第1章 Java GUI设计概述

图形用户界面是当今计算机程序和用户之间的主流接口。本章简要介绍GUI的概念、发展和基本组成，介绍Java GUI程序的实现原理和可视化程序设计的概念及简况。

1.1 GUI简介

1.1.1 GUI概念

GUI是英文Graphical User Interface的简写，中文译作图形用户界面或图形用户接口，是指采用图形方式显示的计算机操作用户界面，是屏幕产品的视觉体验和人机互动操作接口。

与早期计算机使用的命令行界面相比，图形界面使人们不再需要记忆大量的命令，取而代之的是通过窗口、菜单、按键等方式进行操作，极大地方便了非专业用户的使用。GUI使用户在视觉上更易于接受，减少了用户的认知负担，使程序的操作更加人性化。

1.1.2 计算机GUI简史

图形用户界面这一概念是20世纪70年代由施乐公司帕洛阿尔托研究中心提出的，他们在1973年构建了WIMP(即视窗、图标、菜单和点选器/下拉菜单)的范例，并率先在施乐一台实验性的计算机上使用。1983年电子表格软件VisiCalc通过VisiOn的研制，首次引入了PC环境下的“视窗”和鼠标的概念。1984年苹果公司发布了Macintosh计算机，其中配有GUI操作系统而成为首例成功使用GUI的商用产品。1985年末，苹果公司发布Macintosh Office，并首次使用LaserWriter和AppleTalk网络技术。1988年发布的RISC OS是一种彩色GUI操作系统，使用三键鼠标、任务栏和一个文件导航器(类似于Mac OS)。

苹果公司在1984年1月发布的Macintosh计算机的GUI操作系统称为System 1.0，已经含有桌面、窗口、图标、光标、菜单和滚动栏等。1997年7月26日发布的Mac OS 8.0(见图1.1)是具有多线程查找器、三维的窗口界面及新的电脑帮助(辅助说明)等特性的操作系统。1999年10月发布了Mac OS 9，提供了方便的Sherlock 2搜索引擎、多用户管理、安全存储用户名和密码的“钥匙链”、先进的上网、自动化功能和“调色专家”等增强功能。

1984年麻省理工学院与DEC制订计划发展X Window System，同年发布了第一个版

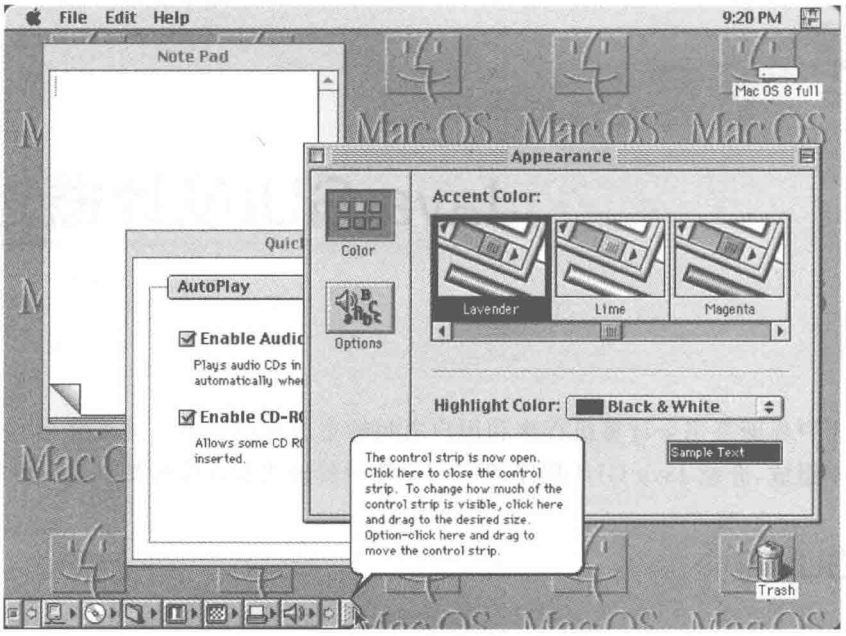
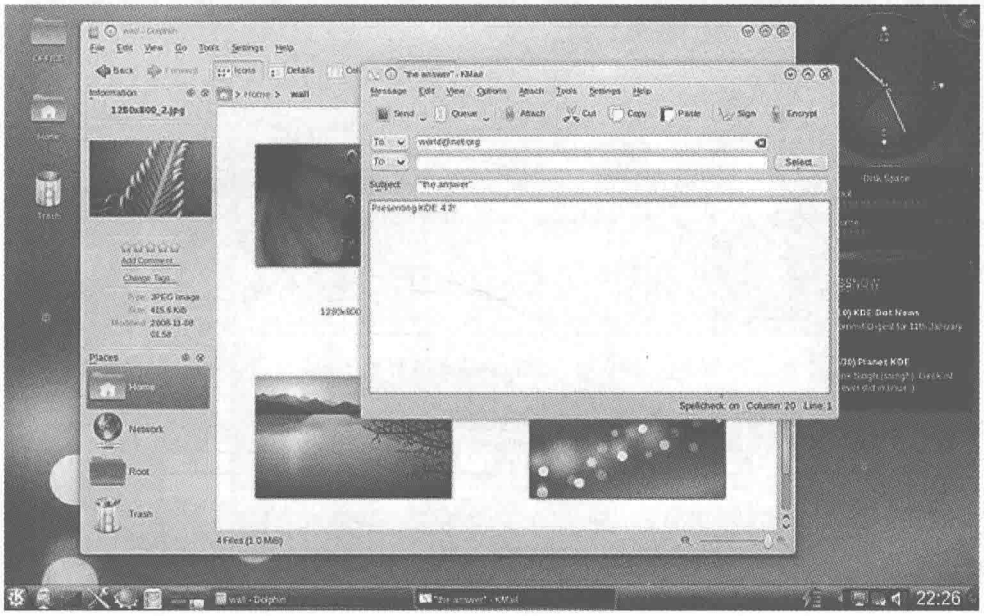


图 1.1 Mac OS 8.0 桌面

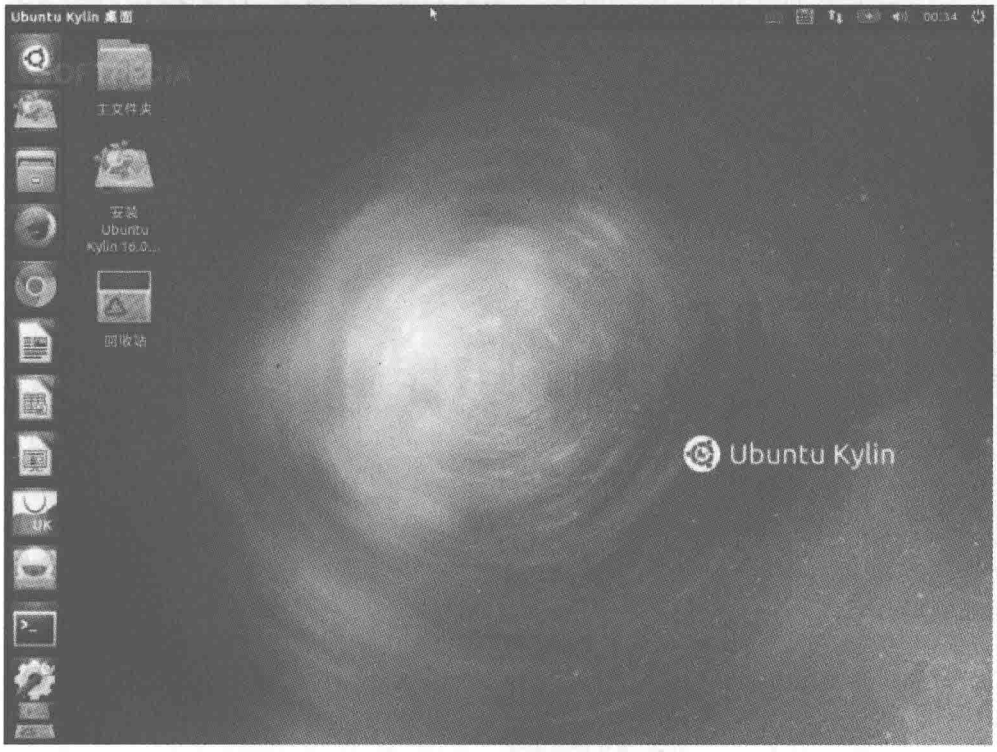
本——X11。1986 年,DEC 公司发布了第一套商业化 X Window System。1987 年 1 月的 X 技术研讨会中,许多工作站销售商共同声明支持 X Window System 作为工作站的标准 GUI,同年 9 月发布了 X Window System 的第 11 版(X11)。1992 年,有 4 位程序员强化改善当时已有的将 X Window System 移植到 x86 结构的 UNIX 系统成果,发起了 XFree86 计划。目前许多免费 UNIX 如 FreeBSD UNIX、NetBSD UNIX 和 Linux 的多种发行版都以 XFree86 作为 GUI 的基础(见图 1.2)。

Microsoft 公司视窗版本 Windows 1.0 在 1985 年发布,是运行于 MS-DOS 操作系统的图形化用户界面(GUI),基于 MAC OS 的 GUI 设计,特点是窗口不可重叠,但可平铺;窗口不会覆盖屏幕下方的图标区域(见图 1.3)。1990 年 Microsoft Windows 3.0 发布,成为当时广泛使用的个人计算机 GUI 操作系统(见图 1.4)。1995 年 Microsoft 公司发布了 Windows 95 操作系统,摆脱了 Windows 3.X 及以前版本对 DOS 的依赖,从此使 GUI 成为个人计算机程序的主要用户接口/界面(见图 1.5)。2012 年 Microsoft 公司发布的 Windows 8 操作系统采用了继承自 Windows 7 的传统桌面(带有 Aero 视觉特效)和同时适用于 PC 及平板电脑的 Metro/Modern 界面(见图 1.6),之后的 Windows 10 界面既采用传统视窗界面,同时也融入了适合移动类系统的 Metro 界面(见图 1.7)。

自 Windows 3.1 开始,PC 上的应用程序逐渐采用图形用户界面,如随 Windows 附带了画图、记事本、写字板、计算器、纸牌和浏览器等几十个 GUI 应用程序,还有相应版本的 Office 软件。Windows 操作系统提供了一整套 GUI 程序设计的 API(应用程序接口),出现了 Visual Basic、Borland Delphi、Borland C++、Visual C++、Visual J++、JBuilder 和 Visual Age 等 GUI 程序设计 IDE 和工具。同样,UNIX 和 Linux 操作系统下的 X Window 也提供了 Xlib、Glib、Gtk+等 GUI 库和 Xt、Motif、Qt 等 GUI 工具包。当前,各种应用程序大多采用了图形用户界面。



(a) 基于XFree86的GUI(桌面环境KDE 4.0(2009))



(b) Ubuntu Kylin 16.04 Alpha1版

图 1.2 以 XFree86 作为基础的发行版

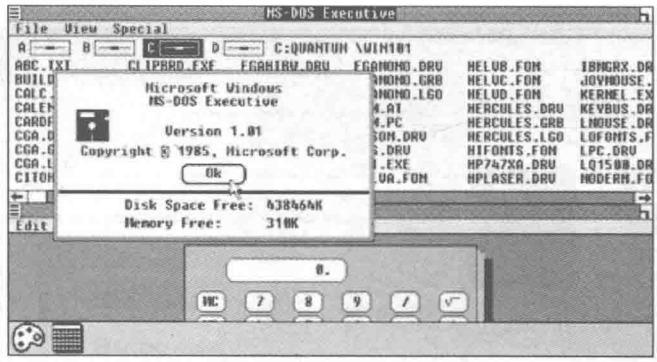


图 1.3 Windows 1.0 界面

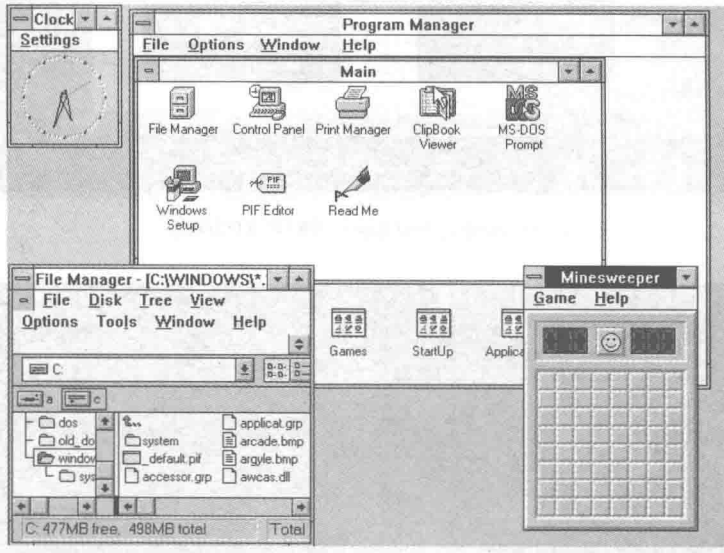


图 1.4 Windows 3.0

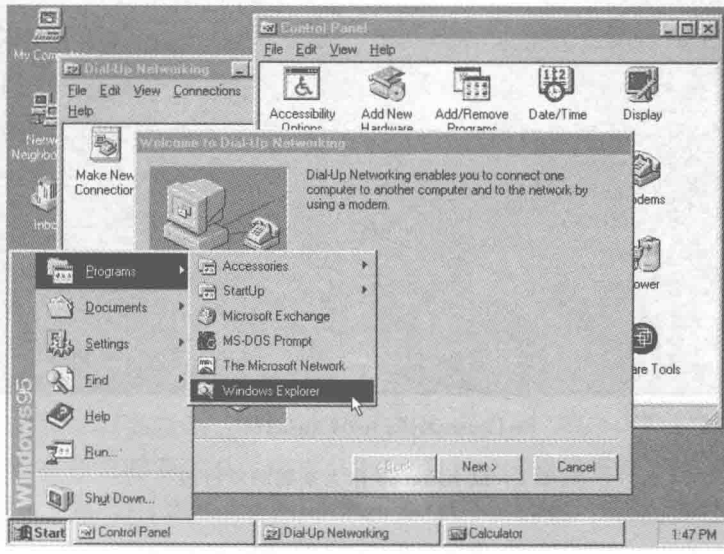


图 1.5 Windows 95

图 1.6 Windows 8 的 Metro/Modern 界面

图 1.7 Windows 10 桌面

1.1.3 GUI 的基本组成

在 GUI 中，计算机屏幕上显示窗口、图标、按钮等图形表示不同资源对象和动作，用户通过鼠标等指针设备进行选择、移动和运行程序等操作。GUI 通常有以下主要组成元素（见图 1.8）。

图 1.8 GUI 的主要组成

1. 桌面

桌面指 GUI 操作系统显示程序、数据和其他资源的计算机屏幕。一般桌面上显示各种应用程序和数据的图标，作为用户对它们操作的入口。如在 Microsoft 公司的 Windows 7 系统中，各种用户的桌面内容实际保存在系统盘（默认为 C 盘）的【\Users\[用户名]\Desktop】文件夹里。

通过将墙纸（即桌面背景）设置为各种图片和某种附件，可以改变桌面的视觉效果。

2. 窗口

窗口是应用程序在图形用户界面中显示的使用界面。应用程序和数据在窗口内实现一体化。用户可以在窗口中操作应用程序，进行数据的管理、生成和编辑。通常在窗口四周设有菜单、图标、滚动条和状态栏等功能部件，数据放在中央。

在窗口中，根据各种数据/应用程序的内容设有标题栏，一般放在窗口的最上方，并设有最大化、最小化、最前面、缩进（仅显示标题栏）等动作按钮，可以简单地对窗口进行操作。

单一文档界面（Single Document Interface）：在窗口中，一套数据在一个窗口内显示和操作的方式。在这种情况下，数据和显示窗口的数量是一样的。若要在其他应用程序的窗口使用数据，将相应生成新的窗口。因此窗口数量多，管理复杂。

多文档界面（Multiple Document Interface）：在一个窗口之内进行多套数据管理的方式。这种情况下，窗口的管理简单化，但是操作变为双重管理。

标签与选项卡：多文档界面的数据管理方式中使用的一种界面（见图 1.9），将数据的标题在窗口中并排，通过选择标签标题显示必要的数据，这样使接入数据更为便捷。多文档界

面主要是 Microsoft 公司视窗系统采用，在其他环境中通常用单文档界面。

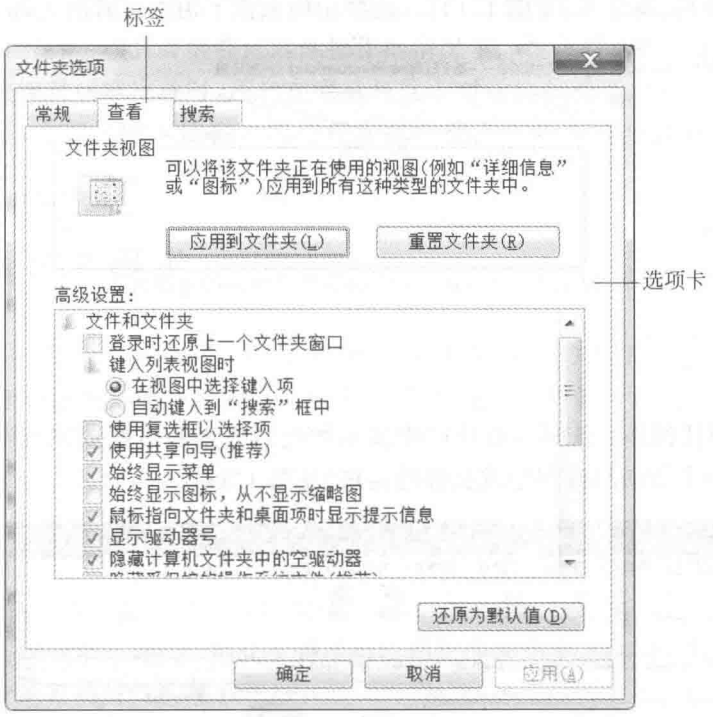


图 1.9 具有标签的对话框

3. 菜单

菜单是把程序提供的执行命令以分级列表的方式显示出来的一种界面，包括下拉式菜单、弹出式菜单等类型，应用程序提供的所有命令几乎全部能组织到菜单中。根据命令的层次还可以组织成多级菜单。使用鼠标的第一按钮(一般是左键)或键盘上的组合键(如 Alt＋F 键)进行操作。

快捷菜单：在菜单栏以外程序窗口的工作区，通过鼠标的第二按钮(一般是右键)调出的菜单称为快捷菜单。根据调出位置的不同，菜单内容也不相同，其中列出了所关联的对象目前可以进行的操作。

4. 工具按钮及功能区(Ribbon)

把菜单中使用频繁的命令用图标表示出来，放置在窗口中较为显眼的位置称为工具按钮。应用程序中的按钮通常可以代替菜单，这样就不必通过菜单逐层翻动调出，从而提高了工作效率。但即使同一个应用程序，各种用户使用同一个命令的频率也是不一样的，因此工具按钮也可以由用户自定义。

Microsoft Office 2007 和 Windows 7 及以后版本的一些程序(如画图程序)中，使用了一种以皮肤及标签页为架构的功能区(Ribbon)用户界面，以替代传统的菜单栏、工具栏和下拉菜单。该界面将相关的选项组织在一组，将最常用的命令放到窗口的最突出位置，用户可以更轻松地找到并使用这些功能，并减少鼠标的点击次数，总体来说，比之前的下拉菜单

效率要高。例如，文件管理器【主页】主功能区中提供了核心的文件管理功能，包括复制、粘贴、删除、恢复、剪切、属性等(见图 1.10)。这些功能包括了用户日常的大部分操作。

图 1.10 Windows 10 的资源管理器的功能区

5. 图标

图标是在 GUI 操作系统桌面或程序中显示的代表应用程序或程序所管理的数据的图形符号，一般是一个指向相应程序或文件的链接(见图 1.11)。

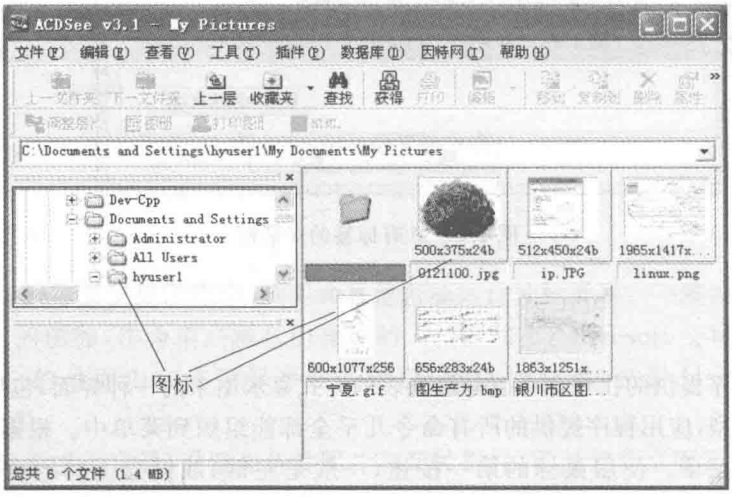


图 1.11 程序中代表数据的图标

文件夹(也称为目录)中的用户数据和程序管理的特定数据通过图标显示出来。通常情况下显示的是数据内容缩图，或与数据相关联的应用程序的代表图案。点击数据的图标，可以完成启动相关应用程序及显示数据本身两步工作。

应用程序的图标只能用于启动应用程序。

6. 对话框

对话框是 GUI 中的一种特殊窗口，用于向用户显示信息，或在需要的时候获取用户的响应，或者两者皆有。程序使用对话框与用户交互的方式就是计算机和用户之间进行"对话"。

非模态对话框：用于向用户请求非必需的信息，可以不理会这种对话框或不向其提供任何信息而继续进行当前工作。对话框所属程序窗口与该对话框窗口均可处于打开并处于活动状态。

模态对话框：这种对话框强制要求用户响应，在用户与该对话框完成交互之前不能再继续进行其他操作。模态对话框用在需要一些必需的信息然后才可以继续进行其他操作，或确认用户想要进行一项具有潜在危险性操作的情况下。模态对话框一般分为系统级和应用程序级。系统级对话框出现时，用户在完成与这个对话框的交互之前不能在该计算机系统进行其他操作，比如关闭对话框。应用程序级的模态对话框则只对它所属的程序有所限制。

1.2 Java GUI 概况

Java 的第一个版本在 1995 年发布的时候就包含了 AWT(Abstract Windowing Toolkit)库，用于构建图形用户界面应用程序。当时，使用 AWT 可以构建运行在浏览器中的 Java applet，极大地增强了网页的表现能力和交互能力。回顾 Java GUI 的发展和演化，主要有 3 个构建窗口程序的程序库：AWT、Swing 和 SWT。

1.2.1 AWT

起初，Java 技术令人激动的特性是基于 applet ——可以让程序通过 Internet 发布并在浏览器内执行的新技术。applet 简化了跨平台应用程序的开发、维护和发布，跨平台是商业软件开发中几个最富挑战性的题目之一。

为了方便用 Java 构建图形用户界面，Sun 最初提供了一个能在所有平台下运行的具有独特 Java 外观的图形界面库。当时，Sun 的首要伙伴 Netscape 提出 applet 应该与运行时平台具有一样的显示外观，在显示和行为上能够像该平台运行的其他应用程序一样。为了实现这个目标，在 JDK 的第一个发布版中包含了 AWT 库，使每一个 Java GUI 窗口部件都在底层的窗口系统中有一个对应的组件。但是，不同的操作系统平台提供的 GUI 元素总有一些不同，为了保持 Java 的“一次编写，到处运行”的特性，Sun 采用了“最大公约数”原则，即 AWT 只提供所有本地窗口系统都提供的 GUI 组件的公有集合，并映射到不同操作系统上的原生窗口组件(native widget)。

一个 AWT 组件通常是一个包含了对等体接口类型引用的组件类。这个引用指向本地对等体实现。如 java.awt.Label 类，它的对等体接口是 LabelPeer，具有平台无关性。在不同平台上，AWT 提供不同的对等体类来实现 LabelPeer。在 Windows 中，对等体类是 WlabelPeer，它调用 JNI(Java Native Interface，Java 本地接口)实现 label 的功能。这些 JNI 方法用 C 或 C++编写，它们关联到一个本地的 label，真正的行为都在本地发生。AWT 组件由 AWT 组件类和 AWT 对等体给应用程序提供了一个全局公用的 API(见图 1.12)。组件类和它的对等体接口是平台无关的，但它们调用的底层对等体类和 JNI 代码则是平台相关的。

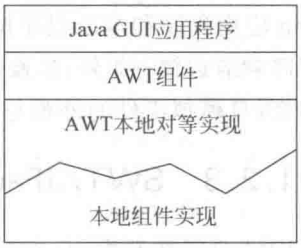


图 1.12 AWT 组件实现机制

由于 AWT 提供的 GUI 组件一般比本地操作系统平台使用的 GUI 组件少，因此需要为非公共子集的更多高级特性开发它们自己的窗口部件。此外，Java applet 运行在一个安全的“沙箱”里，并阻止恶意的 applet 对文件系统和网络

连接等资源的滥用。沙箱在提供安全性的同时，也减少了应用程序的功能。同时，Java GUI应用程序也不能像本地程序一样灵敏地响应。这些问题减缓了人们对 applet 的接受和承认速度。因为用 AWT 开发的应用程序既缺少流行 GUI 程序的许多特性，又不能达到在显示和行为上像用本地窗口构件库开发的程序一样的目标，所以需要一个更好的库来开发 Java GUI 程序。

1.2.2 Swing

1997 年在 JavaOne 大会上提出并于 1998 年 5 月发布的 JFC(Java Foundation Classes)中包含了一个新 GUI 组件库叫 Swing。Swing 使用 Java 开发了一套模拟的 GUI 组件库，遵循“最小公倍数”原则，除了依赖于 AWT 顶层容器(如 Applet、Window、Frame 和 Dialog 等)之外，几乎实现了所有平台上的标准组件。Swing 对组件特征的设计也遵循“最小公倍数”原则，它拥有所有平台上可提供的组件特征。

除了顶层容器外，Swing 的实现不依赖于具体平台，它掌控了所有的控制和资源。Swing 组件在操作系统中没有相应的对等体，普通的 Swing 组件可以看作是 AWT 容器的一块逻辑区域，从顶层容器(AWT 组件)的对等体中借用资源。所有添加到同一顶层容器的 Swing 组件共享它的 AWT 对等体以获取系统资源，如字体和图形处理等。Swing 将组件的数据结构存储在 JVM 的空间中，完全自主地管理绘制处理、事件分发和组件布局。Swing 的事件并不是底层系统产生的事件，而是由顶层容器处理 AWT 事件所产生的伪事件。

Swing 默认情况下采用本地平台的显示外观，此外还可以采用插件式的显示外观。因此 Swing 应用程序可以具有像 Windows 应用程序、Motif 应用程序、Mac 应用程序一样的显示外观，也可以拥有它自己的 metal 或 nimbus 显示外观。Swing 拥有很好的观感(Look And Feel)支持，甚至可以动态地改变 Swing 应用程序的观感。Look 指的是界面显示外观，Feel 指的是它如何响应用户操作。从 JDK 1.1.3 开始，Sun 在 Swing 中提供了三个 LookAndFeel 的子类，分别提供了 Metal、Motif 与 Windows 的界面样式，到 JDK 1.6 增加到 5 个。任何基于 Swing 的界面都可以使用这些观感的其中之一。此外，也可以通过直接或间接继承 LookAndFeel 类开发新的观感。

由于 Swing 自己实现了所有组件，因此程序运行时装载了大量的类，创建了大量小的可变对象，因而导致了额外的堆空间消耗。由于许多小的对象较大对象更难以有效地进行垃圾回收，且大量类的装载导致频繁的 I/O 操作，明显降低了 Java Swing 应用程序的启动和运行速度，从而导致性能下降。

Swing 使用 Java 开发 GUI 模拟组件而不是调用本地操作系统 GUI 库的方式，使 Swing 应用程序和本地程序拉开了一定差距，Windows 平台下的 Swing 程序显得比本地应用程序响应迟缓；此外，当操作系统的界面发生改变时，Swing 需要一段时间才能跟得上这种改变，且模拟组件的外观与本地系统的原生组件可能会有一些不同。

1.2.3 SWT/JFace

IBM 公司在开发 VisualAge for Java 时认为“Swing 是个可怕的充满缺陷的怪兽”，因此开始了一个新的项目——把 Smalltalk 原生窗口组件移植到 Java 上，这个工具集后来称

为 SWT(Standard Widget Toolkit)。他们当时发现 Swing 在读事件队列时用了一种可能留下内存漏洞的方式,因此决定 SWT 和 AWT/Swing 不能共存。IBM 公司把 SWT 工具包放到了 Eclipse 中。Eclipse 原本是 IBM 公司的开放源码计划 IBM WebSphere Studio Workbench 的通用工具平台,后来 IBM 公司把 Eclipse 捐赠出来。2001 年 11 月 7 日,SWT 作为 GUI 重要基础与 Eclipse IDE(Integrated Development Environment)一起集成发布,之后 SWT 发展和演化为一个独立的版本。使用 SWT 可以开发 Microsoft Windows、Mac OS X 以及几种不同风格的 UNIX/Linux 等操作系统下的 Java GUI 程序。

SWT 的设计采用"最小公倍数"原则提供各个平台上包含组件的并集。如果一个组件在操作系统平台中已经提供,SWT 就包装并用 Java 代码和 JNI 调用它。反之,如果某个组件在某一平台上不存在,就继承并以绘制 Composite 的方式模拟该组件。不同于 Swing 的模拟方式,SWT 的 Composite 类有操作系统相应的对等体,它从该对等体中获得所需资源,如图形处理的对象、字体和颜色等。SWT 直接从操作系统获取所有的事件并进行处理,将组件的控制权交给本地操作系统。

可见,在实现机制方面 SWT 吸收了 AWT 和 Swing 的优点,当可以得到本地组件时使用本地组件实现,不能得到本地组件时则使用 Java 模拟实现。这样既保证了 SWT GUI 组件与本地窗口部件最大程度地具有一致的外观和响应速度,又提供了足够丰富的组件类型和特性。

JFace 的构建基于 SWT,提供了在 SWT 基础之上的抽象层,是对 SWT 组件的更进一步的 OOP(面向对象程序设计)封装,提供了 MVC 模式。SWT 使用直接的 API 提供了原生的窗口部件,而 JFace 对抽象层编程,抽象层与 SWT API 交互。例如,SWT 创建表时一般应创建一个 table 组件并且插入要显示的行和列的数据。而要使用 JFace 中的 table,则应先创建 table 组件,但不向表格插入数据,而是指定表格查看器和内容器、标签器及输入数据对象,由表格查看器决定数据内容和显示。JFace 的目的不是取代 SWT,而是为一些复杂编程任务提供更为简单的实现机制和方法,与 SWT 组件共同完成程序功能。

综上所述,3 个 Java GUI 库的优缺点如表 1.1 所示。

表 1.1 三个 Java GUI 库的优缺点

特 性	AWT	Swing	SWT/JFace
组件类型	少(最小子集)	多(最大子集)	丰富
组件特性	少(最小子集)	多(可定制)	丰富(平台+模拟)
响应速度	快	快	快
内存消耗	少	多	少
扩展性	无扩展性	强	不可扩展
Look And Feel	不支持	出色支持	不支持
成熟稳定性	好	好	Windows 平台高 其他平台不高
总体性能	高	一般	Windows 平台高 其他平台不高
API 模型支持	无	MVC	MVC
GUI 库来源	JRE 标准工具集	JRE 标准工具集	程序捆绑
启动速度	快	慢	快
可视化编程	可用	支持	支持

Java发布之初就提供了构建跨平台应用的窗口GUI库，随着对AWT、Swing和SWT/JFace库的持续改进开发，其组件种类、特性、响应和运行速度、对系统资源的消耗、构建Java GUI应用程序的方便快捷性、运行的效率和稳定性等方面都取得了巨大进步，同时计算机硬件本身运行速度也有了极大提升，从而使Java成为一个构建桌面应用程序的可行选择，也使之成为一个具有优势的桌面程序开发平台。鉴于Windows桌面系统在国内庞大的用户群，以及Eclipse在Java开发环境方面的高普及率，本书选择SWT/JFace为可视化构建Java GUI程序的组件库。

1.3 Java GUI程序的实现原理

1.3.1 程序的图形用户界面显示原理

一般地，计算机程序的用户界面显示在计算机的显示屏上。图形用户界面则以图像的方式显示在屏幕上。当应用软件需经显示屏与用户通信时，它首先要以虚屏方式建立消息，然后将需要显示的消息作为内存块，从应用软件提交给操作系统。操作系统再把它格式化成表示图形或文本消息的像素图案，并传送到显示适配器的存储器中。显示适配器硬件读取这些格式化信息，再“画”到PC显示设备上。

组成显示屏上图像的点称为像素(pixel)。在彩色LCD(液晶显示器)面板中，每一个像素都是由3个液晶单元格构成的，其中每一个单元格前面都分别有红色、绿色或蓝色的过滤片。光线经过过滤片的处理照射到每个像素中不同色彩的液晶单元格上，利用三原色的原理组合出不同的色彩。程序的图形界面显示为一副图像，每个图像像素映射为一个显示器像素。描述像素图案的信息包括组成图像的每个点的坐标、颜色、亮度等。

屏幕坐标与显示器分辨率密切相关。显示器分辨率是指显示器所能显示的像素数。屏幕分辨率为1024 × 768，表示当前屏幕水平方向被划分为1024个单位，竖直方向被划分为768个单位，每个水平与竖直方向交界处即为一个屏幕像素，程序中用坐标描述其位置。与数学中的直角坐标系不同，屏幕的左上角为原点，记为(0,0)，水平坐标从左向右依次增加、竖直坐标从上向下依次增加。

图像表示的颜色数取决于每个像素使用的显示存储器的位数。如果用8位存储一个像素，则每像素可有256种颜色，用16位则有65 536种颜色，用24位就可有16.8M种颜色。显示适配器将图像各像素的R(红)、G(绿)、B(蓝)色值等信息传送给显示器，即可在显示屏上显示GUI界面彩色图像。

1.3.2 Java GUI程序的构成

桌面Java GUI程序一般在窗口中运行。从前面介绍的GUI基本构成(见图1.8)可知，首先要生成和管理一个窗口。窗口作为一个Java GUI程序的容器，其中包含了图标、按钮、菜单等组件。在软件开发中，组件通常指可重复使用并且可以和其他对象进行交互的对象，控件则是提供(或实现)用户界面(UI)功能的组件，是以图形化的方式显示在屏幕上并与用户进行交互的对象。在以可视化方法设计SWT/JFace GUI时，除了用到窗口、按钮和表格

等控件外，还会用到不需要显示任何信息或用户界面的组件，如查看器(Viewer)等。为了简化叙述，本书不加区分地将它们统一称为组件。为了对相关组件进行分类组织和统一管理，可以把这些组件一起放到一个容器中。容器是一种能够容纳其他组件或容器的特殊组件。如图 1.13 所示，使用容器能把两种性别和三种爱好组织在一起形成分组。

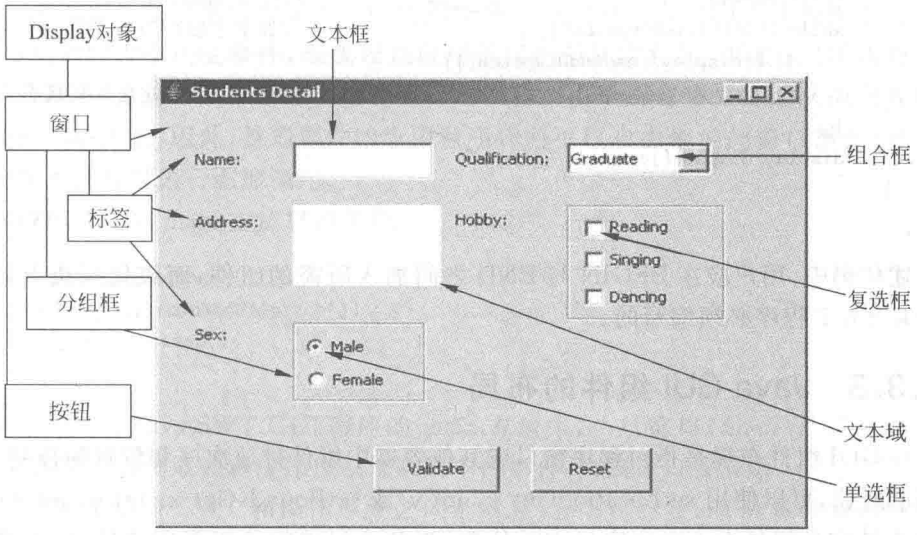


图 1.13 一个简单的 Java GUI 界面

Java SWT GUI 程序结构如程序清单 1.1 所示。

程序清单 1.1：

```
package cn.edu.hyedu.swt;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;

public class HelloWorld {
    public static void main(String[] args) {
        final Display display = Display.getDefault();
        final Shell shell = new Shell();                    //shell 对象是程序的主窗口
        shell.setSize(327, 253);                            //设置主窗口的大小
        shell.setText("Hello World");                       //设置主窗口的标题
        //---------- BEGIN 创建窗口中的其他界面组件 ----------
        Button button = new Button(shell, SWT.NONE);        //创建一个按钮对象
        button.setText("HelloWorld");                       //设置按钮上的文字
        button.setBounds(88, 94, 100, 25);                  //设置按钮在窗口中的位置和大小
        //编写按钮被单击时的事件代码
        button.addSelectionListener(new SelectionAdapter() {
            public void widgetSelected(SelectionEvent e) {
                //弹出一个对话框，MessageDialog 是 JFace 中的类
```

```
                MessageDialog.openInformation(shell,"hello","HelloWorld");
            }
        });
        //------------------ END 其他界面组件与事件代码 -----------------
        shell.layout();                                    //应用界面布局
        shell.open();                                      //打开 shell 主窗口
        while (!shell.isDisposed()) {                      //如果主窗口没有关闭,则一直循环
            if (!display.readAndDispatch())
                display.sleep();                           //让 display 处在休眠状态
        }
        display.dispose();
    }
}
```

上述代码中,用户应在 BEGIN 与 END 之间加入所需的组件,而该代码块上边和下边的部分是 SWT 程序必须编写的。

1.3.3 Java GUI 组件的布局

Java GUI 组件布局是指对程序窗口或其他容器中组件排放次序和位置的控制。

如前所述,可以使用 setLocation(int x, int y)或 setBounds(int x, int y, int width, int height)方法定位组件左上角在窗口中的位置,当各个组件的位置指定之后,也就确定了它们的排放次序。这种方式可以精确控制每一个组件在窗口中的排布,但当窗口大小改变之后,一些组件可能出现在窗口边框之外而不可见,一些组件可能离窗口边框太远,一些边框周围组件太过拥挤,一些边框周围太空旷,结果破坏了原本设计得很美观协调的界面。况且每种类型操作系统对屏幕的定义不一样,界面在一种视窗系统下很美观,但到了另一种下就未必。

为了解决组件绝对定位存在的问题,Java 的各种 GUI 库都提供了托管定位的方式进行自动布局管理,具体工作是由一种叫布局管理器(Layout Manager)的对象完成的。给窗口组件指定了布局管理器并设置了布局信息后,窗口组件显示时将会调用相应的布局方法对窗口组件的子组件进行布局、定位和计算子组件大小的操作,从而使窗口组件以更友好的方式显示在父组件中。

1.3.4 用户交互与事件循环

用户单击鼠标、输入字符或者对窗口的大小等进行调整时,操作系统都将生成应用程序 GUI 事件,例如,鼠标单击事件、按键事件或者窗口绘制事件,确定哪个窗口和应用程序应当接收事件,并把事件添加到应用程序的事件队列中。

任何窗口化的 GUI 应用程序的底层结构都是事件循环。应用程序初始化并启动事件循环,从事件队列中读取 GUI 事件,并相应地做出反应。使用 C 语言的本机 GUI 程序员对使用平台事件循环相当熟悉。但是,用 Java 语言编写的较高级别的窗口工具箱通常试图通过隐藏平台事件循环来向应用程序开发者屏蔽用户界面线程。实现此过程的常见方法是设置专用工具箱用户界面线程,以便读取和调度事件循环,并将事件送至正在单独线程中运行的应用程序服务内部队列中。

目前的GUI平台对事件队列进行了许多优化。常见的优化是将连续的绘制事件叠加到队列中。每当必须重新绘制窗口的一部分时，检查队列是否存在绘制事件重叠或尚未调度的冗余绘制事件，将这些事件合并到一个绘制事件中，从而减少屏幕闪烁，使应用程序的绘制代码不会非常频繁地执行。

SWT遵循操作系统平台直接支持的线程模型。应用程序在它的主线程中运行事件循环，并直接从此线程中调度事件，该线程是应用程序的“UI线程”。事实上UI线程既是创建显示的线程，也是运行事件循环和创建窗口的线程。由于所有事件都是从应用程序的用户界面线程中触发的，因此，处理事件的应用程序代码可以自由地访问窗口组件，且不需要任何特殊技术就可以进行图形调用。

程序清单1.1中main方法里的代码：

```
while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
        display.sleep();
}
```

即为事件循环。在Java SWT GUI程序的main方法中，一旦窗口(Shell)创建并处于打开状态，应用程序就会读取和调度操作系统队列中的事件，直到窗口被清除。

Display对象负责SWT和操作系统之间的联系，将SWT/JFace的各种调用转化为系统的底层调用，控制操作系统为SWT分配的资源，获得操作系统有关信息。在Display的事件循环中，同时处理着系统队列和自定义队列中的事件。display.readAndDispatch的执行流程是：首先从系统事件队列中读取消息，如果读到该程序的事件，就将它发送到窗口去处理；如果该程序没有系统事件，则处理自定义事件队列中的事件，处理完该事件后返回true。如果两个队列中都没有事件返回false，则Display调用sleep方法，显示线程进入睡眠状态，以使其他应用线程有机会运行。若事件队列中有新的事件传来，则UI线程会被唤醒并恢复事件循环过程。

1.4 可视化程序设计

1.4.1 可视化程序设计的概念

可视化程序设计也称为可视化编程，是以“所见即所得”的思想为原则，通过直观的操作方式进行界面的设计，并即时在设计环境看到在运行环境的实际表现结果，从而实现编程工作的可视化及程序代码的自动生成。其实质是设计过程可视化，设计结果即时呈现。通俗地讲，就是“看着画”界面。

此外，把科学数据、工程数据和测量数据等及时、直观形象和客观地以图形化方式呈现出来，并进行交互处理的计算机程序设计技术也叫可视化程序设计。它涉及计算机图形学、图像处理、计算机辅助设计、计算机视觉及人机交互技术等多个领域。近年来，随着网络技术和电子商务的发展，出现了信息可视化(Information Visualization)技术。通过数据可视化技术(Data Visualization)，能够发现大量金融、通信和商业数据中隐含的规律，从而为决策提供依据。数据可视化和信息可视化技术是近几年的一个研究热点。

本书讨论的可视化程序设计是指第一种，即图形用户界面设计手段的可视化技术。

1.4.2 可视化程序设计发展简况

从发明之初，计算机就以存储指令和执行指令序列作为基本工作方式。早期的计算机用户接口是文字形式的命令(或称指令)。不仅操作系统如此，应用软件也是如此。如 PC 中 20 世纪 80 年代风行的 Word Star 和 WPS 等字处理软件，用户都是将排版指令嵌入到文字之中，然后由软件对指令进行解释打印出文稿，或者以图形方式模拟显示出打印效果。

随着 Windows 3.1(1992 年)在个人计算机中的应用和普及，图形用户界面成为了主要的人机界面，使应用软件也出现了“所见即所得”的工作方式。在 Word Perfect、MS Word 5.0 以及新版的 WPS 等字处理软件中，排版指令的结果在下达指令的同时即刻显示在工作界面中。这种方式提高了工作效率，降低了软件的使用难度，拉近了计算机与用户的距离。

在程序设计领域，一贯都是程序员在编辑器或集成开发环境(IDE)中输入程序代码，然后编译、解释运行得到实际效果(结果)。但是，在设计 GUI 程序的时候，这种方式效率很低，设计时得到什么效果一般靠设计者的经验和形象思维进行预判。这对程序员要求就比较高，且想象的结果与实际显示结果之间有或大或小的差距。随着图形用户界面的普及，设计程序的 GUI 成为必需。为了降低 Windows 等图形用户界面系统下的 GUI 设计难度，人们也期望对 GUI 的设计能够以“所见即所得”的方式进行。

1991 年，Microsoft 公司推出了 Visual Basic 1.0 版，它是第一个“可视化”编程软件，有效地连接编程语言和用户界面设计，因此一出现就得到了程序员的喜爱。Visual Basic 最初是基于 DOS 系统的，但它真正的成功是在 Windows 系统下取得的。由于 Visual Basic 基于语法简单的 Basic 语言，又是以“画图”的简单直接方式设计用户界面，因此从一出现就吸引了大批程序员和非程序员，在美国甚至连 10 岁的小孩也能利用 Visual Basic 编写小的应用程序。

继 Visual Basic 获得成功之后，可视化开发工具的发展在 20 世纪 90 年代达到了高潮。Windows 为开发 GUI 应用程序提供了一整套机制和 API(应用程序接口)。首先，Windows 程序的运行采用事件驱动机制，即程序的运行逻辑围绕事件的发生展开，事件驱动程序设计是围绕着消息的产生与处理展开的；把 WinMain 函数作为 Windows 应用程序的入口点，该函数采用 C 语言的语法，包括过程说明、程序初始化和消息循环三部分；WinMain 函数的消息循环管理并发送各种消息到相应的窗口函数中；窗口过程 WinProc 接收并用 switch 处理各种消息。从方便性和效率方面来说，C 语言和 C++语言对于 Windows 应用程序的开发具有天然的优势。1992 年 Microsoft 公司发布 Visual C++ 1.0，之后持续对 VC++进行改进，使其具备了逐步完善的可视化 GUI 设计功能。与 VC++竞争的还有 Borland C++，之后发展为 C++ Builder。后者具有更加强大和完善的可视化设计功能。尽管具有运行效率高等优势，但是 C++过于复杂，因此有个说法是“真正的程序员用 C/C++，聪明的程序员用 Delphi”。Delphi 是 Borland 公司发布的 Windows 平台下可视化的快速应用程序开发环境(Rapid Application Development，RAD)，其 1.0 版本于 1995 年发布。Delphi 的核心是由传统 Pascal 语言发展而来的 Object Pascal，采用图形用户界面开发环境，通过 IDE、VCL(Visual Component Library)工具与编译器，配合连接数据库的功能，构成以面向对象程序设计为中心的应用程序开发环境。

在 Java GUI 开发领域，Borland JBuilder、IBM 公司的 Visual Age for Java、Microsoft

公司的 Visual J++、SUN 公司的 Java Workshop、WebGain 的 Visual Café 等 Java 开发工具一般都具有一定的可视化开发功能。目前主流的 Java 开发平台包括 Eclipse、NetBeans 和 IntelliJ IDEA 等。其中，NetBeans 是来自 Sun Microsystems 的自由、开放源代码的 Java IDE，采用基本系统(Base)＋企业包(EnterprisePack)＋可视化包(Visual Web Pack)方式发布，对基于 AWT 和 Swing 的 Java GUI 设计提供了设计视图、组件面板和属性面板等可视化设计工具(见图 1.14)。

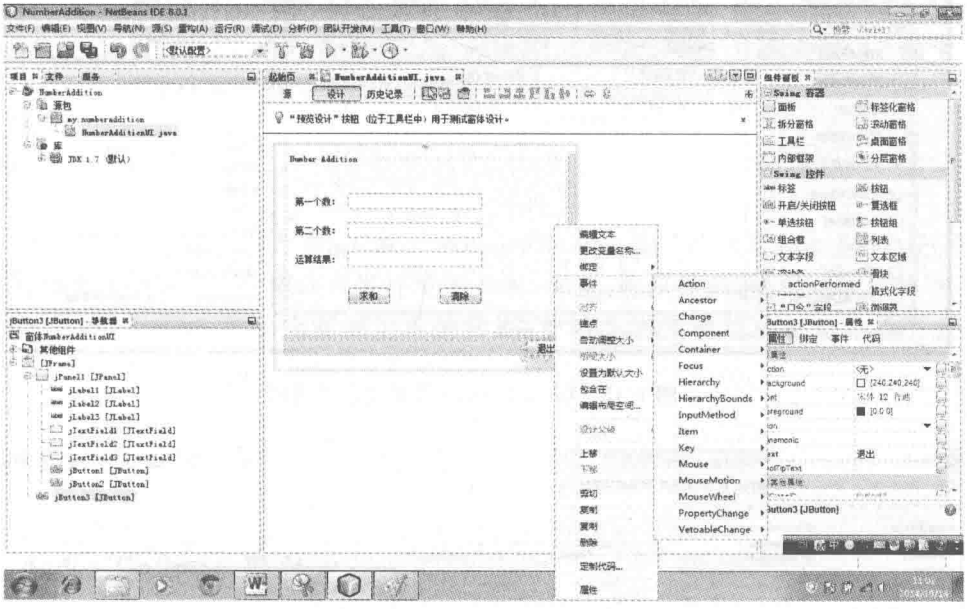


图 1.14 NetBeans 6.7.1 可视化设计工具

IntelliJ IDEA 是 JetBrains 公司发布的一种商业化销售的 Java 集成开发环境，2001 年 1 月发布 IntelliJ IDEA 1.0 版本，IntelliJ IDEA 2016.2.5 于 2016 年 10 月发布。该平台被称为是最好最智能的 Java IDE 开发平台，也提供了对基于 AWT 和 Swing 的 Java GUI 可视化设计支持，包括设计视图、组件选择面板和属性设置面板等(见图 1.15)。

Eclipse 是在 Eclipse.org 协会管理和指导下开发和维护的一个开放源代码的基于 Java 的可扩展开发平台，由不同的组织和公司通过插件的方法提供支持 Java GUI 可视化开发的工具，其中有 Eclipse.org 组织开发的 Visual Editor、Google 贡献的 WindowBuilder(原 Instantiations 的 SWT Designer)、商业机构付费使用但个人可以免费使用的 jigloo 等，都支持 AWT、Swing 和 SWT/JFace 库(见图 1.16)。鉴于 Eclipse 近年来获得的巨大成功和庞大的用户群等优势，JBuilder 从 JBuilder 2007 开始就基于 Eclipse 开发，但同时保留了其优秀的可视化设计库和工具。

基于普及程度、对 SWT/JFace 支持与否及是否免费等方面的考虑，本书选用安装有 WindowBuilder 1.9 插件的 Eclipse 4.6 开发平台作为 Java GUI 可视化程序设计的环境，讲述 Java SWT GUI 的设计思想、图形用户界面的可视化设计方法、各种 GUI 组件的可视化设计及属性的直观设置、组件与用户交互的事件处理代码设计等内容。