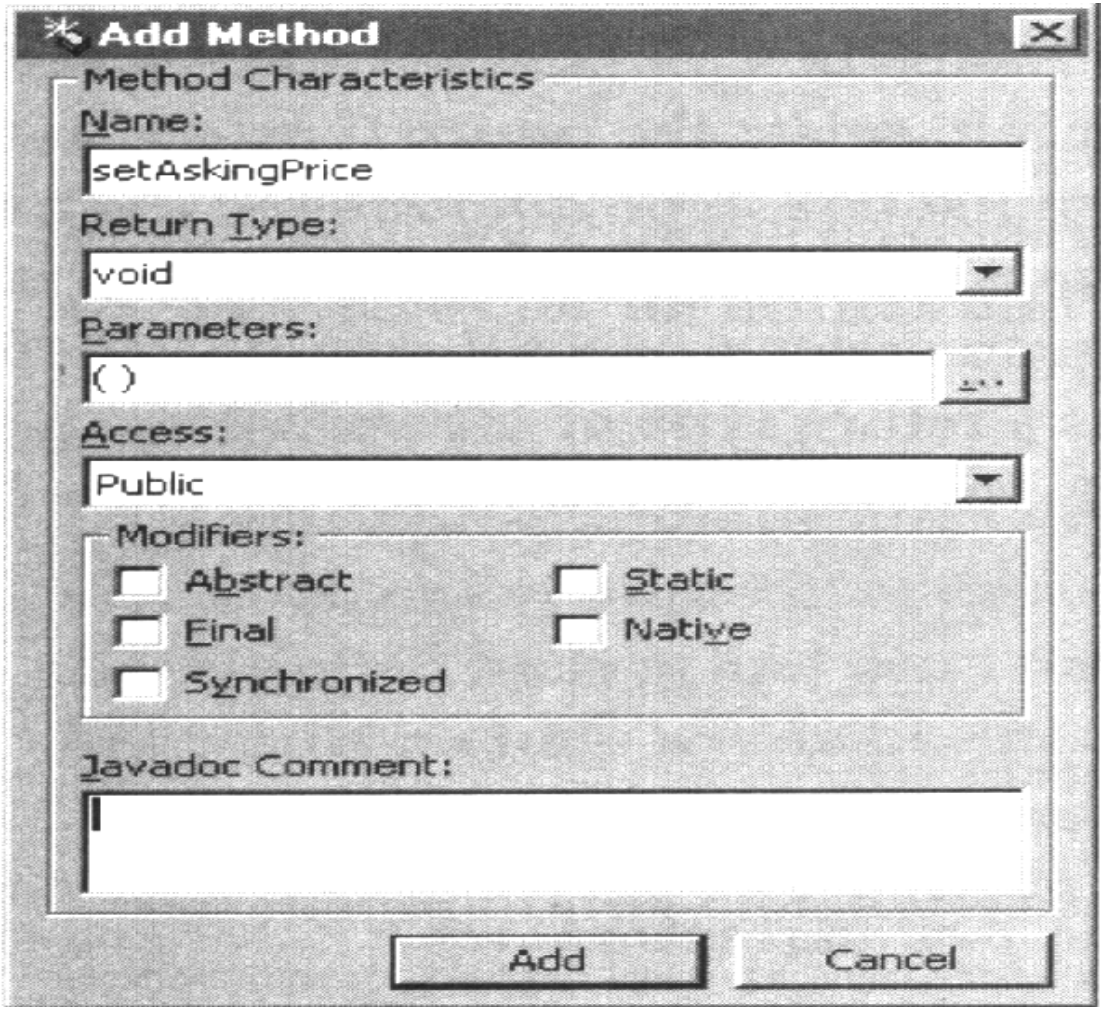


图 3-9　Add Method 对话框

将方法命名为 setAskingPrice，并把返回类型设为 void。按下 Parameters 按钮右侧的按钮。

如果要在方法中加入参数，则应该在 Parameters 框中双击，此时屏幕上就会出现一个 Edit Parameter List 对话框，如图 3-10 所示。

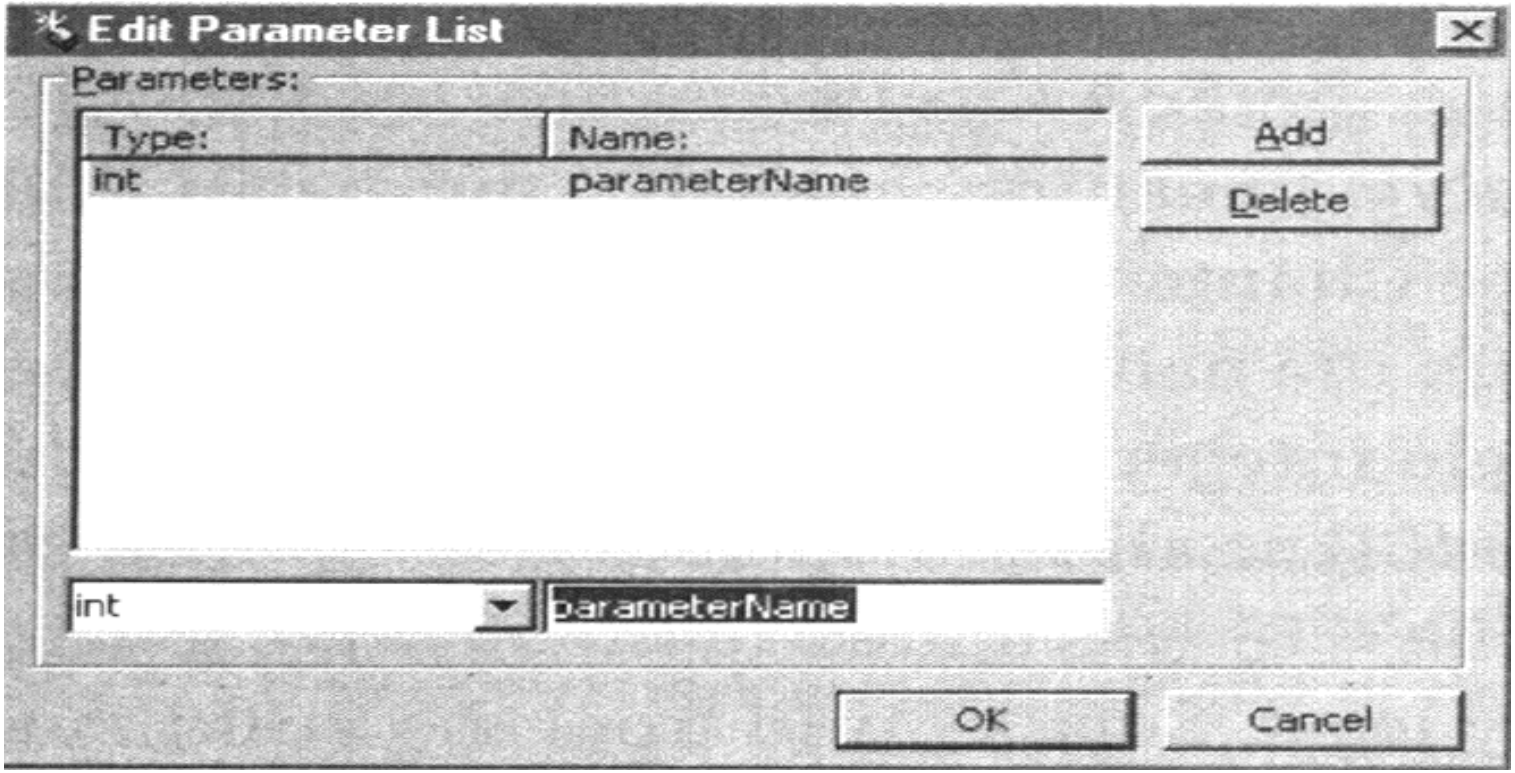


图 3-10　Edit Parameter List 对话框

在此对话框中按下 Add 按钮，在左下方的下拉式菜单中选择返回类型为 int。在这个下拉式列表框的右侧文本框中输入 amount，并按下 OK 按钮。选择访问方式为 public。重复以上的操作，可以创建第二个方法 getAskingPrice，这个方法的返回值类型也是整型 int，并且没有参数。当完成了上述所有的操作以后，关于类 CityViewHouse 的代码（在代码窗口中）如下：

```
public class CityViewHouse
{
```

```
private int askingPrice;
public int getAskingPrice()
{
    // TODO:Add your own implementation.
    Int returnValue;
    ReturnValue = 0;
    Return returnValue;
}

public void setAskingPrice(int amount)

{
    // TODO: Add your own implementation.
}
 }
```

在这个示例当中，对于方法 CityViewHouse.setAskingPrice，其修改符的访问方式为 public（意指在整个项目文件范围内都可以访问，关于 public 更详细的内容，将在第 4 章中讨论）；返回类型为 void（意指不返回任何值）；其名称为 setAskingPrice；参数列表由一个名为 amount 的整型值组成；其主体同所有其他方法的主体一样，以符号{}开始和结束。注意，在方法 getAskingPrice 的代码中，加入了一个简单的定义语句（returnValue=0），以使其在语法上是合法的。这样，在 Build 菜单中选择 Build 项，文件就会顺利地编译通过。

为了调用 getAskingPrice，或者 setAskingPrice 方法，需要以下两点：方法

主体的代码和 CityViewHouse 类的对象。我们将在这一阶段的练习中完成该方法的代码部分，在本章后面的内容当中，将介绍如何为 CityViewHouse 类创建一个对象。

标识符及其命名规则

到目前为止，我们已经学习创建了自己的类、对象、成员变量、方法，很自然，我们还要关心如何为它们更好地命名。在本节中，将详细介绍有关标识符的规则以及命名方式的建议。

标识符

和所有其他的编程语言一样，Java 有自己的内置程序元素或关键字，这些标明了编程语言的基本特征。它们由 Java 语言本身所提供，而不能由程序员定义或改动。关键字包括：class、new、public、static、this 和 null 等。本书附录列出了 Java 的所有关键字。其他的编程元素（或标识符）则是由程序员自己定义的，包括变量、类、方法以及对象的名称。一个合法的 Java 标识符应该遵循以下规则：

- 不能是关键字或保留字
- 必须由统一的编码标准的字符组成
- 必须以一个标准编码的字母为开始
- 标识符区分大小写（Bob 和 bob 是两个不同的标识符）

- 标识符中不能有空格（比如 HotDog 是不合法的标识符）
- 标识符实际上没有长度的限制

注意：由统一标准编码的字符组成，意思是指 Java 标识符可以包括英语字母表以外的字母，比如古斯拉夫语。统一编码字母包括下划线（_）和美元符号（$）以及其他一些符号。在本书中所出现的术语字符，指的是 ASCII 码的字符系列，基本上包括有大写字母 A 到 Z、小写字母 a 到 z、数码 1 到 9 及标点符号。

命名规则

标识符遵循一定的规则，可以促使程序员避免语法错误，同时，由于规则的一致性，可以使得所有知道该规则的用户在总分类中很容易地查出某一项所在的位置。下面列出了创建自己的代码时应遵循的规则，这些规则在本书后面介绍的示例中也将会应用：

- 类的名称以大写字母开头，并且，该大写字母是类名称的一部分，另外,名称中间也可以有大写字母（比如 CityViewHouse）。
- 成员变量与局部变量的名称以小写字母开头，名称中间可以有大写字母（比如 bidAmount）。
- 成员变量与局部变量的名称使用名词（比如：bidAmount）。
- 方法名称以动词开头，后面的字母与成员变量及局部变量的名称相同（比如：getAskingPrice）。
- 在一般情况下，避免使用下划线符号，而在方法名称当中特别使用，因为事件处理程序使用下划线将控件名和事件名隔离开来（比如：

button1_click）。

使用以上命名规则的确需要一些时间来适应，但是，一旦成为习惯，就会发现，这些努力是非常值得的。

构造器

每当一个对象被创建（或者被初始化）之后，就会被自动激活，并调用一个特殊的方法。这个方法就是构造器。在 Java 中，每个类都至少有一个构造器。构造器可以确保用户正确地创建类的对象，同时，构造器也会对对象作初始化工作。

前面，在我们使用 new 关键字时，实际上已经暗中调用了构造器。在使用 new 关键字时，我们已经看到了类名后面的括号，下面将介绍调用以后的代码。

缺省构造器

首先考察一下在使用 CityViewHouse 类时，我们是如何使用 new 关键字的。为 CityViewHouse 创建对象的代码是在类 BidMaker 中，其内容如下：

```
myHouse=new CityViewHouse(); //creates a CityViewHouse object
```

注意，在 new 关键字的后面，是类的名称，其后是一对括号。这个括号的意思是指，该类的一个方法也同时存在，其名称与类的名称相同。这个方法就是该类的构造器，它可以确保类的对象符合类的设计。如果此时回到定义

CityViewHouse 类，那么，就不会看到有一个名为 CityViewHouse 的方法，这是因为，如果并没有在类的定义中加入构造器方法，Java 语言会自动为用户提供。这个由 Java 自动提供的构造器就是所谓的缺省构造器。由于没有在外部提供我们自己的构造器，CityViewHouse 就有了一个缺省构造器。

缺省的构造器实际上是一个占位符：它确保每个 Java 类都至少有一个构造器方法（该方法应符合类的设计）。CityViewHouse 的缺省构造器代码行如下：

```
CityViewHouse()
{
}
```

与所有的构造器一样，该方法有一些自身的特征。首先，它没有返回类型，这又不同于返回类型为空的情形。所有的构造器方法就是没有返回类型，另外，构造器的名称必须与其所在类的名称相同。只有这样，用户才可能从一个类定义中将其提取出来。

因为我们的示例中采用的是缺省构造器，所以，在这个方法当中没有任何代码。当然，由用户自己编写的方法中是有代码的（下文中将详细介绍）。

缺省构造器存在的根本目的在于，为了使 Java 语言的继承功能得以正确的实现。在本书第五章中，将详细介绍关于继承性的问题。

现在，我们可以为所创建的类的构造器添加一些新的内容了。

添加构造器

现在，我们可以对 CityViewHouse 对象作进一步的操作了。如果想要变量 askingPrice 的初值为 $100000，则应在 CityViewHouse 类的构造器方法中加入如

下代码：

```
public CityViewHouse () // constructor for class CityViewHouse
{
askingPrice = 100000;
}
```

到目前为止，类当中就包含了方法的定义，该方法将每个 CityViewHouse 对象的初值设定为 100000 美元。无论何时，一旦 CityViewHouse 对象的创建工作完成，用户所提供的构造器方法就会被调用。因为使用了前文所述的命名规则，即类名大写，并以小写字母开头方法名，所以，该构造器的方法名在类中应是唯一的以大写字母开头的名称。

这样，无论何时，一旦创建了一个新的 CityViewHouse 对象，要价的初始值都会被预置为 100000 美元。这比要价的缺省值为 0 要好一些，但是，仍比不上在每次创建了对象以后，都能对每个对象提出不同的要价。下面将介绍如何实现此项功能。

带有参数的构造器

带有参数的构造器能够实现这样的功能：当创建一个新对象时，创建类的代码可以将一些指定的值传递给构造器方法。现在，我们可以修改 CityViewHouse 类，使构造器中包含参数。修改之后，构造器中的代码将如下：

```
public class CityViewHouse
{     // other member variables as before ...
public CityViewHouse (int amount)
```

```
    {
        askingPrice = amount;
    }
     }
```

作了如上修改的构造器，现在需要一个参数，因此，此时如果在创建CityViewHouse 对象时，不事先设定要价的初值，构造器方法就将是不合法的。也就是说，在创建对象的同时，必须特别指定构造器中的参数值。在 Build 菜单中选择 Build 项，信息窗口中就会显示出由于改动而引起的错误。如果此时屏幕上并没有显示任务列表窗口，选择 View 菜单中的 Other Windows 项，然后再选择 Task List 项，屏幕上就会显示出任务列表窗口。窗口中会有如下信息：

```
Undefined name 'myHouse'(J0049)
```

在任务列表窗口中双击这条信息，又会显示出相应的代码行，如下：

```
myHouse=new CityViewHouse();
```

在创建对应新的构造器方法的对象时，除了在类名之后的括号中应加入数值以外，其他的语法规则与前文所讲述的相同。将这个错误的代码行进行修改，在括号中加入要价的初值（整型值）：

```
myHouse=new CityViewHouse(250000);
```

构造器引入了参数之后，就可以创建多个对象，每个对象在创建时都将被设定要价的初值。到目前为止，我们已经成功地创建了 CityViewHouse 对象，下面就可以使用对象来调用一些方法。在 Java 语言中，调用的方法的一般形式

如下：

```
object.methodname()
```

在本篇所讨论的示例中，其具体形式应为：

```
myHouse.setAskingPrice(260000); // an increase in asking price
```

对于 getAskingPrice 有一个返回值，其调用方式就应为：

```
price= myHouse.getAskingPrice(); // what is current asking price?
```

实验 3-1：修改 Bidmaker 项目文件

实验概述

在本次实验练习当中，我们将学习使用：

- 修改通用类的方法
- 调用通用类的方法

在本次实验题中，将就前面所讲述的有关代码内容作进一步的练习，在实验的过程当中，用户可以调用自己创建的类的方法。

实验设置

1. 启动 Visual J++。
2. 打开 BidMaker 项目文件（Chapter03\Lab3-1\BidMaker.sln）。

实验步骤

1. 在 Project Explorer 窗口展开 BidMaker 项目文件，该项目文件中包括有 BidMaker.java 和 CityViewHouse.java 两个文件。
2. 在 Project Explorer 窗口右击 BidMaker.java，在弹出菜单中选择 View Code 项。在任务列表窗口中会出现一个 TODO 项，提示用户对 CityViewHouse.java 文件作必要的修改。双击任务项，系统会切换到需要改动的代码位置处。
3. 在 CityViewHouse.java 文件中重复第二步操作，直到符合要求为止。
4. 完成了上述操作以后，便可运行项目文件。
5. 当 BidMaker 窗体出现时，在文本框中输入一个整型数字（不要输入美元符号），然后，单击 Accept Bid 按钮。
6. 将本次实验的工作与本章中的示例 Chapter03\Sol3-1\BidMaker.sln 比较一下，看看有什么不同。

下一步

到目前为止，我们已经熟悉了向项目文件中添加类的一些必要步骤，下面将介绍如何在 Visual J++环境下对类进行预定义工作。在下一部分中，将介绍如何使用这些类在应用程序中创建标准 Windows 对话框。除此之外，还将介绍如何向应用程序中添加附加窗体。

添加对话框和附加窗体

Visual J++为用户提供了一些预定义类，它们可以使用户更加容易地创建一些功能强大的基于 Windows 的应用程序。我们首先介绍一个十分便利的 MessageBox 类，它可以使系统向用户显示报警对话框。

MessageBox 类

MessageBox 类可以将信息显示到屏幕上。当用户使用应用程序进行操作时，在屏幕窗口中显示报警信息是最简单的与用户交互的方法。MessageBox.show 方法在显示报警信息时有三种可选用的格式。

启动 Visual J++，打开项目文件 Chapter03\MessageBox\MessageBox Examples.sln.运行项目文件，并试着按窗体上的每一个按钮。每一个按钮都对应一个事件处理程序，并会打开一个相应的 Message Box 对话框。完成了以上的工作以后，按下窗口右上角的 Close 按钮。我们将查看一下每个按钮所对应的事件处理程序，以及各个版本的 MessageBox.show 方法是如何被调用的。

注意： MessageBox 类是在 com.ms.wfc.ui 中定义的。在 Visual J++创建的代码窗体的顶端，有一个十分重要的声明语句：

```
import com.ms.wfc.ui.*;
```

如果在 Java 源代码文件当中没有这一行语句，那么方法将以以下方式被调用：

```
com.ms.wfc.ui.MessageBox.show(Hi User!); //long name!
```

如果想要在自己的 Java 文件当中使用 MessageBox 类，就要在代码中包括这条重要的声明代码，或者使用长方法名。

源代码演示了调用 MessageBox.show 方法的三种格式。现总结在表（3-1）中：

表 3-1　messageBox 类方法示意表

MessageBox 格式	说明
MessageBox.show(String text)	带有消息的消息框
MessageBox.show(String text, String caption)	带有消息和窗口的消息框
MessageBox.show(String text, String caption, int style)	自定义的消息框

实现第一种方案的代码如下：

```
private void simpleBtn_click(Object source,Event e)
{
MessageBox.show(″ Simple MessageBox″ );
}
```

在前面的学习当中，我们已经了解到，simpleBtn_click 是一个事件处理程序。该程序负责处理当一个名为 simpleBtn 的按钮被按下时将要发生的事件。该事件处理程序的进程在下面的窗口中显示，如图 3-11 所示。

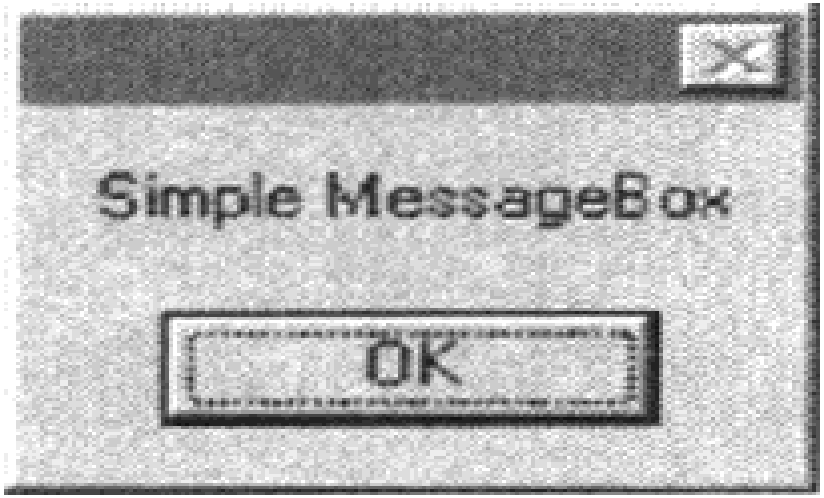


图 3-11　一个简单的消息窗口

我们发现这个消息窗口没有任何标题。如果使用第二个方案情况，就不会这样，只需在代码中作一些细小的改动，就可以为消息窗口加上标题。

```
Private void withTitleBtn_click(Object source,Event e)
{
MessageBox.show(″ MessageBox with Title″ , ″ MessageBox Title″ );
}
```

图 3-12 示意了改动以后的消息窗口。

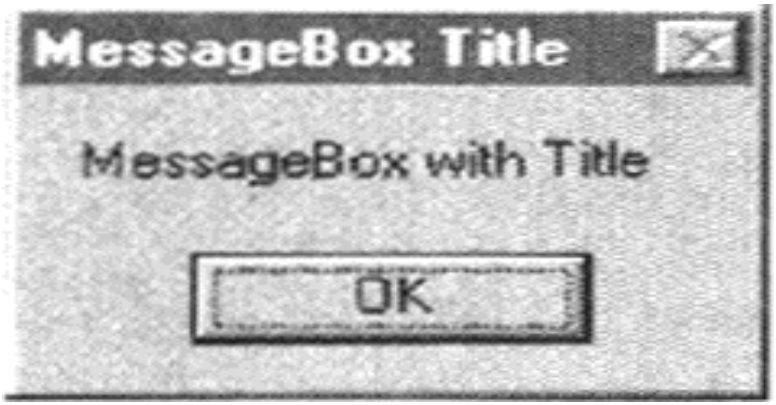


图 3-12　带有标题的消息窗口

调用 MessageBox.show 方法的第三个方案为我们提供了创建各种外观和特征的消息窗口的方法。如果用户希望自己的消息窗口有一个 OK 按钮和一个 Cancel 按钮，并且同时希望在显示消息的同时，也附带显示 Windows 气球状的问号图标，除此之外，当消息窗口关闭时，还希望其缺省选项为 OK 按钮（缺省的 OK 按钮是高亮的，并且，即使不按下该按钮，而只是按回车键，该按钮也将被激活)。总而言之，希望创建一个如图 3-13 所示的消息窗口。示例 MessageBoxExample 项目文件中的 withIconsBtn_click 事件处理程序显示了 MessageBox.show 调用的情形。

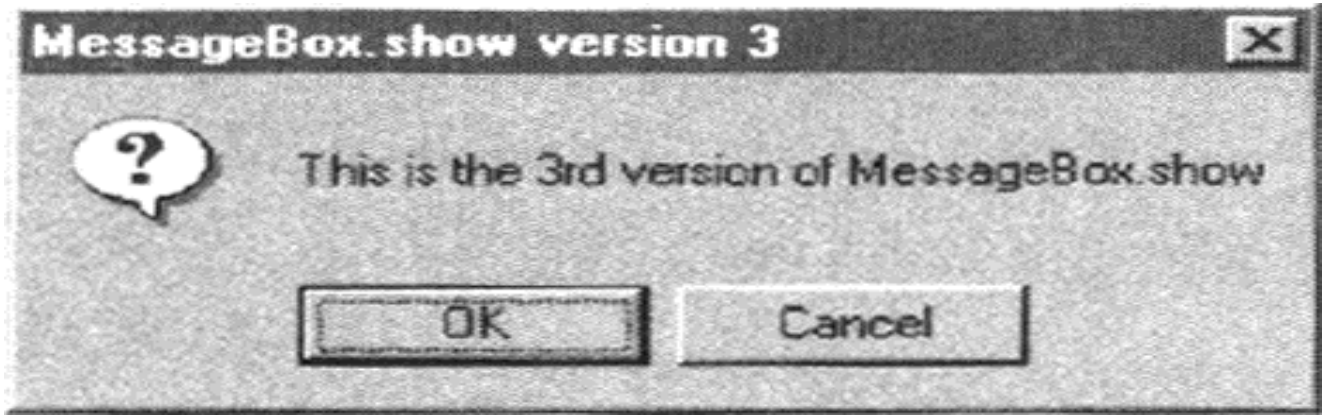


图 3-13　一个出色的消息窗口

对应上述方案的 Java 代码内容如下：

```
private void button1_click(Object source,Event e)
{
MessageBox.show(″ Are you sure you’re done? ″ , ″ Tread Lightly″ ,
                MessageBox.YESNO +
                MessageBox.ICONQUESTION +
                MessageBox.DEFBUTTON1);
}
```

前两个参数同前面讲述的方法调用的参数一样，只是最后一个参数相对复杂一些，下面将对其作深入的讨论。

MessageBox.show 样式

show 方法的最后一个参数是样式值，有时也称为标志。使用样式值，可以指定消息窗口的三条信息：

- 框中按钮的个数及其名称。
- 消息旁边将显示的特殊图标。
- 框的缺省按钮。

可以直接使用数来代表这些标记，不过，MessageBox 类为我们提供了这些类的名称。表 3-2 列出了设置消息窗口所用的各种按钮的样式值。现在，可以从列表当中选取任意一个样式值为 MessageBox.show 方法所调用。列表中的一些标志不能在一起使用，例如：不能将 MessageBox.OK 和 MessageBox.YESNO 放在一起为 MessageBox.show 方法所调用。

表 3-2　按钮布置方案及 MessageBox 类中对应的按钮标记

按钮布置方案	按钮名（代码内容）
只有 OK 按钮	MessageBox.OK
OK 和 Cancel 两个按钮	MessageBox.OKCANCEL
Abort，Retry 和 Ignore 按钮	MessageBox.ABORTRETRYIGNORE
Yes，No 和 Cancel 按钮	MessageBox.YESNOCANCEL
Yes 和 No 按钮	MessageBox.YESNO
Retry 和 Cancel 按钮	MessageBox.RETRYCANCEL

回顾一下前面对 MessageBox.show 方法的调用过程，很容易就会通过逐个重新设定样式值，而了解它们是如何工作的。下面，我们给出如下所示的代码，由按钮布置来开始说明：

```
MessageBox.show(″ Are you sure you’re done? ″ ,
    ″ Tread Light1y″ ,MessageBox,YESNO);
```

注意，只有在按钮布置标记已经指定时，这种调用才会正常运行。如果事先没有选取一个图标值，程序会自动选取一个缺省的图标值，因此，用户可以不必预先选取图标值。下一步，就该我们要确定显示哪个图标了。为了做到这一点，首先要确定与图标相对应的标记，然后，在调用中加入对应的样式值。在本示例中，我们想要的是问号标记图标，所以，现在代码将如下所示：

```
MessageBox.show(″ Are you sure you’re done? ″ , ″ Tread Lightly″ ,
MessageBox.YESNO + MessageBox.ICONQUESTION);
```

应该特别注意，这两个标记是由加号联在一起，而不是由逗号隔开的。表 3-3 中列出了所有消息窗口的图标代码。可以从该列表中选择任意一个，并将其放入 MessageBox.show 方法的调用中去。

表 3-3　MessageBox 类的图标代码以及对应的说明

图标代码	图标说明
Message.Box.ICONASTERISK	气球状的框中有一个 i
Message.Box.ICONERROR	红色圆圈，还有一个白色的 X
Message.Box.ICONEXCLAMATION	感叹号
Message.Box.ICONHAND	红色圆圈，还有一个白色的 X
Message.Box.ICONFORMATION	气球状的框中有一个 i
Message.Box.ICONQUESTION	问号
Message.Box.ICONSTOP	红色圆圈，还有一个白色的 X
Message.Box.ICONWARNING	感叹号

最后一个样式值是在消息窗口第一次出现时缺省按钮的指示。缺省按钮并没有名称编码，而只是放置到消息窗口当中。这就意味着，虽然在消息窗口中有一个 Yes 按钮，并没有与之对应的标记（YESNO 按钮是缺省的按钮），此时，标记已经设定信息窗体中的第一个按钮为缺省按钮。当信息框出现在屏幕上时，用户可以通过按下 Tab 键选择自己想要的缺省按钮。并在下一次消息窗口出现时，用户此时所选的按钮仍是作为缺省按钮。

表 3-4　缺省按钮和相应的 MessageBox 类中的代码

缺省按钮	按钮标志
Button 1	MessageBox.DEFBUTTON1
Button 2	MessageBox.DEFBUTTON2
Button 3	MessageBox.DEFBUTTON3

我们指定信息框中的第一个按钮为缺省按钮（在本示例中为 Yes 按钮），此时，我们将代码加入到对 MessageBox.show 方法的调用中去，其具体内容如下：

```
MessageBox.show(″ Are you sure you are done? ″ , ″ Tread Lightly″ ,
                    MessageBox.YESNO +
                    MessageBox.ICONQUESTION +
                    MessageBox.DEFBUTTON1);
```

MessageBox.show 的返回值

到目前为止，我们已经介绍了，如何利用不同的按钮和图标的布置来定制

一个消息框，同时也介绍了如何选取一个缺省按钮。下面可以考察一下用户在使用消息框时的一些情况。当用户在消息框中进行操作时，系统会作出相应的响应，这些响应实际上是对应了 MessageBox.show 的不同返回值。这些返回值同相应的名称一起，使得系统可以准确地确认用户所按的是哪个按钮，表 3-5 列出了这些返回值及其名称。

表 3-5　返回值及 MessageBox 类中的相应的代码

按下的按钮	返回值
OK	DialogResult.OK
Cancel	DialogResult.CANCEL
Abort	DialogResult.ABORT
Retry	DialogResult.RETRY
Ignore	DialogResult.IGNORE
Yes	DialogResult.YES
No	DialogResult.NO

在事件处理程序中，使用这些返回代码，其内容便为：

```
private void button1_click(Object source,Event e)
        {
int result;
```

```
result = MessageBox.show(″ Are you sure you’re done? ″ ,
        ″ Tread Lightly″ ,
         MessageBox.YESNO +
         MessageBox.ICONQUESTION +
         MessageBox.DEFBUTTON1);

If (result == Dialogresult.YES)
      {
      Application.exit();
      }
else if (result == DialogResult.NO)
      {
        // nevermind
      }
 }
```

从上面的代码中可以看出，当完成了 MessageBox.show 方法调用的工作以后，下一步就是对返回值的检验了。如果用户选择按下的 Yes 按钮，那么结果值就等于 DialogResult.NO，系统就会调用 Application.exit()方法，该方法可以终止应用程序。这与单击窗口右上角的关闭按钮操作是十分类似的。如果用户选择按下 No 按钮，那么系统将不进行任何动作（当然，系统本身仍会在 MessageBox.show 窗体中正常执行，消息框将从屏幕上消失）。如果当用户选择按下按钮时，系统不会做任何工作，那么，在代码当中加入这部分的检验代码，就程序运行的要求上看，是根本没有必要的。这样做的目的是为了便于后

来的程序员能够容易地读懂程序代码，以及很方便进行修改工作。

下面就 MessageBox.show 方法作一个简单的总结。首先，它具有三种格式（简单格式、拥有标题格式和使用标记格式）。MessageBox 类为所有的标记及返回值提供了名字。返回值是通过 MessageBox.show 方法对用户选择按下某个按钮作出相应的反应而实现的。

在此之后，我们将向项目文件中添加一个 MessageBox.show 方法的调用。

ColorDialog 类

ColorDialog 类为用户提供了给屏幕上显示的元素选择颜色的方法。当屏幕上显示有 Display 对话框时，ColorDialog 就是处于工作状态。打开 Display 对话框，只需在 Windows 任务栏中按下 Start 按钮，在弹出菜单中选择 Settings，然后再选择 Control Panel，在 Control Panel 中双击 Display 图标。一旦打开了 Display 对话框，就可以选择 Appearance 选项卡，按下标有 Color 的框体右边的向下箭头，然后按下 Other 按钮。屏幕上就会出现如图 3-14 所示的 Color 对话框。

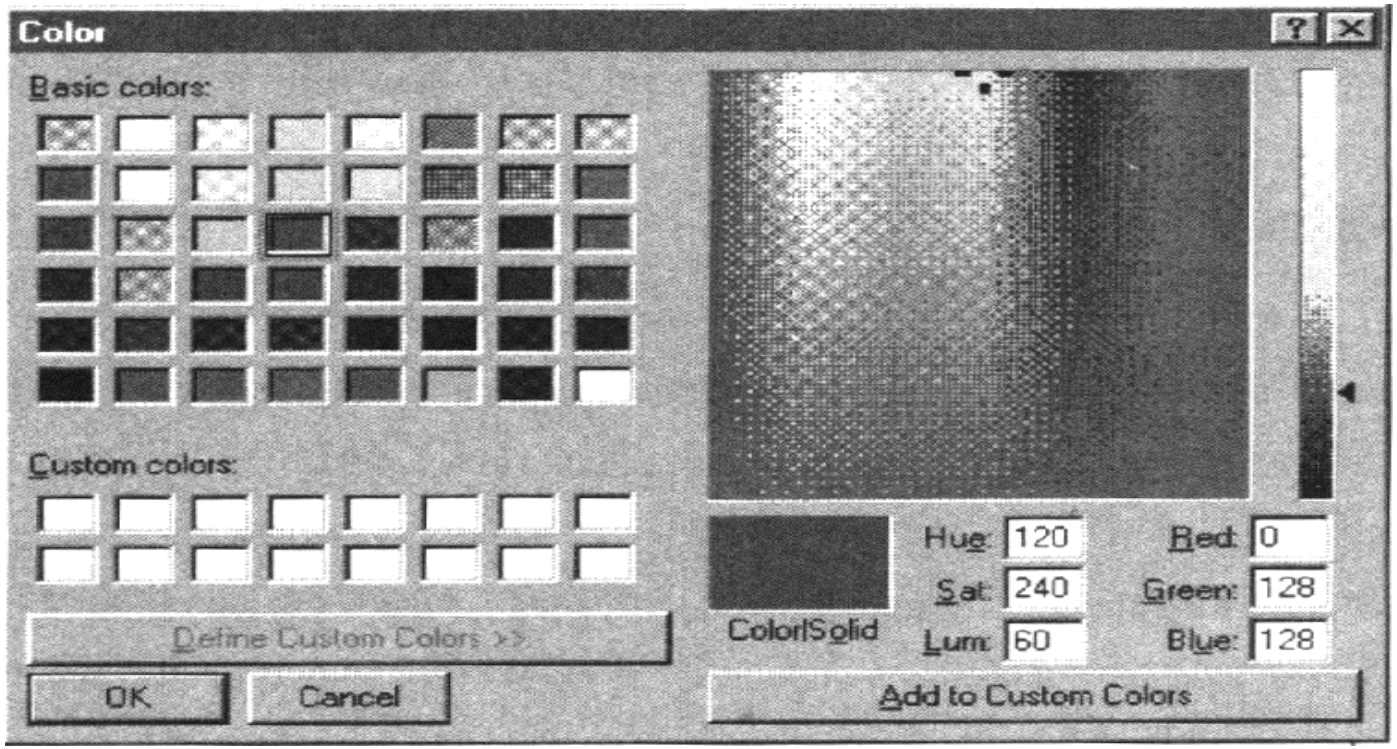


图 3-14　Color 对话框

当我们在一个 Java 程序当中使用 ColorDialog 对象时，该对话框就会出现。向下执行，并关闭 Color 对话框。在一个 Java 程序中打开此对话框是很有趣的，如果想要从 Color 对话框中选取，并设置应用程序中窗体的背景颜色，只需按下几个按钮即可。为了完成该项工作，我们需要有以下几项支持：

- 一个 ColorDialog 对象。
- 两个 Color 对象。
- 一个整型变量。

另外，为了完成窗体背景颜色的设置，还需要使用下列方法：

- ColorDialog.setColor 方法（该方法用于在一个 ColorDialog 对象中，选择并设定初始颜色）。
- ColorDialog.showDialog 方法（该方法用于激活 ColorDialog 对象）。
- ColorDialog.getColor 方法（该方法用于检索 ColorDialog 对象中选择的颜色）。
- getBackColor 方法（该方法用于检索当前窗体的背景颜色）。

注意，getBackColor 方法是属于窗体 Form 类中的方法，而不是 ColorCialog 类中的方法。另外，在许多其他的 Java 类当中，也包含有 getBackColor 方法。

一个 ColorDialog 对象

我们先创建一个 ColorDialog 类的对象。首先应该声明该对象的引用，然后再创建这个 ColorDialog 类的对象（使用 new 关键字）。其代码内容如下：

```
ColorDialog myColorDialog;   // a reference to a ColorDialog object
MyColorDialog = new ColorDialog(); // creates the ColorDialog object
```

如果此时想要显示 Color 对话框，只需调用新建对象的 ShowDialog 方法即可。

两个 Color 对象

在使用 Color 对话框时，如果想要存储用户选取的颜色，就需要一个对应该类的对象，也就是需要一个 Color 类的对象。此时，不必对 Color 类本身作更进一步的操作，而只需创建一个与其对应的引用即可，其代码内容如下：

```
Color currentBackgroundColor;   // a reference to a Color object
```

```
Color newBackgroundColor;   // ditto
```

布尔逻辑变量

用户对 Color 对话框的操作有两种选择方式，可任选其一：按下 Ok 按钮，或者是按下 NO 按钮，都会由 ColorDialog.showDialog 方法返回一个整型值，其代码如下：

```
int result;
```

getBackColor

到目前为止，我们已经实现了变量存储工作，很快就要完成我们的任务了。首先使用所选择的窗体背景颜色设置 Color 对话框，这样，getBackColor 方法就会为 Color 对象设定一个背景颜色，其代码内容如下：

```
currentBackgroundColor=getBackColor（）；
```

这样，程序就会得知我们为窗体背景所设定的颜色了。

ColorDialog.setColor

下一步，就应该设法将窗体背景颜色的消息传递给 ColorDialog 对象。这项工作是由 ColorDialog 类的 setColor 方法来完成的。其代码内容如下：

```
myColorDialog.setColor（currentBackgroundColor）；// initial color set
```

ColorDialog.showDialog

注意，所有的设定都要允许用户为窗体设定颜色。为了激活 Color 对话框，

系统就需要调用 showDialog 方法。因为在 Color 对话框关闭时，我们需要知道用户究竟按下了哪个按钮，所以，就要将 showDialog 方法的返回值赋给 result 变量，以便得到返回值。如果 result 变量的值等于 DialogResult.OK，系统就会改变窗体的背景颜色，否则，如果用户按下的是 Cancel 按钮，系统将不改变窗体的颜色。其代码内容如下：

```
if（result==DialogResult.OK）
```

ColorDialog.getColor

一旦用户选取了一种颜色，并关闭了 Color 对话框，我们就要查看用户是否按下了 OK 按钮，如果用户按下了 OK 按钮，就要使用 ColorDialog.getColor 方法，来识别用户选取的是哪种颜色。在此之后，还要在 newBackgroundColor 颜色对象中存储一个新的颜色对象的引用。其代码内容如下：

```
if (RESULT == dIALOGrESULT.ok)
{
newBackgroundColor = myColorDialog.getColor();
setBackColor(newBackgroundColor);
}
```

将上述各个方法组织在一起

将以上这些零碎的代码段组织在一起，使其对应一个 Click 事件处理程序，就得到了一个允许用户改变窗体颜色的应用程序代码段。其所有的代码如列表 3-1 所示：

列表 3-1　显示 Color 对话框的事件处理程序

```
 private void button1_click(Object source, Event e)
 {
ColorDialog myColorDialog;
// a reference to a ColorDialog object
myColorDialog = new ColorDialog();
// creates the ColorDialog object

Color currentBackgroundColor;
// a reference to a     Color object
Color newBackgroundColor; // ditto
Int result;

CurrentBackgroundColor = getBackColor();
MyColorDialog.setColor(currentBackgroundColor);
// initial color set
result = myColorDialog.showDialog();
if (result == DialogResult.OK)
{
        newBackgroundColor = myColorDialog.getColor();
        setBackColor(newBackgroundColor);
}
 }
```

通过了上面内容的学习，我们就可以启动 Visual J++，来创建一个新的项目文件，并按自己的喜好为其命名，然后向项目文件中加入窗体，并可以向窗体中添加按钮控件。如果将该按钮命名为 button1，可以在该窗体的代码窗口中输入事件处理程序的代码。运行该项目文件，并使用它改变窗体的背景颜色。

FontDialog 类

采用与使用 ColorDialog 类改变窗体颜色相类似的操作步骤，也可以通过使用 FontDialog 类来改变字体。与类相似，FontDialog 类使用方法来激活对话框，同时向系统返回一个整型值，使得系统能够识别用户在关闭对话框时是否按下了 OK 按钮。

FontDialog 类在显示一个文本时，是利用 Font 和 Color 类来存储适当的字体和颜色的。然后，我们就可以使用 FontDialog 对象，来改动任何一个指定文本的字体和颜色了。根据 Visual J++的约定，可以用来访问字体和颜色属性的方法包括：getColor，setColor，getFont 和 setFont。下面，我们将组建一个实例，在这个实例中，用户可以为标签的文本属性选取一个新的字体或颜色。这些操作仍然是对应于窗体中的一个按钮。下面就开始组建这个实例。

首先，我们设定三个引用（一个针对文本的颜色，一个针对文本的字体，最后一个针对 FontDialog 对象）和一个整型变量。这个整型变量用来存储 FontDialog.showDialog 方法的 result 返回值。如果 showDialog 方法的返回值是 DialogResult.OK，则说明用户选择的是 OK 按钮，否则，说明用户选择的是其他按钮（比如：Cancel 按钮）。代码内容如下：

```
Private void button1_click(Object source, Event e)
```

```
{
Color fontColor;
Font font;
FontDialog fontDialog;
Int result;
```

使用 Label 对象的 getFont 方法，可以查看在激活对话框之前标签的字体，用同样的办法，也可以查看字体的颜色，其代码内容如下：

```
font = label1.getFont();
fontColor = label1.getForeColor();
```

下一步，我们将创建一个 FontDialog 对象。一旦创建了 FontDialog 对象，就可以利用 showDialog 方法激活对话框。我们可以将 showDialog 方法的返回值 result 存储到预先设定的整型变量中，其代码内容如下：

```
fontDialog = new FontDialog();
result = fontDialog.showDialog();
```

一旦我们检查了用户是否按下了 OK 按钮（返回值 result 是否为 DialogResult.OK），我们就将实现利用 FontDialog 对象对字体和颜色的设定。其代码内容如下：

```
if (result == DialogResult.OK)
{
    font = fontDialog.getFont();
    fontColor = fontDialog.getColor();
```

此后，就可以获得新的字体和颜色，并利用它们来改变 Label 对象的属性，

其代码内容如下：

```
                label1.setFont(font);
                 label1.setForeColor(fontColor);
        }
    }
```

这样，用户就完成了对标签对象的字体及颜色的设定。

如果想要查看 FontDialog 的工作情况，可以启动 Visual J++创建一个新的项目文件。然后，利用 ColorDialog 检查以前所作的工作，此后再向项目文件中添加一个新的窗体。该窗体需要一个按钮控件（名为 button1）和一个标签控件（名为 label1）。在窗体中加入有关 button1_click 事件的代码。然后，运行项目文件，并利用它来改变标签的字体和颜色。

添加第二个窗体

很少有应用程序只是需要一个窗口。在项目文件当中添加第二个窗体非常容易。首先在 Project 菜单中选择 Add Form 项，来创建一个新的窗体（在此之后，如果此时有已经建立的窗体（用户本人或其他人预先创建好的窗体），可以选择 Existing 选项卡，将该窗体加入到项目文件中)。

因为我们已经通过 Add Form 菜单访问了 Add Item 对话框，所以，此时 Form 文件夹就已经被选中了。选择带有标签的窗体图标，便可以为项目文件创建一个新的窗体。另外,,在 Project Explorer 窗口中右击或者单击任务栏上的 Add Item 按钮（该按钮通常位于任务栏上左边第二个按钮处），可以同样实现上述操作。

显示第二个窗体

激活一个窗体对象，实际上也就是为窗体类创建一个对象，并调用该窗体的 show 方法。假定目前我们的项目文件已经有了两个窗体。在第一个窗体中，我们放置一个按钮，可以通过这个按钮来访问第二个窗体。其代码内容如下：

```
public class Form1 extends Form
{
private void button1_click(Object source, Event e)
{
    Form2 secondFrm;   // a reference to a Form2 form object
    SecondFrm = new Form2();   //   create the new Form2 form object
    SeconaFrm.show();   // display the new Form2 form object
}
// class Form1 continues...
```

事件处理程序所要做的事情是，首先声明这个新窗体的引用，然后，创建该窗体的对象（使用 new 关键字）,最后调用该窗体的 show 方法。一旦新的窗体显示在屏幕上，输入的焦点就将在新窗体中，用户也自然将使用新窗体与系统交互。当用户完成了与该窗体的交互时，可以使用同前面讲述的与关闭其他窗口类似的办法，即单击窗体顶端标题栏右端的关闭按钮，来关闭该窗体。另外一种关闭窗体的方法是调用窗体的 dispose 方法，或者调用 hide 方法。在为第一个窗体添加了按钮以及相应的代码方法之后，可以尝试在第二个窗体中加入一个按钮，来调用下面所示的任意一种代码方法：

```
public class Form2 extends Form
```

```
 {
private void button1_click(Object source,Event e)
{
    hide();   // or dispose();
}
 // class Form2 continues...
```

dispose 方法与 hide 方法的区别在于，hide 方法只是使窗体在屏幕上不再显示，并不将窗体从计算机的内存中调出，而 dispose 方法则是真正将窗体关闭，释放窗体原来所占用的内存空间。如果用户不久就会又用到这个窗体，那么选择 hide 方法是适宜的。

注意： 查看第二个窗体的属性有些复杂。在第一个窗体被选中时，便不能在访问 Properties 窗口查看第二个窗体的属性。要想查看第二个窗体的属性，必须首先在 Project Explorer 窗口中选中第二个窗体，然后单击 View Designer 按钮，此后便可以访问 Properties 窗口查看该窗体的属性。

注意，每个窗体都可以有自己的名为 button1 的按钮，每个窗体的文本（Text）属性相当于窗体的窗口标题栏，所以，在应用程序运行的过程当中，可以分别指定。

为了检验上述操作，可以运行项目文件，按下相应的按钮显示和隐藏对象 Form2。单击 Form1 窗口顶部的 Close 按钮，来关闭应用程序。

实验 3-2：准备出售的房子

到目前为止，我们已经学习了如何使用 Visual J++环境与预定义对话框类和组建多窗体应用程序。下面，让我们应用前面所学的知识来创建一个 Java 应用程序。

实验概述

在本次实验中，我们将练习以下内容：

- 创建对象
- 使用 FontDialog
- 使用 ColorDialog
- 使用 MessageBox:
- 创建并显示多个窗体

本次实验所要创建的项目文件中有两个窗体。第一个窗体当中有三个按钮：第一个按钮用来改变字体和字体的颜色；第二个按钮用来改变窗体的背景颜色；第三个按钮用来查看第二个窗体的变化情况。在第二个窗体中绘有一个房屋的广告画。另外，在第二个窗体中还有一个按钮，使用该按钮，用户可以在广告中给出对出售房屋的标价。如果用户按下此按钮，屏幕上就会显示一个消息框，上面显示了关于房屋的具体消息。

实验准备

1. 启动 Visual J++。
2. 打开 HouseForSale 启动项目文件（Chapter03\Lab3-

2\HouseForSale.sln）。

3. 查看 Project Explorer 窗口，如果在项目文件图标下没有发现任何东西，则要单击加号标记，以显示与该项目文件相联系的文件。
4. 选择 CustomizeAd.java 项，并在 Project Explorer 窗口的顶部单击 View Code 按钮。

实验步骤

1. 代码段中含有许多带有 TODO 标记的注释。每个 TODO 注释在程序员完成代码书写之后，都将提示代码的正确格式。

在代码窗口中写下这些有关代码：

- 创建、修改和显示窗体。
- 为类添加一个成员变量。
- 显示 Color 对话框，判断用户是否按下 OK 按钮，如果是，则相应改变属性。

2. 认真查看代码内容，看看是否能从这些新的代码中得到一些好的提示（注意，特别要认真检查 pickFontBtn_click 方法）。另外，看看 HouseAd.java 文件中的代码，看看各个方法在其中的形式，对于学习也是大有好处的。
3. 在处理 CustomizeAd.java 文件时，可以根据 HouseAd.java 文件源代码的注释内容改动 HouseAd.java 文件来显示 Message Box 对话框。
4. 完成了上述改动之后，按下 Start Without Debugging 按钮。如果按下 Fonts 按钮，屏幕上就会出现 Fonts 对话框，如果按下 Background

按钮，屏幕上就会出现 Color 对话框。按下按钮，屏幕上就会出现用户定制的 HouseAd 类窗体。

5. 将本次实验的工作与本章中的示例 Chapter03\Sol3-2\HouseForSale.sln 比较一下，看看有什么不同。

下一步

到目前为止，我们已经介绍了如何应用窗体、空间、标准方法以及标准对话框创建一个正规的基于 Windows 的应用程序。在以后的学习中，我们将进一步深入地学习 Java 语言，深入讨论有关 WFC 控件的知识。

第四章　菜单、类型和方法

在上一章的内容中，我们学习了 Java 的类和对象。类为一个 Java 程序提供了基本框架。在这个框架中，可以放置方法和成员变量。在本章中，将介绍有关类中的方法和成员变量的详细知识。同时，通过本章的学习，可以提高使用基于 Windows 的应用程序的能力，另外,我们还将学习 Visual J++的 Menu Designer。

在本章中，将介绍的具体内容如下：

- Java 的内置数据类型和预定义类
- 布尔逻辑
- Menu Designer，如何将菜单加入到窗体中
- 方法

在本章中，我们将具体实践以下内容：

- Java 的内置类型
- Visual J++的 Menu Designer
- Class Builder

内置类型

对于 Java 类以及特征变量，Java 语言有相当多的基本类型。这些内置的类型比用户设定的 Java 类往往要简单，而且在处理简单的问题时，又十分方便有效。

注意：程序员通常都是使用类型声明（比如 int，float 和 byte）来设定一个基本类型或 Java 类。例如，说一个参数可以是任何类型，也就等于说，该参数可以属于任何类型或者任何 Java 类。为了避免一开始便混淆，我们建议应该按照如下方法使用这些术语：类型用于设定所有的内置变量和类；内置类型用于声明预定义类；类用于声明所有由 java，Visual J++，或者程序员提供的类。

表 4-1 列出了所有的基本类型及其基本特征。

表 4-1　Java 的基本类型

类型	值	初始值
Boolean	True 和 faulse	False
Byte	8 位整数值，在-128 和 127 之间	0
Char	16 位 Unicode 字符，在\u0000 和 \uFFFF 之间	\u0000
Double	64 位浮点数	+0.0

续表

Float	32 位浮点数	+0.0
Int	32 位整数	0
Long	64 位整数	0
Short	16 位整数	0

从表 4-1 当中可以看出，大部分类型是 Java 从它的祖先 C 和 C++语言那里继承下来的，除此之外，Java 在这一方面也有自己的独到之处。在这部分的学习当中，我们将简单地讨论一下 Java 语言的内置类型。

Boolean 类型

Boolean 是逻辑体系的一种。使用关键字 boolean，可以定义自己的 Boolean 值。同其他任何一种类型一样，Boolean 类型也有自己的值，以及对这些值进行一系列操作的规则。下面首先讨论这些逻辑值。

Boolean 值

在 Boolean 体系中，逻辑变量的值只有两种可能：真（true）或者假（false）。在 Java 程序中，在创建了 Boolean 变量的同时，也就等于指明了该变量只能有两种可能的值：true 或者 false。

注意： C/C++语言在这方面做得很小心：其逻辑值并不是 1 和 0，因此，也不能对其像上面的逻辑体系一样处理。

定义一个 Boolean 变量的方法如下：

```
boolean tooLarge;
```

可以使用任何一种运算符对一个逻辑变量赋值，但是一定要注意，这个值只能是真或者假。等价运算符（= =）正是这种合适的运算符，要注意，等价运算符是有两个等于号的，要时刻记着，不要混淆了等价符号和赋值符号（=）。假定我们创建变量来存储两个整型值。其代码内容如下：

```
int upperLimit = 100;
int count;
```

要求随着程序的运行，变量 count 的值周期性递增。在程序运行中，如果要检验变量 count 的值是否超过其最大的允许值 upperLimit，就可以用下面代码所示的方式来比较其大小：

```
count= =upperLimit
```

既然变量 count 与 upperLimit 的比较结果只会有两种可能，因此，这个表达式的值也只会有两个可能的值：真或者假。如果我们想要将该表达式的值赋给一个逻辑变量 tooLarge，其代码内容如下：

```
tooLarge=count= =upperLimit;
```

为了便于阅读和理解，可以在代码行中适当加上括号，如下所示：

```
tooLarge=（count= =upperLimit;）
```

如果真的能够给我们的阅读和理解带来方便，那就应该毫不犹豫地使用括号，大可不必吝啬。

另外，对于 Boolean 变量还应该补充说明一下：在实际应用中，很少直接对逻辑变量进行赋值。程序员往往会只是在代码中直接使用一个逻辑表达式。例如：

```
count= =upperLimit
```

而并不将该表达式的值赋给某个逻辑变量。尽管如此，时刻注意理解 Boolean 表达式的内容及其在代码中的作用，对一个程序员来说是十分重要的。

Boolean 表达式往往用在 if 语句中、while 循环或者 do while 循环中。下面是两个实例的源代码：

```
// if statement
if (count == 100)
{
// code executes only if count is 100
}
// while loop
while (size == 0)
{
// code executes repeatedly as long
// as size is zero
}
```

注意，在代码中，Boolean 表达式通常都是括在括号当中的。

建议现在阅读附录，总结一下 Java 语言语法的规则。

布尔运算

除了等价运算符之外，布尔表达式还支持大于（>），小于（<），大于等于（>=）和小于等于（<=）运算符，另外，还有与（&&）或（||）非（!）三个

逻辑运算符。

为了声明逻辑与的运算规则，下面举出一个 if 语句的示例，该语句用于判断变量 score 是否处在 80 和 90 之间。其代码内容如下：

```
if ( (score >=80) && (score <= 90) )
{
// code executes only if score is 80 or greater
// and also 90 or less
}
```

逻辑非的符号是一个惊叹号（!）。它的运算是将其后的逻辑表达式的值取反。也就是说，如果表达式本身的值是真，那么，经过逻辑非运算之后，值就应为假，反之亦然。下面就是一个使用了逻辑非的 while 循环：

```
while (!done) // read this as ″ while not done″
{
// code executes repeatedly until done is true
}
```

另外，逻辑非运算符还可以和等号在一起使用，其意义为不等于，下面是一个实例的代码内容：

```
if (score !=100)  // read this as ″ if score is not 100″
{
// code executes if score is anything other than 100
}
```

数值类型

数值类型用来存储数字的值。在计算机程序中，数字的值通常有两种类型：整型值和浮点型。整型数就只有整数部分，而浮点型则包括了小数点后的小数部分。例如，432 是一个整型数，而 432.2 就是一个浮点数。

Java 中用以声明整型的方法有：int，byte，long 和 short；用以声明浮点型的方法有：float 和 double 两种。

字符类型

Java 语言使用自己的统一字符编码系统。与 ASCII（使用 7 位字符）和 ISO（使用 8 位字符）的编码系统不同，Java 语言的编码系统采用的是 16 位字符。这就使得可以有多达 65,000 种之多的字符。Java 程序员可以使用多种语言中的字符，如：希腊语、阿拉伯语、日语的平假名，甚至是梵文字母。想要在程序中存储这些字符，我们需要两种类型的变量：char 和 java.lang.String。

单字符值类型：char

对于统一编码单字符类型，可以采用内置类型 char。

注意：统一编码中包括了许多各国语言的特殊字母。在本书中，如果使用术语字符（character）时，其意思是指 ASCII 字符系统，也就是只包括字母 a 到 z，及 A 到 Z，数字 0 到 9 以及标点符号。

因为有时单字符型 char 变量在适用性上有一定的限制，因此，在字符处理过程中，经常会用到预定义类字符串 String。

多个字符组成的字符串：java.lang.String 类

为了同时存储和处理多个标准编码的字符，我们就得使用预定义类声明 java.lang.String。因为所有 java.lang 软件包中类的前面都需要有 java.lang 类来适用源代码，所以，许多人将该类简单地指为字符串 String。下面就是一个示例，用以声明一个字符串 String 变量：

```
String name;
Name=”Bob White”;
```

文本属性（其他许多 WFC 控件的属性）类型为字符串 String，这样 getText 方法的返回值的类型就是字符串 String。其代码内容如下：

```
// event procedure for an EditBox object
private void edit1_textChanged(Object source, Event e)
{
String name;
Name = edit1.getText();
}
```

上述方法创建了一个名为 name 的变量，并且在控件 edit1 每次得到修改事件消息以后，可以将其文本属性赋值给该变量。类似地，一个字符串 String 在 setText 方法中相当于一个参数。如果想要在上述的示例上更进一步修改一个名为 label1 的标签 Label 控件的文本属性，则应该调用 label1，其代码内容如下：

```
private void edit1_textChanged(Object source,Event e)
{
```

```
String name;
Name = edit1.getText();
Label1.setText(name);
 }
```

事实上，我们可以将从 getText 方法返回的值直接传送到 setText 方法，而废除中间的字符串 String 变量。其代码内容如下：

```
 private void edit1_textChanged(Object source, Event e)
 {
label1.setText(edit1.getText());
 }
```

采用何种方案，完全取决与用户的喜好，或者特殊的环境要求。用户可以任意选择合适的方案。在比较两个字符串时，要特别小心。因为在比较字符串值（即为字符串的内容）与字符串引用时，有一个微妙但是又非常关键的差别。为了声明这一点，下面举出了示例：

```
 String firstName = ″ Bob″ ;
 Edit1.setText(″ Bob″ );
 FirstName == edit1.getText()      // is false!
 FirstName.equals(edit1.getText())     // is true
 Edit1.getText().equals(firstName)     // is also true
```

注意看代码的最后一行。有一个对象 edit1，其后是方法 getText，然后是另一个方法 equals。此处是可以由第一个方法使用返回值所调的方法。方法 getText 的返回值类型为字符串 String。该字符串的值可以用来调用 equals 方法。

在比较两个字符串的值时，要确保使用 equals 方法，而不要使用等号运算符。若要详细了解有关比较值与引用的差别，请参阅第三章的内容。

若想查看 Java 语言基本类型中的典型变量，可以打开名为 BuiltIn 的 Visual J++项目文件，该项目文件所在的路径为 Chapter04\BuiltIn\BuiltIN.sln，运行该项目文件，然后就可以查看有关的代码内容。

成员变量修改符

Java 语言为用户提供了大量的成员变量修改符，使用这些修改符，可以建设和修改变量的作用域和可视性。在本节中，我们将介绍有关类对成员变量的一些操作。

访问修改符

自然地，我们应该首先学习如何访问成员变量的修改符。访问修改符控件方法可以用来修改定义了的成员变量。常用的四个访问修改符为：缺省 Default（软件包）、公共 public、保护 Protected 和个人 private。表 4-2 中列出了以上四种方案的简短声明。

表 4-2 Java 的访问修改符

修改符	声明
缺省(Default)（软件包）	缺省访问的成员变量在同一个程序模块中可以被任意数量的函数所调用
公共(public)	公共 public 修改符访问方式，成员变量可以被 Visual J++ 项目文件的任何方法所调用；访问是不受限制的。为了避免对类作所不希望的改动，我们推荐尽量不要使用这种方式访问成员变量
保护（protected）	该种方式，成员变量可以在同一类层次的方法当中自由调用
个人（private）	该种方式，成员变量只能在同一个 Java 类当中调用。这种方式通常是最为安全的

注意 public、Protected 和 private 都是关键字，而 Default（软件包）是指当前并没有指定访问修改符。

在第五章讲述 Java 语言的层次结构时，将会进一步介绍有关 protected 关键字的内容，对于缺省模块，将在第十章讲述 Java 语言的模块时作进一步的讨论。综上所述，我们可以这样保守地认定，所有的成员变量都应该采用 private 访问修改符，这样是最为安全的。

静态成员变量

迄今为止，我们所讨论的成员变量都是非静态的变量。也就是说，当我们

每次为一个类创建了对象时，该对象都是对应类的一个副本，而具有自己的一套类的成员变量。现在，我们来比较一下非静态与静态成员变量的行为。首先，在 Visual J++ 中打开 Gizmo 项目文件（该项目文件位于 Chapter04\Gizmo\Gizmo.sln 路径中）。运行该项目文件，并按下 Make A Gizmo 按钮，创建一些小物件。这些被创建的小物件会显示在窗体的底部。同时。窗体顶部的文本反映了当前正在使用的物件的状态消息。但是，如果此时我们用同样的方法再创建一个物件，就会发现，虽然在窗体的底部会出现一个物件，但是，在窗体顶部的文本中，仍然显示只有一个物件在使用。这究竟是什么原因呢，为了弄清这一点，让我们看看 Gizmo 类的代码。以下是 Gizmo.java 类文件的一部分：

```
public class Gizmo
{
private String manufacturer = ″ Gizmo Products, Inc. ″ ;
private int gizmosInService = 0;
private int length;
private int width;
private int depth;
public Gizmo (int len, int wid, int dep)
{
    gizmosInService = gizmosInService + 1;
    // more code ...
}
```

```
// more methods ...
}
```

在该类的一开始，定义了五个成员变量。每个成员变量都示例了 Gizmo 类的对象，同时取得了各自的副本。每次创建一个新的物件，成员变量 gizmosInService 也同时被创建，并被赋予初值 0。与此同时，Gizmo 构造器方法也被系统调用，该方法因为新的对象的创建，而将变量 gizmosInService 的值加 1。也就是说。变量 gizmosInService 的值一直都是 1。这些成员变量也被称为实例变量 instance variables，这是因为，对于每个创建的实例（也就是对象），这些变量都有一个副本与之对应。然而，我们在应用过程中，真正需要的是类变量。类变量是一种带有修改符的特殊成员变量，它不专门属于类中的某个特殊的对象，而是属于整个类。另外,任何静态成员变量只有一个副本。因此，我们可以采用将 static 关键字加入到成员变量的定义中的办法，将变量 gizmosInService 改为一个静态成员变量。其代码内容如下：

```
private static int gizmosInService = 0;
```

现在，可以再运行一次这个程序，就会发现，在创建新的物件时，窗体的上方显示了正确的物件数目。图 4-1 示意了 Gizmo 类的一个静态成员变量 gizmosInService。

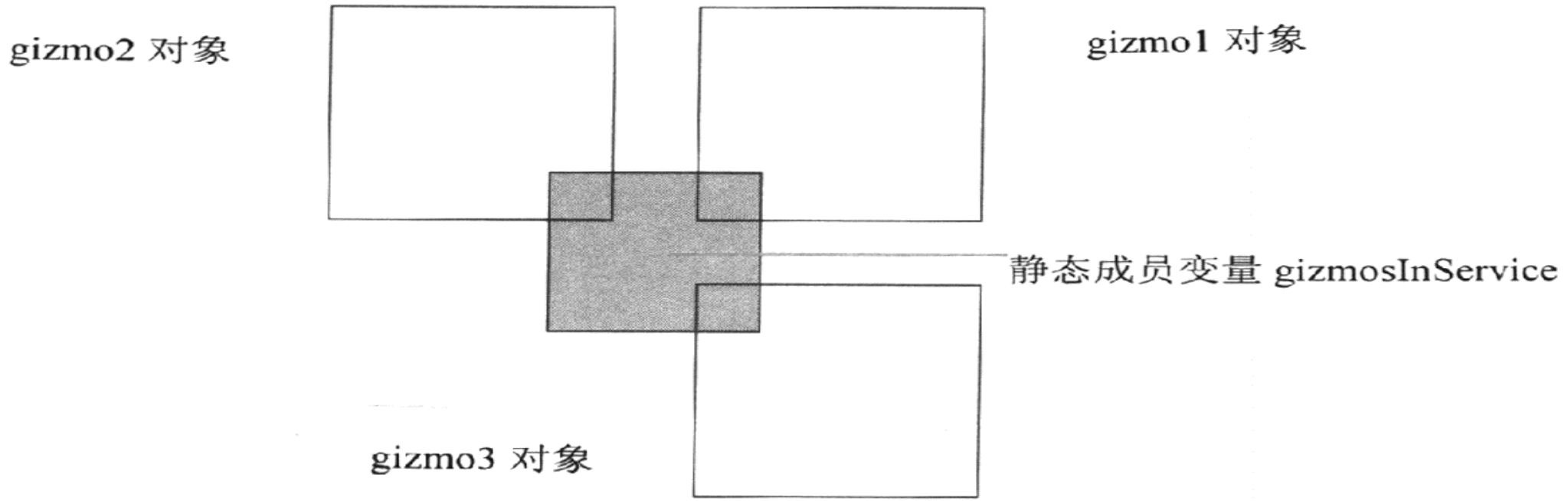


图 4-1　一个类中的静态成员变量属于该类中所有的对象

通过上面的介绍，我们很容易得出以下结论：不能在构造器中设定一个静态成员变量的值，因为一旦我们这样做了，每次创建一个新的对象时，成员变量就都会重置为一个同样的值。但是，我们可以在构造器中修改一个静态成员变量的值，比如像上文当中介绍的那样，将变量 gizmosInService 的值加 1。

在 Gizmo 类当中还包括了这样一个消息，就是物件生产者的名称，这个名称消息不论类中有多少个对象，都不会改变。除了将成员变量设定为静态变量之外，还可以将其设定为最终 final 成员变量，在下一节中，我们将介绍有关 final 关键字的内容。

最终成员变量

所谓最终成员变量，其最终是指无论一开始创建该变量时将其设定为什么值，在以后程序的运行过程当中，变量的值将一直保持这个值不变。在其他计算机语言中，往往将这种最终变量称为常数。下面，让我们回顾一下前面的GizmoFactory的示例，看下列代码：

```
public class Gizmo
{
private String manufacturer = ″ Gizmo Products, Inc. ″ ;
private static int gizmosInService = 0;   // class variable
private intg length;
private int width;
private int depth;

        // code omitted for brevity

public String getManufacturer ()
{
    return manufacturer;
}
}
```

成员变量 manufacturer 是一个十分有用的只读成员变量。我们不希望该变量的值发生变化，为了确保这一点，我们定义了最终成员变量。通常，一个最终变量又会成为静态成员变量，因为为一个常数制作副本是没有意义的。将一

个最终变量同时定义为静态成员变量，可以采用如下代码所示的方法：

```
Private static final String manufacturer=Gizmo Products , Inc. ″ ;
```

注意，任何尝试改变一个最终成员变量值的结果，都会导致编译错误。与其使用一个精确的数字，不如使用静态最终变量赋予数字一个有意义的名称，这样可以增加程序代码的可读性，同时也便于修改。在第二章的学习中，我们在讨论改变一个窗体的背景颜色时曾经用过如下所示的代码：

```
setBackColor（Color.Blue）；// sets the form background color to blue
```

在 Color 类当中，变量 BLUE 实际上就被定义为公共静态最终整型变量。前面我们讨论过，将变量设定为 private 类型是最为安全的，大多情况下，变量都被设定为 private，此处正是例外，需要将变量设定为公共型才有意义。对于我们讨论的关于物件的示例，可以在 Gizmo 类的代码当中加入如下的代码行：

```
public static final int awake = 0;
public static final int asleep = 1;
private int status = asleep;
public void setStatus (int stat)
{
status = stat;
}
public int getStatus ()
{
return status;
```

}

我们所创建的物件可以处于唤醒和睡眠两种状态。注意，物件在创建时是处于睡眠状态的，但是，我们可以使用 setStatus 方法来改变物件的状态。另外,getStatus 方法可以告诉我们物件当前所处的状态。另外,还应该注意成员变量被定义为静态最终整型，同时也是公共类型，所以在该类以外也可以使用。

菜单操作

制作一个用户交互界面，其中一个最重要的任务就是设计菜单。这也正是大家喜欢 Visual J++的 Menu Designer 的原因所在。Visual J++使得菜单设计变得极其简洁，是因为它提供了一个叫做 Menu Designer 的 WYSIWYG（其大意为所见即所得）工具。在下文将有详细的介绍。

使用 Menu Designer

将一个菜单放置到一个窗体中，与将其他的控件放置到其中一样简单，或者放置其他控件到窗体当中，会更加简单一些，因为对于其他控件，将其放置到窗体中的位置并不重要，而菜单往往都要在窗体的顶部。使用 Menu Designer 使得一切都变得十分简单，简直无法想像如果没有它会怎样。

下面，让我们来讨论一个实际的示例。首先启动 Visual J++系统，如果实现已经正在使用 Visual J++，则此时要将以前的消息保存一下，然后在 File 菜单中选择 New Project 项创建一个新的（空的）项目文件。将项目文件命名为

MenuTest，或者是其他的有一定意义的名字。一旦项目文件创建完毕，就可以在 Project 菜单选择 Add Form 项。为了使事情简单一些，可以将窗体文件和项目文件命以相同的名称（比如：menuTest.java）。

我们可以使用工具箱窗口来向窗体中加入菜单。如果此时屏幕上并没有显示出工具箱，在 View 菜单上选 Toolbox 项即可。在工具箱窗口中按下 WFC 控件按钮，然后选中 MainMenu 控件，并将其拖到窗体当中。一个可视的标记“Type Here”就会出现在窗体的顶部。如果不喜欢这种用鼠标拖动来放置控件的方法，也可以用鼠标双击工具箱中的控件，Visual J++会自动将选中的控件放置到窗体当中去。让 Visual J++系统自动放置菜单控件是很好的办法。这样，在窗体中放置菜单的位置对程序员来说就不成问题了。到目前为止，下面要面临的主要问题就是，如何输入菜单的标题，以及如何确定菜单的各个选项的内容了。可视的标记会直接显示在所需输入文本位置的右侧，在缺省位置的文本框中输入第一个菜单的标题即可（例如：输入 File 就加入了 File 菜单）。这样菜单就加入了标题，而缺省文本框的位置也移动到了下一个标题所在的位置，继续输入菜单的标题，直到完成所有的菜单标题的输入为止（图 4-2 示意了 Menu Designer 的工作情况）。

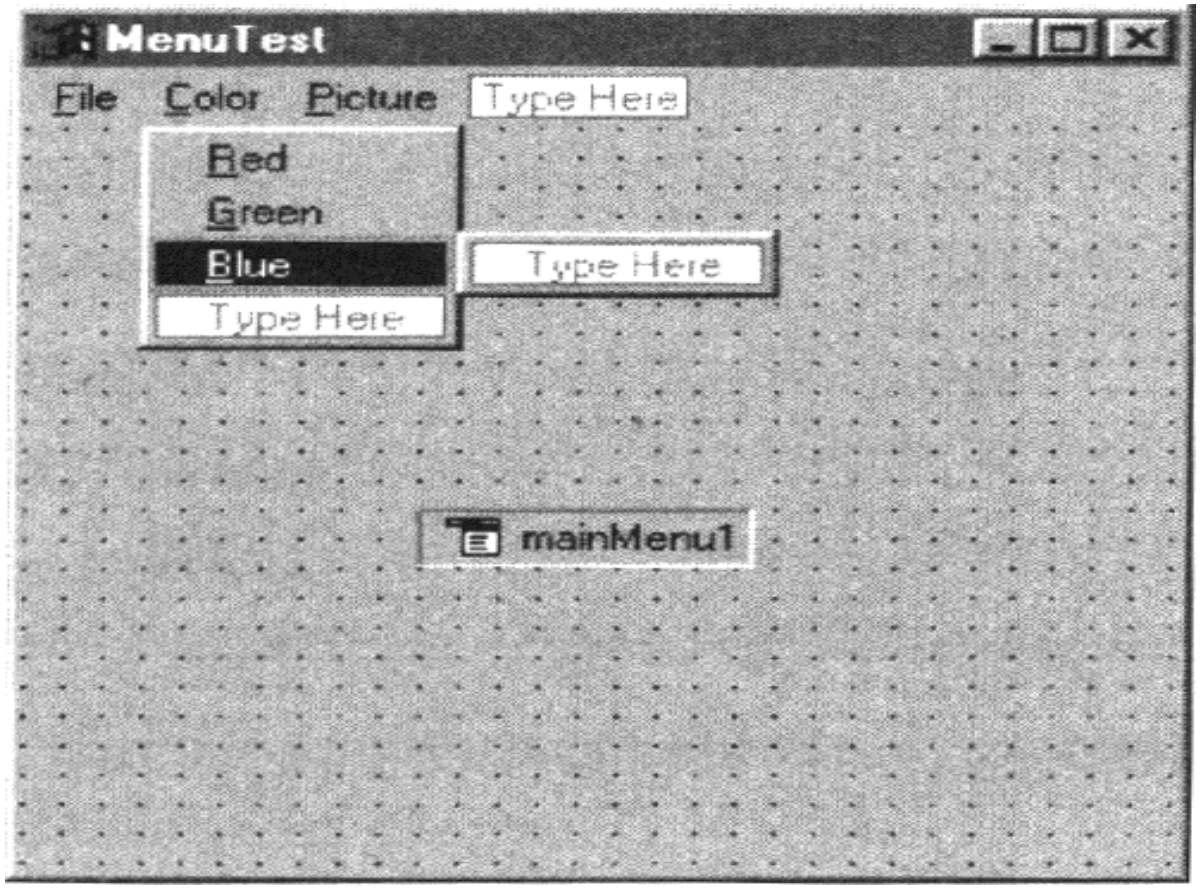


图 4-2　使用 Menu Designer：窗体中的 MainMenu 控件

同前面介绍过的其他 WFC 控件相类似，菜单项也具有自身的属性。一个窗体可以有不同的菜单项，如 File/Save 或者 File/SaveAs，这些变化主要取决于应用程序所处的状态。使用每个菜单项的属性，各个菜单项可以被设定为有效的或无效的，复选的或非复选的，可见的或隐藏的。另外，还可以在 Properties 窗口中改变一个菜单项的文本属性，不过，使用 Menu Designer 改变文本的属性更为容易。

菜单中的文本属性的一个特殊特征就是，可以建立一个访问快捷键。访问快捷键就是 Alt 键和一个字母键，使用这样的快捷键，可以不通过鼠标访问菜单中的各项。如果在菜单项中在某个字母前面放置了&符号，则在菜单项当中，显示的不是加入的&，而是在&符号后的字母上加上下划线。比如说，在设计菜单时，在菜单项中输入的是&File，而屏幕上菜单项中显示的将是

File

而不是&File。这就意味着，用户可以不使用鼠标，而通过在按下 Alt 键的同时按下 F 键而激活 File 菜单。

在本次实例当中，我们希望设计出如图 4-3 所示的菜单。注意，菜单的标题就是菜单栏上的名称，而菜单项则是选中某一菜单标题后下面出现的一系列的选项。

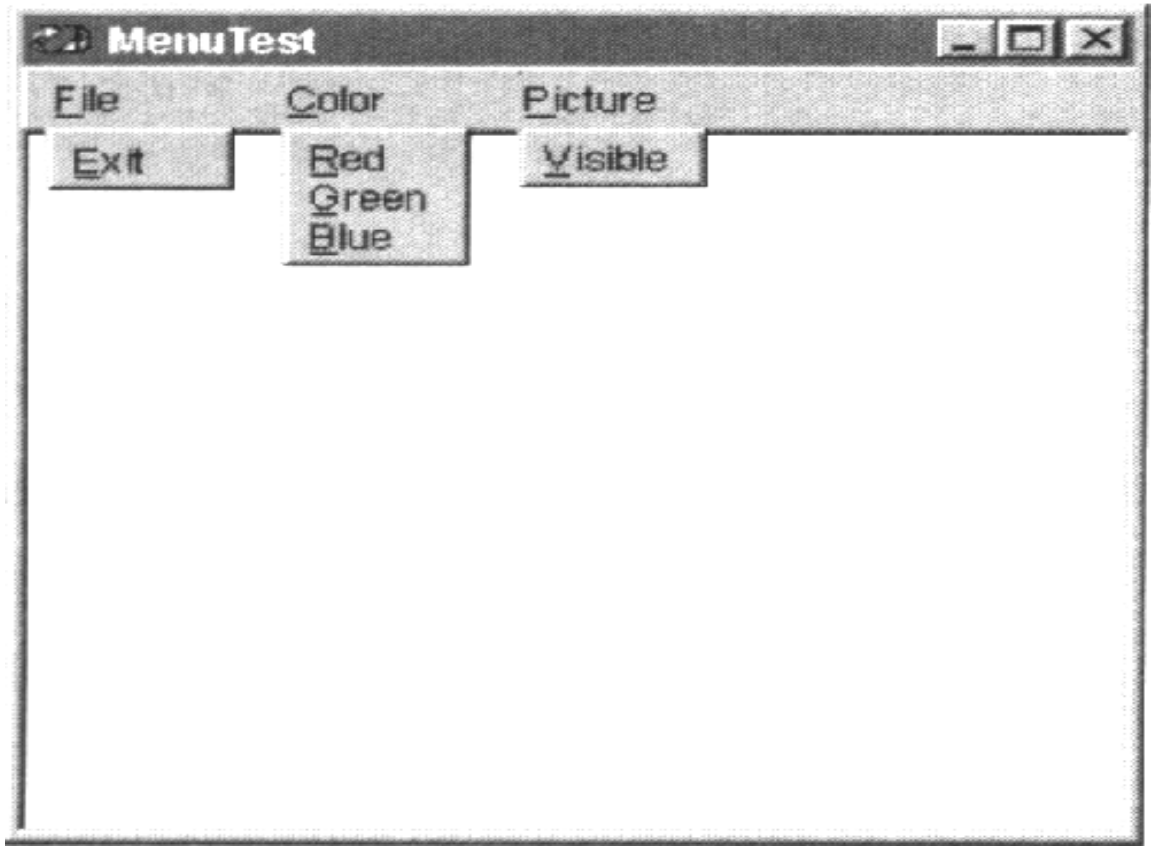


图 4-3　Menu Designer 项目文件中实例中的菜单标题和菜单项

要点：在仍然处于 View Designer 阶段时，不要双击菜单的任何元素。另外,要特别注意，在编写任何事件处理程序之前，要确保每个菜单项都已经正确输入，并检查无误。一个应用程序可以有多个菜单项，如果使用缺省的项名 menuItem1，menuItem2 等，就会显得十分笨拙。

菜单项的名称

每当加入一个菜单标题或者一个菜单项，就等于为窗体创建了一个新的控件。对于这些控件，其缺省名称就是在 menuItem 后面加上一个在窗体当中创建次序号的数码（例如：在窗体当中，第一个创建的控件其名称就为 menuItem1，第二个创建的控件其名称就为 menuItem2）。当然，我们也可以通过改变它们的名称属性，从而为这些控件取一些更有意义的名字。

因此，在我们讨论编写事件处理程序之前，将菜单标题和菜单项重新命名一下。对于大多数程序员来说，将菜单项从 menuItem1 命名到 menuItem100，显然不是最好的命名方式，这样的命名，用户在使用时可能往往会迷惑地问道：来，让我来看看 menuItem57 究竟是 Delete，还是 Rename?一百个菜单项听起来是一个相当大的数目，但是，将它们组织成十个标题，每个标题下都有十项的话，问题就变得简单多了。按照表 4-3 所示，设定每个菜单项的名称属性：

表 4-3　我们的菜单标题和菜单项的文本属性值及对应的名称属性值

菜单项属性值	菜单名属性值
&File	*fileMnu*
E&xit	fileExitMnu
&Color	colorMnu

续表

Red	ColorRedMnu
Green	ColorGreenMnu
Blue	ColorBlueMnu
Picture	PictureMnu
Visible	PictureVisibleMnu

在这个实例当中，我们引入了一种命名规则，就是在菜单标题名称的后面加上一个 Mnu 后缀作为菜单项的名称。

如果想要在 Properties 窗口中修改一个名为 pictureVisibleMnu 菜单项的名称属性，可将复选属性改为 true，此时，在菜单项的旁边就会出现一个复选标记。就像我们操作其他 WFC 控件时所看到的一样，控件的属性值可以在设计的过程中或者在代码中改动（在本次示例中，使用的是 setChecked 方法）；实际上，我们将在以后的实验习题中进一步实践。

除此之外，使用 Menu Designer，我们还可以创建层次式的菜单。也就是说，可以在任何一个菜单项下建立子菜单。使用 Menu Designer 建立子菜单的方法也很简单，只需在一个菜单项区域输入菜单项名称，注意，此时在此右侧会出现一个 Type Here 框体，这个框体就代表该子菜单的第一项。如果此处不需要子菜单，只要在此处不输入任何东西就可以了。为了简单起见，在本示例当中，

我们不建立子菜单。

菜单事件和事件处理程序

菜单可以由单击鼠标键（或者按下访问快捷键）来激活，Click 是该事件的名称，下面，我们要为该事件编写事件处理程序。无论何时，当用户单击菜单的某一项，菜单就会接收到一个 Click 事件。

当完成了菜单的设计（使用 Menu Designer），并且为菜单的各项取了有一定意义的名称以后，就该为菜单的每项加上事件处理程序了。在设计窗口 View Designer 中双击菜单项，可以创建一个事件处理程序在代码窗口中的外壳。在这个代码窗口中，就可以输入事件处理程序的主体代码。

注意，在 File 菜单中选择 View Designer 选项就可以打开设计窗口。在此，我们可以为菜单选项添加事件处理程序，双击菜单项 Exit。当双击一个菜单项时，打开了一个代码窗口，该代码窗口对应于包含了菜单控件的窗体。这时，在代码窗口中会出现一个事件处理程序的框架。下面是 fileExitMnu 控件的事件处理程序代码：

```
private void fileExitMnu_click(Object source,Event e)
{
Application.exit(); // ends the application
}
```

exit 方法会以一种适宜的方式终止应用程序。下面运行项目文件，屏幕上就会出现一个窗体，此时就可以对菜单栏进行各种测试。当然，因为此时这些菜单项中并不对应任何事件处理程序，所以，当菜单项被选中时，系统不会作

出任何反应。

除此之外，使用 Shortcut 属性，也可以为控件添加菜单快捷键。菜单快捷键也就是一些等价于从菜单中选取菜单项的控制键或功能键。为了把一个快捷键与一个菜单项联系起来，只需在菜单项的 Properties 窗口中单击其快捷键属性，然后按下快捷键属性菜单旁边的向下箭头。除此之外，也可以使用 Menu Designer 实现上述操作。如果此时单击菜单项的右边，就会弹出一个菜单，在此菜单中，列出了所有可能的快捷键。针对我们所举的示例，我们来修改一下 menuItem 对象，为其设置如表 4-4 所示的快捷键。

表 4-4　菜单项和对应的快捷键

菜单项	快捷键
colorRedMnu	Ctrl+R
colorGreenMnu	Ctrl+G
colorBlueMnu	Ctrl+B

当为菜单项添加了快捷键以后，再回到 Menu Designer，就会看到在菜单中各个菜单项的旁边出现了对应的快捷键。

实验 4-1 为 Hello 应用程序设计菜单

到目前为止，我们在组建应用程序时学习了菜单的设计。本次实验就为窗体设计菜单做一些练习，以求熟练掌握这些技巧。

实验概述

在本次实验中，我们将练习以下内容：

- Menu Designer 的使用技巧
- 菜单事件处理程序

本次练习的目的在于组建一个包含有菜单的应用程序（与第二章中我们创建的 Hello 应用程序十分类似）。如果你一直都在跟着本书的示例做，就可以跳过实验安装的内容，直接进入实验指导部分。

实验准备

1. 启动 Visual J++；
2. 打开 MnuHello 启动项目文件（Chapter04\Lab4-1\MnuHello.sln）。

实验步骤

1. 使用 Menu Designer，在 Color 菜单中双击各个菜单项（而不是 Color 菜单的标题），并添加适当的事件处理程序。与 Red 控件相对应的事件处理程序的代码内容如下：

```
private void colorRedMnu_click(Object source,Event e)
{
setBackColor(Color.RED);
}
```

对应 Green 和 Blue 的事件处理程序与 Red 的事件处理程序的内

容类似。

2. Picture 菜单中 Visible 菜单项的事件处理程序应该包括切换复选标记的代码内容。修改复选属性，以实现复选标记的开启和关闭。首先，我们将 Properties 窗口复选框中 pectureVisibleMnu 项选中（将其 checked 值设为真）。在事件处理程序中，使用 setChecked 方法再将其值取反。setChecked 将返回复选属性的当前值，同时使用本地非（NOT）运算符对其值取反（如果返回值为假，则产生一个真值，反之亦然）。如果选择了 Visible 项，那么，在窗体中就会看到 forestPic 图片。如果此时已经完成了复选选择，就需要将复选标记去掉，使得图片消失以后，才能在菜单选择 Visible 项，因为只有这样，才能实现有不可视到可视的切换。
3. 在创建了显示和消失图片的复选标记以后，还可以使用 forestPic 图片的 setVisible 方法实现图片由可视到不可视的切换。
4. 输入 OK 按钮的事件处理程序代码。其代码内容如下：

```
        private void OkBtn_click(Object source, Event e)
 {
if (nameEdt.getText().length() !=0)
   {
   nameLb1.setText(″ Hello″  + nameEdt.getText());
   }
        }
```

5. 运行项目文件，并检验菜单，该项目文件的解决方法文件在 Chapter04\Lab4-1\MnuHello.sln 路径中。

下一步

在前面的学习中，我们已经看到了 Java 方法的实例。在以后的内容中，我们将进一步详细地介绍有关 Java 方法的内容。这可以使我们更加深入地开发和利用 Java 语言的潜力，如重载能力和可继承性。

方法

方法（在一些其他编程语言中又称为函数、子程序、过程或者子例程等）是 Java 语言为程序员提供的服务于 Java 类，或者类的对象的工具。对于 Java 程序员来说，许多方法一开始就可以拿来使用，另外，程序员还可以自己编写一些方法，加入到 Java 语言的方法列表当中。

方法的名称中一般都要包含一个动词（例如 set 和 get 等），以表示进行了某个动作。尽管方法的名称并不要求必须用动词，但是，这样的命名方式无疑会提高程序的可读性，大大方便了用户。在使用的过程中不难发现，用动词为方法命名是十分科学的，因为从某种意义上讲，方法就是 Java 语言中的动词。

关于 Java 方法的另一个有趣的地方是，与其他编程语言的相关特征有所不同，Java 方法必须是类的一个成员。在几乎所有的其他编程语言当中，子程序并不与某个特定的对象（如果那个语言当中有对象的话）密切相关，而是定义

在全局范围中。Java 语言程序的组织的严谨性也在于它要求每个方法、每个代码段都与特定的 Java 类紧密联系。参考前面讲述的 Gizmo 实例的一段代码，你对此会有更加深入的了解（为了简单起见，下面的代码段中没有列出方法所对应的类的内容）：

```
public int getInServiceCount()
{
return gizmosInService;
}
```

应该承认，区别方法名称前面的其他关键字而识别出一个方法名是有一定的难度的。幸运的是，在 Visual J++中，这些关键字（在本示例中为 public 和 int）的颜色与方法名的颜色是不同的。总之，方法名称的一个显著的标志就是它紧接着一个括号。

注意： Visual J++方法由 Microsoft 提供，它遵循这样的命名规则：在属性名称前加上诸如 set 或 get 的动词。例如：Label 类具有一个文本属性，同时它也就有一个 getText 方法和一个 setText 方法。

参数

为了使得方法可以有效地工作，多数情况下首先需要一些信息。方法是以参数的方式来获取信息的。通知编译器一个方法中含有参数的方法是，在参数名称的前面加上参数名称类型的定义。如果有时一个方法中用到不止一个参数，就应该在第一个参数的后面加上逗号，然后，再接第二个参数名称类型定义和第二个参数的名称。这样，一个方法就可以使用任意多个参数了。在 Gizmo 类

中的构造器方法引用了三个参数，这三个参数的类型均为整型。其代码内容如下：

```
public Gizmo (int len, int wid, int dep)
{
gizmosInService = gizmosInService + 1;
length = len;
width = wid;
depth = dep;
}
```

Gizmo 构造器所引用的参数的名称为 len，wid 和 dep。当然，根据程序的需要，参数的名称和类型可以任意。但是，我们推荐为参数命名时，应该尽量使名称能够声明为什么该方法会引用该参数，以增加程序的可读性。在这个实例中，三个参数的名称分别是 length，width 和 depth 的缩写。那么，我们为什么又不干脆直接使用 length，width 和 depth 本身呢？我们不能使用全称是为了避免与对应的成员变量名相混淆。关于这一点，在本章中后面要介绍的 Java 如何处理问题，及 The this Keyword 当中有更为详细的阐述。如果一个方法不需要引用事先任何消息，不需要引用参数，那就可以将括号空出，但是不能省略。其代码内容如下：

```
Public int getInServiceCount()
{
Return gizmosInService;
}
```

注意：因为 Java 是以值的方式来为方法提供参数，所以预定义类型（比如整型 int）的参数的方法外部不能改变值的大小。在一个方法的内部，可以任意改变一个整型参数的大小；这些改变对外部没有任何影响。Java 语言的这一特征与其他编程语言（比如 Basic 语言）有着显著的不同。

返回类型

Java 类需要一些消息来引用参数，有时也会将一些消息返回到引用它们的代码。在 Java 语言中，是使用 return 关键字来实现这一功能的。从 getInServiceCount 方法的示例中，我们可以看到这一点。其代码内容如下：

```
Public int getInServiceCount()
{
rwreturn gizmosInService;
}
```

返回值是 gizmosInService 的当前值。注意，在 Java 中，任何一个值都有自己的类型，或者是基本类型，或者是与 Java 类对应。在这个示例中，类被定义为整型，也就是说，方法的返回值为整型，成员变量 gizmosInService 的类型为整型，这也与代码中的内容相一致。从上述代码段中，可以看出，一个特定的方法的返回类型是在方法名称前就作了定义的。另外,有些方法是没有返回值的。对于这些方法，将其定义为空类型 void，表明该方法没有任何返回值。在本示例中，Gizmo 类中没有不含返回值的方法，我们可以向其中加入一个不含有返回值的方法，来设定物件的深度。该方法没有返回值，所以其代码内容应如下所示：

```
    public void setDepth (int dep)
    {
  depth = dep;
    }
```

this 关键字

在一个 Java 程序中，所有的动作都由方法中的可执行代码实现的。方法一般总是以一定的顺序组织在一个类当中。在这一点，Java 与其他编程语言有着显著的不同，大多数其他编程语言的代码执行不需要特定的环境，也就是说，这些代码都是全局性的，而在 Java 语言中则没有全局性的方法。

注意：这里所讨论的 this 关键字不包括静态方法。从技术上讲，也只有非静态方法才有当前对象和 this 引用。在以后的内容中，我们将进一步讨论有关静态方法的内容。

当然，我们需要一个对象来激活一个对应的方法，这个对象就是所谓的当前对象。使用 Java 的 this 关键字，可以使得程序员能够指定当前对象。正是因为有了 this 关键字，一个方法才能始终清楚它是为哪个对象所调用（this 关键字的值实际上是个引用值，因此 this 关键字有时也被称为 this 引用)。当一个成员变量的名称与其作为参数时的名称相同时，使用 this 关键字就显得更加有效。的确，this 关键字是有些复杂，但是，一旦用户掌握并使用了它，就会发现假如 Java 语言中没有 this 关键字，那简直无法想像如何进行工作。

Gizmo 类中的构造器方法是一个声明使用 this 关键字的作用的示例。首先，我们可以适当修改参数的名称，以避免其与对应的成员变量相冲突（如参数 len

对应成员变量 length）。与此同时，我们也可以不牺牲参数和变量的一致性，而使用变量的原名作用参数名称，同时使用 this 引用来区别成员变量和参数。其代码内容如下：

```
Public Gizmo (int length, int width, int depth)
{
gizmosInService = gizmosInService + 1;
this.length = length;
this.width = width;
this.depth = depth;
}
```

在上述代码当中，成员变量名与参数名称具有相同的标识符，不过同时也使用了 this 引用，从而消除了不确定性。如果上述代码中不含有 this 引用，会出现什么样的后果呢？下面，我们就在上述代码中的赋值语句中去掉 this 声明，其代码为：

```
length = length;
width = width;
depth = depth;
```

这样改过之后，不仅带来了名称上的混淆，而且代码将无法正常运行。系统将把值赋给三个参数，而我们所要赋值的成员变量却没有任何变化。除此之外，有些时候，this 关键字也可能是不必要的（比如参数名称与成员变量名称是不同的情况），这时也大可以将其忽略。当然，只要觉得在方法或者数据前加上 this 关键字可以使条理更清晰，那就可以在代码中加入 this。

下面，让我们改变 Gizmo 类中的所有参数的名称，使其与对应的成员变量相匹配。然后，在适当的位置加入 this 关键字，以确保正确地为变量赋值。

在 Visual J++中，为某个类编写成员函数的时候，使用 this 关键字作为一个十分方便的方法，可以提醒程序员，类中所调用的有哪些方法。假定现在想要为一个窗体获取颜色属性，而又忘记了需要调用哪个方法。此时如果输入 this 关键字，并在其后加上点号（.），就会弹出一个菜单，上面列出了该类中当前对象可能会调用的所有方法。图 4-4 中列出了一个具体的实例，声明了 this 关键字的这种用途。

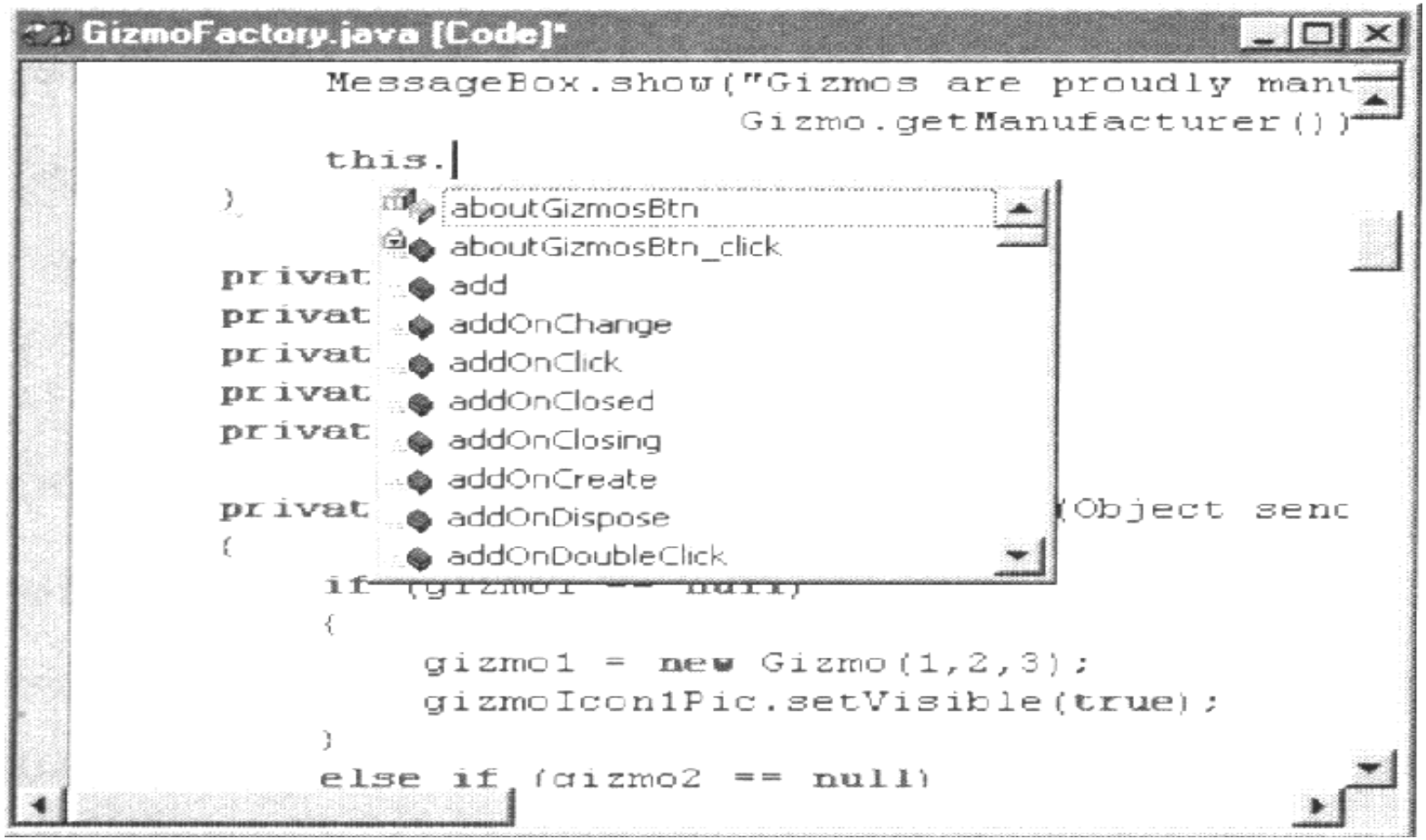


图 4-4　使用 this 关键字列出当前可以使用的方法

静态方法

在前面，我们讨论静态成员变量的时候，了解到静态成员变量不单单属于类中的个别对象，而是属于整个 Java 类。换句话说，非静态成员变量是实例变量，而静态变量则是类变量。这种思想同样也应用于 Java 方法。

在 Java 中，静态方法是类方法，它是唯一可以不必实例化一个对象，就可以被激活的方法。正是因为如此，类方法中不允许访问它们所在类中的任何实例变量。也是因为这个原因，类方法中不能使用 this 关键字。另一方面，类方法中可以使用类成员变量，而且并不依赖于类的存在。

实际上，在第三章中，我们调用 MessageBox 类的方法时，已经使用了类方法。MessageBox 是一个 Java 类，而不是一个对象，因此 show 必须是静态的方法。

下面回到 Gizmo 类的示例中，我们有两个含有类成员变量返回值的方法：getInServiceCount 方法和 getManufacturer 方法。既然这些方法要求即使是在类中没有相应的对象时也可以合理调用，那么，就需要将其设定为静态的方法。

我们可以简单地改动这两个方法的代码，适当改动之后，代码内容应如下所示：

```
public class Gizmo
{
private static final String manufacturer = ″ Gizmo Products, Inc. ″ ;
private static int gizmosInService = 0;

// code omitted for brevity

public static int getInServiceCount ()
{
    return gizmosInService;
}
```

```
public static String getManufacturer ()
{
    return manufacturer;
}
 }
```

到目前为止，我们已经将 getManufacturer 方法改为静态方法，下面就可以消除代码的一些错误。首先运行 GizmoFactory 项目文件。如果在创建一个物件之前按下了 About Gizmos 按钮，就会产生一个错误。错误的原因是引用 gizmo1 的值为空，这就意味着，它并没有引用任何对象。要想修改该错误，就要在 GizmoFactory.java 文件中找到 getManufacturer 的调用，并对其作适当改动，不再使用对象，而是使用 Gizmo 类。改动以后的代码内容如下：

```
private void aboutGizmosBtn_click(Object source, Event e)
{
MessageBox.show(″ Gizmos are proudly manufactured by ″  +
                    Gizmo.getManufacturer());
}
```

这些改动成功地消除了代码中的错误。除此之外，在改动时，我们也可以同时将 getInServiceCount 的调用也改为使用 Gizmo 类，改动后其代码内容如下：

```
gizmoInfoLb1.setText(″ There are currently ″  +
                                Gizmo.getInServiceCount() +
                                ″ gizmos in service″ );
```

再次运行项目文件，核实错误是否已经被排除。因为类方法的调用不依赖

于有没有实例化类的对象，所以，它们正是编写那些不依赖于对象的代码的理想位置，许多系统代码都是以这种方式编写的。在 GizmoFactory 类中（在其他类中情况类似）main 方法如下：

```
public static void main（String args[ ]）
```

main 方法是一个类方法。另外还应注意：在运行项目文件时，main 方法是 Visual J++最先调用并执行的方法，而在项目文件启动前，并没有任何对象产生。

重载方法

在通常情况下，我们在为某个类创建了一个方法以后，往往不止一次地使用这一方法。就像本书第三章中所讲述的，我们在创建构造器方法时，希望有一个无参数的构造器方法，同时，也希望有一个有参数的构造器方法。当在同一个类当中出现了两个同名的方法时，就称为重载。所有方法都可以选择重载属性。参看下列代码片断：

```
public class Television
{
public void setAspectRatio (int width, int height)
{
    // TODO: complete setAspectRatio
}
// other members as necessary ...
}
```

setAspectRatio 方法允许用户在电视屏幕上修改图标。一个规则的 TV 图标

的特征比例为 4 比 3，也就是说，每 4 个单位的宽度，就应有 3 个单位的高度与之对应。例如，一个 20 英寸的电视机屏幕为 16 英寸宽，12 英寸高，20 英寸是屏幕对角线的长度。在不久的将来，数字电视信号将会成为标准的电视信号。采用了新的信号标准，屏幕的特征比例就要改为电影片的屏幕比例（比如 7 比 3）。

因此，更改电视机屏幕的特征比例将是一个非常实用的功能：在欣赏如 My Mother the Car 这样的老节目时，可以用 4 比 3 的特征比例，而在观看新的影片时，就可以使用 7 比 3 的特征比例。我们将使用的 setAspectRatio 方法具有两个参数，分别用以存储屏幕的宽度和高度。这个方法是相对独立的，因为它会将屏幕设定为我们指定的比例，而不考虑当前屏幕的长度和宽度。在有些时候，为了使用双精度值相对地调整屏幕的特征比例，在调整屏幕的长度和宽度时，我们将使用一种特殊的方法来保存当前的屏幕尺寸消息。现在，就让我们加入使用浮点型值的第二种方法。其代码内容如下：

```
public class Television
{
public void setAspectRatio (int width, int height)
{
    // code omitted
}
public void setAspectRatio (double ratio)
{
    // code omitted
```

```
}
// other members as necessary ...
 }
```

现在，类当中就有了两个具有相同名称的方法。

重载解决方案

下面，我们将讨论 Java 是如何识别两个具有相同名称的方法的。有趣的是，Java 识别同名的两个方法与人类的辨别行为相类似。当我们调用方法时，自然会注意不同之处。每个方法都有一个参数列表，根据列表中的类型特征（包括在列表中的位置顺序），我们可以找到任何一个方法区别于其他方法的属性。使用上面讲述的调整电视机屏幕比例的示例，我们可以很容易看出两个重载方法具有不同的参数列表。

第一个方法的参数列表是两个整型的参数，而第二个方法则由两个双精度型参数构成。这样，在调用这两个方法时，就不会产生混淆了。方法调用的代码内容如下：

```
 public class AspectViewer extends Form
 {
private Television tv;
public AspectViewer()
{
      // Required for Visual J++ Forms Designer Support
      initForm();
```

```
            // TODO: Add any constructor code after initForm call
            tv = new Television(viewScreenPic);
    }
    private void tvBtn_click (Object source, Event e)
    {
        tv. SetAspectRatio (100， 75);
    }
    private void movieBtn_click (Object source, Event e)
    {
        tv. setAspectRatio(7.0/3.0);
    }
    // AspectViewer class continues ...
     }
```

只要各个方法都有着不同的参数列表，那么，在一个类当中，可以有任意多个同名的方法。另外，方法重载还可以为我们提供方法选择的缺省方案。在下面的 Television 类当中，我们可以选择整型参数方案、双精度型参数方案或者是缺省的无参数方案。其代码内容如下：

```
 public class Television
 {
public void setAspectRatio (int width, int height)
{
```

```
        // code omitted
    }
    public void setAspectRato (double ratio)
    {
        // code omitted
    }
    public void setAspectRatio () //is called with no parameters
    {
        setAspectRatio(100,75); // standard, default TV aspect ratio
    }
    // other members as necessary ...
     }
```

实验 4-2：设置特征比例

在本次实验中，我们将练习创建三个重载方法，也就是三个不同的方法共用一个方法名，每个方法以不同的参数列表来区分。

实验概述

在本次实验中，我们将练习以下内容：

- 创建重载方法。
- 调用重载方法。
- 调整 PictureBox 对象的大小。

本次实验中的代码内容中包含一个有一个图片框和四个按钮的窗体。实验的任务是为其中三个按钮添加事件处理程序。每个按钮都用于调整图片框的特征比例。

实验准备

1. 启动 Visusl J++。
2. 打开 Aspect Ratio 启动项目文件（Chapter04\Lab4-2\AspectViewer.sln）。

实验步骤

1. 在 Code View 窗口中修改 Television 类。编写三个以 setAspectRatio 为名称的方法。这三个方法的返回类型均为 void。修改符的访问方式为 public。因此每个方法的开始如下：

```
public void setAspectRatio（parameter_list）…
```

注意，参数列表可以是空的（也就是说，方法可以不引用任何参数）。这些方法将分别完成以下任务：

- 引用了两个整型参数的方法将调用 viewScreen.setSize 方法，并将这两个参数值传递给方法。PictureBox 类的对象为 viewScreen，同时 setSize 方法允许调整图片框的大小。
- 不引用参数的方法将调用参数值为 100 和 100 的 viewScreen.setSize 方法。
- 引用了双精度参数方法的主体部分如下列代码所示，在该方法中，

ratio 是 setAspectRatio 方法的双精度参数的名称。

```
{
    int baseHeight = viewScreen.getSize().y;
    viewScreen.setSize((int)baseHeight*ratio), baseHeight);
     }
```

2. 修改按钮，使其满足：
   - 将一个按钮的标签设为 Movie，令其调用双精度参数方法 setAspectRatio，该方法所引用的参数值为 7.0/3.0。
   - 将第二个按钮的标签设为 Square，令其调用不引用参数的方法 setAspectRatio。
   - 将第三个按钮的标签设为 TV，令其调用引用了两个整型参数的方法 setAspectRatio，该方法所引用的参数为 100 和 75。

   修改每个按钮最简单的方法是，在设计模式中，双击窗体中的按钮。这便转换到 Java 代码窗口和正确名称的事件处理程序。在为事件添加了代码后，返回到此窗体，并双击下一个按钮。
3. 重新运行改动后的项目文件，查看每个按钮的工作情况。
4. 将本次实验的工作与本章中的示例 Chapter04\Sol4-2\AspectViewer.sln 比较一下，看看有什么不同。

下一步

我们很快就要进入下一章的学习，现在简单总结一下前面所介绍的内容：窗体、项目文件、WFC 控件、事件和事件处理程序、类、对象、成员变量、方

法、构造器、标准对话框、内置类型、菜单，最后是重载方法。到目前为止，我们已经介绍了相当多的知识，单单靠使用这些知识，我们就已经可以编写相当不错的 Windows 应用程序了。

在下一章中，将进一步巩固前面所讲述的内容，同时还将引入继承性的概念。掌握了 Java 的继承性，我们就可以方便地引用已有的代码，而不必重新编写代码。也就是说，在原有的代码资源的基础上，我们可以做更多、更复杂的工作。

第五章　继承性

在 Java 编程语言中，利用继承性可以获得已有的类，并扩展它的功能。这个已有的类可以是语言本身提供的、其他程序员编写的，或你自己原来编写的。继承性在 Java 中无所不在，从本书开始，我们就一直在暗中使用它。现在，我们来看一看是如何使用它的。

在本章中，你会了解到 Java 的继承机制和概念，包括：

- extends 关键字
- super 关键字
- 方法的超越
- 继承性的修饰符

可以练习使用：

- Visual J++ Class Outline

超类和子类

术语“超类”和“子类”描述了一个类同另一个类的关系。正如你所想象的，超类是它的子类的一种"父亲"形式。超类是拥有一个或多个扩展分支的类，其他的类用超类作为一个起点，并从那里开始构造。

关于子类的一件事情是，它们拥有其超类的所有功能。这同现实世界中消费品的交易方式类似。你可以获得一个普通的电视，或者，你也可以获得超豪华型的电视。普通的电视就是超类的一个例子，它称为 Television 类。你可以打开或关闭、调节音量、换频道等等。如果你希望描述超豪华型的 Television（有时称为特殊的），你可能会先说“它拥有普通电视的所有功能，还有另外一些特色”。

Television 的一个子类可能叫做 WideScreenTv，而它会包括我们希望一个家庭电子娱乐中心应拥有的所有附加功能。重新编写所有原始的 Television 中的代码是没有意义的，因此，为什么在定义 WideScreenTv 时不复制它呢？

在 Java 中，除一个类外，其他每个类都是一个子类。Java 中所有的类都是继承自 java.lang.Object 类。Object 类是 Java 最根本的超类。要了解这些，我们必须做在 Visual J++中，用 Class Outline 窗口来创建一个简单的类并检验它。从图 5-1 中，可以看到类 Object、Television 和 WideScreenTv 之间的关系表。

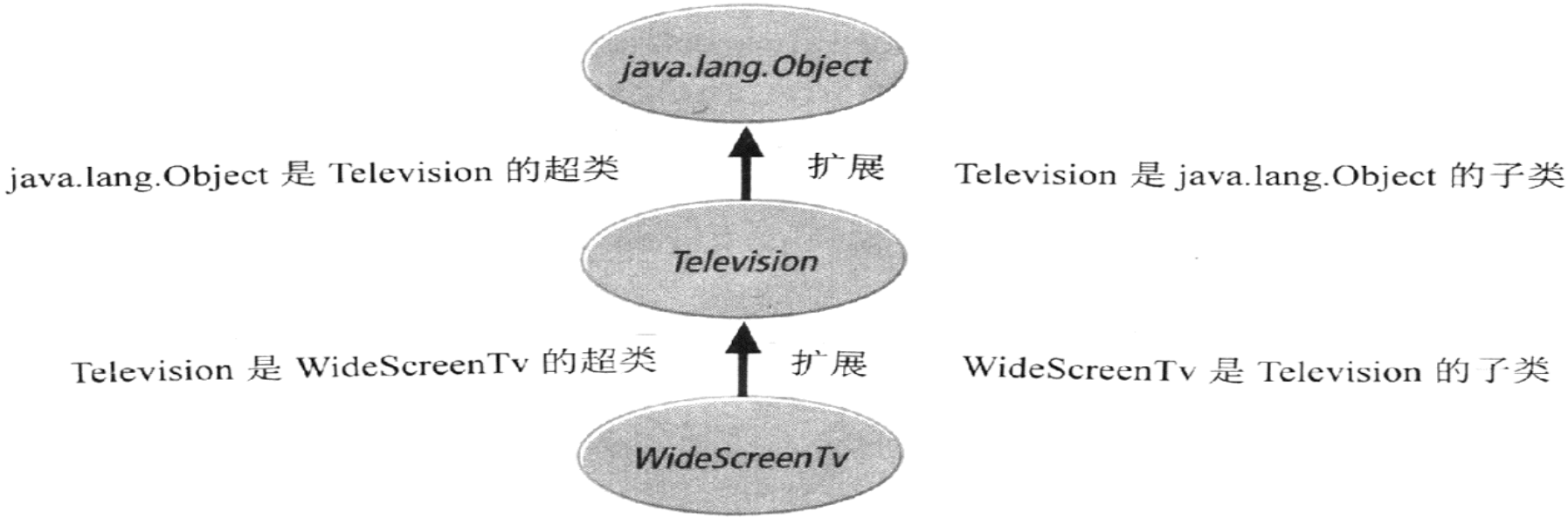


图 5-1　超类和子类的关系

使用 Class Outline 窗口

为帮助我们更有效地使用类，Visual J++开发环境提供了 Class Outline 窗口。有了它，我们可以轻易地看到类在类的层次关系中的位置。“类的层次关系”说明了子类同它们的超类及各个子类之间的关系。从技术上来说，Java 中所有的类都彼此相关，因为所有的类都是 java.lang.Object 的扩展；然而，我们可以从任何选择的类开始，来谈论类的层次关系，而且通常我们把一个基本的或简单的类如 Television 作为超类。

我们将使用 Class Outline 窗口来查看一个 Box 简单类。

在 Visual J++中，打开位于 Chapter05\BoxStarter\Box.sln 中的 BoxStarter 项目，然后调出 Class Outline 窗口。要使 Class Outline 窗口可见，在 View（视图）菜单中指向 Other Windows，然后选择 Document Outline。Class Outline 窗口中会包含下面的消息：

There are no items to show for the selected document.

要调出 Class Outline 窗口中更有趣的东西，在 Project Explorer 窗口中展开 BoxStarter 项目的列表。然后选择 Box.java，单击 View Code（查看代码）按钮（或双击 Box.java）。在 Class Outline 窗口中展开 Box。Class Outline 窗口如图 5-2 所示。

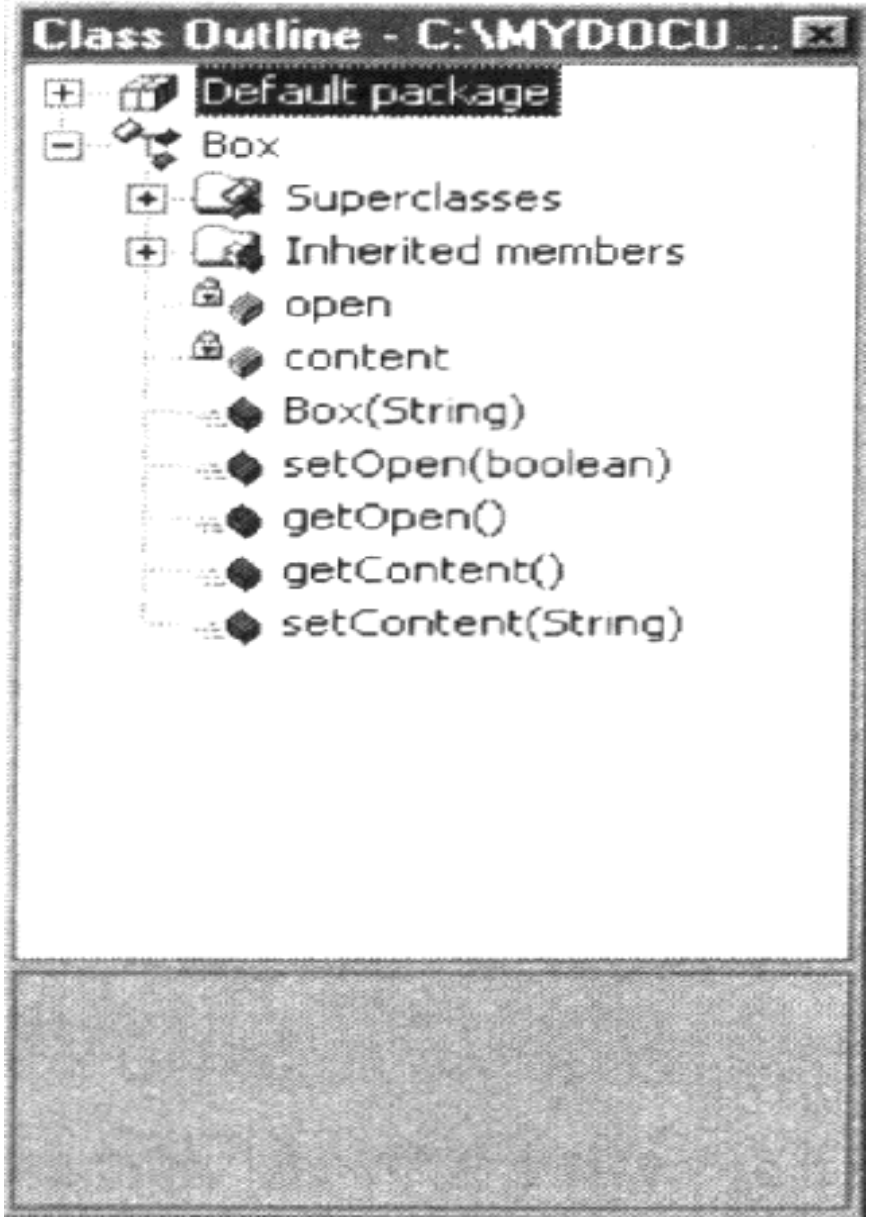


图 5-2　Class Outline 窗口，从中可以检查 Box 类

看一下 Class Outline 窗口，我们会看到 Box 拥有成员变量 open 和 content 及成员方法 Box、setOpen、getOpen、getContent和 setContent。如果打开 Superclasses（超类）文件夹，会看到 Box 是 java.lang.Object 的一个子类。打开 Inherited members（继承成员）文件夹，会看到几个方法。这些方法来自哪里呢？它们来自类 java.lang.Object，因为它是 Box 的超类。因此，Box 的一个实例可以实现所有的这些事情，而我们几乎不费举手之劳。

下面是 Box 类的定义，我们可以看到，类的声明是如何同 Class Outline 窗口中告诉我们的内容相匹配的：

```
public class Box
{
 private boolean open;
 private String content;

        public Box (String content)
        {
 open = true;
        this.content = content;
  }

  public void setOpen (boolean open)
  {
 this.open = open;
  }
```

```
    public boolean getOpen ()
    {
  return open;
    }

    public String getContent ()
    {
  return content;
}

    public void setContent (String newContent)
    {
  this.content = newContent;
    }
}
```

类 Box 没有太多的个体特征。你可以创建一个框，在里面填充内容，关上框，打开框，及取出填充内容。然而，它是开始一个类的层次关系的好地方。在这个例子中，我们说 Box 就是我们感兴趣的层次关系的超类。

extends 关键字

我们曾用 Box 类来开始我们的层次关系，现在，让我们用 Box 类的一个特

例来展开它吧！我们的特例将包括锁住包的功能。我们可以在类的定义中用extends关键字来创建一个LockableBox类，这表示LockableBox是Box的一个子类。

LockableBox拥有Box的所有属性及另外的一些东西。在BoxStarter项目中添加这个类：

```
Import com.ms.wfc.ui.*;
Public class lockableBox extends Box
{
private String password;
private boolean locked;
}
```

就像Box一样，LockableBox的一个实例可以打开、关闭等等。一个LockableBox对象拥有这些功能，是因为Box的方法已经被复制到LockableBox中。这为我们节省了自己复制代码的时间。我们仍需要做的事情是改写那些从Box中复制来的不适合于LockableBox的方法，并为我们希望LockableBox拥有的任何新的功能编写代码。

保持LockableBox在代码窗口中可见，切换到Class Outline窗口来检查LockableBox类。哪些是它的超类？哪些是它继承的方法？它们来自哪里？注意LockableBox在它的Superclasses文件夹中有两个类。Box类是它的直接超类。

如果想要在Class Outline窗口中查看Box类和查看LockableBox类之间来回切换，在Class Outline窗口的顶端打开Default package（缺省包）图标，

然后，就可以在项目中看到所有的类。双击其中任何一个，都可以在 Class Outline 窗口中查看它。

super 关键字

当我们的层次关系处于正确位置时，下一步就是把 LockableBox 同它的超类区分开来。让我们给 LockableBox 添加一个构造器。在 LockableBox 类中添加下列代码：

```
public lockableBox (String password)
{
super (″ ″ );
this.password = password;
super.setOpen(true);
}
```

这个构造器使用两个关键字（super 和 this）。我们以前已经了解了 this 的用法（在第四章），但 super 关键字还没有介绍过。super 关键字允许我们利用面向对象编程的一个最大的优点：当你使用已编写好的代码时，你不需要再编写一遍。super 关键字就允许我们这样做。

从一个构造器方法内部调用时，super 关键字调用直接超类的构造器（本例中是 Box 的构造器方法）。这样使用时，子类构造器确保 LockableBox 对象的 Box 部分被正确地初始化。为了让这个方法起作用，super 调用必须放在子类构

造器的代码开头，甚至要在局部变量定义之前。本例中，超类构造器的调用需要一个 String（字符串）参数。
这个方法中，super 的另一个用法是调用超类 Box 的 setOpen 方法。因此，super 也可以用来调用超类中的任何方法。这里，我们并不是真的需要调用 Box 的 setOpen 方法，因为 Box 对象是由 Box 构造器方法来设置为“打开”。如果 Box 的构造器方法不需要任何参数，我们就无需明确地调用 Box 的构造器方法；它会被自动调用。这意味着，不论你是否在代码中包括它，超类的构造器都会被调用。我们在 Object 类的直接子类（例如 Box）的构造器中，不会看到对 Object 类的构造器方法的调用，原因是 Object 的构造器不需要任何参数。这就解释了如果不提供一个构造器，类就必须有一个缺省的构造器。

自然，为了让 super 关键字起作用，我们必须访问它的方法。在第四章中，我们看到成员变量有四级可见性：Default（包）、public、protected 和 private。方法也有同样的四级可见性。在我们的例子中，我们将仅使用 public 方法和 private 成员变量。
super 关键字不只是用在方法上。同样的语法也可以用在超类的成员变量上。super 关键字的这种用法不常用的原因是：成员变量更经常是 private 的（这意味着，不论你使用什么语法，它们在子类中不是可见的）；而且，除非你在超类和子类中有一个同名的成员变量，你不需要在成员变量的前面指定 super。

超越方法

有时，当继承性被用来创建一个新的子类时，方法的继承定义对子类来说不够充分。在我们的 Box 例子中，setOpen 方法不够充分；我们必须首先决定这个框是否被锁住。这种 Java 机制称为"超越（overriding）"。当我们在子类中创建一个与超类中的方法同名的新方法时，超越就发生了。

我们将用 Class Outline 来超越 LockableBox 中的 setOpen 方法。在 Class Outline 窗口中调出 LockableBox，打开名为 Inherited Members 的文件夹，找到 setOpen 方法。在该方法上单击右键，会出现一个弹出式菜单。从菜单中选择 Override Method（超越方法）。Visual J++识别了被超越的方法的访问修饰符、返回类型、标识符和参数列表，然后把它们复制到你的类中。这是一个非常方便的超越类方法的途径，尤其是当你不能确切地记起方法应该如何出现在代码中的时候。在本书中，我们将超越许多不熟悉的方法。用 Class Outline 来让 Visual J++为你处理细节吧！

选择 Override Method 时，LockableBox 的代码中会出现下面几行：

```
public void setOpen(boolean open)
{
    super.setOpen(open);
}
```

我们看到，自动提供的方法简单地调用了超类的 setOpen。当这些不需要时（你可以删除 super.setOpen(open);行），这正是练习明确地处理超类的初始化，然后为子类做独特的工作的好机会。然而，在我们的例子中，如果 Box 对象被

锁住，我们并不想打开它。

把被超越的 setOpen 方法改成下面的样子：

```
public void setOpen (boolean open)
{
if (this.locked && open == true)
    {
    messageBox.show(″ Sorry, box is locked. Unlock the box first″ ,
          ″ Box is locked″ ,MessageBox.ICONEXCLAMATION);
    }
else
    }
    super.setOpen(open);
    }
{
```

在上面的代码中，如果这个框被锁住（this.locked），却试图打开它（open==true），屏幕上就会出现一个消息，说这是不允许的。否则就会调用超类的 setOpen。

Class Outline 也可以用来快速找到方法的定义。在 Class Outline 窗口中调出 Box 类，当你找到 getOpen 方法时，用鼠标双单击它。getOpen 方法出现在代码窗口的中间。在 Class Outline 窗口中右击任何条目，来查看一个带有附加选项的上下文菜单。

当我们使 getOpen 可用时，请看一下它的定义。注意，不是所有继承的方

法都需要被超越。这就是一个可以被直接继承到 LockableBox 中的方法。

实验 5-1：建立 LockableBox

我们将用 Visual J++ Class Outline 来超越一些从 Box 继承到 LockableBox 中的方法。

实验概述

在本实验中，将练习：

- 超越方法
- 使用 Class Outline

从 Box、LockableBox 的一个程序段和一个显示 LockableBox 对象的当前状态的窗体入手，你将通过超越继承的 getContent 和 setContent 方法来填写 LockableBox 的详细内容。

实验准备

1. 启动 Visual J++，并打开 Box 项目(Chapter05\Lab5-1\Box.sln)。这些代码是你按照本章的说明来建立的扩展代码（你是否已经准确地按照本章说明来做了？如果没有，那也不要紧；这些就是 Box 项目的代码）。
2. 运行项目代码。
3. 通过单击 Create Chest（创建包）按钮来建立一个框。
4. 给框输入内容字符串和密码。注意，密码字符会显示成星号（*）。

这是通过把编辑框的 passwordChar 属性设置成*来预定义好的。

5. 使用单选按钮来关闭和锁住包。
6. 单击 Examine Contents（查看内容）按钮。这时会出现一个带有框内容（你刚才输入的字符串）的消息框。我们的工作是填写代码，让框被锁住时不能查看它的内容。

实验步骤

1. 用 Class Outline 来超越 getContent 方法。在 Project Explorer 中双击 LockableBox（如果 Java 文件被放在一个窗体上，在 View Designer（查看设计器）的列表中双击该文件；否则，View Code（查看代码）会被激活）。在 Class Outline 窗口中找到 LockableBox（你可能需要打开 Default package 图标并双击 LockableBox）。在 LockableBox 的下面打开 Inherited Members 文件夹，并找到 getContent。用右键单击 getContent 并从弹出的菜单中选择 Override Method。被超越的 getContent 方法缺省情况时如下：

```
public String getContent ()
{
   return super.getContent();
}
```

2. 修改 getContent 让它检查框是否被锁住。可以使用布尔成员变量 locked 的值。如果框被锁住，用 MessageBox.show 来输出消息，说明框被锁住，并返回一个 null 值。如果框没有被锁住，返回

super.getContent 的值。

3. 用 Class Outline 以超越 getContent 方法的方式来同样超越 setContent 方法。被超越的 setContent 方法缺省情况如下：

```
public void setContent(String newContent)
{
  super.setContent(newContent);
}
```

1. 修改 setContent 方法，让它检查框是否被锁住。如果框被锁住，用 MessageBox.show 来输出消息，告诉用户在改变内容之前框必须先解锁。如果框没有被锁住，调用超类的 setContent，并把 newContent 传递给它。
2. 运行项目代码来检验当框被锁住时不能查看其内容。为了简化起见，即使框被关闭了，它的内容也能被查看。只有当框被关闭并锁住时，其内容才是不可用的。你可以把自己的程序同 Chapter05\Sol5-1\Box.sln 下的程序作一个比较。

下一步

我们已经知道了使用 extends、super 和超越方法的基本的继承性。现在我们转向类和方法的一些继承修饰符：abstract 和 final。

抽象类和方法

我们使用 Add Method 对话框时（在第三章），跳过了一对修饰符：abstract 和 final。这两个修饰符都同继承性有关。

当我们创建一个类时，我们必须包括它所有方法的代码。而这看起来明显有些盲目，有时，我们只需要为方法放置一个占位符，而不需要任何方法代码。当我们使用 abstract 关键字时，就可以这样做。

为了理解抽象类的使用，记住，当我们使用 Java 的继承性时，我们正在建立一个分支越来越多的类的层次关系。前面，我们曾用一个常规类 Box 来建立一个分支类 LockableBox。当我们把一个类扩展到另一个时，每个新的子类都比它的超类更加特殊化。另一方面，抽象类是类的层次关系上如此普通的一个类，以致于不能用它来创建任何对象。抽象类只是用来扩展的。

你不能创建一个抽象类的对象，是因为抽象类不需要包含它的所有方法的代码。只有当从它那里扩展一个子类时，抽象类才是有用的。详细情况请参阅本章前面的"extends 关键字"。

抽象类允许拥有一些或全部方法的定义代码，或者也可以什么方法定义代码都没有。abstract 关键字让我们可以省去方法定义，并保证不能用它来创建对象。你可以从抽象类扩展出一个子类，并创建子类的对象。为了创建子类的对象，抽象类中所有的抽象方法必须在子类中逐个定义。

启动 Visual J++，并创建一个称为 Shapes 的新项目。在这个项目中添加一个称为 Shape 的类。现在，我们来向抽象类中添加一个称为 area 的抽象方法。这个方法返回一个 double 值，不需要任何参数，并可公共访问。现在，你的 Shape

类的代码应该是这个样子：

```
public abstract class Shape
{
    public abstract double area();
}
```

抽象方法在抽象类的定义中充当占位符。它们只是用来强制后继的子类实现它们。

再次注意，既然 area 是抽象的，Shape 就一定也是抽象的。如果类中的任何方法是抽象的，整个类就必须被设计成抽象的。然而，你也可以拥有一个没有包含任何抽象方法的抽象类。你可以通过这样做来防止任何人初始化该类。

现在，我们把注意力转移到 area 方法上来，我们看到，它同我们在 Java 中看到的其他方法有点不同。当然，它首先包括关键字 abstract；其次就是圆括号后面紧跟着一个分号(;)。这使方法没有任何代码。这是正常的，因为 Shape 类是非常普通，以致于 area 的定义很难编写。它究竟是哪种形状？它的尺寸是多少？我们不能创建一个 Shape 类的对象；我们必须从它那里扩展，我们必须更加具体些。

因此，area 方法的这个占位符要求类从它那里继承两件事情之一：它们自己是抽象的，或者它们明确地定义 area 方法要做什么。当然，你最终必须定义 area 方法要做什么，否则，你就不能创建任何对象。

使用 Project Explorer 窗口在项目中添加一个称为 Rectangle 的新类。新类应该是这样的：

```
public class Rectangle extends Shape
```

{
}

修改 Rectangle 类，让它拥有两个 double 成员变量（length 和 width）。也可以包含一个给 length 和 width 赋值的构造器。最后，像我们超越 LockableBox 的 getContent 和 setContent 方法一样，用 Class Outline 窗口和 Inherited Members 文件夹来超越 area 方法。完成之后，类应该是下面这个样子：

```
public class Rectangle extends Shape
{
private double length;
private double width;
public Rectangle (double length,double width)
{
    this.length = length;
    this.width =   width;
}
public double area()
{
    return length * width;
}
}
```

长方形比起“形状”来要更具体些，因此，现在我们可以定义 area 方法要做些什么了。

Final 类

extends 关键字的使用没有限制。你可以把 extends 关键字用来继承一个又一个类。通常这是非常好的。然而，有时你希望防止类被继承。java.lang.System 类就是 final 类的一个例子。它是最终的类，因为你不能从 System 类中继承子类。final 修饰符通常是出于安全的目的而使用的：因为你不能继承 final 类，你不能超越它的任何方法。这保证了类中的方法不会被除类作者以外的人超越。例如，你可能有一个类包含一个能确定用户是来自局域网还是全球 Internet 的方法。如果你允许这个类被继承，被超越的安全性方法可能会被改写成总是指出用户是来自局域网。然而，实际上，除非你正在为一个敏感的系统编写类，让类保持为可继承的（即非 final 的），而把个人的方法标记为 final，总是比较好的。下一部分，我们将讨论 final 方法。

我们来看一下 Rectangle 类的例子。我们可以以两种方式来用 Rectangle 类建立正方形。我们可以简单地用相等的 length 和 width 值来初始化类：

```
Rectangle square=new Rectangle(5,5);        //a 5X5 square
```

或者，我们可以把 Rectangle 类扩展成 Square 类：

```
 public class Square extends Rectangle
 {
public Square (double length)
{
      super (length,length);
```

```
}
}
```

在上面的代码中，超类构造器（Rectangle 的构造器）的调用创建了一个四个边长度相等的长方形。然后，我们就可以像下面这样来建立一个正方形：

```
Square square=new Square(5);      //a 5X5 square
```

每种方法都有其优缺点，这取决于你的实际情况。如果你想防止定义上面看到的 Square 类，可以把 Rectangle 改成一个 final 类：

```
public final class Rectangle extends Shape
{    //... code continues
```

在这个 Rectangle 下，只有一种方法来创建正方形对象：

```
Rectangle square=new Rectangle(5,5);        //a 5X5 square
```

Final 方法

当你在方法定义中使用 final 关键字时，它意味着不可能再超越这个方法了。前面关于超越的部分告诉我们，类层次关系中的每个子类都可以拥有它自己对继承的方法的定义。然而，你在超越一个用 final 修饰的方法时就会失败。

使用 final 的主要原因是，保证某个特定的方法在类层次关系上的某点以后只有一个定义。正如我们在 final 类部分讨论的那样，这对于那些安全性非常关键的环境是非常必要的。知道方法不会在后面的类中被超越后，你可以好好优

化一下 Java 代码。

通常的 Java 继承机制是继承类中的非私有方法，并允许超越那些方法。final 关键字允许程序员强制方法的继承性不能超越代码。

如果我们把 Rectangle 类中的 area 方法改成 final，我们就会防止子类，例如 Square 超越它：

```
 public class Rectangle extends Shape
 {
private double length;
private double width;
public Rectangle (double length,double width)
{
     this.length = legth;
     this.width = width;
}
public final double area()
{
     return length * width;
}
 }
 public class Square extends Rectangle
 {
public Square (double length)
```

```
{
    super(length,length);
}
public double area () // illegal,method is final in superclass!
{
    //...
}
 }
```

因为 area 是 Rectangle 中的一个 final 方法，它在 Square 中不能被超越。这非常有意义，因为 Rectangle 的子类应该用同样的方法来确定它们的面积。

实验 5-2：用 Windows 基础类来画图

下面我们构造前面讨论过的 Shape 层次关系。新的类将提供椭圆。

实验概述

本实验中你将练习：

- 超越抽象方法
- 使用 final 修饰符

启动器项目包含了一个允许用户查看各种尺寸的长方形的窗体；程序包括了每个长方形的面积。本实验练习添加一个计算椭圆面积的类。启动器代码中包含了画椭圆所必需的代码。

实验准备

1. 启动 Visual J++。
2. 打开项目 DrawingWithWFC(Chapter05\Lab5-2\DrawingWithWFC.sln)。项目包括一个用滚动条来调整两种图形——长方形和椭圆的大小的窗体。当图形画出来时，图形的面积就显示在窗体的底部。
3. 运行项目来看看长方形是如何画出来的，及它们的面积是如何给出的。

实验步骤

1. 向项目中添加一个称为 Ellipse 的新类。Ellipse 类应该是下面的样子：

```
 public class Ellipse extends Shape
 {
private double majorAxis;
private double minorAxis;
public Ellipse (double majorAxis,double minorAxis)
{
      this.majorAxis = majorAxis;
      this.minorAxis = minorAxis;
}
public final double area()
{
```

```
        return Math.PI * majorAxis / 2.0 * minorAxis / 2.0;
    }
}
```

提示：注意前面 Ellipse.area 的定义中 Math 类的使用。值 java.lang.Math.PI 是一个声明为 public static final double 的成员变量，它表示数值π。如果你正在搜索数学函数，Java 的 Math 类是一个好地方。

2. 在窗体文件 RectAndEllipse.java 中的两个面积被标上 TODO 标记，这表明当 Ellipse 类完成时，它们应该被删除。删除带 TODO 标志的注释行，并去掉对使用 Ellipse 类的代码的注释。
3. 再次运行项目。这一次你可以画一些椭圆了。
4. 你可以把自己的程序与 Chapter05\Sol5-2\DrawingWithWFC.sln 下的程序作一个比较。

下一步

现在，我们已经了解了基本的继承性，我们将通过讨论首先使 Java 出名的小程序，来加强我们得来不易的知识。

第六章　创建小程序

直到现在，我们一直都是在使用应用程序（application）。然而，使 Java 语言成名的正是运行在 Web 网页上的 Java 小程序（applet）。应用程序是一个独立的程序，而小程序需要在 Web 浏览器或其他的小程序浏览器中才能执行。Java 小程序的好处是它们可以运行在任何计算机（包括在 Internet 网上的计算机）上，只要它有 Java 的 Virtual Machine（虚拟器）。

本章中你将学习：

- Java 小程序
- 把小程序同 Web 网页关联起来
- HTML（超文本标记语言）

你将有机会：

- 用 Visual J++小程序模块来建立一个 Java 小程序
- 把小程序嵌入到 Web 网页
- 使用 Java 绘图软件包 java.awt
- 用 Java 1.02 事件模块来响应事件

小程序

在我们学习 Java 和 Visual J++的过程中，已经了解了窗体、类、对象、控件、事件、属性、事件程序等等。所有这些将帮助我们来探究小程序、HTML 和 Web 网页。

正如我们已经了解的那样，类是为对象而设计的。一旦你创建了类的一个对象，就可以调用该对象的由类定义的方法。

窗体只是我们可以在上面放置控件的一种简单的类。控件，例如按钮、标签和列表框，是由 Visual J++环境尤其是 WFC 库提供给你的类。当你在 Forms Designer 中把控件拖到一个窗体上时，你就在 Form 类中创建了该类的一个成员变量。记住，你创建的窗体（缺省情况下称为 Form1）是名为 Form 的 Visual J++类的子类。

Java 小程序跟窗体很相似。每个小程序都是 Applet 类的一个子类。因此，正如 Form 是所有窗体类的超类一样，Applet 是所有小程序的超类。

运行一个小程序需要支持 appletviewer（小程序浏览器）。最常用的小程序浏览器是 Web 浏览器，例如 Microsoft Internet Explorer。因为 Web 浏览器起着一个小程序浏览器的作用，它必须是支持 Java 的，就像今天大多数的 Web 浏览器那样。使用简单一些的小程序浏览器，也可以不用打开 Web 浏览器，就可以快速浏览小程序。我们很快就可以看到，Visual J++提供了这种简单的小程序浏览器。

java.applet 软件包

Applet 类在 java.applet 软件包中定义。实际上，java.小程序中唯一的类就是 Applet 类。剩下的定义是关于接口的。我们将在第八章中讨论接口。

java.applet 中定义的接口特指：

- AppletContext（用来帮助小程序与同时运行的其他小程序进行通信）
- AppletStub（用来编写小程序浏览器；这个接口很少被小程序程序员使用）
- AudioClip（当你希望小程序能播放声音文件时使用）

我们将在第七章中使用 AppletContext 和 AudioClip。本章我们将集中讨论 java.applet.Applet 类。

Web 网页

Web 网页是 Java 小程序的宿主。由 HTML 命令组成的 Web 网页用 Internet 浏览器来显示。Web 网页的优秀之处在于，它们允许每个人以相似的格式来访问信息，不论计算机的硬件和软件是什么样的。

本书中，我们不会完全进入到复杂的 HTML 中，但我们将看到，足够的 HTML 可以帮助你好好地支持小程序。

可移植性

小程序可以运行在任何计算机上，只要该计算机有 Java 的虚拟机和某种小程序浏览器。在我们开发小程序的过程中，将会看到，在 Visual J++ Quick View 和 Microsoft Internet Explorer 中运行的小程序的样子。

为了在连接到虚拟机的任何显示屏幕上画图，Java 提供了称为 java.awt 的 Abstract Window Toolkit(AWT)软件包。我们将用 AWT 在 Web 网页上画一些简单的图形。

安全性

由于要运行在几乎所有的计算机上，Java 小程序在设计时不得不考虑到安全性。当你从 Web 浏览器中运行一个 Java 小程序时，该小程序不能：

- 访问当地硬盘
- 使用除启动它的服务器之外的任何服务器

该小程序访问的硬件只是显示屏幕、鼠标和键盘。

这并不意味着 Java 小程序没有用处或趣味。Java 小程序可以接收输入、提供范例、计算数值、显示动画、增强 Web 网址导航，或只是给 Web 网页带来一些魅力。安全约束控制着小程序，它们使 Web 成为网上冲浪的安全之处。

创建小程序

我们现在用 Visual J++中的小程序模块来创建自己的小程序。这同我们在第二章中用来创建应用程序的窗体模块非常相似。用了这个工具之后，我们就可以建立一个新的 Java 小程序，并立即运行它。后面，我们将看到如何从脚本建立一个小程序。

使用 Applet 模板

启动 Visual J++，如果新建项目没有自动出现，从 File（文件）菜单中选择 New Project（新建项目）命令。在 Visual J++ Projects 文件夹下面的左边一栏中选择 Web Pages 夹。在右边一栏中，我们会看到 Visual J++提供给我们的 Web 网页模板。我们打算建立一个正规的小程序，因此选择 Applet On HTML。这个选择给我们提供了一个 Visual J++小程序模板。键入 MyFirstApplet 来作为项目名。这个名字将被用来建立指定目录下的文件夹和文件夹中的解决方案文件。参阅图 6-1 来看使用小程序模板的例子。

单击 Open（打开）按钮，系统将会创建支持小程序所需的文件。在 Project Explorer 窗口中，你可以看到解决方案图标和它下面的项目图标。展开项目，你会看到支持小程序的两个源代码文件：applet1.java 和 Page1.htm。Java 文件是小程序本身代码所在的地方，而 HTML 文件是运行小程序的基本 Web 网页（参见图 6-2）。

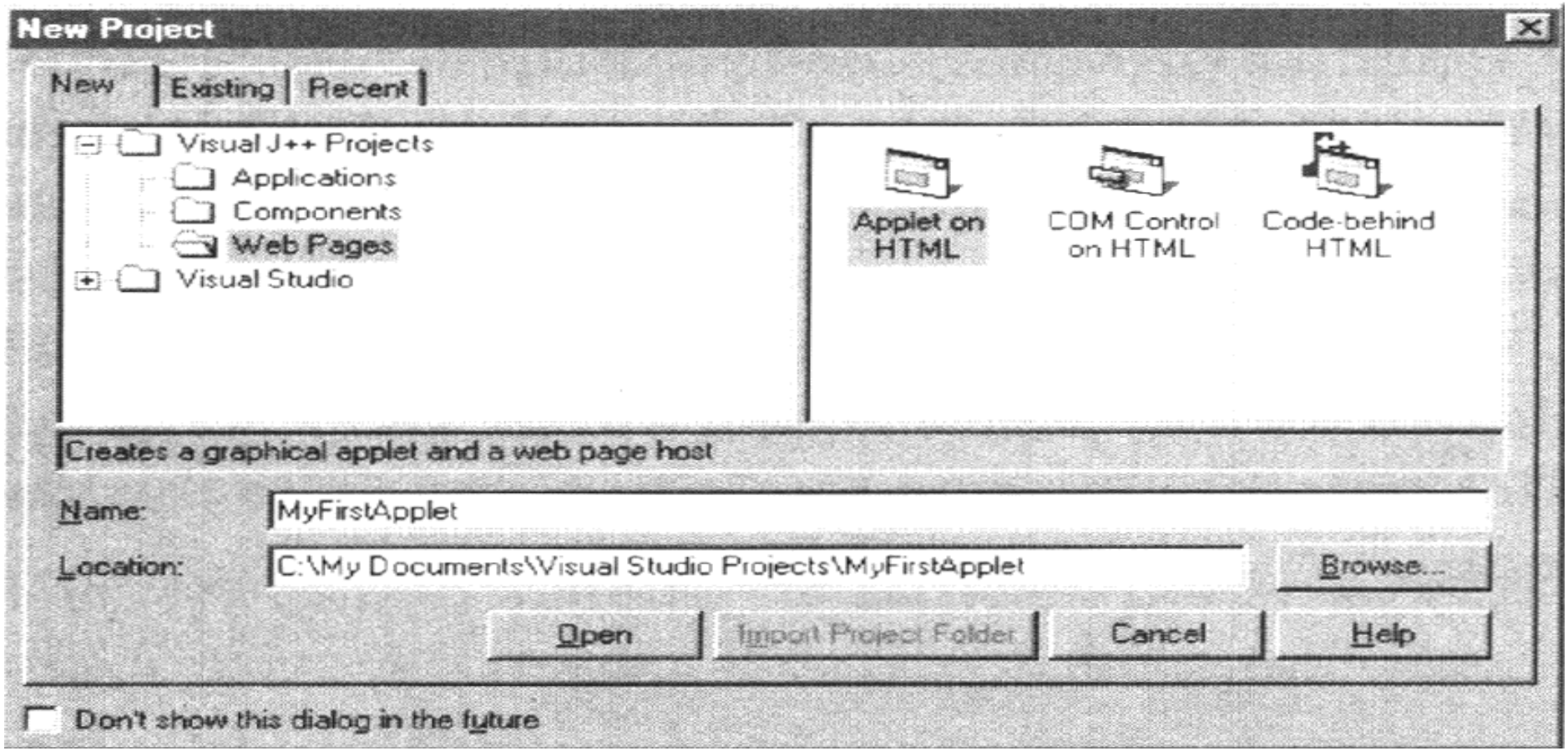


图 6-1　使用小程序模板的 New Project 对话框

图 6-2　使用 Visual J++小程序模板后的 Project Explorer 窗口

小程序模板提供了准备运行在一个 Web 网页中的完整的小程序。它没有许多内容，但拥有一个基本的小程序中所需的所有东西。从 Debug（调试）菜单中选择菜单项 Start Without Debugging（启动时不调试）。Internet Explorer 启动，而自动创建的 HTML Web 网页被载入到浏览器中。你的小程序运行在这个网页

中，同时显示一条简短的消息（参见图 6-3）。现在关闭 Internet Explorer 来返回到 Visual J++环境中。

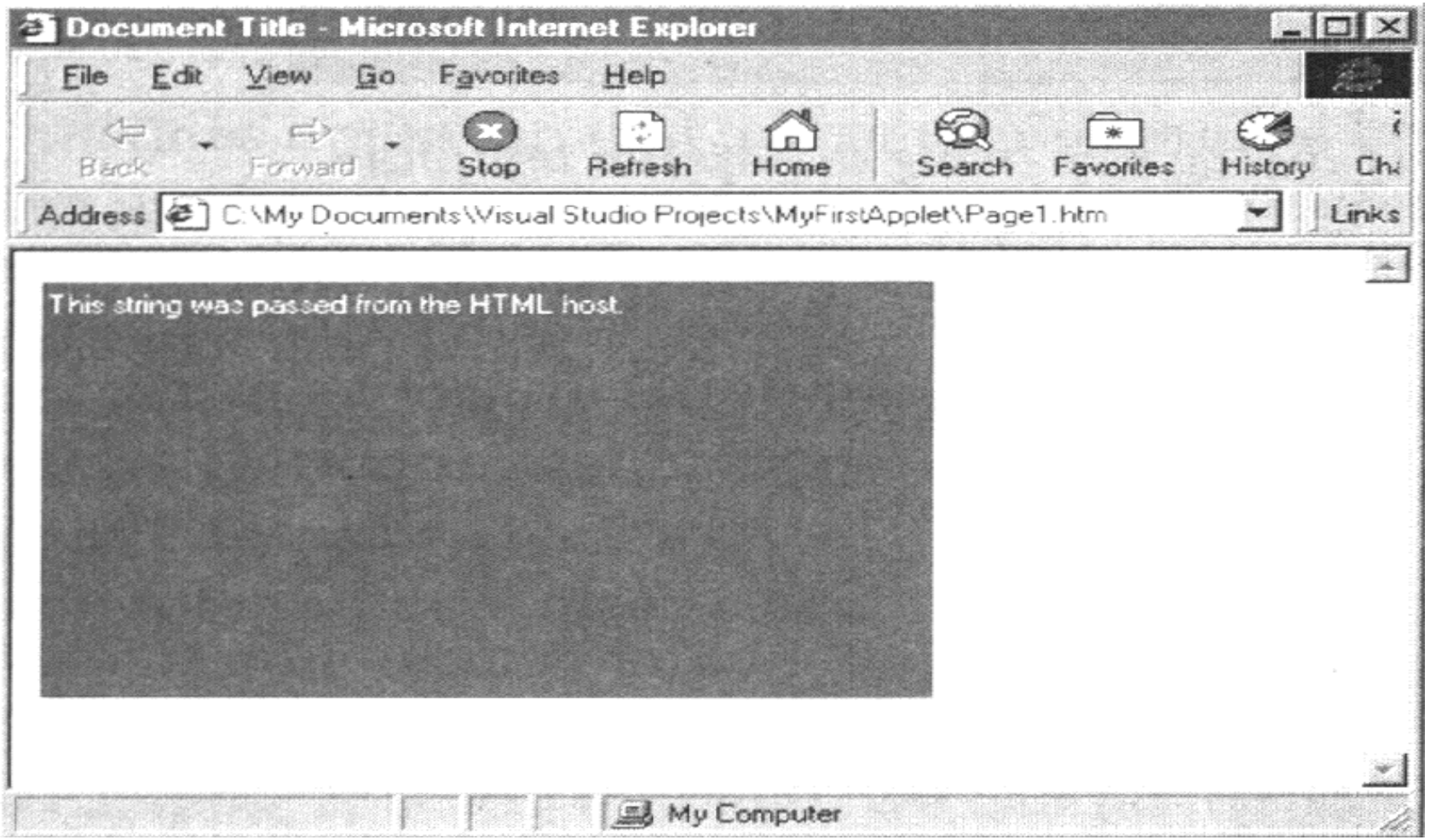


图 6-3　使用 Internet Explorer 查看小程序

我们已经用 Visual J++的小程序模板特色创建了一个小程序和 HTML Web 网页，并把小程序嵌入到 Web 网页中，我们还在 Internet Explorer 中显示了结

果。这些已经足够多了。然而，当继承 Applet 类时，用模板来生成一个小程序可能并不是最好的结果。实际上，我们不必用小程序模板的所有特色，也可以建立一个很好的小程序。但是，切记，在你掌握了更多的概念之后，小程序模板是一个很好用的指导工具。

现在，让我们来看看 MyFirstApplet 中的一些内容可以了解，当从脚本来建立一个小程序时，要寻找什么东西。通过查看由模板创建的 Java 小程序代码，我们将开始有选择地进行漫游。

在 Project Explorer 窗口中右击 Java 文件 applet1.java，并选择 View Code（查看代码）（你可能注意到 View Designer 按钮是灰色的；这是正常的。不像 Visual J++窗体，View Designer 不允许你设计 Java 小程序）。文件开头是下面的样子：

```
import java.awt.*;
import java.applet.*;
```

两个 import 语句说明了支持这个小程序的 Java 软件包。第一行引入了 Java 的 AWT。在这里，你可以找到所有的 Java 内置图形性能。用了 AWT，我们就可以在小程序窗口中画文本、直线、圆弧、长方形、多边形和其他图形。后面，我们将用这个软件包来在我们的小程序中画图。

第二行引入了定义 Applet 类的软件包。所有的 Java 小程序都是类 java.applet.Applet 的子类。因为我们已经引入了 java.applet 软件包，我们可以简单地用小程序来指定类 java.applet.Applet。

代码后面是一个 JavaDoc 注释和我们 applet 类的第一行：

```
/**
 *This class reads PARAM tags from its HTML host page and sets
```

```
 *the color and lable properties of the applet.Program execution
 *begins with the init() method.
 */
public class Applet1 extends Applet
{
```

这一行表示我们的 applet 类（本例中是 applet1）是从小程序继承来的。

在我们建立自己的 Java 小程序之前，让我们把注意力转移到支持小程序代码的 HTML 文档。

初看 HTML

要查看 Visual J++生成的支持小程序的 HTML 代码，到 Project Explorer 窗口中并双击 Page1.htm 图标。注意，HTML 代码窗口底部有三页。如果 Design 页没有被选中，现在请单击它。你在代码窗口中看到的同图 6-4 相似。

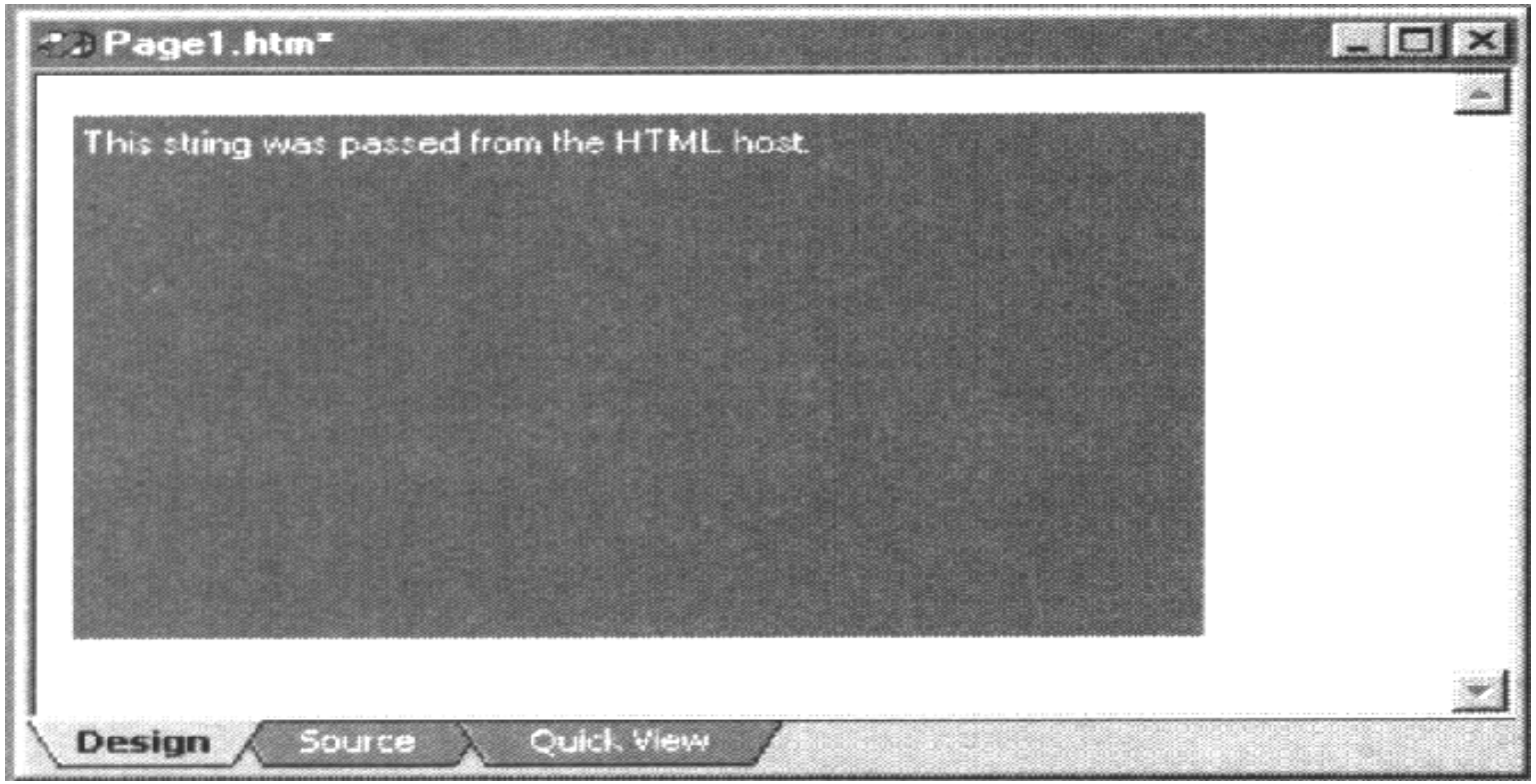


图 6-4　HTML 代码窗口，选择了 Design 选项卡

从 Design 选项卡中，项目的视图与在 View Designer（我们用来开发 Java 应用程序）中的视图类似。它显示了 HTML 页的所有组件。这个 HTML 页除了 Java 小程序之外没有任何组件，因此，这就是我们见到的全部内容。如果你到 Toolbox 窗口中单击 HTML 按钮，会看到控件被激活了（也就是说，不像前面那样是灰色的）。如果把一个 HTML Intrinsic 控件拖到窗体上，它就像 Java application 的控件一样出现在窗体中。我们将在第七章中讨论 HTML 控件。

HTML 是什么样子

单击 Design 选项卡右边的 Source 选项卡，来查看项目的 HTML 代码。我们将看到一个嵌入小程序的 HTML 文件的代码（参见图 6-5）。

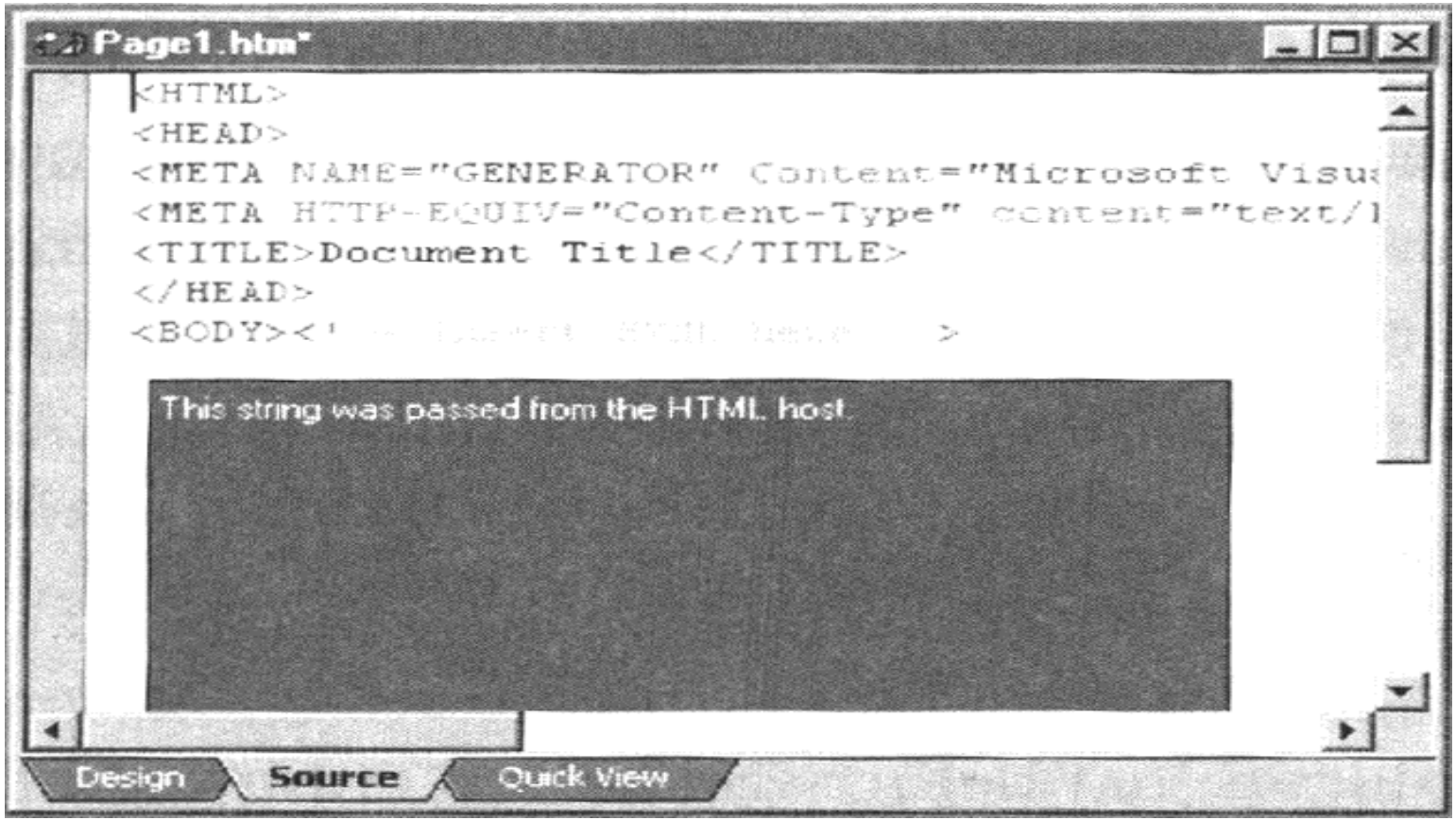


图 6-5　选择了 Source 选项卡的 HTML 代码窗口

为了查看在 Web 网页中嵌入 Java 小程序的 HTML 代码，右击小程序的图片，并从弹出式菜单中选择 Always View As Text。你所看到的是 HTML 标记的

一个集合。HTML 标记是关于 Web 网页如何显示当前页的指令。HTML 标记很容易被挑出来，因为它们以尖括号（<>）开始和结束。有些 HTML 标记在尖括号中也包含了 HTML 属性。这些属性允许我们给 HTML 标记指定特定的值。看一看 Page1.htm 的 HTML 代码，我们看到许多支持小程序的标记、属性和值。

下面是各行 HTML 代码：

```
<HTML>
```

这个标记把它后面的代码当做应该由 Web 浏览器来解释显示的 HTML 信息。在文件的末尾是</HTML>，它表示 HTML 的结束。许多标记用同样的格式来开始和结束；这就是说，一个关闭标志指示特定的 HTML 标记指令的结束。

```
<HEAD>
```

在<HEAD>和</HEAD>之间的所有内容提供了文档的有关信息。

```
<META NAME="GENERATOR"  Content="Microsoft Visual Studio 98">
```

<META>标记被用来指出是什么程序创建了该 HTML 文档、文档的作者是谁及文档来自哪个 Web 站点。

```
</HEAD>
```

这个标记结束<HEAD>部分。

```
<BODY>
```

<BODY>标记应该放在实际的文档内容之前。它为网页上所有的 HTML 组件建

立一个放置位置。<BODY>标记也包含了<APPLET>标记，后者是放置小程序代码的地方。

```
<P> </P>
```

这个代码是在小程序显示区域上方放置一个空段落的 HTML 方式。这种方式下段落周围会有许多空格。

```
<!--Insert HTML here-->
```

文本<!--Insert HTML here-->注释并指出添加 HTML 标记和文本的位置。当你能更熟练地使用 HTML 代码时，可能会在 HTML 文档的主体部分放置附加的标记。这也是 Forms Designer 放置对创建 HTML Intrinsics 控件（按钮、标签等等）十分必要的 HTML 标记的地方。现在，<BODY>区域内唯一的标记是<APPLET>和支持的<PARAM>标记。这就是我们的小程序出现的地方。

```
<APPLET
code=applet1.class
name=applet1
width=320
height=200 VIEWASTEXT>
```

<APPLET>标记有三个必需的属性：CODE、HEIGHT 和 WIDTH。NAME 属性是可选的。

CODE 属性的值是同小程序相关联的类文件的文件名。这个类文件不是 Java 源代码文件，而是 Applet1 类的编译过的 Java 代码（文件扩展名为.class）。

只有当你希望让一个小程序与同时运行的同一页中的另一个小程序进行通

信时，NAME 属性才是必要的（我们将在第七章讨论这个问题）。

HEIGHT 和 WIDTH 属性告诉浏览器给小程序保留多大的空间。VIEWASTEXT 选项让 HTML 代码窗口显示小程序代码，而不是小程序图案。

```
<PARAM NAME="label" VALUE="This string was passed from the HTML host.">
<PARAM NAME="background" VALUE="008080">
<PARAM NAME="foreground" VALUE="FFFFFF">
```

接下来就是<PARAM>标记。<PARAM>标记必须出现在<APPLET>标记的内部，通常位于小程序的"命令行"参数所在的位置。原因是小程序没有命令行，因此，你不能用参数来改变小程序的启动状态。这里的标记设置了前景和背景颜色，并传递一个当小程序运行时将在小程序中打印出来的字符串。我们将在第七章中讨论如何在小程序内部使用<PARAM>标记。

```
</APPLET>
```

这个标记结束<APPLET>部分。

```
</BODY>
```

这个标记结束 HTML 文档的主体部分。

```
</HTML>
```

这个标记结束 HTML 文档。

<APPLET>标记

我们看到了 Page1.htm 中正在起作用的<APPLET>标记。这是小程序程序员

最重要的 HTML 标记，因为它允许小程序运行在 Web 网页内部。表 6-1 列出了<APPLET>标记属性的摘要。注意 HTML 标记不区分大小写。这里按照习惯把它们写成大写。

表 6-1　HTML <APPLET>标记属性的概览

属性	用途	值	是否必需
CODEBASE	文件的可选位置	URL	否
CODE	Java 类文件的文件名	Java 类文件名	是
NAME	运行小程序的名字	字符串	否
ALT	Web 浏览器中 Java 被禁用时显示的字符串	字符串	否
ALIGN	网页上小程序的布置	Left、Right、Top、Middle 或 Bottom	否
HEIGHT	小程序显示区域的垂直高度	数字	是
WIDTH	小程序显示区域的水平宽度	数字	是
HSPACE	小程序周围的水平空格数	数字	否
VSPACE	小程序周围的垂直空格数	数字	否

其他一些有趣的标记

HTML 3.2 包含了大约 70 多条标记，用来显示 HTML 文档。本书中我们不会全部用到它们。表 6-2 概括了一些我们马上要用到的标记。

表 6-2　小程序程序员常用的 HTML 标记概览

标记	属性	用途
<BODY>	BACKGROUND、BGCOLOR、TEXT、LINK、VLINK、ALINK	包含显示网页的 HTML 代码；可以指定背景特征
 	CLEAR	强制换行
<HEAD>	无	包含描述文档本身的 HTML 代码，与显示代码相对
<P>	ALIGN	表示网页中的一个段落
<HR>	ALIGN、NOSHADE、SIZE、WIDTH	在网页中画一条水平线，对视觉上的隔离非常有用
<HTML>	VERSION	包含整个文档的所有 HTML 代码
<TITLE>	无	作为窗口标题而显示的文档标题

下面是使用上面描述过的标记的一个 HTML 文档范例：

```
<HTML>
<HEAD>
<TITLE>Demonstration of HTML tags</TITLE>
</HEAD>
<BODY>
<P>This is a paragraph on the page.</p>
<p>This is a second paragraph with
two forced <BR>line breaks<BR>in it.</p>
```

```
<HR>
<P>This paragraph is surrounded
 by horizontal lines.</p>
<HR>
</BODY>
</HTML>
```

如果你在 Web 浏览器内打开这个文件，浏览器窗口的标题栏包含了"Demonstration of HTML tags"，而窗口中显示了段落文本（<P>...</P>）。注意，第二段在单词"with,"后面没有换行，而只是在
标记特别指出的地方换行。如果改变浏览器窗口的宽度，以致于它的宽度不够，不能在一行中显示一个段落，它会绕到第二行去。

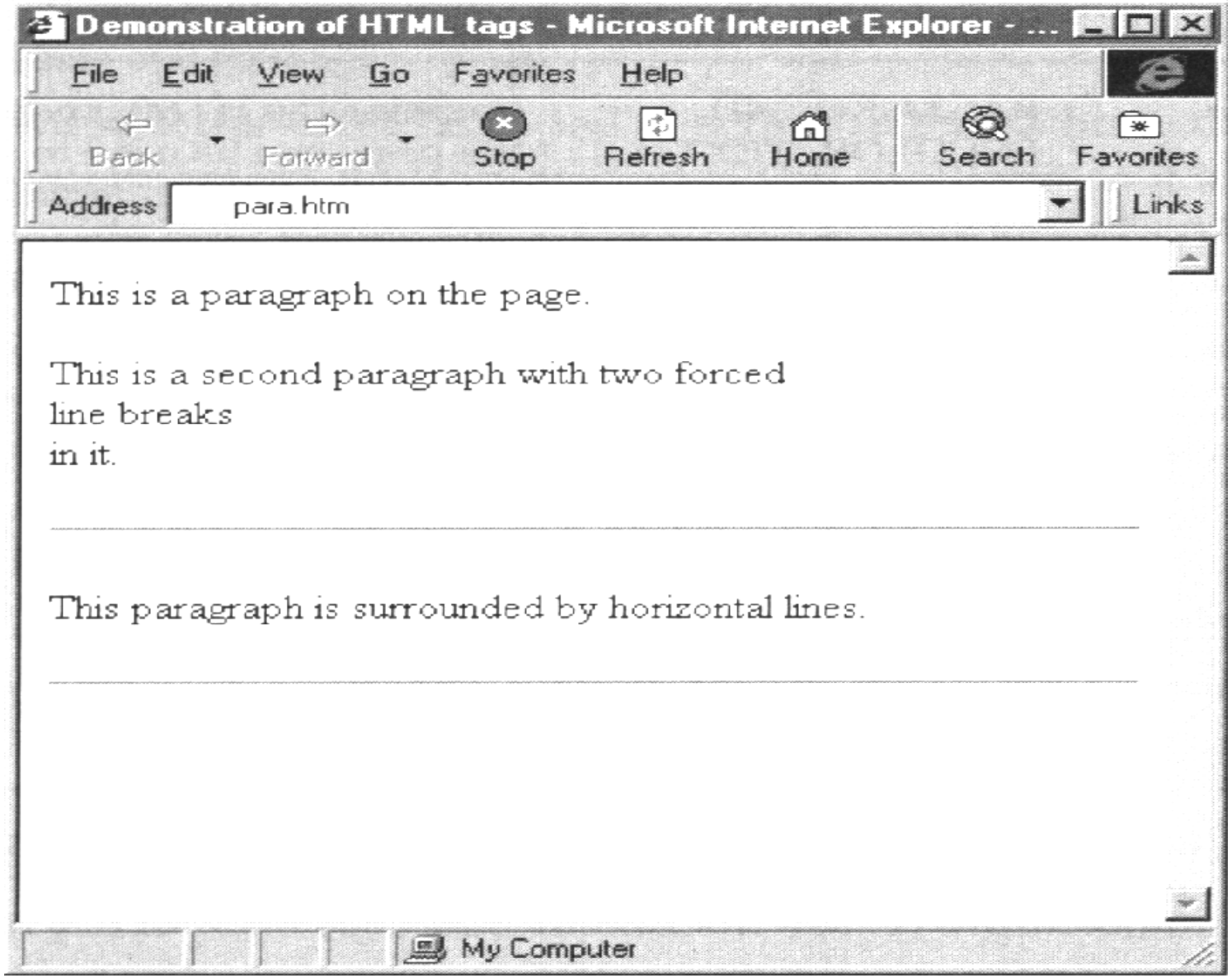


图 6-6　具有段落和分行的 HTML 标记

从头创建一个小程序

我们已经用一个 Visual J++ 小程序模板创建了一个小程序，已经看到了包含小程序代码的 Java 文件和 HTML 文档。我们已经通过从 Visual J++环境中选择 Start 来从 Web 网页中运行小程序。现在是该用脚本来创建自己的小程序的时候了。我们将创建一个新的空项目，并用 Java 和 HTML 代码文件来传播它。

从 File 菜单中选择 New Project 命令。如果系统提示保存任何打开的文件，请保存并继续。确保复选项 Close Current Solution 已被选中。在 New Project 对话框中左边的一栏中选择 Visual J++ Projects 文件夹（不要选择 Web Pages!）。在右边的一栏中选择 Empty Project 文件夹。键入 ScratchApplet，并单击 Open 按钮。

在 Project Explorer 窗口中右击 ScratchApplet 项目图标，并在弹出式菜单中指向 Add，然后选择 Add Class。这时出现 Add Item 对话框，在对话框的左边栏中预先选定了 Class。在右边的一栏中选择 Class，并输入 ScratchApplet.java 作为类名。单击 Open 按钮。

现在我们有了一个代码窗口，其中有一个空类。我们的第一个修改就是添加两个 import 语句。早先我们在 MyFirstApplet.java 中就看到过这两个语句。它们是：

```
import java.awt.*;
import java.applet.*;
```

现在，我们准备把我们的类同类 java.applet.Applet 连接起来。只要在类名后面键入 extends Applet 即可：

```
public class ScratchApplet extends Applet
{
}
```

现在我们有了applet类的基础。下一步就是在项目中添加一个Web网页。

在Project Explorer窗口中右击项目图标，并在弹出式菜单中指向Add，然后选择Add Web Page。Add Item对话框出现，在对话框的左边栏中预先选定了Web Page。在右边的一栏中选择Web Page，并输入ScratchApplet.java作为名字。

当HTML文档的代码窗口出现时，单击Source选项卡。你会看到系统已经创建好了一个HTML骨架。把<TITLE>和</TITLE>间的文本改成Scratch Applet。在代码的<P>和</P>行后面添加一个<APPLET>标记，参见下面的HTML代码：

```
<HTML>
<HEAD>
<META NAME=″GENERATOR″ Content=″Microsoft Visual Studio 98″>
<TITLE>Scratch Applet</TITLE>
</HEAD>
<BODY>
<P> </p>
<APPLET CODE=ScratchApplet.class HEIGHT=200 WIDTH=200>
Hello World! Your Browser does not support Java.
</APPLET>
</BODY>
```

</HTML>

注意，在<APPLET>开始和结束标记之间的文本，只有当浏览器不支持 Java 或 Java 支持被禁用时才显示。如果浏览器不支持 Java，则显示小程序。这时，小程序只显示背景，它是灰色的。现在，我们有了适当的小程序基础，我们就可以把下一步转移到用 Abstract Window Toolkit 软件包，来在小程序中画一些东西。

java.awt 软件包

有了 Java 的 AWT，你可以画一些图形，例如圆、椭圆、长方形和多边形。你可以像在第二、三、四章中添加 WFC 控件一样向小程序中添加控件，而且，你可以用布局管理器来为放在小程序显示区域上的元素创建布局规划（将在本章后面的“面板和布局”部分介绍）。

我们首次看到 AWT 软件包时，我们要使用它画图。注意，使用 AWT 时，显示的文本被认为是一个图形，而不是一个文本。因此，我们将用图形方法来在小程序显示区域“画”文本。

继承性和 applet 类

applet 类能工作，是因为调用它们的 Web 浏览器已经知道调用哪个方法。这是因为激活了 Java 的 Web 浏览器知道类 java.applet.Applet 中包含的方法。如果我们在小程序中创建一个小程序超类的窗体中没有的方法，Web 浏览器就不

知道存在的新方法。小程序超类为小程序定义了一个协议。为了编写我们的 applet 类的代码，必须停留在这个协议内。这并不意味着不能为小程序创建新的方法；它只是表示 Web 浏览器不能直接调用这些方法。就像对待其他 Java 类一样，我们可以添加方法来给小程序提供特殊的功能，或者，只是为了把小程序工作分成小块而添加方法。不同之处是，如果对新方法的调用被包括在继承的方法的超越定义中，Web 浏览器就只能使用新方法中的这一个。

applet 类和 Web 浏览器之间的协议是用 applet 类的层次关系建立起来的。为了确定一个小程序中可用的所有方法，依次到每个超类中（即按照类的层次关系）查看它的方法。有时，继承的方法可能是没有代码的空方法，但方法还是在那儿。

图 6-7 显示了 applet 类中的层次关系。

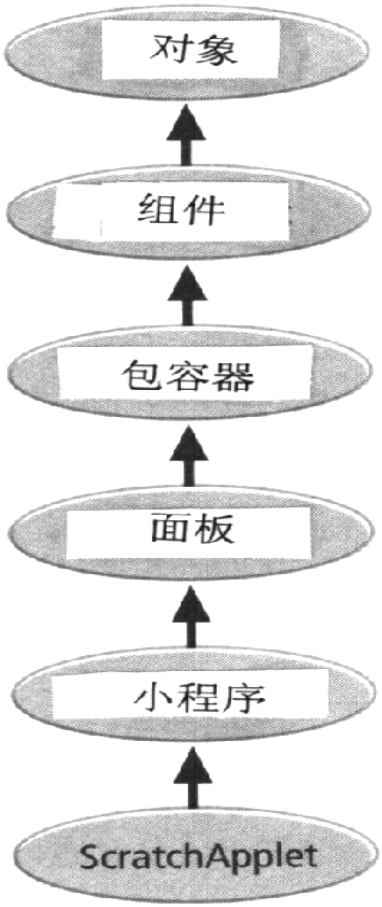


图 6-7　ScratchApplet 家族树

为了在小程序中使用超类的方法，我们要应用在第五章中讨论过的超越技术。Visual J++有一个工具来使这些工作更容易完成，我们很快就会使用它。在本章和第七章中我们也会讨论最普通的超越小程序方法。

添加画图文本

我们的小程序的初始代码会打印出一个消息。为了这些图形，我们使用了paint方法。paint方法是那些通过继承而对所有的applet类都可用的方法中的一个（paint是从java.awt.Container继承而来的）。为了在applet类中建立一个不同的paint方法的定义，我们超越了它的超类定义。

正如以前一样，为了超越类中的一个方法，你可以键入新的方法，或用Class Outline窗口来帮助你。要用Class Outline窗口来超越applet类中的方法，调出ScratchApplet.java的代码窗口。然后到Class Outline窗口中，并打开applet类的图标。打开Inherited Members文件夹，滚动窗口来找到paint方法。正如你看到的那样，我们有许多方法可供选择。幸运的是，它们是按字母顺序排列的，输入你感兴趣的方法（paint）名字的第一个字母，就会移动到以字母 p 开头的方法。paint方法恰巧是第一个方法。右击 paint 方法，并从弹出式菜单中选择Override Method。

下面的代码出现在你的小程序中：

```
        public void paint(Graphics p1)
{
    //TODO: Add your own implementation.
    super.paint(p1);
}
```

调用你正在超越的超类方法是一个很好的练习，而如果我们用Class Outline窗口来超越方法，这一行会自动提供。

在调用 super.paint 之后添加下面一行：

```
g.drawString("Hello",50,50);
```

把 Graphics 参数的名字改成比 p1 更有意义的词，确实是一个好主意。这个参数习惯的名字是 g，就像下面我们用的那样（类型 Graphics 定义在 java.awt 中）。不要忘了在对 paint 的超类方法的调用中也改变这个名字。你的 Java 文件完整的版本是下面这个样子：

```
import java.awt.*;
import java.applet.*;

public class ScratchApplet extends Applet
{
public void paint(Graphics g)
{
    super.paint(g);
    g.drawString(″ Hello″ ,50,50);
}
 }
```

在添加这个方法之后启动该小程序时，Internet Explorer 载入该小程序，窗口将会与图 6-8 类似。

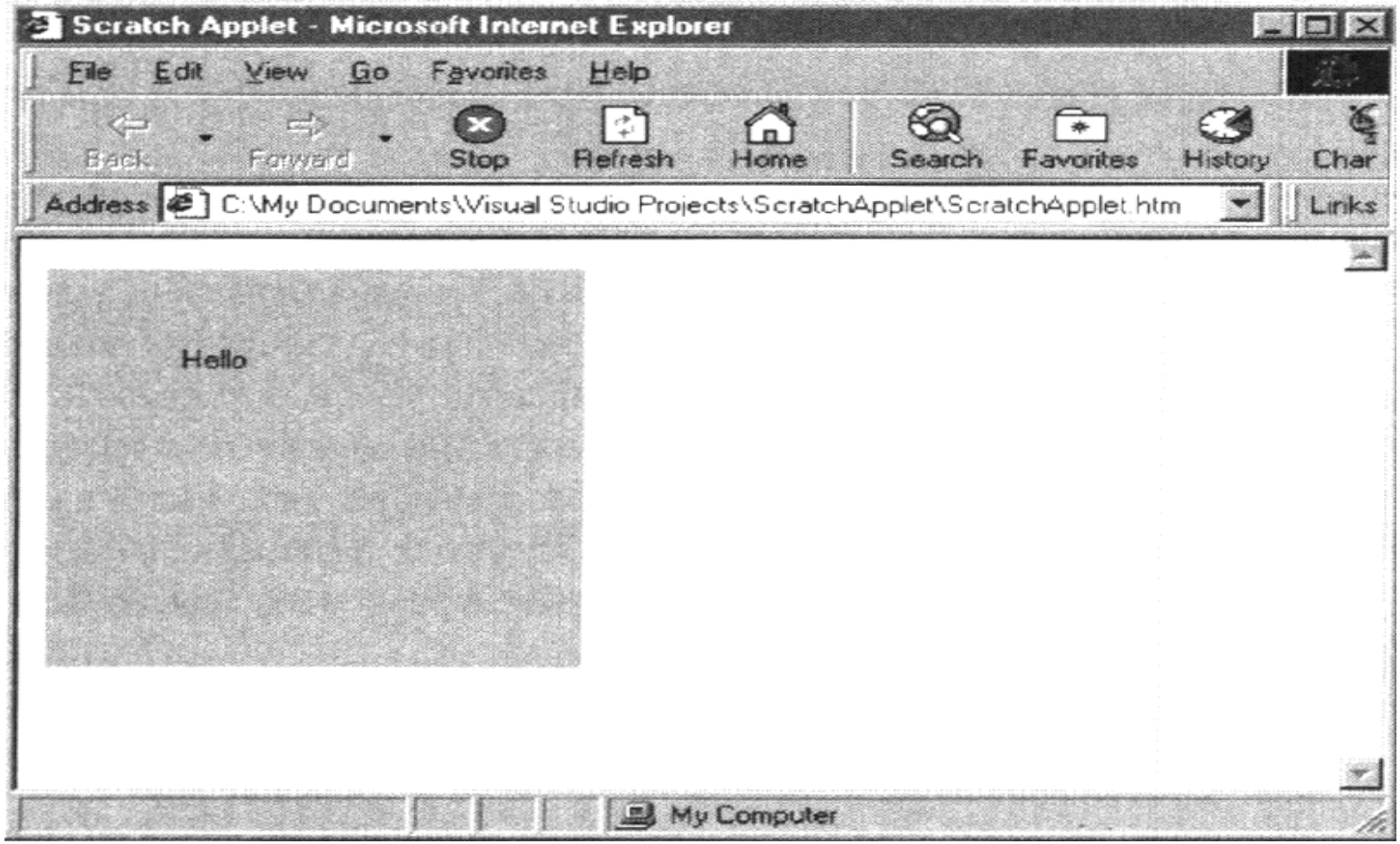


图 6-8　你的小程序具有一个新的 paint 方法

Web 浏览器创建并维护被传递给 paint 方法的 Graphics 对象，因此我们作为小程序程序员就不需要这样做了。这是又一个小程序和 Web 浏览器如何共同工作的例子。

下一步，我们将看一看 AWT 软件包如何处理事件。

响应小程序事件

我们已经知道了 Visual J++如何处理事件和事件句柄。在第二、三、四章中，我们曾用事件处理程序来同用户交流。那些事件和程序是 WFC 的一部分，而不是 Java 中的内容。Java 有它自己在 AWT 软件包内部的事件处理方式。在本节，我们将编写代码来处理小程序中的事件。

Java 中的事件处理模型

AWT 软件包有两个不同的事件模型。第一个是 Java 1.02 版的，而第二个是 Java 1.1 版的。1.02 版模型比 1.1 版模型更小更简单，但它的功能也更少一些，而且不善于处理大的应用程序。每个模型都有它自己的好处。本书中，我们使用 1.02 版模型，因为我们建立的小程序将是很小的。

就像 applet 类一样，Java 1.02 版事件处理程序利用了继承模型和超越方法。简而言之，如果你想处理事件，只要超越适当的方法就可以了。方法名指明了它处理的是什么事件。我们将首先看到鼠标事件的处理。

注意：因为 Java 中有两个事件模型，较旧的一个模型已经被忽视。这意味着 1.02 版模型已经被 Java 1.1 中的新模型代替，而且对它的支持将从语言中删除掉。当你在 applet 类中超越一个事件处理方法时，Task List 窗口会指出这一点。现在，这个消息可以被忽略，因为旧的模型暂时还够用。然而，当你打算在小程序中编写大量的事件处理代码时，应该考虑了解一下新的模型，并要知道你为旧模型写的代码最终会

过时。

在小程序中添加事件处理代码

回到的 ScratchApplet 例子，我们决定要截取一个鼠标单击动作。为了找到要超越的正确的事件处理程序，了解以词"mouse"开头的与鼠标相关的所有事件会有所帮助，实际在 Java 的 1.02 模型中，并没有 click 事件。取代 click 的是一个按下鼠标事件和一个释放鼠标事件的组合。这里，我们要处理的事件是释放鼠标事件。需要超越的方法叫做 mouseUp。下面是改变小程序窗口的背景颜色的 mouseUp 方法：

```
public boolean mouseUp(Event e,int x, int y)
{
setBackground (Color.red);
repaint ();
return true;
}
```

在小程序中添加这个新方法并运行它。当你在 Web 浏览器的小程序显示区域中单击鼠标时，对 setBackground 的调用把背景颜色改成红色。参数值是名为 red 的常量，它来自 AWT 软件包中的 Color 类。注意，这与我们在第四章用的 Color 类的常量 RED 不同（尽管效果是一样的）。对 repaint 方法的调用使我们可以看到显示区域的改变。改变属性值（例如小程序的背景颜色），直到显示屏幕被刷新，才能看得出来。调用 repaint 会在内部调用 paint，因此小程序的外观会改变。注意，我们不能直接调用 paint，因为它需要一个我们的事件没有包

含的 Graphics 型参数值。

最后，我们返回布尔值 true。从 AWT 事件处理程序中返回值 true，表示我们已经处理过这个事件，它不需要被传递到任何其他的事件处理程序。

下一个超越的方法更复杂一点。如果我们添加一个标签、一个放置标签的布局管理和一个 mouseMove 事件处理程序，当鼠标移动进入到小程序中时，我们可以跟踪光标。这些代码如下：

```
private label mousePositionLb1 = new label ();
public boolean mouseMove(Event e, int x, int y)
{
mousePositionLb1.setText(″ Cursor is at ″ +x+″ , ″ +y);
repaint();
return true;
{
```

第一行创建一个 AWT 的 Label 组件。mouseMove 方法把该标签的文本改成鼠标的当前位置。每当鼠标移动时，它就起作用。然而，如果你在添加这个方法之后再运行项目，你会发现，小程序仍然没有被改变。为了在小程序显示区域显示标签，它必须被明确地添加到小程序中。

向小程序中添加组件

为小程序用户界面安排组件，并不像在窗体上放置 WFC 控件那样简单。小程序组件由五个布局管理器中的一个来放在包容器中，这五个管理器是：FlowLayout、GridLayout、BorderLayout、CardLayout 和 GridBagLayout。我们

将在本章后面的部分中详细讨论这些。对于第一个例子，我们将使用BorderLayout管理器，并向它添加我们的组件。

在创建了一个布局管理器之后，我们可以向布局而不是直接向小程序中添加组件。当我们用Visual J++来建立基于Windows的应用程序时，我们用Visual J++ Forms Designer来在窗体上放置WFC控件。当用Forms Designer在窗体上放下一个WFC控件时，Visual J++为我们生成了大量的Java代码。这些自动提供的代码保证WFC控件准确地像我们指定的那样显示出来。然而，为了在小程序显示区域放置AWT控件，我们必须自己提供Java代码。此外，必须确保把代码放在正确的方法中，以使AWT控件在我们希望的时候出现。

在这个例子中，我们希望AWT控件立即显示出来，也就是说，当包含小程序的Web网页一被Web浏览器下载时就显示AWT控件。为了让AWT控件立即出现，我们超越applet类的init方法。Web浏览器在创建小程序实例之后，立即执行小程序的init方法。范例小程序中的init方法如下所示：

```
public void init()
{
super.init();
setLayout(new BorderLayout());
add(″North″,mousePositionLbl);
```

让我们来看看上面三行代码。当然，对super.init的调用是对init的超类方法的调用。setLayout方法被调用，并新建一个BorderLayout对象。BorderLayout对象允许我们指定组件是被放在小程序显示区域的顶部(North)、底部(South)、左边(West)，还是右边(East)。为获得显示区域顶部的标签，我们用North和组

件名作为参数来调用 add 方法。这只是一个例子而已，我们将在本章后面部分详细讨论布局。

一旦把这些代码添加到小程序中，运行它，并把鼠标移到小程序显示区域周围，来看看会发生什么事情。图 6-9 演示了小程序运行的样子。

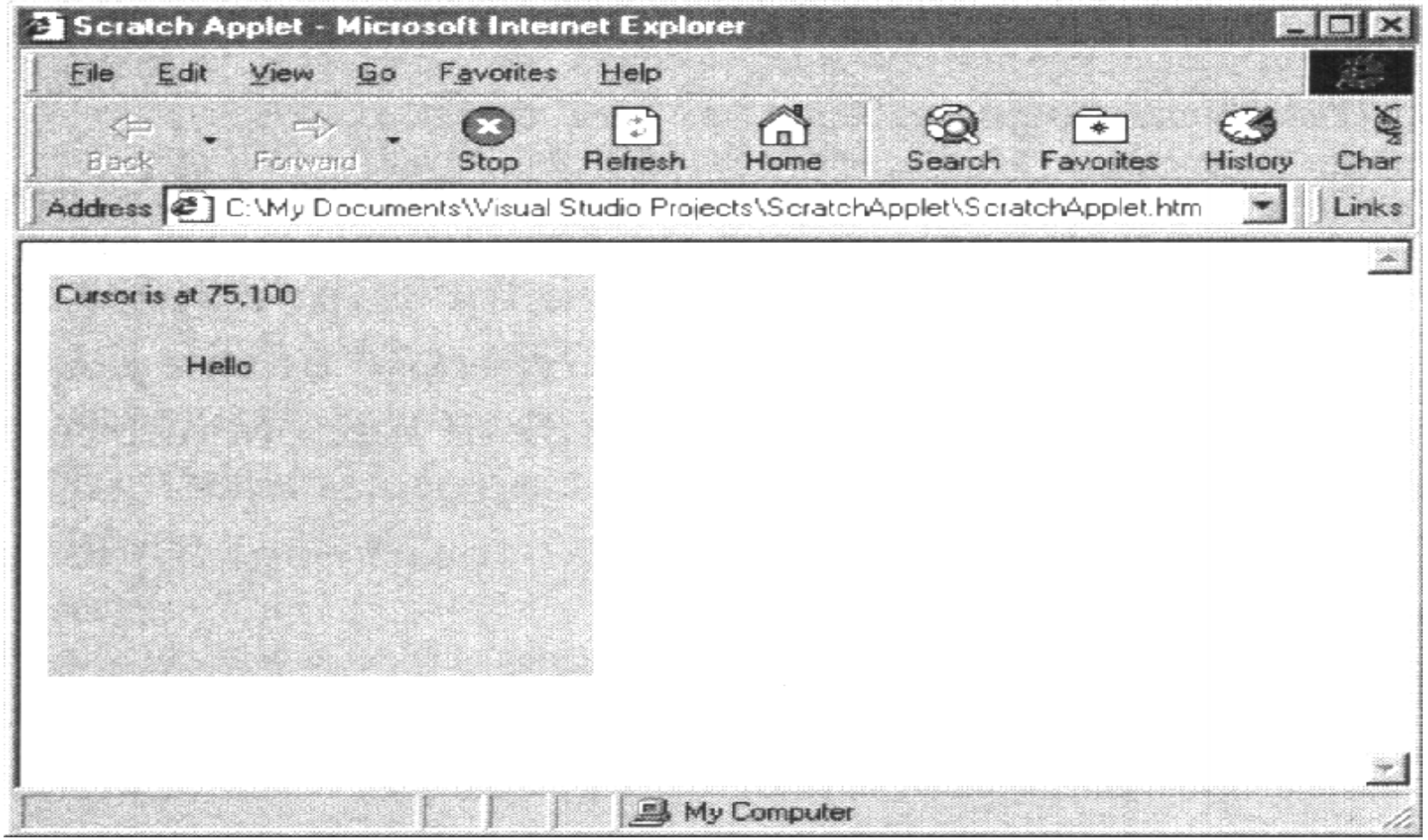


图 6-9　ScratchApplet 小程序使用 AWT Label 组件来报告鼠标光标的位置

用 AWT Graphics 对象画图

正如前面我们看到的那样，<APPLET>必需的三个参数是 CODE、HEIGHT 和 WIDTH。Web 浏览器用 HEIGHT 和 WIDTH 的值来为 Web 网页指定小程序显示区域，我们可以在该区域里面用 Graphics 对象画图。为了在小程序显示区域内画图，我们使用 paint 方法。我们用 x 和 y 坐标来映射画图显示区域，左上角为(0,0)（参见图 6-10）。

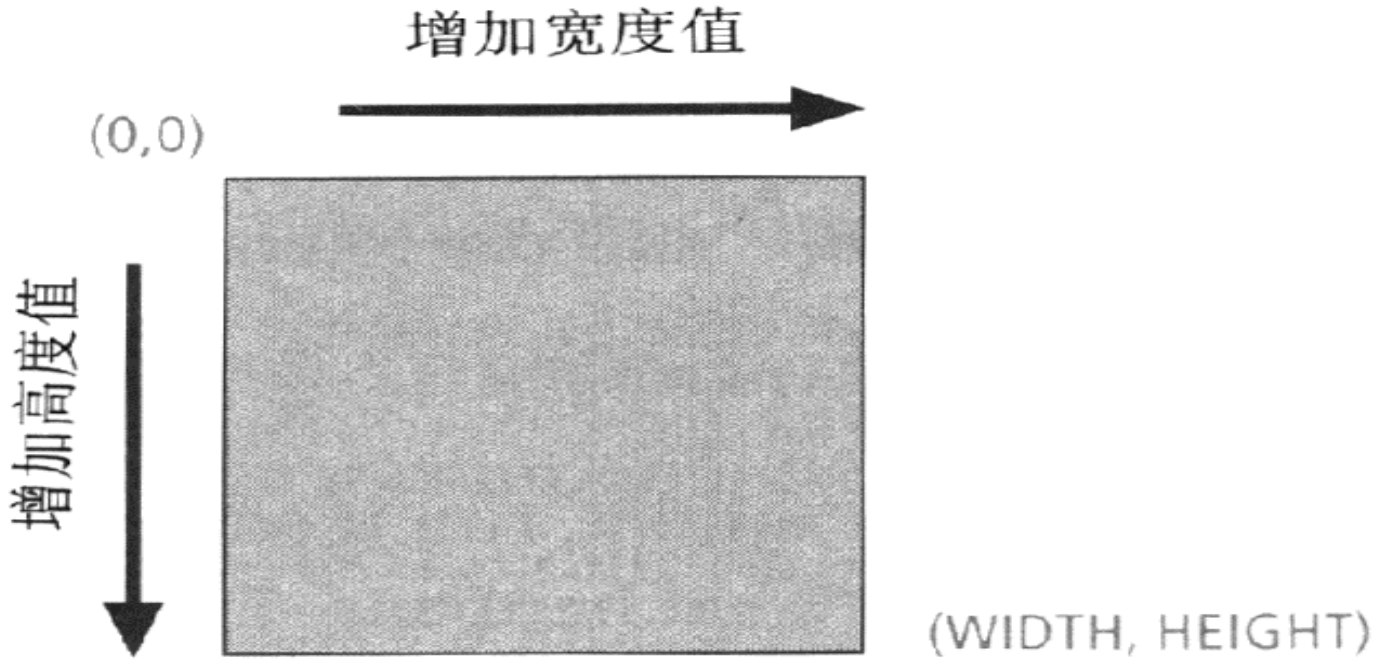


图 6-10　显示小程序的坐标系统

举个例子，假设你有个名为 MyApplet 的 applet 类。你可能会用下面的标记定义来把小程序放在 Web 网页中：

```
<APPLET CODE=MyApplet.class WIDTH=50 HEIGHT=100></APPLET>
```

然后，浏览器为你的小程序留出宽为 50 个像素、高为 100 个像素的显示区域。下面超越的 applet 类中 paint 方法画一条从左上角到右下角的直线（参见图 6-11）：

```
public void paint(Graphics g)
{
    g.drawLine(0,0,50,100);
}
```

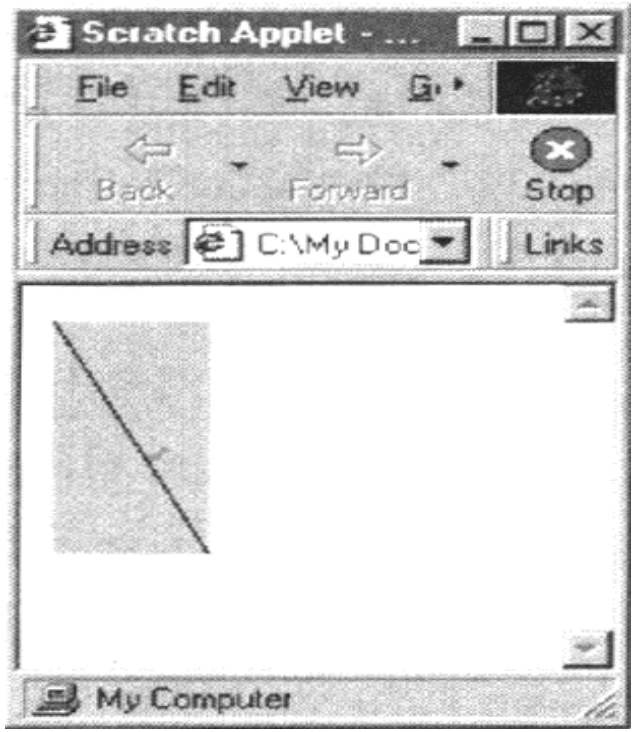


图 6-11　实际起作用的 drawLine 方法

如果你试图在浏览器设置的区域之外画图，图形不会出现在 Web 网页中；当然，小程序不知道这些。

paint 方法的参数类型为 java.awt.Graphics。这种对象类型允许在小程序的显示区域内画图形和文本。已经用 drawString 方法来显示文本，而且，刚刚看到用 drawLine 方法来画一条斜线。我们也可以用这个对象来画长方形、椭圆和多边形。

当我们用 Graphics 对象画图时，可以选择是填充图形，还是只是画出图形的轮廓线。drawRect 方法只画出长方形的轮廓，而 fillRect 方法画出长方形，并填充它。下面的代码用 fillRect 和 drawRect 来画两个正方形：

```
        public void paint(Graphics g)
{
    g.fillRect(10,10,80,80);  //draws a solid square
    g.drawRect(100,10,80,80);    //draws a square outline
}
```

fillRect 和 drawRect 方法的前两个参数表示长方形左上角的 x 和 y 坐标。第三个参数是长方形的宽度，而第四个参数是它的高度（参见图 6-12）。

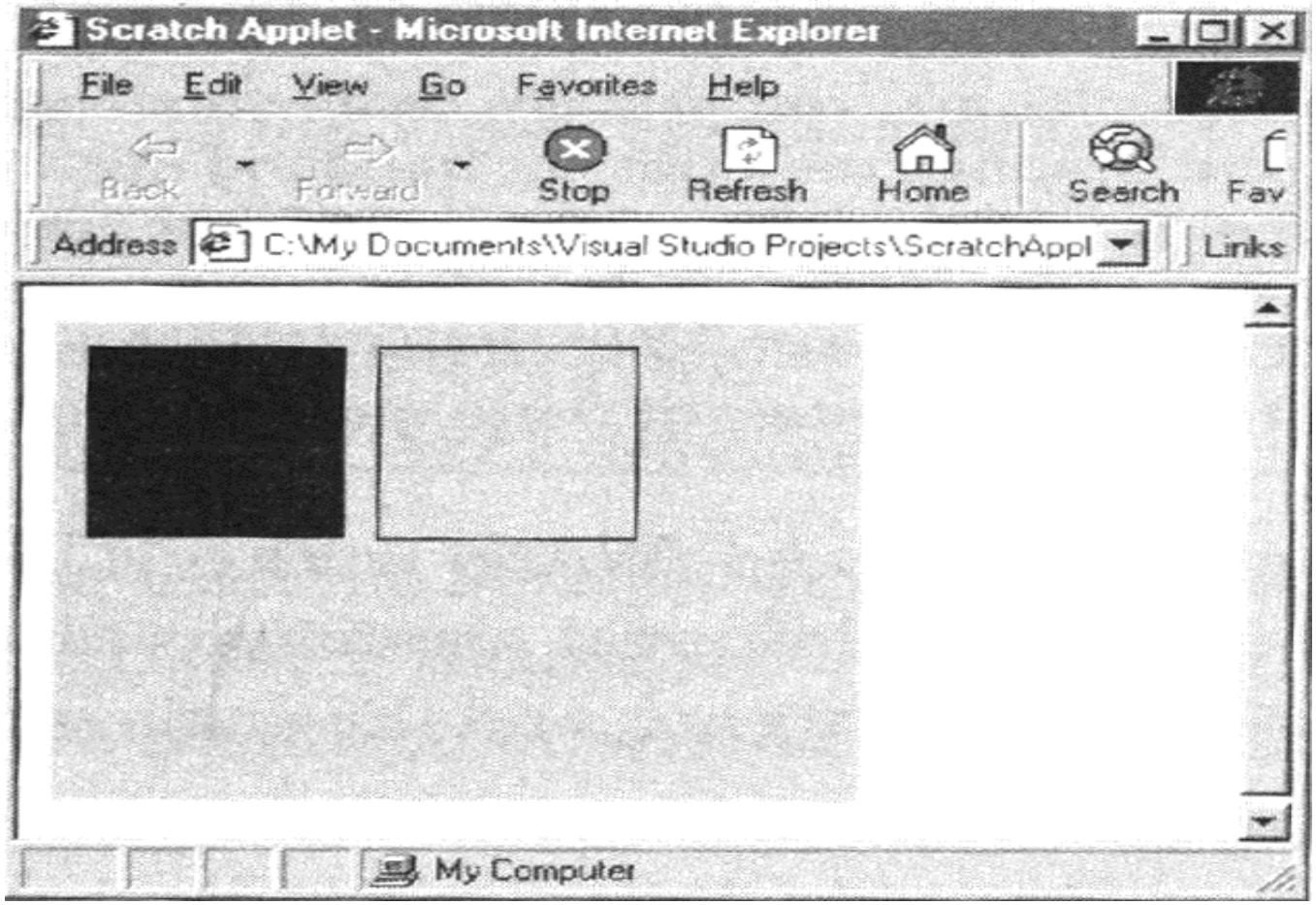


图 6-12　分别使用 fillRect 和 drawRect 画出的实心正方形和空心正方形

下面是另一个 Graphics 方法--drawOval 的调用：

```
g.drawOval(10,10,80,80);
```

这个调用在左上角为(10,10)、右下角为(80,80)的显示区域内画一个圆。简而言之，前两个参数是正方形的左上角的 x 和 y 坐标，而后两个参数是椭圆的

高度和宽度（参见图 6-13）。

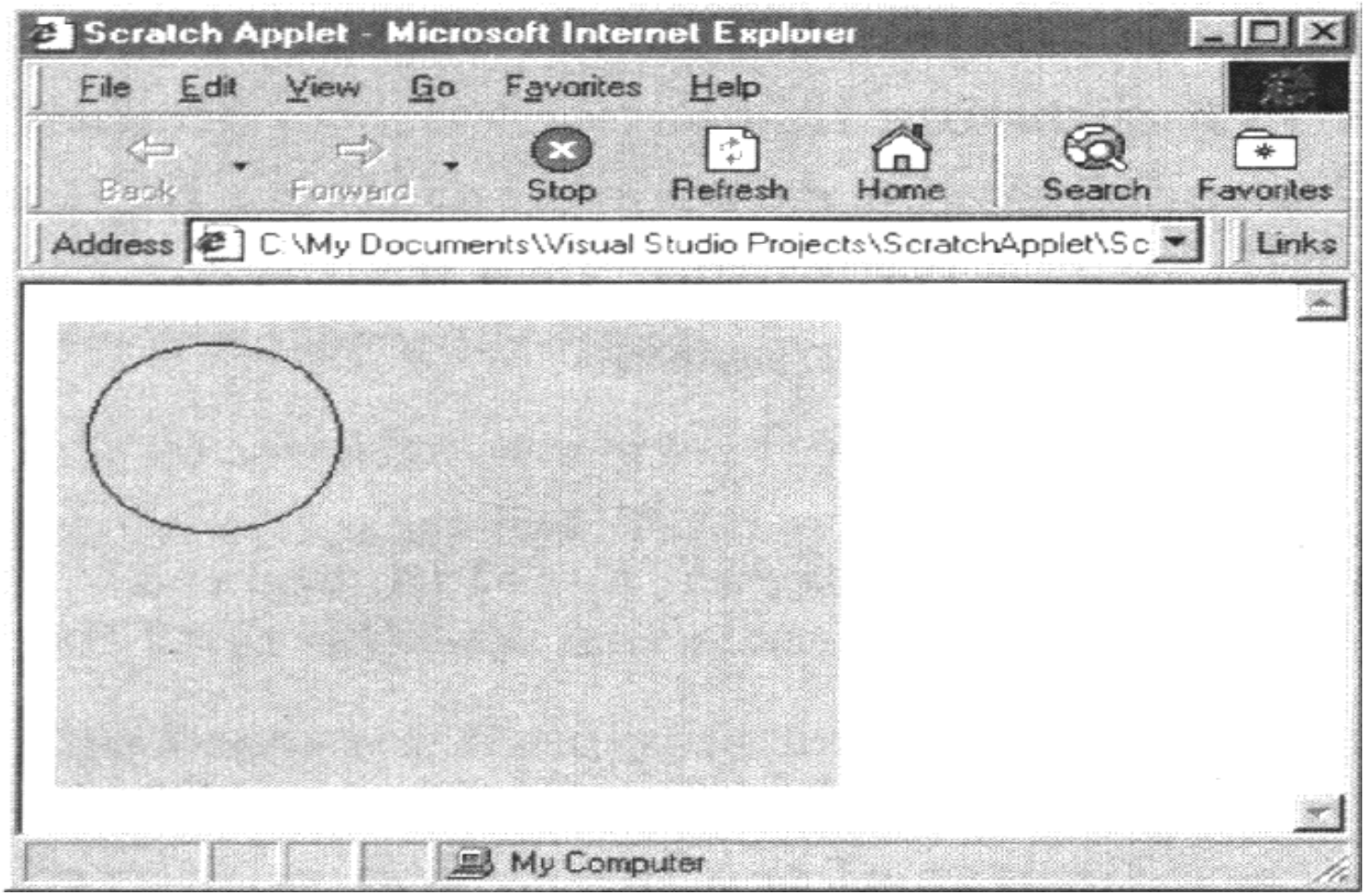


图 6-13　调用 drawOval(10,10,80,80)来画圆

fillOval 方法的作用一样，但填充椭圆。

注意：因为正方形和圆分别只是等比的长方形和椭圆，处理这对对象实例的 Graphics 方法是一样的。准确地说，drawRect 和 fillRect

被用来绘制和填充正方形和长方形，而 draw0val 和 fill0val 被用来绘制和填充圆和椭圆。在正方形和圆的情况下，这些方法的第三个和第四个参数相等。

绘制圆弧的程序(drawArc 和 fillArc)与绘制椭圆的程序一样，只是它还需要另外两个参数来表示圆弧的开始角和圆弧对应的圆心角。

下面对 fillArc 的调用在左上角为(10,10)、右下角为(80,80)的显示区域内绘制一个填充的圆弧：

```
g.fillArc(10,10,80,80,10,90);
```

圆弧从 x 轴正方向 10° 开始，并沿逆时针方向画过 90°（直到 100° 角）。参考图 6-14 来看一看圆弧的样子。

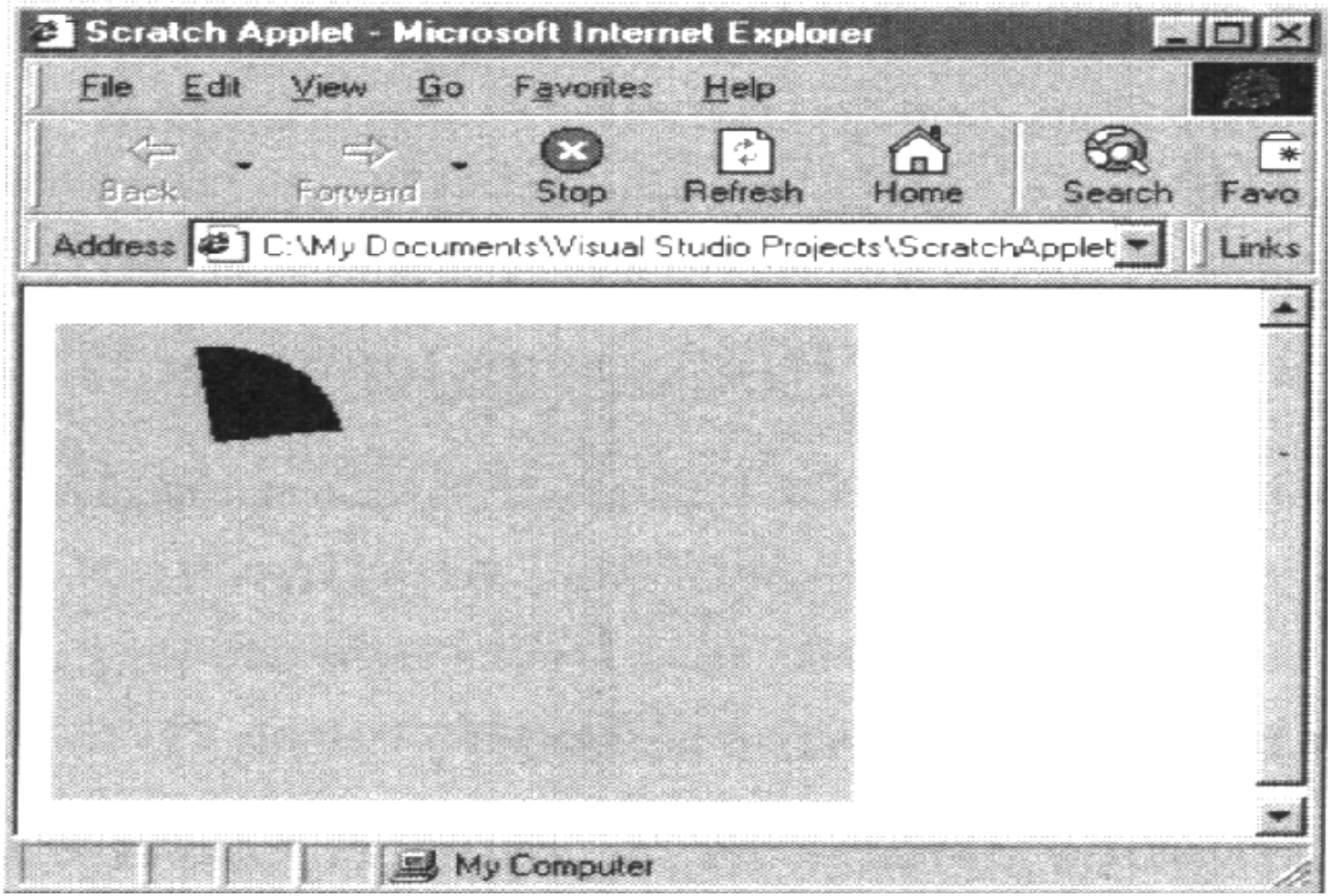


图 6-14　调用 fillArc(10,10,80,10,90)画的圆弧

实验 6-1：建立一个象限小程序

本实验中，我们建立一个在小程序中以鼠标位置为基点的象限中画图的小

程序。

实验概述

你将练习下列内容：

- 用 Visual J++来从脚本创建一个小程序
- 向小程序中添加数据变量
- 编写一个 AWT mouseMove 事件处理程序
- 使用 AWT 画图方法

有了我们刚才创建的 ScratchApplet，本实验中的小程序在鼠标移过它时将监视鼠标。然而，这一次我们只对鼠标位于小程序显示区域的哪个象限感兴趣。我们将用 AWT 程序来在该象限中绘图。

实验准备

1. 启动 Visual J++。
2. 在 New Project 对话框中创建一个名为 Quadrant 的空项目。
3. 向项目中添加一个名为 Quadrant 的 Java 类（文件名为 Quadrant.java）。
4. 向项目中添加一个 Web 网页（文件名为 Quadrant.htm）和一个<APPLET>标记。

实验步骤

1. 向 Quadrant 类中添加数据变量。

- 向类中添加 int 型的数据变量 clear、circle、square 和 arc。这些变量都应

该有修饰符 public、static 和 final。例如，下面是 clear 的定义：

```
public static final int clear=0;
```

你给其他的静态最终数据成员赋什么值无关紧要，只要每个变量有一个唯一的值就可以。

- 添加数据变量 drawnShape，并把它初始化为 clear 的值。

2. 超越 init 方法。记得要调用 repaint。

3. 超越 paint 方法。

这个方法应该首先调用 clearRect 方法来清除显示区域：

```
g.clearRect(0,0,200,200);
```

4. 把小程序显示区域分成四个象限。

- 两次调用 Graphics 方法 drawLine：首先在显示区域的顶部和底部的中间画一条水平线，然后在显示区域的左边界和右边界的中间画一条垂直线。
- 象限数为：左上角是第一象限，右上角是第二象限，左下角是第三象限，而右下角是第四象限。

5. 使用 if 语句来把 drawnShape 的值同 circle、square 和 arc 相比较。根据 drawnShape 的值来在第一象限绘制一个圆的轮廓，或在第二象限绘制一个填充的正方形，或在第三象限绘制一个填充的圆弧。你可以为圆弧选择开始角和划过的角度。

6. 超越 mouseMove 方法。

用 if 语句来决定鼠标目前在哪个象限。检测第二个参数（x 的值）和第三个参数（y 的值）的值，来作出这个决定。如果鼠标位于第一

象限，把数据变量 drawnShape 设置为圆。如果鼠标位于第二象限，把数据变量 drawnShape 设置为正方形。如果位于第三象限，把它设置成圆弧。而如果鼠标位于第四象限，把 drawnShape 设置成 clear。

7. 在设置好 drawnShape 的值之后，调用 repaint 方法来显示结果。返回值 true 来表示你已经处理了 mouseMove 事件，不需要其他方法来处理它了。
8. 使用 <APPLET> 标记来把小程序嵌入到 HTML Web 网页文件 Quadrant.htm。
9. 运行小程序，来看看当鼠标移动进入小程序显示区域时会发生什么情况。你可以把此程序同 Chapter06\Sol6-1\Quadrant.sln 下的解决方案进行比较。

下一步

我们现在可以建立基本的小程序并处理一些事件了。让我们更进一步展开 AWT 组件吧。我们将看到如何创建它们、如何把它们放在我们的小程序显示区域中，及如何处理它们的事件。

AWT 中的组件

java.awt 软件包包括了用于创建同我们前面看到的用窗体和 WFC 控件创建的接口相似的用户接口的组件。这些组件包括按钮、标签、复选项、单选项、

文本条目等等。本部分中，我们将向小程序显示区域添加一些最常用的 AWT 组件，并为这些组件编写事件处理代码。

标签

AWT 中的标签组件同 WFC 中的标签控件相似。它们允许我们在显示区域放置用户不能直接修改的文本。

多数时候，像下面这样创建一个标签，并给它指定一个字符串：

```
Label myLabel=new Label("This is my label");
```

可以用 getText 方法来获得标签的文本：

```
String myLabelText=myLabel.getText();
```

也可以用 setText 方法来改变标签的文本：

```
myLabel.setText("New Label Text");
```

为在小程序显示区域放置一个标签，我们可以使用 applet 类的 add 方法。

按钮

按钮组件是用户可以通过单击来同程序通信的基本的标签。就像标签一样，按钮也有一组 getText/setText 方法来操作显示在按钮上的文本。

按钮（和其他组件）与标签的不同之处是它们使用名为 action 的方法。下面是一个在 applet 类内部创建按钮的例子：

```
class AppletWithButton extends Applet
{
    private Button button1=new Button("Click Here");
}
```

当然，创建一个按钮本身并没有什么用处。我们必须使按钮显示在小程序显示区域中，并给它配上一个事件处理程序，使得单击它时会有所动作。问题的事件处理程序部分将在下一部分讨论。

响应 AWT 组件事件

我们通过超越继承的事件处理方法来处理事件。在 Java 1.02 事件处理模型中，只有四个事件处理方法同组件联系在一起，而其中三个经常被忽略。这使它变得非常容易，不是吗？四个事件处理方法是 action、gotFocus、lostFocus 和 handleEvent。

到目前为止，action 事件处理程序是这些方法中最常用的一个。当用户触发一个组件时，action 方法就被调用。然后，就需要我们在 action 方法中放置正确的代码，使我们能准确确定究竟发生了什么事情。我们很快会看到这样的例子。

当组件由于用户在接口间切换而获得或失去焦点时，gotFocus 和 lostFocus 方法就被激活。例如，当按钮获得焦点时，缺省情况是按钮文本获得一个黑色的轮廓；当失去焦点时，黑色轮廓就消失了。

任何其他事件是由 handleEvent 来处理。我们将超越这个方法，在小程序显示区域显示文本条目。

让我们向小程序 WithButton 类中创建并添加对 action 方法的超越：

```
class AppletWithButton extends Applet
{
private Button button1 = new Button(″ Click Here″ );
public boolean action(Event e, Object helper)
{
    if (e.target == button1)
    {
        setBackground(Color.green);
        }
        repaint();
        return true;
 }
public void init ()
{
    add (button1);
}
 }
```

这个小程序中超越的 action 方法检查 button1 是否被按下；如果它被按下，小程序显示区域的背景颜色变成绿色。在方法的开头，我们看到 action 方法有两个参数。第一个是关于刚刚发生的事件的 Event 对象，而另一个是一种帮助对象。例如，Event 对象包含一个对发送事件信号的组件的引用。因为我们只有

一个组件，发送事件信号的组件必定是 button1。因为发信号的组件是一个 Button 对象，第二个参数实际上是一个 String 对象，它的值是按钮上的文本（本例中这个字符串是"Click Here"）。每个组件都有它自己的帮助对象，它将被传递给 action 方法。

代码中的 if 语句检查事件目标是不是 button1。当然，它必须是 button1，因为它是我们唯一的组件，但它不会妨碍检查（而且这使后来添加其他组件更容易）。一旦我们决定了实际上就是 button1 被按下，我们把显示区域的背景颜色改成绿色（记住，如果控件改变小程序的外观，你必须重画小程序）。

下一步，如果事件处理方法已经做完需要做的事情了（即处理完事件），应该返回值 true。如果你想把事件传递给其他方法，则返回 false，这样其他的组件就可以检测到该事件，并做出相应的反应。在我们的例子中，不需要做更多的事情，因此，我们返回 true。

最后，为了使 Button 对象在显示区域中可用，必须调用 add 方法。因为我们希望 button1 能正确显示，我们超越了 init 方法，并调用 add 方法。这里，我们不知道按钮出现在显示区域中什么位置上。为了控制组件的显示位置（通常我们希望这样做），在此需要使用布局。我们差不多已经讨论完这个问题了。

文本区域和键盘事件

如果希望小程序能响应击键动作，我们可以超越 keyDown 或 keyUp 方法。严格地说，这些事件不是组件事件。组件事件由 action、gotFocus、lostFocus 和 handleEvent 来处理。

下面对 keyDown 方法的超越根据用户按下的按键来改变显示区域的背景颜

色：

```
 public boolean keyDown(Event e, int keyValue)
 {
switch (keyValue)
{
     case ‘r’ : setBackground(Color.red);
                break;
     case ‘g’ : setBackground(Color.green);
                break;
     case ‘b’ : setBackground(Color.blue);
                break;
}
repaint();
return true;
 }
```

keyDown 方法的第二个参数是用户按下的键盘上键的整数值。switch 语句允许我们选择是哪个键，并采取相应的动作。参阅第十一章来了解 switch 语句的详细情况。

注意，背景颜色的改变不会改变显示区域的显示，因此，我们必须调用 repaint 来更新显示。最后，我们返回值 true，因为不需要其他方法来处理这个事件了。

如果想要处理非打印按键（像功能键 F1 到 F12），Event 类已经定义了名字来帮助你。如果我们把上面的 switch 语句改成下面的样子，这样，按下 F1、

F2 和 F3 会把背景颜色分别改成红色、绿色和蓝色：

```
 switch (keyValue)
 {
case e.F1: setBackground(Color.red);
                break;
case e.F2: setBackground(Color.green);
                break;
case e.F3: setBackground(Color.blue);
                break;
 }
```

不是所有的按键都在 Event 中有名字，因此，我们必须用它们在 Unicode 字符集中的数字值。如果你使用过 ASCII 字符集，很容易知道 Unicode 字符集中的前 128 个值与 ASCII 字符集相同。Unicode 字符集中 Ctrl-A 的值是 1、Ctrl-B 是 2、Ctrl-C 是 3 等等。让我们修改 keyDown 方法，以使程序可以接受控制键。也就是说，现在用户必须在按下字母键的同时按下控制键(Ctrl)，来使颜色改变：

```
 public boolean keyDown(Event e, int keyValue)
 {
switch (keyValue)
{
     case 18 : setBackground(Color.red); // control r
                  break;
     case 7 : setBackground(Color.green); // control g
```

```
                    break;
        case 2 : setBackground(Color.blue); // control b
                    break;
    }
    repaint();
    return true;
     }
```

如果你想知道修饰键是否被按下，不论是哪个键，都可以使用 Event 类中的 shiftDown、controlDown 和 metaDown 方法。下面是检测右击的例子：

```
     public boolean mouseDown(Event e, int x, int y)
     {
    if (e.metaDown ())
    {
        // right click detected
    }
     }
```

一旦确定了要处理的事件，便可以 AWT 中的两个组件来帮助你处理文本。这就是类 TextField 和 TextArea。TextField 只允许输入一行，而 TextArea 支持多行输入。它们都是从 TextComponent 继承来的，因而有许多公共的方法。我们这次只讨论 TextField。

当你创建一个 TextField 对象时，可以不指定它的大小：

```
TextField firstNameTxt=new TextField();
```

或者，你可以指定字符域有多少个字符宽：

```
TextField lastNameTxt=new TextField(10);
```

或者，可以指定初始情况下字符域中有那些文本：

```
TextField jobDescriptionTxt=new TextField("Computer Bum");
```

或者，可以同时指定两个参数：

```
TextField currentPostTxt=new TextField("Flunky",20);
```

有了其他组件和控件，你可以得到 getText 和 setText 方法的帮助。TextField 也允许我们处理选中的文本。TextField 中有一个 getSelectedText 方法，来处理用户选择的文本。下面就是一个例子：

```
public class ShowSelectedText extends Applet
{
private TextField blankTxt = new TextField(10);
private Label message1 = new Label(″ Selected Text is ″ );
private Label message2 = new Label (″                ″ );
public void init()
{
    add(blankTxt);
    add(message1);
    add(message2);
    super.init();
```

```
    }

    public boolean keyDown(Event e, int keyValue)
    {
        message2.setText(blankTxt.getSelectedText());
        return super.keyDown(e, keyValue);
    }
     }
```

如果你运行这个小程序并在名为 blankTxt 的 TextField 对象中选择一些文本，文本将被反映为 Label 对象中的 message2 的值。然而，你会注意到还有一些奇怪的事情。

首先，如果你用鼠标在文本域中选择文本，在 message2 Label 对象中不会显示任何东西。这是因为我们只超越了事件处理程序 keyDown，因而 keyDown 是能修改 message2 的文本属性的唯一方法。如果希望程序能对用鼠标选择的文本作出反应，我们将不得不超越一个鼠标事件处理程序，例如 mouseMove 方法。

其次，如果用 Shift 键和方向键来选择 blankTxt 文本域内的文本，你会注意到，只有选择动作后的一个字符被显示出来。举例说，文本域中的文本是"Java"，而第一个 a 和 v 被选中（高亮显示），在"Selected Text is"后将只显示一个 a。如果再选择到第二个 a，出现在"Selected Text is"后面的就是 av。这是因为这个版本的 keyDown 在 blankTxt 的文本属性被更新为包括第二个 a 在内之前就被调用了。通常情况下，这是正常的，因为用方向键来增加选择不会引发对选择的检测。通常，按钮、控制快捷键（例如 Ctrl+C 来复制）或访问键（例如 Alt+F 来拉下 File 菜单）是触发对当前选择的文本进行检测的事件。

这个例子更好的事件处理程序如下所示：

```
 public boolean keyDown(Event e, int keyValue)
 {
// if the user pressed control c or control x, update the
    displayed selection
if (keyValue == 3 || keyValue == 24)
{
     displaySelection();
}
return super.keyDown(e, keyValue); ″
```

面板和布局

迄今为止，我们不能控制放在小程序显示区域中的组件在哪里显示。布局允许我们指定组件的常规行为和位置（"常规"一词是因为我们不能把组件放在显示区域中准确的某点上），就像我们用 WFC 能做到的那样。这是因为 Java 小程序要运行在 WWW 上的任何显示设备上。用相对坐标的方法，而不是用精确坐标的方法，来安排小程序的布局，可以确保小程序尽可能地显示在任何屏幕上。为了做到这一点，我们用 AWT 布局对象来处理这件工作。

java.awt 软件包给我们提供了五种布局管理：

- FlowLayout 是缺省的布局；如果我们不指定自己的布局管理，这就是我们得到的布局。FlowLayout 简单地把下一个控件放在最后添加的组件的右边，并在必要时绕回到下一行。

- GridLayout 允许我们为组件设置网格。每个组件占据网格中的一个单元，而所有的组件都是一样的大小（都是网格中的一个单元大小）。
- BorderLayout 的作用就像一个地图。当我们使用 BorderLayout 布局管理时，我们把各个方向指定为一个字符串（"North"、"East"、"South"、"West"和"Center"），而控件被放在显示区域的顶部、右部、底部、左部和中心。
- CardLayout 用来堆叠组件，一个组件在另一个的上面，就像一副扑克牌。如果我们希望一次只有一套组件中的一个是可用的，这个布局管理效果很好。
- GridBagLayout 类似于 GridLayout，但支持更复杂的布局。用 GridLayout 时，每个组件被固定在一个单元中，而所有的单元都是一样的大小。用 GridBagLayout 时，就没有这样的限制。然而，灵活性是有代价的：GridBagLayout 也是最复杂的布局。由于建立 GridBagLayout 布局管理的复杂性，我们将不详细讨论它是如何做的。

下面，我们将看到用 FlowLayout、GridLayout、BorderLayout 和 CardLayout 在小程序显示区域中安排组件的例子。

记住，在小程序被创建的时候，通过超越 init 方法来安排小程序的组件。我们不需要查看 applet 类中所有的代码来理解布局管理，因此，我们不会显示整个小程序代码，而只是局限于数据变量和 init 方法。当然，你可以在小程序的生存周期中的任何时刻，向显示区域中自由地创建、安排和添加组件。关于小程序的生存周期，你将在第九章中了解到更详细的情况。

让我们用 FlowLayout（缺省的布局管理）来在显示区域中放一些组件吧！这里，我们会使用按钮，但其他组件都可以用这种方法来安排；按钮只是比较

简单的例子而已。

创建布局管理的第一步是：

```
FlowLayout flowingButtons=new FlowLayout();
```

这创建一个 FlowLayout 对象，它首先把组件放在显示区域的左边，直到超出范围。在这之后，它绕回到下一行，并在该行的左边开始放置组件。你也可以用下面的一行代码来指定一个基准：

```
FlowLayout flowingButtons=new FlowLayout(FlowLayout.CENTER);
```

这样，组件的左边和右边的空格一样多。基准选项有 LEFT、RIGHT 和 CENTER。

如果你不喜欢组件太紧凑，可以指定组件间的像素数：

```
FlowLayout flowingButtons=new FlowLayout(FlowLayout.CENTER,10,15);
```

这使按钮水平相隔 10 个像素，垂直每行相隔 15 个像素。

对于组件，简单地创建布局管理不会对小程序有许多影响，因此，我们必须调用小程序的 setLayout 方法，来把布局应用到显示区域中。下面就是这些代码：

```
setLayout(flowingButtons);
```

一旦我们把布局管理应用到小程序上，就可以添加组件了。下面是创建几个按钮，建立一个布局管理，并在 init 方法中添加按钮的一个小程序程序段：

```
Button ok = new Button(″ OK″ );
Button cancel = new Button(″ Cancel″ );
```

```
 Button apply = new Button(″ Apply″ );
 Button options = new Button(″ Options... ″ );
 Button properties = new Button(″ Properties″ );
 Public void init ()
 {
FlowLayout flowingButtons = new
 FlowLayout(FlowLayout.CENTER, 10, 15);
SetLayout(flowingButtons);
Add(ok);
Add(cancel);
Add(apply);
Add(options);
Add(properties);
 }
```

图 6-15 展示了这些按钮的布局管理。

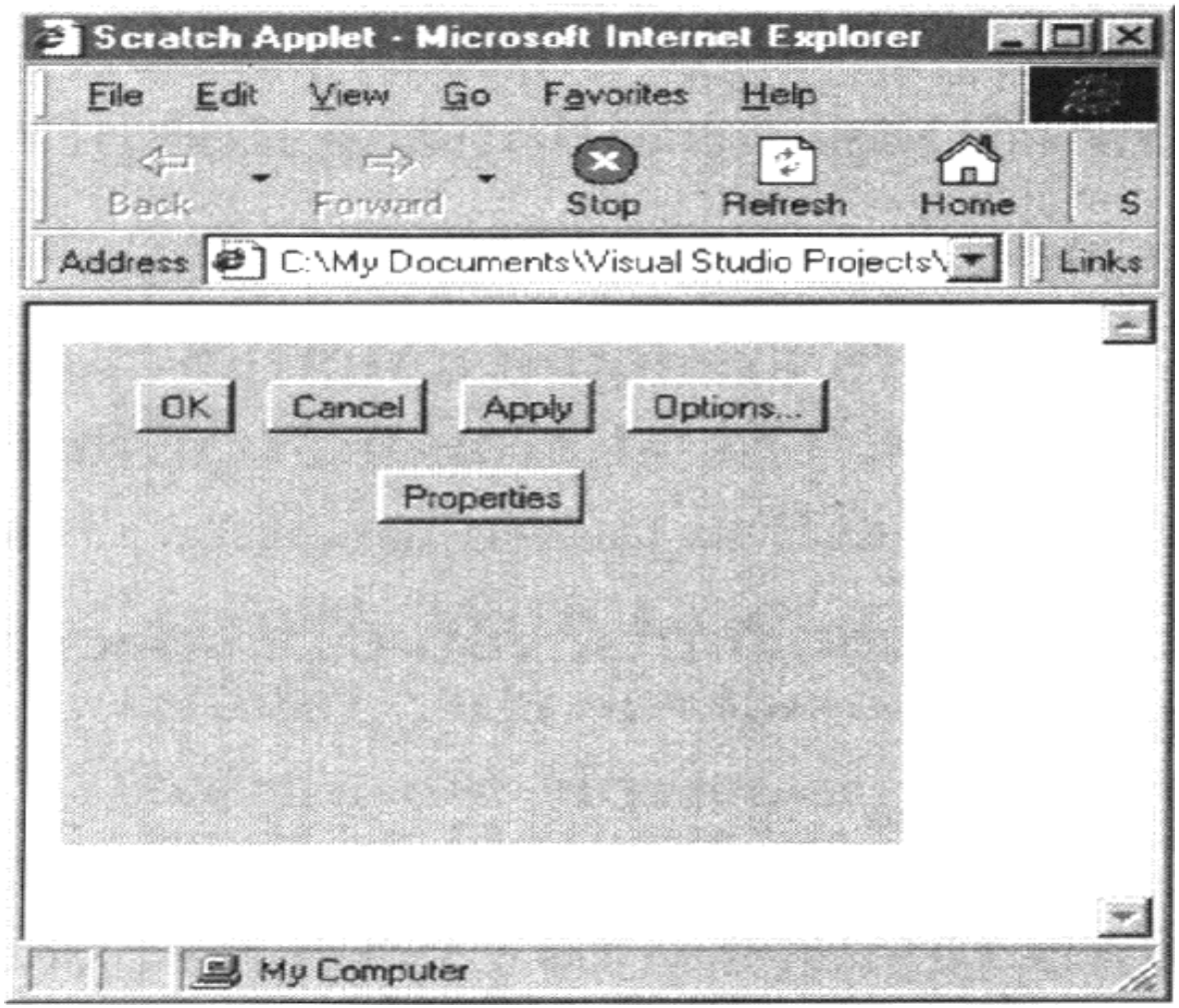


图 6-15　使用 FlowLayout 布局管理器显示的按钮

注意：查看显示中的按钮“流”的另一个方法是调出项目的 Properties 对话框(在 Project 菜单下),并单击 Launch 选项卡。在标签为 When Project Loads, Run 的下拉列表中选择你的 applet 类。现在，当运行项目时，小程序代码不用 Internet Explorer 来运行；相反，小程序运行在 Visual J++的小程序浏览器

中。小程序浏览器窗口可以很容易地调整窗口大小，来反映不同窗口大小的小程序的样子。

下面是用 GridLayout 的同一套按钮：

```
public void init()
{
    GridLayout gridButtons=new GridLayout(0,3,10,15);
```

GridLayout 对象构造器中的前两个参数是网格的行数和列数。这里行数为 0，因为 GridLayout 允许只指定行数或列数。本例中，因为我们在三列的网格中添加五个按钮，网格自动变成两行。如果你试图同时指定行数和列数，列数被忽略。后两个参数分别指定了列间和行间的像素数。

```
setLayout(gridButtons);
```

上面的程序行为前面的小程序设置了布局。然而，可以照常添加按钮（或其他组件）：

```
    add(ok);
add(cancel);
add(apply);
add(options);
add(properties);
}
```

图 6-16 展示了这个布局的图形。

BorderLayout 的作用就像一个地图。为了为小程序创建 BorderLayout 布局管理，必须用一个额外的参数值（North 代表显示区域的顶部，East 代表右边，

West 代表左边，South 代表底部，而 Center 代表中间其他位置）来添加按钮。下面是一个例子：

```
Public void init()
{
BorderLayout borderButtons = new BorderLayout(10,15);
SetLayout(borderButtons);
Add(″ North″ ,ok);
Add(″ East″ , cancel);
Add(″ West″ ,apply);
Add(″ South″ ,options);
Add(″ Center″ ,properties);
}
```

上面的代码创建了水平列间间隔为 10 个像素、垂直行间间隔为 15 个像素的 BorderLayout 对象。

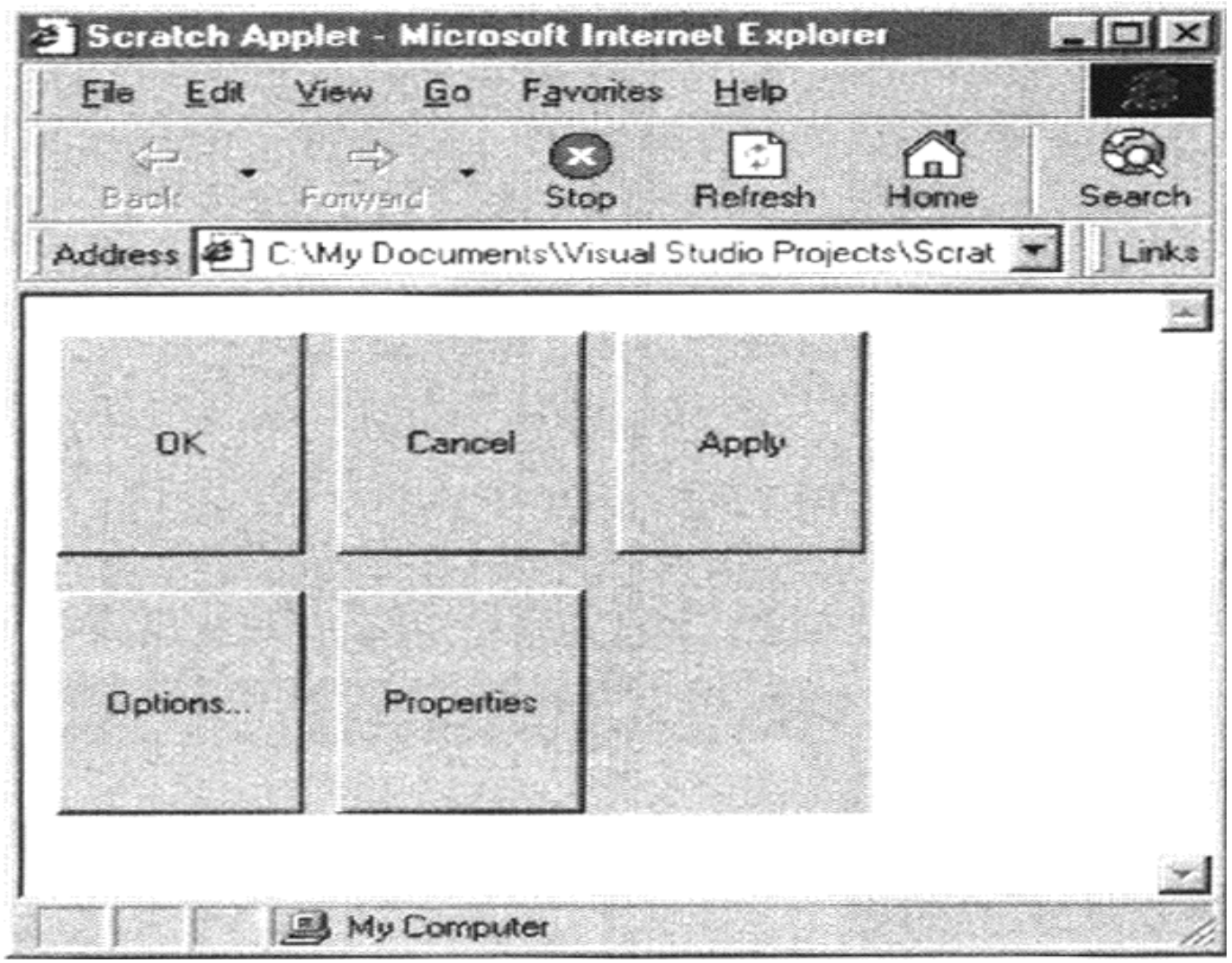


图 6-16　使用 GridLayout 布局管理器

add 方法是 BorderLayout 不同于前两个布局管理的地方。指定方向参数告诉布局管理把按钮放在什么地方。

图 6-17 展示了这个布局的图形。

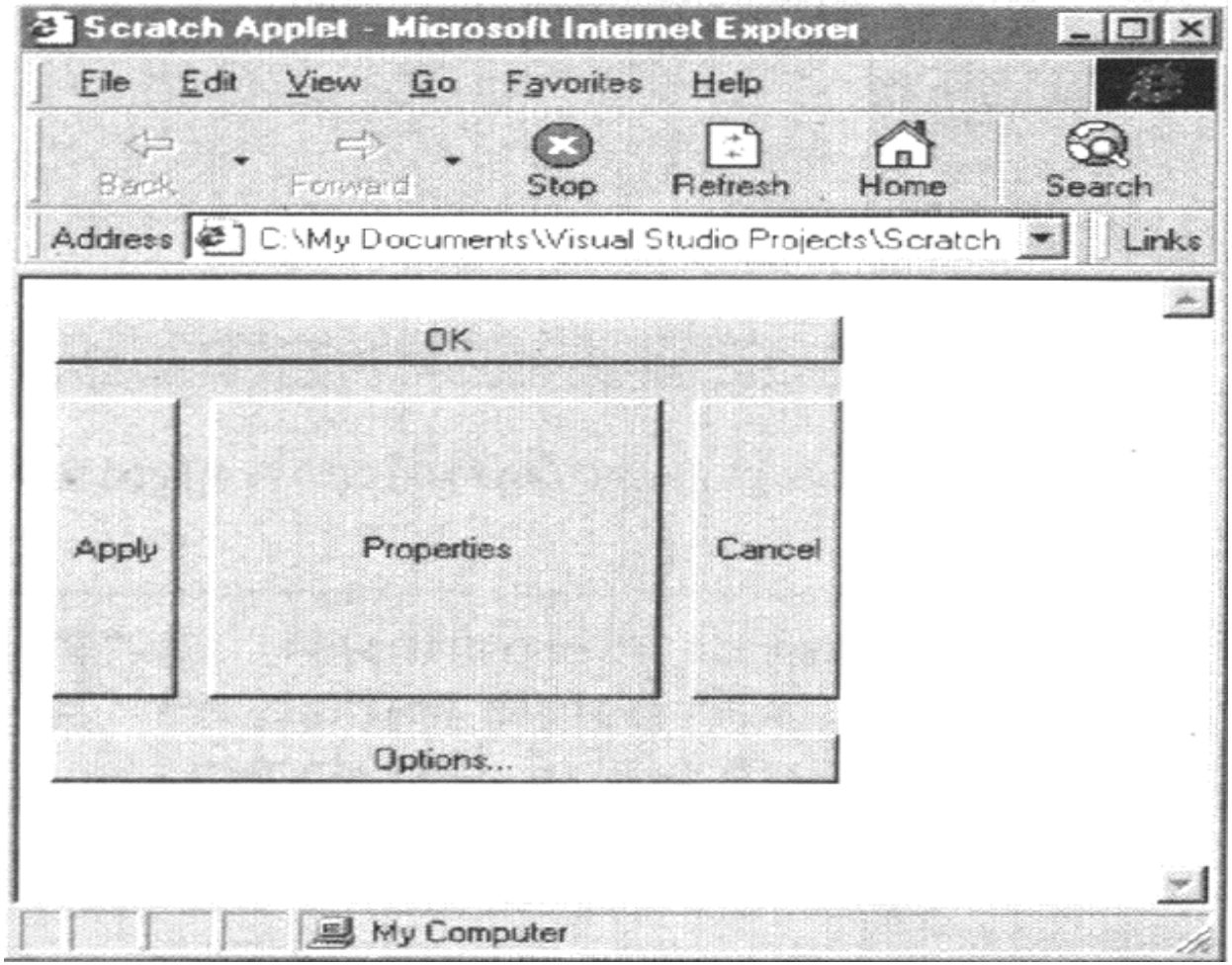


图 6-17　使用 BorderLayout 布局管理器安排按钮

关于 BorderLayout 需要注意的几点是：如果不指定方向参数，组件根本就不会显示在显示区域内；而如果把一个方向参数指定给两个组件，则后添加的组件将完全超越前一个。

在我们讨论 CardLayout 布局管理之前，首先来讨论一下面板。Panel 类包

含了许多 AWT 组件，就像 WFC 的 Form 类包含 WFC 控件一样。实际上，applet 类能包含组件的原因是 applet 类是 Panel 类的子类。

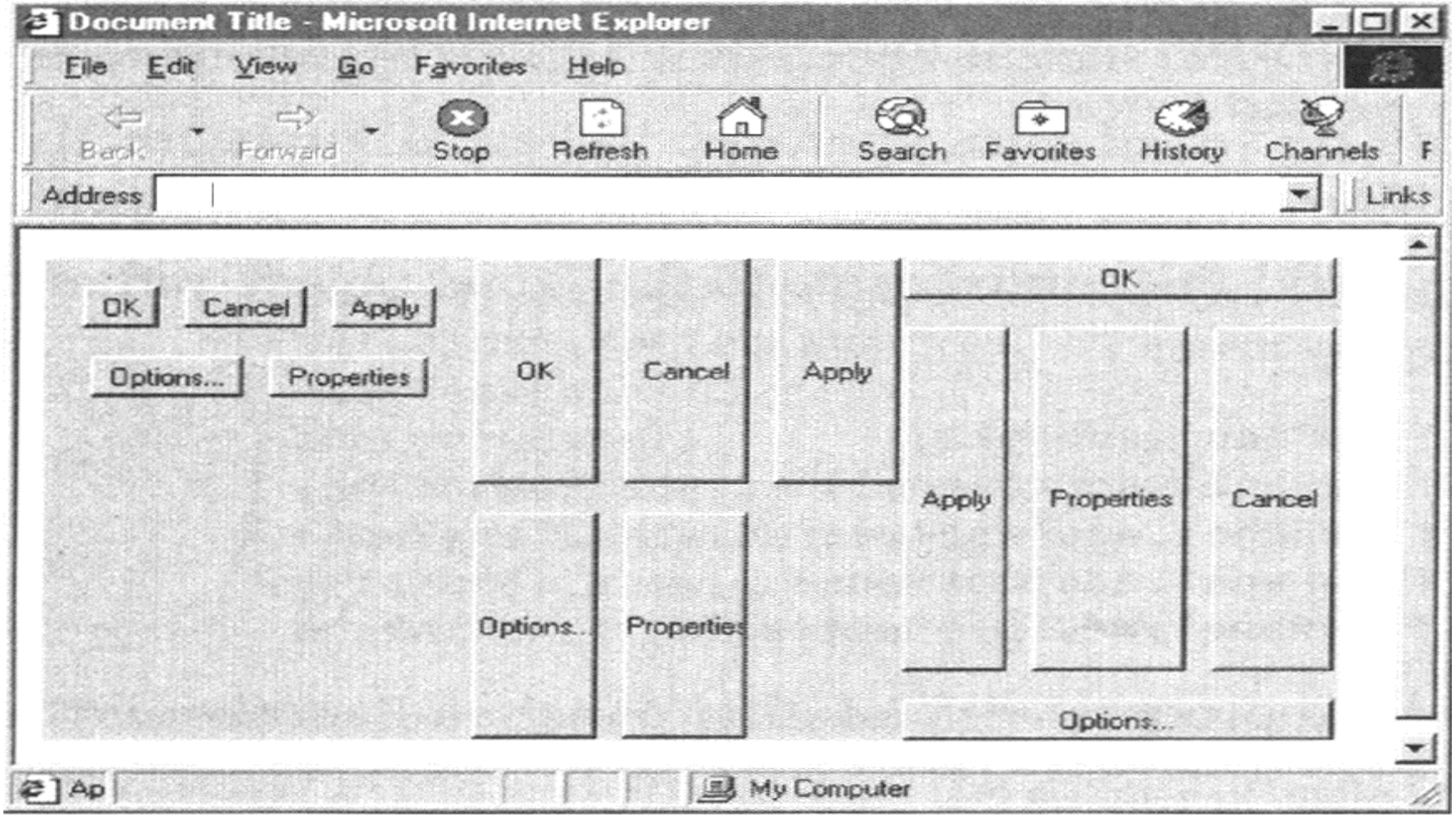


图 6-18　使用不同的布局管理器显示三个面板的小程序

为了对组件在小程序显示区域中的布置有更多的控制，我们可以在面板中组合组件。如果我们把后三个布局管理的例子同时放在显示区域中，但每个放

在不同的面板中，显示区域看起来就像图 6-18 所示的样子。

毫无疑问，这个设计非常杂乱，但这演示了如何在一个显示区域中用多个布局管理来安排组件。记住我们在例子中用的是按钮，但你可以在面板上放置任何组件。你可能会用一个充满了单选按钮和复选框的面板来作为 Option 对话框的一部分。

下一例子展示了如何用三个面板在小程序显示区域中安排组件。注意，你不能把一个组件同时添加到一个以上的面板中，第二个和第三个面板需要另外两套按钮。在这个例子中，将假设我们已经定义了一套原始按钮，并在它们名字的后面加一个数字来区别不同的面板组件（例如，ok、ok2 和 ok3）。

```
public void init()
{
```

首先，我们创建一个新的 Panel 对象：

```
Panel flowPanel=new Panel();
```

然后是一个新的布局管理：

```
FlowLayout flowingButtons=new FlowLayout(FlowLayout.CENTER,10,15);
```

下一步，我们把布局管理应用到面板上，而不是像以前那样应用到整个小程序上：

```
flowPanel.setLayout(flowingButtons);
```

最后，我们向 Panel 对象而不是直接向显示区域中添加组件：

```
flowPanel.add(ok);
flowPanel.add(cancel);
flowPanel.add(apply);
```

```
flowPanel.add(options);
flowPanel.add(properties);
```

我们用另一个面板和布局管理器来重复上述过程：

```
Panel gridPanel = new Panel ();
GridLayout gridButtons = new GridLayout(0,3,10,15);
GridPanel.setLayout(gridButtons);
GridPanel.add(ok2);
GridPanel.add(cancel2);
GridPanel.add(apply2);
GridPanel.add(options2);
GridPanel.add(properties2);
```

然后又是一个：

```
Panel gridPanel = new Panel ();
BorderLayout borderButtons = new BorderLayout (10,15);
GridPanel.setLayout (borderButtons);
GridPanel.add ("North",ok3);
GridPanel. add ("East",cancel3;
GridPanel. add ("West",apply3;
GridPanel. add ("South",options3);
GridPanel. add ("Center",properties3;
```

最后一步是为小程序的整体创建一个布局，然后向显示区域中添加面板：

```
        GridLayout appletLayout = new GridLayout(0,3);
        setLayout(appletLayout);
        add(flowPanel);
        add(gridPanel);
        add(borderPanel);
    }
```

这把我们带回到最后一个布局管理器--CardLayout。如果你在布局中不想一次看到所有的面板，可以用 CardLayout 来布置 Panel 对象，以使每次只有一个面板可见。

如果用下面的代码行替代上面例子中的最后五行代码，那么每个面板都将位于 CardLayout 对象的一张卡片上：

```
CardLayout appletLayout = new CardLayout();
setLayout(appletLayout);
add(flowPanel);
add(gridPanel);
add(borderPanel);
flowPanel.show(false);
gridPanel.show(true);
borderPanel.show(false);
```

这个例子在显示 gridPanel 布局时隐藏了 flowPanel 和 borderPanel。在别的地方被超越的 action 方法将把 flowPanel 布局或 borderPanel 布局调到前面而隐藏 gridPanel。

实验 6-2：作为小程序来再次访问 Hello

本实验中，我们将创建一个与在第二章中创建的 Hello 应用程序功能一样的小程序。

实验概述

你将练习下列内容：

- 用 AWT 组件创建小程序。
- 把小程序嵌入到 HTML 文档。

本实验中的小程序允许你在文本域中键入一个名字，然后单击 OK 按钮，来看看在小程序显示区域顶部的问候信息的改变。

实验设置

1. 启动 Visual J++并创建一个名为 Hello 小程序的空项目。
2. 向项目中添加名为 Hello 小程序的 Java 类（文件名为 Hello 小程序.java）。
3. 向项目中添加名为 Hello 小程序的 Web 网页（文件名为 Hello 小程序.htm）和带 HEIGHT 和 WIDTH 属性的<APPLET>标记。
4. 运行项目代码来确定设置是否正确。如果是，一个 200 个像素宽、200 个像素高的区域（不论你把高度和宽度指定为多少）将出现在 Internet Explorer 窗口中。你可以关闭这个窗口回到 Visual J++环境中继续编写你的小程序。

实验步骤

1. 在小程序中为文本域、按钮和标签添加成员变量。
2. 在小程序中超越 init 方法来为这些引用创建对象。用 FlowLayout 把组件布置在小程序显示区域中。在 init 方法的最后调用 repaint 方法。
3. 超越小程序的 action 方法。如果引发 action 方法的事件目标是 OK 按钮，把 Label 对象的文本设置成 Hello 加上 TextField 对象的文本。
4. 运行项目，在文本域中键入你的名字并单击 OK 按钮。你可以对照 Chapter06\Sol6-2\Hello 小程序.sln 下的解决方案来检查自己的工作。

下一步

我们现在能建立基本的小程序并用 AWT 组件来传播它了。下一步，我们将向小程序中添加图像和声音。

第七章　增强小程序

现在，我们能够用几个 AWT 组件来创建基本的小程序了，下一步就是添加多媒体功能，Java 正是因为它的这一功能而在 Internet 上出名的。我们从图像和声音开始，然后转向从 HTML 文件中将参数读入 Java 小程序中。最后，我们将说明如何向 Web 网页中添加 JScript 代码来激活网页。

本章中，你将了解到下列内容：

- 在小程序中播放声音和显示图像。
- 用 HTML 的<PARAM>标记来改变小程序的特征。
- 用 Jscript 来同小程序进行通信。

你将有机会：

- 播放声音文件。
- 显示图像文件。
- 通过 HTML 文档中的<PARAM>标记同小程序进行通信。
- 用 Jscript 来定制小程序的行为。

在小程序中使用多媒体文件

Java 小程序可以装备图像和声音。我们需要做的只是填空而已。本章中，我们将看到如何在小程序中放置图像和声音。后面，我们将在这个基础上建立提供动画的小程序。

在小程序中显示图像

为了在小程序中显示图像，我们首先创建一个 Image 类型的成员变量。然后，当我们需要把图形调入到程序中时，可以调用 applet 类的 getImage 方法。最后，为了显示图像，我们调用 Graphics 类的 drawImage 方法。通常，getImage 用在我们超越的 init 方法中，而 drawImage 是通过对 paint 方法的超越来调用的。下面是显示一个 Image 对象的 applet 类的一个例子。

我们所需要的资源在 java.applet 软件包或 java.awt 软件包中，因此我们增加这两条 import 语句：

```
    import java.applet.*;
import java.awt.*;

    public class DisplayImage extends 小程序
{
```

成员变量 balloons 用来访问我们的 Image 对象：

```
  private Image balloons;
  public void init()
```

```
  {
```

对小程序进行初始化时，我们调用 getImage 来获得对将要显示在屏幕上的 Image 对象的引用：

```
  balloons=getImage(getCodeBase(),"Balloons.jpg");
  }
```

最后，为了显示图像，我们调用 Graphics 类的 drawImage 方法：

```
public void paint(Graphics g)
  {
    g.drawImage(balloons,0,0,this);
  }
}
```

仔细观察对 getImage 方法的调用，我们看到它有两个参数：

```
balloons=getImage(getCodeBase(),"Balloons.jpg");
```

第一个参数指定了图形文件的位置。为了获得这个信息，我们调用 applet 类的另一个方法：getCodeBase。getCodeBase 方法的返回值是同小程序相关联的 HTML 文件中提供的属性 CODEBASE 的值。这个值是支持小程序运行的文件所在的文件夹名字。如果 CODEBASE 属性没有被包括在 HTML 文件中，getCodeBase 的返回值将是小程序字节代码文件（文件扩展名为.class）所在的文件夹名。看看下面这个 HTML 文件中的<APPLET>标记的例子：

```
<APPLET
```

```
CODE = Applet1.class
CODEBASE=″ FILE:// C:\My Documents\Visual Studio Projects\images″
NAME=Applet1
WIDTH=320
HEIGHT=200>
```

因为这个 <APPLET> 标记的 CODEBASE 属性是 file://C:\My Documents\Visual Studio Projects\images，getCodeBase 方法将返回这个路径名。

在 DisplayImage 例子中，我们访问的类文件与 Java 源代码和 Web 文件在同一个地方，因此，我们在 HTML 文件中没有使用 CODEBASE 属性。因而，调用 getCodeBase 返回小程序所在文件夹。使用 getCodeBase 的优点是，如果你把文件移动到另一个文件夹中，只要它们都在一起，而且你包括或改变 <APPLET>的 CODEBASE 属性值。getImage 调用的第二个参数是载入的图形文件的名字。

这里，我们载入一个 JPEG 图形（文件扩展名为.jpg），但我们也可以载入一个 GIF 图形（文件扩展名为.gif）。

然后，在 paint 方法中用 Graphics 方法 drawImage 来实现这个图形：

```
g.drawImage(balloons,0,0,this);
```

drawImage 方法被重载，因此可以有多种方式调用它。这里，我们用的 drawImage 方法使用了 Image 引用变量，后面跟着显示区域中图形的左上角的坐标，最后是图形的宿主名。本例中，我们通过指定关键字 this 来把图形直接放在小程序显示区域中。

在小程序中播放声音

从小程序中播放声音有三个步骤。首先，我们创建一个 AudioClip 引用变量。AudioClip 位于 java.applet 软件包中。其次，我们调用 getAudioClip 方法来载入音频文件。再次，当我们确实已经准备好播放音频文件时，我们调用 AudioClip 对象的 play 方法。

我们将在显示区域中用一个按钮来从小程序内部激活 AudioClip 对象。下面告诉我们如何做：

```
    import java.applet.*;
import java.awt.*;

    public class PlaySound extends Applet
{
```

下面是 AudioClip 成员变量：

```
private AudioClip drip;
private Button playDrip;
public void init()
{
```

下面是我们把 AudioClip 成员变量同音频文件关联在一起的地方。注意，getAudioClip 方法同我们前面讨论过的 getImage 方法一样，也有一个参数来告诉方法到哪里去找音频文件。第二个参数是音频文件的名字：

```
        drip = getAudioClip(getCodeBase().″ drip.au″ );
```

```
            playDrip = new Button(″ Drip″ );
            add (playDrip);
}
public boolean action (Event e, Object helper)
{
     if (e.target == playDrip)
     {
```

action 方法是我们用 play 方法激活 AudioClip 对象的地方：

```
            drip.play()
      }
      return super.action(e,helper);
  }
            }
```

能从小程序中播放的音频文件是声音片段文件（文件扩展名为.au）。

给小程序传递参数

通常，一个好的小程序的关键是灵活性，而灵活性是通过<PARAM>标记提供的。通过允许别人把我们的 Java 小程序嵌入到他们的 HTML 文档中来定制小程序代码的一些行为或特征，我们可以使程序可以由 WWW 上更多的人使用，而且在更多的情况下可用。

<PARAM>标记

把内部信息送入小程序的机制来自两个部分：<PARAM>及它的相关属性和getParameter方法。标记出现在HTML文档中，而getParameter方法通常是在小程序的init方法中被调用。

让我们用<PARAM>和getParameter来构造一个不同的Hello小程序。我们首先在Visual J++中创建一个空项目。然后，我们添加一个Web网页，并把<!--Insert HTML here-->用下面的内容来代替：

```
<APPLET CODE=HelloName.class WIDTH=200 HEIGHT=200>
<PARAM NAME="who" VALUE="Java 小程序 Master">
</APPLET>
```

正如我们在第六章中了解到的那样，<PARAM>标记同<APPLET>标记关系密切。<PARAM>的两个属性是NAME和VALUE。为使小程序能使用相关值，小程序代码必须知道参数名（即NAME属性的值）。虽然参数名（例如"who"）是在小程序代码中预先决定好的，但VALUE可以是任何字符串。作为小程序程序员，应该用小程序给别人提供一个可用参数列表。

把参数读入到小程序中

我们的下一步是在小程序中编写代码，来从HTML文件中读取参数值。下面就是这样的小程序代码。开头总是同其他的小程序一样：

```
//HelloName.java
import java.applet.*;
```

```
import java.awt.*;

    public class HelloName extends Applet
{
```

我们声明了一个 String 成员变量，来保存从<PARAM>标记中读取的值：

```
  private String who;
```

然后就是初始化程序--init:

```
  public void init()
  {
```

我们用一个 String 参数来调用 getParameter 方法，这个参数的值同 HTML 文件中相应的<PARAM>标记的 NAME 属性值相匹配。该方法返回与 NAME 相关的 VALUE 属性的值。本例中，因为我们给 NAME 值为 who，它返回字符串 "Java Applet Master"，并把该值赋给成员变量 who:

```
  who=getParameter("who");
```

最好是检验是否返回了值，并为没有返回值的情况下提供一个值：

```
            if(who==null)
            {
                who="World";
            }
   }
```

如果成员变量 who 等于 null，要么是<PARAM>标记没有被包括在 HTML

文档中，要么是小程序可能是从一个没有解释 HTML 代码的小程序浏览器中执行的。任何情况下，如果 who 等于 null，我们希望提供一个缺省值。

```
public void paint(Graphics g)
{
   super.paint(g);
```

剩下的就是构造显示在小程序中的文本字符串，并用 drawString 方法来显示它：

```
   g.drawString("Hello"+who+"!",50,50);
    }
}
```

如果你想看看没有包括<PARAM>标记时运行小程序的效果，通过在<PARAM>标记的前后分别放置<!--和-->来把它注释掉。如果已经把它正确地注释掉，这中间的文本应该变灰。重新运行小程序，来看看发生了什么事情。

下面是另一个例子，它把小程序显示区域的背景颜色改成红色。我们再次从需要的 HTML 标记<APPLET>和<PARAM>开始：

```
<APPLET code=RedApplet.class HEIGHT=200 WIDTH=200>
<PARAM NAME="background color" VALUE="Red">
</APPLET>
```

小程序代码如下：

```
import java.applet.*;
```

```
import java.awt.*;

    public class RedApplet extends Applet
{
public void init ()
{
```

我们再一次用 NAME 属性值调用 getParameter 方法来获得 VALUE 属性的内容：

```
String backgroundParameter=getParameter("background color");
```

然后我们可以检测 getParameter 的返回值，并采取相应的动作：

```
if (backgroundParameter.equals(″ Red″ ))
{
    setBackground(Color.red);
}
else if (backgroundParameter.equals(″ Blue″ ))
{
    setBackground (Color.blue);
}
}
}
```

试着通过把<PARAM>标记的 NAME 属性改成"Blue"（注意大小写）来把背景颜色从红色改成蓝色。

实验 7-1：使用图像、声音和<PARAM>标记

我们已经看到如何显示图像、播放声音和用 HTML 标记<PARAM>来定制小程序。本实验中，我们把这些放在一起。

实验概述

本实验中，你将练习下列内容：

- 载入并显示 Image 对象。
- 载入并播放 AudioClip 对象。
- 在 HTML 文件中嵌入<PARAM>标记。
- 从 Java 代码的内部读取<PARAM>标记的值。

本实验的小程序代码将显示一个 Image 对象，并播放一个 AudioClip 对象。它将根据两个 HTML <PARAM>标记的值，来决定显示哪个图形文件和播放哪个声音片段文件。

实验设置

1. 从 Chapter07\Lab7-1 下把*.au 和*.jpg 文件复制到项目目录下。
2. 启动 Visual J++并创建一个名为 AudioImage 的空项目。
3. 为 Java 小程序创建一个类文件，并为 Web 网页创建一个 HTML 文件。

实验步骤

1. 在小程序代码文件中分别创建 Image 类型、AudioClip 类型和 Button 类型的成员变量。
2. 超越 init 方法，并创建局部变量，来存储图形文件和声音片段文件的 String 值。
3. 提供 String 局部变量，并把两次调用 getParameter 方法的返回值（一次为获得 Image 参数，一次为获得 Audio 参数）赋给它们。
4. 检验每个 String 局部变量，看它们是否为 null。如果有一个为 null，就给它赋一个缺省的文件名。在项目目录下放置两个图形文件和两个声音片段文件，来测试 getParameter 的返回值。
5. 创建一个 Button 对象，并把它的引用赋给 Button 成员变量。
6. 用 add 方法向小程序显示区域中添加这个按钮。
7. 用 drawImage 超越 paint 方法来显示 Image 对象。
8. 超越 action 方法使按钮被按下时播放声音片段文件。
9. 在项目的 Web 网页中包括下列 HTML 代码（这些代码应放在<BODY>和</BODY>之间）：

```
<APPLET CODE=AudioImage.class WIDTH=200 HEIGHT=200>
    <PARAM NAME="Audio" VALUE="drip.au">
    <PARAM NAME="Image" VALUE="Balloons.jpg">
</APPLET>
```

10. 运行项目。你应该在小程序显示区域中可以看到图像。单击按钮应该

可以听见播放的声音。你可以把程序同位于 Chapter07\Sol7-1\AudioImage.sln 下的解决方案作个比较。

11. 从 HTML 文件中删除掉两个<PARAM>标记。你的小程序有何不同？

下一步

我们已经看到，小程序如何被嵌入到 Web 网页中，而且我们已经用<PARAM>标记来从 HTML 中同小程序进行通信。小程序在 Web 网页中的这些用法通常是很被动的。我们的下一步是用 JScript 在 Web 网页中编写程序代码。

小程序和 Web 网页

WWW 原先是用来传播文化的机构。Web 网页可以说是能快速发行的电子版的文本，它很容易在世界各地传播。电子印刷有了迅猛的发展，但人们很快就渴望一种交互性更强的 Web 网页。交互式网页通过使用按钮和复选项之类的元素来提供一个更有效的界面，它们能动态做出响应，以适应用户的要求。

HTML 有很多优点，但它不是而且也从来没有被设计成一种编程语言。JavaScript 通过允许脚本或解释性的编程代码嵌入到 HTML 代码中，来扩展 HTML 的功能。

我们要用的 JavaScript 版本称为 JScript。JScript 代码能与小程序互相影响，来定制小程序的行为。这使我们可以在不同的 Web 网页中使用小程序代码和可改写的 Jscript，来获得不同的效果，而不需要重新编写代码和重新编译 Java 小

程序。

尽管 JScript 是设计用来填补被动的 HTML 和强有力的 Java 小程序之间的缺陷的，它自己还是一种编程语言。我们对 JScript 的讨论将只限于概述它的功能。我们要用 Visual J++来建立一个 Web 网页，用一个小程序和嵌入的 JScript 来演示这些基本元素是如何在一起工作的。

小程序与 Web 网页间的通信

建立同小程序相互影响的活动 Web 网页的第一步是创建一个小程序，它包含一个或更多能被 HTML 文档中的 JScript 函数访问的公有方法和公有成员变量。

注意：当编写 JScript 代码时，如果一个小程序变量被声明为 final，你应直接引用它，而用小程序的公有方法来访问其他非最终变量，这一点很重要。这会帮助你在 Web 网页上获得期望的效果。

我们在这一部分末尾的练习是用一个 HTML、一个小程序和 JScript 代码来完成一个 Web 网页。该网页显示要出售的房屋，及它们的简要说明，以供客户在 Internet 上查看。这个练习的小程序已经提供给你，其中包含了你在前面的讨论中已经熟悉了的 Java 代码。在开始这个练习之前，让我们来看看这些代码的关键部分，以使你能了解它们是如何同我们将要写的 Jscript 代码联系在一起的。

如果你想在讨论之前预览一下代码，可以在 House 项目中找到预备文件（Chapter07\House\HouseDisplay.sln）。在 Visual J++中打开项目，并双击 HouseDisplay.java。

```
public class HouseDisplay extends Applet
{
```

首先，我们声明三个 public 的成员变量。JScript 用这些常量来告诉 Java 小程序，我们想在 Web 网页上显示哪个房屋的图像。小程序代码中任何标有 public 的元素在 JScript 代码中都是可用的。剩下的一些声明是创建小程序的典型成员变量：

```
    public static final int house1 =1;
public static final int house2 =2;
public static final int house3 =3;

    private boolean showHouse = false;
private Font font = new Font(″ TimesRoman″ ,Font.BOLD, 14);
private Image house1Img;
private Image house2Img;
private Image house3Img;
private Image currentImg;
private String message;
```

我们要创建的 JScript 代码给 setDisplay 方法传递参数值，来告诉小程序应该显示哪三个图像。注意这个方法的末尾对 repaint 的调用，正是它使最近选择的图像显示出来：

```
    public void setDisplay (int houseNumber)
{
```

```
if (houseNumber == house1);
{
      currentImg = house1Img;
      showHouse = true;
}
else if (houseNumber == house2)
{
      currentImg = house2Img;
      showHouse = true;
}
else if (houseNumber == house3)
{
      currentImg = house3Img;
      showHouse = true;
}
this.repaint ();
       }
```

这个小程序也给我们提供来显示图像或是显示文本信息的选项。后面，我们将通过编写在小程序显示区域中显示图像，或显示文本的 JScript 代码，来利用这个选项。我们选择哪一个，将取决于哪一个更适合于小程序执行时的具体情况。下面是 JScript 代码用来设置显示的文本信息的方法：

```
        public void setMessage (String text)
    {
        message = text;
        showHouse = false;
        this.repaint ();
            }
```

小程序中被超越的 paint 方法显示文本或图像取决于 setDisplay 和 setMessage 的调用。注意下面的代码中 getSize 方法的用法，它带有 width 和 height 成员变量，它决定了 Web 网页上小程序的大小。getSize().width 的值使我们可以确定 HTML 文件中的<APPLET>标记的相关属性 WIDTH 的值。对 g.clearRect 的调用清除一个长方形，它从显示区域的左上角开始，遍及整个显示区域。通过用 x 和 y 坐标的偏移量来调用 Graphics 的方法 drawImage，我们可以把图形放在显示区域的中央：

```
        public void paint (Graphics g)
     {
    int imageWidth;
    int imageHeight;
    int xOffset;
    int yOffset;

    g.clearPect(0,0,
          this.getSize().width,
```

```
        this.getSize().height);
    if (showHouse)
    {
        imageWidth = currentImg.getWidth(this);
        imageHeight = currentImg.getHeight(this);
        xOffset = (this.getSize().width – imageWidth) /2;
        yOffset = (this.getSize().height – imageHeight) /2;
        g.drawImage(currentImg.xOffset,yOffset,this);
    }
    else
    {
        g.setFont(font);
        g.drawString(message,5,50);
    }
         }
```

下面是读取 Web 网页中<PARAM>标记的参数值的方法。如果 houseGif 等于 null，那么，这个值的<PARAM>标记就没有被包括在 Web 网页中；或者，如果 houseGif 是一个空字符串("")，那么<PARAM>标记被包括了，但在 VALUE 属性中没有指定图形文件。我们用一个缺省图形（"NoPicture.gif"）来处理这两种情况：

```
        private void getParameters()
```

```
{
    String houseGif;
    houseGif = this.getParameter(" house1 " );
        If (houseGif == null || houseGif.equals(" " ))
        {
            house1Img = getImage(getCodeBase()," NoPicture.gif " );
        }
        else
        {
            house1Img = getImage(getCodeBase(),houseGif);
        }
        houseGif = this.getParameter(" house2 " );
        if (houseGif == null || houseGif.equals(" " ))
        {
            house2Img = getImage(getCodeBase()," NoPicture.gif " );
        }
        else
        {
            house2Img = getImage(getCodeBase(),houseGif);
        }
        houseGif = this.getParameter(" house3 " );
        if (houseGif == null || houseGif.equals(" " ))
```

```
        {
            house3Img = getImage(getCodeBase(),″ NoPicture.gif″ );
        }
        else
        {
            house3Img = getImage(getCodeBase(),houseGif);
        }
        }
```

超越了的 init 方法调用了 getParameters。getParameters 方法根据<PARAM>的属性值载入相应的图形文件。然后，小程序初始化过程就完成了，它已准备好同用户进行交流：

```
public void init()
{
    super.init();
    getParameters();
}
     }
```

第一步已经完成了：我们已经有了一个拥有几个公有方法和公有变量的小程序，JScript 能用它们来定制小程序的运行方式。下一步我们为小程序、HTML 控件和 JScript 建立一个宿主 HTML Web 网页。

使用 HTML 控件

我们可以用与通过在窗体上放置 WFC 控件来建立 Windows 应用程序接口的方法一样的方式,来用 Visual J++建立 Web 网页上的控件.一旦我们在 Web 网页上有了控件,就可以为特殊事件添加时间处理，就像我们对待 Windows 应用程序一样.在 HTML Web 网页中,这些事件处理程序将用 JScript 来编写.

如果你还没有准备好,请打开预备的 House 项目（Chapter07\House\House-Display.sln）.向项目中添加一个 Web 网页，并把它命名为 HouseOfHouses.htm 这会调出一个空白的 HTML 代码窗口,其中 Design 选项卡已经被选中.在 HTML 文件窗口中键入下列文本:

Welcome to The City View House Company's "House of Houses" Homepage!
To read a description of a house for sale, just move the cursor over
One of the buttons. To see a picture of the house, click the button.

要查看这段文本在 HTML 代码中的样子，单击 Source 选项卡。Visual J++已经在文本周围自动添加了相应的 HTML 代码。再单击 Design 选项卡，我们将向 Web 网页中添加一些按钮。调出 Toolbox 窗口，并选择顶部标有 HTML 标签的下拉列表。图 7-1 显示了一个带有 HTML 工具的 Toolbox 窗口。

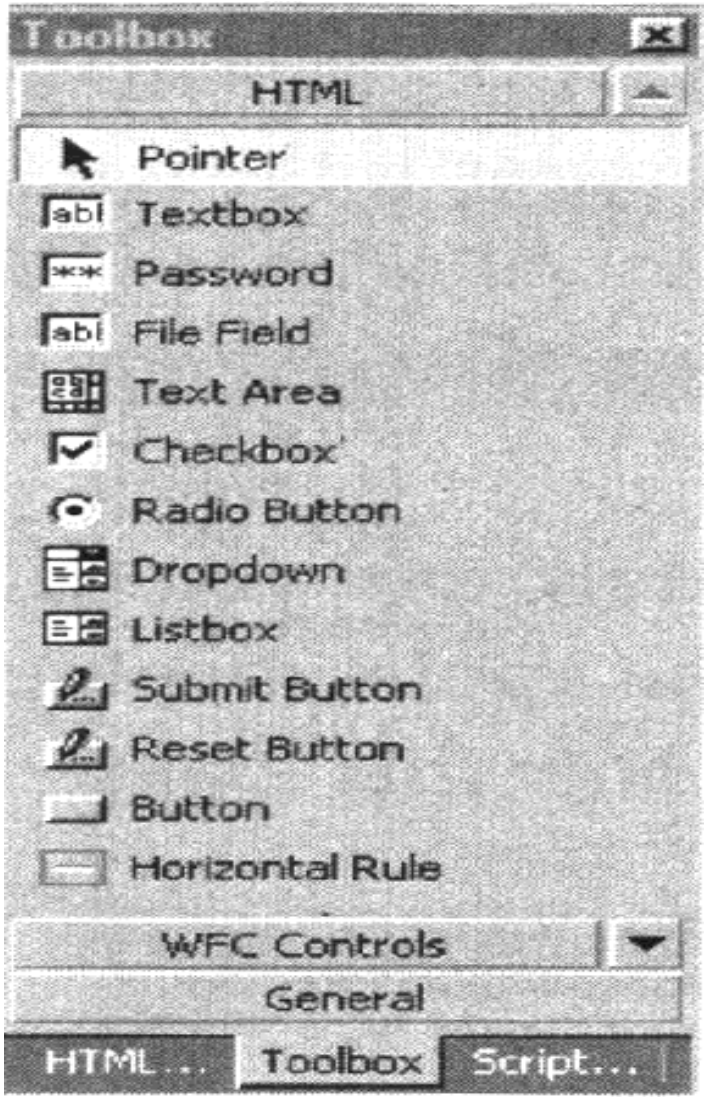


图 7-1　Toolbox 窗口的 HTML 组件

从 HTML Toolbox 中选择一个按钮组件，并把它拖到 HTML 网页中。把按钮放在你刚才键入的文本的下面。为了保证对齐，你可以在文本后添加一个回

车，或直接把鼠标放在按钮前，并双击 Line Break 组件（你可以通过单击 Source 选项卡来查看 HTML 代码中的不同之处）。添加一个空格（用空格键或在工具箱中双击 Space），然后在右边再添加另一个按钮。再在它们的右边添加第三个按钮。

如果用鼠标单击按钮，会看到你可以改变按钮的标题。把三个按钮标题分别改成 City View House、Cute 1 Bedroom 和 Lakeside Beauty。

单击 Source 选项卡，然后在第三个按钮的代码之后、HTML 主体结束标记（</BODY>）之前添加下列 HTML <APPLET>标记：

```
<APPLET code=HouseDisplay.class name=housedisplay
height=300 width=500 >
<PARAM NAME=″ house1″  VALUE=″ CityView.gif″ >
<PARAM NAME=″ house2″  VALUE=″ Cute1Bedroomgif″ >
<PARAM NAME=″ house3″  VALUE=″ LakeSide.gif″ >
Your Browser must be Jave-enabled to see City View’s Houses
</APPLET>
```

这些<APPLET>及相关的<PARAM>标记在 Web 网页上安装 HouseDisplay 小程序，并给它传送用户要求显示的三个 GIF 文件的名字。然后，这些文件名被 HouseDisplay 中的 getParameters 方法“解析”（读取并处理）。

单击 HTML 窗口的 Quick View 页，你会看到，Web 网页上已经为小程序显示区域保留了一个 500×300 的区域。图 7-2 展示了这样的一个窗口。

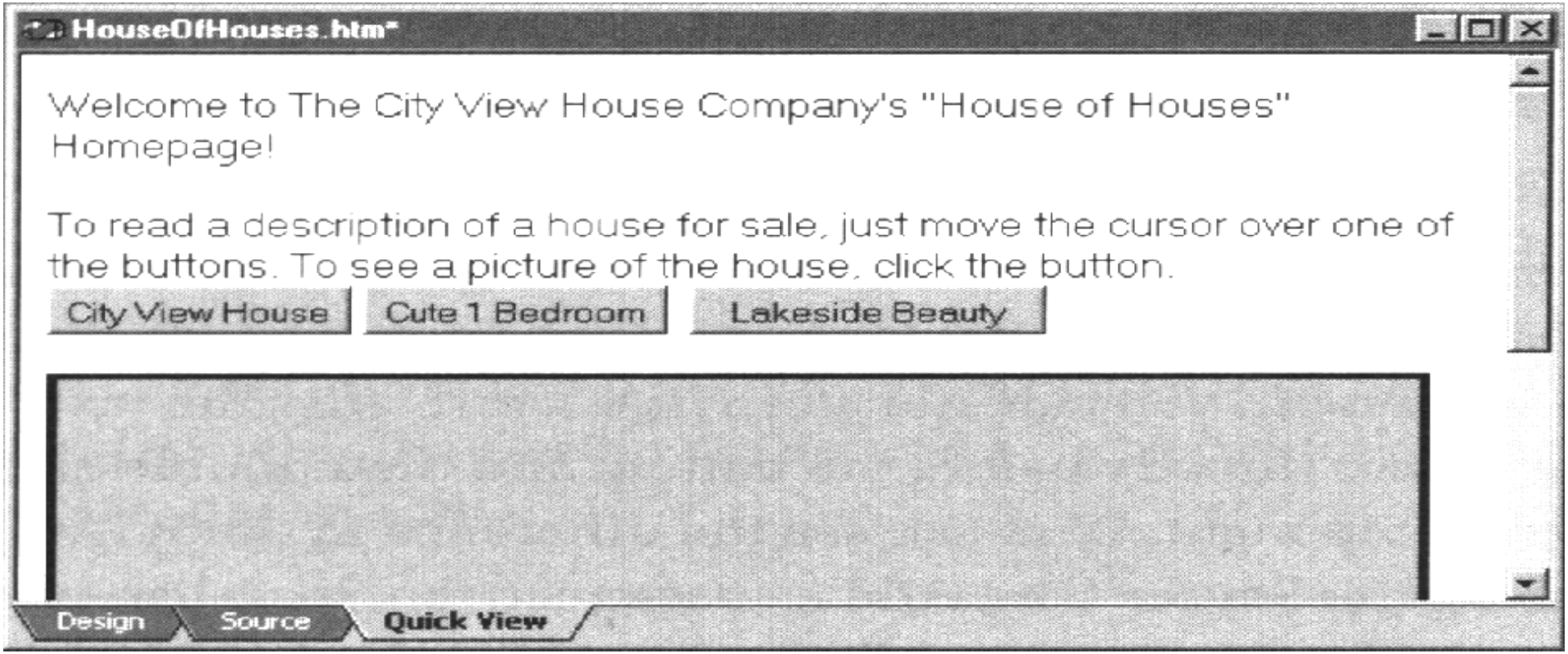


图 7-2　HouseOfHouse 的 HTML 页面

HTML Outline

早在第五章，我们曾经用 Class Outline 窗口来帮助我们查看应用程序的整体结构。类似的工具 HTML Outline 窗口可以用在 HTML 文件上。你可以通过从 View 菜单中选择 Other Windows，然后再选择 Document Outline 来得到这个窗口。当你在激活了一个 HTML 文件时，选择 Document Outline，Visual J++调出一个 HTML Outline 窗口，就像我们在第五章中看到的 Class Outline 窗口一样。

当选中了 Quick View 页时，HTML Outline 窗口显示下列信息：

There are no items to show for the selected document.

单击 Design 或 Source 选项卡，你会看到图 7-3 所示的 HTML Outline 窗口。

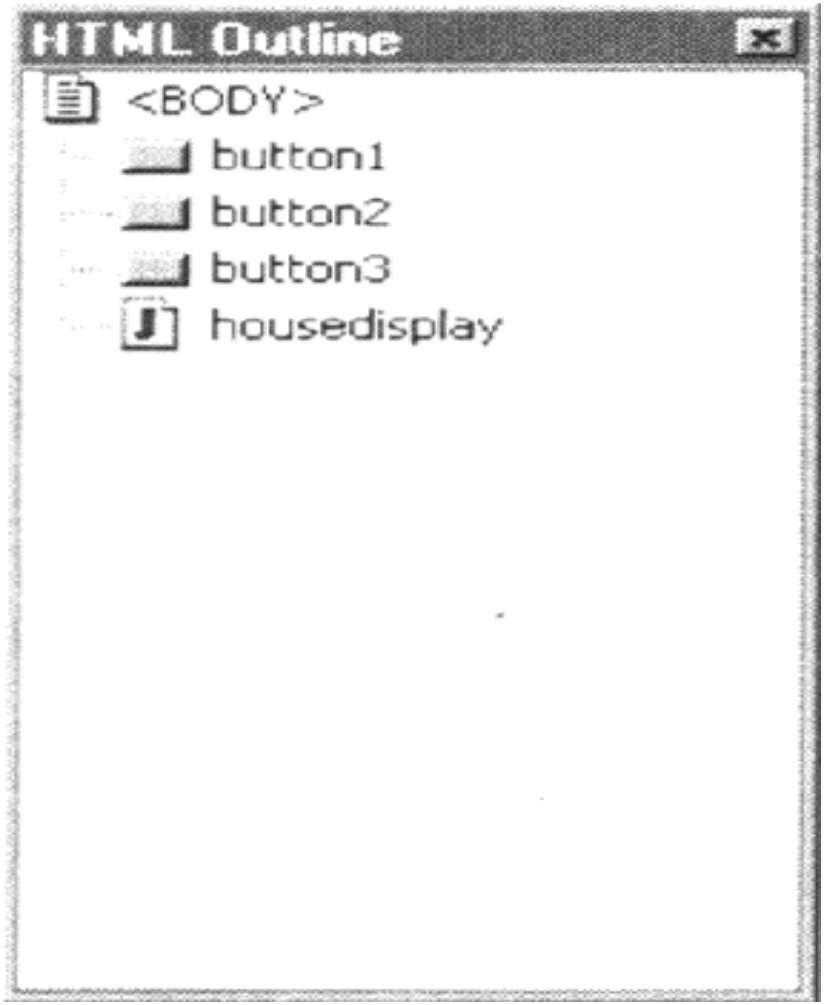


图 7-3　HTML Outline 窗口

在 HTML Outline 窗口中单击任何元素，你会在 HTML 代码窗口中看到，同样的元素也被选中。在 HTML Outline 窗口中选择 button1，然后单击 HTML 代码窗口的 Source 选项卡。注意，鼠标焦点放在 HTML 代码中定义 button1 的位置。单击 HTML Outline 窗口中的 button3，则光标移动到定义 button3 的位置。在 HTML Outline 窗口中单击 housedisplay，来查看原来键入的<APPLET>标记。如果在代码窗口中的小程序占位符上击右键，可以通过选择 Always View As Text 复选项来查看<APPLET>代码。清除掉 Always View As Text 选项将返回到占位符。

HTML 属性

就像我们用 Properties 窗口来设置 WFC 控件的属性一样，我们也可以用它来改变 HTML 组件的属性。在 HTML 代码窗口中，单击 Design 选项卡。在 HTML Outline 窗口中选择<BODY>。Properties 窗口的顶部，指出我们现在正在查看 DOCUMENT 的属性。如果 Properties 窗口不可见，你可以通过从 View 菜单中选择 Properties Window 来调出它。图 7-4 展示了 Properties 窗口中的 DOCUMENT 组件。

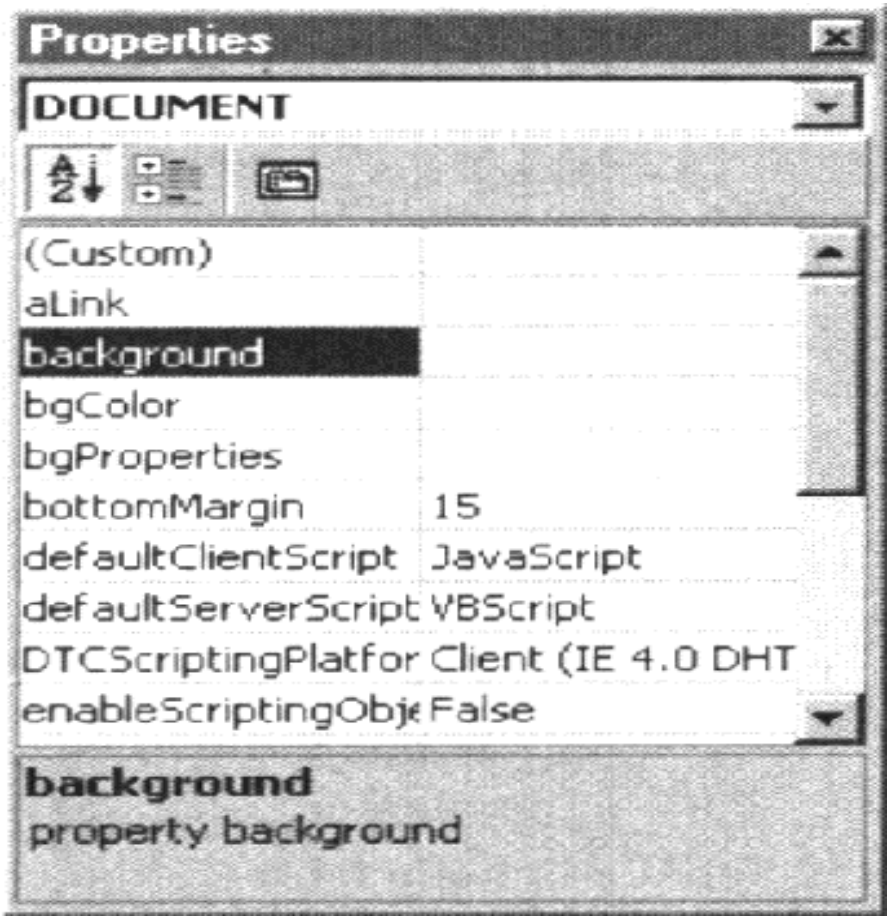


图 7-4　选择了 DOCUMENT 的 Properties 窗口

在 Properties 窗口中，找到背景属性，并把它的设置改成 Background.bmp。现在，小程序的背景将改成载入的这个文件。在 HTML Outline 窗口中，选择 button1。滚动 Properties 窗口到 backgroundColor 属性，然后从 Color 对话框中为 button1 选择一种颜色。对其他两个按钮重复相同的过程。从 Properties 窗口顶部的下拉列表中选择 DOCUMENT，把标题属性的设置改成 City View House

Company。如果查看 HTML 源代码，你会发现，<TITLE>标记反映了这个变化。现在它是下面这个样子：

```
<TITLE>City View House Company</TITLE>
```

为了给小程序添加一个好看的边框，在 HTML Outline 窗口或 HTML 代码窗口中选择小程序，然后到 Properties 窗口中（"<APPLET>"应该出现在顶部的下拉列表中），把 borderBottom、borderLeft、borderRight 和 borderTop 设置成 solid。

你可以通过在 HTML Properties 窗口中改变组件的(Id)属性，来改变它的标识。因为(Id)属性名包括圆括号，它出现在 HTML Properties 窗口列表的顶部。对组件(Id)属性的改动会反映在 HTML Outline 窗口中。在我们的例子中，我们希望简化些，因此，我们让 HTML 按钮组件的(Id)属性维持为缺省值 button1、button2 和 button3。

Script Outline

Script Outline 窗口在跟踪你正在编写或已经编写好的 JScript 代码时，确实是你最好的助手。我们将用这个工具来向 HTML 网页中插入 JScript 代码。它会帮助我们跟踪 HTML 网页中 JScript 代码的许多结构细节，从而让我们可以把精力集中在编写具体的代码上。

在 HTML 代码窗口中，选择 Source 选项卡，通过单击 Toolbox 窗口底部的 Script Outline 页，来打开 Script Outline 窗口。图 7-5 展示了我们的 HTML 文件的 Script Outline 窗口。

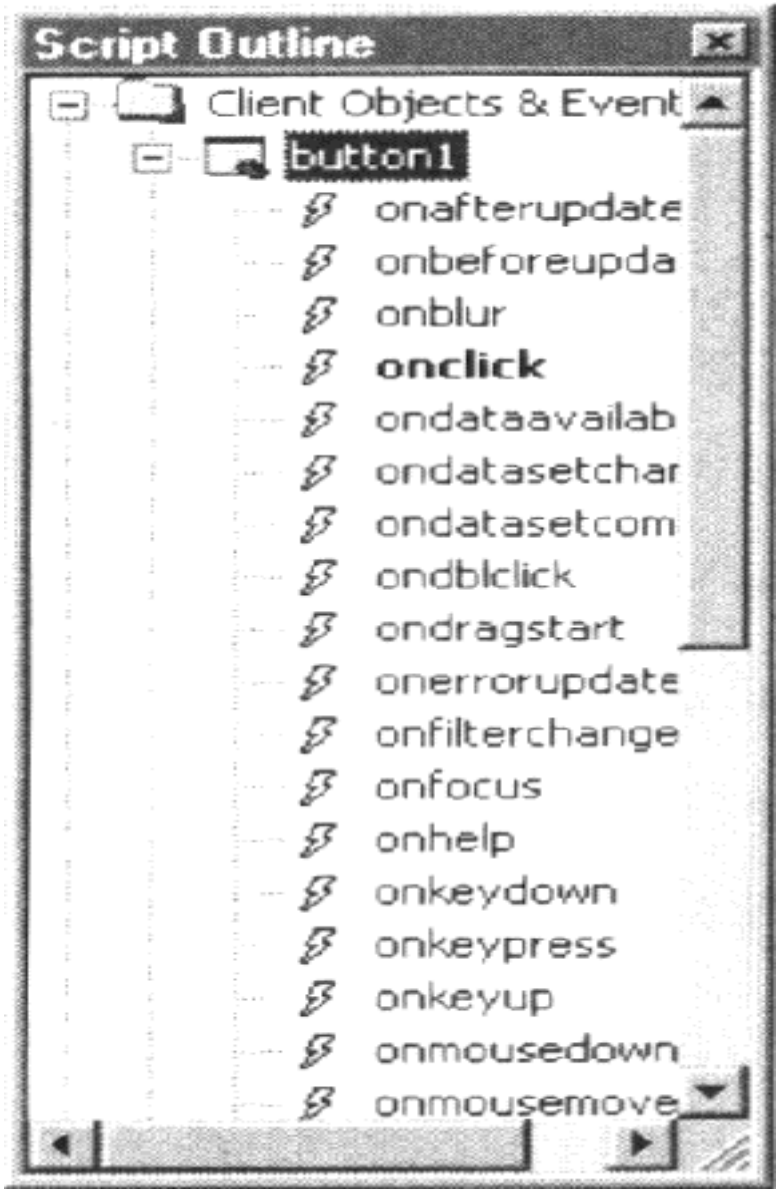


图 7-5　Script Outline 窗口

在 Script Outline 窗口中的四个文件夹中，我们只对与客户对象和脚本有关的两个感兴趣。“客户”是指将要下载我们的 Web 网页的计算机。Client Objects & Events 文件夹包含了我们已经添加到网页的所有 HTML 组件及 Document 和 Window 对象。Window 对象包含了 Document 对象，而 Document 对象则包含了我们放在网页上的所有对象。在我们的例子中，Document 对象包含了三个按钮和名为 HouseDisplay 的小程序。

Client Scripts 文件夹目前还是空的，因为我们还没有添加任何脚本。

HTML 对象在 Script Outline 窗口中显示的名字（本例中是 button1、button2 和 button3）来自 HTML Properties 窗口中的名称属性值。而名称属性的值又是来自 HTML 文件中赋给<BUTTON>标记的 NAME 属性的值。名称属性很容易同(Id)属性相混淆。(Id)属性值出现在 HTML Outline 窗口中，而名称属性值出现在 Script Outline 窗口中。

编写 Jscript

让我们给 button1 添加一个脚本，使这个按钮被单击时会在小程序显示区域中显示一个图形文件。首先，在 Script Outline 窗口中展开 button1 的列表。所有同 button1 相关的事件列表会显示出来。如下图 7-6 所示。

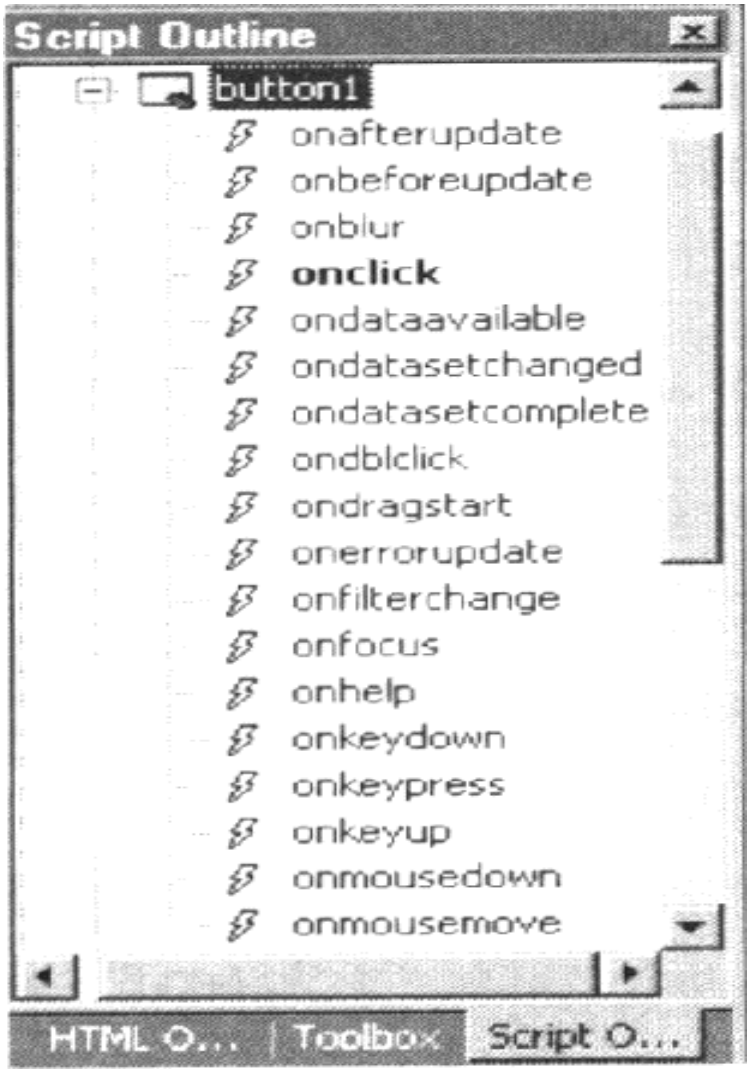


图 7-6　Script Outline 窗口为 HTML 组件列出事件

有了 Script Outline 窗口中显示的这个列表，我们就可以看到同 HTML 按钮相关的所有事件。这与 WFC 控件的相关事件相类似。一个 JScript 函数可以同这些事件的任何一个或全部相关联，甚至也可以一个也不关联。这样的一个

JScript 函数同 Java 中的事件处理方法相类似。

找到 Onclick 事件并双击它。这样做时，HTML 代码窗口发生变化，在窗口中显示下列代码：

```
    <SCRIPT ID=clientEventHandlersJS LANGUAGE=javascript>
<!—

function button1_onclick()
{

}

//→
    </SCRIPT>
```

你在这里看到的是<SCRIPT>标记，它被放在 HTML 文件中处理 button1 的单击事件的事件处理程序框架之后。注意赋给 LANGUAGE 属性的值是 javascript。记住，JScript 是 JavaScript 的一种变种。“<!—”开始一个 HTML 注释，而“-->”结束它。HTML 注释标记被放在文件中，用来隐藏 HTML 脚本代码，这适用于那些不支持脚本的旧版本的浏览器。脚本区域被</SCRIPT>标记关闭。

Visual J++为我们插入了周围的语法格式后，我们为处理这个事件要做的所有事情就是，在花括号中放置一个对小程序公有方法的调用。

然后，当 Web 网页的用户单击 button1 按钮时，这个 JScript 函数被调用。然而，我们首先必须确定基于 applet 类而创建的小程序对象的名字。我们看一看前面插入的<APPLET>标记，我们看到，NAME 属性被设置成 housedisplay。这是小程序的实例名（即类 HouseDisplay 的对象），它允许我们像访问其他对象一样访问小程序（注意，如果<APPLET>标记的 NAME 属性被删除，在 HTML Outline 窗口中，该元素将显示为"<APPLET>"）。由于你会从小程序代码中重新调用，这里有一个名为 setDisplay 的 public 方法。因为该方法是公有的，我们可以在 JScript 代码内部调用它。setDisplay 方法需要一个 int 类型的参数，它代表应该显示的房屋图像。我们可以用 1、2 或 3 来作为参数值，或者我们可以引用小程序中的 public static final 的整型成员变量。button1 的事件处理方法如下所示：

```
    function button1_onclick()
{
  housedisplay.setDisplay(housedisplay.house1)
}
```

在 Internet Explorer 中运行该项目。当 Web 网页出现时，单击标题为 City View House 的按钮，你会看到房屋图像被显示出来。关闭 Internet Explorer 回到 Visual J++环境。

既然我们已经运行了项目，这里，我们可以用 Quick View 来估计它在 Web 网页上的操作。从 HTML 代码窗口中单击 Quick View 页，然后单击 City View House 按钮，你会看到 City View House 图像再次显示在屏幕上。

除了能显示图像外，小程序还能显示信息。仍然是用 button1，让我们为

button1 的 Onmouseover 事件调用方法 setMessage。在 Script Outline 窗口中，展开 button1 文件夹，找到 onmouseover 并双击它。button1_mouseover 函数的框架出现在 HTML 网页的脚本区域中。添加一个对方法 setMessage 的调用，如下所示：

```
    function button_onmouseover()
{
  housedisplay.setMessage("Our flagship.Sweeping view of the City."
+ "Priced to move.")
}
```

单击 Quick View 页并测试 button1。现在当你把鼠标移动到按钮上时，它应该显示一条文本信息，而当你单击按钮时，它应该显示一个图像。好样的！现在你已经成功地用两个不同的 JScript 事件编好你的小程序了。还有更多的 JScript 和 HTML 我们没有讨论到，但这些都是用来联系 Java 和 JScript 的。

顺便提及，如果你想查看 HTML 代码中按钮和事件之间的联系，请查看 button1 的<INPUT>标记。你会看到下列属性和属性值被添加到这个标记中：

```
    LANGUAGE=javascript onclick="return button1_onclick()"
onmouseover="return button1_onmouseover()"
```

如果你不需要这些事件，则应该删除这些代码。

同给予 Web 网页开发者的强大功能相比，用脚本来编写小程序表面上很简单。HTML 开发者甚至在定制小程序及在 Web 网页中使用小程序时,都可以不知道 Java。简化我们的 HouseDisplay 小程序（例如删除文本消息），并把它用

在任何需要存储和显示一组图像的 Web 网页上，这应该是很容易的事情。

如果你认为这个例子需要使用数组，那么你想对了。数组将在第八章中讨论。

实验 7-2：用脚本来编写 HouseOfHouses Web 网页

实验概述

本实验中，你将练习下列内容：

- 在 HTML 中用 JScript 函数来关联事件。
- 在 JScript 代码中使用小程序的公有方法和公有成员变量。

用脚本来编写小程序，就是把 JScript 事件处理函数同小程序对象的公有方法联系起来的过程。本实验中，我们将用 Visual J++来做这些事情。

实验准备

1. 如果你一直都跟随着本书中的范例，你可以跳过本实验步骤部分。
2. 如果没有，请启动 Visual J++并打开 HouseOfHouses 启动器项目（Chapter07\Lab7-2\HouseOfHouses.sln）。

实验步骤

1. 用 Script Outline 窗口添加一个 window 的 Onload 事件的处理函数。Window 对象位于 Script Outline 窗口的 Client Objects & Events 文件夹中。展开 Window 对象下的列表，并双击 Onload。像以前一样，一个

事件处理函数的框架会提供给你。向 JScript 函数添加下列代码：

```
function window_onload()
{
housedisplay.setMessage(″ Move the mouse over a button ″ +
    ″ for a description.Click a button to see″ +
    ″ a picture. ″ );
}
```

2. 给 button2 的 Onclick 事件添加一个事件处理函数。该函数使单击 button2 时显示同 housedisplay.house2 相关的图像文件。
3. 给 button2 的 Onmouseover 事件关联一个事件处理函数。该函数应该在小程序显示区域中显示下列信息：

   "Cute 1 Bedroom.A perfect starter for the new family."

4. 对 button3 重复第二步，这次是显示 housedisplay.house3。
5. 对 button3 重复第三步。这次应该显示下列消息：

   "A Lakeside Beauty with all the extras.Perfect retirement home."

6. 运行项目来检验程序。你可以把它同 Chapter07\Sol7-2\HouseOfHouses.sln 下的解决方案作个比较。

选做实验

下面，你可以试一试这个附加实验。

- 试着从 HTML 源代码中删除掉一个或更多的<PARAM>标记，然后再

运行项目。看看 Web 网页是否如期运行。

第八章　保存信息

迄今为止，我们一直在运用窗体信息。你已经知道如何向用户显示信息、如何从用户那里获得信息，以及校验用户给出的信息，来看看它是否符合一套预先定义好的异常情况。然而，你还不知道如何保存这些信息，以备将来使用。这正是本章的主题。

本章中，你将了解下列内容：

- 数组：Java 中把多个对象当做单个对象处理的一种数据类型。
- 文件及如何用 Visual J++来读取和写入文件。
- 列表：比数组更灵活的 WFC 实用程序类。

你将练习使用下列内容：

- 数组
- 列表
- 文件
- 接口
- ArraySorter 类

注意：程序员经常用数据库来保存信息及处理信息。Visual J++标准版没有提供对数据库的直接支持。在附录 D 中，你会找到关于如何用组件对象模型(COM)技术来运用 Microsoft Access 数据库的有关信息。

使用数组

本部分，我们将讨论一种特殊的数据类型：数组。数组使你可以把多个同类对象当作单个单元来处理。数组中所有的对象必须是同一数据类型。这可能有些抽象；让我们来看一个具体的例子。

让我们考虑一个卡片文件格式的电话号码簿。电话号码簿非常适合用数组来表示，因为它的所有数据都是同一种类型（姓名、地址及电话号码）。在卡片文件中，有一个卡片号，每个卡片都很相似。在它们上面有相同类型的信息，而且，如果所有卡片上的信息格式一样的话，它们用起来很容易。卡片文件（所有的卡片及上面的信息）就是数组。

数组中的每个数据被称为“元素”。在卡片文件的例子中，卡片文件中的每个卡片就是数组中的一个元素。数组中的每个元素都有一个编号或“下标”与之相关。为了访问数组中的给定元素，你可以用下标来代表该元素。

作为一个例子，让我们试着在 Java 中创建一个卡片文件。首先，我们创建 FileCard 类来表示单个文件卡片：

```
class FileCard
{
  String name;
  String phoneNumber;
}
```

注意：我们可以通过包括其他的信息来使这个类变得更为复杂，例如工

作、家庭、传真等等的地址或电话号码。这里只是一个简单的例子。

然后，我们将试图用几个不同的 FileCard 对象来声明几个 FileCard 引用：

```
    FileCard mom;
FileCard work;
FileCard cinema;
FileCard elvisPresley;
    FileCard groceryStore;
```

然而，这并不是卡片文件的工作方式。而且，它也不是数组的工作方式。你不能只用：

```
groceryStore.phoneNumber
```

来获取当地市场的电话号码。在卡片文件中，你可以通过翻阅卡片文件并找到某个卡片（也就是说通过确定该卡片在卡片文件中的位置），来得到食品店的电话号码。数组中用下标（位置）来访问元素（卡片）。就像在卡片文件中一样，你可以通过查找 name 成员变量来找到食品店的卡片。

实际的文件卡片只由它们在卡片文件中的位置来识别。这对数组同样是正确的。让我们把例子改动一下，使它能更好地反映这一点：

```
    FileCard arrayElement0; // Mom
FileCard arrayElement1; //Work
FileCard arrayElement2; //Cinema
FileCard arrayElement3; //Elvis Presley
```

```
FileCard arrayElement4; //Grocery Store
```

然后，为了查明 arrayElement0 是 Mom 的文件卡片，我们可以查看它的 name 成员变量。而且，通过查看所有卡片的 name 成员变量，我们可以找到 Elvis 的那一张——arrayElement3。

> **注意：** Java 中数组的下标值是从零(0)开始的。这就是为什么本例中用 arrayElement0,arrayElement1 等作为名称。

在本部分的实验中，你将把使用单独的 FileCard 引用的简单电话号码簿改成使用数组。

声明数组

Java 数组是一种对象。就像其他对象一样，我们用引用来访问数组。现在的问题是，它是哪种类型的引用？它是一种 array 引用。为了声明一个引用，我们需要引用类型和名称。让我们看一看类似的创建引用的例子：

```
String firstName;
```

本例中，引用类型是 String。引用名称是 firstName。

因此，array 类型的引用是什么样子呢？数组类型是基于数组中的元素类型。这种类型是通过在元素类型名后面加上方括号来建立的。记住，数组中所有的元素都是同一种类型。

让我们看一些例子。下面是声明一个 String 元素的数组。

```
String[] daysOfTheWeek;
```

像其他引用声明一样，这个声明只创建对一个对象的引用。现在还没有数组对象。我们将在后面的部分中创建数组对象。这个数组将保存一星期中七天的名称。这就是本部分的例子。我们将在本部分中逐渐完成这个例子。要查看整个例子的代码，请打开 Chapter08\Days\DayOfTheWeek.sln 下的解决方案。

这个例子使用了一个熟悉的类——String。我们也可以创建其他类的数组或任何内置类型的数组。下面声明一个布尔值的数组：

```
boolean[] booleanArray;
```

而下面是浮点型值数组的声明：

```
float[] floatArray;
```

另外，你可以用一个 array 类型作为另一个数组类型的基础。换句话说，你可以创建一个元素为数组类型的数组：

```
int[][] integerMatrix;
```

这里 integerMatrix 是一个整型数组的数组。你可以认为它是一个二维数组。

创建数组

声明对类的引用，只是给你提供一个使用该类对象的途径。如果你需要新的引用对象，必须明确地创建该对象。就像创建其他类型的对象一样，你可以使用关键字 new。用构造器可以给新对象赋值。实际上，有时，你必须为创建

对象提供有关信息。创建数组时也一样。当你创建一个数组对象时，你必须给出数组中的元素数目。这被称为数组的“长度”。

创建数组对象

- 在元素类型后面使用关键字 new，然后在方括号中标明数组长度。

下面是 DayOfTheWeek 例子中的一些代码：

```
String[] daysOfTheWeek;
...
daysOfTheWeek=new String[7];
```

因为 daysOfTheWeek 数组需要保存一星期中七天的名字，数组对象的创建长度为 7；也就是说有 7 个元素的空间。

现在我们有了 7 个 String 引用的数组，它们的缺省值为 null。这就像是我们已经编写了下列代码：

```
String daysOfTheWeek0;
String daysOfTheWeek1;
String daysOfTheWeek2;
String daysOfTheWeek3;
String daysOfTheWeek4;
String daysOfTheWeek5;
String daysOfTheWeek6;
```

记住，在这一点上，我们的数组对象包含了 7 个 String 引用，而不是 7 个 String 对象。

你也可以通过把各个元素的值放在数组类型后面的花括号中，来用特定的初始值创建数组。下面是用初始值来创建一个五个整数的数组：

```
int[] listOfValues;
listOfValues=new int[] {1,2,3,4,5};
```

注意，这里数组长度是由列表中值的个数决定的，而不是由方括号中的数字给出。

访问数组元素

那么，如何访问数组中的元素呢？

访问数组中的元素

- 在数组引用后面的方括号中使用下标值即可。

下面是例子中的数组的初始化代码：

```
String [] daysOfTheWeek;
DayOfTheWeek = new String [7];
DayOfTheWeek[0] =″ Sunday″ ;
DayOfTheWeek[1] =″ Monday″ ;
DayOfTheWeek[2] =″ Tuesday″ ;
DayOfTheWeek[3] =″ Wednesday″ ;
```

```
DayOfTheWeek[4] =″ Thursday″ ;
DayOfTheWeek[5] =″ Friday″ ;
    DayOfTheWeek[6] =″ Saturday″ ;
```

如果没有包括方括号，就是引用整个数组。

daysOfTheWeek 数组中的每个元素都是一个 String 引用。这里我们给 7 个 String 引用分别指定了一个 String 对象。记住，引号中的值是一个 String 对象。在这之前，每个元素的值都是 null。

数组作为对象

Java 中的数组是基于特定类的对象。这个类是来自一个已有的 Java 数据类型。类定义了你能用这种类型的对象做什么。

你已经了解了用数组能做的一些事情：

- 如何声明数组引用（同其他类型没有什么不同）。
- 如何用关键字 new 来创建数组对象及指定数组对象的长度。
- 如何通过方括号中的下标号来访问数组中的单个元素。

你也会看到这些数组下标值是从零(0)开始的整型值。

关于数组对象，你还可以做其他事情。为了获得数组对象的长度，请使用 length 成员变量。数组对象的 length 变量是只读的；也就是说，你可以读取这个值，但不能更改它。例如，下面说明了如何获得 daysOfTheWeek 数组的长度：

```
    String [] daysOfTheWeek;
DaysOfTheWeek = new String [7];
```

```
...
if (input >= 0 & input <daysOfTheWeek.length)
{
msg =″ That's ″ + daysOfTheWeek[input] +″ . ″ ;
            }
            else
            {
                msg = ″ Value out of range. Values between 0 and 6 only. ″ ;
            }
```

实验 8-1：改进电话簿

实验概述

本实验中，你将使用一个应用程序来显示名字和电话号码。现在，它用 10 个独立的变量来保存名字和电话号码。你将把它改成用数组，而不是用独立的成员变量。

你将练习下列内容：

- 声明数组。
- 创建数组对象。
- 使用数组中的元素。

实验设置

1.启动 Visual J++。
2.打开 PhoneList 启动器项目（Chapter08\Lab8-1\PhoneList.sln）。

实验步骤

1.声明一个数组来保存电话号码簿。
- 用一个数组声明来代替十个单独的卡片声明。把数组命名为 cardFile。
- 用 currentIndex 的声明来取代 currentCard 的声明。它的数据类型为 int。它保存了当前的 FileCard 对象（即显示在窗体上的那一个）的下标值。

2.创建数组来保存电话号码簿。
- 在创建 FileCard 对象的语句之前，添加一条创建十个元素的数组的语句。
- 修改创建 FileCard 对象的语句，把它们赋给数组元素。
- 设置 currentIndex 来指向数组中的第一个元素。

3.修改事件处理程序来使 Previous 和 Next 按钮能用于数组下标。
- 找到 Previous 按钮的事件处理程序。
- 修改程序逻辑，来选择前一个 FileCard。使用 currentIndex（新代码应该明显短于已有的代码）。
- 对 Next 按钮重复上述过程。

4.修改 saveDate 和 setUI 方法来使用 currentIndex。

5. 修改电话号码簿开头和末尾的复选项来使用数组下标。
   - 修改 setUI 方法使在数组的开头能选择激活或禁止 Previous 按钮。使用 currentIndex。
   - 修改 setUI 方法，使在数组的末尾能选择激活或禁止 Next 按钮。
6. 测试应用程序。你可以对照 Chapter08\Sol8-1\PhoneList.sln 下的解决方案。

选做实验

下面是你可以添加到本实验中的选做内容：

- 为名和姓创建单独的域。向窗体中添加一个标签控件来显示其全名，用下面两种格式之一：FIRST LAST 或 LAST,FIRST。提供一个菜单项来让用户选择全名的格式。

下一步

这确实很好，但用名字和电话号码创建数组元素的应用程序的代码难于编写。你对数组中的元素的改动在你下次运行程序时不会再显示出来。在后面的章节中，你将了解到如何用文件来保存程序运行间的值。

使用文件

本节中我们将练习使用文件。用文件输入输出应用程序，就可以保存用户

对电话号码簿的改动，这样，每次用户运行应用程序时，都会看到这些改动。

文件 I/O

术语“输入”和“输出”指信息如何传入和传出应用程序。你在前面我们说过的 DayOfTheWeek 代码中已经看到了一些这样的例子。数据输入的 Edit 控件被命名为 edtInput；显示用的 Label 控件被命名为 lblOutput。“文件输入”指应用程序从文件中读取信息；“文件输出”指应用程序向文件中写入（或保存）信息。这些术语通常合起来简称为“文件 I/O”。

File 类

与 Java 中的其他事情一样，读文件或写文件都要使用类。Visual J++提供的 I/O 类在 WFC 软件包 com.ms.wfc.io 中。

最重要的 I/O 类是 File。File 类提供了打开磁盘文件来读或写的方法。File 类的超类是 DataStream。DateStream 类提供了读取或写入不同数据格式的方法，因为 File 类是从 DataStream 继承来的，File 类也继承了 DataStream 的这些公有方法。

注意：还有其他的 I/O 库。本节只讨论 WFC 库提供的 I/O 功能。

打开文件

让我们用另一个例子来看一看如何使用 Java 中的文件。这个例子中，你将用文件 HALL_OF_FAME.SCORES 来跟踪你编写的游戏的最高分。在 Visual J++

中打开 Chapter08\HiScores\HiScores.sln 下的 HisScores 项目。

在读写文件之前，需要建立一个 File 对象。应用程序中的 File 对象提供从文件中读取信息及向文件写入信息的方法。它让你访问硬盘上的文件，但它不是真正的文件。物理文件是计算机硬盘上用来保存信息的一个区域（文件可以在软盘或其他地方的例如网络上，这里我们只讨论它在本地硬盘上的情况）。你已经创建的所有范例程序都被保存在计算机文件中。

为了让应用程序访问物理文件，应用程序中的 File 对象首先需要通过“打开”文件的过程来同物理文件关联起来。

在 WFC 中，你可以以两种方式打开文件：用 File 构造器，或用 File 类中的 open 或 create 静态方法。

用静态方法来打开文件

用两个静态方法是打开文件的最简单的方式。读取或写入一个已有文件，用 File.open 方法，给出文件名。打开一个文件来写入新的数据，用 File.create 方法，给出文件名。

用构造器来打开文件

尽管 File 构造器更为灵活，但它也有些复杂。你可以通过构造器的两个参数来用指定的选项打开文件，这两个参数是一个 String 文件名和打开文件的选项。

表 8-1 列出了一些常用的打开文件的选项。这些选项在 File 中被声明为静态最终成员变量（注意，它们同第三章中解释的 MessageBox 类的按钮设置代

码相类似）。

表 8-1　打开文件的文件构造器选项

选项	说明
File.READ	打开文件来读取信息
File.WRITE	打开文件来写入信息
File.OPEN	如果文件已存在，只打开它（如果不存在就返回错误）
File.CREATE	创建文件（如果它已存在就超越它）
File.CREATE_NEW	只创建新文件（如果它已存在就返回错误）
File.OPEN_OR_CREATE	如果文件存在，就打开它；否则就新建一个

表 8-1 中有些选项会报告错误，这被称为“异常”。你将在第九章中了解到异常的有关信息。

静态方法 File.open 只打开已有的文件。它就像使用 File.OPEN、File.READ 和 File.WRITE 选项的构造器。

静态方法 File.create 当文件已存在时超越它。这就像使用 File.CREATE、File.READ 和 File.WRITE 选项的构造器。

如果你想指定除文件名外的打开文件的选项，则必须使用构造器格式。例如，如果想打开一个文件，不论它是否存在于磁盘上，都必须使用带 File.OPEN_OR_CREATE 选项的构造器。

一些例子

下面是打开 HALL_OF_FAME.SCORES 文件来读写的一些例子。前两个来自 HiScores 例子。

- 使用静态方法 open：

```
File fileScores;
    static final String FILENAME="HALL_OF_FAME.SCORES";
    fileScores=File.open(FILENAME);
```

- 使用静态方法 create：

```
File fileScores;
    static final String FILENAME="HALL_OF_FAME.SCORES";
    fileScores=File.create(FILENAME);
```

- 使用构造器：

```
File fileScores;
    static final String FILENAME="HALL_OF_FAME.SCORES";
    fileScores=new
File(FILENAME,File.OPEN_OR_CREATE|File.READ|File.WRITE);
```

现在已经打开文件了，你可以从文件中读取信息或向文件中写入信息。

写文件

写文件用 DateStream 类的 write 方法之一。下面是这些方法：

```
public void writeByte(byte value);
public void writeShort(short value);
```

```
public void writeInt(int value);
public void writeLong (long value);
public void writeChar(char value);
public void writeBoolean(boolean value);
public void writeFloat(float value);
public void writeDouble(double value);
public void writeString(String value);
```

让我们看看如何用这些方法来写 HALL_OF_FAME.SCROES 文件。例如，在游戏结束时，为了记录最高分和玩家姓名，你首先要创建一个新文件：

```
File fileScores;
static final String FILENAME="HALL_OF_FAME.SCORES";
fileScores=File.create(FILENAME);
```

然后，你可以记录前三名个姓名和得分：

```
fileScores.writeString(name[0]);
fileScores.writeInt(score[0]);
fileScores.writeString(name[1]);
fileScores.writeInt(score[1]);
fileScores.writeString(name[2]);
fileScores.writeInt(score[2]);
```

注意：信息以计算机容易使用的数据格式保存在文件中。如果用 Notepad 来打开 HALL_OF_FAME.SCORES 文件，很可能辨认不出记录的信息。你可能会找到姓名，因为它们是以字符串保存的，但得分值就很难找到了。

下一步，我们将用 DateStream 类的方法来读取这些信息。

读文件

读取文件用 DataStream 的 read 方法之一。例如：

```
    public byte readByte();
public short readShort();
public int readInt();
public long readlong();
public char readChar();
public boolean readBoolean();
public float readFloat();
public double readDouble();
    public String readString();
```

现在，我们取出 HALL_OF_FAME.SCORES 文件，并在下一个游戏中用它来确定这个玩家的得分是否足够高以进入前三名。为了打开一个已有的文件，我们使用静态方法 open。

如果文件不存在，open 方法报告一个错误，因此，在试图打开文件之前，我们需要检验它的存在。为了检测一个文件是否存在，我们调用 File.exists 方法：

```
    static final String FILENAME = ″ HALL_OF_FAME.SCORES″;
IF(File.exists(FILENAME))
{
```

```
            File fileScores;
            FileScores = File.open(FILENAME);
            Name[0] = fileScores.readString();
            Score[0] = fileScores.readInt();
            …
}
else
{
            name[0] =″ None yet″ ;
          score[] = 0;
          …
}
```

注意： 按照写入文件的顺序来读取文件中的信息非常重要。

关闭文件

在打开文件之后关闭文件是非常重要的，因为通常对文件的改变要在文件被关闭之后才是永久性的。当你用完文件之后关闭它，以确保其他需要访问该文件的人能够访问它。

关闭文件要调用 close 方法。典型的文件 I/O 包括打开文件、读或写文件，然后关闭它：

```
static final String FILENAME = ″ HALL_OF_FAME.SCORES″ ;
```

```
File fileScores;
FileScores = File.open(FILENAME);
FileScores.writeString(name[0]);
FileScores.writeInt(score[0]);
...
fileScores.close();
```

重复动作：一个简单的循环

通常，文件中的信息是重复性的，就像我们的 HALL_OF_FAME.SCORES 例子一样。我们可以重复代码来三次读取名字和得分，如下所示：

```
    static final int SIZE = 3;
String [] name = new String [SIZE];
Int[] score = new int[SIZE];
Static final String FILENAME = ″ HALL_OF_FAME.SCORES″ ;
File fileScores;

    FileScores = File.open(FILENAME);
Name[0] = fileScores.readString();
Score[0] = fileScores.readInt();
Name[1] = fileScores.readString ();
Score[1] = fileScores.readInt();
Name[2] = fileScores.readString();
```

```
Score[2] = fileScores.readInt();
    FileScores.close();
```

然而，这种方式十分单调乏味。它也容易出错，因为有更多的机会输错代码；在上面的例子中，用 score[2]来代替 score[1]，会将错误的信息载入到数组中。

注意，读取文件的 6 行代码可以分成只有数组下标号不同的三对代码。我们可以用“循环”技术来更加有效地处理这些重复动作。循环允许程序员多次执行一行或多行代码来完成一个特定的任务。

Java 提供好几个循环的方法。其中一个就是 while 语句，或称为 while 循环。while 循环看起来像 if 语句（没有 else 部分）。实际上，它们非常相似，因为它们都计算一个布尔表达式的值，如果表达式为真，就执行某些动作。

```
    While( trueFalseCheck )
{
// stuff-to-do
            }
            if trueFalseCheck )
            {
                // stuff-to-do
    }
```

不同之处是在 while 循环中，在 stuff-to-do 部分被执行完后，程序返回到 while 部分，再次计算表达式的值。如果它还是 true，该程序块被再次执行。它循环

到条件表达式为 false。

让我们用 while 循环改写上面的例子，来从文件中读取三个名字和得分：

```
    static final int SIZE = 3;
String[ ] name = new String[SIZE];
Int[] score = new int[SIZE];
Static final String FILENAME =″ HALL_OF_FAME.SCORES″ ;
File fileScores;

    FileScores = File.open(FILENAME);

Int index = 0;
While(index < SIZE)
{
name[index] = fileScores.readString();
score[index] = fileScores.readInt();
index = index + 1;
}
```

要点：你一定要记得改变下标号，否则程序将一直循环。这种编程错误被称为“死循环”。

这里，如果你忘记增加 index，循环不会是死循环。相反，文件中的信息会被读完。读完之后，readString 会产生一个异常。你会在第九章中了解到异常。编程时还是要避免异常。

实验 8-2：更新保存的电话号码簿

本实验中，你将继续完善电话号码簿应用程序。你将添加在文件中保存名字的功能，因此，用户对名字和电话号码的改变能在应用程序结束后保存下来。

实验概述

本实验中，你将练习下列内容：

- 打开文件
- 从文件中读取信息
- 向文件中写入信息

实验设置

1.启动 Visual J++。
2.打开 PhoneList 启动器项目（Chapter08\Lab8-2\PhoneList.sln）。

实验步骤

1. 记录数据文件 MYLIST.PHONE。这个数据文件的格式如下：
   - 电话号码簿中的条目数。
   - 单个条目（每个条目有两个字符串：名字和电话号码）。
2. 在文件开头添加对 com.ms.wfc.io 软件包的 import 语句。
3. 添加对 File 类的声明。把引用命名为 phoneListFile。
   - 添加一个 PhoneList 类成员变量的引用。
   - 注意静态的最终字符串 FILENAME。

4. 在构造器中替换代码，来初始化 FileCard 对象，使它使用电话号码文件中的值。
   - 打开文件以便读取它。使用静态的 File.open 方法。
   - 从数据文件中读取数组的长度。
   - 用从文件中读取到的长度来创建数组。
   - 创建数组元素，并用从文件中读取到的值初始化它们。用 while 循环来重复数组中每个元素的这个动作。
   - 关闭文件。
5. 在关闭事件的处理程序中添加保存电话号码簿文件的代码。
   - 调用 saveData 来保存窗体的值。
   - 用静态方法 File.create 来打开文件写入数据。
   - 写入数据文件的数组的大小。
   - 将 FileCard 元素的值写入到文件。用 while 循环来重复数组中每个元素的这个动作。
   - 关闭文件。
6. 测试应用程序。你可以把它同 Chapter08\Sol8-2\PhoneList.sln 下的解决方案进行比较。

选做实验

下面是对这个实验的改进建议：

- 用窗体来提示用户在离开应用程序时保存改动。
- 让用户指定数据文件的名字。提供几个“标准”的文件以供选择。

- 在 File 菜单中添加菜单项，使用户可以打开文件和关闭文件。

下一步

数组允许我们把多个独立条目作为一个单元来使用。然而，它们还是非常死板的，因为数组中元素的数目是固定的。在后面的章节中，你将了解到有关列表的内容。

使用列表

WFC 库包含了好几个功能强大的实用程序类，它们可以在 com.ms.wfc.util 软件包中找到。其中一个就是 List 类。

列表与数组类似，它们都允许我们把多个对象作为一个单元来处理。列表中的一个单元被称为“条目”。另外，列表的优点是可以按需要而加长或缩短。

让我们通过创建一个简单的购物列表应用程序，来看一看几种基本的列表使用方法。在 Chapter08\Shopping1\ShoppingList.sln 下打开 ShoppingList 解决方案，就可以看到这个例子的代码。

注意：在 WFC Util 软件包中，还有其他有趣和有用的类。你可以浏览整个文档，来找出这些实用程序类。

创建列表

创建列表时，可以选择是创建一个空列表，还是创建一个带有初始值的列

表。这里，我们创建一个空列表。创建空列表时可以给它两条信息：

- 列表初始为多大。
- 列表生长时每次加多少元素空间。

你可以把列表想像为记录待购商品的几页表格纸。创建列表时，你可以指定列表有多大。你也可以指定一张表格纸有多大。也就是说，当列表生长时，一页新增的列表有多大。

不带参数使用 List 构造器时，它用缺省值建立一个空列表。如果不指定初始大小，列表开始时的大小为 10 个条目空间。要指定初始大小和“生长大小”，可以用两个参数来调用 List 构造器。如果你不指定生长大小，列表将成倍地生长。

对于购物列表，我们创建一个初始大小为 15、生长大小为 5 的列表：

```
import com.ms.wfc.util.*;

List shoppingList;
shoppingList=new List(15,5);
```

现在，shoppingList 对象是空的，但它拥有 15 个条目的空间。当我们向 shoppingList 中添加第 16 个条目时，它长大 5 个条目空间，因此它将拥有 20 个条目空间。

向列表中添加条目

向列表中添加条目可以使用 addItem 方法：

```
public void addItem(Object item);
```

注意： addItem 方法的参数为一个对象。在第四章中你已经了解到，Java 中的每个对象同 java.lang.Object 类都有一个“Is-A”关系，因为所有的类都是 java.lang.Object 的子类。

例如，下面向 shoppingList 中添加一个新的 String 条目：

```
shoppingList.addItem(edtInput.getText());
```

查看列表中的条目

像数组一样，从列表中获得条目需要使用下标。

要从列表中获得一个条目，使用 getItem 方法：

```
public Object getItem(int index);
```

在下面的例子中，我们将从 shoppingList 对象中获得第三个条目。getItem 方法的下标号参数我们定为 2，因为列表中的第一个条目下标号值为零(0)。我们调用 java.lang.Object 的 toString 方法来获得对 setText 调用的一个 String 引用。

```
Object item;
item=shoppingList.getItem(2);
edtInput.setText(item.toString());
```

getItem 的返回值类型为 java.lang.Object。为了以指定的方式处理这个条目，你首先需要进行类型转换（改变类型），把它改为合适的类。

下面是类型转换的一个例子。这是处理从列表中获得对象的典型方法。如果你不能确定对象的类型，可以用 instanceof 来检查。如果没有发生类型转换，该操作会抛出一个 ClassCastException。你将在第九章中了解到关于类型转换和

异常的详细情况。

```
String item
item=(String)shoppingList.getItem(2);
edtInput.setText(item);
```

删除列表中的条目

用 removeItem 方法来删除列表中的条目：

```
public Object removeItem(int index);
public boolean removeItem(Object item);
```

例如，下面是从 shoppingList 中删除第三个条目的代码：

```
shoppingList.removeItem(2);
```

本例中，我们对被删除的对象不感兴趣，也许是因为用户已经购买了购物列表中的这件物品。因此我们忽略了 removeItem 方法的返回值。

实验 8-3：创建动态的电话号码簿

现在，你可以把电话号码簿应用程序改成用 WFC 列表类，而不是用数组。列表类可以添加和删除条目，因此，用户将可以添加和删除电话号码。

实验概述

本实验中，你将练习下列内容：

- 创建列表。

- 向列表中添加元素。
- 删除列表中的元素。

实验准备

1.启动 Visual J++。

2.打开 Chapter08\Lab8-3\PhoneList.sln 下的 PhoneList 启动器项目。

3.如果不想使用自己的实验数据，可以从前面的项目中复制 MYLIST.PHONE。

实验步骤

1. 在文件的开头，为 com.ms.wfc.util 软件包添加 import 语句。
2. 声明一个列表引用来保存电话号码簿。
   - 找到数组的声明。
   - 用列表引用的声明来代替它。
3. 在构造器中创建列表对象，并用文件中的数据填充它。
   - 把创建数组的语句改成创建列表对象。
   - 找到把 FileCard 对象赋给数组元素的语句。把它修改成向列表对象中添加新的 FileCard 对象。
4. 更新 saveData 以使用列表。
   - 在 saveData 中声明一个 FileCard 引用，让它等于列表中 currentIndex 指向的条目。
   - 把表格中的值保存在 FileCard 对象中。

- 对 setUI 作类似的改动。用列表类的 getSize 方法来获得列表中条目的数目。
- 对 btnNext 的 Click 事件处理程序作类似的改动。

5. 在 Closing 事件处理程序中，把列表中的值保存到文件。
   - 修改写入电话号码簿大小的语句。
   - 找到写入 FileCard 名字和电话号码的语句。把它们改成用列表中的值。

     提示：你可能会发现，创建两个局部变量非常有帮助：一个用来保存列表中的条目数的整型变量，一个用来访问列表中的单个条目的 FileCard。
6. 实现添加新文件卡片的功能。
   - 为 Add 按钮添加事件处理程序。
   - 在这个事件处理程序中创建新的 FileCard 对象，把它添加到列表中，设置 currentIndex，来访问新的 FileCard 对象，并调用 setUI 来更新用户界面。

     提示：用 List 类的 findIndex 方法来获得列表中对象的位置。
7. 实现删除文件卡片的功能。
   - 为 Delete 按钮添加事件处理程序。
   - 在这个事件处理程序中删除列表中当前的 FileCard 对象。如果列表为空，添加一个空的 FileCard 对象，并设置 currentIndex 来访问新的卡片。否则，更改 currentIndex，来访问列表中下一个卡片。调用 setUI 来更新用户界面。

提示：除非是在列表的末尾，currentIndex 仍然是列表中的合法下标值。

8. 测试应用程序。你可以把它同 Chapter08\Sol8-3\PhoneList.sln 下的解决方案进行比较。

下面是本实验的一些改进建议：

- 在从列表中删除对象之前，询问用户是否真的想要从列表中删除当前的文件卡片。
- 提供一个 Copy Entry 函数，它用当前条目的名字和电话号码创建一个新的条目。

下一步

现在，用户可以添加和删除电话号码了。但在列表中查找电话号码仍然很困难。很快，我们将了解到如何给一组对象排序。为了做到这些，我们需要首先了解一下接口的有关事项。

接口

列表提供很大的灵活性。然而，在添加和删除了许多条目之后，便很难在列表中找到一个条目。就像真实的卡片一样，如果你想查找时能快一些，你需要把它们按一定的顺序排列好。如果条目不是按某种逻辑顺序排列的，为了查找需要的东西，你将不得不查看每个条目。

目前，还没有任何实用程序类能够给列表排序。然而，com.ms.wfc.util.ArraySorter 实用程序类可以给数组排序。而在数组和列表之间存在几种转换方式。然而，为了使用 ArraySorter，你需要了解接口。本节中我们将详细讨论接口。下一节我们将讨论如何在数组和列表之间进行转换。

什么是接口

接口同类很相似。实际上，简单的类和简单的接口的代码只有关键字不同：

```
public class SimpleClass
{
}

public interface SimpleInterface
{
}
```

像类一样，编译好的接口保存在.class 文件中。接口也可以拥有成员变量和方法，尽管对它们有一些限制，我们将在本章后面的部分中看到这一点。在 WFC 中所有的接口名开头，都有一个大写的“I”，它代表“interface”。

接口和类之间的区别在哪里呢？接口描述了类实现的动作。除了两个重大的差别外，接口同继承机制（子类和超类）非常相似：

- 类只能继承一个类，而对于接口，类可以实现或多或少的接口。
- 对于继承性，类继承了超类的方法；子类可以选择是否超越超类的方法。接口的方法没有实现，因此，类必须实现接口中的每个方法。

刚开始时，这可能听起来有些神秘。让我们来看看接口能拥有的成员类型。

接口的成员

像类一样，接口可以有两种成员：成员变量和成员方法。这些成员同类的成员非常相似，但它们由于接口的本质还是有一些差别。记住，接口没有实现任何动作，但它为其他类定义了一套动作：一种行为方式。

注意：接口中的成员变量和方法的这些特征的技术原因已经超出了本书讨论的范围。

接口中的方法

接口定义了动作方式，因此，接口的基本成员是方法。接口中的方法的语法格式为返回值类型、方法名、参数和分号：

```
int increment(int x);
```

这个方法是公有的和抽象的，即使没有提供这两个关键字。接口中所有的方法都被定义为公有的和抽象的。

注意：抽象方法没有主体，即花括号之间的执行代码，但它们还是由分号来结束。

现在的习惯是省略接口方法声明中的 abstract 关键字。

接口中的成员变量

像接口方法一样，接口中的成员变量也有许多修饰符。接口中的成员变量都被定义为 public、static 和 final。

在接口中，下面这行代码：

```
int STEP=5;
```

等同于：

```
public static final int STEP=5;
```

现在的习惯是包括所有的接口修饰符。同样是出于习惯，静态最终变量的名字都用大写。

最终的成员变量不能更改其值；它们必须被初始化。静态成员变量可以用接口的名字来访问；它们不会同特定的实例联系起来。

一个接口例子

让我们看看一个基本的增加数字的接口，它使用了你迄今为止已经见过的成员。根据 WFC 命名习惯，接口名为 IIncrement。

```
public interface Iincrement
{
  public int increment(int x);
  public static final int STEP=5;
}
```

要查看这个例子的代码，打开 Chapter08\Incrementer\Incrementer.sln 项目。

implements 关键字

现在，我们可以用基本的接口做什么呢？你不能直接创建这个接口的实例，但可以创建一个类来实现这个接口。为了在类中实现接口，使用关键字 implements。下面是实现 IIncrement 接口的类的例子：

```
    public class Adder implements Iincrement
{
...
}
```

为了让这个类能编译通过，它必须实现接口中的所有方法。本例中只有一个：increment。为了实现 increment 方法，我们使用 STEP 成员变量。因为它是静态的，用接口名就可以访问它：

```
    public class Adder implements Iincrement
{
  public int increment(int x)
  {
    return x+IIncrement.STEP;
  }
}
```

Adder 类实现了 IIncrement，因此 IIncrement 的成员也是 Adder 的成员。

这意味着 STEP 可以被直接访问：

```
public class Adder implements Iincrement
{
  public int increment(int x)
  {
     return x+ STEP;
  }
}
```

其他的类也可以实现这个接口：

```
public class SingleStep implements Iincrement
{
  public int increment(int x)
  {
     return x+1;
  }
}
```

你可以创建一个 IIncrement 的引用，并用 Adder 对象或 SingleStep 对象来访问它：

```
lIncrement incr1=new Adder();
IIncrement incr2=new SingleStep();
```

“动作类似”关系

迄今为止，我们已经可以用继承来代替定义一个 IIncrement 接口。接口是这样的：任何类都可以实行任何接口。Adder 不需要是 IIncrement 的一个子类；它可以是一个窗体：

```
public class AdderForm extends Form implements Iincrement
{
        public int increment (int x)
        {
                return x + STEP;
        }
        ...
}
```

AdderForm 是一个窗体，也就是说，它是 Form 的一个子类。AdderForm 继承了超类 com.ms.wfc.ui.Form 的“窗体特性”。AdderForm 也可以像 IIncrement 一样动作。

你可以编写任何类来实现 IIncrement。也就是说，任何类都可以像 IIncrement 一样动作。或者，从代码的角度上说，任何实现了接口的类都可以通过对该接口的引用来访问它。

在我们的例子中，有时 Iincrement 引用访问窗体：

```
public class AdderForm extends Form implements IIncrement
{
```

```
        IIncrement incr;
        ...
        {
                incr = this;
        }
}
```

而有时 IIncrement 引用访问 SingleStep 的一个实例：

```
Iincrement incr;
SingleStep ss=new SingleStep();
...
{
  incr=ss;
}
```

数组和列表之间的转换

现在，你已经了解接口，我们可以用 com.ms.wfc.util.ArraySorter 来给数组排序了。然而，在开始之前，你还需要知道一件事：如何在数组和列表之间转换。

我们的例子中，我们将对 shoppingList 进行排序，来帮助我们找出列表中需要的东西。要查看这个例子的代码，打开 Chapter08\Shopping2\ShoppingList.sln

项目。

从列表中获得数组

在使用 ArraySorter 类之前，我们需要一个包含列表中的值的数组。我们用 getAllItems 方法来创建这样一个数组：

```
public Object[] getAllItems();
```

下面是 shoppingList 例子的样子：

```
Object[] array;
array=shoppingList.getAllItems();
```

排序数组

让我们来看看如何使用 ArraySorter 类。最简单的方法是使用静态方法 sort：

```
public static void sort(Object[] items,IComparer comp);
```

第一个参数是你想要排序的数组。第二个参数是一个实现了 IComparer 接口的对象。下面是 Icomparer 接口的声明：

```
public interface Icomparer
{
  public int compare(Object a,Object b);
}
```

compare 方法返回一个整数。表 8-2 概括了 compare 方法的返回值。

表 8-2 Icomparer.compare 的返回值

返回值	说明
0（零）	两个值相等(a==b)
负数	第一个值小于第二个值(a<b)
正数	第一个值大于第二个值(a>b)

因此，要给数组排序，你需要一个实现了 IComparer 接口的类，即一个能比较两个对象的类。这个类只用来演示排序，我们不用这个类的实例做任何其他事。它被称为 helper 类。

下面是 shoppingList 的一个比较类的例子：

```
class StringComparer implements Icomparer
{
        public int compare(Object a, Object b)
        {
                String paramA;
                ParamA = a.toString ();
                Return paramA.compare(b.toString());
        }
}
```

注意：这个执行代码用 Object 的 toString 方法来获得两个参数的字符串表示。通常，你需要对对象进行类型转换来作比较。

这个执行程序使用了 String 的 compare 方法。这个方法返回 IComparer 的 compare 方法的适当的值。通常，你需要编写更多的代码来比较这些值。

下面是给 shoppingList 排序的代码：

```
    Object[ ] array;
array=shoppingList.getAllItems();
ArraySorter.sort(array,new StringComparer());
```

现在，我们来看看另一个实现 IComparer 的类的例子：

```
class IntComparer implements IComparer
{
        public int compare(Object a, Object b)
        {
                Integer paramA = (Integer)a;
                Integer paramB = (Integer)b;

                int x = paramA.intValue();;
                int y = paramB.intValue();
                return x – y;
        }
}
```

从数组创建列表

我们可以用带数组参数的列表构造器来从数组创建一个列表。

```
public List(Object[] itemsT);
```

下面是排序 shoppingList 的代码：

```
Object[] array;
array=ShoppingList.getAllItems();
ArraySorter.sort(array,new StringComparer());
ShoppingList=new List(array);
```

实验 8-4：给电话号码簿排序

本实验中，你将继续使用电话号码簿应用程序，向应用程序中添加排序功能，以使用户可以按名字来排序列表。

实验概述

本实验中，你将练习下列内容：

- 创建一个比较方法。
- 排序列表。

实验准备

1.启动 Visual J++。

2.打开 PhoneList 启动器项目（Chapter08\Lab8-4\PhoneList.sln）。
3.如果你要使用自己的实验数据，从前一个项目中复制 MYLIST.PHONE。

实验步骤

1. 创建一个实现 IComparer 的 helper 类。
   - 向项目中添加一个名为 FileCardSorter 的类。
   - 在 FileCardSorter 中实现 IComparer。
   - 用 Class Outline 来添加 comparer 方法的存根。实现这个方法。
2. 为 Sort 按钮添加一个 Click 事件处理程序。
   - 在事件处理程序中创建一个列表中的数据数组。
   - 用 ArraySorter 来排序数组。
   - 把排序过的数组保存在电话号码簿中。
   - 设置 currentIndex 来访问列表的第一个条目。
   - 调用 setUI 来更新用户界面。
3. 测试应用程序。你可以把程序同 Chapter08\Sol8-4\PhoneList.sln 下的项目进行比较。

选做实验

下面是一些你可以添加的实验改进措施：

- 添加菜单项来让用户按名字或电话号码排序。
- 如果你前面把名字分成名和姓，让用户按名或姓来排序。

第九章　小程序中的动画

这是本书中关于小程序（Java 的小程序，它是非独立的运行于 Internet 上的程序）内容的第三章；在第六章中介绍了创建和显示小程序的基础知识，第七章中介绍使用<PARAM>标记和脚本编程同小程序通信的方法，本章将介绍创建能显示动画图像的小程序。

动画图像实际是一组静止图片按照一定顺序以较快速度显示，由于视觉暂留，人会产生流畅画面的感觉，从而产生动画效果。要实现这种效果，首先得了解 Java 关于线程和异常的有关内容。

在本章中，读者将学习到：

- Java 中的多线程；
- Java 中的异常；
- 小程序中的动画。

读者将学会创建：

- 象节拍器一样动作的小程序；
- 显示动画图像的小程序。

多线程

本书中到现在为止编写的所有应用程序都只有一个流程或者说线程，来控制应用程序所做的一切。应用程序调用一个又一个方法，以顺序方式推进，应用程序经常等待用户做某件事情，但在同一时刻，应用程序只做一件事情。

可以创建具有不止一个线程的应用程序，这样的应用程序叫做多线程的应用程序，可以这样来理解:这些线程是完全不相关的。

通常的计算机只有一个 CPU，所以，同一时刻计算机不可能做多件事情。但是，可以这样来理解计算机在同一时刻做多件事情这个概念。举例来说，你正在等待从某个 Web 站点下载，同时可以写信感谢别人送你生日礼物，这是两个不同的应用程序，但它们共享 CPU。这就很像多线程，不过，真正的多线程是在一个应用程序中共享 CPU。又比如，你在使用 Word 字处理应用程序给 Claire 姨妈打印感谢信的同时，还可以使用该应用程序给母亲写感谢信。

AWT 中的事件处理程序

在每一个小程序中，有一个主线程，它只调用四个方法：init，start，stop 和 destroy。所有别的事件处理程序，比如 action，mouseDown 和 keyDown 都是通过其他线程去调用。AWT 为每个处理程序创建一个新的线程。

要获得生动的动画效果，就必须显示许多不同的图像。也许读者想采用事件处理程序来管理动画（因为它有自己的一个线程），但这个方法并不好；应该创建新的线程来管理多幅图像的显示。

java.lang.Thread 类

到现在为止编写的应用程序和小程序都是 VM（虚拟机）为用户创建线程。要应用程序和小程序编程独立做事情的话，用户须自己创建线程。Java 应用程序中的所有线程都用到 java.lang.Thread 类。在这一部分，着重介绍 Thread(线程)类中最重要的七个方法。

Start 和 run 方法

最主要的 Thread 方法是 start 和 run：

```
public void strat();
public void run();
```

典型的情况是，扩展 Thread 的类可以超越 run 方法：即可以把要 Thread 对象执行的代码放到这个方法里。比如，想用创建的线程设定某个值，就可以编写下面的代码：

```
class Stuff
{
int myValue = 0;

class MyThread extends Thread
{
    public void run()
    {
```

```
            myValue = 5;
        }
    }
    public Stuff()
    {
        Thread t = new MyThread();
    }
     }
```

创建 Stuff 对象后，myValue 的值是 0。Mythread 对象已经创建了，但它没做任何事情；调用 start 方法可以启动线程：

```
     class Stuff
     {
    int myValue = 0;

    class MyThread extends Thread
    {
        public void run()
        {
            myValue = 5;
        }
    }
    public Stuff()
    {
```

```
        Thread t = new MyThread();
        t.start();
    }
    }
```

调用 start 方法后，Thread 对象（在本例中是 t）获得 Java VM 创建的控制流程，该控制流程执行 run 方法中的代码。一旦 run 方法执行完毕，Java VM 就把线程清除，并标识它，以便进行自动垃圾收集（Java 的一种与内存管理有关的机制）。

Suspend 和 resume 方法

Suspend 方法暂停给定 Thread 对象的控制流程，而 resume 方法让控制流程重新运行起来：

```
Public void suspend( );
Public void resume( );
```

suspend 方法可以被自己和别的线程调用，通常是别的线程调用，比如鼠标单击事件处理程序调用它。前边说过，一个线程管理动画的显示；为了响应调用 suspend 方法的鼠标单击事件（对应一个线程），动画线程将暂停；暂停后，该线程不能做任何事情，甚至没有办法调用 resume 方法使自己重新启动，直到有别的线程调用了 resume 方法（比如可能是另一个单击事件处理程序）。

sleep 方法

sleep 方法使控制流程暂停睡眠给定的一段时间，以毫秒为单位：

```
public static void sleep(long millis) throws InterruptedException;
```

sleep 方法有些有趣的语法：

首先，sleep 方法是静态的，不需要特定的 Thread 对象，就可以调用它。即使应用程序中没有引用 Thread 对象，线程调用 Thread.sleep 后也转入睡眠状态；即在任意方法中，都可以调用 sleep 方法。

其次，sleep 方法有一个和它关联的异常：InterruptedException，关键字 throws 标明这种关联；关于异常的内容将在本章稍后的“异常处理”中介绍。

interrupt 和 stop 方法

interrupt 方法唤醒正在睡眠的线程：

```
public void interrupt( );
```

如果某个线程调用了 sleep 方法而暂停，可以调用 interrupt 方法，后者用 InterruptedException 来强制使 sleep 的调用提前返回。

stop 方法使线程停止，它激发 ThreadDeath 给出 errors（错误）：

```
public void stop( );
```

本章稍后将介绍 errors 的知识。

同步

多线程应用程序有特别值得考虑的内容，其中最重要的关注内容是共享。

仍然以字处理应用程序为例：假定你已经写好了那封给 Claire 姨妈的信，

并用后台线程打印；至于给妈妈的信，你决定不另外写，而只是在给 Claire 姨妈的信的基础上做一些修改。由于你正在前台的编辑窗口修改给妈妈的信，所以不希望后台线程使用它；即前后台这两个线程要独立各做各的事，避免冲突。字处理应用程序是这样解决冲突的：它创建前台编辑窗口中的内容副本给后台打印，前台从此以后做的修改不影响副本。

如果有多个线程要更新同一些值，或者说有不同的线程更新，使用同样的值时，可以用关键字 synchronized 来避免冲突。

关键字 synchronized 是方法的修饰符。当调用了有 synchronized 修饰的方法后，Java VM 阻塞该对象上任何别的同样有 synchronized 修饰方法的调用，直到第一个方法调用完成。因此，如果有多个成员变量需要同步，就可以创建 accessor 方法来处理同步。

比如，假定用三个成员变量来代表三维，可以创建 accessor 方法来操作这些值：

```
class Box
{
private int dimensions[] = new int[3];

public void setDimensions(int x, int y, int z)
{
    dimensions[0] = x;
    dimensions[1] = y;
    dimensions[2] = z;
}
```

```
public int[] getDimensions()
{
    int retValue = new int[3];
    retValue[0] = dimensions[0];
    retValue[1] = dimension[1];
    retValue[2] = dimensions[2];
    return retValue
}
 }
```

可以看到，如果在调用 getDimensions 时被 setDimensions 的调用中断，就可能产生不期望得到的结果。

Thread 1（线程 1）	Thread 2（线程 2）
Box myBox=new Box();	
MyBox.setDimensions(2，3，4);	
int dims;	
dims=getDimensions();	myBox.setDimensions(7，8，9);

假定 getDimensions 的调用只执行了两行语句，就被来自线程 2 的 setDimensions 调用中断，这时 getDimensions 的返回值是{2， 8， 9}，这就是不期望的结果，因为稍后一点调用 getDimensions 得到返回值是{7， 8， 9}。

更坏的情况是在调用 setDimensions 时被另外一个线程的 setDimensions 调用中断。

Thread 1（线程 1）	Thread 2（线程 2）
Box myBox=new Box();	
myBox.setDimensions(2，3，4);	myBox.setDimensions(7，8，9);
int dims;	
dims=getDimensions();	

假如线程 1 的 setDimensions 的调用执行两行语句后被线程 2 中断，三维数组的值将是{7，8，4}；以后，所有调用 getDimensions 都将返回这组错误值。

只要给 getDimensions 和 setDimensions 方法加上修饰符 synchronized，就可以避免这种错误的出现：

```
class Box
{
private int dimensions[] = new int[3];

public synchronized void setDimensions(int x, int y, int z)
{
    dimensions[0] = x;
    dimensions[1] = y;
    dimensions[2] = z;
}
public synchronized int[] getDimensions()
{
    int retValue = new int[3];
    retValue[0] = dimensions [0];
```

```
        retValue[1] = dimensions [1];
        retValue[2] = dimensions [2];
        return retValue
    }
     }
```

java.lang.Runnable 接口

创建线程的一种方法是扩展 java.lang.Thread，另一种方法是使用 java.lang.Runnable 接口。

Runnable 接口很简单：

```
public interface Runnable
{
        public void run();
}
```

Thread 类有一个以 Runnable 对象为参数的构造器；任何类都能实现 Runnable 接口。调用具有 Runnable 对象的 Thread 构造器就可以创建一个 Thread 对象。

下面是本章前面例子 Stuff 的代码：

```
class Stuff
{
int myValue = 0;

class MyThread extends Thread
{
```

```
        public void run()
        {
                myValue = 5;
        }
}
        public Stuff()
        {
                Thread t = new MyThread();
                t.start();
}
 }
```

下面使用 Runnable 接口来改写代码：

```
 class Stuff2 implements Runnable
 {
int myValue = 0;

public void run()
{
        myValue = 5;
}

public Stuff2()
{
```

```
        Thread t = new Thread(this);
        t.start();
    }
    }
```

由于类 Stuff2 使用了 Runnable ，它就必须覆盖 run 方法。这样，Thread 对象的构造器获得 this 引用，Thread 对象 t 从 this 引用调用 run 方法。

异常处理

编程时，不可能事事按计划进行，有时将出现一些意外；这一部分将介绍有关异常的内容。异常是 Java 中用来表明有没有预料到的事情发生的方法之一。异常发生后，有一些规范的措施来处理它；那么，异常和规范的措施中，到底谁给谁发信号呢？

一个组件（component）或对象使用异常向另外一个组件发信号，这两者的关系就是，一个组件调用另一个组件上的方法。调用组件期望被调用组件完成某些操作；如果被调用组件不能完成期望的操作，它就使用异常向调用组件发信号。

举例来说，假定已经创建了一个 Queue（队列）类，队列是一种 FIFO（先进先出）的数据结构，好比电影院买票时排队：人们排成一列，谁第一个到就买到第一张票，谁越晚到，谁排队等待的时间就越长。

很简单的一个例子，Queue 类看起来像下面这样：

```
 public class Queue
 {
public void add(Object o)
{
      …
}
public object getNext()
{
      …
public int getCount()
{
      …
}
 }
```

于是，别的程序员就可以使用这个队列了。他们可以创建 Queue 对象，向队列中添加对象，从队列中取得下一个对象（从队列的前面取），或检查一下队列中所有的对象个数。

如果队列是空的或者某人调用 getNet 该怎么办？应该提出一个异常来警告其他程序员说队列中无对象。

注意：异常的讨论是从引出异常的组件开始的，更一般的作法是先写异常处理程序，而不是引出异常。但是，先从引出异常开始讨论，可以更容易理解异常。

异常的声明

Java 中的异常是 Java.lang.Exception 的子类，因此，可以创建一个新的异常类：

```
public class QueueEmptyException extends Exception
{
}
```

在 getNext 方法的开头，可以检查队列中的对象（项目）数，如果队列中没有项目，可以创建一个 QueueEmptyException 对象并引出它：

```
public class Queue
{
public void add(Object o)
{
    …
}
public object getNext()
{
        if(getCount() ==0)
            throw new QueueEmptyException();
        …
}
public int getCount()
```

```
{
    …
}
 }
```

要指出 getNext 方法可以引出这个异常，就必须在方法头中包含 throws 子句。

注意： Visual j++要求任何可能返回异常的方法都要包含 throws 子句。从技术上说，允许在同一个方法中引出和捕获异常，在这种情况下，throws 子句可以省略。但是，异常机制的开销很大，通常，不大可能在方法中引出异常的地方捕获到异常。

不过，幸运的是，作为 java.lang.RuntimeException 子类的异常不必在 throws 子句中报告；这些异常可能在任何方法中发生。如果必须在 throws 子句中报告这些异常，那将是难于负担的一件事情。

下面是用 throws 子句更新了的 getNext 方法：

```
 public class Queue
 {
public void add(Object 0)
{
    …
}
public Object getNext() throws QueueEmptyException
{
```

```
        if(getCount() == 0)
            throw new QueueEmptyException();
        …
    }
    public int getCount()
    {
        …
    }
     }
```

java.lang.Exception 超类有两个构造器：一个没有参数，一个带有一个字符串参数。

字符串参数是为异常提供的消息，Exception 类有一个 getMessage 方法来返回这个消息，处理程序通常调用 getMessage 方法，来获取关于异常的更多的信息。要充分利用这一点，可以用下面的代码更新 Exception 类：

```
 public class QueueEmptyException extends Exception
 {
public QueueEmptyException()
{
    super(″ The queue is empty″ );
}
 }
```

异常的处理

考查使用 Queue 类的例子代码组件：

```
public class Cinema
{
Queue theLine = new Queue();

Public void sellTicket()
{
    Object customer;
    Customer = the Line.getNext( );
    // sell the ticket and record the sale using customer
}
...
}
```

getNext 方法包含了 QueueEmptyException 的 throws 子句，sellTickets 方法必须处理异常或传递它。下一部分介绍怎样传递一个异常。使用 try-catch 块来处理异常，try-catch 块的语法如下：

```
try
{
// statements where an exception may be thrown
}
catch( exceptionName variableName)
```

```
{
// statements to handle the exception
}
```

在异常可能引出的语句附近创建 try 块，如果 try 块中任何地方引出异常的话，try 块中的其他语句就会跳过去，控制流程跳转到和发生的异常相匹配的 catch 块。不同的 Exception 类可以有多个 catch 块：

```
try
{
// statements where an exception may be thrown
}
catch( exceptionName1 variableName )
{
// statements to handle exceptionName1
}
catch( exceptionName2 variableName )
{
// statements to handle exceptionName2
}
```

于是，Cinema 类可以更新成这样：

```
public class Cinema
{
Queue theLine = new Queue();
```

```
Public void sellTicket()
{
      Object customer;
      Try
      {
            customer = theLine.getNext();
            … // sell the ticket
      }
      catch(QueueEmptyException e)
            StatusBar.setText(″ Error: ″  + e.getMessage());
            Customer = null;
      }
      … // record the sale using customer
}
…
 }
```

注意，customer 在 try 块外声明，如果在 try 块里声明的话，在 catch 块的设置和使用“record-the-sale”部分的代码时，将引用不到 customer。

异常的传递

有时，不能有效地处理异常，但可以把异常传递给另外的组件。我们以改写以前的 cinema 类来作例子：

```
 public class Cinema
 {
Queue theLine = new Queue;

Public void sellTicket()
{
      Object customer;
      Customer = theLine.getNext();
      … // sell the ticket and record the sale using customer
}
…
 }
```

下面是 cinema 类用到的 Queue 类：

```
 public class Queue
 {
public void add(Object 0)
{
    …
}
public Object getNext() throws QueueEmptyException
{
      if(getCount() == 0)
            throw new QueueEmptyException();
```

```
        ...
    }
    public int getCount()
    {
        ...
    }
     }
```

Cinema.sellTickets 调用 Queue.getNext，Queue.getNext 引出异常 QueueEmpty-Exception。因此，Cinema.sellTickets 必须处理异常或传递它。前一部分已经介绍过异常的处理。

当一个方法把异常传递给调用程序时，该方法必须把异常引出到调用对象，通过使用方法第一个语句中的 throws 子句，该方法必须报告这个异常：

```
 public class Cinema
 {
Queue theLine = new Queue();

Public void sellTicket() throws QueueEmptyException
{
      Object customer;
      Customer = theLine.getNext();
      ... // sell the ticket and record the sale using customer
}
...
```

}

错 误

异常向一个组件发信号，说明被调用的方法没有完成期望的操作，该组件将采取一些规范的措施来处理这个异常。错误也发出信号，表明被调用的方法没有完成预期的操作；但是，对错误来说，通常没有合理的规范化的应对措施；错误发生通常标志着应用程序的结束。

异常是 java.lang.Exception 的子类，错误是 java.lang.Error 的子类，它们确实存在着共同点：java.lang.Exception 和 java.lang.Error 都是 java.lang.Throwable 的子类，Throwable 是所有“oops”消息的根；还有，Throwable 声明了 getMessage 方法，并且 throw 操作以 Throwable 对象为参数，因此，可以像引出 Exception 对象一样引出 Error 对象。

但要记住：异常必须在方法的 throws 子句中报告，而错误则没必要。

异常和错误的另外一个区别是：错误通常是不可修复的。以 ThreadDeath 错误为例，当调用 stop 方法终止一个线程时，ThreadDeath 错误引出，这时没有办法修复：这个线程不能重新启动了。

实 验 9-1:创 建 节 拍 器

这个实验将创建一个节拍器，节拍器就是用重复的间隔来计时的设备。

实 验 概 述

在这个实验中，将练习以下内容：

- 实现 Runnable 接口；
- 创建 Thread 对象；
- 用 sleep 方法暂停 Thread 对象
- 编写异常处理程序

音乐家用节拍器来定音乐的拍子或速度；拍子通常以每分钟的拍数为计量单位，60 的拍子表示一分钟拍击 60 次，也即一秒钟一次；典型的拍子是 40 到 240 之间。

实验准备

1. 启动 Visual J++;
2. 打开 Metroneme 启动程序项目（Chapter09\Lab9-1\Metroneme.sln）。

实验步骤

1. 修改 Metronome 类来实现 java.lang.Runnable;
   - 把 implements Runnable 添加到类头（class header）；
   - 为 run 方法添加覆盖它的方法代码。
2. 在 run 方法中，用下面的结构创建一个无限循环：

```
construct:
while(true)
{
}
```

3. 在循环中，用下面的代码让 Panel main 闪烁；

```
main.setBackground(Color.black);
main.repaint();
Thread.sleep(flash);
Main.setBackground(Color.white);
Main.repaint();
Thread.sleep(interval – flash);
```

4. 由 sleep 方法引出异常 InterruptedException，在循环中和 catch 块中，建立处理程序来处理异常，并在浏览器的状态栏上更新显示来自异常的消息；
5. 使用 Runnable 接口，修改 init 方法来创建线程并启动它；
6. 测试这个小程序，把解决方案同 Chapter09\Sol9-1\Metronome.sln 目录下的比较。

选做实验

这个实验提供如下选做部分来增强练习：

- 把这个小程序改写成 WFC 应用程序；
- 在单个小程序或应用程序中提供多个节拍器。

下一步

既然有了这些模块，就可以创建一个显示动画的小程序了。

编写动画代码

这部分将编写动画代码，实际上，读者已经知道编写实际动画代码所需要的所有模块了。不过，在运行动画程序之前，先得下载所有需要的图像。这部分先介绍用 MediaTracker 类下载图像的作法。

下载图像

下载图像是个同步进程。调用 getImage 时，下载请求就提交了；请求一提交，便返回 getImage 的调用，图像就在后台下载。使用 MediaTracker 对象可以监视图像集的下载状态。

MediaTracker 的关键方法有：

```
Public void addImage(Image img, int id);
Public void waitForAll() throws InterruptedException;
Public void waitForID(int id) throws InterruptedException;

Public boolean isErrorAny();
Public boolean isErrorID(int id);

Public int statusAll(boolean load);
Public int statusID(int id, boolean load);
```

一旦创建了 MediaTracker 对象，就可以添加图像以便追踪。使用 addImage 方法添加图像到 MediaTracker 对象，addImage 方法有两个参数：Image 对象和

一个 ID（标识号）；使用 ID 值可以把相关图像组合到一起。

典型的情况是：把所有的 Image 对象添加到 MediaTracker 对象后，再调用作用在 MediaTracker 对象上的 waitForAll 方法，该方法等下载完所有图像后才返回。当下面三种情况之一发生时，一幅图像的下载操作就算完成：

- 图像已经成功下载；
- 在下载时发生错误；
- 用户取消下载.

另外，可以调用 waitForID 来等待一组图像下载完毕，组是通过传递给 addImage 的 ID（标识号）的值来区分的。

图像下载操作完成后，可以使用 isErrorAny 或 isErrorID 来检查下载过程中发生的错误。调用 statusAll 或 statusID 可以获取更多关于下载状态的信息，这些函数的返回值由下面的 MediaTracter 的静态最终成员变量定义：

```
public static final int ABORTED;
public static final int COMPLETE;
public static final int ERRORED;
public static final int LOADING;
```

下面是使用 MediaTracter 的一个例子：

```
MediaTracker tracker = new MediaTracker(this);
int i = 0;
While(i < MAX)
{
```

```
imgList[I] = getImage(
      getDocumentBase(), BASENAME + (i + 1)+″ .jpg″ );
tracker.addImage(imgList[i], i);
i ++;
 }

 try
 {
tracker.waitForAll();
bAllLoaded = ! tracker.isErrorAny();
 }
 catch(InterruptedException e)
 {
 }
```

这个例子的代码存放在 Chapter09\AnimDemo\AnimDemo.sln 目录下。

实验 9-2：自旋字母 E

实验概述

这个实验修改一个小程序来显示一系列的文件名。修改的程序将下载和那些文件名相关的图像，而不是显示文件名。所有的图像成功下载完毕后，就显示一个我们所熟悉的动画：Internet Explorer 的徽标：自旋的字母 E。

实验目的

这个实验将练习以下内容：

- 使用 MediaTracker 对象来监视图像下载；
- 创建动画图像.

实验启动

1. 启动 Visual J++;
2. 打开 AnimLab starter 项目（Chapter09\Lab9-2\AnimLab.sln）。

实验步骤

1. 创建一个 Image 对象数组，命名为 imgList;
2. 修改 displayFile 方法以便显示 Image 对象而不是文件名；
3. 在 run 方法中下载图像：
   - 声明一个 Boolean（布尔）成员变量来追踪下载，变量命名为 bAllLoaded;
   - 创建一个 MediaTracker 对象；
   - 为每个 Image 对象调用 getImage 方法；
   - 为每个 Image 对象调用 addImage 方法；
   - 等待所有的对象下载完成；
   - 如果所有图像成功下载，则显示它们；否则，显示错误信息。
4. 测试修改过的程序，可以和 Chapter09\Sol9-2\AnimLab.sln 对比检查。

选做实验

这个实验提供如下选做部分来增强练习：

- 修改程序，以便使用参数来指定图像的名称和编号；
- 修改程序，以便使用参数来确定下载图像间隔的毫秒数。

第十章　Java 中的软件包

由于所有的 Java 类组合在一块形成软件包，所以，没有它们将什么也做不了。

本章先概览 Java 软件包的语法和语义，接下去讨论 Java 和 Visual J++的一些软件包。读者可能已经注意到，实际上，编程已经用到的类都包含在这些预定义的软件包中。使用现成的软件包比创建新的软件包要普遍，但是，有时用户也需要创建自己的软件包，所以，本章最后将举例说明怎样实现自己创建软件包。

在本章中，读者可以学习到：

- 软件包和文件夹的关系；
- Java 中的固有软件包；
- Windows Foundation Class（WFC，窗口基类）的框架结构和 WFC 的软件包；
- 创建用户自己的软件包。

用户将创建：

- 能从剪贴板上读取字符串和把字符串写到剪贴板的应用程序；
- 一个时钟应用程序；
- 一个实用程序：包含一个能对两个数进行求和的简单类。

什么是软件包

软件包是 Java 最主要的组织实体，它从逻辑上把相关类组合到一起，还定义了类之间的一种访问形式。每个应用程序都有一个默认的软件包，它包含了代码中引用到的所有软件包。

定义一个类作为命名数据包的成员

- 在源代码中的第一行包含“package”语句：

```
package pkgname;
```

pkgname 的取值就是软件包的名称，它必须同源代码所在的文件夹名或目录名匹配，下面是 MyStuff 软件包的 package 语句：

```
package MyStuff;
```

package 语句必须出现在源代码的开头，它前头只允许有空行和注释。package 语句对在源代码文件中定义的所有类都适用。

软件包和文件系统

软件包和文件系统有密切的关系，这种关系是公用类和文件联系的纽带。

公用类必须存放在与它同名的文件中，这就是说，取名 MyClass 的公用类必须存放在名为 MyClass.java 的源代码文件中。

包含 package 语句的文件也必须存放在同名的文件夹中，比如，下面这几

行代码表明：名称为 MyStuff 的文件夹中有名称为 MyClass.java 的文件：

```
package MyStuff;
public class MyClass
```

类文件的位置也很重要，编译这个源代码文件创建两个类文件，MyClass.class 和 MyHelper.class，这些类文件必须存放在命名为 MyStuff 的文件夹中：

```
package MyStuff;
public class MyClass
{
        …
}
class MyHelper
{
        …
}
```

生成两个类文件 MyClass.class 和 MyHelper.class。这些类文件一定在名字为 MyStuff 的文件夹中。

软件包的名称可以包含多个由英文句号分隔的标识符；名称的每一部分隐含着一个独立的文件夹。比如，下面的代码表明：名称为 MySubPackage 的文件夹中有名称为 MyClass 的文件，还有，MySubPackage 文件夹在 MyStuff 文件夹中。

```
package MyStuff.MySubPackage;
public class Myclass
{
        …
}
```

软件包的名称表明一个存放在特定路径下的特定的软件包，存放在不同文件夹中的同名的软件包之间、或者不同软件包中同名的类之间都没有特殊的关系；在上面的第一个例子中，软件包的名称是 MyStuff，第二个例子中软件包的名称是 MyStuff.MySubPackage。一个软件包可以存放在另外一个软件包的子文件夹中，但是，这两个软件包中的类却没有关系；类似的，MyStuff.MyClass 和 MyStuff.MySubPackage.MyClass 这两个类之间也没有特殊的关系。

类路径

类路径是一些文件夹（或目录）的序列，这些文件夹组合起来形成一个应用程序的默认软件包；所有命名的软件包都可以在类路径上文件夹的子文件夹中找到，软件包文件夹是类路径文件夹的一个子文件夹。

比如，假定类路径由下面的文件夹组成：

- C:\Windows\Java\Classes
- C:\Windows\Java\Lib
- C:\MyLibrary

类文件 C:\Windows\Java\Classes\Utilities.class 就直接存放在默认的软件

包中。

现在，假定通过编译下面命名为 MyClass.class 文件中的几行代码来创建一个类路径上的类文件：

```
package MyStuff;
public class MyClass
{
        ...
}
```

MyStuff.MyClass 文件可能在以下文件夹的任意一个当中：

- C:\Windows\Java\Classes\MyStuff
- C:\windows\Java\Lib\MyStuff
- C:\MyLibrary\MyStuff

另外，大多数的 Java 虚拟机（VM）能够使用以 ZIP、JAR 或 CAB 形式存储的类文件，不过，这些 ZIP、JAR 或 CAB 文件必须显式地列在类路径当中；这些压缩文件可以包含别的文件，也可以包含带有许多子文件夹的文件夹的信息。

可以用下面两个文件夹来更新类路径：

- C:\Windows\Java\Classes\Classes.ZIP
- C:\Library\MyClasses.CAB

现在，类文件 MyClass.class 可能出现在 Classes.Zip 或者 MyClasses.CAB 中。如果它包括在 MyClasses.CAB 中，则把它保存在子文件夹 MyStuff 中。

访问控制

第四章的访问控制已经介绍过跟访问有关的三个访问修饰字：public（公共），private（私有）和 protected（保护）。

我们已经介绍过没有关联关键字的第四种访问方式：default access（默认访问）；具有默认访问的项只能被在同一软件包中定义的项访问；由于它只给相同软件包中的成员分配访问权限，所以也称它为软件包访问（package access）。比如，在下面的代码中，一个类和一个方法是公共的，同时它们都有默认访问：

```
 package MyStuff;

 public class MyClass
 {
public void getMoreStuff()
{
     …
}

void doSomeMore()
{
     …
}
 }

 class Helper
 {
```

}

通过上面的介绍可以知道，下面的说法都是正确的：

- 任意类都可以定义 MyStuff.MyClass 类型的成员变量；
- 任意类中的方法可以定义 MyStuff.MyClass 类型的局部变量；
- 如果给定一个 MyStuff.MyClass 对象，则任意类中的方法可以调用 getMoreStuff 方法；
- 只有在 MyStuff 软件包中的类才可以定义 MyStuff.Helper 类型的成员变量；
- 只有在 MyStuff 软件包的类中的方法才可以定义 MyStuff.Helper 类型的局部变量；
- 如果给定一个 MyStuff.MyClass 对象，则只有在 MyStuff 软件包中的类中的方法才可以调用 doSomeMore 方法。

有了默认访问或软件包访问，软件包的的创建者就可以定义只能被相同软件包中的其他类进行访问的类、方法和成员变量。

Java 软件包

我们来快速浏览一遍 Java VM 自带的一些软件包。这些包是作为以“java.”开头的可移植、扩展的 Java 库的一个标准部分发布的；下面将详细地介绍 java.lang 软件包，并汇总说明其他八个软件包。

Java.lang 软件包

Java 语言的核心部分就是 java.lang 软件包，它定义了 Java 中的大多数基本的类。它的核心地位是显然的：每一个 Java 源代码文件都引入了 java.lang 类中的语句：

```
import java.lang.*;
```

java.lang 软件包包含了 Object（目标）类，java.lang.Object 类式 Java 中整个类层次结构的根节点，这个软件包还定义了基本数据类型的类：

String　　Boolean
Character　　Byte
Integer　　Short
Long　　Float
Double

这些类支持数字型的转换和字符串操作，在前面的章节里已经详细地介绍了这些内容。

定义在 java.lang 中的其他一些类列在下表 10-1 中：

表 10-1　java.lang 软件包中定义的几个重要的类

java.lang 的类	说明
Class	为运行时搜集的信息—如 instanceof 操作符提供支持
Math	提供像 pi 和 e 这样的数学常数，还支持三角函数

续表

System	提供对操作系统的访问，包括默认的 I/O 流、环境变量、自动垃圾收集、系统时间和系统属性；许多 System 方法访问 Runtime 类的方法
Runtime	提供对操作系统的访问；使用 java.lang.System 可以更容易地访问大多数 Runtime 方法；唯一的例外是 exec 方法，它开始一个新进程
Thread	和 java.lang.Runnable 接口协同作用提供 Java 中的多线程支持；线程的有关内容请参见第九章
Throwable	它是 Java 中所有异常（Exception）的基类，是 java.lang.Exception、java.lang.Error 和 java.lang.RuntimeException 的超类。应用程序运行时发生意外时，异常和错误就发出信号；有关这方面的详细内容请参见第九章

现在再看看别的一些 Java 软件包：

- java.applet—为 Java 的小程序和声音编辑提供支持，这方面的详细内容请参见第六章和第七章；
- java.awt—支持 Java 固有的窗口和绘图软件包：抽象窗口工具（AWT-Abstract Window Toolkit），该工具包含像字体、颜色和形状一类的基本绘图功能，还支持一些组件，如按钮、列表框、菜单、文本框以及控制安排组件的布局管理器。AWT 是可移植的（本书中许

多图形 I/O 使用 WFC 而不是 AWT），第五章中有 AWT 的概要讨论。

- java.beans—为 Java Beans 定义了应用程序编程接口（API）；Java Beans 是 Java 应用程序环境的中性平台组件结构；
- java.io—定义了大量的本地 I/O 支持类；它既支持字节流 I/O，也支持字符流 I/O。详细内容请参见第八章。
- java.math—支持任意精度值的数学运算，它包含两个类：BigDecimal 和 BigInteger。
- java.net—既支持基于 URL 的，也支持基于套接字的网络 I/O；详细内容请参见第十三章。
- java.text—为国际化的应用程序提供类，包括：日期格式、数字、货币格式以及排序的顺序。
- java.util—包含其他方面的类来支持数据结构、随机数、日期、时区以及日历等。

WFC 软件包

到现在为止，出现过的多数代码使用了 Visual J++带来的窗口基类库（WFC library－Windows Foundation Class library），所有的 WFC 类在 com.ms.wfc 中的软件包里。

- app—包含支持应用程序的类，包括这样一些类：Application（应用程序）、Message（消息）、Window（窗口）、Clipboard （剪贴板）、

DataFormat（数据格式）、Registry（注册）、Time（时间）、Timer（定时器）和 Version（版本）。

- 它是个很重要的的软件包，在大略浏览了主要的 WFC 软件包之后，将花些时间介绍这个包中的几个重要的类。
- core—为 WFC 应用程序和视图设计器（View Designer）定义了核心类，包括 Component（组件）、Container（包容器）、Event（事件）、EventHandler（事件处理程序）、EventInfo（事件信息）、ClassInfo（类信息）和 Enum（枚举）等类。该软件包还定义了在视图设计器中使用的一些基本编辑器。
- data 和 data.ui—提供使用 ADO（ActiveX Data Objects）的数据库访问。
- html—支持动态的超文本链接（DHTML-Dynamic HTML）。
- io—支持 Win32 格式的文件 I/O，具体内容请参见第八章。
- ole32—对 OLE 的拖放提供支持。
- ui—定义 WFC 中的用户接口组件，读者其实已经大量地使用了该软件包中的类了：因为每次用到一个窗体（Form）时，都会用到包中的类。
- util—为数据结构定义一些实用类，第八章中介绍了关于列表框的几个实用类。

WFC 应用程序软件包

WFC 应用程序软件包提供了对应用程序需要的许多基本服务的访问。

Application 类

Application 类支持基本的应用程序服务，本书中几乎每个实验和演示都用到它，另外，我们还使用这个类中的 run 和 exit 方法来开始和停止一个应用程序。

这个类还包括一个重载的 createThread 方法：

```
public static Thread createThread(Delegate method);
public static Thread createThread(Delegate method, Object args[]);
public static Thread createThread(Delegate method, Object args[], int priority)
```

createThread 方法创建一个基于代理（delegate）的线程；代理是依赖于某个对象中特定方法的对象。使用代理来创建线程，允许一个类去创建具有多种动作的线程，与之相对的通过扩展 java.lang.Thread 或用 java.lang.Runnable 来创建线程的类所创建的线程只执行 run 方法中的动作。一个类只有一个 run 方法，所以，要创建能执行多种动作的线程，就需要为每一类的动作创建一个新的类。

注意： Delegates 形成 WFC 中事件模型的基础，关于 delegates 的完整讨论超出了本书的范围；读者可以在下面的 Microsoft Web 站点获取关于 delegates 的更多信息：http://www.microsoft.comlvisualj/techmat/feature/delegates/default.htm .

使用 Application.createThread 创建线程

1. 编写一个没有参数的方法来执行要线程执行的动作；
2. 用 MethodInvoker 为该方法创建一个代理，如：

```
new MethodInvoker(this.doSomething);
```

3. 用新创建的代理调用 Application.createThread；
4. 启动线程。

下面举一个简单的例子，该例子创建一个线程来响应事件；可以参看完整的项目文件 Chapter10\Threading\DelegateThread.sln。

```
// This method specifies the actions for the thread.
Public void doSomething()
{
addOutput(″ doing something″ );
for(int i = 1; i < 4; i ++)
{
    addOutput(″ Number ″  + i);
    try
    {
        Thread.sleep(500);
    }
    catch(InterruptedException e)
```

```
        {
        }
}
addOutput(″ done″ );
 }

 // This is a helper method to handle output.
 Public synchronized void addOutput(String s )
 {
lbOutput.addItem(s);
 }

 // This is the event handler that creates the thread.
 Private void startThread(Object source, Event e)
 {
// Declaring the thread reference.
Thread t;

// Creating the thread using the delegate MethodInvoker
t = Application.createThread(new MethodInvoker(this.doSomething));
try
{
            t.start();
}
```

```
catch(IllegalThreadStateException itse)
{
}
```

Clipboard 类

Clipboard 类是一个很简单的类，提供对系统剪贴板的访问，它包含两个方法来把数据复制到剪贴板，或从剪贴板复制数据到别的地方：setDataObject 和 getDataObject 方法。setDataObject 方法可以对任意的数据类型操作；getDataObject 返回一个 IdataObject 接口。

放置数据到剪贴板上

- 调用 setDataObject。

```
public static void setDataObject(Object data);
    public static void setDataObject(Object data，  boolean copy);
```

单参数方法可以通过把双参数方法的第二个参数的值置为 false 来调用后者。这并不是把数据复制到剪贴板，只是把对数据的引用放到剪贴板。当使用单参数方法时，剪贴板上的数据在应用程序结束时被清除掉。

从剪贴板获取数据

- 调用 getDataObject。

```
public static IDataObject getDataObject();
```

IDataObject 接口和 DataObject 类

IDataObject 接口处理统一数据传输（UDT），UDT 是 OLE 和 ActiveX 中数据传输的基础，它通过剪贴板和 OLE 拖放来使用。下面是 IDataObject 接口的定义：

```
Public interface IDataObject
{
public Object getData(String format);
public Object getData(Class format);

public void setData(String format, Object data);
public void setData(Class format, Object data);
public void setData(Object data);

public boolean getDataPresent(String format);
public boolean getDataPresent(Class format);

public String[] getFormats();

        }
```

Class 参数取值于在 java.lang.Class 中定义的类，String 参数取值于在 DataFormats 类中定义的值。

使用 DataObject 对象可以支持多剪贴板格式，DataObject 类完成 IdataObject 接口。调用 setData 可以取得需要支持的每种剪贴板格式。

要指定获取剪贴板数据的格式，可以调用 getData 来给定需要的剪贴板格

式；要检查获取剪贴板数据的某种特定格式，可以调用 getDataPresent，这也指定了需要的剪贴板格式。

DataFormats 类

DataFormats 类为标准剪贴板格式定义了字符串常量，也为 WFC 的类定义了一些新的格式，下面是 DataFormats 类的静态终结 String 成员变量的列表：

CF_TEXT
CF_UNICODETEXT
CF_DIB
CF_BITMAP
CF_ENHMETAFILE
CF_METAFILEPICT
CF_SYLK
CF_DIF
CF_TIFF
CF_OEMTEXT
CF_PALETTE
CF_PENDATA
CF_RIFF
CF_WAVE
CF_HDROP
CF_LOCALE

CF_HTML
CF_RTFTEXT
CF_CSV
CF_STRINGCLASS
CF_WFCOBJECT

实验 10-1：操作剪贴板字符串

在这个实验中，将创建一个应用程序来读取剪贴板的字符串或写字符串到剪贴板。

实验概述

在这个实验中，将练习：

- 使用剪贴板；
- 检查数据格式。

实验准备

- 启动 Visual J++。

实验步骤

1. 创建一个 Windows 应用程序项目，把它命名为 ClipboardStrings；
2. 在窗体上添加一个 Edit 控件，两个 Button 控件；
   - Edit 控件取名为 edtClipboard；

- 一个 Button 控件取名为 btnGetData，并把它的 Text 属性设为 Get Data；
- 另一个 Button 控件取名为 btnSetData，并把它的 Text 属性设为 Set Data。

3. 在 btnGetData 的 Click 事件处理程序中，做以下处理：
   - 从剪贴板取得数据对象；
   - 如果数据对象有 String（字符串）数据，就用字符串数据更新 edtClipboard；
   - 如果数据对象没有字符串数据，就在 edtClipboard 中显示一条信息。
4. 在 btnSetData 的 Click 事件处理程序中，把 edtClipboard 中的文本放到剪贴板；
5. 测试这个应用程序：
   - 使用 Notepad 程序复制文本到剪贴板；
   - 使用编写的应用程序获取剪贴板的数据；
   - 使用编写的应用程序设置剪贴板的数据（把文本放到剪贴板）；
   - 把剪贴板的数据粘贴到 Notepad 程序；
   - 使用 Paint 画笔程序复制一个图形到剪贴板；
   - 使用编写的应用程序获取剪贴板的数据，这时应该得到错误信息。

可以把编写的程序同 Chapter10\Sol10-1\ClipboardStrings.sln 中的解决方案作个比较。

选做实验

以下是可以尝试选做提高的内容：

- 添加一个方法，如果剪贴板上不是字符串数据时，能使按钮 Get 不可用（通常是置灰），从定时器事件处理程序来调用这个方法；
- 用菜单控制取代按钮来实现功能，即用菜单项来响应命令。

Time 类

WFC 的 Time 类有很多构造器用来跟踪使用 Win32 时间格式的日期和时间，这个类可以很容易地处理 1970 年 1 月 1 日之前的日期（而 java.util.Date 类，跟 C 的运行库一样，只支持从 1970 年 1 月 1 日到 2037 年的这个时间段）。

创建 Time 对象

有几种方法可以用来创建一个 Time 对象。

用当前时间来创建 Time 对象

- 使用无参数的构造器：

```
public Time();
```

基于一个字符串创建 Time 对象：

- 使用带字符串参数的构造器：

```
public Time(String time);
```

使用指定日期来创建 Time 对象：

- 使用三参数的构造器：

```
public Time(int year， int month， int dat);
```

使用指定的日期和时间来创建 Time 对象：

- 使用六参数的构造器：

```
public Time(int year，int month，int day，int hour，int min，int sec);
```

检查 Time

Time 类有许多访问或转换方法用来访问日期和时间，下面是 get、set 和 format 方法的例子。

获取 Time 值的某个部分

- 使用下面的访问方法：

```
public int getYear();
public int getMonth();
public int getDay();
public int getHour();
public int getMinute();
public int getSecond();
public int getMillis();
```

```
public int getDayOfWeek();
public int getDayOfYear();
public Time getDate();
public Time getTimeOfDay();
```

设置 Time 对象日期或时间的某个部分的值

- 使用下面的访问方法：

```
public Time setDate(Time date);
public Time setTimeOfDay(Time timeOfDay);
```

设置 Time 的字符串输出格式

- 使用下面的转换方法：

```
public String formatShortTime();
public String formatLongTime();
public String formatShortDate();
public String formatLongDate();
```

每一种方法的输出形式列在下表中：

方法	输出形式
formatShortTime	小时和分钟
formatLongTime	小时、分钟和秒钟

续表

formatShortDate	月、日和年（比如，10/5/97）
formatLongDate	字符串形式（比如，星期天，10 月 05，1997）

更新 Time

Time 对象可以使用一些 add 方法来更新。

更改 Time 的值

- 使用以下的 add 方法：

```
public Time addYears(int interval);
public Time addMonths(int interval);
public Time addDays(int interval);
public Time addHours(int interval);
public Time addMinutes(int interval);
public Time addSeconds(int interval);
public Time addMillis(int interval);
```

这些方法是添加整数单位的 Time 值，使用下面 addDays 的重载可以添加分数部分的天数：

```
public Time addDays(double interval);
```

Timer 类

Timer 类创建了一个窗口定时器，该定时器每隔相同的间隔时间触发（这叫

定时器事件），间隔的长短是用毫秒数来指定的。

在窗体上使用 Timer 对象

1. 在窗体上放置一个 Timer 控件；
2. 在属性窗口设置一些定时器的属性，请注意：interval 属性指的是两次定时器事件触发间隔的毫秒数。

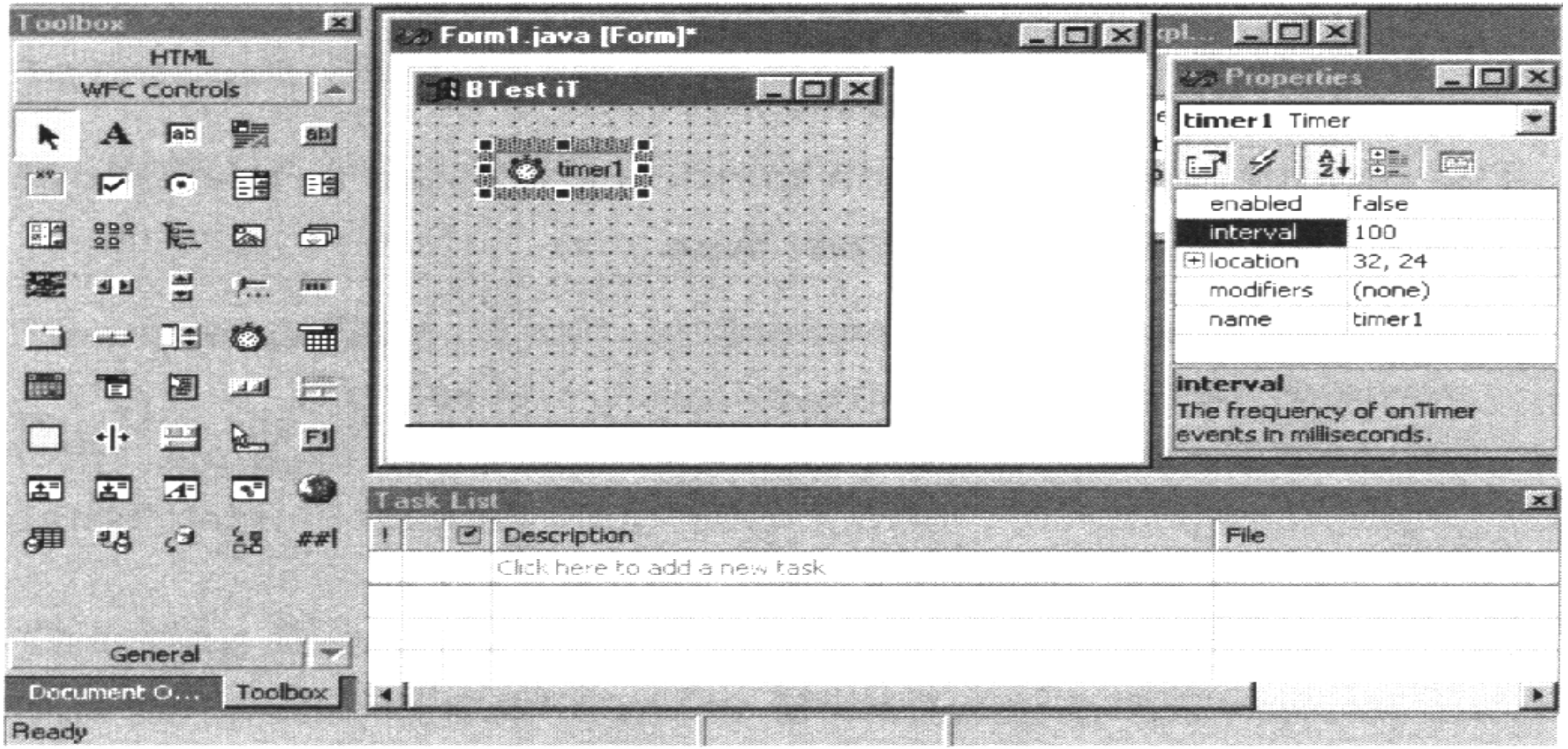


图 10-1　窗体上的 timer 控件及其属性显示

下面是图 10-1 中 timer1 的默认构造器：

```
private void timer1_timer(Object source,  Event e)
        {
        }
```

启动定时器

- 调用 start 方法，
  或
- 将 enable 属性设置为 true。

停止定时器

- 调用 stop 方法，
  或
- 将 enable 属性设置为 false。

实验 10-2：建立自己的 WFC 时钟

在该实验中，给读者提供机会，使用 WFC 来创建一个数字时钟。

实验概述

在这个实验中，将练习：

1. 使用 Timer 对象；
2. 创建一个 Time 对象来获取当前时间；

实验准备

- 启动 Visual J++。

实验步骤

1. 创建一个新的窗口应用程序项目，取名为 DigitalClock;
2. 在窗体（form）上，放置一个 label（标签），取名为 lblTime，将 text 属性的值设为 99:99:99，并调整好字体及字的大小，使之看起来比较舒服;
3. 在窗体上放置一个 Timer 控件，将 interval 属性的值设为 500（注意，单位是毫秒），即半秒;
4. 创建一个取名为 showTime 的方法来完成以下功能：
   - 用当前时间来创建一个 Time 对象;
   - 设置 Time 对象显示时间的格式，要能显示秒;
   - 用格式化的时间值去更新 lblTime 的显示。
5. 让窗体的构造器启动定时器，并调用 showTime 方法去更新初始显示;
6. 为定时器时间创建一个处理程序，并让它调用 showTime。

   测试编写的应用程序，并同存放在 Chapter10\Sol10-2\DigitalClock.sln 的解决方案比较。

选做实验

以下是可以尝试选做提高的内容：

- 增加一个菜单，它包含能修改时间显示格式的命令项；
- 为窗体添加处理程序，使得当鼠标单击窗体时就显示日期，当鼠标再次单击时，又转换成显示时间；或者使得鼠标按下时，切换成显示日期，而鼠标按钮松开时，又回到显示时间状态。

创建自己的软件包

这一部分将介绍怎样使用 Visual J++来创建软件包，还将介绍在项目的类路径上添加文件夹的方法。

创建新的软件包

Visual J++采用基于目录的项目系统，这就是说，每个项目把所有的文件包含在项目文件夹中。只要实际看看一个项目中的所有文件的位置，就可以证实这一点：在 Project Explorer 工具栏上，选择 Show All Files，把结果同 Windows 资源管理器的显示作以比较。

在 Project Explorer 窗口中还可以创建位于项目中的文件夹：在项目或项目项上单击鼠标右键，然后选择 New Folder（新文件夹）；在 Add Package/Folder 对话框中，输入文件夹的名称，如果要让一个文件夹成为 Java 的软件包，那么，取名必须符合 Java 的命名约定（详细内容请参见第三章）。

再次右击一个项目项时，新文件夹就成为所选项目项中的文件夹；每开始一个项目或文件夹时，新文件夹会在所选择的项目或项目项中。

要在一个文件夹中创建源文件，右击该文件夹，然后选择 Add 即可。在子文件夹中创建的任何源文件都会自动包含正确的软件包语句。

在类路径上添加文件夹

Visual J++的项目和方案机制支持一个方案可以包含多个项目，方案中的项目可以是新建的，也可以是已经存在的。

在开发过程中，方案的每个组件可能拥有多个项目；在创建软件包的时候，创建的每一个软件包也可以拥有不同的项目。

在一个方案中，所有项目文件夹都包含在类路径上，因此，通常就不需要特殊的操作去访问在方案中其他项目中定义的类；另一方面，一旦有了一些相对稳定的项目后，再把这些项目包含到每个使用类的方案中就没有多少意义了。

要理顺这些关系，有两种方法可以实现：

- 把软件包安装在开发机器的类路径上；
- 把项目或方案用到的软件包包含到它们的类路径上。

以上两种方法各有优缺点：第一种方法把软件包安装在开发机器的类路径上意味着那些类可以按照统一的方式使用，这很可能是用户机器上的配置；但是，在这样的安排下，要进一步开发原始的软件包或软件包的改进版本就很困难了。

第二种方法把项目或方案用到的软件包包含到它们的类路径上，这保持了使用软件包的灵活性，但是，必须为每个项目用到的软件包设置类路径。

添加文件夹到项目的类路径

1. 从 View（视图）菜单中选择 Project Properties（项目属性）；
2. 在 Project Properties 对话框中，单击 Classpath；
3. 单击 Add，输入要添加的文件夹。

实验 10-3：编写自己的软件包

在这个实验中，将让读者一试身手：创建一个基本的软件包，它包含一个实用的类，能对两个数求和。

实验概述

在这个实验中，将练习：

- 创建 Visual J++的软件包。

实验准备

- 启动 Visual J++。

实验步骤

1. 创建一个项目，取名为 CreateAdder；
2. 添加一个新文件夹到此项目，取名为 MyUtilities；
3. 在新建文件夹中添加一个新的类：Adder；
4. 在 Adder 中，添加一个 add 方法，它具有 public 访问方式，它返回两

个参数的和；

5. 测试 Adder 类：
   - 创建一个窗体，有两个数并把它们相加；
   - 实现 Click 事件处理程序来使用 MyUtilities.Adder。
6. 把实验结果同存放在 Chapter10\Sol10-3\CreateAdder.sln 的解决方案比较。

选做实验

以下是可以尝试选做提高的内容：

- 在同一方案的不同项目中创建测试应用程序；
- 创建一个命令行方式的测试应用程序版本。

实验 10-4：另用方案使用自己的软件包

实验概述

在这个实验中，将练习：

- 添加文件夹到类路径；
- 复制已有文件到项目中；

实验准备

- 启动 Visual J++。

在这个实验中，将使用实验 10-3 中创建的 MyUtility 软件包从另外一个方案去访问 Adder 类。如果在同一个方案中创建了两个项目，那这两个项目文件夹都在类路径上。

实验步骤

1. 在新的方案中创建一个新项目，取名为 TestAdder;
2. 从 CreateAdder 复制测试窗体到新项目：
   - 在 Project Explorer 中右键单击 TestAdder 项目，然后选择 Add;
   - 在 Add Item 对话框中单击 Existing（现成）；
   - 浏览 CreateAdder，选择测试窗体。
3. 尝试创建项目，这时不会成功，因为 MyUtilities 软件包不在类路径上；
4. 添加 CreateAdder 文件夹到项目的类路径上；
5. 尝试再次创建项目。
   这时应该成功。
6. 测试应用程序，并同存放在 Chapter10\Sol10-4\TestAdder.sln 的解决方案比较。

第十一章　控制台应用程序

这一章的章名实际上应当叫做“应用程序”。在本章中学习的内容对于创建 Windows 应用程序与创建基于命令行的应用程序是同等重要的，原因是你将要学习如何在 Java 中进行常规的工作。在学习应用程序的过程中，我们同时也附加了一些内容。这一类应用程序在 Visual J++中的入口指向 Control Application（控制台应用程序）模板。

本章学习的有关内容：

- 控制台 I/O
- main 方法
- 流程控制
- 循环语句

在本章中，将练习使用：

- 命令行参数
- 命令行开关
- for，do 和 switch 语句

控制台应用程序的不同之处

控制台应用程序没有图形化的用户界面，也就是说，它们没有使用Windows。因此，所有的输入和输出都显示在命令行上，或者打印输出，并且，控制台应用程序对鼠标的单击不会产生响应。

使用 Console Application 模板

我们要创建一个控制台应用程序，最简捷的方式就是调用 Console Application 模板，这是 Visual J++提供的部分模板。

创建控制台应用程序

1. 从 File 菜单中，选择 New Project 项；
2. 在 New Project 对话框左侧的窗格中，选择 Visual J++ Project/Application 子文件夹；
3. 在 New Project 对话框右侧的窗格中，选择 Console Application 项，并且键入新的项目的名字；
4. 单击 Open 创建一个新的项目。

按照上面的操作，我们所建立的控制台应用程序中包括了类 class.java1。打开这个类，可以看到以下一段程序：

```
/**
 * This class can take a variable number of parameters on the command
```

```
 * line. Program execution begins with the main() method. The class
 * constructor is not invoked unless an object of type ‘class1’
 * created in the main() method.
 */
public class Class1
{
        /**
         *The main entry point for he application
         *
         *@param args Array of parameters passed to the application
*via the command line.
         */
         public static void main (String[] args)
         {
              // TODO: Add initialization code here
         }
}
```

在这个类中，只有 main 方法的代码段，就是 main 方法把这段代码生成一个应用程序。我们观察其他应用程序，都可以看到在程序中存在着 main 方法。

main 方法

所有的应用程序中都有一个 main 方法，这个方法描述了在应用程序运行时

所发生的事情。

```
    public static void main ( String args[ ])
{
    //stuff-to-do
}
```

我们可以看到，在上面的程序中，main 方法是：public（公共的）、static（静态的）和 void（无返回值的）。下面，让我们具体来了解一下所描述的 main 方法的特性（有关参数的讨论在下一节中介绍）：

- static——在应用程序刚刚开始运行的那一刻，该应用程序的对象都不存在，仅仅有 Java 的 VM（Virtual Machine，虚拟机）存在。VM 可以调用一个静态方法，而不需要这个类的一个实例。所以，静态方法存在于对象之外，并且可以被任意一个对象访问。
- public——很显然，运行应用程序时，VM 需要去访问 main 方法。既然 VM 存在于这个类的外部，那么 main 方法就必须是公共的，以便外部的实体可以访问这个方法。
- void——事实上，VM 并没有被设计成可以处理返回的数值。执行程序时，如果产生了实时错误，Windows 就会产生异常。所以，这儿没有必要返回一个数到 VM，而且如果返回一个数，VM 也会忽略而不处理。

Windows 应用程序

在 Windows Application 模板中，main 方法如下：

```
public static void main (String args[ ])
{
    Application.run (new Form1());
}
```

上面程序中，main 方法创建了一个窗体的实例，并且把这个窗体交给了 Application.run 方法，然后，由 Application.run 把这个窗体挂到 Windows 操作系统上去。

控制台应用程序

在 Control Application 模板中，main 方法如下：

```
public static viod main (String [ ]args)
{
//TODO: Add initialization code here
}
```

缺省情况下，应用程序没有任何操作。此时系统处于等待输入命令状态。

注意：当 Control Application 模板使用“main (String args[ ])”时，Windows 应用程序模板使用“main (String args[ ])”，而 Java 可以灵活地处理参数列表前的空格。

命令行参数

再举一个使用 main 方法最简单的例子：

```
public static void main ( String args[ ])
{
    //stuff-to-do
}
```

参数是一个 String 对象的数组，对于这种方法的参数，还有另外一种写法，如下：

```
public static void main ( String [ ] args)
{
    //stuff-to-do
}
```

参数 args 中包含了一些外加的信息，这些信息是在应用程序开始时写入的。你可能会产生疑问，应用程序是如何得到这些外加信息的，为了加深了解，让我们看一个例子，一个 Java 程序是如何运行的。

运行 Java 应用程序

Java 应用程序运行时，需要访问 Java 的 VM，并且需要给 VM 传递一个类。在命令行中有两条命令可以运行一个 Java，分别是 WJVIEW 和 JVIEW。使用 WJVIEW 命令，命令提示马上恢复可用；而使用 JVIEW 命令，命令提示在 Java 应用程序运行期间无效，直到程序运行完毕后，命令提示才恢复可用。

不带参数运行 Java 应用程序

1. 首先进入命令提示，然后进入到包含类文件的目录，使这个目录成为

当前目录。

类文件应该处在类路径中，这一点很重要，类路径包括了当前的工作目录。

2. 在命令提示中，键入命令 WJVIEW，并在后面写入类的名字，例如：

```
WJVIEW MyClass
```

或者

键入命令 JVIEW，后面加上类的名字，例如：

```
JVIEW MyClass
```

注意： Java 区分字母的大小写，而 DOS 系统不区分。

为 Java 应用程序提供参数的话，则在命令行中输入类名后加上参数即可。举例说，如果要打印句子“This is a test”，则可以用下面的命令来实现：

```
WJVIEW Form1 This is a test
```

这里，我们输入了四个参数，每个参数分别包含了一个独立的单词：This 、is、 a 和 test。

上面的参数输入以空格隔开，如果我们想要输入一个参数，其中包括空格的话，则只要把这个参数用引号包括起来即可。例如，在下面的命令中，我们把两个带空格的参数传递给 main 方法：

```
WJVIEW Form1 ″This is a test.″ ″It is only a test.″
```

在上面的例子中，两个 args 参数分别是 This is a test. 和 It is only a test.

注意：在命令提示中使用的类名应该与在 Java 文件中的类名对应一致。如果类的名字错误的话，上面的例子可能仍然可以工作，但并不能保证。wjview 命令不区分大小写。

设置启动文件

在前面，我们学习了如何在命令行中运行一个 Java 应用程序，另外还介绍了如何给应用程序输入命令行的参数。掌握了以上的内容，就可以在 Visual J++ 环境中，按照相同的步骤运行自己的应用程序了。运行应用程序时，可以给它输入参数，也可以不输入参数。

在一个项目中，往往包括了若干个不同的文件，这些文件都可以启动这个项目。例如，有一个多 Java 文件的项目，其中包括了 public static void 命名的 main 方法，并且 main 方法以 String 对象的数组为参数。对于那些使用 Form 模板的窗体，都包括这个方法。所以，在多文件项目中，任何一个这类窗体都可以作为项目的主窗体使用。

这儿还有另外一个例子：如果有一个 Web 站点的项目，在项目中有多个 HTML 文件。任意一个 HTML 文件都可以作为启动文件，也就是指向这个 web 站点的典型初始化人口。

当手动运行应用程序时，要明确地输入类的名字，这个类是包含了 main 方法，或者，对于 Web 站点，应该输入“主”HTML 文件的名字。程序在 IDE 内工作时，也要有相同的选项。

不带参数，为项目设置启动文件的步骤

1. 首先在 Project Explorer 窗口中，在要求的项目上右击鼠标，出现一个弹出菜单，选择<PROJECT>属性，这里<PROJECT>是指项目的名字；
2. 在 Project Properties 对话框中，单击 Launch 选项卡；
3. 选择 Default 单选钮，并且，在下拉列表中，从 Java 文件列表选择启动文件。所列的每一个 Java 文件都包括一个可随时调用的 main 方法或者是 HTML 文件；
4. 单击 OK 按钮。

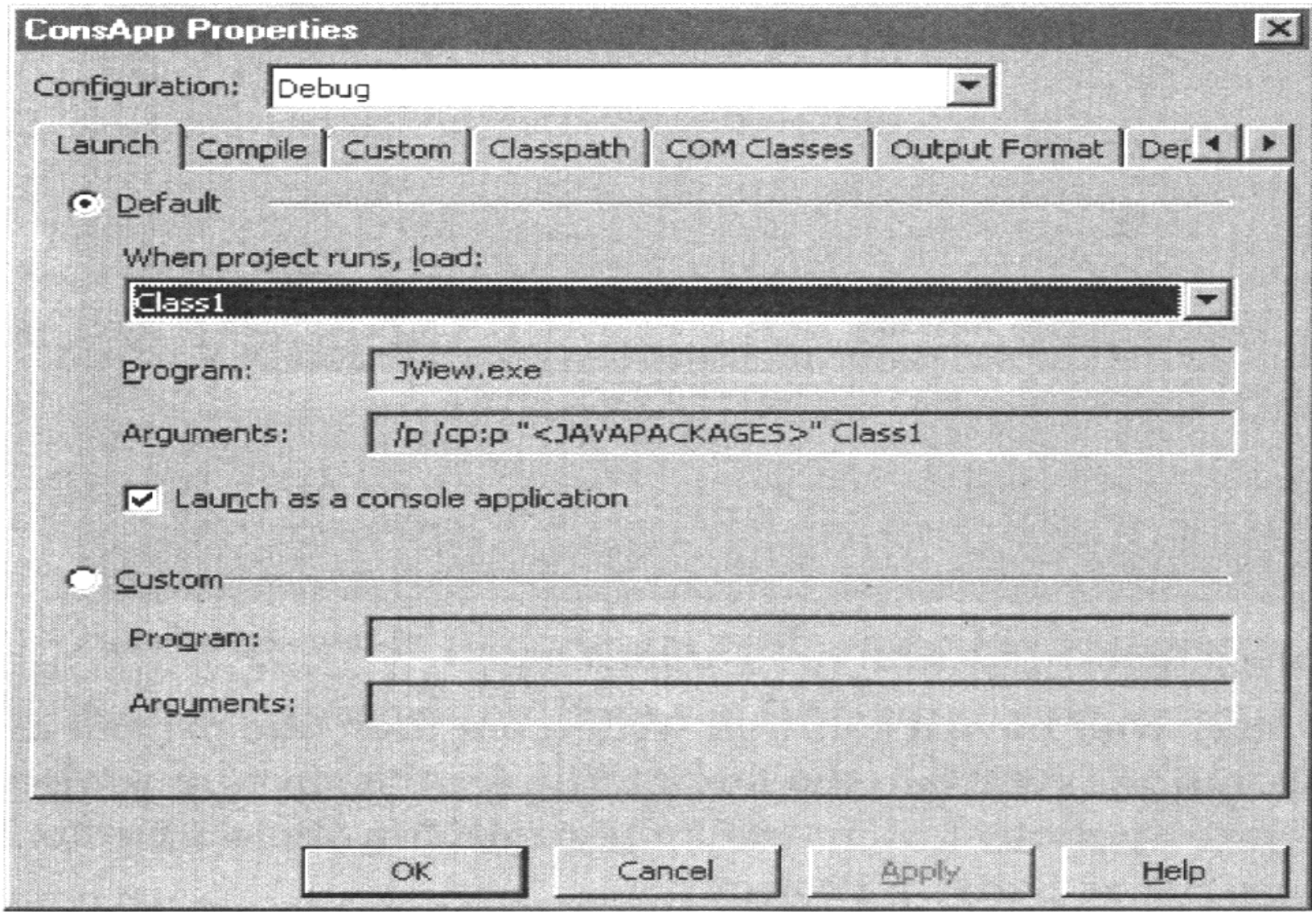


图 11-1　设置无参数的启动文件

上面介绍的是无参数情况，下面我们介绍其他情况，即输入参数时的设置

操作。就在单击 Project Properties 对话框的 OK 按钮（前面过程的第 4 步）之前，先选择单击 Custom 单选钮。对话框显示如图 11-2 所示，在对话框中，Custom 中的 Program 和 Arguments 两项设置是将原先 Default 对应设置复制过来的。接下来，在 Arguments 编辑框中可以输入应用程序的参数。其中的 Arguments 参数将在这个项目带调试运行或者不带调试运行时使用。

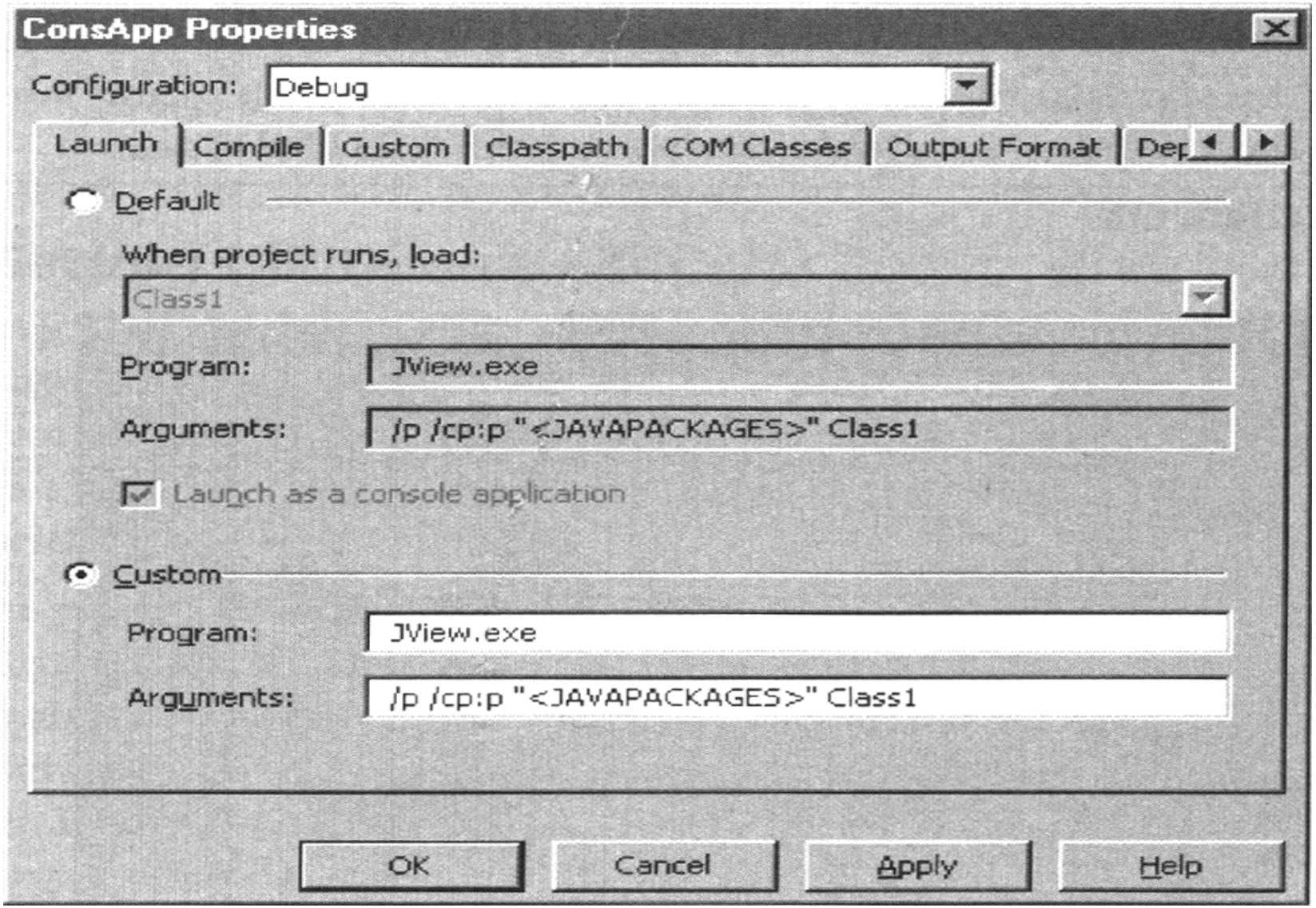


图 11-2　设置两个参数的启动文件

在一个解决方案里，可以存在多个项目，如果是这样，那么，在这个解决

方案开始时，运行其中的启动项目的启动文件。要设置启动一个项目，则在 Project Explorer 中用鼠标右击该项目，并选择设置 StartUp 项。

实验 11-1：从命令行运行应用程序

在这个实验中，将创建一个应用程序，返回显示命令行参数。

实验概述

在本实验中将练习：

- 在一个 Java 应用程序中访问命令行参数
- 从命令行中运行应用程序
- 设置 IDE 以提供命令行参数

实验准备

- 启动 Visual J++。

实验步骤

1. 创建一个新的 Windows Application 项目，命名为 ListParameters;
2. 命名窗体为 ListParameters.java;
3. 在窗体中加一个 ListBox 控件，命名为 lbParams;
4. 在 main 方法中，args 参数的数值为 lbParams;
5. 建立这个项目；
6. 打开 Command Prompt 窗口，进入到 ListParameters 目录中去；

7. 从命令行运行这个应用程序，注意 List Box 中的参数，并使用引号标记，设法生成一些嵌有空格的参数；
8. 从 IDE 运行这个应用程序，输入各式参数；
9. 在窗体中，用 ListView 控件代替 ListBox 控件，重复上面的操作，实现对参数进行计数；
10. 对工作结果与第 11 章\Sol11-1\Listparams.slm 中的解决方案进行比较。

下一步

应用程序的参数被用于处理命令行标志，为了指出这些标志，在前面加上连字符（-）或是斜线（/）。在下一节中，我们学习进一步更新应用程序，使之能够带命令行标志，允许参数列表发送到控制台窗口。

控制台 I/O

在前面一节中，我们学习了没有图形化用户界面的控制台应用程序。实际上，控制台应用程序以“控制台”命名，是因为应用程序的输入输出如同旧式的控制台和终端，界面中没有图形，只有文本行。

选择了一个 Java 文件作为启动文件时，在 Project Properties 对话框中的 Launch 选项卡中，Launch As A Console Application 复选框变为可用。选择这个复选框，应用程序开始运行，出现一个 MS-DOS 提示窗口，在窗口中显示输出，

并且提示用户输入。

如果在没有控制台窗口的情况下，应用程序设法执行控制台的 I/O，则会出现异常情况，其代号为 com.ms.wfc.io.WinIOException。

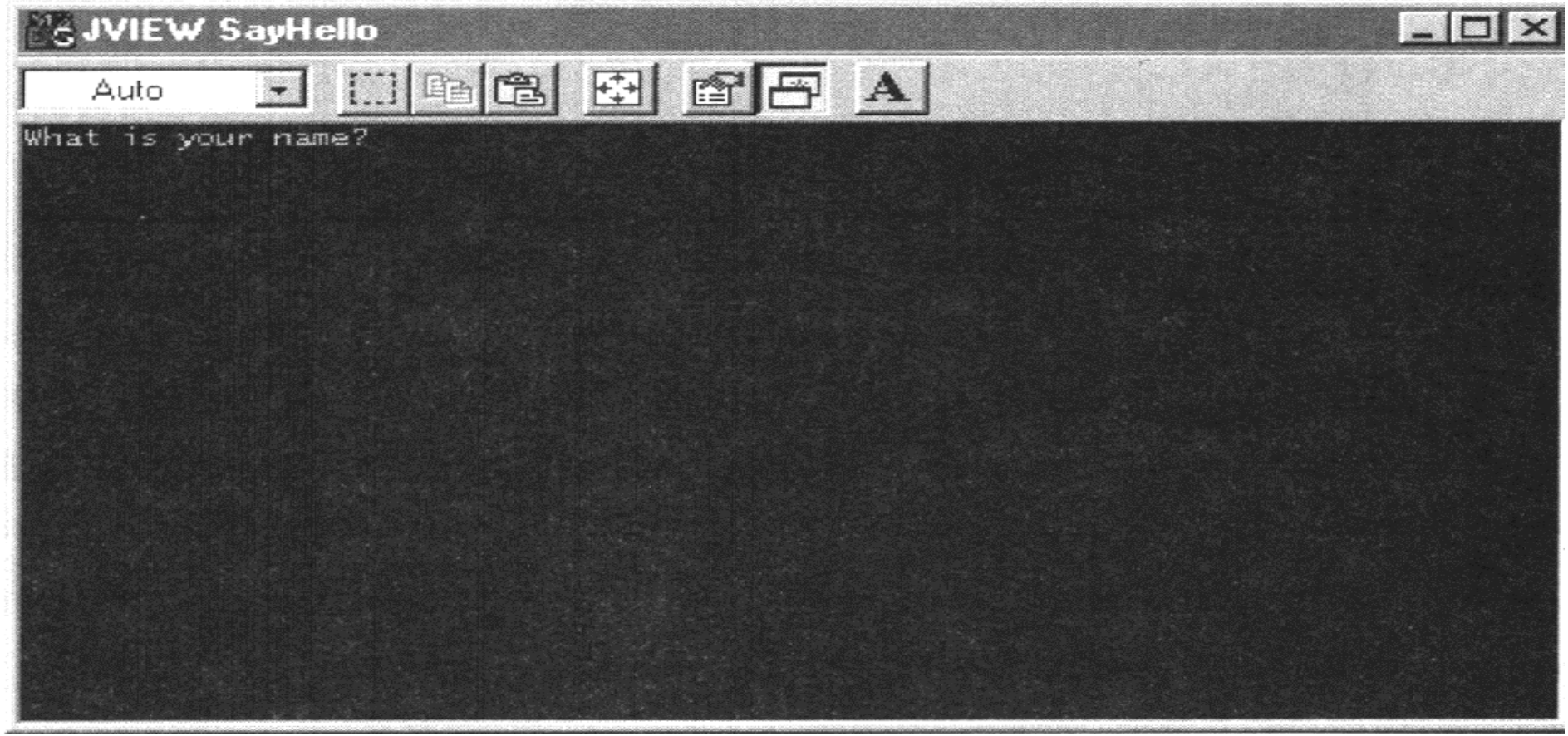


图 11-3　在 DOS 窗口中运行一个控制台应用程序

控制台 I/O 方法

com.ms.wfc.io 软件包支持控制台（文本）的输入输出。其输入输出有两个主要的接口，分别是 IReader 和 IWriter，在 com.ms.wfc.io 软件包内，也包括了执行这种接口的一些类。

读字符串

让我们先从控制台 I/O 的“I”开始，读控制台的输入。

- IReader 接口—用于简单访问字符和字符串

```
public interface IReader
{
  public void close();
  public int peek();
  public int read();
  public int read(char[ ] buffer, int index, int count);
  public String readLine();
}
```

在上面的程序中，没有指定所读内容的路径，因此，在执行 IReader 接口的类中，必须提供一个 open 方法，或者提供一个构造器，以用来读入。

- Reader 类—用于执行 IReader 接口，对从缓冲区中读字符和句子提供支持。它是类 TextReader 和 StringReader 的超类。
- TextReader 类—把 Reader 缓冲区关联到一个文件或者是其他的字节

流（byte stream）。TextReader 有两个构造器：

- 一个构造器把 Reader 缓冲区与字节流相联系，这个构造器使用 IByteStream 参数，例如 com.ms.wfc.io.File 对象。
- 另外一个构造器使用一个 String 类型的参数，这个参数作为一个文件名来打开一个文件，把 Reader 缓冲区关联到这个打开的文件中去。

- StringReader 类—这个类把 Reader 缓冲区与一个字符串关联，它有一个构造器使用一个 String 类型的参数。

写字符串

现在，让我们来看一下控制台 I/O 中“O”的部分，就是向控制台写。

- IWriter 接口—对转换“标准”类型为字符串并且向控制台写提供简单访问。注意许多重载函数。

```
Public interface IWriter
{
public void close();
public void flush();
public void write(char value);
public void write(char[] buffer);
public void write(char[] buffer,int index, int count);
public void write(boolean value);
public void write(int value);
```

```
public void write(long value);
public void write(float value);
public void write(double value);
public void write(String value);
public void write(Object value);
public void writeLine();
public void writeLine(char value);
public void writeLine(char[] buffer);
public void writeLine(char[] buffer, int index, int count);
public void writeLine(boolean value);
public void writeLine(int value);
public void writeLine(long value);
public void writeLine(float value);
public void writeLine(double value);
public void writeLine(String value);
public void writeLine(Object value);
 }
```

IWriter 接口描述了方法的数目。在写出数据的文本表达方式时，这是适当的。然而，对于读取数据来说，提供这样丰富的内容就不实际了。在读取时，问题是不能假定信息的精确性。最好是读取一个字符串，并使用分析方法，来尝试将字符串解释为数字或布尔值。

- Writer 类—实现 Iwriter 接口，并提供写向缓冲区的支持。它还处理对

值的 String 表达的转换。它是 TextWriter 和 StringWriter 的超集。

- TextWriter 类—使用文件或其他字节流来关联 Writer 缓冲区。TextWriter 有两个构造器：
  - 一个采用 IbyteStream 参数，例如 com.ms.wfc.io.File 对象，并使用字节流关联 Writer 缓冲区。
  - 另一个使用 String 参数作为文件名，来创建一个使用新创建的文件编写和关联 Writer 缓冲区的文件。
- StringWriter 类—使用 StringBuffer 对象关联 Writer 缓冲区。StringWriter 有两个构造器：
  - 一个采用 StringBuffer 参数。getStringBuffer 方法返回关联的字符串缓冲区。
  - 另一个是无参数的构造器，它创建一个新的 StringBuffer。getStringBuffer 方法返回关联的字符串缓冲区。

Text 类

com.ms.wfc.io.Text 类提供读取和写向控制台的简单访问：

```
public final class Text
{
public static final TextReader in ;
public static final TextWriter out;
public static final TextWriter err;

private Text()
```

```
}
static {
    in = new TextReader(File.openStandardInput());
    out = new TextWriter(File.openStandardOutput());
    out.setAutoFlush(true);
    err = new TextWriter(File.openStandardError());
    err.setAutoFlush(true);
}
 }
```

成员变量 in, out 和 err 与标准输入、标准输出和标准错误关联。在这里，是一个使用 in 和 out 读取和写向控制台窗口的例子。

```
Import com.mc.mfc.io.x
 Pulbic class SayHello
 {
public static void main(String[] args)
{
    Text.out.write(″ What is your name? ″ );
    String name = Text.in.readLine();
    Text.out.write(″ Hello ″ );
    Text.out.writeLine(name);
    Text.out.write(″ Press <enter> to end this application″ );
```

```
        name = Text.In.readLine();
    }
     }
```

此例的代码在 Chapter11-1\ConsoleIO\ConsoleIO.sIn 中。

实验 11-2：使用控制台 I/O

在本实验中，将要更新此应用程序，以便使用控制台 I/O 列出参数。

实验概述

在本实验中，将要练习：

- 解释命令行标志。
- 使用控制台方法。

实验准备

如果从实验 11-1 的现有解决方案中工作，就不需要进行设置。然而，你可能想从实验 11-2 提供的起始代码来开始，因为其中有有用的 TODO 注释：

1. 启动 Visual J++。
2. 打开 ListParam 项目（Lab11-1\ListParam.sIn）。

实验步骤

1. 使用如下格式添加名为 writeArgsToConsole 的方法。

```
void writeArgsToConsole(String[] args);
```

2. 在新的方法中，做下面的事情：
   - 打开标准输出。
   - 参数值写到标准输出，一次一行。
3. 在这个应用程序中找到 main 方法。
4. 在 main 方法中，检查第一命令行参数。
   如果是/c，使用命令行参数调用 writeArgsToConsole。从列表中删除标志/c。
   如果第一个命令行参数不是/c，则显示实验 11-1 的窗口版本。
5. 使用/c 标志运行这个应用程序，再不使用它运行应用程序。
6. 将结果与 Chapter11\Sol11-2\ListParams.sLn 中的比较。

选做实验

下面是可选实验部分：

- 包括错误检查，检查不能识别的命令行标志。
- 包括一个/help 命令行开关，以解释命令行标志和参数。

更多的控制流

读者已经见过了多种类型的控制流语句，在本节中，我们看一看 Java 语言中的其它控制流语句。

switch 和 break 语句

switch 语句可能是 Java 语言中最难处理的语句。switch 语句根据独立的值来进行分支处理。你可能会熟悉这个概念，因为在其他编程语言中也有这样的概念：Visual Basic 中的 switch 语句和 Pascal 中的 case 语句是类似的。

switch 语句看起来像：

```
    switch( discrete-expression )
{
 case value1:
// stuff-to-do
break;
 case value2:
// stuff-to-do
break;
 default:
// stuff-to-do
 }
```

```
// stuff-after-the-switch
```

switch 语句的开始是关键字 switch，随后是括号，括号内是分离的表达式。表达式可以是如下类型：

int
short
long
char
byte

再后是一个语句块。其中包含 case 标号。每个 case 标号都是关键字 case，然后是相应类型的值，以及一个冒号。控制流将跳转到与分离表达式匹配的 case 标号。case 标号后面的语句便执行。break 语句使控制流跳转到缺省的标号。缺省的标号匹配的是不具有 case 标号的任何值。default 标号是可选的。

我们来看看来自 Days 项目的例子。它在 Chapter11\Days\TwelveDays.sln 中：

```
String ordinal;
Switch(nDayNumber)
{
case 1:
ordinal = ″ first″ ;
break;
 case 2:
ordinal = ″ second″ ;
break;
```

```
 case 3:
ordinal = " third" ;
break;
 case 4:
ordinal = " fourth" ;
break;
 case 5:
ordinal = " fifth" ;
break;
 case 6:
ordinal = " sixth" ;
break;
 case 7:
ordinal = " seventh" ;
break;
 case 8:
ordinal = " eighth" ;
break;
 case 9:
ordinal =" ninth" ;
break;
 case 10:
```

```
ordinal = ″ tenth″ ;
break;
 case 11:
ordinal = ″ eleventh″ ;
break;
 case 12:
ordinal = ″ twelfth″ ;
break;
 default:
ordinal =″ oops″ ;
 }
```

在 switch 语句中包括 break 语句是非常重要的。在下一节中，将看到 break 语句如何改变 switch 语句的行为。

switch 语句和“直落”

对于 switch 语句来说，break 语句非常重要。在遇到另一个 case 标号时，控制流不会自动跳转到 switch 语句的结束处。实际上，获得与一组语句关联的多个值的方法是，在语句中放多个 case 标号。例如，在如下的 switch 语句中，从值 4 到 12 将会使执行的语句是相同的。

```
    Switch(nDayNumber)
{
case 1:
```

```
output.write(″ lst″ );
break;
 case 2:
output.write(″ 2nd″ );
break;
 case 3:
output.write(″ 2nd″ );
break;
 case 4:
 case 5:
 case 6:
 case 7:
 case 8:
 case 9:
 case 10:
 case 11:
 case 12:
output.write(nDayNumber);
output.write(″ th″ );
break;
 }
```

对于 4 到 12 的分支，有 9 个 case 标号。它们都应用于相同的三个语句：

- output.write(nDayNumber);
- output.write("th");
- break;

有时，这种特性叫做直落（fall-through），因为控制流“直落”到下一个 case。下面是另一个“直落”情况的例子：

```
 switch(nDayNumber)
 {
 case 12:
output.writeLine(″  Twelve drummers drumming, ″ );
sleep(3 * pulse);
 case 11:
output.writeLine(″  Eleven pipers piping, ″ );
sleep(3 * pulse);
 case 10:
output.writeLine(″  Ten lords a-leaping, ″ );
sleep(3 * pulse);
 case 9:
output.writeLine(″  Nine ladies dancing, ″ );
sleep(3 * pulse);
 case 8:
output.writeLine(″  Eight maids a-milking, ″ );
sleep(3 * pulse);
```

```
 case 7:
output.writeLine("   Seven swans a-swimming, " );
sleep(3 * pulse);
 case 6:
output.writeLine("   Six geese a-laying, " );
sleep(3 * pulse);
 case 5:
output.writeLine("   Five gold rings, " );
sleep(3 * pulse);
 case 4:
output.writeLine("   Four calling birds, " );
sleep(3 * pulse);
 case 3:
output.writeLine("   Three French hens, " );
sleep(3 * pulse);
 case 2:
output.writeLine("   Two turtle doves, " );
sleep(3 * pulse);
 case 1:
output.writeLine("   And a partridge in a pear tree. " );
sleep(3 * pulse);
 }
```

其他循环语句

在 Java 中，有几种循环方式：先测试，后测试和计数方式。我们将依次介绍每一种方式，并加以比较：

先测试

在第八章中，我们学习了 while 循环，它使用的格式为：

```
    while( true-false-check )
  {
// stuff-to-do
  }
    // stuff-after-the-loop
```

这个简单的循环在每次循环的开始处进行检查。如果检查结果为真，则应用程序做 stuff-to-do 部分，并再次循环回到检查的前面。如果检查结果为假，则应用程序移到 stuff-after-the-loop 部分。

后测试

Java 中的 do 循环与 while 循环非常类似：

```
  Do
  {
// stuff-to-do
  } while( true-false-check );
```

```
// stuff-after-the-loop
```

这个简单的循环在每次循环的结束处进行检查。应用程序执行 stuff-to-do 部分，然后进行检查，如果结果为真，则应用程序返回循环的前面，再执行 stuff-to-do 部分，并再次检查。如果检查结果为假，则应用程序移到 stuff-after-the-loop 部分。

注意： do 循环至少执行一次 stuff-to-do 部分。在 while 循环中，如果检查结果为假，将完全跳过 stuff-to-do 部分。

计数

for 循环比其他两个循环更复杂些。通常，for 循环可以以许多方式来使用。使用 for 循环最简单和最常见的例子是作为计数循环：

```
for( initialization ; true-false-check ; update )
{
// stuff-to-do
}
    // stuff-after-the-loop
```

在 for 循环中，与在 while 循环中一样，在 stuff-to-do 之前进行检查。然而，在第一次进行真假判断检查之前，会进行初始化。我们会看到，这有一点复杂。我们使用一个例子，将这点说明得更清楚些。下面是一个 for 循环，它从 0 到 9 计数。

```
    Output.writeLine(″ Starting to count″ );
```

```
For(int nCounter = 0; nCounter < 10: nCounter ++)
{
output.writeline(nCounter);
}
    output.writeLine(″ All done. ″ );
```

在写出了“Starting to count”后，便启动了 for 循环。发生的第一件事情是初始化：

```
int nCounter=0
```

创建一个名为 nCounter 的临时变量。它具有初始值 0。下面，将进行真假判断检查：

```
nCounter < 10
```

这个结果是真，所以，便执行 stuff-to-do 部分。

```
output.writeLine(nCounter);
```

在 stuff-to-do 之后，将进行更新：

```
nCounter ++
```

现在，nCounter 从 0 更新为 1。而且，进行真假检查判断：

```
nCounter < 10
```

所以，我们继续 stuff-to-do 部分。最终，nCounter 是 9，stuff-to-do 打印出

9。在进行更新时，nCounter变成10。检查失败：

```
nCounter < 10
```

而且，应用程序继续stuff-after-the-loop部分：

```
output.writeLine("All done.");
```

注意：nCounter的声明在for循环内部。在这个循环结束时，nCounter不再可用。这与C++中的for循环不一样。

比较循环例子

这里是从0到9的例子，再次使用for循环。

```
Output.writeLine(″ Starting to count″ );
For(int nCounter = 0; nCounter < 10; nCounter ++)
{
output.writeLine(nCounter);
}
output.writeLine(″ All done. ″ );
```

下面是while循环，也是从0到9计数：

```
output.writeLine(″ Starting to count″ );
int nCounter = 0;
while(nCounter < 10)
{
output.writeLine(nCounter);
```

```
nCounter ++;
 }
 output.writeLine(″ All done. ″ );
```

注意，nCounter 的声明在 while 循环的外面。还有，更新（nCounter++）是在循环括号的内部。在这个例子中，nCounter 在循环的结束处不出现。

下面是一个 do 循环，从 0 到 9 计数：

```
 output.writeLine(″ Starting to count″ );
 int nCounter = 0;
 do
 {
output.writeLine(nCounter);
nCounter ++;
 } while(nCounter <10);
 output.writeLine (″All done.″);
```

与 while 循环一样，nCounter 在循环的外面声明。在循环完成时，它还可以使用。

实验 11-3：观察控制台应用程序：The Twelve Days of Christmas

这是一个简单的实验，它使用 switch 语句来进行通知。在本实验中，我们将检查简单的控制台应用程序，这个程序给出歌曲“The Twelve Days of Christmas”中的歌词。

实验概述

在本实验中，你要：

- 查看 switch 语句的例子。

实验准备

1. 启动 Visual J++。
2. 打开 TwelveDays 项目（Lab11-3\TwelveDays.sln）。

实验步骤

1. 检查应用程序中的代码，注意 startVerse 和 listGifts 方法。

   这个应用程序有一些 sleep 语句。这些语句暂停输出，这样，它将与唱这首歌的人同步。暂停与成员变量 pulse 为基础，它给出每一节拍的毫秒数。它设置为 500，所以，每一节拍都是半秒。
2. 运行应用程序。注意两个 switch 语句的行为。
3. 删除 startVerse 中 switch 语句中的 break 语句，看看会发生什么事情。
4. 删除 listGifts 中 switch 语句中的 break 语句，看看会发生什么事情。
5. 使用不同的循环语句，重新编写代码。
6. 将工作结果与 Chapter11\Sol11-3\TwelveDays.sln 中的解决方案进行比较。

第十二章　使用 MFC 中的外部组件

Microsoft Windows 为编程提供了丰富的外部环境，利用 WFC（Windows Foundation Class，Windows 基本类）库，Java 编程可以使用绝大多数的外部环境，但不完全是。

在 Windows 里，访问其他资源可以通过不同的方法来实现。你可以使用 COM（Component Object Model，组件对象模型），这是多数 Windows 应用程序的通信结构。例如，Microsoft Visual J++由许多不同的部分组成，这些部分互相之间的通信就是使用了 COM。COM 同时使你的应用程序和其他应用程序互相作用，如 Microsoft Word 和 Microsoft Excel 之间的作用。COM 可以让你访问 Windows 中许多以前就建好的组件。

在这方面，另一端是 Win32 的 API（application programming interface，应用程序编程接口），这是一套底层函数，API 函数是每个 Windows 程序的基础。J/Direct 使应用程序可用这些底层函数。

在这章中，将学习如何：

- 使用 J/Direct 调用 API 函数
- 在设计中，向工具箱加要使用的组件
- 使用 COM 组件

本章中，将要：

- 创建一个应用程序，播放不同的系统声音功能。

通过在本章的学习，你可以对这个应用程序使用不同的技术，创建几种不同的版本。

通过 Win32 API 工作

所有那些为 32 位 Windows 编写的应用程序的基础是 Win32 API，这是一种通用的 API，应用于 Windows 95、Windows 98、Windows NT 操作系统和其他的后继操作系统。Win32 API 基于以前 16 位 Windows 的 Windows API，所以，Win32 API 的许多函数和结构都可以追溯到 Windows 1.0 或者 Windows 3.1。Win32 API 应用很灵活，但有些复杂。许多程序员使用函数库、应用程序结构和工具来简化建立 Windows 应用程序的操作，但这些结构可能不能支持所有 API 的丰富特性。

WFC 是用 Win32 API 写成的，在这本书中使用的所有 WFC 窗体和控件，都是建于 Win32 窗口方案的高层。在 Visual Studio 中，其他的应用程序结构包括：

- Visual C++中的 MFC
- Visual Basic 中的 Ruby 窗体

普通操作都可以让这些结构来处理，在没有碰到具有特殊性质的应用程序时，这都可以。但有时候，你也可能想要一些东西，而这些东西不能被结构所支持。

使用 J/Direct Call Builder

J/Direct 可以让你声明一个本机方法，并且从对应的源文件中调用；在 Java 中，由 VM（Visual Machine，虚拟机）实现所有种类的转换。使用 J/Direct，Java 应用程序能够方便地访问所有的 Win32 API 系统功能，或者是 Win32 任意的 DDL（dynamic link library，动态链接库）。

让我们来看一个例子，MessageBox 类和它的 show 方法是与 Win32 的 MessageBox 函数相关的，下面是重载 com.ms.wfc.ui.MessageBox 的 show 方法：

```
public static int show (String text, String caption, int style);
public static int show (String text, String caption);
public static int show (String text);
```

第 1 种调用方式参数输入是完全的，第 2 和第 3 种方式调用中的一些参数使用默认值。

下面是 Win32 的 MessageBox 函数的原型，从 C 或者 C++语言编写的 Windows 的头文件中取得：

```
WINUSERAPI int WINAPI MessageBox(
    HWND hWnd, LPCSTR lpText, LPCSTR lpCaption, UNIT uType);
```

你可以看到，show 的第 1 种调用方式是如何把参数传递到这个 C 语言的函数的。J/Direct 可以直接访问这个函数，为这个方法它使用了特殊的标记，在 JavaDoc 注释中：

```
/**
*@ dll.import (“USER32”, auto)
```

```
*/
public static native int MessageBox (
    int hWnd, String lpText, String lpCation, int uType);
```

这个 JavaDoc 注释标记指示这个函数是 USER32 DLL，并且，后面跟随的参数种类是自动映射的。为了帮助建立这个导入的声明，Visual J++中有一个工具窗口 J/Direct Call Builder，它可以进入到 IDE（Integrate Design Environment，集成设计环境）中，并且，这个窗口可以停靠到其他工具窗口的侧面。J/Direct Call Builder 允许在你的项目中加入 API 函数的声明。

打开 J/Direct Call Builder

1. 在 View 菜单中，选中 Other Windows 项。
2. 在 Other Windows 菜单中，选中 J/Direct Call Builder 项后，就出现了 J/Direct Call Builder 窗口。

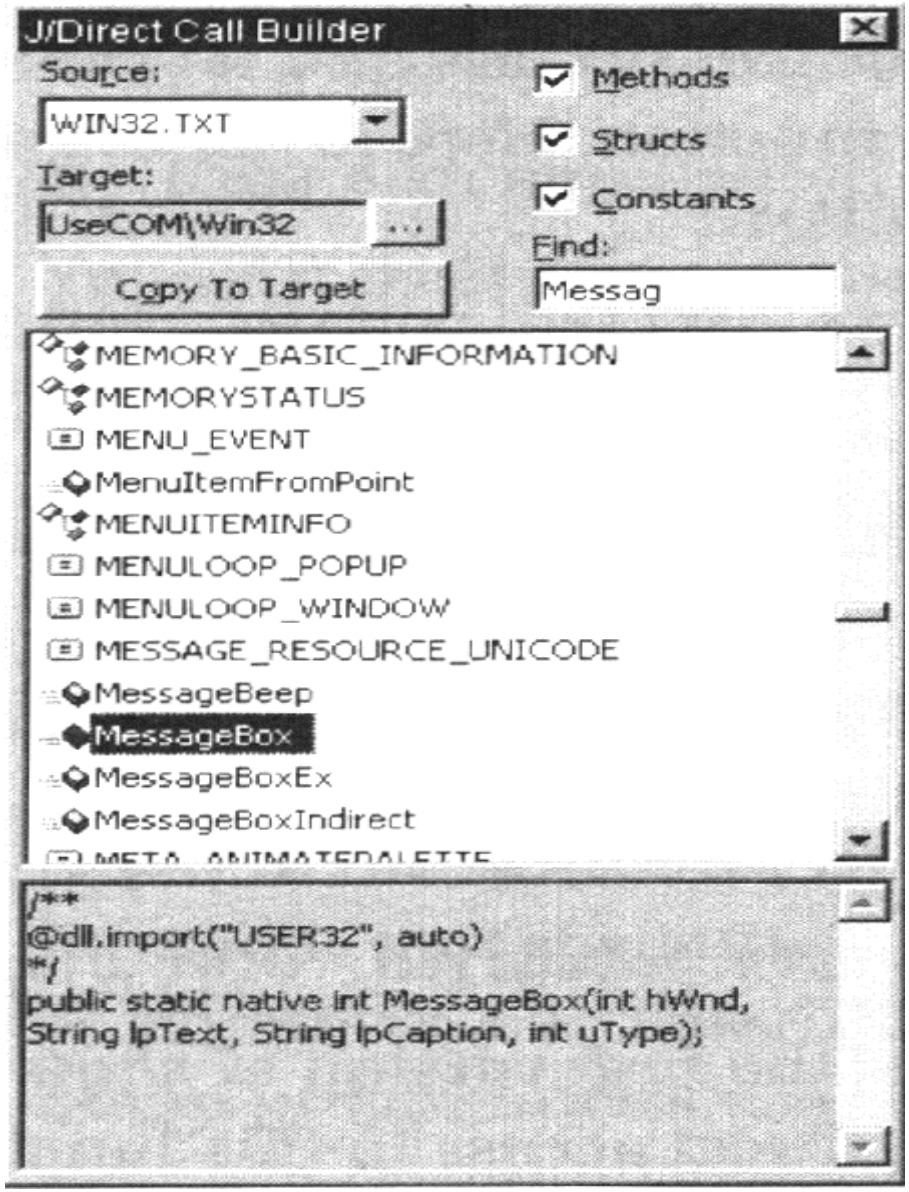


图 12-1　J/Direct Call Builder 窗口

通常，使用 J/Direct Call Builder 工具加入对 Win32 API 函数的访问，它往一个类中加方法的声明。在缺省情况下，这种方法加到了公共的类，在默认的软件包中，是 Win32 公共的类。如果它们不存在，J/Direct Call Builder 就创建了这个类（或者是文件）。你也可以使用 J/Direct Call Builder 来加一个常数（static final 的变量）和结构（需要类）的声明。

在项目中加入 Win32 API 资源的声明

1. 打开 J/Direct Call Builder 窗口；
2. 选择导入的条目，利用 Shift 键选择相邻的条目，用 Ctrl 键选择不相邻的条目；
3. 选择复制的对象，默认目标是在默认软件包中的 Win32 类。
4. 单击 Copy To Target。

为了便于在 J/Direct Call Builder 窗口中寻找条目，可以在 Find 编辑控件中输入条目名的起始部分。

下面，我们来创建一个调用 Win32 的 MessageBox 函数的样例，在这个例子中，只调用带文字的函数，其文字在一个 WFC 窗体中由用户提供。这个例子的代码存放在 Chapter12\MessageBox\ MessageBox.sln 中。

在导入 MessageBox 函数和一些相关的常数后，这儿有一些 Win32.java 的内容：

```
public class Win32
{
    /**
```

```
    *@ dll.import ("USER32", auto)
    */
    public static native int MessageBox (
        int hWnd, String lpText, String lpCaption, int uType);
    public static final int MB_YESNO = 0x00000004;
    public static final int IDNO = 7;
    public static final int IDYES = 6;
}
```

下面是调用 MessageBox 的事件处理程序方法：

```
private void btnShow_click (object source, Event e)
{
    String msg;
    msg = edtMessage.getText( ) + "\n\nDo you want to continue?";
    int response;
    response = Win32.MessageBox(
        this.getHandle( ), msg, edtCaption, getText( ), Win32.MB_YESNO);
    if (response == Win32.IDNO)
    {
        Application.exit( );
    }
}
```

注意： 用 MessageBox.show 可以简单地实现上面的功能，例子中只是示

范了如何使用 J/Direct 和 J/Direct Call Builder。

实验 12-1：MessageBeep

在这个实验中，我们通过调用 MessageBeep 来使用 Win32 API，创建一个应用程序用来播放系统的声音。MessageBeep 只使用一个参数：需要播放的声音，其参数的数值（参考下面的列表）与前面例子 MessageBox 的标志相联系，而且，它们同时也与 Windows 的声音关联：

- MB_OK — 默认声音
- MB_ICONERROR— 发生错误的声音
- MB_ICONQUESTION — 提问的声音
- MB_ICONEXCLAMATION —感叹的声音
- MB_ICONASTERISK — 星号的声音

在 Control Panel 的 Sounds 这一项中，设置这些系统声音：

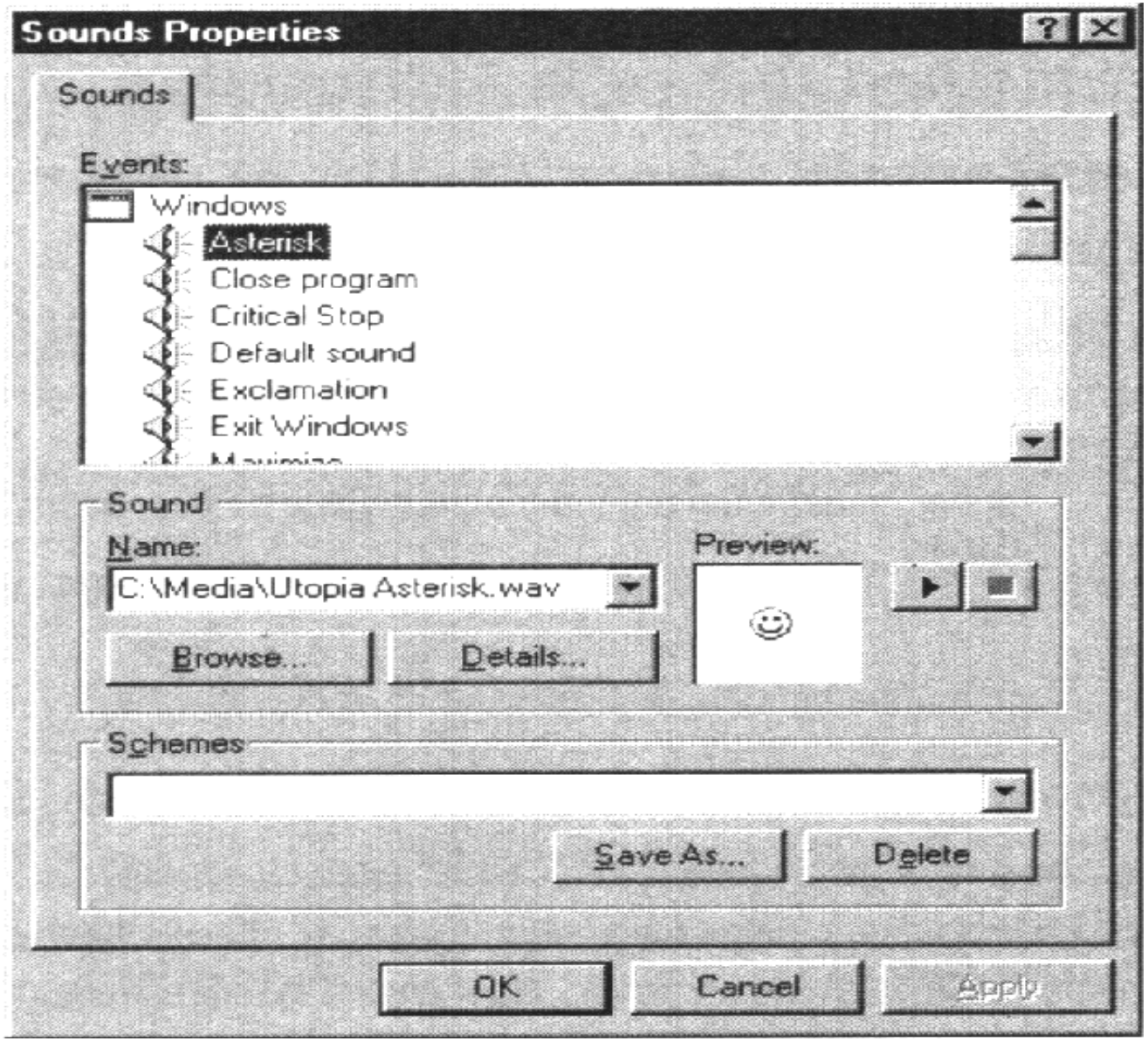


图 12-2　Sounds Properties 对话框

实验概述

本实验中的练习有：

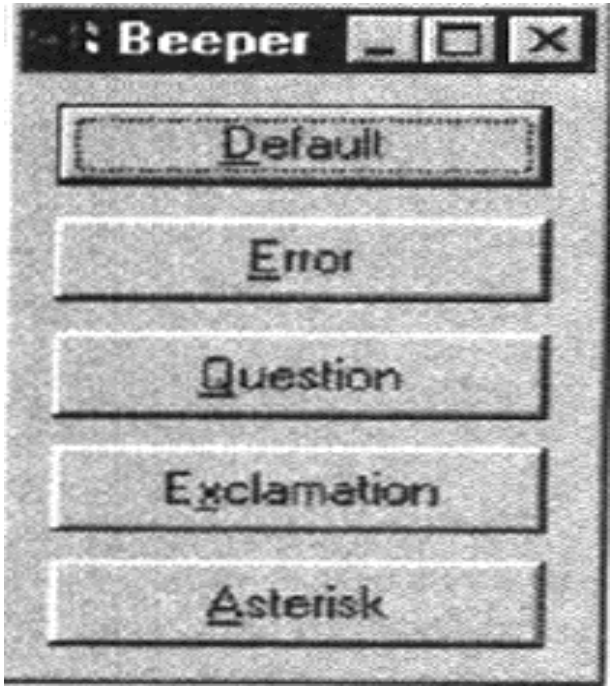
- 使用 J/Direct 以访问 Win32 API
- 使用 J/Direct Call Builder 窗口

实验准备

1. 启动 Visual J++。
2. 创建一个新的 Windows 应用项目，取名为 Jdirect。

实验步骤

1. 为应用程序创建如下图所示的用户界面：



2. 使用 J/Direct Call Builder 往应用程序中加声明，把声明加到 Win32.java。这些声明包括：

   MessageBeep
   MB_OK
   MB_ICONERROR
   MB_ICONQUESTION
   MB_ICONEXCLAMATION
   MB_ICONASTERISK

3. 在按钮的单击事件处理程序中，调用 MessageBeep 函数，注意使用正确的参数；

4. 测试应用程序，并把实验结果和位于 Chapter12\Sol12-1\JDirectBeep.sln 中的解决方案比较。

选做实验

这儿有一些可供选做的附加操作：

- 修改用户界面，使用一个列表对话框提供声音的选择，加一个播放的按钮。
- 修改列表对话框，实现在双击响应中播放声音。

下一步

使用 Win32 API，我们能够自由地使用 Windows 的所有属性，但是，API 级别的编程是非常复杂的。程序员经常使用已建好的库、结构和组件，在下一节中，将学习在 WFC 项目中使用附加的控件。

利用附加控件工作

Win32 API 提供了 Windows 的低级访问，普遍的做法是把 API 的使用打包加到组件中去，然后把有 API 的组件打包成控件，以易于使用。迄今为止，我们放在 WFC 窗体中的所有的控件都来自于 com.ms.wfc.ui 包，这些控件包括了基本的 WFC 结构，另外，WFC 同时也支持其他控件。

在这节中，我们要学习使用 WFC 窗体中的其他控件：ActiveX 控件和其他

的 WFC 控件。

往工具箱中加 ActiveX 控件

ActiveX 控件为额外范围的应用程序和函数提供了组件级别的支持，它们主要在 Visual Basic 应用程序中的主要的构件之间，也叫做 OLE 控件或 OCX。

下面，我们来创建一个使用 ActiveX 控件的简单应用程序。在这个例子中，我们使用 Masked Edit 控件，这个例子代码位于 Chapter12\UseActiveX\UseActiveX.sln。如果 Masked Edit 控件没有在系统中注册，那么这个样例将不能工作，你需要注册 MSMASK32.OCX 文件。

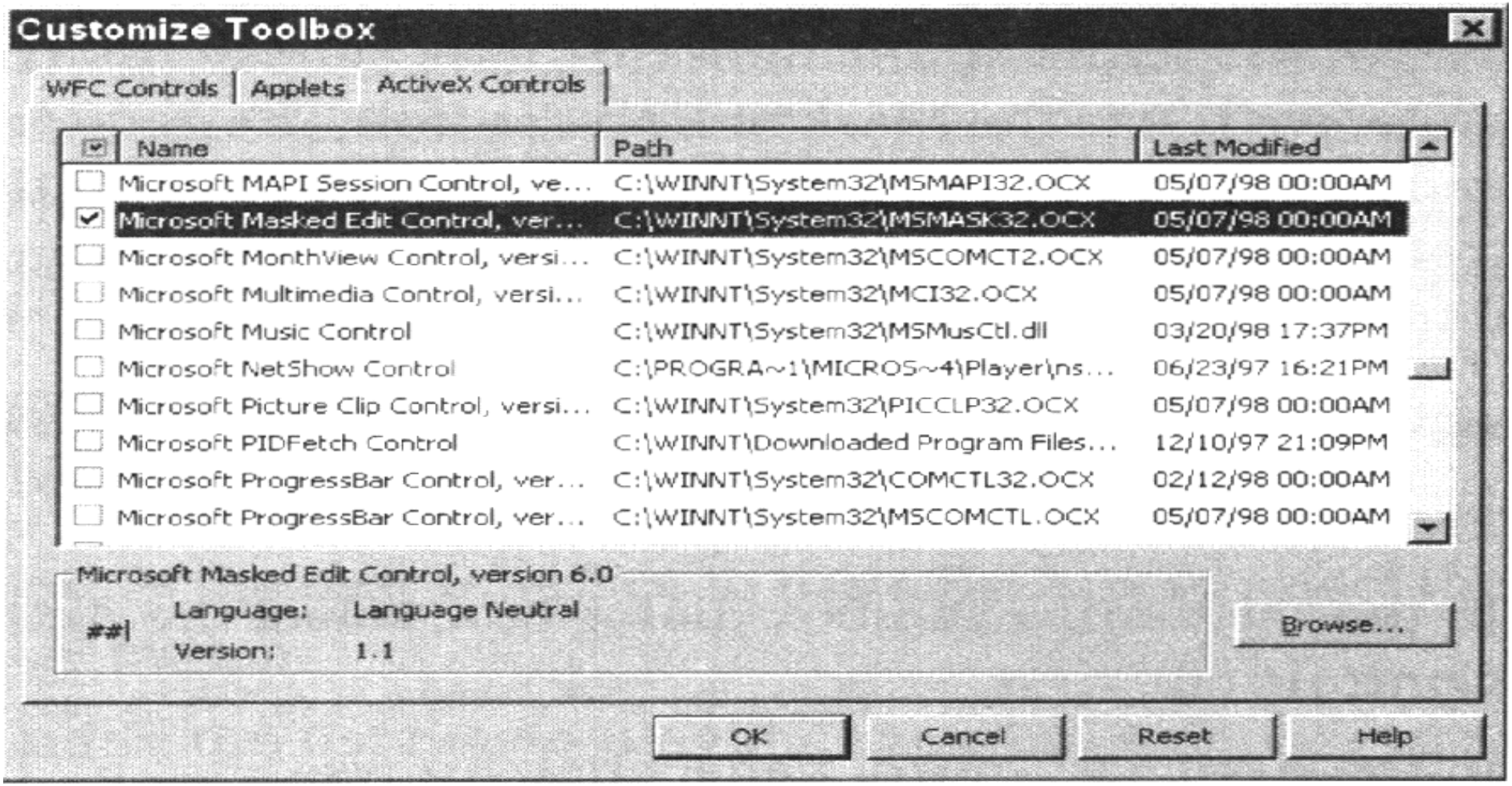
注册 ActiveX 控件并且把它加到工具箱上

1. 从 Tools 菜单中，选中 Customize Toolbox 项；
2. 在 Customize Toolbox 的对话框中，选择 ActiveX 控件的选项卡；
3. 在 ActiveX 控件窗格中，单击 Browse 浏览；
4. 在 Browse 对话框中，设法找到控件文件，选中文件，单击 Open，对话框关闭；
5. 单击 OK。

如果 ActiveX 控件已经进行了注册，把它加到工具箱中的简单方法如下：

1. 从 Tools 菜单中，选中 Customize Toolbox 项；
2. 在 Customize Toolbox 的对话框中，选择 ActiveX Control 选项卡；

3. 在控件列表中，选中对应控件的复选框，就可以把所需控件加到工具箱；



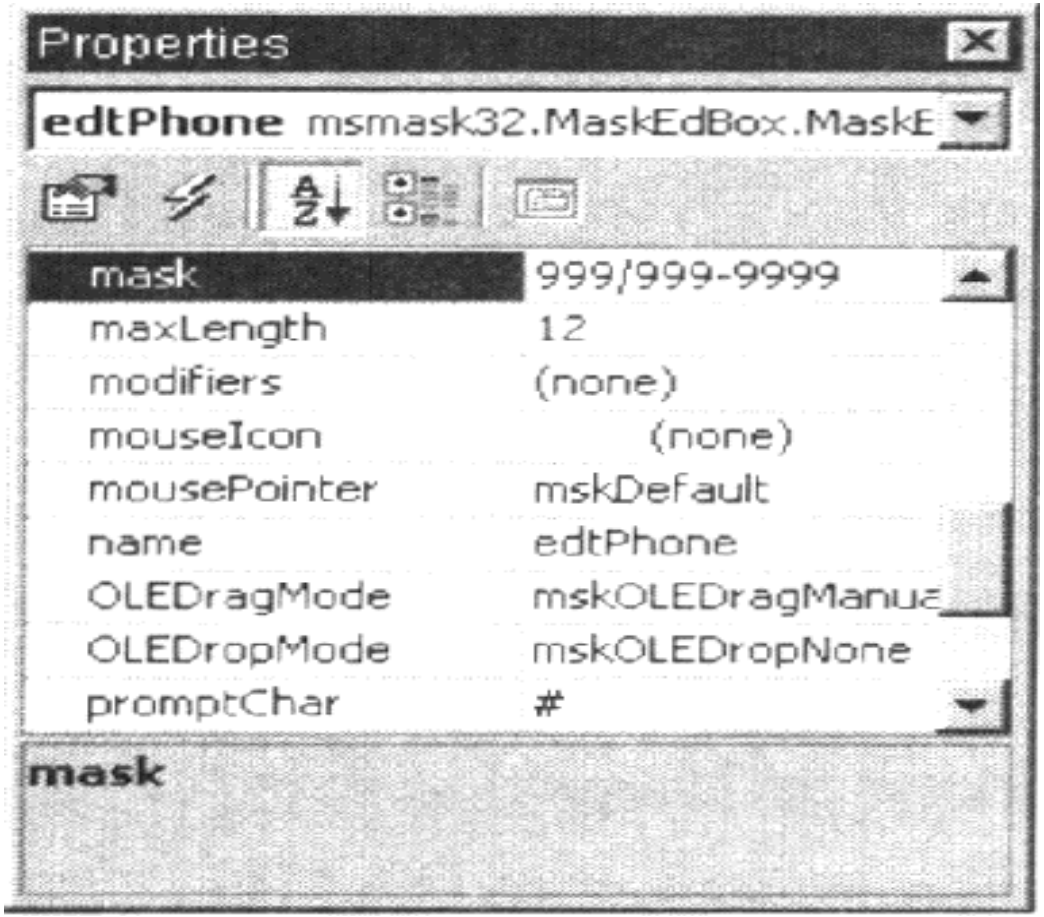
4. 单击 OK。

把 ActiveX 控件加入到工具箱后，可以把它加到 WFC 窗体中，使用方法和其他控件完全一样。例如，可以使用一个数字掩码控件来做一个电

话号码编辑框。

把 ActiveX 控件加到项目后，就产生了 ActiveX 控件的包装类。包装类把项目浏览窗口也进行了更新。如果要有更多的包装类，参见本章后面的内容“COM 包装类”。

包装类使得 ActiveX 控件的属性、方法、事件在 Java 中可以使用，你可以在属性窗口中，设置控件的属性和事件启动的处理程序。



IntelliSense 提供了完全自动的方法调用。

基本上，不能区分在应用程序中使用的 WFC 控件和 ActiveX 控件。

把 WFC 控件加到工具箱中

在 Visual J++中，除了支持 ActiveX 控件外，还支持往工具箱中添加新的 WFC 控件。

往工具箱添加 WFC 控件

1. 在 Tools 菜单中，选中 Customize Toolbox 项。
2. 在 Customize Toolbox 对话框中，选择 WFC Control 选项卡。
3. 在控件列表中，选择要加到工具箱中去的控件。
4. 单击 OK。

注意： 在类路径中的 WFC 控件，必须是可用的。

在把 WFC 控件加到了工具箱后，就可以像工具箱中其他控件一样来使用它了。

实验 12-2：WFC MessageBeep

在本实验中，我们要使用 WFC 控件来播放 MessageBeep 项目 。在实验 12-1 中，我们创建了 MessageBeep 控件。在这儿，这个控件项目是 Visual J++解决方案的一个部分，所以需要把新的控件放到类路径中去。在其他时候，我们可能必须往类路径加一个包，使一个控件可用。更多的信息请查看第十章的“把目录加到类路径”。

实验概述

实验中需要练习：

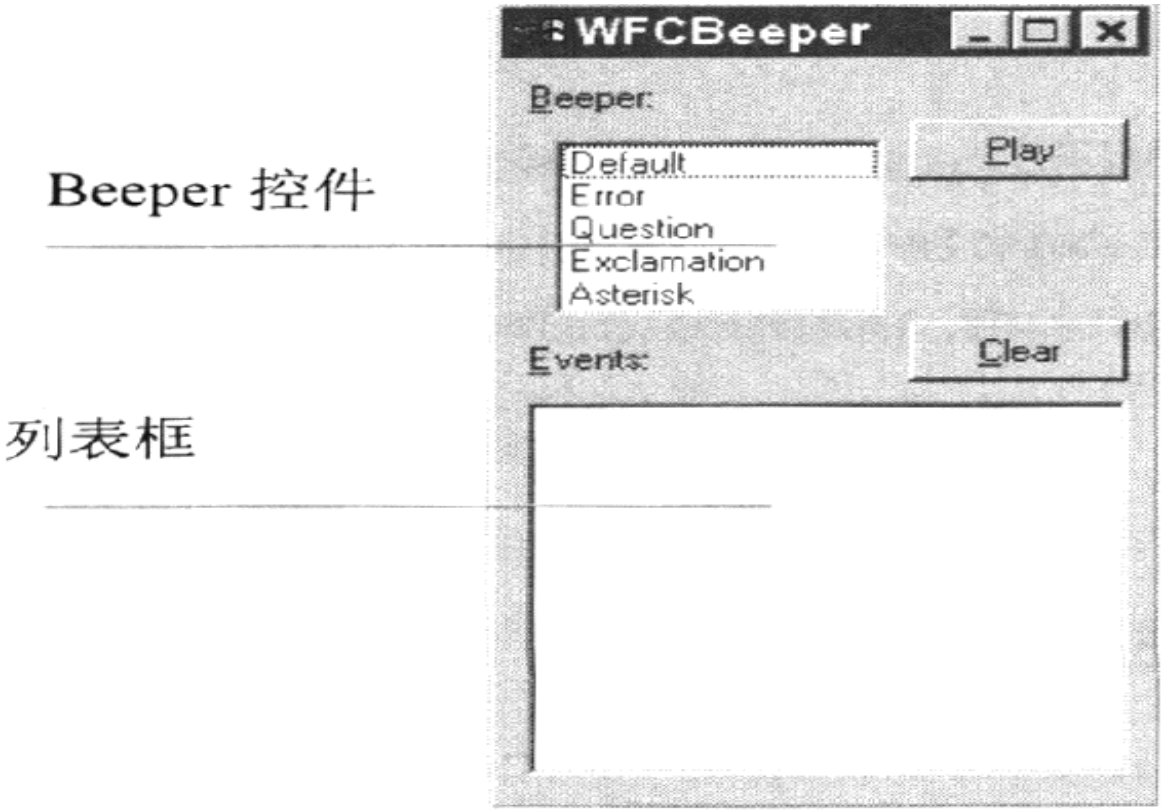
- 往工具箱中加一个新的控件；
- 使用自定义的 WFC 控件

实验准备

1. 启动 Visual J++；
2. 打开位于 Chapter12\Lab12-2\NewControl.sln 中的项目 NewControl。

实验步骤

1. 在 Project Explorer 窗口中，展开 BeeperControl 项目；
2. 在 Designer View 中打开 Beeper.java 程序；
3. 在这个窗体中，创建下面的用户界面：



4. 对于 Play 按钮的 Click 事件添加一个处理程序，在这个处理程序中调用 Beeper 的 play 方法；
5. 在 Beeper 的已播放事件中加入一个处理程序，这个处理程序对播放的事件在列表框中进行记录，这个记录应该指明发生的事件和当前的 Choice 属性的数值；
6. 给 Beeper 的已更改事件加一个事件处理程序，这个处理程序在列表框中记录 Change 事件，并使得该记录信息和已播放的处理程序的记录相似；
7. 对 Clear 按钮的鼠标单击事件 Click 添加一个处理程序，这个程序清除列表框中所含内容；
8. 测试应用程序，并把实验结果和 Chapter12\Sol12-2\NewControl.sln 进行比较。

选做实验

在这里有一些可选做的附加操作：

- 在这个窗体中加入多个按钮，以选择播放的声音。
- 在这个窗体中加入其他 WFC Beeper 控件。

下一步

ActiveX 控件和 WFC 控件能够支持许多复杂的过程，使用这些控件的典型方法就是利用 Designer 把这些控件加入到窗体中去。但是，有许多其他组件不是基于窗体的，在下一节中，我们要学习使用不基于窗体的组件 COM。

使用 COM 组件

COM 是组件交互的模型和二进制标准。COM 在 OLE、ActiveX 和自动功能的通信层，COM 对象形成了 Windows 应用程序不断发展的数量的基础。实际上，在 Visual J++ 6.0 中，有许多使用 COM 进行通信的独立组件，而 Visual J++6.0 就是由这些单独的组件组成的。

COM 和 Java

COM 是二进制的组件通信标准，所以，对于经常访问 COM 对象的机制，必须在程序语言和程序平台间进行统一。为了做到这一点，COM 使用 COM 接口和共存类（coclass）。在使用 COM 接口和共存类的 COM 结构和 Java 的结构之间，有一些类似的地方。

注意：这节讲述利用 COM 对象工作的一些基础，这里的许多信息都不是直接用于 Visual J++的，但这些都是作为背景材料。

COM 接口和共存类

与 Java 接口一样，COM 接口是基本的。它列出了函数和方法，在 COM 接口中，没有与函数关联的实现信息，有关 Java 接口的更多信息，请参看第八章中的“接口”。

共存类是一个组件对象类，共存类是在 COM 中的一个可创建的实体。一个实体执行一个或多个 COM 接口，所以，共存类就类似于一个 Java 类，这个

Java 类执行一个或多个接口。

创建对象

我们使用一个共存类来创建 COM 对象。创建 COM 对象时，返回一个接口指针给新的对象。在 Java 的术语中，这是一个接口引用。这相当于 Java 中的 new 运算符。

访问对象

我们有了一个 COM 对象的接口指针后，可以使用指针访问这个对象。与 Java 不同的是，我们不能在类中使用引用，去访问这个 COM 对象。这一点很重要。

COM 对象的寿命

COM 对象一直保持到没有什么再使用它为止，也就是说，直到没有任何对象还有这个对象的引用为止。这个称为引用计数。在 COM 中，引用计数使用 AddRef 和 Release 方法，为了增加一个 COM 对象的引用数，需使用 AddRef 方法，要减少一个 COM 对象的引用数，需调用 Release 方法。

创建 COM 对象时，把 COM 对象的引用数设置为 1，当该 COM 对象的引用数变成 0 时，这个 COM 对象被破坏。

在 Java 中，引用数由 VM 处理，VM 跟踪与每一个 Java 对象相关联的赋值和参数传递。当一个对象不再保留引用时，这个对象就可以进行垃圾收集了。关于 Java 的引用数和垃圾收集的详细资料，请参看第四章“对象的引用”。

多接口

一个 COM 对象可实现多个 COM 接口，实际上，如我们所见，这也是正常的。为了得到不同接口的指针，可以调用 QueryInterface 方法。

在 Java 中，一个对象能够实现多个接口，类型转换可以使一个对象得到不同的引用类型。详情请参看第九章的“类型转换”。

IUnknown 接口

我们在上面学习了三个函数，它们是形成每个 COM 对象的基础。它们都集中在 COM 接口 IUnknown 中，这个接口是所有 COM 接口的根接口。在最低限度，一个 COM 接口必须实现 IUnkown 接口。IUnkown 除了引用计数和访问其他 COM 接口外，不提供其他操作。这个接口等同于 Java 中的终极超类 java.lang.Object，所以，每个 Java 对象都是一个 java.lang.Object。java.lang.Object 类支持的功能比 IUnknown 稍微多一点，但也多不到哪儿去。有关 java.lang.Object 的其他信息，请参考第四章中的“java.lang.Object 类”。

查找类和接口

在 COM 中，所有的共存类和 COM 接口都在系统注册表中注册。如果一个共存类或 COM 接口没有在注册表中注册，则不能为 COM 对象所用（注册表是一个数据库，包含了有关系统的设置信息：硬件、软件和个人的用户设置。可以使用 REGEDIT.EXE 查看注册表）。

要注册一个自注册的 COM 对象，需运行 REGSVR32 程序，把控件的文件名作为命令行参数。在 Windows 95 中，REGSVR32 在 Windows 安装目录中的

SYSTEM 子目录中；在 Windows NT 中，REGSVR32 在 Windows 安装目录中的 SYSTEM32 子目录中。

> **注意：**在查看系统注册表时，不要改变其中的设置，REGEDIT 可以随时更改注册表，所以，在进行了改变操作后，不可能把操作撤销。或者是不保存退出。被更改的注册表可能会使系统不能正常工作，或者甚至是不能启动。

通过类路径，VM 可以找到 Java 类，有关详细介绍请参考第十章的“类路径”。

COM 包装类

我们在上面可以看到，COM 对象通过使用共存类和 COM 接口而创建，Java 的 VM 已经嵌入了对 COM 对象的支持。所以，使用 COM 对象工作时，我们只需要使用一个 Java 的方法来引用共存类和 COM 接口。有一些特殊的 Java 类用来引用共存类和 COM 接口，它们使用特殊的类，这些类在 VM 中被认为是共存类和 COM 接口的包装。

添加包装类

按下面的步骤，注册一个 COM 组件，并且往项目中加入包装类：

1. 在 Project 菜单中，选中 Add COM Wrapper 项。
2. 在 COM Wrappers 对话框中，单击 Browse。
3. 在 Select COM Componet 对话框中，找到所需文件，单击 Open，以

选择组件文件，并关闭对话框。

4. 单击 OK。

如果 COM 组件已经注册，对于得到包装类的操作，可以稍微简单一点：

1.在 Project 菜单中，选中 Add COM Wrapper。

2.在 COM Wrappers 对话框中，在 COM 组件列表中选中所需要的组件的复选框。

3.单击 OK。

通过上面的操作，把包装类加到了当前项目中，包装类放置在包中，包的名字源于 COM 组件的文件名。因为 Java 区分大小写，所以按照惯例，包的名字全部小写。

可以把 ActiveX 控件加到窗体中，COM 的包装类也可以同样操作。在下面的图中，Project Explorer 显示 Masked Edit 控件中的包装类。

Masked Edit 控件在文件 MSMASK32.OCX 中，所以，包装类的包是 msmask32。

使用包装类

给一个 COM 组件加一个包装类，我们就可以得到 Java 类和 Java 接口。这些类相当于共存类，接口相当于 COM 接口。

为了包装类，必须用类创建一个项目。通常使用接口访问对象，例如，在下面，我们使用了包装类 COMClass 和 COMInterface：

```
COMInterface pInf;
PInf = new COMClass();
```

```
PInf.foo();
```

对象的引用是接口类型 COMInterface。类 COMClass 用于创建 COM 对象。到对象的任何访问都使用接口引用。

为了使用不同的 COM 接口访问一个 COM 对象，我们转换了对应包装接口的引用。

实验 12-3：COM MessageBeep

在本实验中，我们创建另外一个应用程序来播放 MessageBeep 声音。这次，我们使用 COM 对象来调用 Win32 API 函数。这个 COM 对象是用 Visual Basic 写成的，如果你已经在系统中安装了 Visual Basic，可以在 Chapter12\VBObject 中找到项目，这个项目创建了这个 COM DLL。

COM 对象导出两种方法:GetString 和 PlaySound。它也导出 BeeperEnum 枚举种类 BeeperEnum。

实验概述

在本实验中，将练习以下内容：

- 往项目中加入 COM 封装类。
- 使用封装类给共存类创建一个 COM 对象。
- 使用封装类为接口访问 COM 对象。

实验准备

1. 注册 VbBeeper.dll 文件，操作步骤位于 Chapter12\Lab12-

3\RegisterDll.txt。

2. 启动 Visual J++。
3. 打开 COMBeeper 对象，该对象位于 Chapter12\Lab12-3\COMBeeper.sln。

实验步骤

1. 创建 VBBeeper 的 COM 封装类，如果 VBBeeper 不在 COM 类的列表中，需要使用 Brower 按钮注册组件，与创建 Java 封装类时的一样操作。这便创建了本地的包 vbbeeper,在其中包含了三个类:Control，_Control 和 BeeperEnum。Control 是一个共存类，_Control 是这个共存类的接口，BeeperEnum 是 beeper 数值的一个参数类型。
2. 加入一个成员变量，以保存 COM 对象的引用，它的类型是 vbbeeper._Control，作为 vbbeeper 的接口类，成员变量命名为 ctl。
3. 加入 int 数组成员变量，以关联列表框中带 BeeperEnum 数值的条目，命名数组变量为 values，使得该变量有 5 个元素那么长。
4. 在这个窗体的构造器，创建一个 vbbeeper.Control 对象。给这个构造器赋值成员变量。使用 vbbeeper.BeeperEnum 数值，Control 对象填入列表框。关联这个使用 BeeperEnum 数值的列表框的入口，使用 value 数组。BeeperEnum 中的数值是 DefaultBeep，ErrorBeep，QuestionBeep，ExclamationBeep 和 AsteriskBeep。

   下面有一些操作代码：

   ```
   String str;
   ```

```
int val;
int idx;

val = vbbeeper.BeeperEnum.DefaultBeep;
str = ctl.GetString(val);
idx = lstBeeps.addItem(str);
values[idx] = val;
```

5. 给 Play 按钮的单击事件 Click 加入一个处理程序，在这个处理程序中，调用 VBBeeper 对象的 PlaySound 方法。对于 PlaySound 的参数，使用 values 数组，该数组由列表框中的 selectedIndex 属性的值作为下标。
6. 测试应用程序，可以把它同 Chapter\Solb12-3\COMBeeper.sln 的解决方案进行比较。

可选实验

可以尝试如下的附加实验：

- 加入对于列表框的双击处理程序，这将播放所选中的声音。
- 用一个下拉列表框替代前面所使用的列表框，下拉列表框是具有 Dropdownlist 属性的组合框。

下一步

在这一章中，我们已经学习了如何使用控件，在下一章中，我们将学习如何创建 WFC 组件，以及通过使用 WFC 和 DHTML 来创建 Web 应用程序。

第十三章　可移植 I/O

这一章的内容包括 java.io 和 java.net 这两个可移植的 I/O 软件包。这些软件包向 Java 提供了对文件和网络进行访问的能力。WFC（Windows Foundation Class，Windows 基类）也提供了一个文件 I/O 软件包，即 com.ms.wfc.io。在第 8 章中，已经学过了有关 WFC I/O 软件包的内容。java.io 软件包提供了 WFC I/O 软件包的同类功能，但实现方法不同。学习 java.io 可使学习 java.net 的工作变得相对容易些。

在本章将学到以下内容：

- 用于对文件进行操作的 java.io 类。
- 用于访问网络资源的 java.net 软件包。

你将能够：

- 创建一个应用程序，从本地文件中读取数据，并显示其内容。
- 创建一个应用程序，从网络上的文件中读取数据，并显示其内容。

注意：不要将 WFC I/O 的操作与 java.io 的操作弄混。它们是两个来源完全不同的文件 I/O 软件包。

处理文件

java.io 软件包通过数据流进行 I/O 操作。数据流就是一个由字节组成的有序集合。很容易就能看出，文件是一个数据流。在本章后面，我们还将看到，其他各类数据流也使用 java.io 的类和方法。

可以按不同标准对 java.io 类进行划分。很正常的，这里包括输入类和输出类。也可以按照数据流的类型对 java.io 类进行再划分，根据数据流是二进制的还是文本的，数据流可分为字节数据流和字符数据流（请记住，在 java 语言中，字符数据类型为两字节宽）。这里也包括用作相关类的一般基类的抽象类。你会发现，二进制数据流类和字符数据流类之间存在很多相似之处。

在本章中，我们将讨论作为基础的抽象类、二进制 I/O 类，然后是文本 I/O 类。

抽象输入类

在 java.io 中，二进制输入的基础是抽象类 java.io.InputStream，文本输入的基础是抽象类 java.io.Reader。这两个抽象类定义了面向字节输入和面向字符输入的基本方法。

注意：为简单起见，下面我们将使用“原子”的概念来描述一个字节或一个字符，而不再反复重复“字节或字符”这个说法。

从数据流中读取数据

要从 InputStream 中读取字节数据，可以调用下面几种 read 方法：

```
public int read() throws IOException;
public int read(byte[] b) throws IOException;
public int read(byte[] b, int offset, int len ) throws IOException;
```

要从 Reader 中读取字符，可以调用下面几种 read 方法：

```
public int read() throws IOException;
public int read(char[] b) throws IOException;
public int read(char[] b, int offset, int len ) throws IOException;
```

无参数的方法读取单独一个原子，并返回该原子的整数值。单参数方法和三参数方法从输入流中读取一个数组，并返回读入数组中的原子数目。所有这些方法在遇到文件结束符时都返回-1。

关闭数据流

要关闭 InputStream 数据流或 Reader 数据流，要调用 close 方法：

```
public void close () throws IOException;
```

注意，close 方法对于 InputStream 类和 Reader 类看起来都是一样的。

close 方法关闭相关的数据流，并释放相关的系统资源。无需说明的是，在使用 close 之后，就不能再使用那个数据流了。

放弃数据流中的一些原子

要想从 InputStream 中放弃一些字节，或从 Reader 中放弃一些字符，要调用 skip 方法：

```
public long skip (long n ) throws IOException;
```

注意，skip 方法对于 InputStream 类和 Reader 类看起来都是一样的。

skip 方法在数据流中跳过指定数目的原子。

抽象输出类

在 java.io 中，二进制输出的基础是抽象类 java.io.OutputStream，文本输入的基础是抽象类 java.io.Writer。这两个抽象类定义了面向字节输出和面向字符输出的基本方法。

向数据流中写入数据

要向 OutputStream 中写入字节，可调用以下 write 方法之一：

```
public void write (int b ) throws IOException;
public void write (byte[ ] b) throws IOException;
public void write (byte[ ] b , int offset , int len ) throws IOException;
```

要向 Writer 中写入字节，可调用以下 write 方法之一：

```
public void write (int c ) throws IOException;
public void write (char[ ] b) throws IOException;
public void write (char[ ] b , int offset , int len ) throws IOException;
public void write (string str) throws IOException;
public void write (string str , int offset , int len ) throws IOException;
```

整型参数的方法写入单个原子的值。其他单参数类型的方法写入一系列原子（一个原子数组或一个字符串）。三参数类型的方法只写入原子集中的一部分。

关闭数据流

要关闭 OutputStream 数据流或 Writer 数据流，要调用 close 方法：

```
public void close () throws IOException;
```

注意： close 方法对于 OutputStream 类和 Writer 类看起来都是一样的。

---

close 方法关闭相关的数据流，并释放相关的系统资源。对于 OutputStream 数据流来说，close 方法同时还向该数据流写入内容。无需说明的是，在使用 close 之后，就不能再使用那个数据流了。

清空数据流的相关缓冲区

要强制 OutputStream 数据流或 Writer 数据流的相关缓冲区进行输出，可调用 flush 方法：

```
public void flush ( ) throws IOException;
```

注意： flush 方法对于 OutputStream 类和 Writer 类看起来都是一样的。

flush 方法强制缓冲区中的输出内容被写入数据流中。

二进制输入类

InputStream 的非抽象子类是 java.io.DataInputStream 和 java.io.FileInputStream。java.io.DataInputStream 扩展了 InputStream。它提供了读取一般数据类型的方法。下面列出了一些比较重要的方法：

```
Public DataInputStream (InputStream in);

Public final boolean readBoolean () throws IOException;
Public final byte readByte() throws IOException;
Public final char readChar() throws IOException;
Public final short readShort() throws IOException;
Public final int readInt() throws IOException;
Public final long readlong() throws IOException;
Public final float readFloat() throws IOException;
Public final double readDouble() throws IOException;
Public final String readUTF() throws IOException;
```

这里有三件事值得注意：

- DataInputStream 的构造器使用 InputStream 作为参数。
- readUTF 方法是一个字符串方法。

- DataInputStream 是 final 方法，它们不能被超越。

java.io.FileInputStream 对 InputStream 进行了扩展。它将输入流与文件相关联。下面是两个比较重要的方法：

```
public FileInputStream ( String name ) throws FileNotFoundException;
public FileInputStream ( File file ) throws FileNotFoundException;
```

第一种形式的构造器采用字符串参数，即文件名。第二种方式使用 java.io.File 对象。将在本章后面学习更多有关 File 类的内容。

示例

下面是一个从二进制文件中读取数据的例子：

```
FileInputStream fisInput = new FileInputStream(fileInput);
DataInputStream disInput = new DataInputStream(fisInput);
String strValue = disInput.readUTF();
Int nValue = disInput.readInt();
Float fValue = disInput.readFloat();
Booean bValue = disInput.readBoolean();
```

FileInputStream 类用于打开文件。该对象是 InputStream 类型的，所以它被用来创建一个 DataInputStream 对象以读取特定类型的数据。本例的源代码在第 13 章的 Demo13-1 项目中。

注意：这段代码不包含任何异常处理程序，以便让人容易看清怎样使用 I/O 方法。要想看包含异常处理程序的例子，请参阅第 13 章的 Demo13-1 项目。

二进制输出类

OutputStream 的非抽象子类是 java.io.DataOutputStream 和 java.io.FileOutputStream。java.io.DataOutputStream 扩展了 OutputStream。它提供了写入一般数据类型的方法。下面列出了一些比较重要的方法：

```
Public DataOutputStream(OutputStream Out);
Public final void writeBoolean(boolean v) throws IOException;
Public final void writeByte(int v) throws IOException;
Public final void writeChar(int v)throws IOException;
Public final void writeShort(int v) throws IOException;
Public final void writeInt(int v) throws IOException;
Public final void writeLong(long v) throws IOException;
Public final void writeFloat(float v) throws IOException;
Public final void wrtieDouble(double v) throws IOException;
Public final void writeUTF(String s) throws IOException;
```

该类提供了在字节数据流和一般数据类型间进行转换的功能。请注意 DataOutputStream 的构造器要使用 OutputStream 作为参数。

java.io.FileOutputStream 也对 OutputStream 进行了扩展。它将输出流与文件相关联。下面列出了三个比较重要的方法：

```
public FileOutputStream ( String name ) throws IOException;
```

```
public FileOutputStream ( String name , boolean append) throws IOException;
public FileOutputStream ( File file ) throws IOException;
```

第一种方式的构造器使用字符串参数——文件名。第二种方式使用文件名及一个标记以表明是否添加到文件的结尾。第三种方式使用 java.io.File 类。将在后面学到更多有关 File 类的内容。

示例

下面是一个向二进制文件写入数据的例子：

```
FileOutputStream fosOutput = new FileOutputStream(fleOutput);
DataOutputStream dosOutput = new DataOutputStream(fosOutput);
DosOutput.writeUTF(strValue);
DosOutpur.writeInt(nValue);
DosOutput.writeFloat((fValue);
DosOutput.writeBoolean(bValue);
```

本例的详细代码也在第 13 章的 Demo13-1 项目中。

文本输入类

Reader 的非抽象子类是 java.io.InputStreamReader 和 java.io.FileReader。

我们使用 java.io.InputStreamReader 类将 Reader 与 InputStream 关联起来。InputStreamReader 是对 Reader 的扩展，所以它可以利用 Reader 的所有方法。其最重要的构造方法如下：

```
public InputStreamReader ( InputStream in );
```

java.io.FileReader 对 InputStreamReader 进行了扩展。用该类可将 FileReader 与给定文件进行关联。下面列出了两种比较重要的构造方法：

```
public FileReader ( String name ) throws FileNotFoundException;
public FileReader (File file ) throws FileNotFoundException;
```

示例

下面的例子从文件中读取了 25 个字符：

```
FileReader fileInput = new FileReader( filename );
Char[] buffer = nerw char[25];
Int nNumberOfCharactersRead;
NNumberOfCharactersRead = fileInput.read(buffer);
If(nNumberOfDCharactersRead < 25)
{
```

文本输出类

Writer 的非抽象子类是 java.io.OutputStreamWriter 和 FileWriter。

我们使用 java.io.OutputStreamWriter 将 Writer 与 OutputStream 相关联。OutputStreamWriter 是对 Writer 的扩展，所以它可以利用 Writer 的所有方法。其最重要的构造方法如下：

```
public OutputStreamWriter ( OutputStream out );
```

java.io.FileWriter 类对 OutputStreamWriter 进行了扩展。在生成该类时，构

造器将 FileWriter 与给定文件进行关联。下面列出了两种比较重要的构造方法：

```
public FileWriter ( String name ) throws IOException;
public FileWriter (File file ) throws IOException;
```

示例

下面是一个将字符串写入文本文件的例子：

```
FileWriter fileOutput = new FileWriter (filename);
fileOutput.write (edtTextForOutput.getText ( ) );
```

也可以使用 java.io.PrintStream 类来扩展 OutputStream。该类创建一个二进制的输出，在该类中包含大量的重载，以将一般数据类型转换为它们的字符串表述。下面是 PrintStream 的一些重要方法：

```
public PrintStream(OutputStream out);
public PrintStream(OutputStream out, boolean autoFlush);

public void print(boolean b);
public void print(char c);
public void print(int i);
public void print(long 1);
public void print(float f);
public void print(double d);
public void print(char[] s);
public void print(String s);
public void print(Object o);
```

```
public void println();
public void println(boolean b);
public void println(char c);
public void println(int i);
public void println(long l);
public void println(float f);
public void println(double d);
public void println(char[] s);
public void println(String s);
public void println(Object o);
```

这些方法是自解释的。你可能会疑惑，为什么没有针对字节类型和短类型的方法呢。这是因为这些类型被自动转换成整型了。这种转换是安全的，因为整型的精度比这两种类型的精度都要高。在下一部分给出了一个使用 PrintStream 对象的例子。

java.lang.System 类和标准 I/O

在 java.lang.System 类中定义了三种数据流。它们都是公有、静态的，而且是最终类型。

```
    public static final InputStream in;
public static final PrintputStream out;
    public static final PrintStream err;
```

这几种数据流为应用程序实现标准输入、标准输出和标准错误数据流。这类似于 WFC 的 Text 类。如想了解更多有关 WFC Text 类的知识，请参阅第 11 章。

下面是 java.io 版的 Hello World 程序：

```
public class SayHello
{
    public static void main ( String [ ] args )
    {
        System.out.printIn("Hello, World!");
    }
}
```

下面是 WFC 版本：

```
import com.ms.wfc.io.Text;
public class SayHello
{
    public static void main ( String [ ] args )
    {
        Text.out.writeLine ("Hello, World!");
    }
}
```

请注意，以上两种版本的结果都是输出相同的一行文字：“Hello, World !”。

java.io.File 类

java.io.File 类提供了访问文件和文件系统的途径。下面是 File 类的一些比较有趣、比较重要的方法：

```
public File(String path);
public File(String path, String name);
public File(File dir, String name);

    public boolean canRead();
public boolean canWrite();
public boolean isFile();
public boolean isDirectory();
public boolean exists();
public long length();
public long lastModified();

    public String getName();
public String getPath();
public String getParent();

    public boolena mkdir();
public boolean mkdirs();

    public boolena delete();
    public boolean renameTo(File name);
```

这些方法的绝大部分都是自解释的。但是，请注意这其中没有任何一种方法用来创建或打开文件。要想创建文件，我们使用 FileOutputStream 或 FileWriter。要想打开文件，我们使用 FileInputStream, FileOutputStream, FileReader 或者是 FileWriter。

getParent 方法返回文件所在目录。mkdir 方法将 filename 参数作为目录名，并创建该目录。mkdirs 方法创建路径所需的中间多个目录。

示例

在下面的例子中，在向某一文件写入内容之前，使用了 File 类来检查该文件是否存在。

```
File fileOutput = new File(cbNames.getText());
If(fileOutput.exists())
{
if(MessageBox.show(
    ″ File exists. Overwrite it? ″ ,
    ″ File Error″ ,
    MessageBox.YESNO) == MessageBox.IDNO)
{
    return
}
}
```

本例源代码在第 13 章的 Demo13-1 项目中。

实验 13-1：从磁盘中读取文件

在本实验中，我们将在一个简单的应用程序中添加一些代码，使其打开一个文本文件，读取数据，并将文件内容显示在一个编辑控件中。

实验概述

在本实验中，将练习以下内容：

- 使用 java.io 软件包打开文件。
- 使用 java.io 软件包从文件中读取数据。

实验准备

1. 启动 Visual J++。
2. 打开名为 FileReader 的项目（Chapter13\Lab13-1\FileReader.sln）。

实验指示

1. 为 java.io 软件包添加一条 import 语句。
2. 找到 btnRead 控件的 click 处理程序。
3. 在该处理程序中，声明对 FileReader 对象的引用。为 edtInput 控件中的 filename 创建一个 FileReader 对象。

   FileReader 的构造器可能引发 FileNotFoundException。因此，必须为这种异常添加一个处理程序。在 catch 块中，要完成以下步骤：

   - 将 edtOutput 的 foreColor 设置为红色。

- 将 edtOutput 的 backColor 设置为白色。
- 给 edtOutput 设置一条文本以显示错误信息。FileNotFoundException 的 getMessage 方法返回没有找到的文件的文件名。

在 try 块外，将 edtOutput 设置为正常输出：

- 将 edtOutput 的 foreColor 设置为 windows 字体颜色。
- 将 edtOutput 的 backColor 设置为 windows 背景颜色。
- 清空 edtOutput 中的文字。

4. 声明一个字符数组以存放读取到的字符。数组名为 buffer。长度设为 256 单元长。
5. 声明一个整型变量以记录从文件中读取到的字符数目。
6. 建立一个循环从文件中读取字符。用从文件中读取到的那些字符更新 edtOutput 的 text 属性。

建立一个循环从文件中读取字符，直到遇到了文件结束符为止，此时 read 将返回-1，否则，它返回读入缓冲区中的字符的数目。

将这些字符添加到 edtOutput 的结尾。下面的字符串构造器可能对你有所帮助：

```
public String ( char [ ] buffer, int offset, int length ) ;
```

read 方法可能引发 IOException。必须为这种异常添加一段处理程序。如果发现了异常，退出循环，并在 edtOutput 的结尾显示一条错误信息。将 edtOutput 的颜色设置为白底红字表示错误。

7. 从文件读取完毕后，关闭文件。close 方法可能引发 IOException。必

须为这种异常添加一段处理程序。catch 块可能是空的。

8. 在从处理程序退出之前将焦点设置在 edtInput。
9. 测试程序。可以将工作与 Chapter13\Sol13-1\FileReader.sln 中的解决方案进行一下比较。

下面给出了一些可以在本实验的基础上进行进一步工作的建议：

- 生成一个 SYSEDIT 的替代版本。把 edtInput 换为一个下拉组合框。将下拉组合框的内容设置为下列系统文件：

  C:\AUTOEXEC.BAT
  C:\CONFIG.SYS
  C:\WINDOWS\WIN.INI
  C:\WINDOWS\SYSTEM.INI
  C:\WINDOWS\PROTOCOL.INI

- 用 System.in 和 System.out 分别作为输入数据流和输出数据流重写实验内容。

下一步

我们在这里使用的功能与在第 8 章所学到的功能是等价的。在下一部分，将会使用 java.net 软件包中的许多相同的类来读取网络文件。

访问 Internet

Java 语言对于 Internet 的支持包含在 java.net 软件包中。java.net 软件包中包含两个非常有趣的类：

- java.net.Socket
- java.net.URL

java.net.Socket 类

Socket 类支持网络通信。它是一个一般性的类，可以用来进行任何类型数据的传输。它包含 getInputStream 方法和 getOutputStream 方法。一旦套接字连接建立完成，就可以使用这两种功能来实现 java.io.InputStream 和 java.io.OutputStream 的机能。

java.net.URL 类

java.net.URL 类支持对 URL 的操作。利用该类可以下载与 URL 关联的数据。实际上，我们在用 applet 处理媒体文件时已经使用了 URL。Applet.getCodeBase 和 Applet.getDocumentBase 方法返回 URL 引用。Applet.getAudioClip，Applet.getImage 和 Applet.play 方法将 URL 引用作为参数。

经常使用的 java.net.URL 方法包括：

```
public URL(String protocol, String host, String file)
```

```
throws MalformedURLException;
  public URL(String spec) throws MalformedURLException;
  public URL(URL context, String spec) throws MalformedURLException;

      public String getProtocol();
  public String getHost();
  public String getFile();

      public final InputStream openStream() throws IOException;
```

可以利用这些方法来获取 Internet 上的资源。如果在连接 URL 时需要更多的灵活性，可使用下面的功能：

```
      public URLConnection openConnection ( ) throws IOException;
```

java.net.URLConnection 类支持相当多的 I/O 操作。要了解有关该类的更多信息，请参阅产品文档。

实验 13-2：从 Web 读取数据

在本实验中，我们将向一个简单程序添加代码，使其将 URL 返回的数据流显示出来。本实验与本章前面的实验 13-1 有些相似。可以将前面文件操作实验中完成的程序拷贝过来作为本实验的基础，或者可以使用启动项目文件。

实验概述

在本实验中，将会练习以下内容：

- 使用 URL 打开网络连接。
- 从网络连接上读取数据。

实验准备

1. 启动 Visual J++。
2. 打开 URLReader 项目（Chapter\Lab13-2\URLReader.sln）。

实验步骤

1. 为 java.io 和 java.net 软件包添加 import 语句。
2. 找到 btnRead 控件的 click 处理程序。
3. 在处理程序中，声明对 URL 对象的引用。为 edtInput 的 text 文本值建立一个 URL。

   URL 构造器可能引发 MalformedURLException。必须为这种异常添加一段处理程序。在 catch 块中，完成以下工作：
   - 将 edtOutput 的 foreColor 设置为红色。
   - 将 edtOutput 的 backColor 设置为白色。
   - 给 edtOutput 设置一条文本以显示错误信息。MalformedURLException 的 getMessage 方法返回 URL。

   在 try 块外，将 edtOutput 设置为正常输出：
   - 将 edtOutput 的 foreColor 设置为 windows 字体颜色。
   - 将 edtOutput 的 backColor 设置为 windows 背景颜色。
   - 清空 edtOutput 中的文字。

4. 声明一个字符数组以存放读取到的字符。数组名为 buffer。长度设为 256 单元长。
5. 声明一个整型变量以记录从 URL 中读取到的字符数目。
6. 建立一个循环从 URL 中读取字符。用从 URL 中读取到的那些字符更新 edtOutput 的 text 属性。

   建立一个循环从 URL 中读取字符，记住，当遇到了文件结束符时，read 将返回-1，否则它返回读入缓冲区中的字符的数目。

   将读取到的这些字符添加到 edtOutput 的结尾。

   read 方法可能引发 IOException。必须为这种异常添加一段处理程序。如果发现了异常，退出循环，并在 edtOutput 的结尾显示一条错误信息。将 edtOutput 的颜色设置为白底红字表示错误。
7. 从 URL 读取完毕后，要将其关闭。close 方法可能引发 IOException。必须为这种异常添加一段处理程序。catch 块可能是空的。
8. 在从处理程序退出之前将焦点设置在 edtInput。
9. 测试程序。可以将工作与 Chapter13\Sol13-2\URLReader.sln 中的解决方案进行一下比较。

选作实验内容

可以尝试以下选作实验内容：

- 将 edtInput 编辑控件换为一个组合框控件。将该组合框控件加载为指向最喜欢的有些 Web 站点的 URL。
- 向程序中加入一个 HTML 控件，以将输入数据流作为 HTML 显示出

来。

附录 A　快速格式对比

本附录提供了 Java, C, C++和 Visual Basic 代码模块以进行对比。每个模块都定义了一个由两个整型变量 x 和 y 组成的“坐标”类型。同时，每个模块还定义了将两个“坐标”相加的方法和给出某一“坐标”相角位置的方法。

可以看到 C++版本与 Java 版本是多么相似。从结构上说，Visual Basic 版本也很相似。这三个版本都是按照面向对象的模块编写的。C 模块的单独一行看起来和 C++或 Java 模块中的一行很相像，这是因为这三者共享相同的格式。

注意：设计完善的模块还应该包括其他功能，以操纵“坐标”对象。

“坐标”模块的 C 版本

```
// file: coordinate.h
    struct Coordinate
    {
            int x;
            int y;
    };
```

```
    struct Coordinate add(
      const struct Coordinate pt1,
      const struct Coordinate pt2);

    int Quadrant(const struct Coordinate pt);

    // file:coordinate.c
#include ″ coordinate.h″
struct Coordinate add(
        const struct Coordinate pt1,
        const struct Coordinate pt2);
{
      struct Coordinate returnValue;
      returnValue.x = pt1.x + pt2.x;
      returnValue.y = pt1.y + pt2.y;
      return returnValue;
}

    int Quadrant(const struct Coordinate pt)
{
      if(pt.x >=0) {
            if(pt.y >=0) {
                  return 1;
            } else {
```

```
                return 4;
            }
        } else {
            if(pt.y >=0) {
                return 2;
            } else {
                return 3;
            }
        }
}
```

“坐标”模块的C++版本

```
// file: coordinate.h
class Coordinate
{
private:
    int xValue;
    int yValue;
public:
    int getX() const;
```

```
        int getY() const;
        void setX(int value);
        void setY(int value);
        Coordinate add(const Coordinate& pt2) const;
        Int getQuadrant() const;
}

    // file: coordinate.cpp
#include ″ coordinate.h″

    int Coordinate::getX() const
{
        return xValue;
}

    int Coordinate::getY() const
{
        return yValue;
}

    void Coordinate::setX(int value)
{
        xValue = value;
}
```

```
    void Coordinate::setY(int value)
{
    yValue = value;
}

    Coordinate Coordinate::add(const Coordinate& pt2) const
{
    Coordinate returnValue;
    ReturnValue.xValue = this.xValue + pt2.xValue;
    ReturnValue.yValue = this.yValue + pt2.yValue;
    Return returnValue;
}int Coordinate::Quadrant() const
{
    if(this.xValue >=0) {
            if(this.yValue >=0) {
                    return 1;
            } else {
                    return 4;
            }
    } else   {
            if(this.yValue >= 0) {
                    return 2;
            } else {
```

```
                return 3;
            }
        }
}
..
```

“坐标”模块的 Java 版本

```
// file: Coordinate.java
public class Coordinate
{
        private int xValue;
        private int yValue;

            public int getX()
        {
                return xValue;
        }

            public int getY()
        {
                 return yValue;
        }
```

```
        public void setX(int value)
    {
          xValue = value;
    }
        public void setY(int value)
    {
          yValue = value;
    }
        public Coordinate add(Coordinae pt2)
    {
    Coordinatge returnValue = new Coordinae();
    ReturnValue.xValue = this.xValue + pt2.yValue;
    ReturnValue.yValue = this .yValue + pt2.yValue;
}       Return returnValue;
    public int getQuadrant()
{
    if(this.xValue >=0) {
          if(this.yValue >=0) {
                return 1;
          } else {
                return 4;
```

```
            }
        } else {
            if(this.yValue >=0) {
                return 2;
            } else {
                return 3;
            }
        }
}
 }
```

“坐标”模块的 Visual Basic 版本

```
 ‘ file: Coordinate.cls
 Dim xValue As Integer
 Dim yValue As Integer

 Public Property Get X() As Integer
        X = xValue
 End Property

      Public property Get Y() As Integer
```

```
        Y = yValue
End Property

    Public Property Let X(value As Integer)
        XValue = value
End property

    Public Property Let Y(value As Integer)
        YValue = value
End Property

    Public Function Add(ByVal pt2 As Coordinate) As Coordinate
        Set Add = New Coordinate
        Add.X = Me.X + pt2.X
        Add.Y = Me.Y + pt2.Y
End Function

    Public Function GetQuadrant() as Integer
        If Me.X >= 0 Then
                If Me.Y >= 0 Then
                        GetQuadrant = 1
                Else
                        GetQuadrant = 4
                End If
        Else
```

```
            If Me.Y >= 0 Then
                GetQuadrant = 2
            Else
                GetQuadrant = 3
            End If
        End If
End Function
```

附 录 B　Java 格 式 快 速 参 考

Java 语 言 关 键 字

关 键 字 /保 留 字

abstract	extends	multicast
boolean	false	native
break	final	new
byte	finally	null
byvalue	float	operator
case	for	outer
cast	future	package
catch	generic	private
char	goto	protected
class	if	public
const	implements	rest

continue	import	return
default	inner	short
delegate	instanceof	static
do	int	super
double	interface	switch synchronized
else	long	this

throw
throws
transient
true
try
var
void
volatile
while

对于上面列出的关键字/保留字有以下几点说明：

- 这里列出的是不能（或不应该）用作标识符的名字，因此比 Java1.1 语言所使用的保留字更多一些。
- 保留字 delegate 和 multicast 是 Visual J++ 6.0 新增的。它们定义在 Windows 基类（WFC）中，用来帮助处理事件。
- Visual J++ 6.0 把 const 和 goto 认作保留字（即它们以其他颜色显示），

但在 Java1.1 中，它们并无特殊含义。不过由于 Visual J++ 把它们认作保留字，就不能把它们用作标识符。

- 保留字 byvalue, cast, future, generic, inner, operator, outer, rest 和 var 在 Java 语言中也是保留字，但在 Java1.1 中也并无特殊含义，不过 Visual J++并不将它们标记为格式颜色。Visual J++ 允许将这些字用作标识符。但是，为了尽可能增加程序的可移植性，让程序清楚明了，最好不要这样做。
- 从技术的角度而言，布尔值 true 和 false 并不是保留字，但不能将它们用作标识符，因此，它们看起来、用起来和给人的感觉都像是保留字。

内部类型

类型	值	缺省值
Boolean	true 和 false	false
Byte	从 -128 到 128 之间的整数值	0
Char	从 \u0000 到 \uFFFF 之间的双字节编码字符	\u0000
Double	64 位浮点数	0.0
Float	32 位浮点数	0.0
Int	32 位整数	0
Long	64 位整数	0
Short	16 位整数	0

Java 格式概述

下面对于 Java 语言格式的简要概述提供了对该语言常用语法结构的快速参考，而并不是对于 Java 语言完整正式的语法说明。

流程控制

do while 循环（相关关键字：break 和 continue）

```
do
    {
    loop body
    } while (boolean expression )
```

for 循环（相关关键字：break 和 continue）

```
for ( initialization; boolean expression; increment )
    {
    loop body
    }
```

if（相关关键字：else）

```
    if (boolean expression)
{
```

```
    statement(s)
    }
else
    {
    statement(s)
    }
```

switch（相关关键字： break, case 和 default）

```
switch (integral variable)
{case integral value:
        statements(s)
        break;
other case (s) as necessary
default;
        statements(s)
}
```

try 块（相关关键字： catch, finally, throw 和 throws）

```
try
    {
    statement(s)
}
```

```
catch (exceptionType identifier )
      {
      statement ( s )
}
other catch blocks as necessary
finally
      {
      statement ( s )
}
```

while 循环（相关关键字： break 和 continue）

```
            while ( boolean expression )
      {
      statement ( s )
      }
```

类

方括弧表示可选格式

```
[public][abstract] class ClassName
{
methods and member varibles
}
```

继 承

```
class SubclassName [extends SuperclassName] [implements InterfaceName]
{
methods and member varibles
}
```

接 口

```
[public][abstract] interface InterfaceName [extends SuperInterfaceName]
{
methods declarations and final member varibles
}
```

软 件 包

```
package packageName;
import packageName.className;
import packageName.*;
```

方 法

Access 方 法

```
[accessModifier]   returnType methodName ( parameter_list )
{
```

```
body of method
}
```

其中，accessModifier 可以是 public, private, protected 或缺省（package）。

线程

```
synchronized returnType methodName ( parameter_list ) {... }
```

继承

```
[final | abstract ] returnType methodName ( parameter_list ) {... }
returnType methodName ( parameter_list ) {... }
{
super (parameter_list );
statements ( s );
return value;
}
```

类方法

```
static returnType methodName ( parameter_list )
{
     statement ( s )
}
```

native 方法

```
class className
{
      native returnType methodName ( parameter_list );
      static
      {
            native method initialization ( s )
        }
        remainder of class definitions
}
```

异常

```
returnType methodName ( parameter_list ) throws exceptionName
{
      statement ( s )
      throw new exceptionName;
      statement ( s )
}
```

成员变量

访问

[accessModifier] type variableName [= initialValue]
其中，accessModifier 可以是 public, private, protected 或缺省（package）。

行为

static
final
volatile
transient

Typecasting, RTTI (Run-Time Type Information, 运行期间类型信息)

```
if (variable instanceof type )
{
    ((type ) variable ).method ( parameterList);
    statement( s )
}
```

附录 C　程序设计的趋势

我在大学学习一门计算机编程的课程时，有一位助教坚决要求我们在写出第一行代码以前，必须仔细地计划清楚我们的程序要做什么，如何做，以及 I/O 接口如何等。如果我们没有事先给他看我们的这些计划工作，他就不帮我们调试程序。和我们同班的一个家伙从来不作什么计划。他总是坐到计算机终端前就开始敲程序。他很聪明，同时他也很幸运，那时的作业都很容易，让人可以直接写下来。这就体现出这样一个问题，到底计算机程序设计的终点是什么呢，是计划、计划、再计划，还是跳过它。计划的问题是，如果程序员能够预测到未来，那我们也就不是程序员了，不是吗？但是显而易见的是，你也不可能不做任何计划，坐下来就写出需要几千行代码的程序，更不必说那些上万、上百万行代码的程序了。所以我们所需要的是一种能在程序员编程时辅助程序员工作的编写软件的方法。如果是一位细致的计划者，这种方法能够帮助作出计划，而不致再陷入那些无关紧要的细节当中。如果属于那种喜欢坐下来直接开始书写代码的类型，那么这种方法将帮助能够尽可能快地坐到键盘前，同时帮你尽可能容易地记录下正在做什么，为什么要这样做而不那样做，甚至可能包括下一步要如何去做。要找到这种方法可不是一件容易的事。现在人们已经在这个问题上研究了一段时间，但，并未找到容易的答案。由于历史原因，许多软件

系统，特别是大型软件系统，是使用“水流（Waterfall）”模型开发出来的。图 C-1 描述了水流模型的示意图。

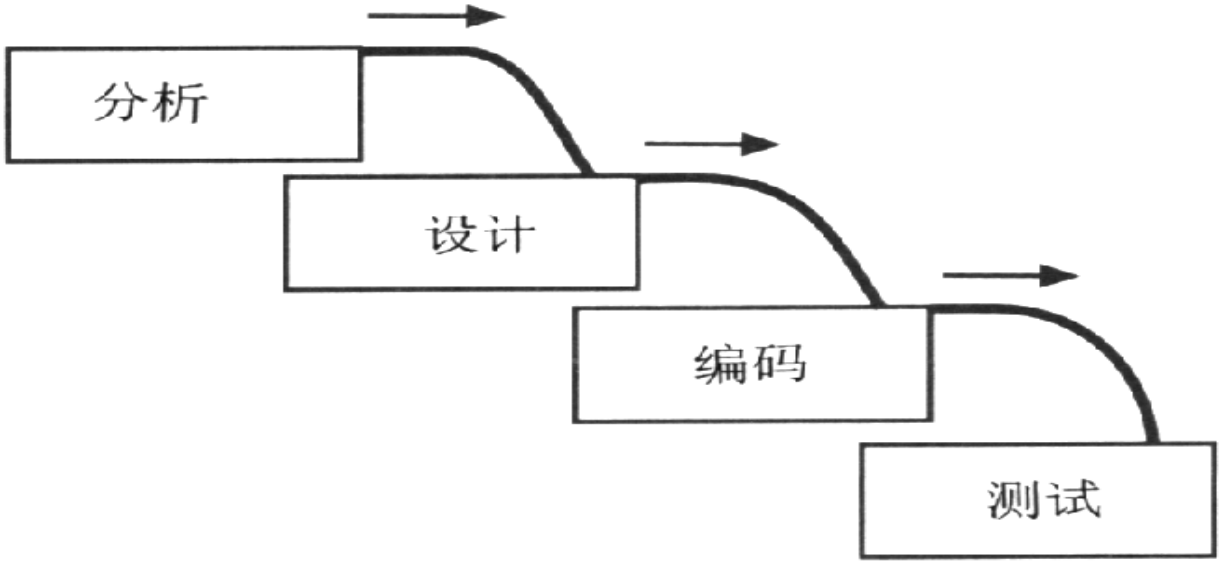


图 C-1　软件开发的水流模型

之所以把它称为水流模型，是因为你依次进行，从一步到下一步从不需要回退；只需按照步骤流程进行下来，然后就完成了。水流模型的缺点在于大多数人的工作方法，并不是这样一条路径。并且，如果打算用这种方法进行工作，恐怕在某一步工作中出现错误可能性是难以避免的，若果真出了错，那么一切就都要重新开始。事实上，最可能发生的“错误”是请编写程序的人自己实际上并不确切地知道他想要什么，或者有时他们会改变想法。

不过，还是不能在对于想要或将要完成什么毫无想法时就坐下来开始写代码。所以，这里引入了一种有更多循环步骤的软件开发方法。图 C-2 展示了这种循环方法的示意图。

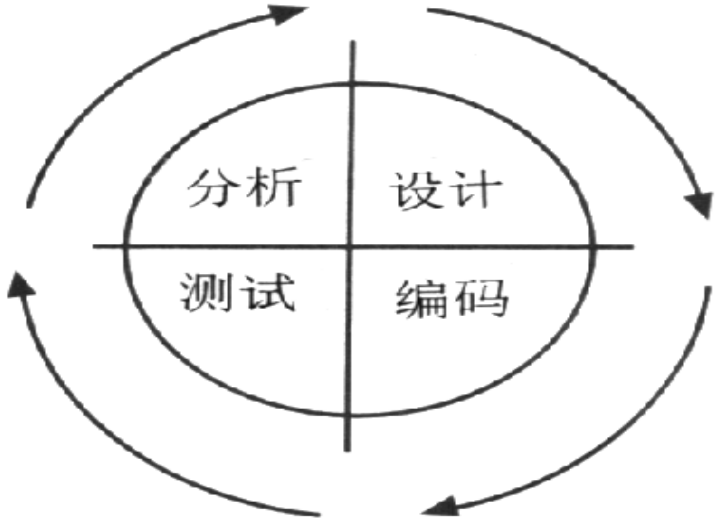


图 C-2　软件开发的循环模型

请注意，这个模型同上一个模型的四个步骤都是一样的。改变之处在于我们希望对各个步骤的访问次数。这种模型有时被称作“设计一点，编写一点，测试一点”的方法。模型的四个步骤是分析、设计、编写代码和测试。下面我们将通过一个简单的示例程序定义在各步中要完成什么工作。在例子中，让我们假设有人请编写一段 Windows 程序，用来计算家庭收入。这个例子相当简单，也许不必动脑筋，就可以直接坐下来写完这个程序。但请不要把注意力放在例子本身上，而应放在我们解决问题的步骤上，然后可以用同样的思路思考一个更复杂的问题（例如设计出 NASA 的火星探路者）如何解决。

分析

在项目的分析阶段，任务是要精确地描述出软件将要完成什么工作。在这个阶段充满了问题。如果有客户（即那个请写这段程序的人），就请他一起来讨论这些问题。如果客户对问题也不确定，就提出一些建议。看到建议被否决，总比看到最终完成的程序被否决要好。下面列出了一些在咱们的家庭收入项目中值得让客户考虑的问题：

- 什么是家庭收入？
- 使用该软件的是些什么人（是专家还是 Windows 新手）？
- 程序怎样获取输入数据？
- 用户需要什么格式的输出（屏幕输出还是写入文件）？
- 用户是否需要在线帮助？
- 需要连接万维网（World Wide Web）吗？
- 程序需何时完成？
- 是否有程序必须遵守的标准？

分析工作应该弄清编程的目的，以及程序该如何实现该目的。

设计

设计阶段的内容包括确定软件怎样实现它应该完成的工作。我们可以作如下假设，在分析阶段已经确定对 Windows 熟练程度各不相同的各级用户都要使

用该软件，输入他们的小时工资和工作小时数，屏幕上将显示出他们的应得收入。

在设计阶段，你认为为了达到这些要求，需要一个具有两个标签（label)，两个编辑框（edit box）和两个按钮（button）的表单。你画了一张表单的草图，大致如图 C-3 所示。

图 C-3 “家庭收入”表单草图

在分析阶段，已经确定家庭收入等于小时工资同工作小时数的乘积，再减去应交收入所得税，再减去其他扣款。在设计阶段，确定出为了完成这项计算功能，要做以下工作：

1. 把 pay-rate 编辑框中的文本转化成数字。
2. 把 hours-worked 编辑框中的文本转化成数字。
3. 把这两个数字相乘。
4. 减去税款。
5. 减去其他扣款。
6. 将 result 标签的文本内容设置等于计算的结果。

完整的设计过程应该能告诉程序要做的每一件事应如何完成。

编码

当进入到编写代码的阶段时，应该已经知道程序应该做什么（分析阶段），及应该怎么做（设计阶段）了。编写代码阶段是将设计结果编写成源代码的阶段。

对我们的例子而言，要进入 Visual J++环境；新建一个项目；添加一个表单；并在表单上创建标签、编辑框和按钮。我们在表单上的 WFC 控件的事件处理程序中写入相应的源代码，从而将设计的要求用源代码的形式表达了出来。

测试

你知道，程序第一次就能成功，我也知道，程序第一次就能正常工作，但那些疑心重重的局外人并不知道这一点，所以我们要为他们进行测试。下面是几个测试要点：

- 输入处在允许范围边缘的数值（即最小允许数值和最大允许数值）。
- 输入允许范围之外的数值（如果有人给小时工资数输入了一个负值，会发生什么情况呢？）。
- 输入无意义的数值（“A”美元是多少钱呢？）。

在所有这些测试中，程序是否都表现良好呢？

当你觉得一切工作都很满意时，就可以把程序拿给客户。你可能会发现，客户在看到实在的东西后，可能又会有一些新的想法和需要。当然，不必非得等到弄出一个可以运行的程序之后才让客户加入进来。例如，设计阶段的表单草图就应当在编写代码之前拿给客户去看。

客户的反馈，加上克服困难的经历，就构成了下一循环工作设计阶段的初始输入信息。工作将不断循环进行下去，直到每一个人都满意为止。或者至少是达到书面要求为止。

附录 D　事件驱动的程序设计

紧接着“面向对象的程序设计”，“事件驱动的程序设计”就是 1990 年软件开发领域最热门的词语了。事件驱动的程序设计与一般的程序设计有什么不同呢？事件驱动的程序设计的全部内容都与给予用户对程序流程的控制有关。

在计算机发展的历史中，人们一直想把对程序流程的控制权交给程序的用户。曾几何时，计算机是通过连接或断开电缆进行编程的。这可以被称作“硬件驱动的程序设计”。如果程序员希望修改程序，他就不得不对计算机本身进行修改。程序员必须懂得计算机操作的各种最细微的细节——对用户很不友好（由于这些麻烦，程序员找到了一种改进的通过计算弹道曲线表来确定如何对敌人投弹的方法）。

后来，开关的出现使得计算机编程对硬件的依赖性有所降低。在往后，人们采用了各种媒体（打孔带、纸带、磁带及类似物品）向计算机传递信息。计算机编程不再需要对硬件进行改动了，而代之以通过软件编程。我们大概可以将这称为“媒体驱动的程序设计”。媒体中包含有指令信息，它告诉计算机对哪些信息进行处理及按什么顺序进行处理。

再往后，“交互计算”出现了，这是人们第一次能够在程序运行时改变程序的流程。但是控制能力是很有限的，在很大程度上，需要依靠编程的程序员的预见力。在 GUIs（Graphical User Interfaces, 图形用户界面，发音是“Gooeys”）

出现以前，程序的流程完全由程序员事先安排。用户被迫从提示的一条命令进行到下一条，按照程序员或代码设计者认为最佳的方式填写数据。

这肯定不适于所有人，但它是从打孔带向前发展的一次巨大飞跃。以前，用户只能把他们的要求告诉程序设计人员，然后由程序设计人员决定如何将数据输入到计算机中去。现在用户可以和计算机直接进行交互对话了，这意味着，软件开发人员不仅要明白用户需要软件做哪些工作，而且要知道用户输入数据的顺序。这个顺序实际上可以是任意的，但软件开发人员只能选择其中一种，并按照这一种顺序编写程序。在另一方面，用户所不得不面对的软件系统往往显得不那么直观，并且难以学习。根据虚拟“平均类型用户”所进行的最佳设置经常对任何特定用户来说都并不适合。但当时交互计算的领域还是新鲜事物，计算机的内存和存储能力也很昂贵和有限，所以我们也只能做到这样了。

随着计算机运行速度的提高和保存信息量的增加，开发人员开始向计算机输出界面设计投入越来越多的努力，并尽可能用方便用户的方式接受信息。在GUIs出现以前，计算机的绝大部分计算能力都直接用来进行数字计算了，人们几乎不花费什么努力来使输入输出界面变得更易于使用。虽然某些人喜欢满屏都是数字，但绝大多数人还是更喜欢图形界面。

因此，我们开始走向事件驱动的程序设计。现在，计算机程序的流程的控制权就像我们一直所希望的那样，交到了用户的手中。这对于我们程序员来说意味着什么呢？我们的程序的 I/O 部分现在是由一组没有次序先后的交互组件组成的，这样，在使用软件时，用户可以自由地（在给定范围之内）按照任何有意义的顺序来进行操作。我们的程序等待用户选择一个菜单项目、单击一个按钮或者移动滚动条。当用户确实执行了一项操作时，程序将执行相应代码，

并等待下一事件发生（顺便提一句，并非所有的事件都由用户产生。在我们的程序之外，还有相当多的情况可能会触发事件，例如：硬件中断，时钟，其他软件等等）。

早期的事件驱动的程序有一个被称作“主事件循环”的部分，它将持续循环下去，寻找事件的发生。程序捕获到事件，并确定它属于哪种类型的事件：鼠标单击或拖动，时钟到点，以及诸如此类的东西。然后就会把事件交付给相应的处理程序进行处理。这种程序结构同现有的其他任何事件驱动的程序都是一样的。Visual J++环境帮我们料理了这些细节中的绝大部分，所以我们在代码中看不到类似“主事件循环”的东西。环境根据建立程序的方式，自动将事件处理程序与特定动作相联系，例如单击一个按钮。这使我们不必费心考虑事件的发生是怎样传递给我们的，而只需专心考虑程序如何对特定事件作出反应就可以了。