JAVA LOOK AND FEEL DESIGN GUIDELINES

ALPHA DRAFT

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303 USA
650 960-1300 fax 650 969-9131

Part No.: 806-0026-01
Revision A, June 1999

Java Look and Feel Design Guidelines

Copyright 1999 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303 U.S.A. All rights reserved.

This product or documentation is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or documentation may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Sun, Sun Microsystems, the Sun logo, Java, JavaOS, JavaHelp, Java 2D, HotJava, JavaBeans, Java Plug-in, Java Development Kit, JDK, PersonalJava, Java Foundation Classes, JavaScript, JavaStation, Java virtual machine, the Java Coffee Cup logo, Solaris, and Write Once, Run Anywhere are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

U.S. Government approval required when exporting the product.

DOCUMENTATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, ANY KIND OF IMPLIED OR EXPRESS WARRANTY OF NON-INFRINGEMENT OR THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright 1999 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, Californie 94303 Etats-Unis. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Sun, Sun Microsystems, le logo Sun, Java, JavaOS, JavaHelp, Java 2D, HotJava, JavaBeans, Java Plug-in, Java Development Kit, JDK, PersonalJava, Java Foundation Classes, JavaScript, JavaStation, Java virtual machine, Java Coffee Cup logo, Solaris, et Write Once, Run Anywhere sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun. L'accord du gouvernement américain est requis avant l'exportation du produit.

LA DOCUMENTATION EST FOURNIE "EN L'ETAT" ET TOUTES AUTRES CONDITIONS, DÉCLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES DANS LA MESURE AUTORISÉE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE À LA QUALITÉ MARCHANDE, À L'APTITUDE À UNE UTILISATION PARTICULÈRE OU À L'ABSENCE DE CONTREFAÇON.

Please Recycle

CONTENTS

FIGURES

TABLES

PREFACE

Java Look and Feel Design Guidelines provides essential information for anyone involved in the process of creating cross-platform Java™ applications and applets. In particular, this book offers design guidelines for software that uses the Java™ Foundation Classes (JFC) together with the Java look and feel. (Unless specified otherwise, this book uses "application" to refer to both applets and applications.)

Who Should Use This Book

Although the human interface designer and the software developer might well be the same person, the two jobs require different tasks, skills, and tools. Primarily, this book addresses the **designer** who chooses the interface components, lays them out in a set of views, and designs the user interaction model for an application. This book should also prove useful for developers, technical writers, graphic artists, production and marketing specialists, and testers who participate in the creation of Java applications and applets.

Java Look and Feel Design Guidelines focuses on design issues and human-computer interaction in the context of the Java look and feel. It also attempts to provide a common vocabulary for designers, developers, and other professionals. If you require more information about technical aspects of the Java Foundation Classes, visit the Swing Connection web site at `http://java.sun.com` or `http://java.sun.com/products/jfc`.

The guidelines provided in this book are appropriate for applications and applets that run on personal computers and network computers. They do not address the needs of software that runs on consumer electronic devices.

How This Book Is Organized

Java Look and Feel Design Guidelines includes the following chapters:

Chapter 1, "The Java Look and Feel," introduces the design concepts underlying the Java look and feel and offers a quick visual tour of an application and an applet designed with the JFC components and the Java look and feel.

Chapter 2, "Java Foundation Classes," provides an overview of the Java Foundation Classes, suggests effective ways to use the JFC components, and describes the concept of pluggable look and feel designs.

Chapter 3, "Design Considerations," discusses some of the fundamental challenges of designing Java look and feel applications and offers recommendations for applet design, accessibility, internationalization, and localization.

Chapter 4, "Visual Design," provides suggestions for the use of the Java look and feel themes mechanism to change color and fonts in your application, provides guidelines for the capitalization of text in the interface and makes recommendations for layout and visual alignment.

Chapter 5, "Designing Application Graphics," discusses the use of cross-platform color, the creation of application graphics to fit with the Java look and feel, and the design of graphics to enhance corporate and product identity.

Chapter 6, "Behavior," tells how users of Java look and feel applications utilize the mouse, keyboard, and screen. It provides recommendations regarding user input and human-computer interaction, including a discussion of drag and drop operations.

Chapter 7, "Windows, Panes, and Frames," discusses and makes recommendations for the use of primary, secondary, and utility windows as well as scroll panes, tabbed panes, and split panes.

Chapter 8, "Dialog Boxes," describes and makes recommendations for the use of dialog boxes, the supplied alert boxes, and the color chooser.

Chapter 9, "Menus and Toolbars," presents details about and makes suggestions for the use of drop-down menus, contextual menus, toolbars, and tool tips.

Chapter 10, "Basic Controls," covers the use of controls such as command buttons, toggle buttons, checkboxes, radio buttons, sliders, and combo boxes. It also describes progress bars and provides suggestions for their use.

Chapter 11, "Text Components," explains and makes recommendations for the use of the JFC components that control the display and editing of text.

Chapter 12, "Lists, Tables, and Trees," discusses and makes recommendations for the use of lists, tables, and tree views.

Appendix A, "Keyboard Navigation, Activation, and Selection," contains tables that specify keyboard sequences for the components of the Java Foundation Classes.

Glossary defines important words and phrases found in this book. They appear in boldface at first occurrence.

Graphic Conventions

Screen shots in this book illustrate the use of JFC components in applications with the Java look and feel. Because such applications typically run inside windows provided and managed by the native platform, which might include, among many others, Microsoft Windows, Macintosh, or CDE (Common Desktop Environment), the screen shots show assorted styles of windows and dialog boxes.

Throughout the text, symbols are used to call your attention to design guidelines. Each type of guideline is identified by a unique symbol.

Java Look and Feel Standards Requirements for the consistent appearance and compatible behavior of Java look and feel applications. These standards promote flexibility and ease of use in cross-platform applications and the creation of applications that support all users, including users with physical and cognitive limitations. These standards require you to take actions that go beyond the provided appearance and behavior of the JFC components.

Occasionally, you might need to violate these standards. In such situations, use your discretion to balance competing requirements. Be sure to engage in user testing to validate your judgments.

Cross-Platform Delivery Guidelines Recommendations for dealing with colors, fonts, keyboard operations, and other problems that arise when you want to deliver your application to a variety of computers running a range of operating systems.

Internationalization Guidelines Advice for creating applications that can be adapted to the global marketplace.

Implementation Tips Technical information and useful tips of particular interest to the programmers who are implementing your application design.

Related Books and Web Sites

This book does not provide detailed discussions of human interface design principles, nor does it present much general information about application design. However, many excellent references are available on topics such as fundamental principles of human interface design, design issues for specific (or multiple) platforms, and the issues relating to accessibility, internationalization, and applet design.

Design Principles

The resources in this section provide information on the fundamental concepts underlying human-computer interaction and interface design.

Baecker, Ronald M., William Buxton, and Jonathan Grudin. *Readings in Human-Computer Interaction: Toward the Year 2000,* 2nd ed. Morgan Kaufman Publishing, 1995. A collection of research from graphic and industrial design, and cognition and group process, this volume addresses the efficiency and adequacy of human interfaces.

Hurlburt, Allen. *The Grid: A Modular System for the Design and Production of Newspapers, Magazines, and Books*. John Wiley & Sons, 1997. This is an excellent starting text. Although originally intended for print design, this book contains many guidelines that are applicable to software design.

IBM Human-Computer Interaction Group. "IBM Ease of Use." Available: `http://www.ibm.com/ibm/easy`.This web site covers many fundamental aspects of human interface design.

Laurel, Brenda, ed. *The Art of Human-Computer Interface Design*. Addison-Wesley, 1990. Begun as a project inside Apple, this collection of essays from computer industry experts explores strategies and reasoning behind human-computer interaction and looks at the future of the relationship between humans and computers. It surveys diverse design techniques and examines work in drama and narrative, industrial design, animation, and cognitive and interpersonal psychology.

Mullet, Kevin, and Darrell Sano. *Designing Visual Interfaces: Communication-Oriented Techniques.* Prentice-Hall, 1995. This volume covers fundamental design principles, common mistakes, and step-by-step techniques in several visual aspects of interface design: elegance and simplicity; scale, contrast, and proportion; organization and visual structure; module and program; image and representation; and style.

Nielsen, Jakob. *Usability Engineering.* Academic Press, 1994. This classic contains a substantial chapter on international user interfaces, including gestural interfaces, international usability engineering, guidelines for internationalization, resource separation, and interfaces for more than one locale.

Norman, Donald A. *The Design of Everyday Things.* Doubleday, 1990. A well-liked, amusing, and discerning examination of why some products satisfy users while others only baffle and disappoint them. Black-and-white photographs and illustrations throughout complement the astute analysis.

Shneiderman, Ben. *Designing User Interface Strategies for Effective Human-Computer Interaction*. 3rd Edition. Addison Wesley, 1997. The third edition of the best seller, which provides a complete, current, and authoritative introduction to user interface design. Shneiderman writes new chapters on the World Wide Web, information visualization, and computer-supported cooperative work and expands earlier work on development methodologies, evaluation techniques, and user-interface-building tools.

Tognazzini, Bruce. *TOG On Interface.* Addison-Wesley, 1992. Based on a human interface column that Tog wrote for Apple developers, this book delves into the pivotal challenges of user interface design, including the difficulties inherent in multimedia software.

Tufte, Edward R. *Envisioning Information.* Graphics Press, 1990. One of the best books on graphic design, this volume catalogues instances of superb information design, with an emphasis on maps and cartography. Tufte culls examples from Galileo's observations of Saturn, a three-dimensional map of a Japanese shrine, a visual proof of Pythagoras' theorem, color studies by the artist Joseph Albers, and a New York train schedule, and he explicates the concepts behind their implementation.

Tufte, Edward R. *The Visual Display of Quantitative Information,* reprint edition. Graphics Press, 1992. Tufte explore the issues relevant to the graphical presentation of statistical information in charts and graphs. The design of the book exemplifies its principles: exquisite typography and layout, and splendid weaving of clear writing and apt graphical examples.

Tufte, Edward R. *Visual Explanations: Images and Quantities, Evidence and Narrative.* Graphics Press, 1997. As the third volume in Tufte's series on information display, this book focuses on data that changes over time. Tufte explores the explosion of the space shuttle Challenger (which could have been prevented, Tufte argues, by better information display on the part of the rocket's engineers), magic tricks, a cholera epidemic in 19th-century London, and the principle of using "the smallest effective difference" to display distinctions in data.

Design for Specific Platforms This section provides a list of informative resources on application design for the CDE, IBM, Java, Macintosh, and Microsoft Windows platforms.

CDE Three volumes address the needs of designers and related professionals creating applications using CDE and Motif 2.1.

The Open Group. *CDE 2.1/Motif 2.1–Style Guide and Glossary*, 1997.

The Open Group, 1997. *CDE 2.1/Motif 2.1–Style Guide Reference*.

The Open Group, 1997. CDE 2.1/Motif 2.1–Style Guide Certification Checklist.

They can be ordered from the Open Group at `http://www.opengroup.org/public/pubs/catalog/mo.htm`.

IBM *Object-oriented interface design: IBM Common User Access Guidelines*. Que Corp, 1992. Available: `http://www.ibm.com/ibm/hci/guidelines/design/ui_design.html`. This book is out of print but available from most or all IBM branch offices. A small portion of the IBM printed book *Common User Access Guidelines* is intertwined with a modest amount of more current material in this IBM web site.

Java Eckstein, Robert, Mark Loy, and Dave Wood. *Java Swing*. O'Reilly & Associates, 1998. An excellent introduction to the most current developments in Java interface technology, this book discusses how (and why) to use Swing components. It also provides comprehensive documentation on the Swing and Accessibility application programming interfaces. A detailed chapter explains how to create a custom look and feel.

Geary, David M. *Graphic Java 1.2: Mastering the JFC.* 3 vols. Prentice Hall, 1998. A comprehensive guide to the Java Foundation Classes, these three volumes describe the skills needed to build professional, cross-platform applications that take full advantage of the Java Foundation Classes. The books include chapters on drag and drop, graphics, colors and fonts, and image manipulation. Geary covers double buffering, sprite animation, clipboard and data transfer, and other advanced topics.

Topley, Kim. C*ore Java Foundation Classes.* Prentice-Hall Computer Books, 1998. After a tour of the JFC/Swing packages, the book explains how to build basic Swing applications, focusing on layout managers (which control the arrangement of JFC components on screen) and basic graphics programming. The book also describes menus and the toolbar, keyboard handling (and mouseless operation), dialog boxes, tree and table components, and the creation of multiple document interface (MDI) applications.

Macintosh *Macintosh Human Interface Guidelines.* Addison-Wesley, 1992. This volume is the official word on Macintosh user-interface principles. It includes a superb bibliography with titles on animation, cognitive psychology, color, environmental design, graphic and information design, human-computer design and interaction, language, accessibility, visual thinking, and internationalization.

Apple Computer, Inc. *Mac OS 8 Human Interface Guidelines.* Available: `http://developer.apple.com/techpubs/mac`.This site offers a supplement to *Macintosh Human Interface Guidelines.*

Microsoft Windows *Windows Interface Guidelines for Software Design*. Microsoft Press, 1995. Available: `http://www.microsoft.com/win32dev/uiguide`. The official book on Microsoft user interface design is essentially a specification for designers who would like to save training time, increase productivity, and enhance user confidence in their programs. These guidelines are available in print and a modest portion of them is on the World Wide Web. You can download an addendum to the Microsoft *Windows Interface Guidelines for Software Design* book at `http://msdn.microsoft.com/developer/userexperience/winuiguide.html`.

Design for Multiple Platforms The books in this section discuss issues relevant to designing software to run on many operating systems.

McFarland, Aland, and Tom Dayton (with others). *Design Guide for Multiplatform Graphical User Interfaces* (LP-R13). Bellcore, 1995. (Available only from Bellcore. Call 800-521-2673 from US & Canada, +1-908-699-5800 from elsewhere.) This is an object-oriented style guide with extensive and detailed guidelines, and a good explanation of object-oriented user interface style from the user's perspective. McFarland and Dayton primarily provide a very detailed reference although explanations of overall style of GUI objects and interaction are also included.

Marcus, Aaron, Nick Smilonich, and Lynne Thompson. *The Cross-GUI Handbook: For Multiplatform User Interface Design.* Addison-Wesley, 1995. This source is handy for anyone designing software to run on a large number of platforms. The book describes the graphical user interfaces of Microsoft Windows and Windows NT, OSF/Motif, NeXTSTEP, IBM OS/2, and Apple Macintosh. The text includes design guidelines for portability and migration, and recommendations for handling contradictory or inadequate human interface guidelines.

Design for Internationalization This section lists resources on designing your software for adaptation to the global marketplace.

Fernandes, Tony. *Global Interface Design: A Guide to Designing International User Interfaces*. Boston AP Professional, 1995. Fernandes addresses developers of internet software designed for a global market. He explains what needs to be considered about languages and their variations, visual components, national formats, familiar objects, taboos, aesthetics, and ergonomic standards.

Guide to Macintosh Software Localization. Addison-Wesley, 1992. A thorough and thoughtful discussion of the internationalization and localization processes that should prove helpful for developers on any platform.

Kano, Nadine. *Developing International Software for Windows 95 and Windows NT*. Microsoft Press, 1993. Kano presents Microsoft's guidelines for creating international software targeting an audience of Microsoft programmers and interface designers with knowledge of Microsoft Windows coding techniques and C++. The work contains information on punctuation, sort orders, locale-specific code-page data, DBCS/Unicode mapping tables, and multilingual API functions and structures.

Luong, Tuoc V., James S.H. Lok, and Kevin Driscoll. *Internationalization: Developing Software for Global Markets*. John Wiley & Sons, 1995. The Borland internationalization team describes its procedures and methods with a focus on testing and quality assurance for translated software. This hands-on guide tells how to produce software that runs anywhere in the world without requiring expensive recompiling of source code. The authors go beyond theory to provide actual solutions to everyday problems.

Nielsen, Jakob, and Elisa del Galdo. *International User Interfaces*. John Wiley & Sons, 1996. This book explores what user interfaces can and must do to become commercially viable in the global marketplace. Contributors explore issues such as international usability engineering, developing a cultural model, managing a multiple-language document system, and an intelligent lexical management system for multilingual machine translation. The book includes chapters on cultural issues that affect training, breaking the language barrier with graphics, and Arab and Chinese text displays.

O'Donnell, Sandra Martin. *Programming for the World: A Guide to Internationalization*. Prentice-Hall, 1994. This handbook explains how to modify computer systems to accommodate the needs of international users. O'Donnell takes readers on a tour of the many linguistic and cultural conventions used throughout the world, discussing how to break old

programming habits to design with the flexibility needed to handle differing user needs. More focused on theory than practice, this book explains why certain practices spell trouble when it comes to localizing products.

Uren, Emmanuel, Robert Howard, and Tiziana Perinotti. S*oftware Internationalization and Localization: An Introduction*. Van Nostrand Reinhold, 1993. This professional guide to software adaptation aims at enabling localized software to do the same things as the original software while integrating local user rules and conventions.

Design for Accessibility These resources explore considerations relevant to software designing software to support all users, including those with physical and cognitive limitations.

Sun Microsystems, Inc. *Accessibility Quick Reference Guide.* Available: `http://www.sun.com/tech/access/access.quick.ref.html`. This site defines accessibility, tells why it is so important, lists steps to check and double check your product for accessibility, and offers tips for making applications more accessible.

Bergman, Eric, and Earl Johnson. "Towards Accessible Human Interaction." In Advances in Human-Computer Interaction, edited by Jakob Nielsen, vol.5. Ablex Publishing, 1995. Available: `http://www.sun.com/tech/access/updt.HCI.advance.html`. This article discusses the relevance of accessibility to human interface designers, points out its economic and social benefits as well as the legal requirements for it. It also explores the process of designing for ranges of user capabilities with useful discussions of the concept of universal design and the inclusion of people with disabilities in the design process. It provides design guidelines for accessibility, with background and guidelines for physical disabilities such as repetitive strain injuries (RSI), low vision, blindness, and hearing impairment. It also contains an excellent list of additional sources on accessibility issues.

Schwerdtfeger, Richard S. *IBM Guidelines for Writing Accessible Applications Using 100% Pure Java*. IBM Corporation 1998. Available: `http://www.austin.ibm.com/sns/access.html`. This web site presents principles of accessibility, a checklist for software accessibility and a list of references and resources. In addition, it provides discussions of accessibility for the web and for Java applications.

Schwerdtfeger, Richard S. *Making the GUI Talk*, BYTE. 1991 Available: `ftp://ftp.software.ibm.com/sns/sr-os2/sr2doc/guitalk.txt`. This speech deals with off-screen model technology and GUI screen readers.

Sun Microsystems, Inc. “Enabling Technologies.” Available: `http://www.sun.com/access`. This web site includes a primer on the Java platform and Java accessibility and describes the support for assistive technologies now provided by Swing components of the Java Foundation classes.

Walker, Will. *The Multiplexing Look and Feel*. This article describes how the Swing classes of the JFC support a special look and feel that provides a way to extend the features of a look and feel design without having to create a new look and feel design. Walker uses the example of how an application that is designed to generate output for users with special needs can simultaneously provide and audio output and a Braille output, along with the standard visual output that ordinary Swing applications generate. Available: `http://java.sun.com/products/jfc/tsc/archive/plaf_papers_arch/multi_update`.

Design for Applets These books provide a range of information on designing applets.

Hopson, K.C., Stephen E. Ingram, and Patrick Chan. *Designing Professional Java Applets*. Sams Publishing, 1996. A reference to developing commercial-level Java applets for business, research, and education. The authors provide advanced programming information for creating applets for finance, offices, commerce, science, and research.

Gulbransen, David, Kenrick Rawlings, and John December. *Creating Web Applets With Java*. Sams Publishing, 1996. An introduction to energizing web pages with Java applets, this book addresses non-programmers with the need to incorporate pre-programmed Java applets into web pages. The CD offers more than 30 Java applets, examples of web pages utilizing these applets, and associated files.

PART I: OVERVIEW

1: THE JAVA LOOK AND FEEL

As the Java language has matured, designers and developers have recognized the need for consistent, compatible, and easy-to-use cross-platform Java applications. The Java look and feel meets that need by providing a distinctive platform-independent appearance and standard behavior for the open cross-platform environment. The use of this single look and feel in a heterogeneous networked environment reduces design and development time and lowers the cost of training and documentation for all users.

This book makes recommendations to user interface designers for the use of the Java look and feel. By following these guidelines, you can create Java applications with flexibility, visual appeal, consistency, and ease of use.

Fundamentals of the Java Look and Feel

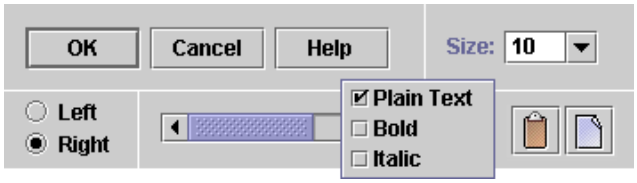
The Java look and feel can provide:

- Consistency in the appearance and behavior of common design elements
- Compatibility with industry standard components and interaction styles
- Aesthetic appeal that does not distract from application content

Three distinctive visual elements underlie the consistency, compatibility, and aesthetic appeal that are the hallmarks of the Java look and feel components: a flush three-dimensional appearance, a drag texture, and a color model.

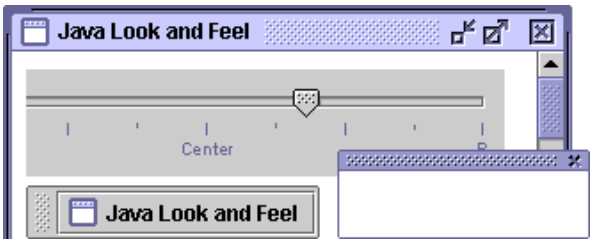
With a flush three dimensional appearance for user interface items, component surfaces appear at the same level as the surrounding canvas. A clean, modern appearance enhances aesthetic appeal in the interface. The reduction of visual noise associated with the exaggerated beveled edges allows components to fit in with a variety of applications and operating systems.

FIGURE 1 Consistent Use of 3D Flush Appearance



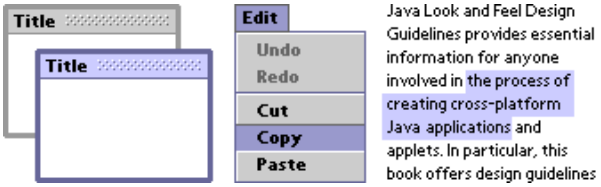
A texture pattern, used throughout the Java look and feel, indicates items that users can drag. Such an indication cues cross-platform users in a reliable way.

FIGURE 2 Consistent Use of Drag Texture



A simple and flexible color model ensures compatibility with a variety of platforms and devices capable of displaying various depth of color. The default colors provide an aesthetically pleasing and comfortable scheme for interface elements.

FIGURE 3 Role of the Color Model in Compatibility



The default appearance and behavior for JFC applications is designed for cross-platform use.

Visual Tour of the Java Look and Feel

Designed for cross-platform use and consisting of widely understood interface elements (windows, icons, menus, and pointers), the Java look and feel works in the same way on any operating system that supports the Java Foundation Classes. This visual tour shows off two example applications using the JFC with the Java look and feel. MetalEdit is a standalone text-editing application; Retirement Savings Calculator is a browser-based applet.

The following figure shows a Microsoft Windows desktop with two example Java programs. MetalEdit has a menu bar and toolbar as well as a text-editing area. Retirement Savings Calculator is running inside a web browser. Other Microsoft Windows applications are also present; some are represented by minimized windows.

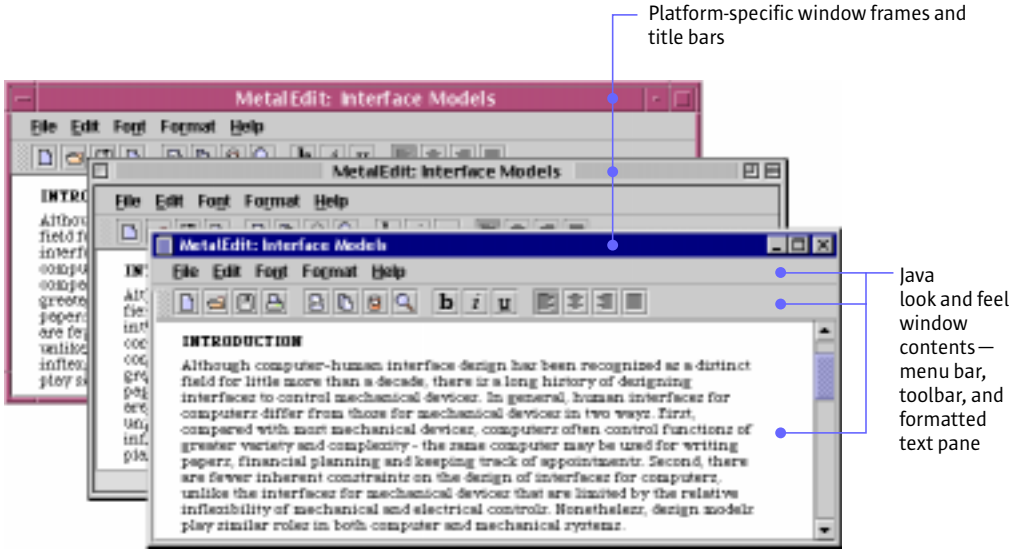
Although the windows of many applications can be open on the desktop, only one can be the active window. In the figure, MetalEdit is the active window (indicated by the color of the title bar), whereas the Netscape Navigator™ browser, which contains Retirement Savings Calculator, is inactive. As an applet, Retirement Savings Calculator is displayed within an HTML page.

FIGURE 4 Typical Desktop With Applications on the Microsoft Windows Platform

MetalEdit Application This section uses a hypothetical text-editing application called MetalEdit to give examples of some of the most important visual characteristics of the Java look and feel, including its windows, menus, toolbars, text dialog boxes, and alert boxes.

Example Windows The windows in Java look and feel applications use the borders and title bars of the platform they are running on. For instance, the MetalEdit document window shown in the preceding figure is running on a Microsoft Windows desktop and uses the standard Microsoft window frame and title bar for that platform. The following figure shows the same MetalEdit document window as it would appear on several platforms, using platform-specific window frames, title bars, and window controls. In all these cases, the window contents have been created using JFC components with the Java look and feel.

FIGURE 5 Example Windows on the CDE, Macintosh, and Microsoft Windows Platforms

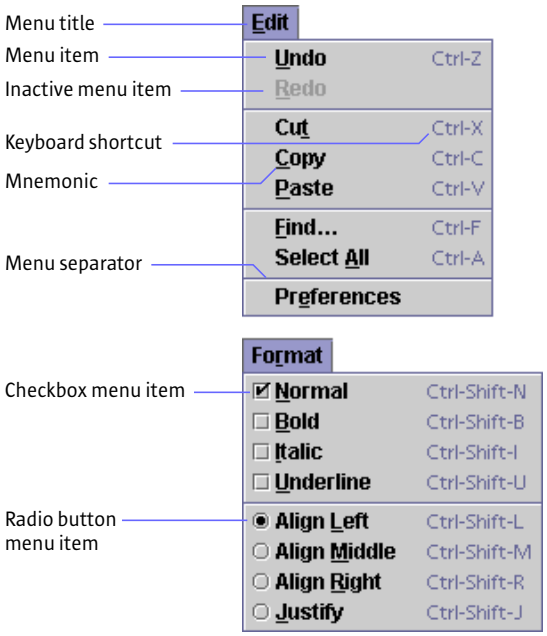


Example Menus The menu bar, which is the horizontal strip under the window title, displays the titles of application menus, called drop-down menus. Drop-down menus provide access to an application's primary functions. They also enable users to survey the features of the application by looking at the menu items. Chapter 9 contains discussions of drop-down menus, submenus, and contextual menus and provides guidelines for the creation of menus and menu items for your application.

FIGURE 6 Example of Menu Bar

The following figure shows the contents of several menus in the MetalEdit menu bar. The menu items in all three menus are divided into logical groupings by menu separators (with the flush 3D appearance). For instance, in the File menu, the Page Setup and Print commands, which affect printing of the document, are separated from the Save and Save As commands, which affect saving. For more information, see "Separators" on page 129.

FIGURE 7 Example Drop-down Menus



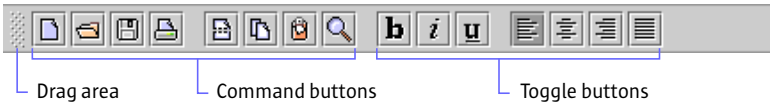
Keyboard shortcuts provide an alternative to the mouse for choosing a menu item. For instance, to create a new document, users can press Control-N. For details, see "Keyboard Shortcuts" on page 85.

Mnemonics provide yet another way to access menu items. For instance, to view the contents of the File menu, users press Alt-F. Once the File menu has keyboard focus, users can press N to create a new document. These alternatives are designated by underlining the "F" in File and the "N" in New. For details, see "Mnemonics" on page 86.

The menus shown in the preceding figure illustrate several commonly used menu titles, menu items, and menu item arrangement for Java look and feel applications. For details, see "Drop-down Menus" on page 125 and "Menu Items" on page 126.

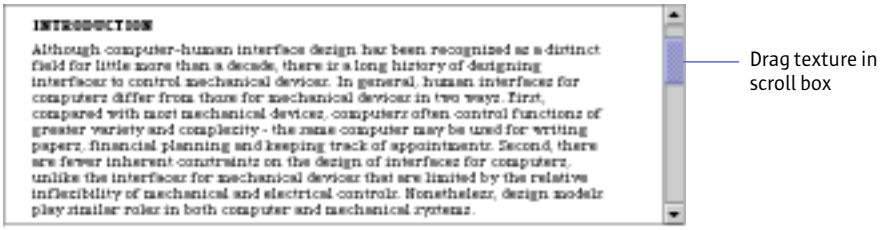
Example Toolbar A toolbar below the menu bar displays command and toggle buttons that offer immediate access to the functions of many menu items. The MetalEdit toolbar is divided into four areas for functions relating to file management, editing, font styles, and alignment. Note the use of the flush 3D appearance on the borders of the command and toggle buttons and the textured drag area to the left of the toolbar. For details, see "Toolbars" on page 134.

FIGURE 8 Example Toolbar



Example Editor Pane The document text being edited is displayed in an editor pane with a styled text editor plug-in, which is embedded in a scroll pane, (note the use of the drag texture in the scroll box). For more on styled text editor plug-in kits, see "Editor Panes" on page 166. For details on scrollbars, see "Scrollbars" on page 100.

FIGURE 9 Example Editor Pane



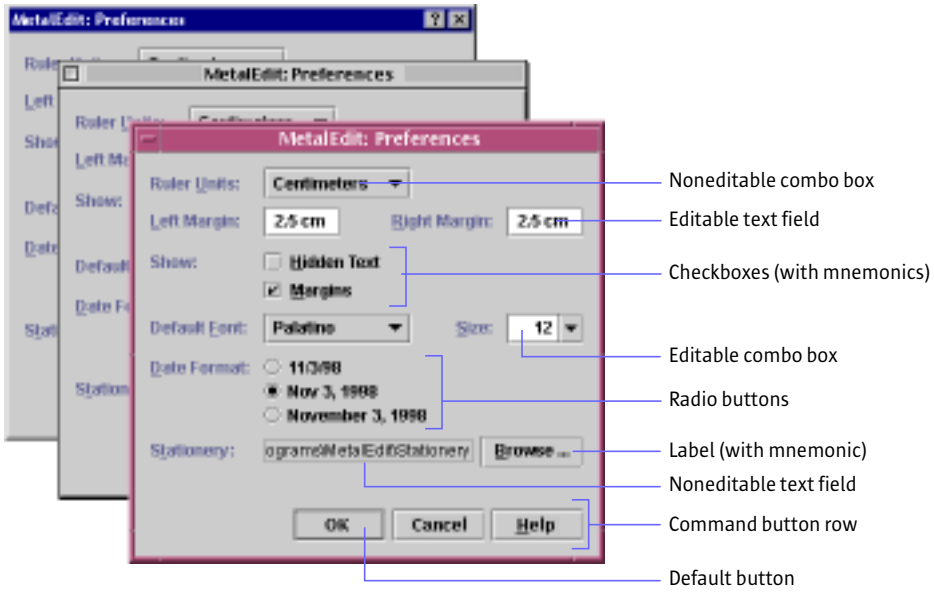
Example Dialog Box In the Java look and feel, dialog boxes use the borders and title bars of the platform they are running on. However, inside the title bar and border, the basic controls, text components, icons, and command buttons have the Java look and feel. Chapter 8 provides comprehensive details on dialog boxes in the Java look and feel and contains recommendations for their use.

The following figure shows a Preferences dialog box with the title bars and borders of several platforms. The dialog box enables users to specify preferences for the MetalEdit application. Noneditable combo boxes are used to choose ruler units and a font. Text fields are used to specify the margins. The user specifies the font size with an editable combo box. Radio buttons and checkboxes are used to set other preferences. Clicking a command button displays a file chooser in which the user can select the stationery folder.

Note the flush 3D appearance of the borders of the combo boxes, text fields, radio buttons, checkboxes, and command buttons. Note the use of the primary 1 color, one of eight colors in the default Java look and feel theme, in the labels. For a description of dialog boxes in the Java look and feel and recommendations for their use, see Chapter 8. For a thorough treatment of basic controls including combo boxes, radio buttons, checkboxes, and command buttons, see Chapter 10. For a detailed discussion of text fields and labels, see Chapter 11.

MetalEdit provides mnemonics and keyboard navigation and activation sequences for each of the interactive controls in the Preferences dialog box. The dialog box in the following figure illustrates two ways to create a mnemonic: directly in a component, indicated by an underlined letter in the component, or in a label associated with the component, indicated by an underlined letter in the label.

FIGURE 10 Example Dialog Box on CDE, Microsoft Windows, and Macintosh Platforms



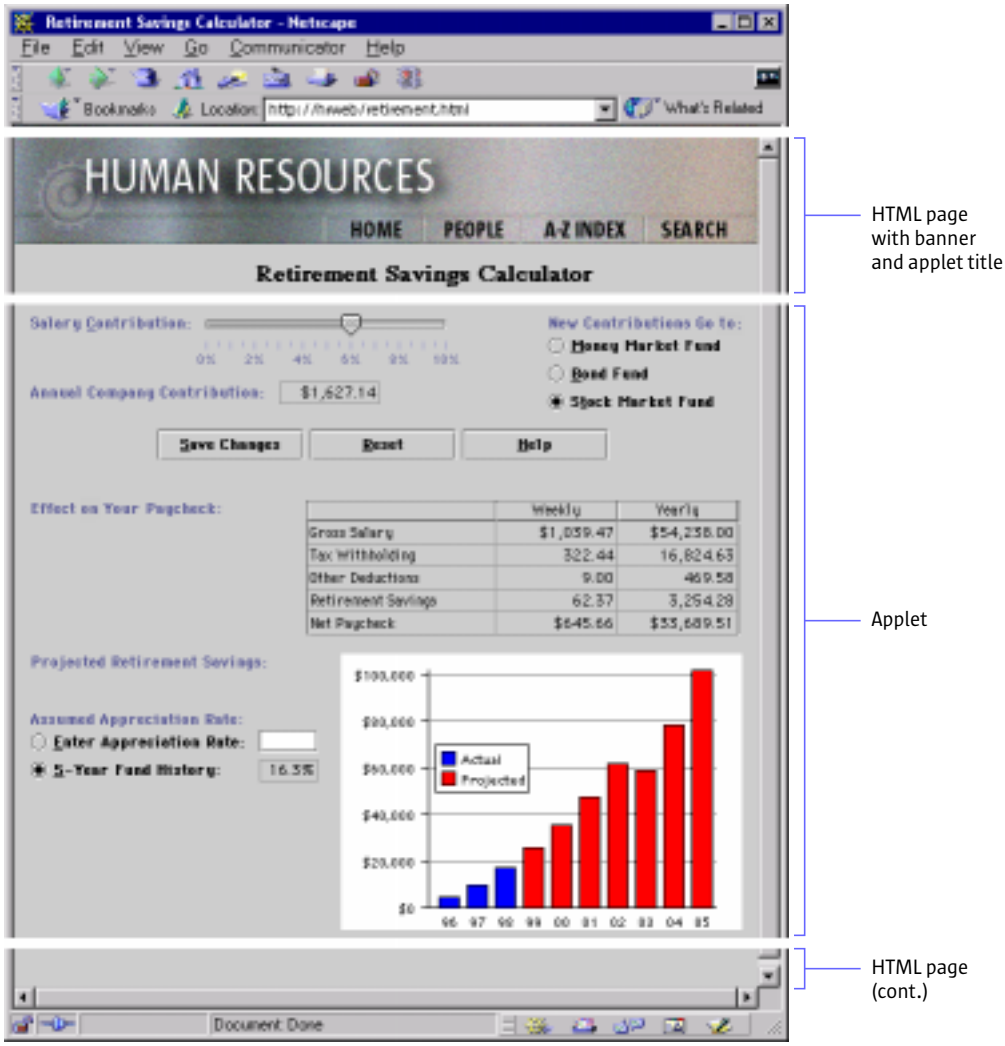
Example Alert Box The alert boxes in a Java look and feel application use the borders, title bars, and window controls of the platform they are running on. However, the supplied icons, messages, and command buttons use the Java look and feel.

When users try to close a window, the Warning alert box asks them about saving changes if changes have been made since the last save, or if the document has never been saved. Of the three command buttons in MetalEdit's Warning alert box, shown in the following figure, the default button is Save. The Don't Save button closes the window without saving changes. The Cancel button closes the dialog box but leaves the unsaved document open. Note the consistent 3D flush appearance of the command buttons. For details, see "Alert Boxes" on page 118.

FIGURE 11 Example Alert Box for MetalEdit on CDE, Microsoft Windows, and Macintosh Platforms

Retirement Savings Applet The sample applet, Retirement Savings Calculator. is part of a web page displayed in the Netscape Navigator browser. This human resources applet enables employees of a fictitious company to determine their contributions to a retirement savings plan. To make it easy for all employees to access information on their retirement savings, the company provides the applet in a web page. (Note the boundaries of the applet as it appears on the HTML page, which includes a banner in the GIF format as well as an HTML header with the title of the page.) All the JFC components shown in the sample applet use the Java look and feel.

FIGURE 12 Applet on an HTML Page



Once the applet obtains an employee's current retirement savings contribution and other salary data from a database, noneditable text fields are filled in with relevant data. A slider can be dragged to specify a contribution ranging from a minimum of 0% to a maximum of 10% and a radio button clicked to specify whether new contributions go to a money market, bond, or stock market fund. A row of command buttons gives employees a choice of whether to save changes, reset the salary contribution, or display help. Depending on the employee's input, the applet calculates and displays the company's yearly contribution to the fund, the tax withholding,

the retirement savings contribution, and the net weekly and yearly paychecks in a table. Finally, the user can type a yearly appreciation rate in an editable text field to see the accumulated future savings or instruct the applet to use the five-year fund history to project the savings in the chart at the bottom of the applet.

For more on noneditable and editable text fields, see “Text Fields” on page 162. For details on sliders, see “Sliders” on page 156. For information on radio buttons, see “Radio Buttons” on page 151. For a discussion of command buttons, see “Command Buttons” on page 142. For complete reference on tables, see “Tables” on page 171.

FIGURE 13 Retirement Savings Applet

2: JAVA FOUNDATION CLASSES

This book assumes that you are designing software based on the Java Foundation classes and utilizing the Java look and feel. This chapter provides an overview of that technology: the components of the Java Foundation Classes, and the pluggable look and feel architecture.

Java Development Kit

The **Java Development Kit** (JDK™) is the software that includes the APIs and tools that developers need to write, compile, debug, and run Java applications.

The guidelines in this book assume the usage of the Java®2 SDK, Standard Edition, or of Java Development Kit 1.1 versions 1.1.3 through 1.1.7 (referred to hereafter as JDK 1.1). The Java Foundations Classes are available for use with JDK 1.1, but are an integral part of the Java®2 Platform, Standard Edition.

Java Foundation Classes

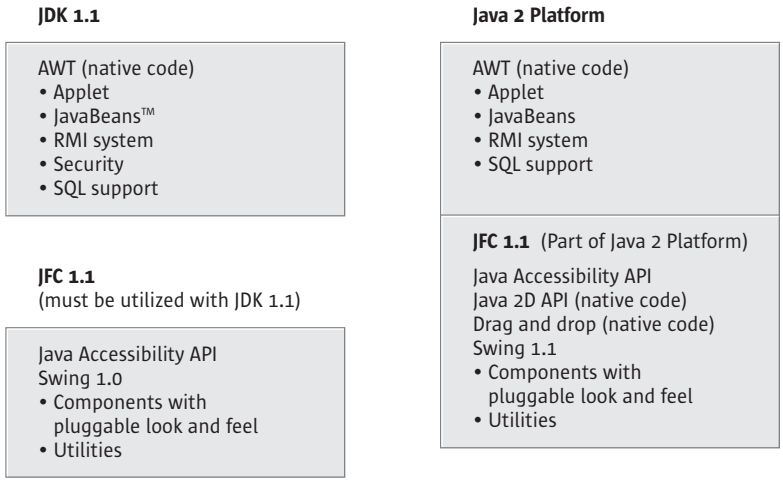
The **Java Foundation Classes** (JFC) include the **Swing classes**, which define a complete set of graphic interface components used to create JFC applications. The JFC is an extension to the original Abstract Window Toolkit. The JFC adds the Swing classes, pluggable look and feel designs, and the Java Accessibility API, which are all implemented without **native code**, that is, code that refers to the methods of a specific operating system or is compiled for a specific processor. The JFC components include windows and frames, panels and panes, dialog boxes and choosers, menus and toolbars, buttons, sliders, combo boxes, text components, tables, lists, and trees —plus an accessibility package.

All the components have look and feel designs that you can specify. The cross-platform, default look and feel is the **Java look and feel**. For details on the design principles underlying the Java look and feel and a visual tour of its features, see Chapter 1.

 The Java look and feel is synonymous to the code referred to as “Metal”.

JDK 1.1 and Java 2 The following figure summarizes the differences in the Java Foundation Classes for the two versions of the JDK. Both JDK versions contain the **Abstract Window Toolkit** (AWT), the original class library that provides the standard application programming interfaces for building graphical user interfaces for Java programs. The AWT code in both versions as well as the Java 2D API in the Java 2 Platform, Standard Edition contains native code.

FIGURE 14 Java Foundation Classes for JDK 1.1 and Java 2 Platform, Standard Edition



In Java 2, the Java Foundation Classes also include the Java 2D™ API, drag and drop, and other enhancements. The **Java 2D API** provides an advanced two-dimensional imaging model for complex shapes, text, and images. Features include enhanced font and color support and a single, comprehensive rendering model.

Font metrics are calculated differently in JDK 1.1 and in the Java 2 Platform, Standard Edition; therefore, layouts are not the same for strings and labels in the two versions of the JFC.

Due to the graphics model in Java 2D, filled areas may appear differently in JDK 1.1 and Java 2 Platform, Standard Edition.

Accessibility and JDK The **Java Accessibility API** provides ways for an **assistive technology** to interact and communicate with JFC components. A Java application that fully supports the Java Accessibility API is compatible with technologies such as screen readers and screen magnifiers that help people with disabilities use a computer and provide alternative means of use to all users. It supplies the following features:

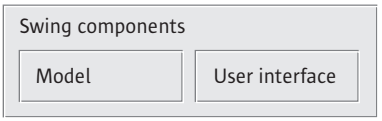
- An API that supports communication between assistive technologies and applications. A separate utility package, the Java Accessibility Utilities, provides support in locating the objects that implement the accessibility API.
- A pluggable look-and-feel architecture to build both visual and non-visual designs, such as audio and tactile user interfaces. For more on the pluggable look and feel, see "Pluggable Look and Feel Architecture" on page 20.
- Keyboard navigation that allows users to move between components, open menus, highlight text, and so on. This support makes an application accessible to users who do not use a mouse. For details on keyboard navigation, see Appendix A.

Internationalization and JDK Java 2D provides internationalized text handling. This feature includes support for the bi-directional display of text lines, important for displaying documents that contain a mix of languates with a left to right text direction (for instance, English, German, or Japanese) and languages with a right to left direction (for example, Arabic or Hebrew).

User Interface Components of the Java Foundation Classes

The Java Foundation Classes include, among other things, a complete set of user interface components based on the Swing classes. These components include windows, dialog boxes, alert boxes, and choosers, panels and panes, basic controls, and other components. Each JFC component contains a model (the data structure) and a user interface that specifies the presentation and behavior of the component, as shown in the following illustration.

FIGURE 15 Contents of the JFC Components



Such a component is sometimes referred to as a lightweight component because it is implemented entirely in the Java language and does not require the use of native data structures or classes. Because both presentation and behavior are separate and replaceable (“pluggable”), you can specify any of several look and feel designs for your application—or create your own look and feel.

The following table provides an alphabetical list of the major user interface components in the JFC, their names in the code, their English names used in this book, and the location of more detailed information on each component.

TABLE 1 Java Look and Feel of the JFC User Interface Components

JFC Component	Code Name	Common Name	For Details
	`JApplet`	Applet	page 27
	`JButton`	Command button	page 142
	`JCheckBox`	Checkbox	page 148
	`JCheckBoxMenuItem`	Checkbox menu item	page 128
	`JColorChooser`	Color chooser	page 122
	`JComboBox`	Noneditable and editable combo boxes	page 153
	`JDesktopPane`	Desktop pane	page 105
	`JDialog`	Dialog box, secondary window, and utility window	page 109, page 96, and page 97

TABLE 1 Java Look and Feel of the JFC User Interface Components *(Continued)*

JFC Component	Code Name	Common Name	For Details
	`JEditorPane`	Editor pane	page 166
	`JFrame`	Primary window	page 92
	`JInternalFrame`	Internal frame, minimized internal frame, and palette	page 105, page 106, and page 107
	`JLabel`	Label	page 159
	`JList`	List	page 165
	`JMenu`	Drop-down menu and submenu	page 125 and page 125
	`JMenuBar`	Menu bar	page 124
	`JMenuItem`	Menu item	page 126
	`JOptionPane`	Alert box	page 118
	`JPanel`	Panel	page 98
	`JPasswordField`	Password field	page 163

TABLE 1 Java Look and Feel of the JFC User Interface Components *(Continued)*

JFC Component	Code Name	Common Name	For Details
	`JPopupMenu`	Contextual menu	page 133
	`JProgressBar`	Progress bar	page 157
	`JRadioButton`	Radio button	page 151
	`JRadioButtonMenuItem`	Radio button menu item	page 128
	`JScrollBar`	Scrollbar	page 100
	`JScrollPane`	Scroll pane	page 99
	`JSeparator`	Separator	page 129
	`JSlider`	Slider	page 156
	`JSplitPane`	Split pane	page 103
	`JTabbedPane`	Tabbed pane	page 101
	`JTable`	Table	page 171

TABLE 1 Java Look and Feel of the JFC User Interface Components *(Continued)*

JFC Component	Code Name	Common Name	For Details
	`JTextArea`	Plain text area	page 165
	`JTextField`	Editable and noneditable text field (single line)	page 162
	`JTextPane`	Formatted text pane	page 166
	`JToggleButton`	Toggle button	page 147
	`JToolBar`	Toolbar	page 134
	`JToolTip`	Tool tip	page 138
	`JTree`	Tree view	page 182
	`JWindow`	Plain (undadorned) window	page 96

In JFC, the typical primary windows that users work with are based on the `JFrame` component. Unadorned windows that consist of a rectangular region without any title bar, close box, or controls are based on the `JWindow` component. The `JWindow` component is not typically used by designers and developers except to create windows without title bars, such as splash screens.

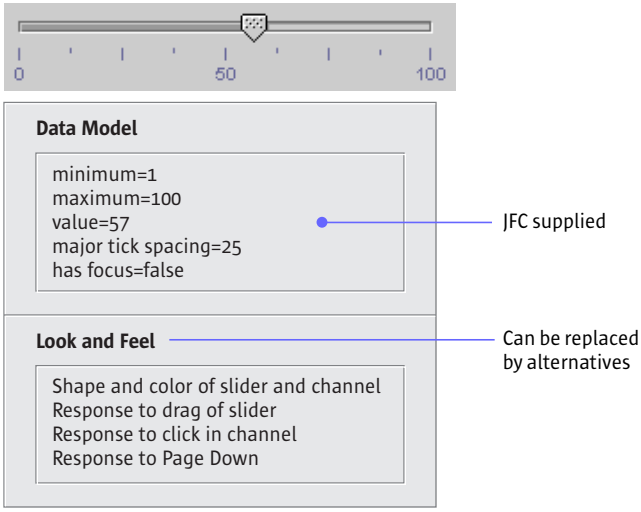
For details on the use of windows, frames, palettes, and panes, see Chapter 7.

Pluggable Look and Feel Architecture

The separation of a component's model (data structure) from its user interface (display and interaction behavior) is the empowering principle behind the **pluggable look and feel architecture** of the JFC. One JFC application can present a Java look and feel style, another style of interface, or a customized interface (for example, an audio look and feel).

Consider a simplified slider as an example. The slider's model contains information about the slider's current value. The slider's user interface determines how users see or interact with the slider. The model knows almost nothing about the user interface—while the user interface knows a great deal about the model, as shown in the following figure.

FIGURE 16 Pluggable Look and Feel Architecture



Client Properties

You can use the client properties mechanism to display an alternate form of a specific JFC user interface component. If a property is not supported by a specific look and feel design, it will be ignored and the components displayed as usual. You can set alternate appearances for sliders, toolbars, trees, and internal frames. For instance, a nonfilling slider is displayed by default. However, by using the client properties mechanism, you can display a filling slider.

Look and Feel Options

You, the designer, have the first choice of a look and feel. You can determine the look and feel you want users to receive on a specific platform or you can choose a cross-platform look and feel. With a cross-platform look and feel, your application will look and perform the same everywhere, simplifying the application's development and documentation.

 Specify the Java look and feel, which is a cross-platform look and feel, explicitly. In addition, f you do not specify a look and feel or if an error occurs while specifying the name of a look and feel, the Java look and feel is used by default.

 The following code can be used to specify the Java look and feel explicitly:

```
UIManager.setLookAndFeel(
UIManager.getCrossPlatformLookAndFeelClassName() );
```

Alternatively, you can specify:

- A particular look and feel—one that ships with JFC or one someone else has made (for instance, Microsoft Windows). Note, however, that not all look and feel designs are available on every platform. For example, Microsoft Windows is only available on the Microsoft Windows platform.
- A changeable or adaptable look and feel — one or more auxiliary look and feel designs to be used in addition to the primary look and feel (for instance, a multiplexing look and feel). By combining look and feel designs, you can target different ways of perceiving information.

Because there is far more to the design of an application than the mere substitution of components, it is unwise to give end users the ability to swap look and feel designs while working in your application. Switching look and feel designs in this way only swaps the look and feel designs of the components from one platform to another. The layout and vocabulary used are platform specific and won't be changed. For instance, swapping a look and feel does not change the titles of the menus.

 Make it possible for your users to specify an auxiliary look and feel design, which provides alternative methods of information input and output for people with special needs.

The look and feel designs available in JDK 1.1 and in the Java 2 Platform, Standard Edition are:

- **Java look and feel** The Java look and feel is designed for use on any platform that supports the JFC. This book provides recommendations on the use of the Java look and feel.
- **Microsoft Windows** (called Windows in the code) The Windows style look and feel can be used only on Microsoft Windows platforms. It follows the behavior of the components in applications that ship with Windows NT 4.0. For details, see *Windows Interface Guidelines for Software Design*.
- **CDE** (called CDE/Motif in the code) The CDE style look and feel is designed for use on UNIX® platforms. It emulates OSF/Motif 1.2.5, which ships with the Solaris™ 2.6 operating system. It can run on any platform. For details, see the *CDE 2.1/Motif 2.1 Style Guide and Glossary*.
- **Macintosh** (called Mac OS in the code) The Macintosh style look and feel can be used only on Macintosh operating systems. It follows the specification for components under Mac OS 8.1. For details, please see the *Mac OS 8 Human Interface Guidelines*.

PART II: FUNDAMENTAL JAVA APPLICATION DESIGN

3: DESIGN CONSIDERATIONS

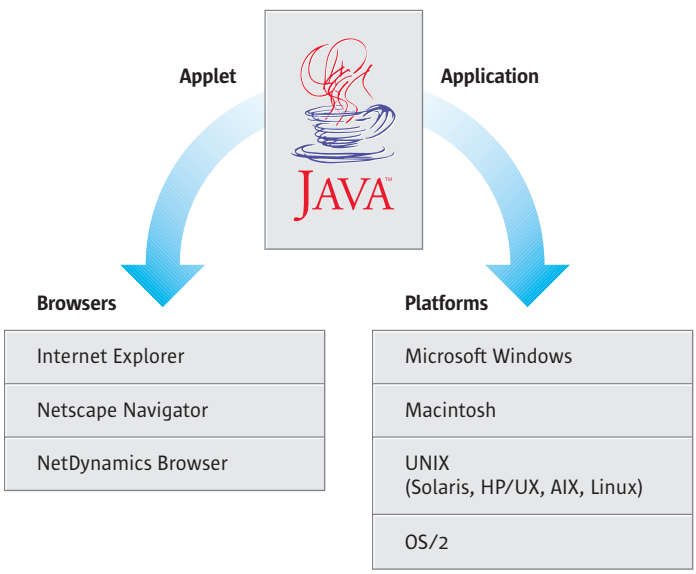
This chapter describes three fundamental issues to keep in mind when you design your Java application.

- **Applications and applets.** Creating a standalone application versus creating an applet, which is launched from within a web browser
- **Accessibility.** Supporting all users who want to interact with your application, including users with special needs
- **Internationalization and localization.** Designing menus, dialog boxes, error messages, and graphics that are meaningful worldwide

Applications and Applets

At the beginning of the development process, you must decide if you want to create a standalone application or an applet, which is launched from within a web browser.

FIGURE 17 Differences Between Applications and Applets



Many differences exist between applications and applets, but the two with the biggest impact are distribution and level of access, as described in the following sections. An issue unique to applets, its placement, ...

For an examples of an application and applet that use the Java look and feel, see "MetalEdit Application" on page 5 and "Retirement Savings Applet" on page 10. For a list of additional reading on applets, see "Design for Applets" on page xxvi.

Distribution A standalone application is usually distributed on a CD-ROM disc or floppy disk, or over a network. Users can install the application on their local hard disk or access it from a local area network (LAN). Starting an application from a local hard disk is quicker than downloading one over a LAN. However, distribution and maintenance of local applications are complicated because separate copies of the application exist on each user's local computer.

Distribution and maintenance of applets are simpler. An applet is installed on a central web server. Using their local machines and a web browser, users access the latest version of the applet from anywhere on the intranet or Internet. The disadvantage of this approach is that users must download the applet over the network each time they start it.

<if the subject here is distribution, it should be corrected. The discussion regarding server-based applets also applies to apps /usr/dist. The real advantage is applets can go through firewalls.>

If the JFC is not stored on the users' local machine, then users will have to download the JFC every time they run the applet. Ensure that your users have a browser that contains the JFC or that they are using Java™ Plug-In.

<this section is really discussing latency or what price the user pays in terms of time if JFC is not currently loaded onto the system's hard drive and actually running. Also the phrase "users must download" is very scary--it makes one think of running to an ftp site eery time they want to use the applet.>

Level of Access A standalone application, such as a word processor, can read and write files on the user's local hard disk. Because applets are designed for display in a web page, which might come from an unknown source, they usually cannot access the user's hard disk. Applets are well suited for tasks like updating a central database, whereas applications are well suited for reading and writing to a local disk.

<point reader to a security reference rather than the Level of Acess section>

<explore this more>

<can we use a term other than access so it is not confused with accessibility?>

Placement of Applets You can launch an applet from one of two places: from within the main browser window, as shown in the "Retirement Savings Applet" on page 10; or, you can open a new browser window and launch your applet from that window.

Browser Window The main browser window is well suited for launching applets in which users users perform a single task. This enables users to perform the task and then go back to performing other activities in the browser, such as web surfing. <ask Don for help...>The applet terminates when the user navigates to another web page. The current settings and data in the applet are lost. To use the applet again, users must navigate back to the page that contains the applet and reload the page.

An applet in the main browser window generally does not include a menu bar—having a menu bar in both the applet and the browser confuses the user. The mnemonics assigned in the applet must also be different than the mnemonics used to control the main browser window; otherwise, the mnemonics might conflict.

Separate Applet Window A separate window is well suited for launching an applet that ties up a browser window for an extended period of time or that has complex controls, such as a menu bar. This enables users to both interact with the applet and maintain the main browser window for other activities, such as surfing the web. Designing an applet for a separate browser window is simpler if you remove the browser's normal navigation controls. You will avoid confustion between the browser's controls and the applets controls and also conflicts with mnemonics.

Accessibility

Accessibility implies the removal of barriers that prevent people with disabilities from participating in social, professional, and practical life activities. In software design, **accessibility** requires taking into account the needs of people with functional differences: for example, users who are unable to operate a conventional mouse or users who cannot process information by using traditional sensory modalities.

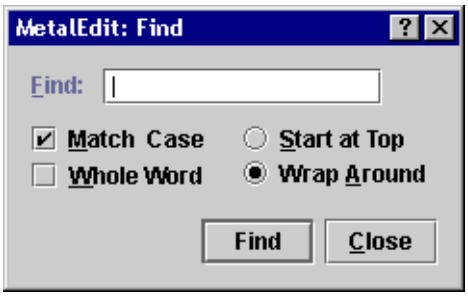
Benefits of Accessibility Providing computer access to users with disabilities offers social, economic, and legal benefits. Accessible software enables users with disabilities to be more productive. Users who are mobility impaired, for

example, can accomplish tasks using the keyboard instead of the mouse. Users with visual disabilities can receive auditory feedback for the text they type.

Accessibility makes your application easier to use not only for people with disabilities but for everyone who uses it. For example, mnemonics, which enable users to activate commands from the keyboard, aid users with physical disabilities as well as blind and low vision users. Mnemonics also provide faster access to commands for all users.

In the following dialog box, which includes mnemonics, users press the Alt-F keys to search for a specified text string.

FIGURE 18 Mnemonics in a Dialog Box



Accessible software is a legal requirement in many countries. For example, in the United States, the Americans With Disabilities Act requires employers to make reasonable accommodations for disabled workers. Internationally, ISO 9421, a usability and accessibilty standard, is being developed. In addition, disability advocacy groups are succesfully putting pressure on companies to support accessibility in the software they seel..

Accessible Designs

There are three major steps you can take to create an application that meets the needs of users with functional limitations. The first step is to follow the standards in this book. You'll find recommendations for supporting accessible software, such as when to employ mnemonics, how to use color and animation, and how to assign keyboard focus.

The second step you can take is to test the application with users with different disabilities. Low vision users, for example, are sensitive to font sizes and color, as well as layout and context problems. Blind users are affected by poor interface flow, tab order, layout, and terminology. Users who are functionally-impaired can be sensitive to tasks that require an excessive number of steps or a wide range of movement.

A third step you can take is to evaluate your software using the Explore utility tool, which is part of the Java Accessibility Utilities package. Explore identifies missing API information on a component, such as a missing `AccessibleName` property.

Assistive technologies use a component's `AccessibleName` property to distinguish it from another component of the same type. The `AccessibleDescription` property provides additional information about a component, such as how it works.

For more information on the Java Accessibility API, see "Accessibility and JDK" on page 15. For a list of books on accessibility, see "Design for Accessibility" on page xxv.

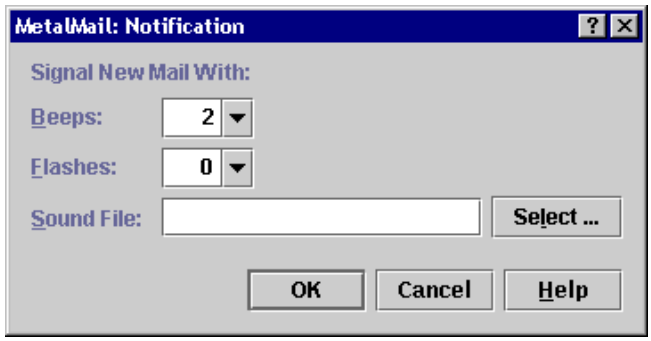
Internationalization and Localization

Internationalization is the process of writing an application that is suitable for the global marketplace, taking into account variations in regions, languages, and cultures. Localization is the process of customizing an application for a particular locale. Following is a list of data that is commonly changed during localization.

- Colors
- Currency formats
- Date and time formats
- Graphics
- Icons
- Labels
- Messages
- Number formats
- Online help
- Page layouts
- Personal titles
- Phone numbers
- Postal addresses
- Sounds
- Units of measurement

The following figure shows a notification dialog box in both English and Japanese. Much of the localization of this dialog box involves the translation of text. Note that the Japanese dialog box is wider than the English dialog box. Also note how mnemonics appear in the Japanese dialog box. Mnemonics appear similarly in non-Roman languages.

FIGURE 19 English and Japanese Color Choosers



The following sections describe the main areas you need to think of in terms of internationalization. More internationalization guidelines appear throughout the book and are identified by the symbol attached to this paragraph. For a list of additional reading on internationalization, see "Design for Internationalization" on page xxiv.

Layout Managers

A layout manager controls the size and position of the components in an application. A layout manager specifies the layout in relative terms so that when a window or dialog box is resized, its components retain the same distance from the edge and from each other. For example, the spacing between label and its text field remains the same when the dialog box is made bigger or smaller. The text field may become bigger or smaller.

A layout manager deals with text strings of different lengths in different locales. A button labeled Cancel in English becomes larger in German, where its label is Abbrechen, as shown in the following figure.

component orientation

For more information on layout managers, see ....

NOTE – The spacing guidelines that appear in the Visual Design chapter and throughout the component chapters of this book are not necessarily transferable when you work to localize your application using the layout managers. <Then give a cross-reference to the layout manager articles. >

Resource Bundles A resource bundle is a place outside your application source code where you can store data specific to a locale, such as text and graphics. Resource bundles make your application easier to localize because they provide locale-specific data without changing the source code.

For more information on creating resource bundles, see *The Java Tutorial Continued*. You can view this information online at `http://java.sun.com/docs/books/tutorial/i18n/resbundle/index.html`.

Formatting Classes Formats for numbers, currencies, dates, and times vary among countries. For example, in English, a date appears in the form of July 26, 1987, and the time appears as 3:17 p.m. In German, the same date and time are written as 26 Juli 1987 and 15:17 Uhr. You can use the formatting classes to automatically format numbers, currencies, dates, and times for your locale.

For numbers and currencies, the class is `NumberFormat`; for dates and times, the class is `DateFormat`; and for strings that contain variable data, the class is `MessageFormat`.

Text Text size may change when an application is localized. The text may become bigger or smaller, as shown in the following figure of a Cancel button in English, German, and Japanese. To aid in the translation of text, you can use a layout manager and place the text in a resource bundle, as described in the previous sections.

FIGURE 20 Cancel Button in English, German, and Japanese

Graphics Like text, you can place graphics in resource bundles so that, if necessary, your localizers can translate them without changing the application code. You can also do several things when designing a graphic: use graphics that are understood in the global market place; avoid using text in graphics, and do not use graphics whose meaning may be either specific to a locale or offensive to a locale.

Word Order The order of the words in sentences may change in translation. Depending on the locale, units (such as currencies) may come before or after the number. In addition, adjectives come before nouns in some languages and after nouns in other languages.

The following figures demonstrates how you can use labels in place of sentences in your application. Both figures use a noneditable combo box. In the figure on the left, the noneditable combo box appears in the middle of a sentence. This structure reads fine in English, but not in French, where the adjective should come after the noun. The figure on the right, which uses a label before the component, reads fine in both English and French.

FIGURE 21 Word Order in Dialog Boxes

Sort Order The sorting order of text strings may also change in translation. For example, a list item may not always appear in the same location in the list. Also, the rows in a table may be sorted differently in different locales. For example, in an email application, where you can sort message headers by date ..... You must order lists according the to sorting conventions of the locale.

The `Collator` class sorts strings in a manner appropriate for different locales.

Keyboards Keyboard layouts also differ among countries. Therefore, do not use characters that don't appear on all of your users keyboards. Also, when designing an onscreen keyboard, you must make sure the shapes, locations, and number of the keys can be localized.

Fonts Make sure that all fonts (name, size, and style) are not hardcoded in the application and can be changed by the localizers. Styles such as underline and bold differ among writing systems. If you define a corporate font for the English version of the application, also define a corporate font for the localized versions.

Usability Testing In the end, your application will be easier to localize if you send draft designs to translators early in the design process. Perform a trial translation in at least one foreign language.

Test the application with users from different locales. Determine whether the users understand how to use the product, if they understand the icons and colors appropriately, and if there is anything offensive in the product.

4: VISUAL DESIGN

Visual design and aesthetics affect user confidence, comfort, and ease. A polished and professional look without excess or oversimplification is not simple to attain. This chapter discusses the role of the theme mechanism in supporting visual elements, the use of text in your interface, guidelines for the layout and alignment of interface elements, and the use of animation in your interface.

Themes

You can use a theme mechanism to control many of the fundamental attributes of a look and feel design, including colors and fonts. With the themes mechanism, which is built into the Java look and feel, you can make changes to the colors and fonts in your application. You might want to change the colors to match your corporate identity or to increase color contrast and font size to enable users with sight limitations to use your application.

 Themes can be an easy way to make changes to the values in the defaults table. See the technical documentation on the class `javax.swing.plaf.metal.DefaultMetalTheme` at the Swing Connection web site at `www.java.sun.com`.

Colors

If you want to change the color theme of your application, it is important to understand the color model used in the Java look and feel so that your interface elements remain visually coherent. The Java look and feel uses a simple color model, so that it can run on a variety of platforms and devices capable of displaying various depths of color. Eight colors are defined for the interface:

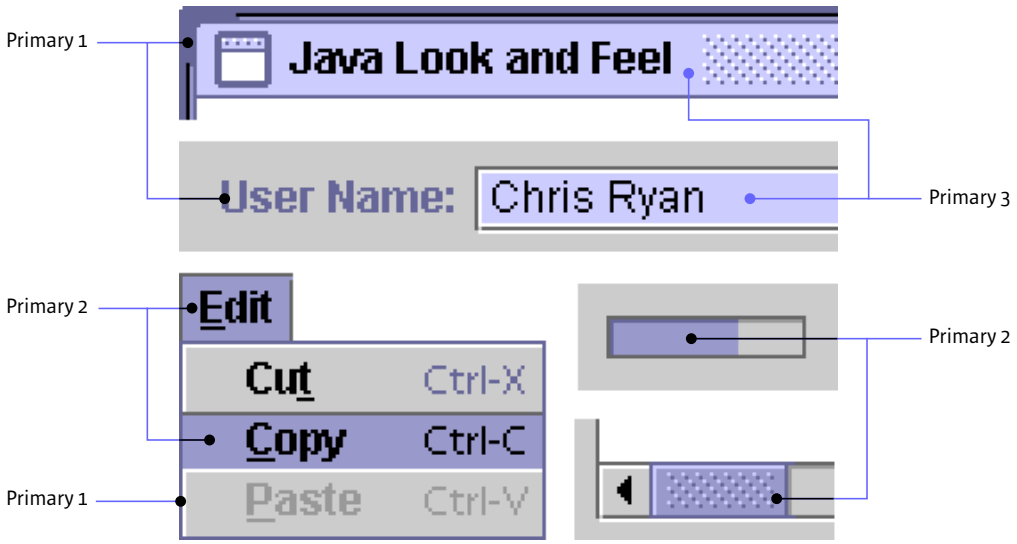
- Three primary colors to give the theme a color and callout active and selected items.
- The three secondary colors, typically shades of gray, are for neutral drawing and inactive and disabled items.
- Two additional colors, usually defined as black and white, display text and highlights.

The visual elements of Java look and feel applications use the primary colors as shown in the following figure:

- Primary 1 for active window borders, shadows of selected items, and labels.
- Primary 2 for highlighting menu titles and items and in the fill for progress bars and scroll boxes.
- Primary 3 for large colored areas such as the title bar of internal frames and the background of a selected text field.

Black is used for user text such as the entry in the editable text field, and control text such as the menu titles and menu items. White is used as a shadow for highlighting in menu, as shown in the following figure.

FIGURE 22 Primary Colors in Default Color Theme <<callouts to be added>>

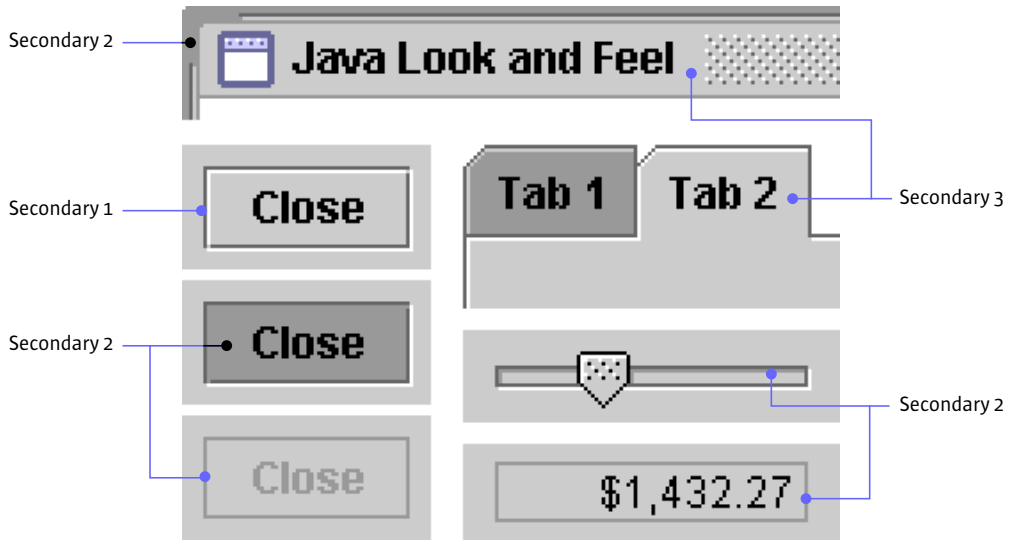


The visual elements of Java look and feel applications use the secondary colors as shown in the following figure:

- Secondary 1 for the dark border for flush 3D effects such as command buttons
- Secondary 2 for inactive window borders, basic gray shadows, pressed buttons, and dimmed command button text
- Secondary 3 for the background canvas and inactive title bars for internal frames

Black is used for the title text in the internal frame as well as the button text in command buttons, tabs in tabbed panes, and text in noneditable text fields. White is used as a highlight for the 3D flush appearance of such components as command buttons.

FIGURE 23 Secondary Colors of Default Color Theme <<callouts to be added>>



This table summarizes the eight colors defined in the Java Look and Feel, along with swatches and values representing the default theme; and the roles each color plays in basic drawing, visual 3D effects, and text.

TABLE 2 Colors of the Default Java Look and Feel Theme

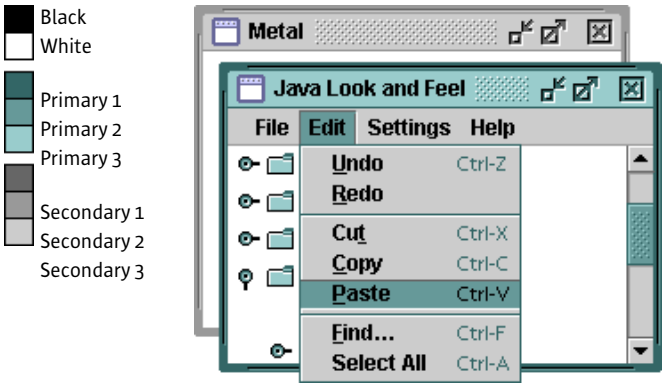
Name	Basic Drawing	3D Effects	Text
Black RGB 000-000-000 Hex #000000			User text and control text (including items such as menu titles)
White RGB 255-255-255 Hex #FFFFFF		Highlights	Background for user text entry area
Primary 1 RGB 102-102-153 HEX #666699	Active window borders	Shadows of selected items	System text (for example, labels)
Primary 2 RGB 153-153-204 Hex #9999CC	Highlighting and selection (for example, of menu titles, menu items, and slider fill); indication of keyboard focus	Shadows (color)	
Primary 3 RGB 204-204-255 HEX #CCCCFF	Large colored areas (for example, the active title bar)		Text selection
Secondary 1 RGB 102-102-102 HEX #666666		Dark border for flush 3D effect	
Secondary 2 GB 153-153-153 HEX #999999	Inactive window borders	Shadows (basic or gray); button mousedown	Dimmed text (for example, inactive menu items or labels)
Secondary 3 RGB 204-204-204 HEX #CCCCCC	Canvas color (that is, normal background color); inactive title bar		

Once you understand the functions of each color, you can define color themes that make sense. Note that primary 1 is darker than primary 2, which in turn is darker than primary 3. The same rule applies to the secondary group.

Unless you are defining a reverse-video type of theme, maintain a dark-to-light gradation like the one in the default Java look and feel theme so that interface objects are properly rendered. For instance, interface elements such as the background color occupy more space and are lighter than elements such as shadows, which, in turn take up less space and are lighter than borders.

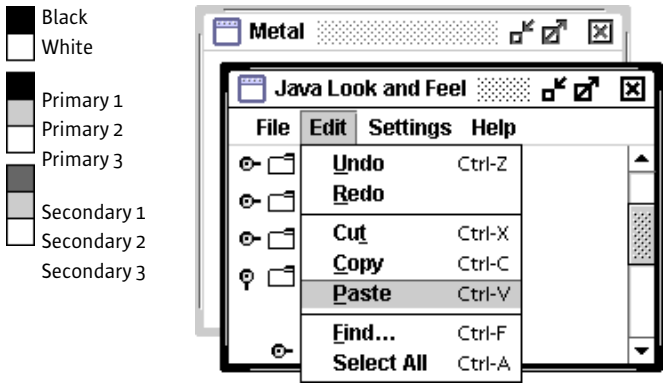
The simplest modification you can make to the color theme is to redefine the primary colors—for instance, substituting greens for the purple-blue used in the default theme, as shown in the following figure.

FIGURE 24 Green Color Theme <<callouts to be added>>



You can use the same value for more than one of the eight colors—for instance, a high-contrast theme might use mostly black, white, and grays. The following figure shows a theme that uses the same grays for primary 2 and secondary 2. It uses white for primary 3 and secondary 3 as well as in its normal role.

FIGURE 25 High Contrast Color Theme <<callouts to be added>>



Fonts

As part of the theme mechanism and parallel to the color model, the Java look and feel provides a default font style model for a consistent look. You can also use themes to redefine font typefaces, sizes, and styles in your application. he default Java look and feel theme defines four typestyles: the control font, the system font, the user font, and the small font. Note that fonts vary across platforms.

The following table shows the mappings to Java look and feel components for the default theme.

TABLE 3 Java Look and Feel-Defined Type Styles and Their Mappings

Type Style Name	Default Style	Uses
Control	12-point bold	Buttons, checkboxes, menu titles, and window titles
Small	10-point plain	Keyboard shortcuts in menus and tool tips
System	12-point plain	Tree views and tool tips
User	12-point plain	Text fields and tables

In the default color theme, six methods can be used to return references to the four type styles. The `getControlTextFont()`, `getMenuTextFont()`, and `getWindowTitleFont()` methods return the control font; the `getSystemTextFont()` method returns the system font; the `getUserTextFont()` method returns the user font; and the `getSubTextFont()` method returns the small font.

All fonts in the Java look and feel are defined in the default Java look and feel theme as dialog, which maps to a platform-specific font.

To ensure consistency, ease of use, and visual appeal, use the default fonts provided unless there is compelling reason for an application-wide change, such as higher readability. In this case you could use the theme mechanism.

Make sure that the layout of your application can handle different font sizes.

Ensure that the font typeface settings you choose are legible and can be rendered well on your target systems.

Capitalization of Text in the Interface

Even though the Java look and feel uses graphics as the principal way that users interact with the computer, many interface elements contain some text. It is important to be concise and consistent with language. This section describes the standards for the capitalization of text in the Java look and feel.

All text that appears in the interface elements of your application should follow one of two capitalization conventions: headline capitalization or sentence capitalization. Use headline capitalization for most names, titles, labels, and short text. Use sentence capitalization for lengthy text messages.

Use the standard typographical conventions for sentences and headlines. Let translators determine the standards in your target locales.

Place all text in resource bundles. <<Need a cross-reference here to a technical document about these. Gail, have you talked to Brian? Sent email to Brian on March 18.>>

Because you may encounter situations in your interface when capitalization is not appropriate, such as the window titles for documents users have named without using capitalization, do not capitalize words automatically.

Headline Capitalization in English

Most items in your application interface should use headline capitalization (sometimes called “caps/lowercase”), which is the style traditionally used for book titles. Capitalize every word except articles (“a,” “an,” and “the”), coordinating conjunctions (for example, “and,” “or,” “but,” “so,” “yet,” and “nor”), and prepositions with fewer than four letters (like “in”). The first and last words are always capitalized, regardless of what they are.

Use headline capitalization for the following interface elements. Note the examples of use in parentheses:

- Checkbox text (Automatic Backup, Automatic Save Every Five Minutes)
- Combo box names and text (Centimeters, Inches)
- Command button text (Save, Don't Save)
- Icon names (with the exception of situations when they stand for a name users can change)
- Labels for groups of buttons or controls (Default Font)
- Menu items (Page Setup)
- Menu titles (File, Edit, View)
- Radio button text (Assumed Appreciation Rate)
- Slider text (Left, Center, Right)
- Tab names (Swatches, RGB Color)
- Text field labels (Enter Appreciation Rate)
- Titles of windows, panes, and dialog boxes (for instance, Color Chooser; with the exception of window titles reflecting file names not capitalized by users)
- Tool tips (Use headline capitalization for command names like Cut Selection, but use sentence capitalization for longer phrases like "Show security information for this page")

Sentence Capitalization in English When text is in the form of full sentences, capitalize only the first word of each sentence or phrase unless the text contains proper nouns, proper adjectives, or acronyms that are always capitalized. Avoid the use of long sentences that are not full sentences

Use sentence capitalization in the following interface elements. Examples are in parentheses.

- Dialog box text (The document you are closing has unsaved changes.)
- Error, help, or messages (The printer is out of paper.)

Layout and Visual Alignment

It is essential to give careful consideration to the layout of components in your windows and dialog boxes. A clear and consistent layout streamlines the way users move through an application and helps them utilize its features efficiently. The best designs are aesthetically pleasing and easy to understand. They orient components in the direction in which people read them, and they group together logically related components.

Throughout this book, the spacing illustrations for all user interface elements use pixels as the unit of measurement with a screen at 72 to 100 dpi.

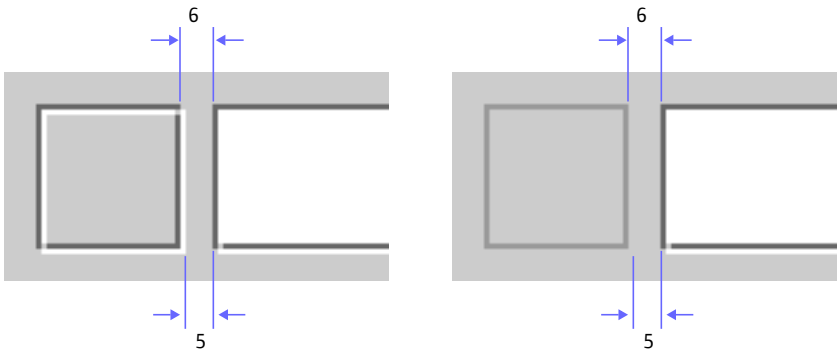
When you lay out your components, remember that users may use the mouse, keyboard, or other assistive technology to navigate them; therefore use a logical order (for instance, place the most important elements in the upper left corner of a dialog box).

Between-Component Spacing Guidelines

Increments of six pixels should ideally be used for spacing between components. Because the dark portion of the flush 3D border is visually more significant than the lighter border, the actual measurement between elements is reduced by one pixel.

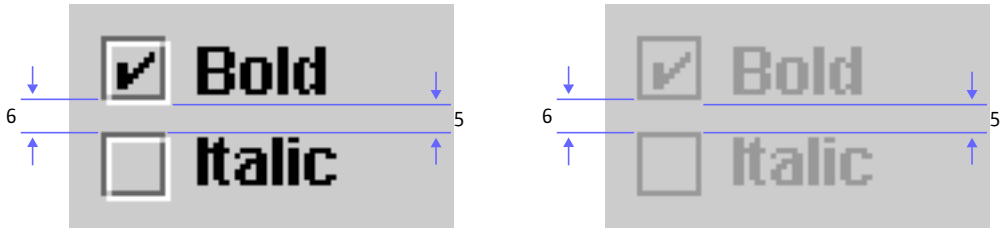
The following illustration shows how a perceived six pixel horizontal space is actually five pixels between components.

FIGURE 26 Horizontal Spacing of Components



In the following figure, a perceived six pixel vertical space is actually five pixels between checkbox components.

FIGURE 27 Vertical Spacing of Components



The following figure shows how this spacing is preserved for disabled items.

FIGURE 28 Horizontal Spacing of Disabled Components

Patria,
I moved this illustration in with Fig.25
–gary

The dimensions of disabled components do not change, although the white border is not drawn.

FIGURE 29 Vertical Spacing of Disabled Components

Patria,
I moved this illustration in with Fig.26
–gary

Insert five pixels (six minus one) between closely-related items such as checkboxes in a set. Insert eleven pixels (twelve minus one) for greater separation between sets of components (such as between a set of radio buttons and a set of checkboxes). Insert twelve pixels between items without the white flush 3D highlight border (for instance, text labels, bordered pane borders, and padding at the top and left edges of a pane).

For guidelines on the spacing of individual JFC components with the Java look and feel, see "Toolbar Button Spacing" on page 136, "Command Button Spacing" on page 145, "Radio Button Spacing" on page 152, "Checkbox Spacing" on page 150, and "Spacing of Noneditable, Editable, and Passwords Fields" on page 164.

Bidirectional Layout and Spacing the length of text strings varies in different languages, so discuss the behavior you want to occur during resize operations with the developer.

Be specific about layout, spacing, and ordering. Use the layout managers appropriately to accommodate these differences.

FIGURE 30 Bidirectional Layout and Spacing <<Ask Brian Beck for help with this>>

Design Grids The most effective method of laying out user interface elements in the Java look and feel is to use a design grid with blank space to set apart logically related sets of components.

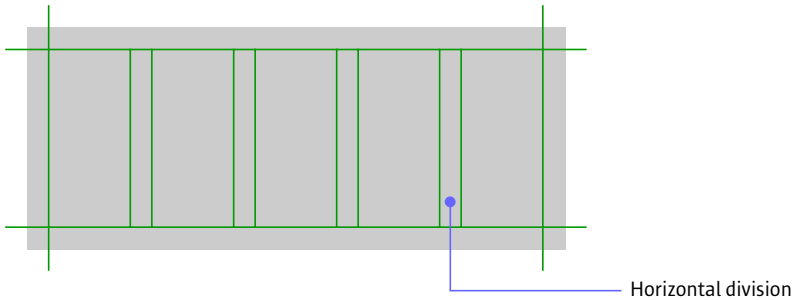
A grid divides the available space into areas that can help you to arrange and align the components in a dialog box or other layout.

The following illustration shows a sample grid that provides a standard 12-pixel margin and divides the remaining space into four columns, each separated by 12 pixels.

Horizontal divisions aid in scanning and interpreting the components and sets of related options.

Use the appropriate layout manager to leave adequate horizontal space for the variable width of internationalized text strings.

FIGURE 31 Sample Grid With Horizontal Divisions



Vertical divisions are more difficult to set initially because they depend on the depth of components and sets of components.

You can use any reasonable number of rows and columns.

Use the grid as a guide to produce a pleasing layout that makes it easy for the user to scan and understand the relationships between sets of components and the logical ordering of tasks.

Define a grid that works for a number of layouts to provide a more consistent appearance in your interface.

Developing a grid is an ongoing process. If you know how much space is available, you can start working with the components to determine the most effective use of space. A grid can also help you to determine how much space to allocate to a given set of components.

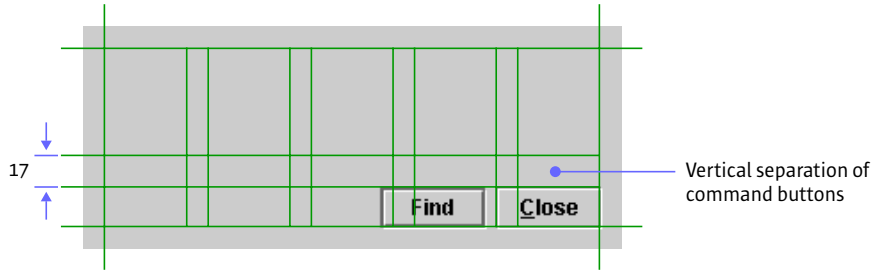
For spacing between rows and columns, use multiples of 6 pixels minus one, to allow for the flush 3D border (see "Between-Component Spacing Guidelines" on page 43).

Design grids are not to be confused with the AWT Grid Layout Manager.

Laying out a simple dialog provides an example of how to use space in an interface. The following illustrations show steps in the process of using a grid to lay out a simple Find dialog box.

First, determine the functional requirements of the dialog box or other interface element. Then add the components that require Java look and feel placement and spacing standards. For instance, command buttons in dialog boxes should be right-aligned at the bottom, and be separated vertically from the rest of the components by 17 pixels. (Note that command button rows in alert boxes are aligned automatically with the leading edge of the message text.)

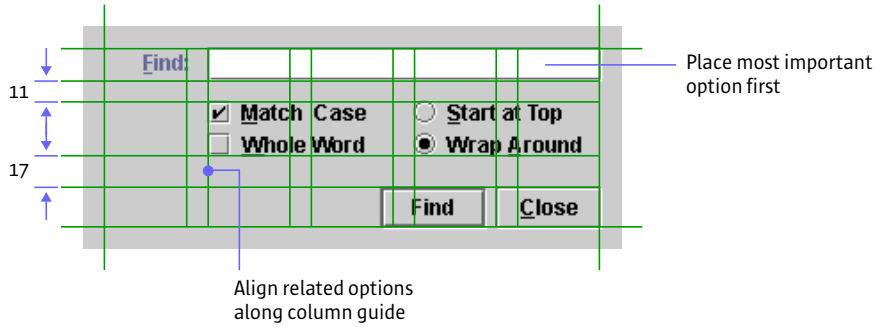
FIGURE 32 Beginning With Functional Requirements



Using the grid as a guide, add the rest of the components placing the most important option first with the related options aligned with it along one of the column guides.

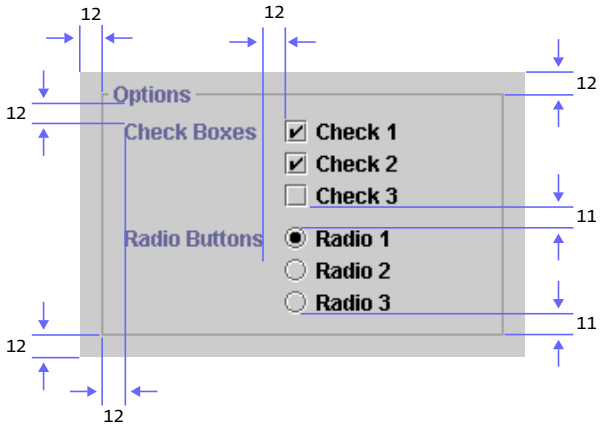
In the following illustration, the most important option—the text field for the search string, has been placed first—related options are aligned with it along one of the column guides. Spacing between components and groups of components follow the Java look and feel standards.

FIGURE 33 Using a Grid When Adding Remaining Components



Bordered Panes A bordered pane is a grouping of user interface elements in a pane with a label.

FIGURE 34 Bordered Pane Spacing Guidelines



A bordered pane can be created as follows:

```
myPanel.setBorder(new TitledBorder(new LineBorder
(MetalLookAndFeel.getControlShadow()),
"<< Your Text Here >>"));
```

The top position for the title of bordered panes should be used, with left justification.

Use bordered panes sparingly: they are best when you must emphasize one group of components or separate one group of components from other components in the same window. Do not use multiple rows and columns of bordered panes; they can be distracting and more confusing than grouping the elements using a design grid.

Use a bordered pane to group two or more sets of related components—for instance, do not draw bordered panes around a single set of checkboxes or radio buttons. Never nest bordered panes.

Spacing Guidelines The following figure shows guidelines for spacing components within a bordered pane.

Insert 12 pixels of space between the left border of the label for the bordered pane and the edge of the pane, the lower edge of the label for the bordered pane and the top edge of the label for a group of components, and the right edge of the labels for the component groups and the left edge of the components themselves. Insert 11 pixels of space between component

groups and between the lower edge of the bottom component group and the lower border of the bordered pane. Insert 12 pixels between the bottom edge of the bordered pane and the lower border.

 Allow for internationalized titles and labels in your bordered panes. When text strings are translated, they can grow or shrink. Use the appropriate layout manager to keep the bordered pane border the same size and let the contents grow or shrink.

Text Layout

<<New section in development: discuss label alignment; refer to capitalization section>>

Animation

Animation has great potential to be a useful and attractive part of a user interface if you use it appropriately. You can use animation to let users know that the system is busy with a task or to draw users' attention to important events.

 Since it distracts users and removes attention from all other elements of your application, do not overuse animation.

 Because inappropriate use of animation can cause eye irritation and unnecessary stress, use it with caution. In addition, screen readers, which are used by people with visual disabilities, do not recognize images that move, so be sure to provide `accessibleName` and `accessibleDescription` fields for that information.

Progress or Delay Indication

Feedback lets users know the application has received their input and is operating on it. In most cases, they receive feedback immediately (for example, a dialog box closes when users click the Close button).

Animation is especially useful when you want to show progress or to indicate delay (that is, communicate that the system is busy). Properly used, animation can be of minimal disruption to the user.

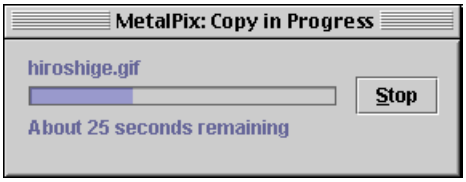
 When the application is processing a long operation and users can continue to work in other areas of the application, provide them with information regarding the state of the process.

 During a long operation, when the user can do nothing but wait until the operation is complete or switch to another application, change the shape of the pointer.

For example, an application's pointer might change to the wait pointer between the time the user selects a file and the file opens. For information on pointer shapes available in the Java look and feel, see Table 7 on page 76.

If you know the estimated length of an operation (for example, if the user is copying files) or the number of operations, use the Java look and feel progress bar. This bar fills from left to right as the operation progresses, as shown in the following figure.

FIGURE 35 Example of a Progress Dialog Box



For more on progress bars, see "Progress Bars" on page 157.

If an operation of unknown length is underway (for example, an operation that takes place over a network), animate a button, icon, or other application component.

For example, in today's web browsers, an animated product icon indicates that the page is still loading.

In-button animation can be used to indicate delays on the web and in multithreaded applications. This technique animates the image on a button to indicate progress on the associated task.

Another way to indicate delay is to use the animated cursors that are available with the Java 2 Platform. Instead of just changing to a wait pointer, you can go one step further by animating the pointer image while the system is busy.

System Status Animation

Animation is also useful when you want to call attention to events occurring in the system. For instance, in a mail application, you might use animation to indicate that new mail has arrived. Another example is a monitoring system that uses animation to alert users when failures occur.

When creating system status animation, consider the target users and their environment. If the animation needs to be visible from across the room, a bolder animation coupled with sound might be just the right thing. On the other hand, that same animation viewed by a user sitting at the workstation would be annoying.

When feasible, let users configure this type of animation, so they can adapt their systems to the environment.

Overcoming flicker and choppiness in animation is a key issue. Smooth animation requires fewer than 16 images per second and some require more. A smoothly moving object (like real cell animation) may require 20. In many systems, this number is neither attainable nor necessary if your goal is simply to indicate some activity. It is helpful to understand the limitations of your target platform before designing your animations.

5: DESIGNING APPLICATION GRAPHICS

The visual aspects of the Java look and feel balance appeal with compatibility. The degree to which your application graphics mesh with the system-generated graphics on your target platforms is equally important. This chapter provides details on the use of cross-platform color and the design of application graphics, such as button graphics, icons, and symbols, as well as the use of graphics that can enhance your product and corporate identity.

 Because the quality of your graphics can affect user confidence and even the perceived stability of your application, seek the advice of a professional visual designer.

Working With Cross-Platform Color

Since the Java platform essentially provides a cross-platform delivery environment, you need to ensure that the visual components of your application reproduce legibly and aesthetically on all your target systems. In many cases, you might not know which platforms will be used to run your software, or what display capabilities they might have.

Online graphics consist of the visual representations of JFC components in the Java look and feel, which are drawn for you by the toolkit, and application graphics such as icons and splash screens, which you supply.

The Java look and feel components use a simple color model that reproduces well on displays even with a relatively small number of available colors. You can use themes to change the colors of the components. For details, see "Themes" on page 35.

 Use themes to control the colors of Java look and feel components, for instance to provide support for display devices with minimal available colors (fewer than 16 colors, or four bits), or for visually impaired users.

You need to supply graphics such as icons, button graphics, and pictures and logos for splash screens and about boxes. Since these graphics might be displayed on a number of different platforms and configurations, it is your responsibility to develop a strategy for ensuring a high quality of reproduction.

 Use color only as a secondary means of representing important information. Use other characteristics such as shape, texture, size, or intensity contrast because they do not require color vision (or a color monitor) to recognize.

The colors available on your users' systems, along with graphic file formats, determine how accurately the colors you choose are displayed on screen. Judging color availability is the more difficult task, especially when you are designing applications to be delivered on multiple configurations or platforms.

Using Available Colors

The number of colors available on a system is determined by the **bit depth**, which is the number of bits of information used to represent a single pixel on the monitor. The lowest number of bits used for modern desktop color monitors is usually 8 bits (256 colors); 16 bits provide for thousands of colors (65,536, to be exact); and 24 bits, common on newer systems, provide for millions of colors (16,777,216), greater than the number that the human eye can distinguish. The specific colors available on a system are determined by the way in which the target platform allocates colors. Sometimes, available colors differ from application to application.

Designers sometimes use predefined color palettes when producing images. For example, some web designers work within a set of 216 "web safe" colors that reproduce, on 8-bit systems, in many web browsers without **dithering** (as long as the system is capable of displaying at least 256 colors). Dithering occurs when a system or application attempts to simulate an unavailable color by using a pattern of two or more colors or shades from the system palette.

Outside web browsers, available colors are not so predictable. Individual platforms have different standard colors or deal with palettes in a dynamic way. The web safe colors might dither when running in a stand-alone application, or even in an applet within a browser that usually does not dither these colors. Since the colors available to a Java application can differ each time it is run, especially across platforms, you cannot always avoid dithering in your images.

Different monitors and different platforms might display the same color differently. For instance, a given color in the same GIF file, might display a color that looks different to the eye from one system to another.

 Identify and understand the way that your target platforms handle colors at different bit depths. To achieve your desired effect, test your graphics on all target platforms at depths less than 16 bits.

Using Graphic File Formats

You can use two graphic file formats for images on the Java platform: **GIF** (Graphics Interchange Format) and **JPEG** (named after its developers, the Joint Photographic Experts Group).

GIF is the common format for application graphics in the Java look and feel. GIF files tend to be smaller on disk and in memory. Each GIF image is limited to 256 colors, or 8 bits of color information per pixel. A GIF file includes a list of colors used in the image (a palette). This palette can include fewer than 256 colors. The number of colors in the palette and the complexity of the image are two factors that affect the size of the graphic file.

On 8 bit systems, some of the colors specified in a GIF file will be unavailable unless they are all part of the system's current color palette. These unavailable colors will be dithered by the system. On 16 and 24 bit systems, more colors are available so different sets of colors can be used in different GIF files, but each image is still restricted to a set of 256 colors.

JPEG graphics use a "lossy" compression algorithm that yields different image quality depending on the compression setting, whereas GIF graphics use lossless compression that preserves the appearance of the original 8 bit image. JPEG graphics are generally better suited for photographs than for the more symbolic style of icons, button graphics and the corporate type and logos used in many splash screens and about boxes.

Choosing Colors

At monitor depths greater than 8 bits, most concerns about how any particular color reproduces become less significant. Any system capable of displaying thousands (16 bits) or millions (24 bits) of colors can find a color very close to, or exactly the same as, each value defined in a given image. Newer systems typically display a minimum of thousands of colors. Phosphor colors per monitor vary slightly which also may affect color appearance.

Many monitors or systems still display only 256 colors, however. In these cases, it might be advantageous to use colors known to exist in the system palette of the target platforms. Most platforms include a small set of "reserved" colors that are always available. Unfortunately, these reserved colors are often not useful for visual design purposes and for interface elements because they are highly saturated (the overpowering hues one might expect to find in a basic box of magic markers). Furthermore, little or no overlap exists between the reserved color sets of different platforms.

Select colors that do not overwhelm the content of your application or distract users from their tasks. Stay away from saturated hues. For the sake of visual appeal and ease of use, choose groups of muted tones for your interface elements.

Since there is no lowest common denominator solution for choosing common colors across platforms, or even colors that are guaranteed to reproduce on a single platform, some of the colors in your application graphics will dither when running in 8 bit color. The best strategy is to design images that dither gracefully.

FIGURE 36 Java Look and Feel Palette <<forthcoming from Chris>>

<<Forthcoming guideline about what to do with Java look and feel palette; for instance, should designers use it for the first and last color they work with during a design task?>>

Maximizing Color Quality Images with fine color detail often reproduce better on 8- bit systems than those that are mapped to a predefined palette (such as the web safe palette) and use larger areas of solid colors. Dithering in small areas will be less noticeable than it is over a larger area, and, for isolated pixels of a given color, is reduced to color substitution. Often colors are available in a system palette that can provide a fair to good match with those specified in a GIF file. The overall effect can be preferable to the dithering patterns produced for single colors, or the limited number of colors resulting from pre-mapping to a given color palette.

The following table shows a graphic with a blur effect that contains a large number of grays. Remapping this graphic to the web-safe palette reduces the number of grays to two, and results in a terrible approximation of the original

graphic. However, the original GIF file displays acceptably in a Java application running in 8 bit color on various operating systems, even though they may not have available the exact colors in the image.

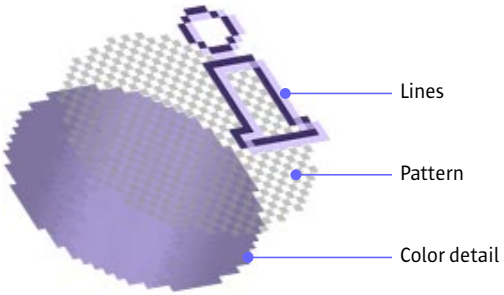
TABLE 4 Remappings of Blurred Graphic

	Original Graphic	Microsoft Windows	Macintosh	CDE
Original colors				
Remapped to web safe palette				

Areas of solid color often dither, producing distracting patterns. There are absolutely no safe cross-platform colors. One effective way to take advantage of this result is to use a pattern to “pre-dither” your artwork intentionally. This approach minimizes obvious patterned dithering on 8-bit systems while still permitting very pleasing effects on systems capable of displaying more than 256 colors.

To achieve this effect, overlay a semi-transparent checkerboard pattern on your graphics. The following figure shows how to build a graphic using the technique described in the steps that follow. <<3-D layering needs to be made more clear in the illustration. Gary, can you help? Perhaps we should use the order of the layering that is used in the illustration on page 69 after step 6?>>

FIGURE 37 Adding a Pattern to Prevent Dithering



1. Use a graphics program with layers.
2. Only apply the pattern to color detail areas.

   Leave borders and other detail lines as solid colors.

3. Play with the transparency setting for the pattern layer until the pattern is dark enough to mix with the color detail without overwhelming it.
4. Use a 25% transparency with the default Secondary 2 color (RGB 153-153-153) to produce a good result for most graphics.
5. Test your results on your target 8 bit platforms.

The following table shows the variable results of graphic reproduction in 8 bit color, using different styles in various operating systems.

TABLE 5 Variations in Reproduction in 8-Bit Color

Styles	Original Graphic	Windows 95 (8 bit)	Mac 8.5 (8 bit)	CDE (8 bit)
Plain				
Gradient				
Dithering Added				

The plain graphic in the preceding table, which uses a large area of a single web-safe color, dithers badly on Windows 95 and CDE. A gradient effect is added to the graphic to add some visual interest; this produces a bad banding effect on Mac 8.5. Adding the dithered pattern produces good results on all three platforms. In 16 and 24 bit color, the graphic reproduces very close to, or exactly the same as, the originals in the preceding table.

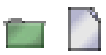
Designing Graphics in the Java Look and Feel Style

The following sections provide guidelines for designing application graphics in the style of the Java look and feel. Such graphics fall into three broad categories:

- Icons, which represent objects that users can select, open, or drag
- Button graphics, which identify actions, settings, and tools (modes of the application)

- Symbols, which are used for general identification and labelling (for instance, conditions or states)

TABLE 6 Examples of Application Graphics to Fit in With the Java Look and Feel

	Examples	Basic 3D Style	Pre-dithering
Icons			
Button Graphics			
Symbols			

Use the GIF file format for iconic and symbolic graphics. It usually results in a smaller file size than the JPEG format and uses lossless compression.

Where possible, use globally understood icons and graphics. Where none exist, create them with input from international sources. If you can't create a single symbol that works in all cultures, define appropriate graphics for different locales and include them in resource bundles. Examples of symbols that work in all cultures include a graphic of a car driving off a cliff to symbolize danger or the kinds of graphics used in hospitals and airports.

Designing Icons

An **icon** (a graphic representing an object that users can select, open, or drag) usually appears with identifying text. <<None of the examples currently show text; we need to make recommendations about where to put text in relationship to the icons; on small ones, place it to the right; on larger ones, place it below?>> Icons typically represent containers, documents, network objects, or other data that users can open or operate on within an application. For details on drag and drop operations on icons in Java look and feel applications, see "Drag and Drop Operations" on page 79.

The two standard sizes for icons are 16 x 16 pixels, and 32 x 32 pixels. The smaller size is more common and is used in JFC components such as the internal frame (to identify the contents of the window or minimized window) and tree view (for container and leaf nodes). You can use 32 x32 icons for low

vision applications or for representing objects in a diagram such as a network topology. For more on internal frames, see "Internal Frames" on page 105. For details on tree views, see "Tree Views" on page 182.

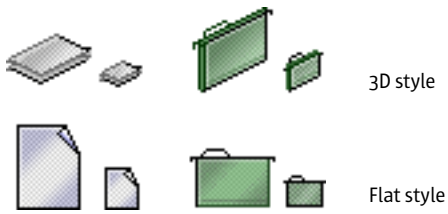
Design icons to identify clearly the objects or concepts they represent. Keep the drawing style symbolic, as opposed to photo-realistic. Too much detail can make it more difficult for users to recognize what the icon is intended to represent.

When designing large and small icons that represent the same object, make sure that they have similar shape, color, and detail.

An `accesibleName` and `accessibleDescription` field should be provided for each icon so that assistive technologies can find out what it is.

The following figure shows sample 32 x 32 and 16 x 16 icons for files and folders in two different styles, 3D and "flat."

FIGURE 38 Examples of Two Families of Icons



However, many objects are difficult to draw in a 3D style, particularly at the smaller 16 x 16 size.

Use a single style to create a "family" of icons that utilize common visual elements to reflect similar concepts, roles, and identity. Such icon families might use a similar palette, size, and style for different icons, such as folders and documents.

Don't mix two- and three-dimensional styles in the same icon family.

Note the visual elements of icons in the illustrations of the details of sample folder icons:

- interior highlight (to preserve the flush style used throughout the Java look and feel)
- dithering (described in "Using Available Colors" on page 54)
- dark border

Use a clear, dark exterior border and ensure that there is no anti-aliasing or other detail around the perimeter of the graphic, for satisfactory display on a wide range of background colors and textures.

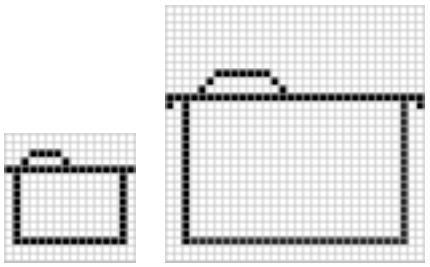
Drawing Icons

The following section uses a simple folder icon as an example of how to draw an icon. The first step in drawing an icon is to decide on a general design for the object to be represented. In this example, a hanging file folder is used to represent a directory.

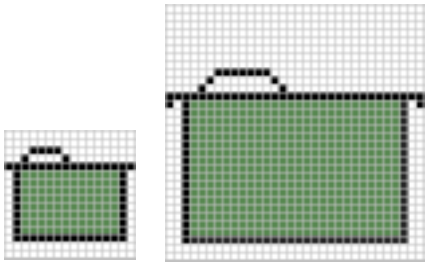
1. Draw a basic outline shape first.

   Icons can use as much of the available space as possible, since they are displayed without borders

   If both sizes are required, it is a good idea to work on them at the same time, rather than trying to scale down a detailed 32 x 32 icon later; this way both sizes can evolve into a design that is recognizable as representing the same object.

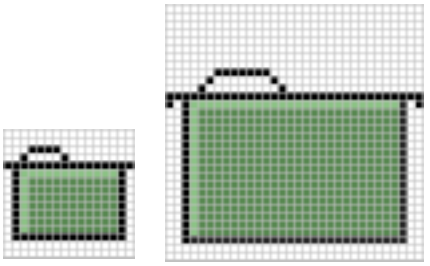


2. Next, add some basic color.



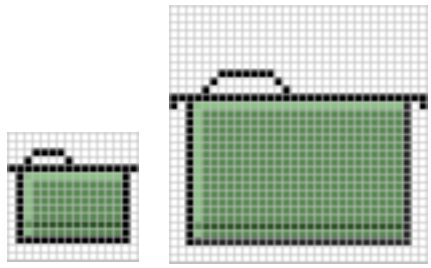
3. Draw a highlight on the inside top and left.

   This practice creates the flush three dimensional appearance of the Java look and feel.



4. Add some detail to the icon.

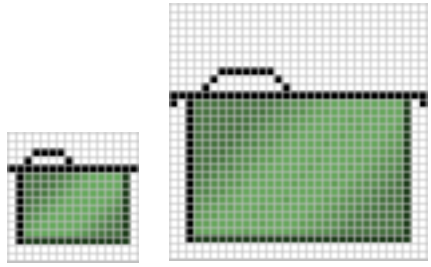
   In this case, the crease or “fold” mark in the hanging folder is drawn.



5. Try a gradient that produces a “shining” effect instead of the flat green.

   Here is an example of the use of a gradient in an icon.

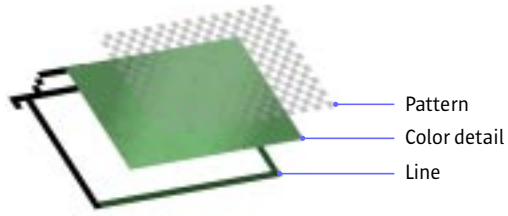
   A dark green has replaced the black border on the right and bottom; black is not a requirement as long as there is a well-defined border.



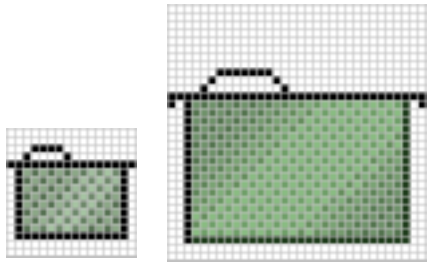
6. Add dithering.

   This technique minimizes banding and dithering on displays with 256 or fewer colors (see “Maximizing Color Quality” on page 56).

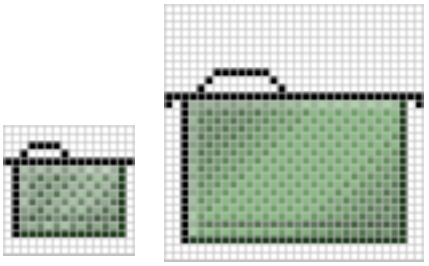
   Here is an exploded view of an icon that shows the layers of the dithering process:



   Here is an example of an icon in which dithering has been added to the color detail.

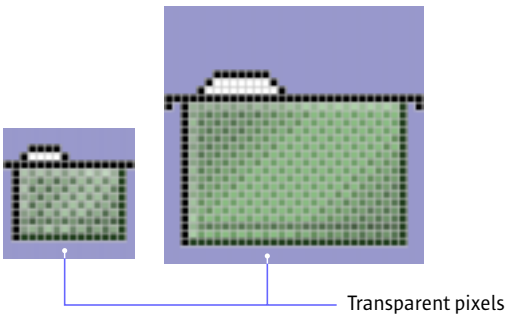


7. Reintroduce the interior flush highlight and the "fold" detail. <<Chris: why does this need to be reintroduced if it is created on a separate layer?>>.



8. Define the empty area around the icon graphic (in which you have not drawn anything) as transparent pixels in the GIF file.

   This practice ensures that the background color will show through; if the icon is dragged to or displayed on a different background, the area surrounding it will match the color or pattern of the rest of the background.



Designing Button Graphics

Button graphics appear inside buttons—for instance, in toolbar buttons. Such graphics identify the action, setting, mode, or other function represented by the button. For instance, clicking the button might carry out an action (creating a new file) or set a state (boldfacing text).

Do not include text as part of your button graphics. Instead include it in your button text. If you include both text and graphics in a button, the size of the button will probably exceed 16 x16 or even 24 x 24 pixels. If the button size is an issue, consider adding labels underneath the button.

However, note that toolbar buttons can also display text instead of graphics, particularly if your usability testing establishes that the action, state or mode represented by the button is difficult for users to comprehend.

The two standard sizes for button graphics are 16 x 16 pixels, and 24 x 24 pixels. Either size (but not both at the same time) can be used in toolbars or tool palettes, depending on the amount of space available. For details on toolbars, see "Toolbars" on page 134. For more on palette windows, see "Palettes" on page 107.

When designing your button graphics, clearly indicate the action, state, or mode that the button initiates.

Keep the drawing style symbolic; too much detail can make it more difficult for users to understand what the button does.

Use a flush 3D border style to indicate that the button is clickable.

Draw a clear, dark border without anti-aliasing or other exterior detail (except the flush 3D highlight) around the outside of the graphic.

The following figure shows sample button graphics.

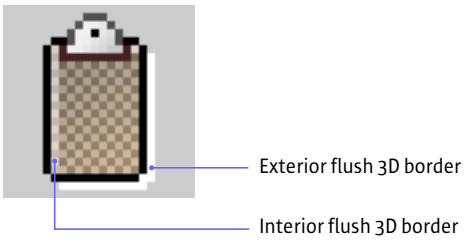
FIGURE 39 Button Graphics for a Toolbar and a Tool Palette

Use a single style to create a "family" of button graphics that utilize common visual elements to reflect similar concepts, roles, and identity. Such button families might use a similar palette, size, and style for different button graphics, such as toolbar buttons, toggle buttons, or command buttons.

Don't mix two and three dimensional styles in the same button family.

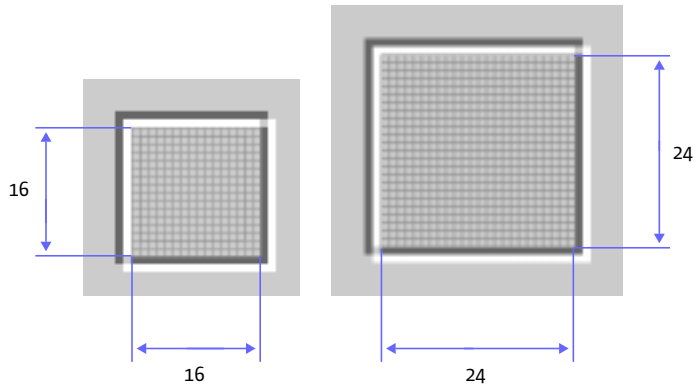
Producing the 3D Effect To produce the flush three-dimensional effect, you add an exterior white highlight on the outside right and bottom of the graphic and an interior highlight on the inside left and top.

FIGURE 40 Flush 3D Effect in a Button Graphic <<needs callouts>>



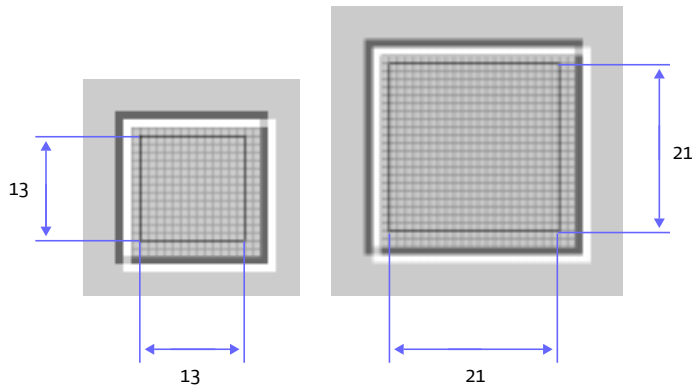
Working With Button Borders The size of a button graphic includes all the pixels within the border. As shown in the following illustration, horizontal and vertical dimensions are both either 24 or 16 pixels. The border abuts the button graphic as shown (there are no pixels between the border and the graphic).

FIGURE 41 Button Graphics With Borders <<need callouts>>



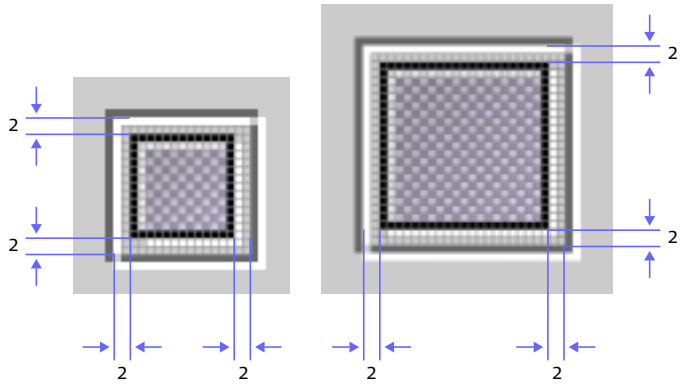
Determining the Primary Drawing Area Because the white pixels in both the button border and the button graphic are less visually significant than the darker borders, the area used for most of the drawing is offset within the 16x16 or 24x24 space. The following illustration shows the standard drawing area for both button sizes. Note that the white highlight used to produce the flush 3D appearance in the button graphic may fall outside this area.

FIGURE 42 Primary Drawing Area in Buttons <<need callouts>>



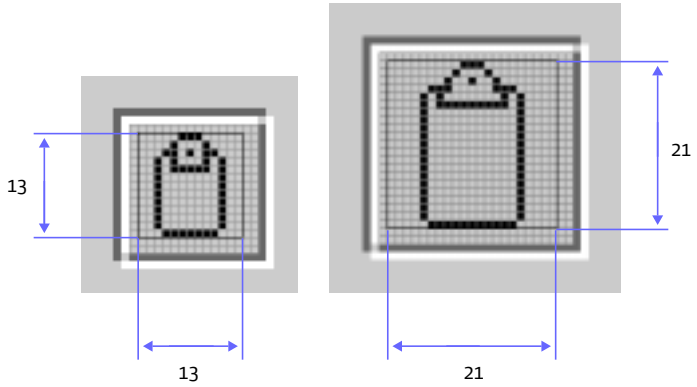
The following illustrations show hypothetical 16 x 16 and 24 x 24 button graphics that use the maximum recommended drawing area. Notice that there are two pixels all the way around between the dark border of the button graphic and the dark portion of the button border.

FIGURE 43 Maximum Size Button Graphics

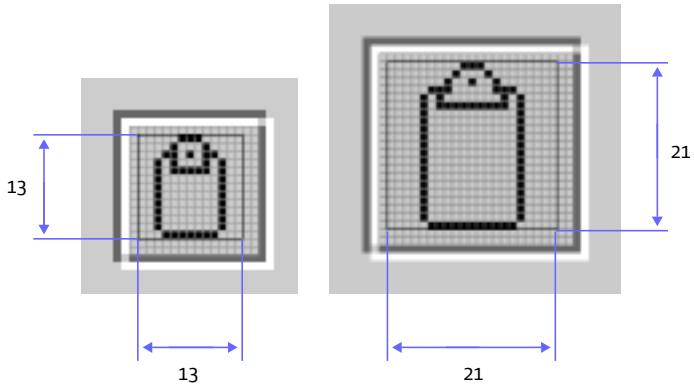


Drawing the Button Graphic When drawing a button graphic, first decide on a general design for the action or setting that the graphic is to represent. In the following series of examples, a clipboard suggests the copy command.

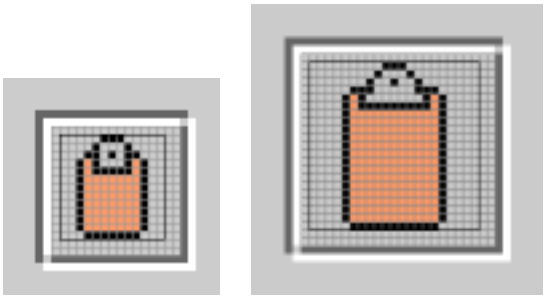
1. Decide which size you want to use for the button or tool bar graphic.



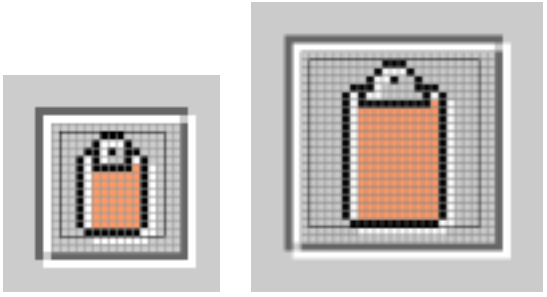
2. Draw a basic outline shape, taking care to remain within the primary drawing area.



3. Add some basic color.

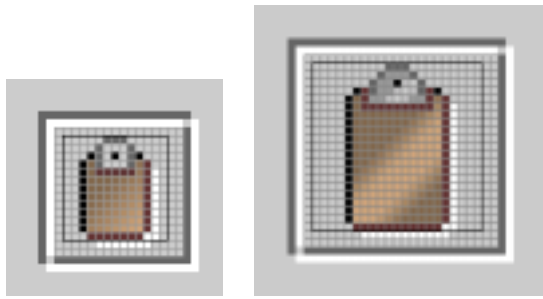


4. Add the flush three dimensional effect by drawing highlights on the inside left and top, and the outside bottom and right of the outline.



This is a good basic design, but because of the large area using a single color, the graphic lacks realism and visual interest and might not reproduce well on some systems (see "Working With Cross-Platform Color" on page 53).

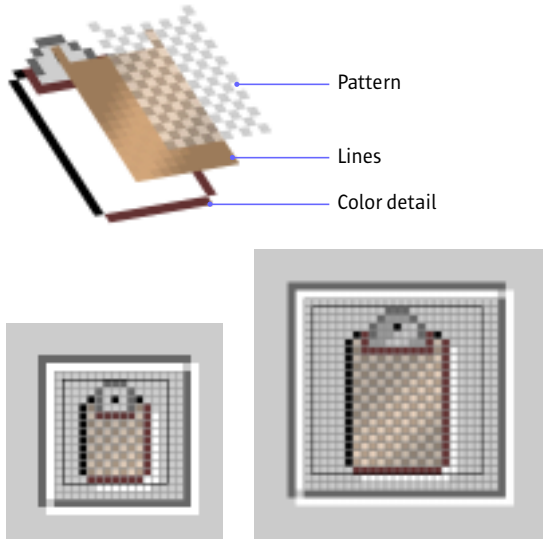
5. Try a gradient instead of the flat brown.



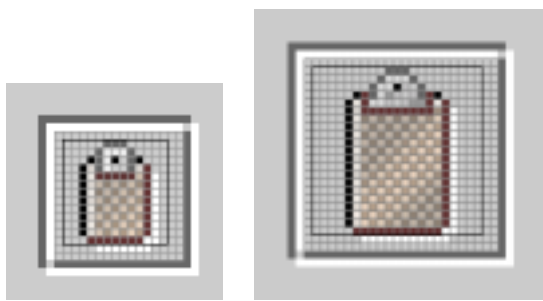
6. Add dithering.

   This technique minimizes banding and dithering on displays with 256 or fewer colors (see "Maximizing Color Quality" on page 56). The following figure shows an exploded view of the button graphic without flush 3D highlights.)

   The next figure shows the effect of dithering on the color detail of the button graphic.

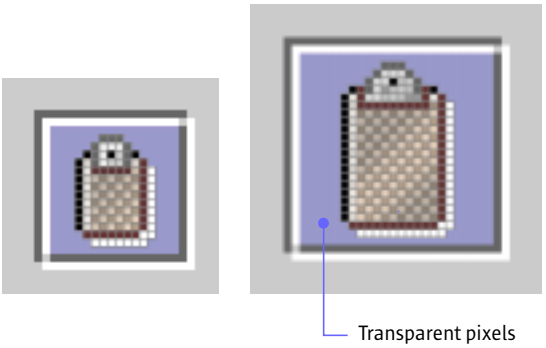


7. Reintroduce the interior flush highlight.



8. Define the empty area around your button graphic (in which you have not drawn anything) as transparent pixels in the GIF file.

   This practice ensures that the background color will show through; if the theme is changed, the area around the button graphic will match the rest of the background canvas in the interface.



Designing Symbols

Symbols include any small graphic (typically 48 x 48 pixels or smaller) that stands for a state or a concept but has no directly associated action or object. You can use symbols within dialog boxes or system status alert boxes. Saturated colors might be useful for status or warning graphics.

The examples in the following figure show the graphic from the Java look and feel Info alert box and a warning symbol superimposed on a folder icon to indicate a hypothetical state. The style for symbols is not as narrowly defined as that of icons and button graphics. The examples in the following figure use a flush or etched effect for interior detail, but not for the border of the graphic.

FIGURE 44 Examples of Symbols

Information symbol Warning symbol

Designing Graphics for Corporate and Product Identity

Application graphics present an excellent opportunity for you to enhance your corporate or product identity. This section presents examples of installation screens, splash screens and About boxes for the sample text-editing application, MetalEdit.

Use the JPEG file format for any photographic elements in your installation screens, splash screens, and about boxes.

Designing Installation Screens An installation screen is a window containing images that are displayed in an application installer. Often the first glimpse users have of your application is the installer. Consequently, an installation screen introduces and reinforces your corporate and product identity. The number of screens in a given installer can vary.

Use a plain window for installation screens, and draw any desired border inside the window.

The `JWindow` component is typically used to implement plain windows.

Provide a way for your users to move through the steps required to perform the installation, and to cancel or stop the installation at any point.

See “Layout and Visual Alignment” on page 43 for general guidelines on how to arrange and align items.

Designing Splash Screens A splash screen is a plain window that appears briefly in the time between the launch of a program and the appearance of its main application window. Nothing other than a blank space is provided with a plain window, The black border on the window in the following figure is part of the GIF file not part of the plain window component.

FIGURE 45 Splash Screen for MetalEdit

Although not required, splash screens are included in most commercial products. Splash screens typically have the following elements:

- Product name
- Company logo
- Product logo or a visual identifier of the product
- Copyright notice

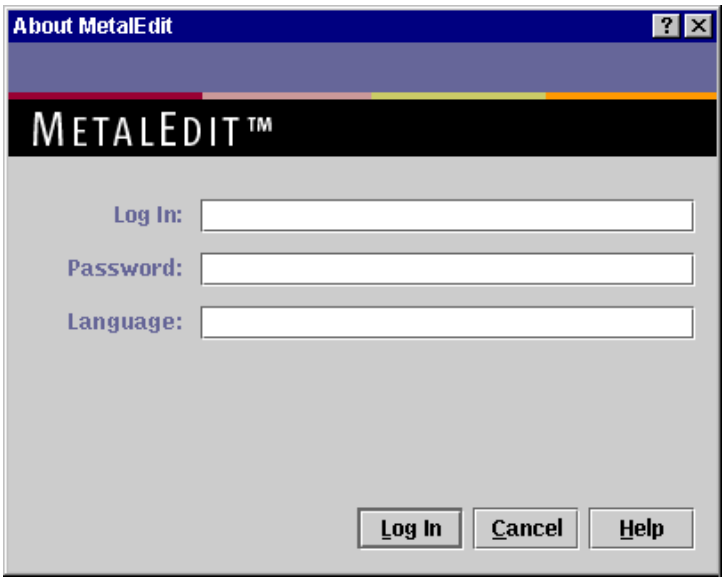
Check with your legal adviser regarding your legal requirements.

The `JWindow` component, not the `JFrame` component, is typically used to implement the plain window that provides the basis for splash screens.

To get the black border that is recommended for splash screens, you must include a 1 pixel black border as part of the image you create.

Designing Login Splash Screens If your application requires users to log in, you might consider replacing the traditional splash screen with a log-in splash screen.

FIGURE 46 Sample Login Splash Screen



The elements of this screen are similar to splash screens and might include the following:

- Label and text field for a user name
- Label and text field for a password
- Label and text field for any other information required by the system
- Buttons for login and exit

To save users time while waiting for a splash screen to appear and to increase the chance of users viewing a splash screen, it is a good idea to use combined login/splash screens.

Provide a way for users to exit the login splash screen without logging in.

Designing About Boxes

An About box is a dialog box that contains basic information about your application.

FIGURE 47 About Box for MetalEdit

An About box typically contains the following elements:

- Product name
- Version number
- Company logo
- Product logo or a visual reminder of the product logo
- Copyright, trademarks, and other legal notices
- Names of contributors to the product

Because users typically access About boxes from the Help menu, make the About box accessible while your application is running.

Include Close buttons in your About boxes so that users can dismiss them after reading them. Follow the guidelines for button placement in dialog boxes described in "Spacing and Alignment" on page 111.

6: BEHAVIOR

Users interact with the computer via the mouse, keyboard, and the screen. Interaction is the “feel” portion of the Java look and feel. This chapter provides input guidelines and recommendations for interaction techniques. It covers mouse operation, drag and drop, and keyboard operations.

Mouse Operations

In Java look and feel applications, the following common mouse operations are available to users:

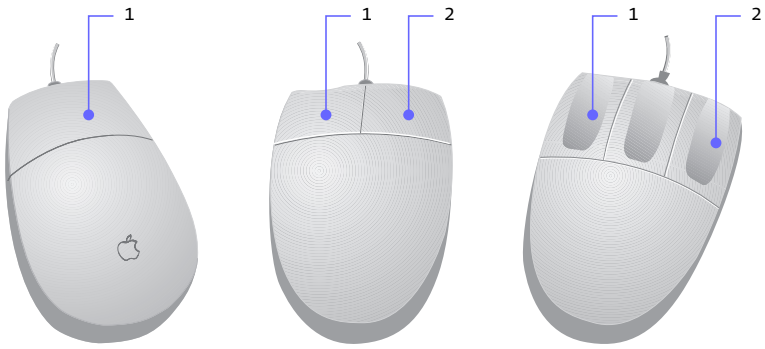
- Moving the mouse moves the onscreen mouse pointer (often called the “cursor”).
- Clicking (pressing and releasing a mouse button while the mouse is over the same object) selects or activates the object. The object is normally highlighted when the button is pressed and then selected or activated when the button is released. For example, a mouse click is used to activate a command button or to select an item from a list or to place an insertion point in a text area.
- Double-clicking (clicking a mouse twice in rapid succession without moving the mouse) is used to select larger units (for example, to select a word in a text field) or to select an open an object.
- Dragging the mouse (pressing a mouse button, moving the mouse, and releasing the mouse button) is used to select a range of objects, to choose items from drop-down menus, or to move objects in the interface.


In your design, assume a two-button mouse. Use mouse button 1 (usually the left button) for selection, activation of components, dragging, and to post or drop down menus. Use mouse button 2 (usually the right button) to drop down or post contextual menus. Do not use the middle mouse button; it is not available on most target platforms.


Macintosh systems usually have a one-button mouse, whereas personal computers and network computers usually have a two-button mouse. However, UNIX systems usually have a three-button mouse.


Restrict interaction to the use of mouse button 1 and mouse button 2. Macintosh users can simulate mouse button 2 by holding down the Control key while using mouse button 1.

FIGURE 48 Cross-Platform Mouse Buttons and Their Default Assignments



Pointer Feedback

The mouse pointer can assume a variety of shapes. For instance, in a text-editing application, the pointer might assume an I-beam shape (called a text pointer in JDK) to indicate where the insertion point would be placed if the user presses the mouse button. The **insertion point** is the place, usually set by clicking with the text pointer, where typed text or a dragged or pasted selection will appear. When the pointer moves back out of the text pane, it then returns to its initial appearance as a default pointer.

The Java look and feel defines a set of pointer types that map to the corresponding native platform pointers; therefore, the appearance of pointers can vary from platform to platform, as shown in the following table. When no corresponding pointer exists in the native platform toolkit, the pointer is supplied by the JDK.

TABLE 7 Pointer Types Available in JDK 1.1 and Java 2 Platform

Pointer	Macintosh	Windows 95	CDE	Usage in Java Look and Feel Applications
Default				Pointing, selecting, or moving
Crosshair				Interacting with graphic objects
Hand				Panning objects by direct manipulation
Move				Moving objects
Text				Selecting or inserting text
Wait				Indicating that an operation is in progress and the user cannot do other tasks
S Resize				Changing the size of an object

TABLE 7 Pointer Types Available in JDK 1.1 and Java 2 Platform *(Continued)*

Pointer	Macintosh	Windows 95	CDE	Usage in Java Look and Feel Applications
N Resize				Changing the size of an object
E Resize				Changing the size of an object
W Resize				Changing the size of an object
NW Resize				Changing the size of an object
NE Resize				Changing the size of an object
SE Resize				Changing the size of an object
SW Resize				Changing the size of an object

 In addition to the shapes in Table 7, pointer graphics can be defined as an image and created using `Toolkit.createCustomCursor()` if you are using Java 2 or later.

Mouseover Feedback

Mouseover feedback is a visual effect that occurs when the user rolls the mouse over an area of an application window.

Mouseover feedback is used in the Java look and feel to show borders on rollover toolbar buttons. A slightly different effect is used to display tool tips. For details, see "Toolbars" on page 134 and "Tool Tips" on page 138.

Clicking and Selecting Objects

In the Java look and feel, the selection of objects with the mouse, including text, lists, and icons is similar to the general practice on other graphical user interfaces. The user selects an object by clicking it (that is, by positioning the mouse pointer over the object and pressing and releasing mouse button 1). Clicking also deselects any previous selection.

 Provide for the selection of text as follows:

- Single-click to deselect any existing selection and set the insertion point.
- Double-click on a word to deselect any existing selection and select the word.
- Triple-click in a line of text to deselect any existing selection and select the line.
- Drag (that is, move the mouse while holding down mouse button 1) through a range of text to deselect any existing selection and select the range.

- Shift-click to adjust the boundary of an existing selection. If there is an existing selection, adjust the nearest boundary of the selection to the click location. If there is an insertion point, select the text from the insertion point to the click location.

Provide for selection in lists and arrays as follows:

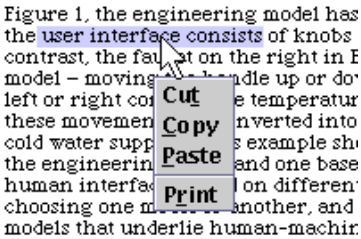
- Click an object to deselect any existing selection and select the object.
- Shift-click an object to extend the selection from the most recently selected object to the current object.
- Control-click an object to toggle its selection without affecting the selection on any other objects.

Contextual Menus It can be difficult for users to find and access desired features among all the commands in the menus and submenus of a complex application. Contextual menus (sometimes called “pop-up menus”) enable you to distribute the functions throughout the graphical interface and associate them with the relevant objects.

Users can access contextual menus two different ways:

- To pull down the menu, press and hold mouse button 2 over the relevant object. Then drag the mouse to the desired menu item and release to select it.
- To post the menu, click with mouse button 2 over a relevant object. Then position the mouse over the desired menu item and release to select it.

FIGURE 49 Contextual Menu on a Text Selection



Display contextual menus at the pointer’s current location for the convenience of the user.

Since users often have difficulty knowing whether contextual menus are available and what is in them, ensure that the items in your contextual menu also appear in the menu bar of the primary windows in your application.

Be sure that the commands in your contextual menu apply only to a selected object or group of objects. For instance, a contextual menu might include cut, copy, and paste commands limited to a selected text range, as shown in the preceding figure.

Remember that users on the Microsoft Windows and UNIX platforms display a contextual menu by clicking or pressing mouse button 2. Macintosh users utilize the Control key in conjunction with mouse button 1.

Drag and Drop Operations

Drag and drop operations include moving, copying, or linking selected objects by dragging them from one location and dropping them over another. These operations provide a convenient and intuitive way to perform many tasks using direct manipulation. Common examples of **drag and drop** in the user interface are moving files by dragging file icons between folders or dragging selected text from one document to another. Java 2 supports drag and drop between two Java applications or between a Java application and a native application. For example, on a Microsoft Windows system, the user can drag a text selection from a Java application and drop it into a Microsoft Word document.

Typical Drag and Drop

Drag and drop with Java applications is similar to other platforms. The user presses mouse button 1 while the pointer is over a source object and then drags the object by moving the pointer while keeping the button pressed. The object is dropped by releasing the button when the pointer is over a suitable destination. A successful drop triggers an action that depends on the nature of the source and destination. If the drag source is part of a range selection, the entire selection (for example, several file icons or a range of text) is dragged.

Pointer and Destination Feedback

Sometimes it may not be possible to move an outline or other representation of the drag source along with the pointer, so a changed pointer shape might be the only indication the user has that a successful drag is in progress.

Provide the user with feedback that a drag operation is in progress by changing the shape of the mouse pointer when the drag is initiated.

 Provide destination feedback so the user knows where the dragged object can be dropped. Use one or both of the following methods to provide destination feedback:

- Change the pointer shape to reflect whether the object is over a possible drop target.
- Highlight drop targets when the mouse pointer is over them to indicate that they can accept the target.

 Java objects are specified by their **MIME** (Multipurpose Internet Mail Extensions) type, and the Java Runtime environment automatically translates back and forth between MIME types and system-native types as needed. As the object is dragged over potential targets, the potential target can query the drag source to obtain a list of available data types and compare that with the list of data types that they can accept. For example, when dragging a range of text, the source may be able to deliver the text in a number of different encodings or as plain text, styled text, or HTML. If there is a match in data types, potential targets should highlight as the pointer passes over them to indicate that they can accept the dragged object.

Keyboard Operations

The Java look and feel assumes a PC-style keyboard. The standard ASCII keys are used, along with the following modifier keys: Shift, Control, and Alt (Option on the Macintosh); the function keys F1 through F12; the four arrow keys; Delete, Backspace, Home, End, Page Up, and Page Down. Enter and Return are equivalent. (Return does not appear on PC keyboards.)

A **modifier key** is a keyboard key that does not produce a character but can be used in combination with other keys to modify an action. Typical modifier keys in Java look and feel applications are Shift, Control, and Alt.

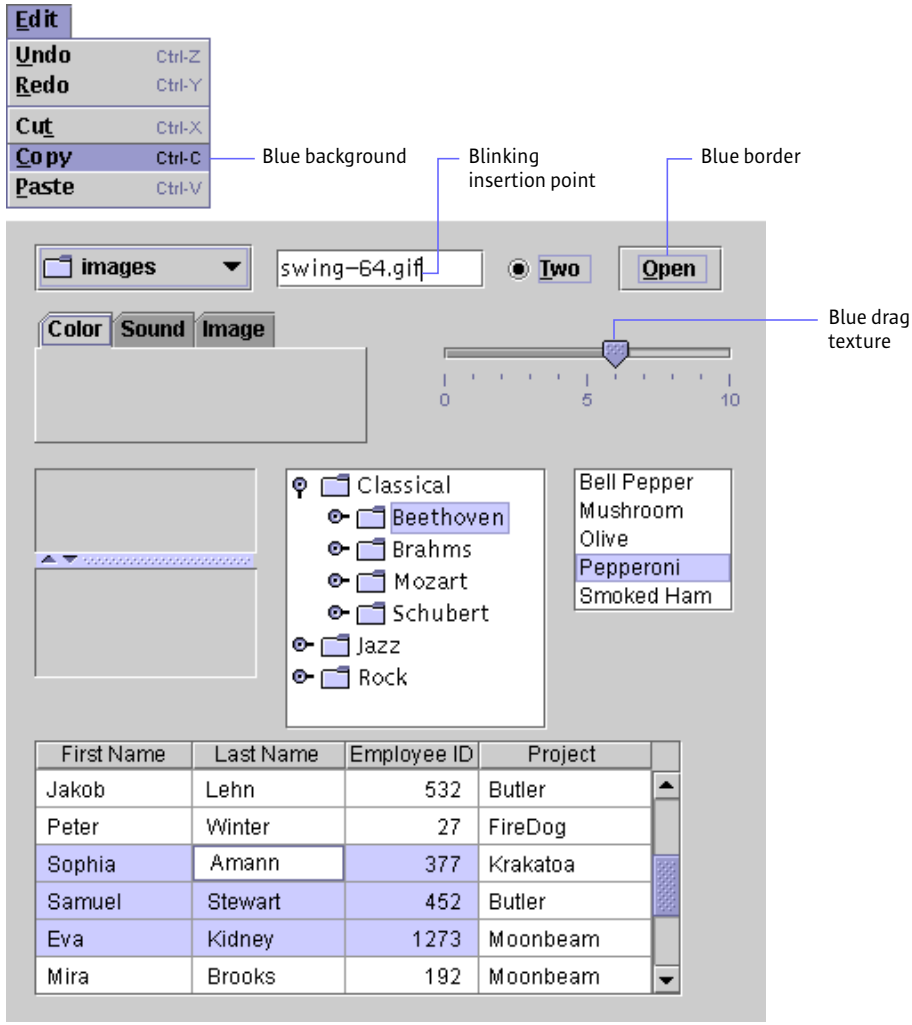
This section describes and provides recommendations for the use of keyboard operations, which refer to keyboard shortcuts, mnemonics, and other forms of navigation, selection, and activation that utilize the keyboard instead of the mouse. A **mnemonic** consists of a key or keys that users type to navigate to or activate a component. For instance, you could use a mnemonic to give focus to a text area or to activate a command button or a menu command. You indicate mnemonics in menus, command buttons, dialog boxes, and labels with underlined characters. A **keyboard shortcut** consists of a combination of keys, such as Ctrl-A, which activate a command.

Keyboard Focus

For any action to be performed through a user interface, components need to get focus. One way focus occurs is for a user to click with a pointing device on a component. Keyboard focus can also be controlled from the keyboard. Either way, the user designates the window or component within a window that receives input. The **keyboard focus** designates the active window or component, where the user's next keystrokes will take effect. (There are exceptions: for instance, a formatting button on a toolbar for left justification should not permanently take focus away from the text area where the actual work is taking place.)

In the Java look and feel, many components (including command buttons, checkboxes, radio buttons, toggle buttons, lists, combo boxes, tabbed panes, editable cells, and tree views) indicate keyboard focus by displaying a rectangular blue border. Editable text components such as text fields show a blinking insertion point. Menus indicate focus with a blue background. Split panes and sliders show the drag textured areas in blue. The following figure shows the appearance of keyboard focus in several components.

FIGURE 50 Keyboard Focus in Several Java Look and Feel Components



When a window is first opened, assign initial keyboard focus to the most logical component (typically the component that would be used first or the component in the upper left of the window). If keyboard focus is not assigned to a component, the keyboard navigation and control mechanisms cannot be used.

Appendix A provides details on keyboard navigation, activation, and selection.

Keyboard Navigation and Activation Keyboard navigation and activation allows the user to move keyboard focus from one user-interface component to another using the keyboard.

In general, Tab cycles through the major components; Shift-Tab cycles through the components in the reverse direction. Control-Tab and Control-Shift-Tab work in a similar fashion and are particularly useful when the keyboard focus is in an element that accepts tabs, such as a text area or a table. Arrow keys are often used to move within groups of components; for example, Tab gets the focus into a set of radio buttons and then the arrow keys move the focus among the radio buttons. However, the Tab key is used to move among checkboxes.

Once an element has focus, pressing the Spacebar typically activates it or selects it. In a list, pressing Shift-Spacebar extends the selection; pressing Control-Spacebar makes another selection without affecting the current selections.

Some components do not need explicit keyboard focus to be operated. For example, the default button in a dialog box can be operated by pressing the Return key or Enter key if the keyboard focus is anywhere in the dialog box. Similarly, scroll bars can be operated from the keyboard if focus is anywhere within the scroll pane.

The keyboard navigation and can be useful not only for accessibility purposes, but also for power users, those users who prefer the keyboard over the mouse, or those who choose to use alternative input methods like voice input or onscreen keyboards.

 Some of the keyboard navigation operations in the tables in Appendix A might be temporarily incomplete or unimplemented. However, you should reserve the key sequences listed in this appendix for future versions of JFC and the JDK.

 The `setNextFocusableComponent` method from `JComponent` can be used to set the order for tabbing by chaining components together, specifying for each component, what the next one in the sequence is.

 Ensure that all application functions are accessible from the keyboard.

The general operations for keyboard navigation and activation in the Java look and feel are summarized in the following table. Within the table, the term *group* refers to a group of radio buttons, toolbar buttons, menu bar titles, text, or table cells.

TABLE 8 Common Navigation and Activation Keys

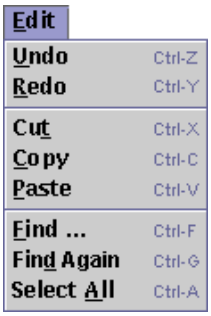
Keyboard Operation	**Action**
Tab[1]	Navigate in, navigate out
Control-Tab[1]	Navigate out of component that accepts tabs
Left Arrow	Move focus left one character or component within a group
Right Arrow	Move focus right one character or component within a group
Up Arrow	Move focus up one line or component within a group
Down Arrow	Move focus down one line or component within a group
Page Up	Move up one view
Page Down	Move down one view
Home	Move to beginning of data; in a table, move to the beginning of a line
End	Move to end of data; in a table, move to the last cell in a row
Enter or Return	Activate default command button
Escape	Dismiss menu or dialog box without changes
Spacebar	Activate or select the component (with keyboard focus)

1. With Shift key, reverses direction

Keyboard Shortcuts

Keyboard shortcuts are keystroke combinations (consisting of a modifier key and a character key, like Control-X) that activate the command associated with a menu item from the keyboard even if the menu for that menu item is not currently displayed. Unlike mnemonics, keyboard shortcuts do not post menus; rather, they perform the indicated actions directly.

FIGURE 51 Menu with Keyboard Shortcuts



To choose a keyboard shortcut, users can hold down the Control key and type a specified letter that is shown to the right of the menu item. Typing the keyboard shortcut has the same effect as choosing the menu item. For instance, to cut information, users can either choose the Cut command from the Edit menu or hold down the Control key and press X.

Do not use the Meta key for a keyboard shortcut, except as an alternate for Control.

Specify keyboard shortcuts for frequently used menu items to provide an alternative to mouse operation. The JFC displays them next to the menus using standard abbreviations for key names separated by hyphens).

Be aware of and use the common shortcuts across platforms that are summarized in the following table.

TABLE 9 Common Keyboard Shortcuts

Sequence	Equivalent
Ctrl-N	New (File menu)
Ctrl-O	Open (File menu)
Ctrl-S	Save (File menu)
Ctrl-P	Print (File menu)
Ctrl-Z	Undo (Edit menu)

TABLE 9 Common Keyboard Shortcuts *(Continued)*

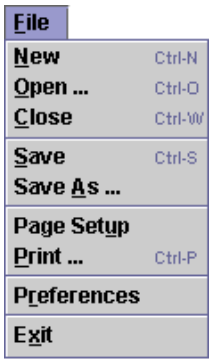
Sequence	Equivalent
Ctrl-X	Cut (Edit menu)
Ctrl-C	Copy (Edit menu)
Ctrl-V	Paste (Edit menu)
Ctrl-F	Find (Edit menu)
Ctrl-A	Select All (Edit menu)

Since keyboard shortcuts are not all equivalent on different platforms, ensure that any new keyboard shortcuts you have created are compatible with the existing shortcuts on all your target platforms.

Mnemonics

Mnemonics provide still another keyboard alternative to the mouse. A mnemonic is an underlined letter in a menu title, menu item, or interface component. It reminds the user how to activate the equivalent command by simultaneously pressing the Alt key and the character key that corresponds to the underlined letter.

FIGURE 52 Menu with Mnemonics and Keyboard Shortcuts



The text labels of components typically have one letter (the mnemonic) underlined. Pressing Alt plus the key associated with the letter moves the keyboard focus to that component and activates it. When keyboard focus is not in a text element, the Alt modifier may not be required (menus are an example).

Mnemonics are indicated by underlined letters in the user interface. For instance, to choose the New command from the File menu, the user can hold down the Alt key and type F, then type N.

Once users have displayed a menu with a keyboard sequence, the subsequent letter they type must be unique to that menu. Hence, users can press Alt-F to display the File menu and then type A to activate the Save As command, or press Alt-E to display the Edit menu, then type A to select all the elements in a document window.

You can also provide mnemonics for components within the dialog boxes in your applications. However, it is important to note that this situation requires that you use a modifier key. For instance, within a dialog box, you may want to provide a mnemonic for the Help button. Once keyboard focus has moved within the dialog box, users press Alt then type “H” to activate the Help button.

 Do not associate mnemonics with the default button and the cancel button in a dialog box. Use Enter or Return for the default button and Escape for the cancel button instead.

 Adjust mnemonics often to avoid conflicts. For instance, you cannot use the letter “P” as the mnemonic for both the Print and Page Setup commands.

 When you assign mnemonics, follow these guidelines in the specified order.

1. Use common mnemonics across platforms as they appear in the following table.

TABLE 10 Common Mnemonics

Menu Titles	Menu Items
File	New, Open, Close, Save, Save As, Page Setup, Print, Preferences, Exit
Edit	Undo, Redo, Cut, Copy, Paste, Find, Find Again, Select All
Help	Contents, Tutorial, Index, Search, About Application

2. If the mnemonic does not appear in the table of common mnemonics, choose the first letter of the menu item. For instance, choose J for Justify.

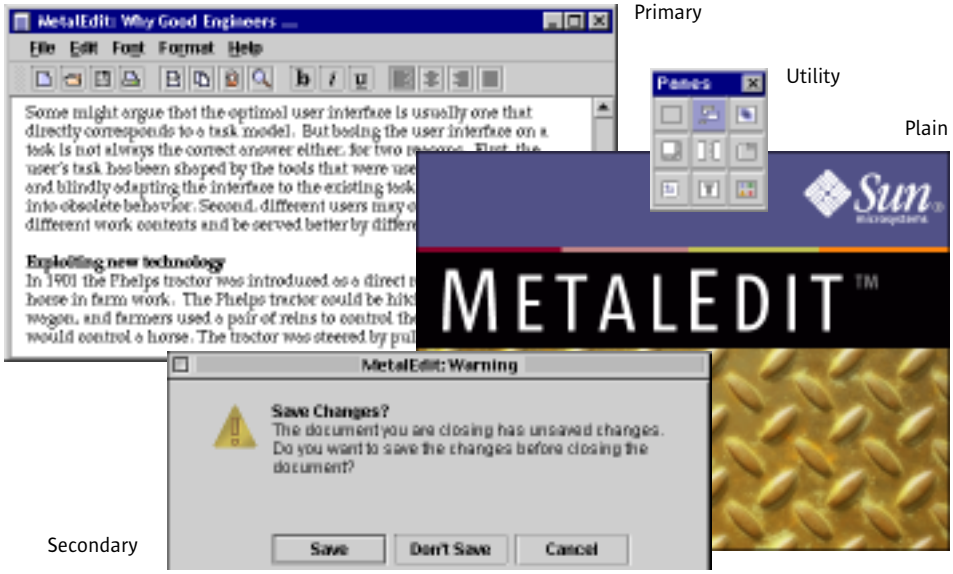
3. If the first letter of the menu item conflicts with those of other menus, choose a prominent consonant. For instance, the letter S has already been designated as the mnemonic for the Style command. Therefore, choose the letter Z as the mnemonic for the Size command.
4. If the first letter of the menu item and the prominent consonant conflict with those of other menu items, choose a prominent vowel.

PART III: THE COMPONENTS OF THE JAVA FOUNDATION CLASSES

7: WINDOWS, PANES, AND FRAMES

Windows provide the space in which users see and work with information. They also serve as the top-level containers in which you, as a designer, can place primary windows, secondary windows, utility windows, and plain windows. A **primary window** is a window in which the main interaction with the data or document takes place. An application uses any number of these windows, which are equivalent and can be opened, closed, minimized, or resized independently. Users do the majority of their viewing and editing in primary windows, such as document windows. A **secondary window** is a supportive window that is dependent on a primary window (or another secondary window) in which users can view and provide additional information about actions or objects in a primary window. Examples of secondary windows are dialog boxes and alert boxes. A **utility window** is a window whose contents affect an active primary window. Unlike secondary windows, utility windows remain open when any primary windows are closed or minimized. An example of a utility window is a tool palette that is used to choose a graphic tool. A **plain window** is a window with no title bar or window controls, typically used for splash screens.

FIGURE 53 Primary, Secondary, and Utility Windows



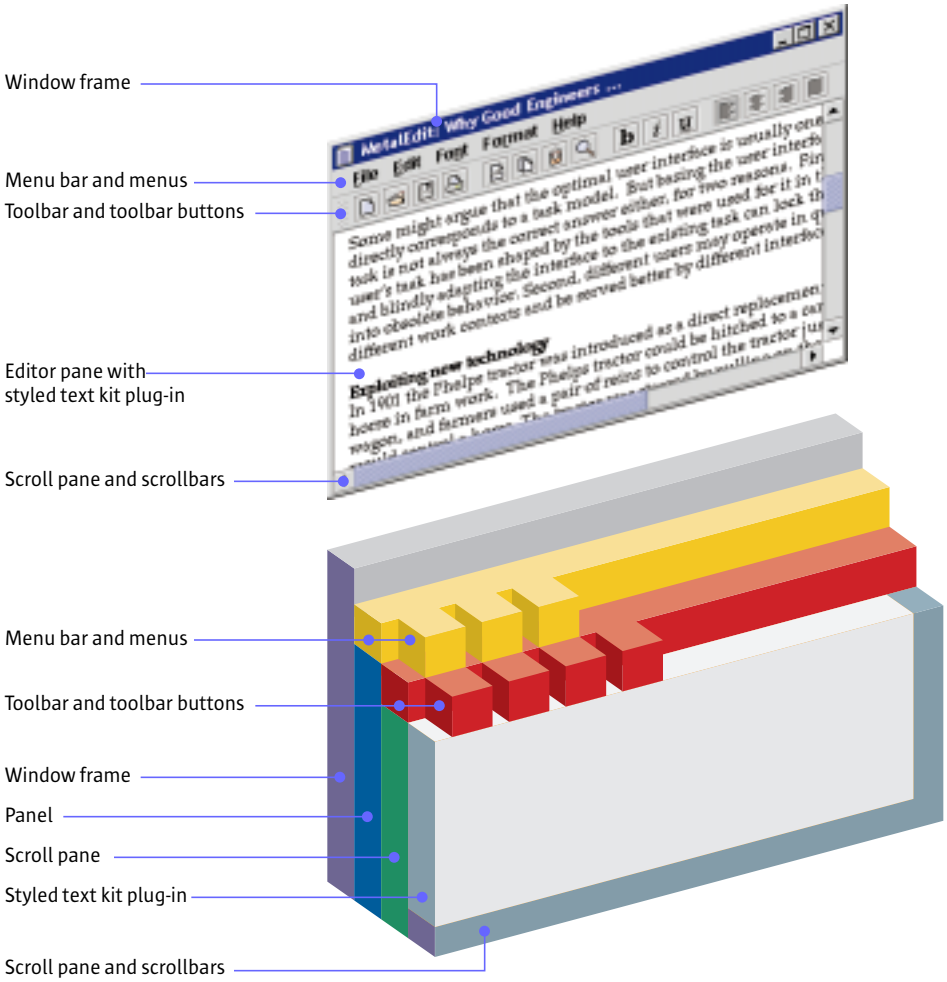
Similarly, you can use panels, panes, and internal frames as intermediate-level containers for the organization of primary and secondary windows. A **panel** is a container for organizing the contents of a window, dialog box, or applet. A **pane** is a collective term for scroll panes, split panes, and tabbed panes, which are described in this chapter. An **internal frame** is container used in MDI applications to create windows that a user cannot drag outside of the desktop pane.

Anatomy of a Primary Window

Primary windows act as top-level containers for user interface elements that appear inside them. A primary window might hold a series of embedded containers. For example, a primary window in your application might contain these parts, as shown in the following figure:

- The window frame contains a panel and a menu bar
- The menu bar contains menus
- The panel contains a toolbar and an editor pane
- The toolbar contains toolbar buttons
- The scroll pane contains an editor pane with a styled text kit plug-in

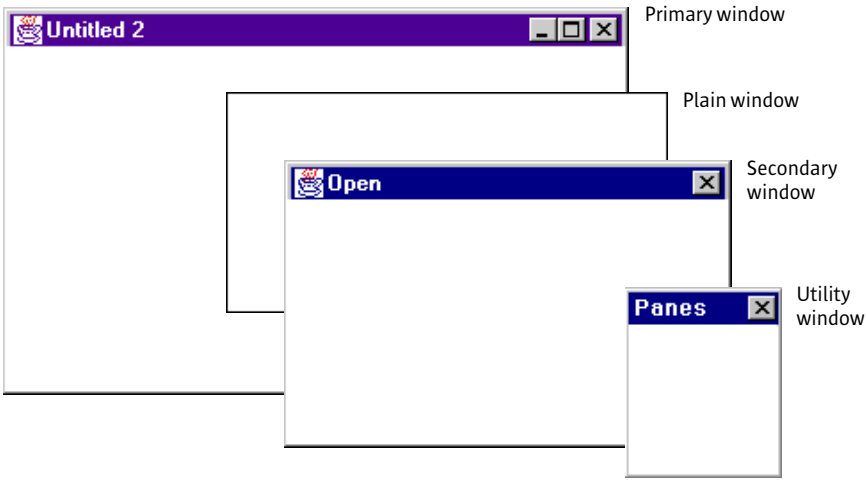
FIGURE 54 Anatomy of a Primary Window <<needs to be corrected according to Don's new specifications>>



Constructing Windows

You place the interface elements of your application in these top-level containers: primary windows, plain windows, and secondary windows.

FIGURE 55 Top-Level Containers



Primary windows are implemented using the `JFrame` component. Secondary windows are implemented using the `JDialog` component. Plain windows are implemented using the `JWindow` component.

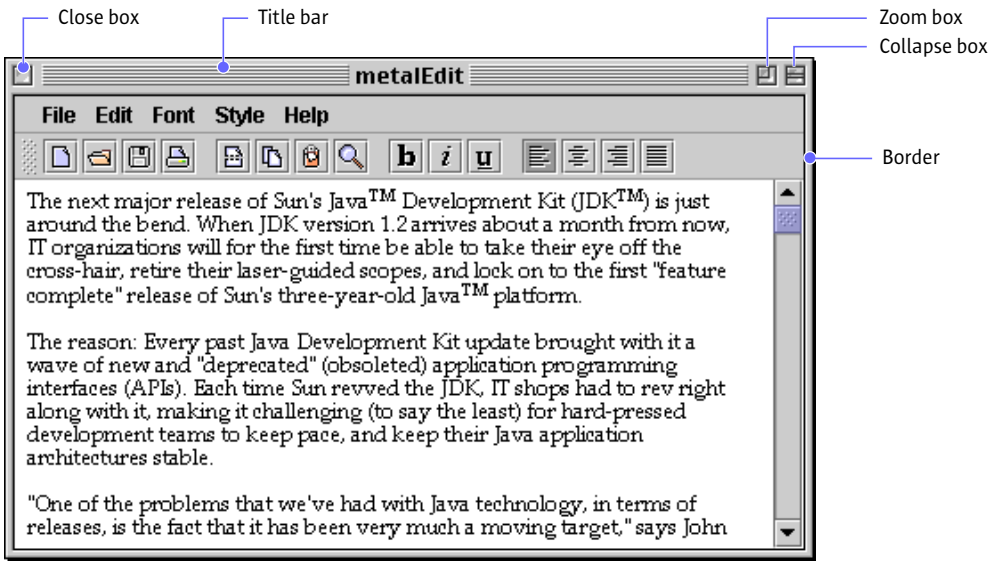
Primary Windows

Java applications display information such as documents inside primary windows. Such windows are provided by the native operating system of the platform on which the Java application is running, for instance—UNIX, Microsoft Windows, OS/2, or Macintosh.

Specifically, the window border and title bar, including the window controls, are provided by the native operating system. The content provided by your application assumes the Java look and feel, shown in the following illustration of a MetalEdit document window as it appears on the Microsoft Windows platform.

Window behavior, such as resizing, dragging, minimizing, positioning, and layering, is controlled by the native operating system.

FIGURE 56 Primary Window on Microsoft Windows Platform With Native Border, Title Bar, and Window Controls <<new illustration to be provided by Chris>>



Typically, when users close or minimize a window, the operating system closes any associated secondary windows as well. However, the operating system does not take care of this behavior automatically for JFC applications.

Keep track of the secondary windows in your application; close them if the primary window is closed or hide them if their primary window is minimized.

Although native operating systems display a Close control on the title bar of standard windows, also provide a Close item or Exit item in your File menu.

In JFC, primary windows are created using the `JFrame` component. This component appears with the border, title bar, outline, and window controls of the platform on which it is running, as shown in Figure 58. This is the JFC component you are most likely to use as the top-level container for a primary window.

Secondary Windows Secondary windows, such as dialog or alert boxes, are displayed in a window supplied by the native operating system. In JFC, this component is called `JDialog`. It appears with the border and title bar of the platform on which it is running. Chapter 8 provides more guidelines for the design of dialog boxes. The following figure shows a standard Warning alert box for the sample text-editing application, MetalEdit.

FIGURE 57 Alert Box With Macintosh Title Bar and Border

The `JOptionPane` component is used to implement an alert box. If what is provided does not suit the designer's needs, the `JDialog` component can be used.

Keep in mind that some platforms do not provide close controls in the title bar for dialog boxes. Always provide a way to close the window in the dialog box or alert box itself.

Plain Windows You can create a window that is a blank plain rectangle. The window contains no title bar or window controls. It does not provide dragging, closing, minimizing, and maximizing. A plain window is used as the container for the splash screen, which appears and disappears without user interaction in the following figure.

FIGURE 58 Plain Window Used as Basis for Splash Screen

The `JWindow` component is used to implement plain windows, whereas the `JFrame` component is used to implement primary windows.

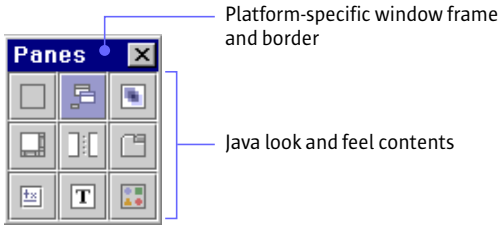
Utility Windows

In a non-MDI application with the Java look and feel, a utility window is a modeless dialog box that might display a collection of tools, colors, or patterns. Unlike palette windows, utility windows do not float above all the other windows.

User choices made in a utility window refer to and affect whatever window is active. Users work in utility windows the same way they work in secondary windows.

For information on keyboard operations for utility windows, see Table 17 on page 188.

FIGURE 59 Example of Utility Window



As opposed to secondary windows, which you close when their associated windows are closed, you do not close utility windows when primary windows are closed.

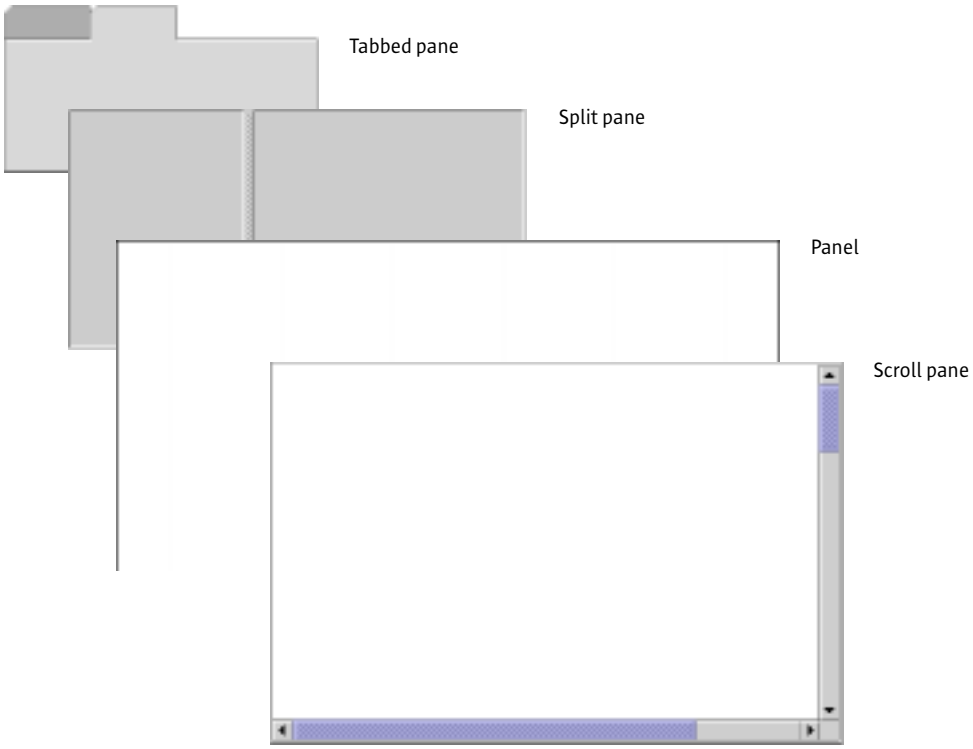
Since utility windows are not dependent on a primary window, do not automatically dismiss utility windows when primary windows are closed.

Utility windows in your application are implemented using the `JDialog` component. Palettes to be used within MDI applications are implemented as a form of the `JInternalFrame` component.

Organizing Windows

The JFC provides a number of user interface elements you can use as intermediate-level containers for the organization of windows: panels, tabbed panes, split panes, and scroll panes. Panels and panes can be used to structure windows into one, two, or more viewing areas. A panel is a JFC component that you can use for grouping other components inside windows. A pane is a collective term for scroll panes, split panes, and tabbed panes.

FIGURE 60 Intermediate-Level Containers



Panels

In contrast to other components such as scroll panes and tabbed panes, which typically play an interactive role in an application, a panel is an intermediate-level container that helps to simplify the positioning of basic components, such as buttons or labels.

Scroll Panes

A **scroll pane** is a specialized intermediate-level container that provides vertical and horizontal scrollbars that enable users to change the visible portion of the window contents.

Here is an example of a scroll pane with a vertical scrollbar. The size of the scroll box fills less than one fifth of the scrollbar, indicating that slightly less than one fifth of the current content is displayed.

FIGURE 61 Scroll Pane

You can choose whether scrollbars are always displayed in the scroll pane or if they appear only when needed.

Unless otherwise indicated, use the default setting for horizontal scrollbars, which specifies that they appear only when needed.

If the data in a list is known and will fit in the available space, such as a predetermined set of colors, do not specify a vertical scrollbar.

If the extent of data in a list is unknown and likely to expand, always display a vertical scrollbar.

If the data in a scroll pane sometimes requires a vertical scrollbar, specify that the vertical scroll bar be always present. Otherwise, the data must be reformatted whenever the vertical scroll bar appears and disappears.

Scrollbars are obtained by placing the component, such as a text area, inside a scroll pane.

Scrollbars A **scrollbar** is a component that enables users to control what portion of a document or list (or similar information) is visible on screen. Scrollbars appear along the bottom and the right sides of a scroll pane, a list, a combo box, a text area, or an editor pane. In locales with right to left writing systems, such as Hebrew and Arabic, scrollbars appear along the bottom and left sides of the relevant component. By default, scrollbars appear only when needed to view information that is not currently visible, although you can specify that the scrollbar is always present.

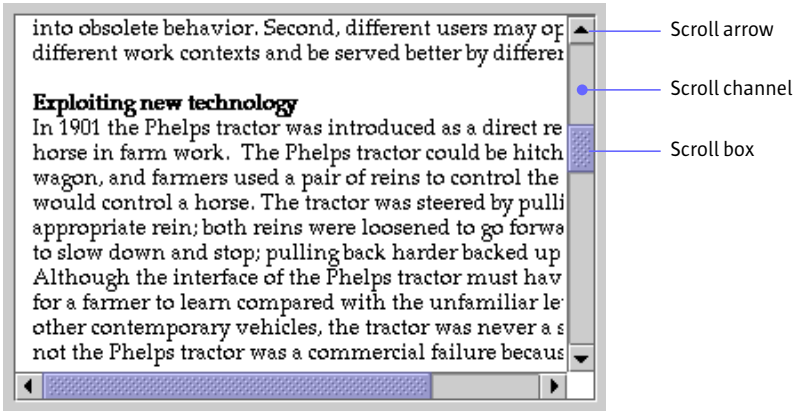
The size of the **scroll box** represents the proportion of the window content that is currently visible. The position of the scroll box within the scrollbar represents the position of the visible material within the document. As the user moves the scroll box, the view of the document changes accordingly.

The maximum size for scroll boxes is 16 pixels less than the length of the scroll channel, preserving a target for users to click within the channel in the event of very long documents or lists. (When users reach the actual end or beginning of such a document or list, the target area closes up.)

Both horizontal and vertical scroll boxes have a minimum size of 16 x 16 pixels so that users can still manipulate them when viewing very long documents or lists.

The following figure shows horizontal and vertical scrollbars in a simple text-editing application. Each scrollbar is a rectangle consisting of a textured scroll box and a recessed channel as well as scroll arrows.

FIGURE 62 Vertical and Horizontal Scrollbars



Do not confuse the scrollbar with a slider, which is used to select a value. For details on sliders, see "Sliders" on page 156.

At either end of the scrollbar is a **scroll arrow**, which is used for moving a small amount of data at a time. If the entire document is visible, the scroll bar fills the entire channel.

Users drag the scroll box, click the scroll arrows, or click the channel to change the contents of the viewing area. When the user clicks a scroll arrow, more of the document or list scrolls into view. The contents of the pane or list move in increments based on the type of data. When the user holds down the mouse button, the pane or list scrolls continuously after a short delay.

For a description of keyboard operations for scrollbars, see Table 22 on page 191.

 Scroll the content about one view at a time when the user clicks in the scrollbar's channel. For instance, in a document, a view might represent a page of text.

 Leave about one small unit of overlap from the previous view to provide context for the user. For instance, in scrolling through a long document, help the user become oriented to the new page by providing one line of text from the previous page.

 Scroll the content one small unit at a time when the user clicks a scroll arrow. (A small unit might be one line of text, one row in a table, or 10 to 20 pixels of a graphic.)

 Display a horizontal scrollbar only if the view cannot show everything that is important, for instance, in a word processing application that prepares printed pages where the user might want to look at the margins as well as the text.

 If you are using the Java 2 Platform, Second Edition of the JFC, place scrollbars in the orientation that is suitable for the writing system of your target locale. In the U.S. locale, the scrollbars appear along the right of the scroll pane or other component. In other locales, they might appear along the left of the scroll pane.

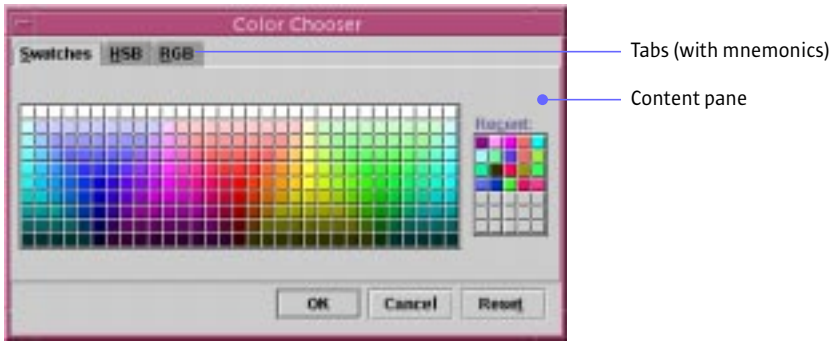
Tabbed Panes

A **tabbed pane** is an intermediate-level container that enables the user to switch between several components (usually `JPanel` components) that appear to share the same space on screen.

The tabs can contain text or images or both. The typical tabbed pane appears with tabs displayed at the top. Alternatively, the tabs can be displayed on one of the other three sides. If the tabs cannot fit in a single row, additional rows

are created. Note that tabs do not change position when they are selected. For the first row of tabs, there are no separator lines between the tab and pane.

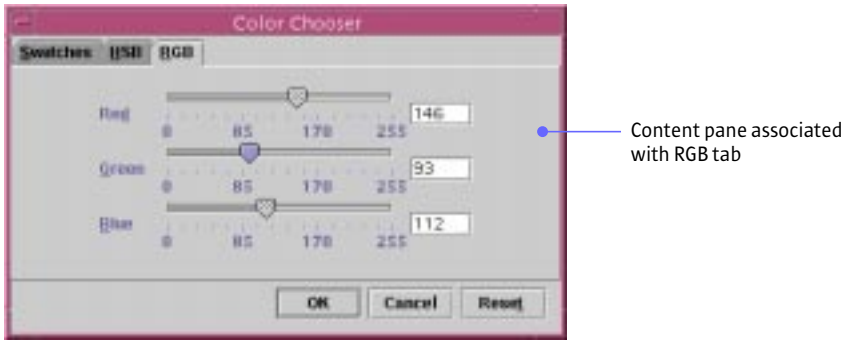
FIGURE 63 Tabbed Pane



The user chooses which information area to view by clicking the tab corresponding to it, and the content pane changes accordingly.

For a list of keyboard operations for tabbed panes, see Table 25 on page 192.

FIGURE 64 Tabbed Pane to Show Different View of Data



You can use tabbed panes to good advantage in dialog boxes, such as a Preferences dialog, when you want to fit a lot of information into a small area. In fact, tabbed panes provide an excellent alternative to multiple levels of submenus.

You can also use tabbed panes to provide a way for users to switch between information areas that represent

- different views of information like a color chooser or
- different parts of an informational unit like worksheets that are part of a workbook in a spreadsheet application.

Use headline capitalization for tab names.

Provide mnemonics so users can navigate from tab to tab using keyboard operations.

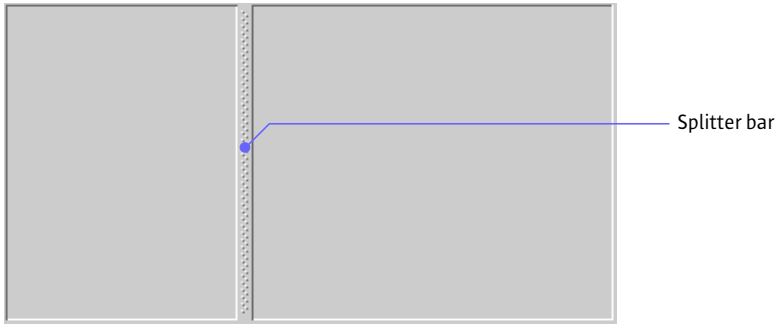
An `accesibleName` and `accessibleDescription` field should be provided for each icon so that assistive technologies can find out what it is.

Split Panes

A **split pane** is an intermediate-level container that divides an information area into resizable panes. Split panes enable users to adjust the relative sizes of two adjacent panes. The Java look and feel drag texture (along with a mouse pointer change) indicates that users can resize split panes.

To adjust the size of the split panes, users drag the splitter bar, as shown in the following figure.

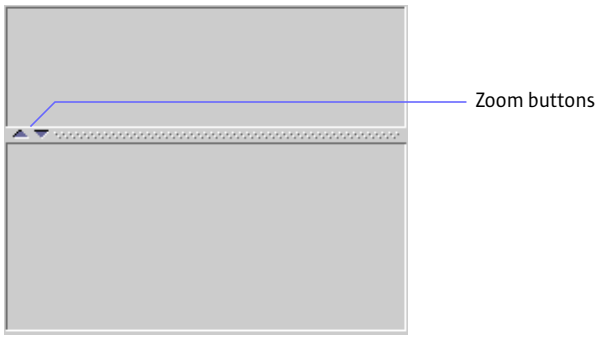
FIGURE 65 Horizontally Split Pane



Users can also control the splitter bar by clicking one of the optional zoom buttons shown in the preceding figure. Clicking a button moves the splitter bar to its extreme position. If the splitter bar is already at its extreme, clicking restores the panes to the size they were before the zoom operation (or before the user dragged the splitter bar to close one of the panes).

For a list of keyboard operations for split panes, see Table 24 on page 192.

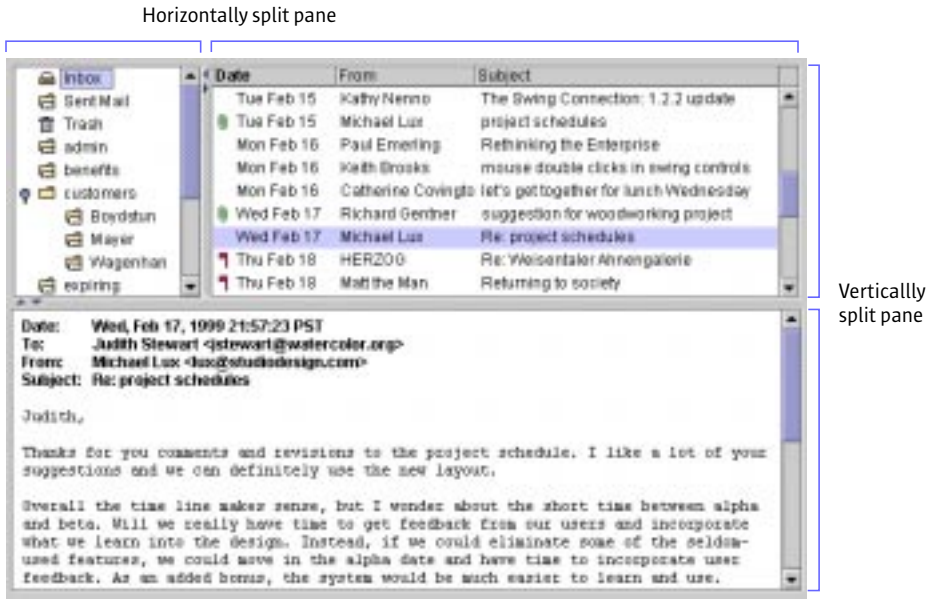
FIGURE 66 Zoom Buttons in Vertically Split Pane



Include zoom buttons in split panes because they are very convenient for users.

Nesting Split Panes In addition to splitting panes either horizontally or vertically, you can nest one split pane inside another. The following figure portrays a mail application in which the top pane of a split pane has another split pane embedded in it.

FIGURE 67 Nested Split Panes



Working With Multiple Document Interfaces

A multiple desktop interface (**MDI**) provides a way to manage multiple primary windows that are confined inside a main window. Among many disadvantages to using the MDI application model, users cannot drag the application's primary windows outside the main window.

To accommodate MDI designers, the JFC provides the internal frame, minimized internal frame, and palette window in the Java look and feel.

 Because users cannot move the application's primary windows outside the main window, avoid the use of MDI.

 If you are working with an MDI using the Java look and feel, the `JDialog` component can be used to create secondary windows.

Internal Frames

To get standard window features in an MDI, you must put an internal frame inside the desktop pane. A **desktop pane** is a utility component placed inside a window that holds internal frames for an MDI application.

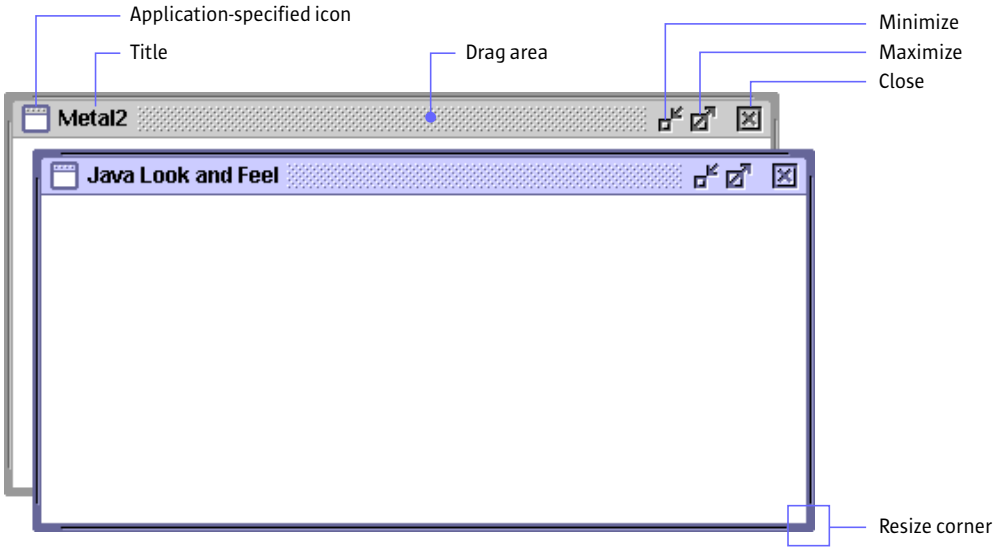
The **internal frame** is a container used in MDI applications to create windows that a user cannot drag outside of the desktop pane. In an MDI application that uses the Java look and feel, internal frames have a window border, title bar, and standard window controls with the Java look and feel, but the window that contains the desktop pane has the look and feel of the native platform.

Users can use the mouse to:

- Activate a window (and deactivate the previously activated window?) by clicking on the title bar
- Resize an internal frame from any side or corner
- Drag the internal frame by the title bar within the desktop pane
- Minimize, maximize, restore, and close the internal frame by clicking the appropriate window controls, shown in the following figure.

For details on keyboard navigation, selection, and activation in MDI applications that use internal frames, see Table 16 on page 187.

FIGURE 68 Internal Frames



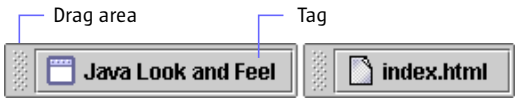
Note the Java look and feel texture in the title bar, which indicates it is draggable.

Since the internal frame is suitable only for documents in MDI-style applications, do not use it in applications where users need to drag secondary windows outside the main application window.

Minimized Internal Frames

A **minimized window** is a horizontally oriented component that represents an internal frame that has been minimized. These tags have a default left-to-right order. The width of these minimized internal frames is sized to accommodate the window name.

FIGURE 69 Minimized Internal Frames



Users rearrange them by dragging on the textured area, so that the tags snap to a grid. Users click a minimized internal frame to restore the window to full size.

For details on the keyboard operations for navigation and selection of minimized internal frames, see Appendix A.

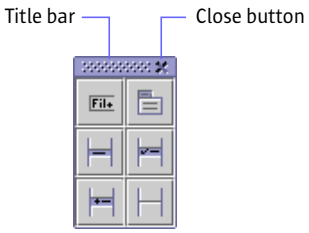
JFC applications should not use the desktop icon API, as it will go away in future versions of JFC when its behavior is moved into `JInternalFrame`.

Palettes

A **palette window** is a form of an internal frame that can float above other internal frames within the desktop pane for an MDI application.The close box is optional. Palette windows always float on top of the other primary windows and accept mouse clicks; in that sense, they always remain active.

The following figure shows a palette window for a graphical interface builder that lets users select menu components.

FIGURE 70 Palette Window



Users can click on items in a palette window to select them. They can also close palette windows, but cannot resize, minimize, or maximize them.

For keyboard operations for palette windows, see Table 16 on page 187.

Because it is confined to the main window, use this palette window only for MDI-style applications.

If you are writing a non-MDI application, create utility windows from modeless dialog boxes for palettes so that the user can drag them anywhere on the screen.

A palette window is a specific style of `JInternalFrame` and therefore can only be used within a desktop pane. The client properties mechanism can be used to set the palette style property as follows:
`"JInternalFrame.isPalette"`

8: DIALOG BOXES

This chapter provides information and guidelines on the modes and layout of dialog boxes. This chapter also describes and makes recommendations about the use of the alert boxes and choosers built into the JFC.

A **dialog box** is a temporary, secondary window in which users perform a task that is peripheral to the task in the primary window. For example, a dialog box might enable users to set preferences or choose a file from the hard disk. A dialog box may contain text, graphics, controls (such as checkboxes, radio buttons, and sliders), and one or more command buttons. Dialog boxes use the native window frame of the platform on which they are running.

An **alert box** is a type of dialog box that is used for brief interaction with users. Alert boxes present error messages, warn of potentially harmful actions, obtain information from users, and display informational messages. A basic alert box contains a graphic that identifies the type of the alert, a textual message, and one or more command buttons. The layout of a basic alert box is prebuilt in the Java look and feel, so you don't have to worry about the spacing and alignment of the components.

A chooser is a component that asks users to select from available options. The JFC includes a **color chooser**, which enables users to pick a color and then look at that color in a preview panel. Although you can place the color chooser anywhere in your application, you will typically place it in a dialog box.

Applications based on a multiple document interface (MDI) should use the dialog boxes, alert boxes, and choosers presented in this chapter. Because these secondary windows use the platform's native windows (and not the Java internal frame), they are free to move outside the desktop pane.

Modal and Modeless Dialog Boxes

Dialog boxes can be modal or modeless. Modal dialog boxes block users from interacting with the application until the dialog box is dismissed. However, users can move the modal dialog box and interact with other applications while the modal dialog box is open. This behavior is sometimes called "application-modal."

Modeless dialog boxes do not block users from interacting with either the application they are in or any other application. Modeless dialog boxes behave similarly to utility windows. Modeless dialog boxes, however, are associated with a particular primary window and utility windows are not.

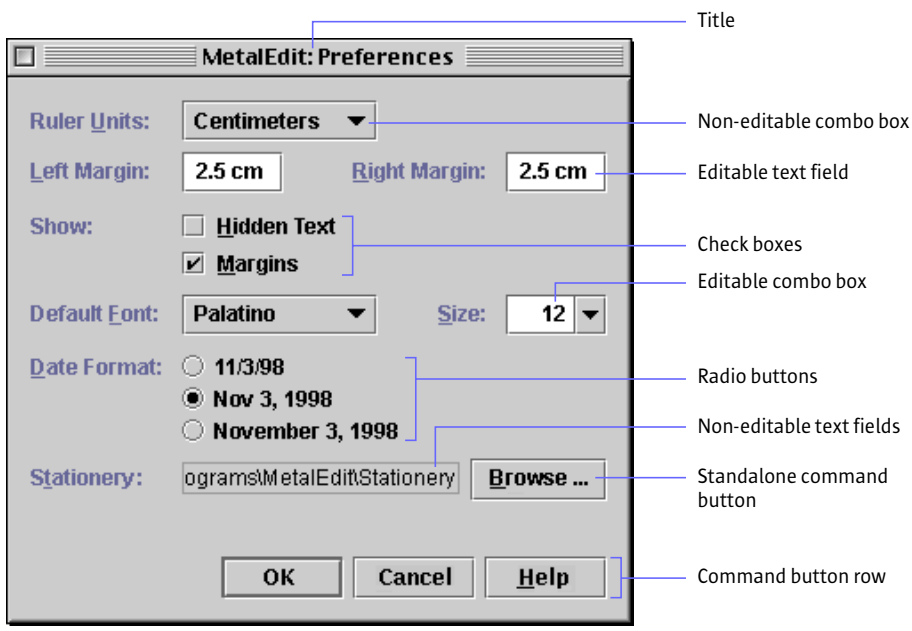
Use modeless dialog boxes whenever possible. Users might want to do tasks in different orders or check information in other windows before completing the dialog box. Users might also want to go back and forth between the dialog box and the primary window.

Use modal dialog boxes when interaction with the application cannot proceed while the dialog box is displayed. For example, make a progress dialog box that appears while the application is loading its data a modal dialog box.

Dialog Box Design

The following dialog box illustrates the design guidelines for the Java look and feel. The dialog box has a title in the window title bar, a series of components with which users can interact, and a row of command buttons.

FIGURE 71 Sample Dialog Box



Use the form “ApplicationName: Title” for the title of the dialog box (which is displayed in the title bar).

When opening a dialog box, provide initial keyboard focus to the control that you expect users to operate first. This focus is especially important for users who navigate through applications using only the keyboard.

Include mnemonics for all controls that users interact with, with the exceptions of the default button and the Cancel button.

Specify a logical traversal order for tab elements. Make sure that the traversal order matches the reading order for the locale you are in. For example, in English, the traversal order is left to right, top to bottom. By default, the traversal order is the order in which you added the components to the dialog box.

Consider the effect of internationalization on your design. Use a layout manager, which allows for text strings to expand when translated to another language. For guidelines, see “Internationalization and Localization” on page 29.

For keyboard support for navigating through dialog boxes, see Table 17 on page 188.

Spacing and Alignment

The following figure shows the recommended spacing between the borders of the dialog box and the components in the dialog box.

FIGURE 72 Spacing Between Dialog Border and Components

Include 12 pixels between the top and left borders of the dialog box and the components. Include 11 pixels between the bottom and right borders of the dialog box and the components.

Align command buttons along the lower-right edge of the dialog box (with the exception of an alert box, where command buttons are aligned with the message text. The spacing and alignment of the command buttons in alert boxes is prebuilt in the Java look and feel.)

See "Design Grids" on page 45 for a general description of how to place components and text in a dialog box. See Chapter 10 for guidelines on laying out buttons, combo boxes, sliders, and progress bars. See Chapter 11 for guidelines on labels, text fields, text areas, and editor panes. See Chapter 12 for guidelines on lists, tables, and tree views.

Command Buttons When clicked, a command button initiates an action. Command buttons can appear alone or in groups, and they are aligned along the lower-right edge of the dialog box (except in an alert box, where they are automatically aligned with the message text.) Command buttons commonly found in dialog boxes are Help, Close, OK, Cancel, Apply, and Reset.

For consistency in the look and spacing of command buttons, follow the guidelines in "Command Buttons" on page 142. For keyboard operations appropriate to command buttons, see Table 15 on page 186.

Help Button The Help button displays help information in a separate secondary window. It can be used in any dialog box. For example, in an Error alert box, the Help button can provide additional information on the cause of the error.

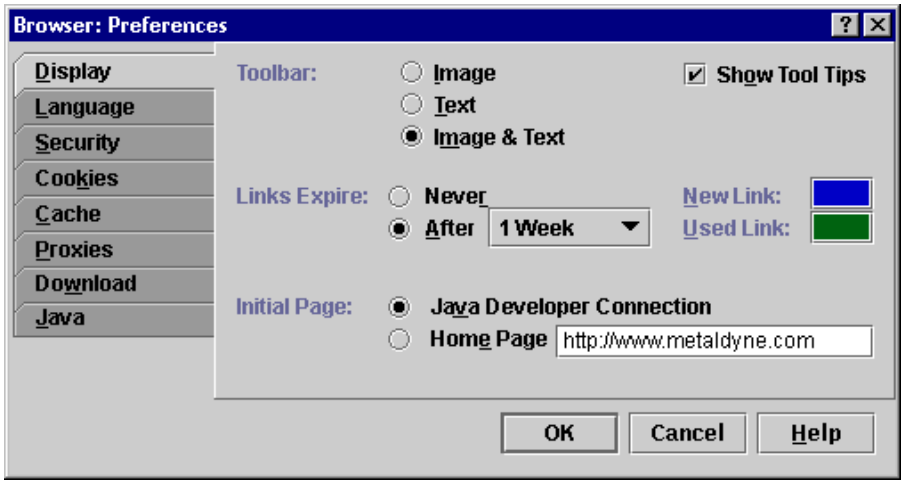
Place the Help button last in a group of command buttons.For languages that are read left to right, the default button is the rightmost button.

Close Button The Close button dismisses the dialog box and does not make additional changes to the system. The Close button is often used in simple dialog boxes, such as the Info alert box. The Close button is also used in dialog boxes in which user actions take effect immediately (that is, users do not need to press an OK button for the settings to take effect). The following dialog box, which contains a schedule reminder, includes a Close button.

FIGURE 73 Dialog Box With Close Button

OK and Cancel Buttons The OK and Cancel buttons are common in dialog boxes in which users specify parameters. The OK button sets parameters or carries out the commands in the dialog box and closes the dialog box. The Cancel button closes the dialog box and restores the system to the state it was in when the dialog box was opened. The following Browser Preferences dialog box includes OK, Cancel, and Help buttons.

FIGURE 74 Dialog Box With OK, Cancel, and Help Buttons



Where reasonable, use specific labels such as Print, Open, or Find, which describe the action they perform, instead of OK.

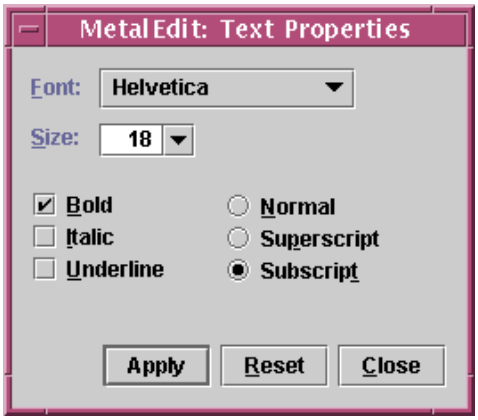
If the dialog box has a Cancel button, activate it when the user presses the Escape key. The Cancel button does not need keyboard focus for this interaction; only the dialog box must have focus. This functionality is not built into the JFC, and you must implement it yourself.

Do not add a mnemonic for the Cancel button.

Do not use Cancel in the same in the same dialog box as a command whose changes are persistent, such as Apply. In cases where you want users to be able to view and then cancel changes, use Preview and Cancel buttons.

Apply and Reset Buttons The Apply and Reset buttons are common in dialog boxes that remain open for repeated use. The Apply button carries out the changes users specify in the dialog box, but does not close the dialog box. The Reset button restores the values in the dialog box to the values specified at the last Apply command or when the dialog box was opened. Reset does not close the dialog box. The following Text Properties dialog box includes Apply, Reset, and Close buttons.

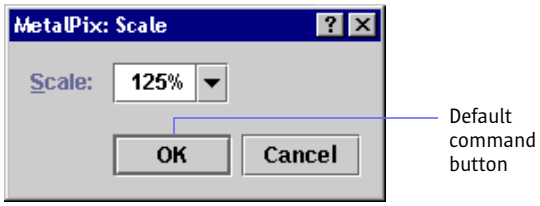
FIGURE 75 Dialog Box With Apply, Reset, and Close Buttons



Use Apply, Reset, and Close buttons in modeless dialog boxes.

Default Command Buttons The default command button is the button that the application activates if a user presses Enter or Return. The default button represents the action that the user is most likely to perform and is identified with a heavier border than the other command buttons. The default button does not need to have keyboard focus when the user presses Enter or Return. In the following figure, the default command button is OK.

FIGURE 76 Dialog Box with Default Command Button



In cases where keyboard focus is on a component that accepts the Enter or Return key, such as a text field, the default button is not activated when users press the key. Instead, the insertion point moves to the beginning of a new line. To operate the default button, users must move focus to a component that does not accept Enter or Return.

Make the default button the first command button in the group. For languages that are read left to right, the default button is the leftmost button.

Do not add a mnemonic for the default command button.

You are not required to have a default command button in every dialog box. A command that might cause users to lose data is never the default button, even if it is the action that users are most likely to perform. The following dialog box asks users if they want to replace an existing file. The dialog box has Replace and Cancel buttons, neither of which is the default command button.

FIGURE 77 Dialog Box Without Default Button

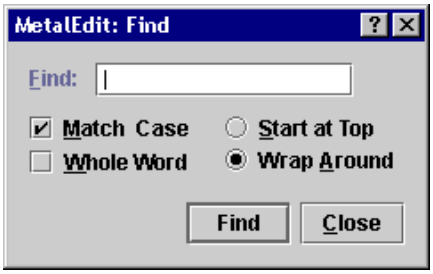
Common Dialog Boxes

The find, login, preferences, print, and progress dialog boxes are common in many applications. These dialog boxes are not prebuilt in the Java look and feel. The following sections show simple versions of these dialog boxes that are consistent with the standard Java look and feel dialog boxes. You can adapt these dialog boxes to suit your needs.

Find Dialog Boxes

A find dialog box enables users to search for a specified text string. An example is shown in the following figure.

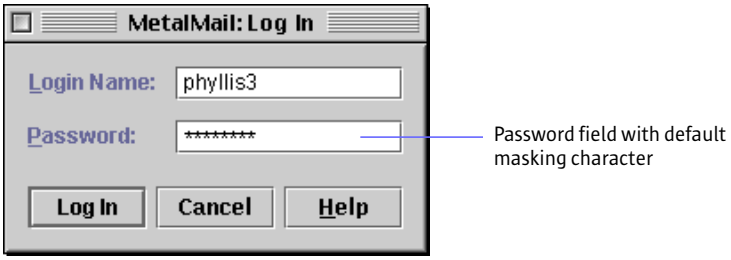
FIGURE 78 Sample Find Dialog Box



Login Dialog Boxes

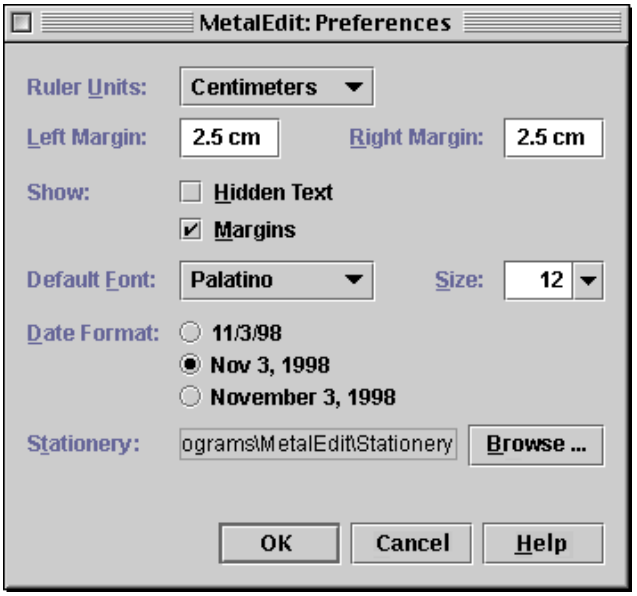
A login dialog box (shown in the following figure) requires users to identify themselves and enter a password each time they start the application. By default, the character echoed in the password field is an asterisk. You can change the asterisk to another ASCII character.

FIGURE 79 Sample Login Dialog Box



Preferences Dialog Boxes A preferences dialog box (shown in the following figure) enables users to view and modify the characteristics of an application.

FIGURE 80 Sample Preferences Dialog Box



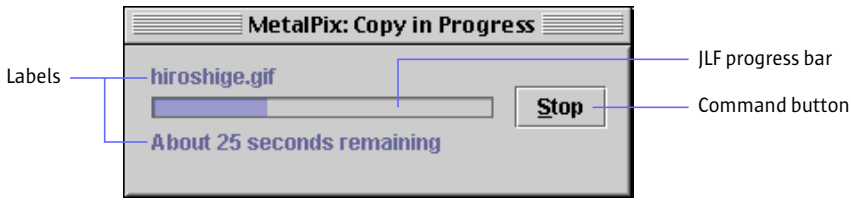
If your preferences dialog box is very complex, consider using a tabbed pane to organize the options, as shown in Figure 74 on page 113.

Print Dialog Boxes A print dialog box enables users to determine how to print a document (such as the number of copies and which printer to use) and then print the document.

Use the print dialog box available from the AWT. On Microsoft Windows and Macintosh platforms, the print dialog box is the native print dialog box. For other environments, the print dialog box is supplied with the JDK.

Progress Dialog Boxes A progress dialog box provides feedback for long operations and lets users know that the system is working on the previous command. A progress dialog box normally includes the Java look and feel progress bar and a command button that users can click to stop the process. The following progress dialog box monitors the progress of a file copy operation.

FIGURE 81 Sample Progress Dialog Box



Display a progress dialog box (or supply a progress bar elsewhere in your application) if an operation takes longer than two seconds.

Include labels inside a progress dialog box to further explain the operation. Use sentence capitalization for the labels.

In the progress dialog box, place the command button, which interrupts the process, to the right of the progress bar. If the state will remain as it was before the process started, use a Cancel button. If the process might alter the state as it progresses (for example, deleted records will not be restored), use a Stop button. Give users a chance to confirm the Stop command if stopping the process could lead to further data loss.

Close the progress dialog box automatically when the operation is complete.

If delays are a common occurrence in your application (for example, in a web browser), build a progress bar into the main application window so that you don't have to keep displaying a progress dialog box.

Send the specialized meaning of the Stop button in the progress dialog box to your translators. The Stop button indicates the process might alter the state.

Setting the `labelFor` property enables screen readers to associate a name with the progress bar when you include labels in the progress dialog box.

Alert Boxes

An alert box, which conveys a message or warning to users, provides an easy way for you to create a dialog box. The Java look and feel offers four types of alert boxes: Info, Warning, Error, and Question. An alert box can be modal or modeless.

Each alert box is provided with a graphic that indicates its type. You provide the title, the message, and the command buttons and their labels. The layout of an alert box is provided in the Java look and feel, so you don't have to

worry about the spacing and alignment of the standard components. If you want to provide additional components, such as a text field, follow the layout guidelines for that component.

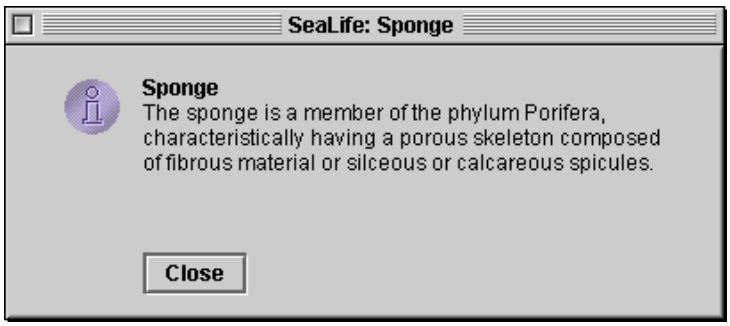
An alert box is created using the `JOptionPane` component.

Info Alert Boxes

An Info alert box presents general information to users. For example, an Info alert box might indicate that a disk is being checked for unnecessary files. Users can dismiss the alert box and optionally open a separate online help window. The graphic in the Info alert box is a blue circle with the letter “i”.

The following Info alert box from an encyclopedia application provides information about a sponge. It includes a Close button.

FIGURE 82 Info Alert Box



Use Close as the label for the command button that dismisses an Info alert box.

Warning Alert Boxes

A Warning alert box warns users about the possible consequences of an action and asks for a response. For example, a Warning alert box might ask, “Do you want to save this data?” Users can perform the action that might cause data loss, cancel the action, or optionally open an online help window. The graphic in the Warning alert box is a yellow triangle with an exclamation point.

The following alert box warns that a file save operation will replace an existing file. It includes Replace and Cancel buttons.

FIGURE 83 Warning Alert Box

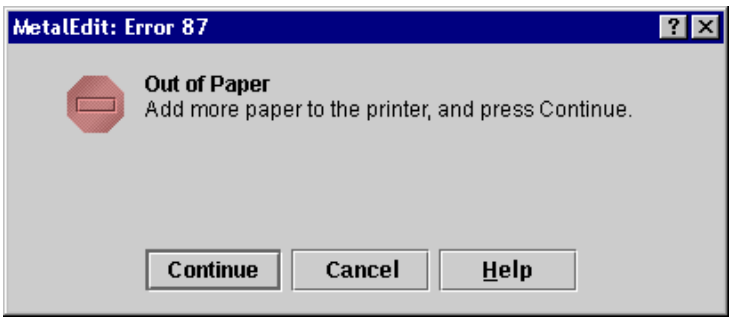
- Include at least two buttons in a Warning alert box: one button to perform the action and the other to cancel the action. Provide the command buttons with labels that describe the action they perform.
- In a Warning alert box, do not make a command button whose action might cause loss of data the default button. Users might automatically press the Enter or Return key without reading the message.

Error Alert Boxes

An Error alert box reports system and application errors to users. Users can resolve the error, dismiss the alert box, or optionally open a separate online help window. The graphic in the Error alert box is a red octagon with a rectangle.

The following Error alert box reports that a printer is out of paper. The alert box includes three command buttons: Continue, for restarting the printer; Cancel, for dismissing the dialog box; and Help, for opening a separate online help window.

FIGURE 84 Error Alert Box



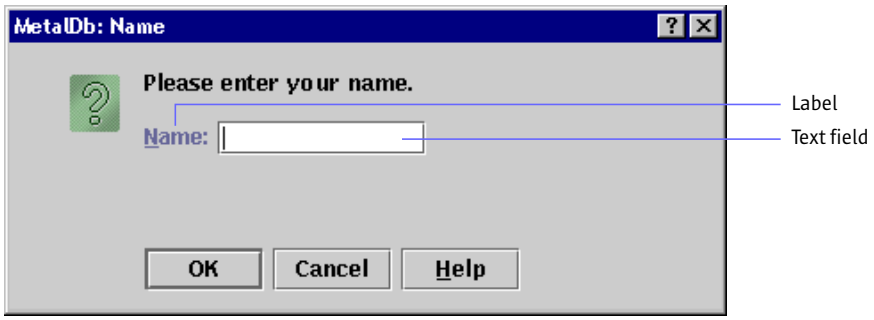
- If possible, provide buttons or other controls to resolve the error noted in an Error alert box. Label the buttons according to the action they perform. Otherwise, provide a Close button.
- In the message of an Error alert box, explain what happened, the cause of the problem, and what the user can do about it. Keep the message brief and use terms familiar to the user. If appropriate, provide a Help button to open an online help window that gives background information about the error.
- Include an error number in the title bar of an Error alert box. The error number is helpful for users in obtaining technical assistance, especially if the error message is localized in another language.

Question Alert Boxes

A Question alert box requests information from users. It often includes one or more components (for example, a text field, list, or combo box) in which users can type a value or make a selection. The graphic in the Question alert box is a green rectangle with a question mark.

The following figure shows a Question alert box with a text field and OK, Cancel, and Help buttons.

FIGURE 85 Question Alert Box



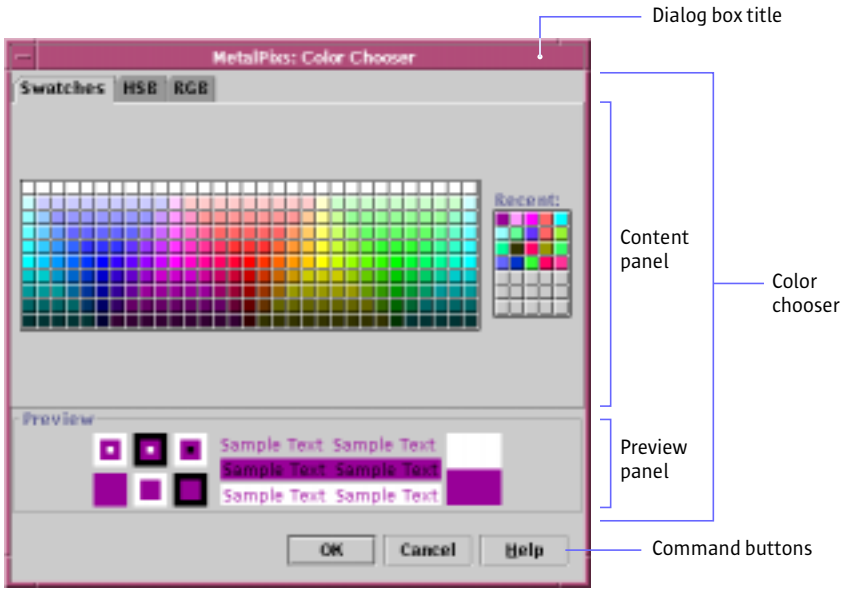
In the Question alert box, the layout of the standard components (the graphic, message, and command buttons) is provided in the Java look and feel. When providing additional components in a Question alert box, such as the label and text field in the preceding figure, follow the layout guidelines for that component. See the individual component sections for details.

Color Choosers

A **color chooser** displays the colors available in the application and includes controls that enable users to define custom colors. A preview panel at the bottom of the chooser illustrates the selected color in context.

A color chooser is commonly displayed in a dialog box, as shown in the following figure. The three command buttons (OK, Cancel, and Reset) are part of the dialog box, not the color chooser.

FIGURE 86 Standard Color Chooser



Users pick a color in the color chooser using one of three modes:

- **Swatches.** Users pick a color by selecting it in a palette on the left. The palette on the right shows the recently selected colors.
- **HSB**. Users click a radio button to choose hue, saturation, or brightness, and then drag a slider to set the value.
- **RGB**. Users drag sliders to select the amounts of red, green, and blue.

You can customize the color chooser by providing your own preview panel and by adding and removing color panels. If you use only one color panel, the tab disappears.

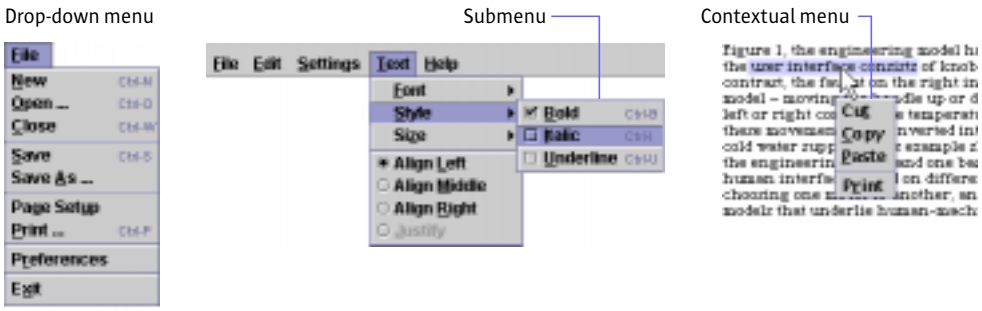
The color chooser is a panel. To use it as a color chooser dialog box, insert it in a `JDialog` container.

9: MENUS AND TOOLBARS

A **menu** displays a list of choices (menu items), logically grouped and displayed by an application so that a user need not memorize all available commands or options. Menus in the Java look and feel are "sticky"—that is, they remain posted on screen after the user clicks the menu title. Usually the primary means to access your application's features, menus also provide a quick way for users to get an overview of your application's features.

A **drop-down menu** is a menu whose titles appear in the menu bar. A **submenu** appears to the right of a menu item in a drop-down menu. An arrow next to the item indicates that more choices are available in a submenu. A **contextual menu** displays lists of commands, settings, or attributes that apply to the item or selected items under the pointer.

FIGURE 87 Types of Menus

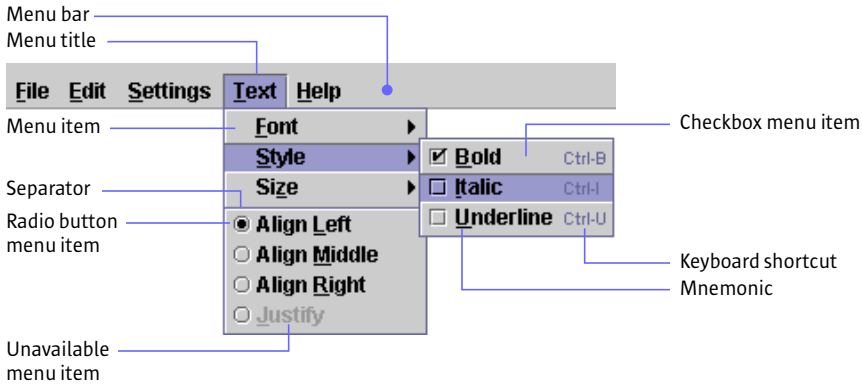


Menu Elements

In the Java look and feel, menus use a highlight color (primary 2) for selected menu titles, menu items, and submenus. In the following figure, the Text menu is selected and displayed. Within the Text menu, the Style item is selected; a submenu appears that includes the Bold, and Italic, and Underline checkbox menu items. (The Italic checkbox menu item is highlighted.)

A separator divides the alignment radio button items from the menu items for specifying font, style, and size. Keyboard shortcuts appear to the right of the frequently used menu items, and mnemonics are included for each menu title and menu item.

FIGURE 88 Menu Elements <<callouts need to be reapplied>>



Menu Bars

The **menu bar** appears at the top of a primary window and contains menu titles, which describe the content of each menu. Menu titles usually appear as text; however, it is possible to use a graphic or a graphic with text as a menu title. Menu titles in the Java look and feel contain mnemonics only if they are explicitly set. See "Mnemonics" on page 86 for details

Display the menu bar as a single line across the top of all primary windows, just below the title bar.

Do not display menu bars in secondary windows unless you have a compelling reason to do so.

Even on Macintosh systems, which ordinarily place a menu bar only at the top of the screen, display menu bars in windows in your Java look and feel application. On the Macintosh, the screen-top menu bar remains, but, since all the application menus are in the windows, the only command in the screen-top menu bar is Quit in the File menu.

Be sure to include mnemonics for every menu title in your menu bar.

If your applet runs in the main browser window (with the browser menu bar), do not display your own menu bar in the applet. Although applets running inside a browser window can theoretically have their own menu bars, users are often confused when both the browser window and the applet have menu bars. If your applet requires a menu bar, open a separate browser window without a menu bar or navigation controls.

Drop-down Menus

A **drop-down menu** appears when users choose a menu title in the menu bar. The menu bar contains all of the menus. Each menu in the menu bar is represented by its menu title. The titles describe the content of each menu. (The title for a submenu is its menu item in the drop-down menu.)

When users click a title, it appears highlighted, and the menu is displayed below it, left-aligned with the menu title.

- To post a menu (that is, to display it and have it stay up until the next click even though the mouse button has been released), users click the menu title. Users can then move the pointer over other menu titles to view other menus.
- To pull down a menu, users press the mouse button over the menu title. The menu title appears highlighted, and the menu drops down.

For details on keyboard navigation, selection, and activation in menus, see Table 20 on page 190.

 If possible, use single words for your menu titles.

 In your menu titles, describe the kinds of commands and settings that appear in the menu items to help users guess which menu contains the item of particular interest at a given moment.

 Include mnemonics in all your menu items.

 Any mnemonic character in your menu will only be underlined if it is explicitly set as the mnemonic with `setMnemonic`.

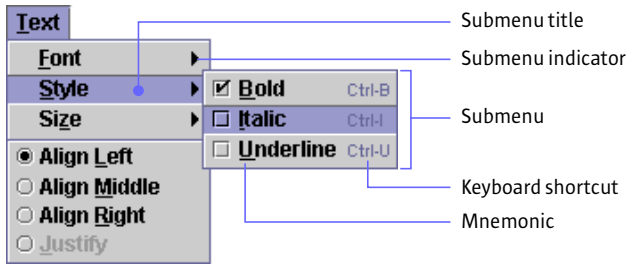
Submenus

A **submenu** is a menu that users open by choosing a menu item in a higher-level menu. Sometimes you can shorten a menu by moving related choices to a submenu. Submenus (such as the Style submenu shown in the following figure) typically appear to the right of the submenu indicator. For instance, the Style item opens a submenu consisting of three items: Bold, Italic, and Underline. Note that the items in the Style submenu include both keyboard shortcuts and mnemonics.

Users display submenus by clicking or by dragging the pointer over the corresponding menu item. The first item in the submenu aligns with the submenu indicator, slightly overlapping the main menu. Just as in other menus, items in the submenu are highlighted when the user moves the pointer through them.

For a list of keyboard operations in submenus, see Table 20 on page 190.

FIGURE 89 Menu Item With Submenu <<callouts will be reapplied>>



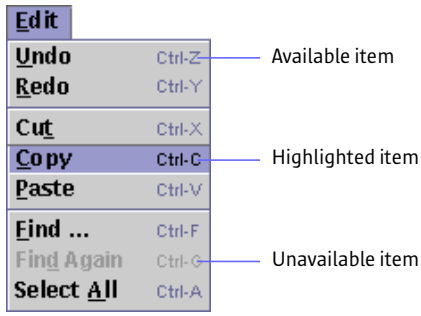
Since many people find submenus difficult to use, avoid the use of a second level of submenus. If you want to present a large or complex set of choices, display them in a dialog box.

Submenus are created using the JMenu component.

Menu Items

A simple menu item consists of the command name, such as Undo. When a menu item is available for use, it appears black.

FIGURE 90 Typical Menu Items



When the user positions the pointer over an individual item within a menu, the menu item appears highlighted.

- Within a posted menu, the user clicks a menu item to choose it and to close the menu.
- Within a pulled down menu, the user drags over a menu item to highlight it. To choose the command and close the menu, the user then releases the mouse button.

For a list of keyboard operations for menu items, see Table 20 on page 190.

Make your menu items brief, and confine them to a single line.

If a menu item is not currently available in this window, but the user can do something to make it available, disable the menu item and dim its text.

If all the items in a menu are disabled, do not disable the menu. In this way, users can still pull down and post the menu and view all its (disabled) items. Similarly, if all the items in a menu item's submenu are currently not available, do not disable the original menu item. In this way, users can display the submenu and view all its (disabled) items.

If there is nothing users can do to make a menu item available, omit the item entirely rather than just disabling it. Disabling an item implies that the user can do something to make the item available. A similar rule applies to submenu items and contextual menus.

Use headline capitalization in menu titles and menu items.

Include mnemonics for all menu items.

Offer keyboard shortcuts for frequently used menu items.

If a menu item appears in several menus (for instance, if a Cut command appears in a contextual menu as well as in a drop-down Edit menu), use the same shortcut (in this case, Control-X).

When a menu item does not fully specify a command and the user needs a dialog box to finish the specification, use an ellipsis (...) after the menu item. For example, after choosing the Save As... item, users are presented with a file chooser to specify a file name and location.

<<Internationalization issue: Ellipses don't exist in many cultures as a concept. Don is investigating what do say about this.>>

Do not use an ellipsis simply to indicate that a secondary or utility window is to appear. For example, choosing the Preferences item displays a dialog box; however, because that display is the entire effect of the command, Preferences is not followed by an ellipsis.

Use separators to group similar menu items to help users find desired items and better understand their range of choices. For instance, in a typical File menu, the commands that affect saving are separated from those that are relevant to printing.

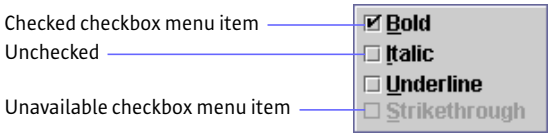
If a menu has the potential to become very long (for instance in menus that present lists of bookmarks or email recipients), a grid layout should be used in the menu to display the menu choices in multiple columns.

Typical keyboard shortcuts are described in "Typical File Menu" on page 130, "Typical Edit Menu" on page 131, and "Typical Help Menu" on page 132.

Checkbox Menu Items

A **checkbox menu item** is a menu item that appears with a checkbox next to it to represent an on or off setting. A check mark in the adjacent checkbox graphic indicates that the value associated with that menu item is selected. A dimmed checkbox menu item shows a gray box to indicate that the setting cannot be changed.

FIGURE 91 Checkbox Menu Items



For a list of keyboard operations for checkbox menu items, see Table 20 on page 190.

You can use checkbox menu items to present users with a nonexclusive choice.

You can also accommodate mixed states in checkbox menu items. See "Checkboxes" on page 148 for details on mixed states in checkboxes.

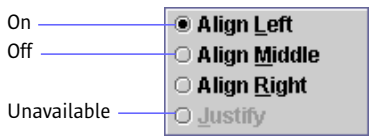
For consistency, use the standard checkbox graphic for checkbox menu items.

As with all menu items, users choose a checkbox menu item, the menu disappears. To choose another item, users redisplay the menu. Because users must redisplay checkbox menu items every time they want to make a choice, use this component with restraint. If the user must set more than one or two related preferences, provide a menu item that displays a dialog box containing the checkboxes (or provide a palette or toolbar buttons for the preferences).

Radio Button Menu Items

A **radio button menu item** is a menu item that appears with a radio button next to it. Each radio button menu item offers the user a single choice within a set of radio button menu items, as illustrated in the following set of alignment options.

FIGURE 92 Radio Button Menu Items



For a list of keyboard operations for radio button menu items, see Table 20 on page 190.

You can use separators to indicate which radio button menu items are in a group.

You can accommodate mixed states in radio button groups See “Radio Buttons” on page 151 for details on mixed states in radio buttons.

To indicate that the radio button items are part of a set, group them and use separators to distinguish them from other menu items.

When a user chooses a radio button menu item, the menu is dismissed. To choose another menu item, the user must redisplay the menu. Because users must redisplay radio button menu items every time they want to make a choice, use this component with restraint. If users must set more than one or two related preferences, provide a menu item that displays a dialog box containing the radio buttons (or provide a palette or toolbar buttons for the preferences).

Separators

A separator is a line graphic that divides menu items into logical groupings, as shown in Figure 90.

Users can never choose a separator.

You can use separators help break up lengthy menus to make them easier to read.

While separators serve important functions on menus, avoid using them elsewhere in your application. Instead, use white space or an occasional titled pane to delineate areas in dialog boxes or other components.

Common Menus

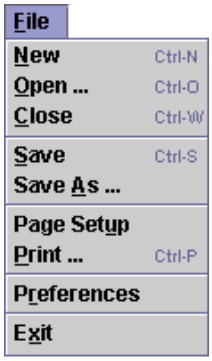
Several drop-down menus, such as File, Edit, and Help occur in many applications.

If your application needs these commonly used menus, place the menu titles in this order: File (leftmost, left aligned), Object, Edit, Format, View, and Help (rightmost, not right aligned). If needed, insert other menus between the View and Help menus.

Typical File Menu The leftmost menu displays commands that apply to an entire document or application as a whole. Typically, this is called the File menu, but in some cases another title might be more appropriate. The following figure illustrates common File (leftmost) menu items (in the typical order) with mnemonics and keyboard shortcuts.

You can adjust the menu items and their order as needed.

FIGURE 93 Typical File Menu



☕ Place commands that apply to the document, other object, or application as a whole in the File menu.

☕ If your application manipulates objects that your users might not think of as “files,” give the leftmost menu another name. Ensure that the name corresponds to the type of object or procedure represented by the entire window in your application. For example, a project management application could call this leftmost menu “Project,” or a mailbox application could call it “Mailbox.”

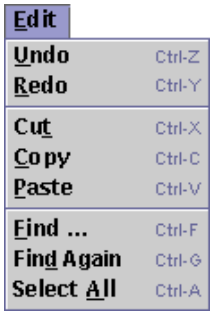
☕ Since the Close item dismisses the active window, close any dependent windows at the same time. The corresponding application or object might remain active if that makes sense in the user’s environment. For example, users might close the primary window of an email application, but the application would remain active to check the mail server periodically for new mail.

☕ If you provide an Exit item, have it terminate the application and close all associated windows. (Be sure to use the term Exit, not Quit.)

Typical Object Menu Object menu items provide actions that users can perform on a selected object or objects. An object might be almost anything— for instance, an icon representing a person for whom you want to add an email alias.

Typical Edit Menu The Edit menu displays items that enable users to change, or edit, the contents of their documents or other data files. These items give users the typical text editing features users have come to expect. The following figure shows common Edit menu items (in the typical order) with their mnemonics and keyboard shortcuts.

FIGURE 94 Typical Edit Menu

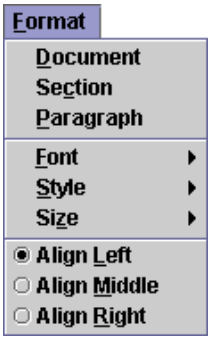


Place commands that modify the contents of files in the Edit menu, including Undo, Redo, Cut, Copy, Paste, and Find.

The Swing Undo package can be used to provide the Undo and Redo commands.

Typical Format Menu The Format menu displays items that enable users to change such formatting elements of their documents as font, size, styles, characters, and paragraphs. The following figure shows common Format menu items with their mnemonics.

FIGURE 95 Typical Format Menu



Typical View Menu View menu items provide ways for users to adjust the view of data in the active window. For instance, the View menu in a word-processing application might have items, which allow users to view the same information in different ways, such as with or without text symbols.

Typical Help Menu Help menu items provide access to onscreen information about the features of an application. This menu also provides access to the applications About box, which displays basic information about the application. For details, see "Designing About Boxes" on page 73. The following figure shows common Help menu items (in the typical order) with their mnemonics.

These items will vary according to the needs of your application.

FIGURE 96 Typical Help Menu

In your Help menu, allow access to on-screen information about the features of the application.

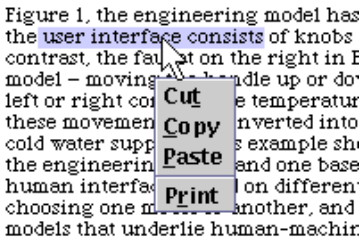
Place a separator before an About Application item (for example, About MetalEdit) that displays a dialog box with the product name, version number, company logo, product logo (or a visual reminder of the product logo), legal notices, and names of contributors to the product.

JavaHelp extension to the Java Development Kit can be used to build a help system for your applications.

Contextual Menus

Sometimes called a “pop-up menu,” a **contextual menu** offers only menu items that are applicable or relevant to the object or region at the location of the pointer. The appearance of contextual menus in the Java look and feel is identical to that of drop-down menus, including the display of mnemonics and keyboard shortcuts.

FIGURE 97 Contextual Menu



Users can display a contextual menu by clicking or pressing mouse button 2 while the pointer is over an object or area associated with that menu (or clicking while pressing the Control key on the Macintosh platform).

For keyboard operations in contextual menus, see Table 20 on page 190.

When closed, contextual menus do not provide any visible indication of their presence. Users might not find them, especially if your application does not use this kind of menu widely. Therefore, ensure that any features you present in contextual menus are also available in more visible and accessible places, like drop-down menus.

Display keyboard shortcuts and mnemonics in contextual menus that are consistent with their usage in corresponding drop-down menus.

Contextual menus are created using the `JPopupMenu` component.

Toolbars

A **toolbar** provides quick and convenient access to a set of frequently used commands or options. Toolbars typically contain buttons, but other controls (such as text fields and combo boxes) can be placed in the toolbar as well. An optional, textured “drag area” on the toolbar indicates that users can drag the toolbar to a different edge of the window or into a separate utility window. The drag area is on the left edge when the toolbar is horizontal and on the top when it is vertical.

The following figure shows a toolbar with a drag area on the left edge.

FIGURE 98 Toolbar

Users typically gain access to the controls in the toolbar by clicking the mouse button. For information on the keyboard operations that are appropriate for toolbars, see Table 30 on page 197.

If your window does not have menus, or if the toolbar buttons represent commands that are not in the menus, make special provisions for accessibility, including a text identifier, either in the button, or in a label below the button with a mnemonic for that label.

Toolbar Placement

In general, a toolbar is located at the edge of the window or area on which it operates.

If your window has a menu bar, place the toolbar horizontally in the row immediately under the menu bar.

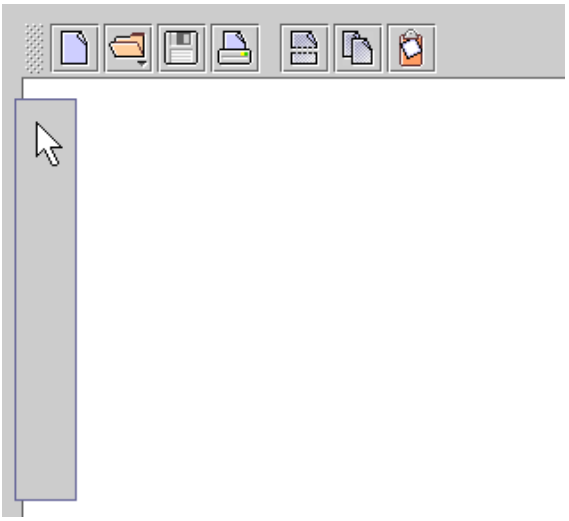
Include all toolbar commands in the menus as well.

Since multiple rows of components create clutter and make the functions harder to find, limit your window to a single toolbar with a single row.

Draggable Toolbars

You can design your toolbar to be draggable so users can move it or display it in a separate utility window. Users drag the toolbar by holding the mouse button down over the drag area. An outline of the toolbar moves as the user moves the pointer. The outline provides an indication of what will happen when the user releases the mouse button. When the pointer is over a “hot spot,” the outline has a dark border, indicating the toolbar will anchor to an edge of the container, as shown in the following figure.

FIGURE 99 Outline of Toolbar Being Dragged



If the pointer is outside a hot spot, the outline has a light border, indicating that the toolbar will be displayed in a separate utility window, as shown in the following figure. Closing the utility window returns the toolbar to its original location.

FIGURE 100 Toolbar in a Separate Utility Window

A toolbar can dock, or attach, along the top, bottom, left, and right edge of the container. One or more sides of the container must remain open so the toolbar can find a spot to which it can dock.

Draggable toolbars can be supported with the `BorderLayout` container.

Toolbar Buttons

A **toolbar button** is a command button or toggle button that appears in a toolbar, typically as part of a set of such buttons. Toolbar buttons can also act as titles to display menus. In other contexts, command buttons typically use text to specify the operation or state they represent, but toolbar buttons typically use graphics. In a toolbar button, that graphic occupies the entire interior space of the button.

Use buttons that are either 16 x 16 or 24 x 24 pixels (but not both), depending on the space available in your application.

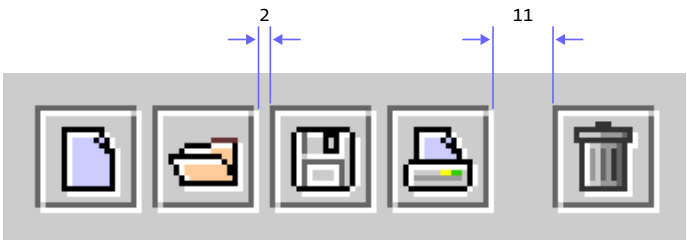
Since toolbar graphics can be difficult for users to understand, weigh the relative merits of user comprehension of graphics and the space taken up by button text before you decide to use button text in addition to the button graphics.

If you use text on the toolbar buttons, provide a setting to display the graphics only. Using this mode, you conserve space so you can display more commands and settings in the toolbar.

To facilitate keyboard access, define a mnemonic for a toolbar button (or other component) that has a text label.

Toolbar Button Spacing Space individual toolbar buttons two pixels apart. Space groups of toolbar buttons 11 pixels apart.

FIGURE 101 Toolbar Button Spacing



The margin on toolbar buttons should be set to have an inset of zero.

Mouse-over Borders To conserve space, you can use mouse-over borders (also called “rollover borders”) on toolbar buttons. This border appears around a button when the user moves the pointer over it; otherwise, the border is invisible. The following figure shows a toolbar with a mouse-over border activated for the Save button.

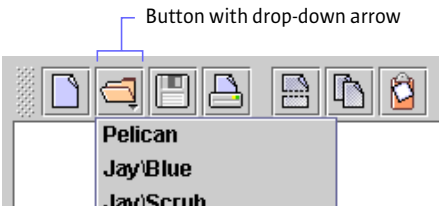
FIGURE 102 Mouse-over Borders on Toolbar Buttons

When you use mouse-over borders, do not include any space between the buttons within a group.

The client property `JToolBar.isRollover` is set to true to enable mouse-over borders.

Drop-down Menus You can attach a drop-down menu to a toolbar button. The menu appears when the user clicks (or presses and holds the mouse button over) the toolbar button. The following figure shows a drop-down menu indicated by a **drop-down arrow** on the Open button. Te menu provides a list of files to open.

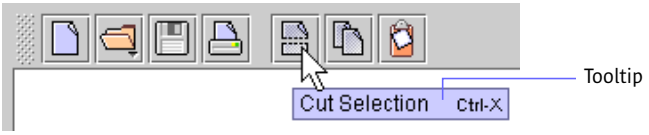
FIGURE 103 Toolbar Button With Drop-down Menu



Provide a drop-down arrow in the graphic for any toolbar button that has a drop-down menu.

Tool Tips for Toolbar Buttons You can provide tool tips for the toolbar components. The tool tip displays information about the component when the user rests the pointer on it. If the component has a keyboard shortcut, it is automatically included in the tool tip. The following figure shows a tool tip that describes the Cut button.

FIGURE 104 Tool Tip for a Toolbar Button



Attach tool tips to all toolbar components that do not include text labels.

When your application menus do not display keyboard shortcuts, display the keyboard menus in tool tips for the toolbar components.

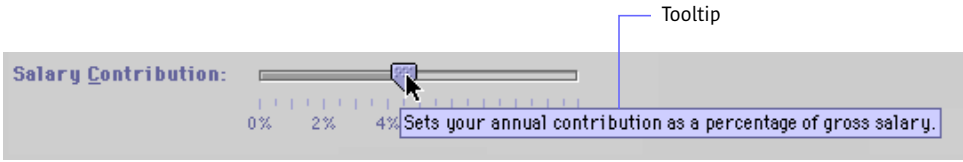
Tool Tips

A **tool tip** provides information about a component or area when the user rests the pointer on it (and does not press a mouse button). This small, rectangle of text can be used anywhere in the application. A tool tip is commonly attached to an interface component, where it provides a short description of the component's function. If a component has a keyboard shortcut, the shortcut is automatically included in the tool tip.

For keyboard operations in tooltips, see Table 31 on page 197.

The following figure shows a tool tip that describes a slider.

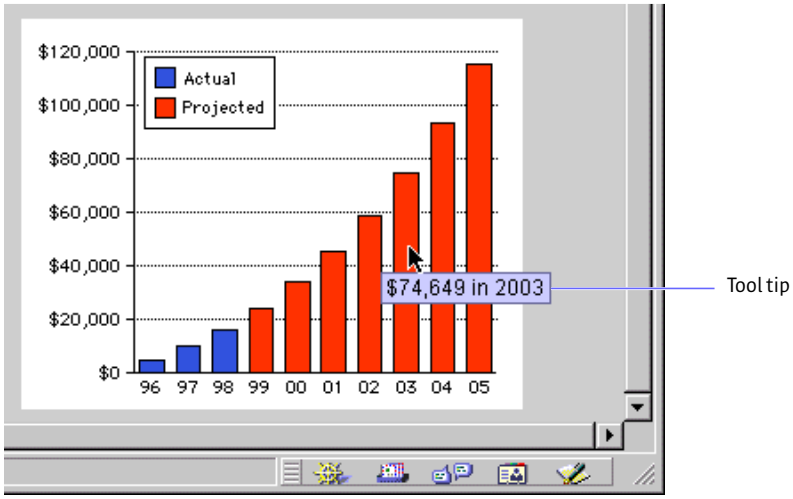
FIGURE 105 Tool Tip for a Slider



You can also use tool tips with graphics. A graphic might have one tool tip that provides the name and size of the graphic or several tool tips that describe different areas of the graphic.

The following figure shows a tool tip on an area of the bar chart in the sample applet, Retirement Savings.

FIGURE 106 Tool Tip on an Area Within a Graphic



You can adjust the timing of the tooltips in your application. By default, a tool tip appears after the user rests the pointer on the component or area for 750 milliseconds. It disappears after four seconds or when the user activates the component or moves the pointer off it.

 Make tool tips active by default, but provide users a way to turn them off. For example, you might provide a checkbox in either a menu or in the Preferences dialog box.

 A tool tip is specified in its associated component (and not by calling the `JToolTip` class directly).

 Tool tips are used as the component's `accessibleDescription` property. Assistive technologies use this property (in addition to `accessibleName`, which distinguishes the component from other components of the same type) to provide a description of the component's purpose in the interface.

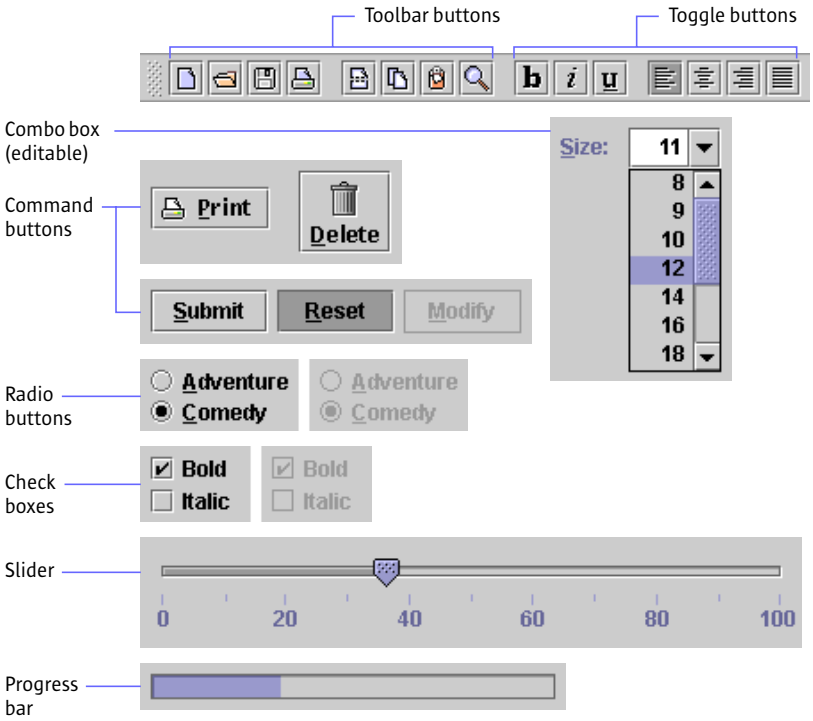
For details on the Java 2 Accessibility API, see "Accessibility and JDK" on page 15.

10: BASIC CONTROLS

Buttons, combo boxes, and sliders are examples of controls—interface elements that users can manipulate to perform an action, select an option, or set a value. A **button** is a control that users select to perform an action, set or toggle a state, or set an option. In the Java look and feel, buttons include command and toggle buttons, toolbar buttons, checkboxes, and radio buttons. A **combo box** is a control with a drop-down arrow that the user clicks to display a list of options. A **slider** is a control that enables users to set a value in a range—for example, the RGB values for a color.

A **progress bar** is an interface element that indicates one or more operations are in progress and shows the user what proportion of the operation has been completed. In contrast to the other components described in this chapter, no user manipulation is involved.

FIGURE 107 Buttons, Combo Boxes, Sliders, and Progress Bars



For text in buttons, sliders, and combo boxes, use headline capitalization.

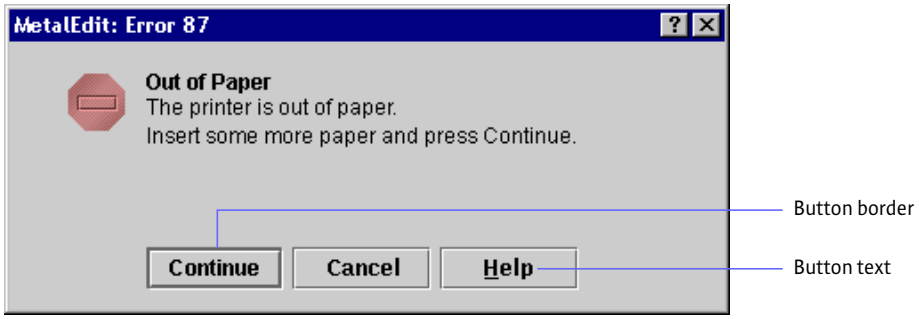
Make sure you use the appropriate layout manager to lay out your controls so they allow for the longer text strings frequently associated with localization.

Command Buttons

A **command button** is a button with a rectangular border that contains text, a graphic, or both. These buttons typically use text, called button text, often a single word, to identify the action or setting that the button represents. See "Command Buttons" on page 112 for a list of commonly used command button names and their recommended usage.

Command buttons can appear standalone, as shown in the illustration of an About Box in Figure 47 on page 73, or in a row, as shown in the following illustration of an alert box.

FIGURE 108 Command Button Row



Command buttons can also appear in toolbars, as shown in the following figure.

FIGURE 109 Command Buttons in Toolbar

When a command button is unavailable, the dimmed appearance indicates that it cannot be used.

FIGURE 110 Available, Pressed, and Unavailable Command Buttons

Users can click command buttons to specify a command or initiate an action, such as Save or Cancel or Submit Changes.

For details on layout and spacing of command buttons, see “Command Button Spacing” on page 145.

For a list of keyboard operations for the activation of command buttons, see Table 15 on page 186.

Display mnemonics in button text (such as an underlined “S” in a Submit button), when possible with the exception of default button and Cancel button in dialog boxes, which should not allow mnemonics.

For general details on keyboard operations and mnemonics, see “Keyboard Operations” on page 80 and “Mnemonics” on page 86.

For guidelines on the spacing of command buttons, see “Command Button Spacing” on page 145.

Default Command Buttons

One of the buttons in a window can be the default command button. The default command button is activated when the suer presses Return or Enter. A default command button (such as Save in the following figure) should represent the action most often performed.

FIGURE 111 Default and Nondefault Command Buttons

Return or Enter are the keyboard equivalents for the default button. These equivalents work unless the focus is currently on a component that accepts the Return or Enter key. For instance, if the insertion point is in a multi-line text area and the user presses Return, the insertion point moves to the beginning of a new line rather than activating a default button. Keyboard focus must be moved to another component before the default button can be activated.

The Escape key is not supplied as the keyboard equivalent for the Cancel button. As with the Enter and Return keys for the default command button, the Cancel button does not require keyboard focus to be activated by the Escape key.

Since you are not required to have a default button in every circumstance, you can use discretion about including them in your interface elements.

Make the default button a safe choice, a choice that does not lead to the loss of any user data.

Never make an unsafe the choice the default button, for instance, “Close without saving.”

Do not supply mnemonics for the default and Cancel buttons.

Combining Graphics With Text in Command Buttons In some circumstances, you might use a graphic along with text to identify the action or setting represented by a command button.

FIGURE 112 Command Buttons Containing Both Text and Graphics

Place the text to the right or below the image, as shown in Figure 112. The orientation of the graphic is automatically adjusted in locales with other writing systems.

Include mnemonics in your command button text with the exception of default and cancel buttons.

Use the button component’s orientation that takes into account the button’s locale, so that when the orientation is switched the text will be displayed on the appropriate side.

For a list of commonly used mnemonics, see Table 10 on page 87.

Using Ellipses in Commands In circumstances in which a command button does not fully specify an action or operation and users need a dialog box to finish the specification, you can notify the user that this situation is about to occur by placing an ellipsis after the button text. For example, after clicking a Print... button, users are presented with a Print dialog box in which to specify printer location, how many copies to print and so forth.

By contrast, users might click a Print button without ellipses and expect that one copy of the document is to be printed by the default printer. In this case, no ellipses are required.

When users must view a dialog box to finish the specification of a command initiated in a command button, use an ellipsis (...) after the button text.

When a full specification of the command is made in the button text, do not uses ellipses.

Command Button Spacing

For a consistent appearance, follow the guidelines described in this section to create text within and space between command buttons.

Since the length and height of translated text varies, use layout managers properly to allow for differences. <<Ask Brian Beck to give details on which layout managers to use and how>>

Command Button Padding Padding is the white space between objects. In the following figure, the padding is the space between the font ascender and the inside highlight border (not the outside button border), and the space between the font baseline (not the descender) and the bottom button border. <<For callout specs, see visualButtonSpacingExternal.gif on Chris' Visual Design page>>

Provide 12 pixels of padding (for the button with the widest button text).

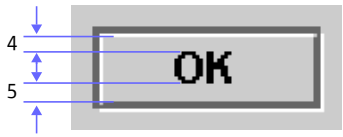
Insert five pixels of space between the top of the button text and the inside white border. Insert six pixels between the text's baseline (not the descenders) and the bottom button border.

Center text within buttons. Center button text and graphics within the dark borders, since they are more visually significant.

FIGURE 113 Command Button Padding

Default Button Spacing In the Java look and feel, default buttons (for instance, the OK button in the following figure) display a heavier border, which is inset, so the vertical spacing measurements decrease by a pixel. The horizontal measurement is determined by the widest button in the set.

FIGURE 114 Default Button Padding



Command Button Group Spacing Often command buttons appear in groups within a dialog box or an applet. In such a case, the button in the group with the widest text determines the inner padding. The distance from the left edge of the text to the interior white highlight is eleven pixels, and the distance from the right edge of the text to the inside border is twelve pixels.

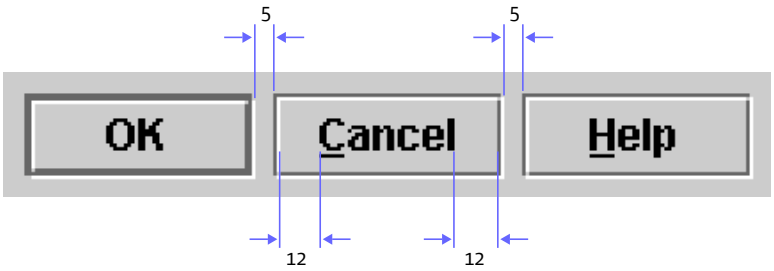
Make all buttons in a command button group as large as the largest button.

The width of the widest string and its padding and borders determines the width of all the other buttons.

When the default button's text is the widest in the group, the values for interior padding are both reduced by one pixel.

Space buttons in a group five pixels apart.

FIGURE 115 Spacing in Command Button Groups



Button Graphic Spacing The standard sizes for button graphics are 16 x 16 pixels and 24 x 24 pixels.

To achieve a balanced effect, make the space between the left border of the button graphic (b) one pixel less than the space between the right edge and the inside right of the button border (c). This space does not include the white highlight used to achieve the flush appearance for the button graphic

Toggle Buttons

A **toggle button** is a button that represents a setting with two states—on and off. Toggle buttons look similar to command buttons and typically display a graphic or text that identifies the button. The graphic or text remains the same whether the button is in the on or off state.

Users can click toggle buttons to turn a setting on or off, for instance, to toggle between the display and nondisplay of italics in selected text.

You can use toggle buttons to represent an independent choice, like checkboxes (see page 148), or an exclusive choice within a set, like radio buttons (see page 151).

Toggle buttons can be placed in a button group to get radio button behavior.

Always display the border on a toggle button.

Independent Choice

An independent toggle button behaves like a checkbox. Whether it appears alone or with other buttons, its setting is independent of other controls. An example of an independent toggle button is a Bold button on a toolbar.

FIGURE 116 Independent Toggle Buttons

Bold style has been applied to selection

When the user clicks the Bold button, it is highlighted to indicate that the bold style has been applied to the selection, or that text to be entered will be bold. If the button is clicked again, it reverts to the normal button appearance and the bold style is removed from the selection.

Although checkboxes and independent toggle buttons have the same function, use checkboxes in dialog boxes and toggle buttons with a graphic in toolbars as a general rule.

When toggle buttons are independent (like checkboxes) and used outside a toolbar, separate them with 5 pixels. Within a toolbar, separate independent toggle buttons by 2 pixels.

For details on the spacing of toggle buttons, see "Command Button Spacing" on page 145.

Exclusive Choice

e A toggle button can also work as part of a group to represent an exclusive choice within the set. A common example is a set of toolbar toggle buttons representing left, center, and right text alignment.

FIGURE 117 Standard Separation of Exclusive Toggle Buttons

If the user clicks the button representing left alignment, the button is highlighted to indicate that text is aligned flush with the left border of the document. If the user then clicks the button representing center alignment, the appearance of the Align Left button reverts to the normal button appearance and the Center button is highlighted to indicate center alignment of the selected text.

Grouped Toggle Buttons With Labels

You can use grouped toggle buttons with labels as text identifiers equally well in toolbars or dialog boxes.

FIGURE 118 Grouped Toggle Buttons with Text Identifiers

When toggle buttons form a radio set, separate them with 2 pixels.

Checkboxes

A **checkbox** is a control associated with text that users click to select or deselect an object. Each checkbox represents a setting or value with an on or off choice. The setting of an individual checkbox is independent of other controls.

A check mark within the checkbox indicates that the setting is selected. The following figure shows checkboxes and disabled checkboxes in both states.

When the user clicks a checkbox, its setting toggles between off and on. When a checkbox is disabled, the user cannot change its setting.

For a list of keyboard operations for checkboxes, see Table 13 on page 186.

FIGURE 119 Checkboxes

Use the standard checkbox graphics, that is, the square box with the check mark inside.

Display checkbox text to the right of the graphic except in locales with right-to-left writing systems, such as Arabic and Hebrew. In this case, display the text should to the left of the graphic.

Although checkboxes and independent toggle buttons have the same function, use checkboxes in dialog boxes and use toggle buttons with a graphic in toolbars.

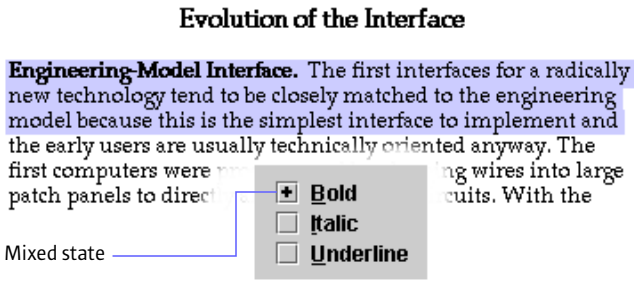
In addition to standard checkboxes, the JFC includes a component that is the functional equivalent of the checkbox for use in menus. See "Checkbox Menu Items" on page 128 for more information.

Representing Mixed States in Checkboxes

You can use checkboxes to indicate the state of selected text or objects. If the selection includes items that have both the on and off setting represented by a checkbox, you can indicate this mixed state with a plus symbol within the box.

In the following figure, some of the text in the range is bold and some is plain text; this is indicated by the plus (+) sign in the Bold checkbox. Clicking a mixed-state checkbox sets its value to on (that is, checked). After that, until the selection is changed, the checkbox only toggles back and forth between on and off states.

FIGURE 120 Suggested Mixed-State Checkbox



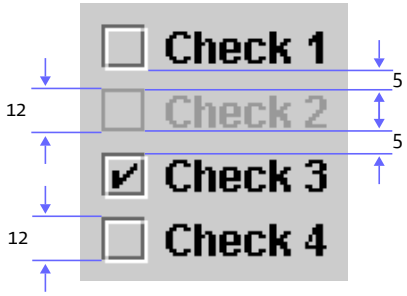
Mixed-state checkboxes are not yet a standard feature of the JFC and require a significant amount of implementation, such as subclassing and overriding behaviors.

Checkbox Spacing

This section provides the spacing guidelines for checkbox components.

Space checkboxes five pixels apart.

FIGURE 121 Checkbox Spacing <<callouts need adjustment>>



The height of the checkbox square is 12 pixels, not counting the white highlight border. As shown in the preceding figure, inactive checkboxes do not have white borders. Hence, while the checkbox is the same size, the last row and column of pixels on the bottom and right are the same color as the background canvas. The apparent spacing is six pixels between components; however, the actual spacing is five pixels.

Use the appropriate layout manager to achieve consistent spacing in checkbox button groups.

Radio Buttons

A **radio button** represents an exclusive choice within a set of related options. Within a set of radio buttons, only one button can be on at any given time. The following figure shows radio buttons and disabled radio buttons in both on and off states.

FIGURE 122 Radio Buttons

When users clicks a radio button, its setting is always set to on. A bullet within the round button indicates that the setting is selected. If another button in the set had previously been selected, its state changes to off. When a radio button is disabled, the user cannot change its setting.

For a list of keyboard operations for radio buttons, see Table 21 on page 190.

Use the standard radio button graphics (that is, the open circles with the inner filled circles).

Display radio button text to the right of the graphic, except in locales with right-to-left writing systems, such as Arabic and Hebrew. In this case, place the text to the left of the graphic.

Although radio buttons and toggle buttons in a radio set have the same function, use radio buttons in dialog boxes and use grouped toggle buttons with graphics in toolbars. Grouped toggle buttons with text identifiers work well in either situation.

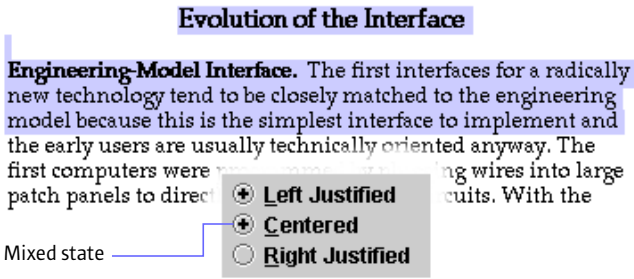
The JFC includes a component that is the functional equivalent of the radio button for use in menus. See "Radio Button Menu Items" on page 128 for more information.

Mixed State Radio Buttons

You can use radio buttons to indicate the state of selected text or objects. If the selection includes items that have both the on and off setting, you can indicate this mixed state with a plus sign displayed within the circle, as shown in the following figure.

The selected range of text includes some text that is left aligned and some that is center aligned; this is indicated by the plus (+) sign in the Left and Middle radio buttons. Clicking a mixed-state radio button sets its value to on and turns off all other radio buttons in the set. From that point forward, unless the selection is changed, only one radio button in the set can have the on state.

FIGURE 123 Suggested Mixed State in a Range of Radio Buttons



Mixed-state radio buttons are not yet a standard feature of the JFC and require a significant amount of implementation, such as subclassing and overriding behaviors.

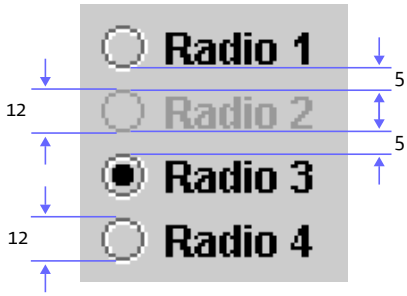
Radio Button Spacing

This section provides guidelines for the spacing of radio buttons.

Space radio buttons five pixels apart, as shown in the following figure.

Do not provide a white border for disabled controls since the space that would have been occupied by the border is still part of the control.

FIGURE 124 Radio Button Spacing <<callouts off>>



The height of the radio button is 12 pixels, not counting the white highlight border. As shown in the preceding figure, inactive radio buttons do not have white borders. Hence, while the radio button is the same size, the last row and column of pixels on the bottom and right are the same color as the background canvas. The apparent spacing is six pixels between components; however, the actual spacing is five pixels.

Use the appropriate layout manager to achieve consistent spacing in radio button groups.

Combo Boxes

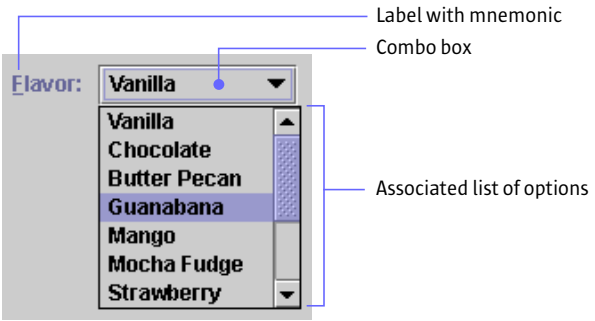
A **combo box** is a component with a drop-down arrow that the user clicks to display a list of choices.

Until users click a drop-down arrow to post the list or press the mouse button to pull down the list, only the currently selected item appears in the combo box.

Users can close either editable or noneditable combo boxes three ways: clicking the drop-down arrow, choosing an item from the list, or clicking anywhere outside the combo box.

For a list of keyboard operations for combo boxes, see Table 14 on page 186.

FIGURE 125 Three Ways to Close Combo Boxes <<needs callouts>>



You can use combo boxes to provide a way for the user to indicate a choice from a set of mutually exclusive options. Noneditable combo boxes enable users to choose one item from a limited set of items, whereas editable combo boxes enable users to choose one item from an unlimited set of items.

Use headline capitalization for the text in the items in the combo box list.

Provide labels with mnemonics for combo boxes.

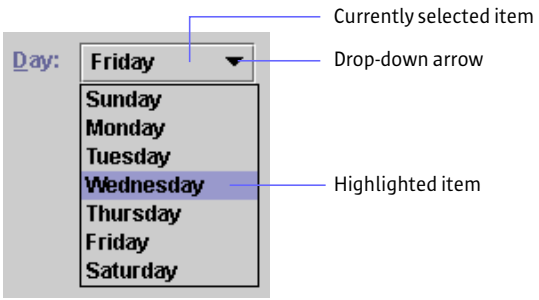
In JFC, the term “combo box” component includes both of what Microsoft Windows calls “list boxes” and “drop-down combo boxes.”

Noneditable Combo Boxes

Noneditable combo boxes (sometimes called “list boxes” or “popup menus”) display a list from which the user can select one item.

The following figure shows a noneditable combo box with a drop-down arrow to the right of the currently selected item. (Note the gray background in the default Java look and feel theme indicates that users cannot type or select text).

FIGURE 126 Noneditable Combo Box <<more callouts needed>>



To make a choice, users have two options:

- Click the combo box to display the list, position the mouse over the desired option to highlight it, then click
- Drag from the combo box to the desired choice and release

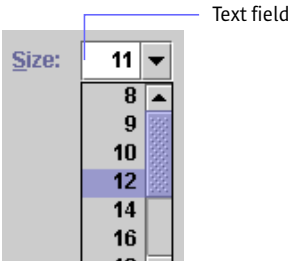
In either case, the combo box reflects the choice.

You can use a noneditable combo box instead of a group of radio buttons or a list to display one of many choices when space is limited in your application.

Editable Combo Boxes

Editable combo boxes combine a text field with a drop-down arrow to display an associated list of options.

FIGURE 127 Editable Combo Box



As shown in the preceding figure, editable combo boxes appear as editable text fields with a drop-down arrow. The white background of the editable combo box indicates that users can type, select, and edit text.

To make a choice, users have three options:

- Click the drop-down arrow to display the list, position the mouse over the desired option to highlight it, then click
- Drag from the drop-down arrow to the desired choice and release.
- To make a customized choice, type the desired text in the field. The chosen option then appears in the editable combo box and the associated list closes.

If the number of items is too big to fit in the list, a vertical scroll bar appears.

You can use an editable combo box to save users time by making the most likely menu choices available, and, at the same time, to allow users to type other values in the text field. An example might be the specification of a font size. The combo box initially displays the most often used size, say 12. Users can select from a menu of standard sizes (10, 12, 14, 18, or 24 points) or type in their own values, for instance, 22 points.

Whenever possible, interpret the user's input in a case-insensitive way. It should not matter whether the user types "Blue," "blue," or "BLUE."

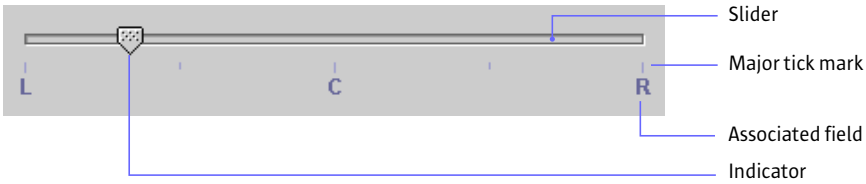
The maximum number of items displayed before a scroll bar appears can be specified by the developer.

Sliders

A slider is a control that is used to select a value from a continuous or discontinuous range of values. The indicator designates the current setting of the slider. Major tick marks indicate large divisions along the range of values (for instance, every ten units); minor tick marks indicate smaller divisions (for instance, every five units).

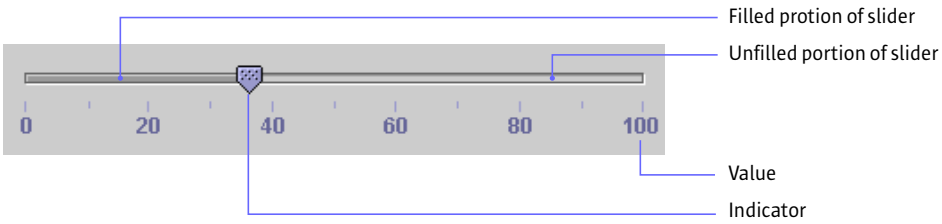
The default slider in the Java look and feel is a non-filling slider. An example is a slider that adjusts left-right balance in a stereo speaker system, as shown in the following figure.

FIGURE 128 Non-Filling Slider <<add callouts for slider, indicator, major tick mark, and associated text>>



A filling slider is also available. The filled portion of the slider, shown in the following figure, represents the range of values below the current value.

FIGURE 129 Filling Slider <<add callouts for indicator, values, unfilled portion of slider, filled portion of slider, major tick marks>>



Users can drag the indicator to set a specific value or click the slider to move back and forth by one unit. Sliders can represent a series of discrete values, in which case the indicator snaps to the value closest to the spot where the user clicked (or closest to the end point of the drag operation).

For a list of keyboard operations for sliders, see Table 23 on page 191.

You can indicate values along the slider with major and minor tick marks, which can also have associated text. You can use the nonfilling slider to represent discrete values (as opposed to continuous values, for which a filling slider is more appropriate).

If the slider represents a continuous range or a large number of discrete values and the exact value that is chosen is important, provide a text field where the chosen value can be displayed. For instance, a user might want to specify an annual retirement savings contribution of 22.35%. In this case, consider making the text field editable to give users the option of typing in the value directly.

The `ClientProperties` `"JSlider.isFilled"` can be used to enable the optional filling slider.

Progress Bars

A **progress bar** indicates that one or more operations is underway and shows users what proportion of the operations has been completed. The progress bar consists of a rectangular bar that fills as the operations progress, as shown in the following figure.

FIGURE 130 Progress Bar

Users cannot interact with a progress bar; it is for display only. If you would like to enable users to set a value in a range, use the component, described in "Sliders" on page 156.

You can orient the progress bar horizontally, so it fills from left to right, or vertically, so it fills from bottom to top. Within the bounds of the progress bar, you can display a text message that is updated as the bar fills. By default, the message shows the percentage of the process completed, for example, 25%.

The following figure shows another use of the progress bar. In this example of a process control application, the progress bar is not used to track the progress of an operation; rather, it displays the temperature of a vat in a candy factory. The temperature is indicated graphically and the text message within the progress bar shows the actual value.

FIGURE 131 Text Inside Progress Bar

If you create your own message to display inside the progress bar, make it concise.

11: TEXT COMPONENTS

The JFC includes four types of components that you can use to provide text in your application. The simplest text you can add to your application is a **label**, which presents read-only information. A label usually accompanies a control to describe its function. You can also provide a single line of text using a **text field**, which can be editable or noneditable. A **password field** is an editable text field that displays asterisks in place of the characters that the user types.

Other text components enable you to provide multiple lines of text that users can view and edit. A **text area** contains multiple lines of text in a single font size and style. An **editor pane** can display different types of text, such RTF (rich text format) and HTML (Hypertext Markup Language) text, through the use of a plug-in editor.

Make your text easier to localize by using a layout manager and resource bundles. A layout manager automatically adjusts the height and width of the application when the translated text becomes bigger or smaller. A resource bundle stores the text outside the application so that localizers don't have to change the source code when they translate the text. For guidelines on translating text, see "Internationalization and Localization" on page 29.

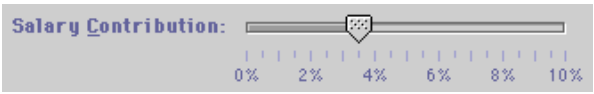
Labels

A label consists of read-only text, images, or both. Labels serve two functions in an application: to identify controls and to communicate status. Users cannot select a label.

Labels that Identify Controls

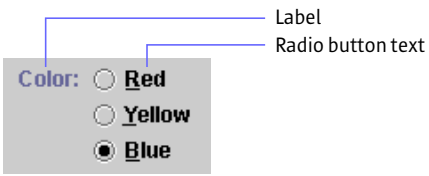
You can associate a label with a control (such as a text field, slider, or checkbox) to help users understand how to use the control. In the following figure, the label "Salary Contribution:" identifies a slider.

FIGURE 132 Label for Slider



You can also use a label to introduce a group of controls. In the following figure, the label “Color:” introduces three radio buttons. The text Red, Yellow, and Blue is part of the radio button and not a separate component, as is the label “Color:”.

FIGURE 133 Label for Radio Button Group

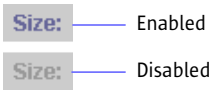


A label is drawn in the primary 1 color defined in the application’s color theme. This color distinguishes the label text, which is read-only, from text that users can select and edit.

Keep the label text brief, and use language familiar to users.

Disabled Labels You can disable a label, in which case the label is drawn in the secondary 2 color defined in the application’s color theme. The following figure compares an enabled and disabled label.

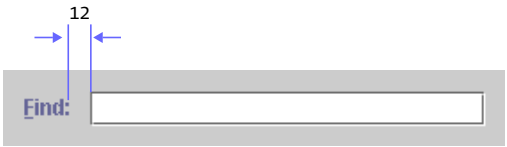
FIGURE 134 Disabled Label



Disable a label when the control it describes is disabled.

Spacing, Alignment, and Capitalization of Labels The following figure shows the recommended spacing, alignment, and capitalization of labels.

FIGURE 135 Spacing Between A Label and a Component



Insert 12 pixels between a label and the component it describes.

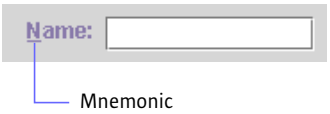
Display a label before or above the control it describes. For languages that are read left-to-right, “before” is on the left of the component.

Use headline capitalization and colons in the label.

For more information on aligning labels in the user interface, see "Text Layout" on page 49.

Mnemonics in Labels You can apply a **mnemonic** to a label. When activated, the mnemonic gives focus to the component that the label describes. This technique is often used with a label that accompanies an editable text field. In the following figure, the text field gets focus when users type Alt-N.

FIGURE 136 Label With Mnemonic That Gives Focus to Text Field



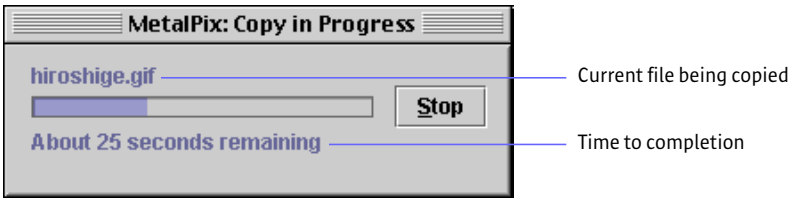
If you can't add a mnemonic to a component, as in the case of an editable text field, provide the component with a label and give the label a mnemonic.

The `labelFor` property associates a label with another component so that the component gains focus when the label's mnemonic is activated.

Labels That Communicate Status

You can use a label to communicate status and give instructions to users. In addition, you can alter the label to show a change in state. The progress bar in the following figure uses two labels that change as the operation progresses: the top label changes to reflect the file currently being copied, and the bottom label updates as the progress bar fills.

FIGURE 137 Labels That Clarify Meaning of Progress Bar



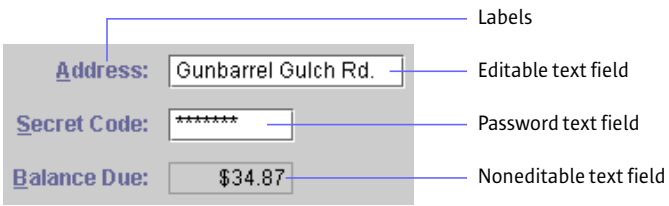
Use sentence capitalization when using a label to communicate status.

Keep the text of the label brief, and use language familiar to users.

Text Fields

A **text field** is a rectangular area that displays a single line of text. The JFC provides three types of text fields: noneditable, editable, and password. In most cases, a text field is accompanied with a label that describes the purpose of the text field, as shown in the following figure. The label is a separate component from the text field.

FIGURE 138 Noneditable, Editable, and Password Text Fields

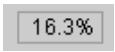


The label of a text field commonly includes a mnemonic that, when pressed, gives focus to the text field. For information on associating labels with text fields, see "Mnemonics in Labels" on page 161.

Noneditable Text Fields

A noneditable text field displays a single line of text that users can select and copy, but not change. Only the application can change the contents of a noneditable text field. By default, the background of a noneditable text field is gray, as shown in the following figure.

FIGURE 139 Noneditable Text Field



Editable Text Fields

An editable text field is an area in which users can type or edit a single line of text. For example, a find dialog box has a text field in which users type the word for which they want to search. A text field has keyboard focus when it displays a blinking insertion point. When users type in text that is too long to fit in the field, the text scrolls horizontally. By default, the background of an editable text field is white.

The following figure shows an editable text field with keyboard focus. The label "Language:" is a separate component from the text field.

FIGURE 140 Editable Text Field With Insertion Point

In an editable text field, users can perform the following actions:

- Set the insertion point by single-clicking.
- Select a word by double-clicking.
- Select the entire line of text by triple-clicking.
- Select a range of characters by dragging.
- Insert characters and replace selected text by typing at the insertion point.
- Cut, copy, and paste text by using menu commands or the keyboard shortcuts Ctrl-X, Ctrl-C, and Ctrl-V.

The following figure shows a text field with the letters “Jeffer” selected. The insertion point is at the end of the selected text and indicates that the text field has keyboard focus. The selected text will be overwritten when the user types or pastes new text.

FIGURE 141 Editable Text Field With Selected Text

For keyboard operations appropriate to text fields, see Table 28 on page 196.

If users type an illegal character into a text field (for example, type an “f” into a text field that should only contain numbers), do not display the character in the field. Instead, play the system beep. If users type three illegal characters in a row, post an Error alert box that explains the legal entries for the text field.

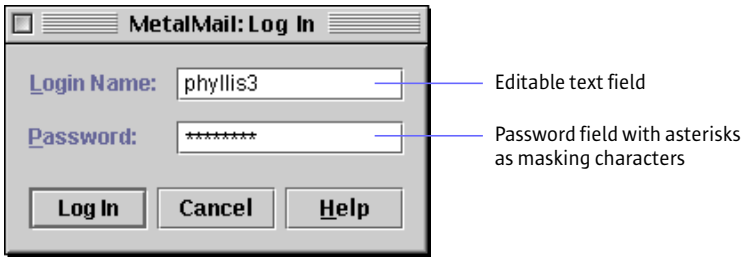
If you plan an action based on the text in the text field, such as checking the text for errors or applying the value, do it when users signify that they have completed the entry by typing Enter or Return or by moving keyboard focus outside the text field.

Password Fields

The **password field** is an editable text field that displays a masking character instead of the characters that users type. Asterisks are displayed in the password field by default, but you can designate any standard ASCII character as the masking character.

The password field is commonly used in a login dialog box, as shown in the following figure. The label “Password:” is a separate component from the password field.

FIGURE 142 Password Field

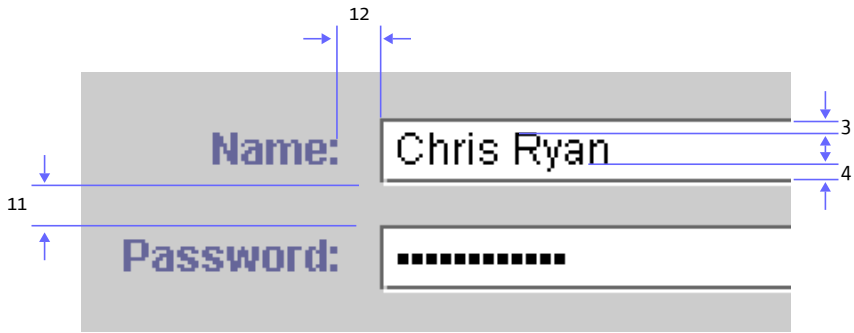


A password field provides users the same editing capabilities as an editable text field, except for cut and copy. For keyboard operations appropriate to password fields, see Table 28 on page 196.

The `setEchoChar` method changes the masking character.

Spacing of Noneditable, Editable, and Passwords Fields The following figure shows the recommended spacing of the text in a text field and the spacing between text fields. These guidelines apply to noneditable, editable, and password fields.

FIGURE 143 Text Field Spacing



Insert three pixels between the top of the text and the border of the field. Insert four pixels between the font baseline (not the descender) and the border at the lower border of the text field.

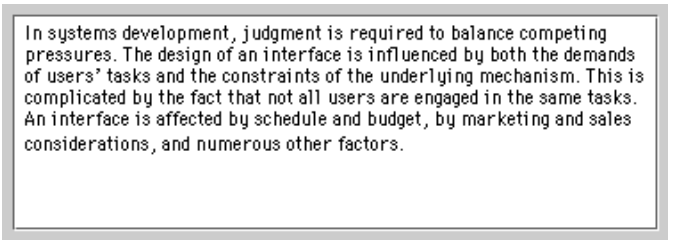
Insert eleven pixels between text fields.

Insert twelve pixels between a text field and its label.

Text Areas

A **text area** provides a rectangular area in which users can view, type, and edit multiple lines of text. The text is rendered in a single font size and style, as shown in the following figure.

FIGURE 144 Plain Text Area

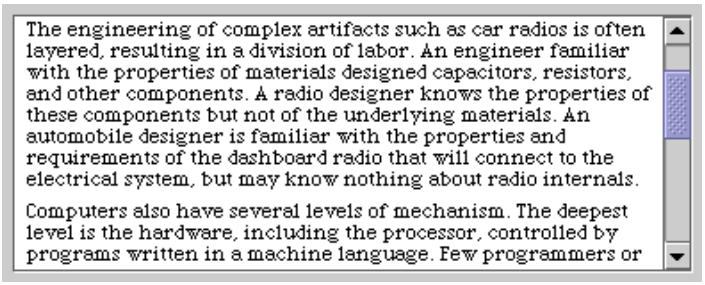


As in text fields, users can insert and replace text in a text area. See "Text Fields" on page 162 for a description of text editing features. For keyboard operations appropriate to a text area, see Table 27 on page 194.

You can enable scrolling in a text area by placing it inside a scroll pane. In this case, the text scrolls horizontally and vertically when it is too long to fit in the text area (instead of running outside the component). For information on scroll panes, see "Scroll Panes" on page 99.

You can enable word wrap so that the text wraps to the next line when it reaches the edge of the component. The following figure shows a text area with word wrap enabled. The text area is inside a scroll pane.

FIGURE 145 Text Area in a Scroll Pane



Setting the `lineWrap` and `wrapStyleWord` properties to true enables word wrap on word boundaries.

Editor Panes

An editor pane is a multi-line text pane that uses a plug-in editor kit to display a specific type of text, such as RTF (rich text format) text or HTML (Hypertext Markup Language) text. The JFC provides three kits that you can plug into an editor pane:

- **Styled text editor kit.** Displays formatted text.
- **RTF editor kit.** Displays formatted text and RTF text.
- **HTML editor kit**. Displays HTML 3.2 text.

You can also create your own editor kit with a custom text format.

An editor kit gives users everything they need to work with the text in that kit. For example, users can edit the text in the styled text and RTF editor kits. Users cannot edit the text in an HTML editor kit.

The `setEditable` method turns text editing on or off in a styled text for RTF editor kit.

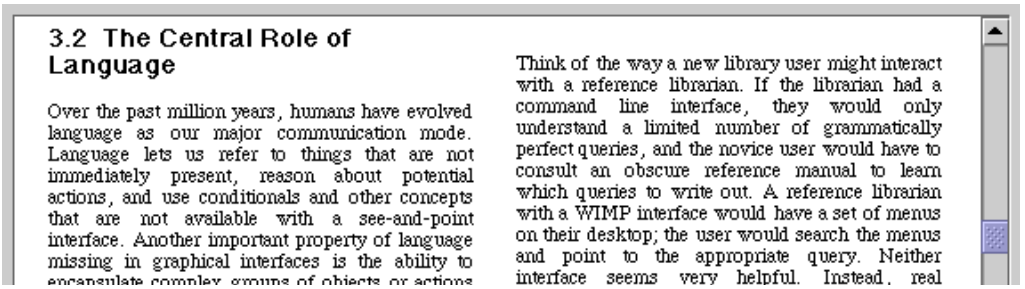
Styled Text Editor Kits

A styled text editor kit displays multiple font sizes and styles, as shown in the following figure. You can also embed images and other components (such as tables) in a styled text editor kit. Users can edit the text in the styled text editor kit.

FIGURE 146 Styled Text Editor Kit

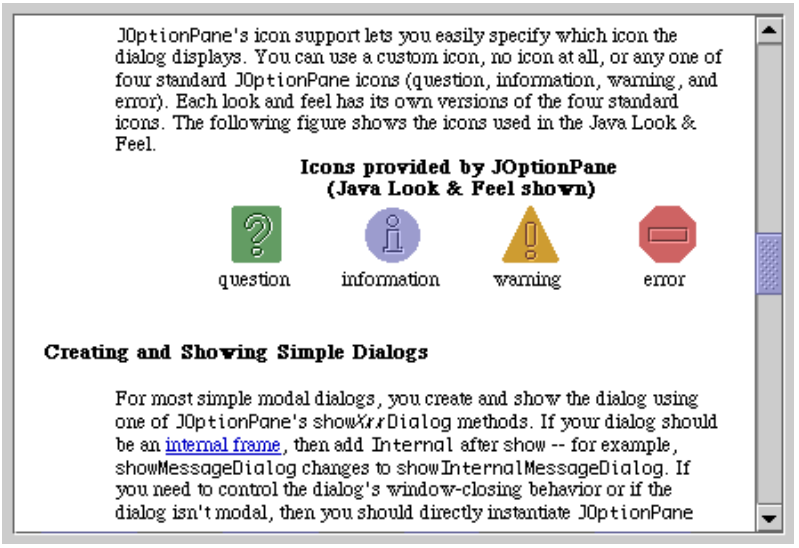
RTF Editor Kits An RTF editor kit displays RTF text, as shown in the following figure. Users can edit the text in an RTF editor kit.

FIGURE 147 RTF Editor Kit



HTML Editor Kits An HTML editor kit displays HTML 3.2. An HTML editor kit can display all fonts provided in the AWT. Users can select and copy the text and click links to display related information. Users cannot edit the text in an HTML editor kit.

FIGURE 148 HTML Editor Kit



12: LISTS, TABLES, AND TREES

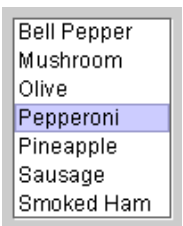
Lists, tables, and trees provide a way to organize related information so users can easily make comparisons of the data. A list is a one-dimensional arrangement of data, and a table is a two-dimensional arrangement of data. A tree view is an outline of hierarchical relationships.

Lists

A **list** displays a series of exclusive or non-exclusive choices for the user. The list items can consist of text, images, or both. Lists automatically include a vertical scroll bar when the number of items in the list is longer than the list's viewing area. Lists can also include a horizontal scroll bar when an item is wider than the viewing area.

Lists are well-suited for enabling users to select one or more items from a limited set of items, as shown in the following figure.

FIGURE 149 List



For a component better-suited for enabling users to select one item from a limited set of items, see "Noneditable Combo Boxes" on page 154; for selecting one item from an unlimited set of items, see "Editable Combo Boxes" on page 155; for selecting one value from a continuous or discontinuous range of values, see "Sliders" on page 156.

For the keyboard operations appropriate for lists, see Table 19 on page 189.

Scrolling Scrolling in lists works as follows. On the vertical scroll bar, users click the scroll bar arrow to scroll the list one line at a time or click the scroll bar channel to scroll one view minus one line (to maintain context for the user). Clicking the scroll bar channel never scrolls past the end or beginning of the list. Similarly, users interact with horizontal scroll bars by clicking the scroll bar arrow or channel.

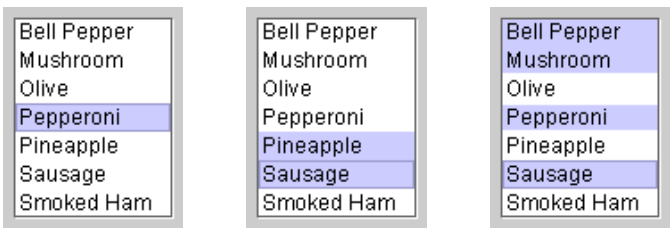
You can turn off the display of horizontal scroll bars. These scroll bars take up space at the bottom of the list, limiting the number of items users can view at a time. In most cases, your list should be wide enough so users can view the items without having to scroll.

Use a condensed font to adjust the length of lines that are too wide to display completely.

When resizing a list, be sure that it always display a whole number of lines.

Selection Models You can choose one of three selection models for your list: single item, single range, and multiple range. The following figure shows the selection models.

FIGURE 150 Selection Models in Lists



Single item Single range Multiple ranges

- **Single Item.** Users select a single item by clicking it. The item gets keyboard focus. The prior selection, if any, is deselected.
- **Single Range.** Users can select a single item or a range of items. Clicking an item selects the item and sets an anchor point. The selected item has keyboard focus. Moving the pointer to another item and Shift-clicking extends the range from the anchor point to the item that was Shift-clicked.

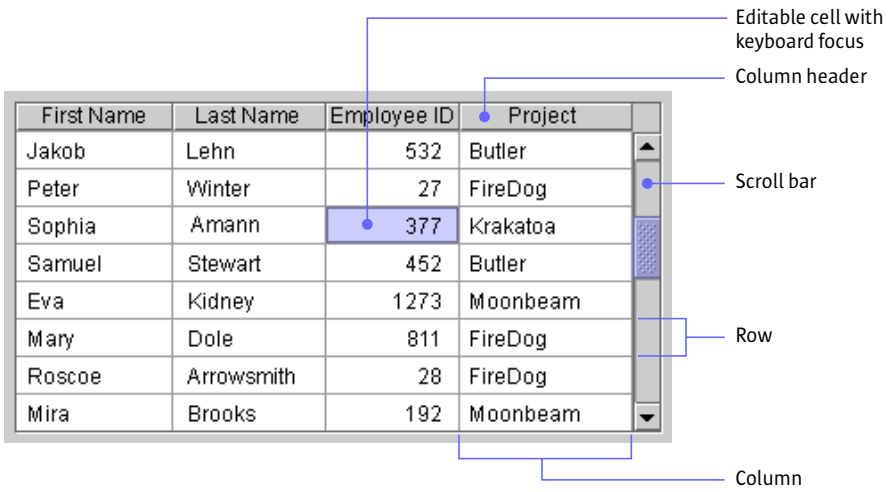
- **Multiple Range.** Users can select a single item, a range of items, or multiple ranges of items (also known as discontinuous selection). Users select a single item by clicking it and extend the selection by Shift-clicking. Moving the pointer to another item and Control-clicking sets a new anchor point and toggles the selection of the current item (if the item is selected, it is deselected, and vice versa). This item gets keyboard focus. Shift-clicking extends the range.

Tables

A **table** organizes related information into a series of rows and columns. Each field in the table is called a "cell". By default, a cell contains a text field, but it might also contain other JFC components, such as a checkbox or combo box. A cell might also contain a graphic. The cell that has keyboard focus has an inner border.

The following figure illustrates the use of table to display the records of employees in a company database. The cell "337" is selected and has keyboard focus.

FIGURE 151 Sample JFC Table



The background color of a cell depends on whether the cell is selected, whether the cell is editable, and the background color of the table. The following table shows how a cell gets its background color.

TABLE 11 Background Color of Table Cells

Type of Cell	Background Color
A cell that is selected	The primary 3 color defined in the color theme. For information on color themes in the Java look and feel, see "Colors" on page 35.
A cell that is not selected	The background color of the table, which is white by default.
The cell that has keyboard focus	White if the cell is editable, and primary 3 if it is not. The inner border is drawn in primary 1.

Users can select and edit a cell if the component in that cell supports editing. For example, if a cell contains a text field, users can insert, cut, copy, and paste text. For more information on editing the text in a table, see "Noneditable Text Fields" on page 162. For the keyboard operations that are appropriate for tables, see Table 26 on page 193.

Table Appearance

You have several options for defining the appearance of your table. You can turn on the display of horizontal and vertical lines that define the table cells, as shown in the preceding figure. You can set the horizontal and vertical padding around the content of a cell. You can also set the width of the columns.

You can automatically resize your table when the user resizes the window by placing the table in a window that was constructed with a layout manager.

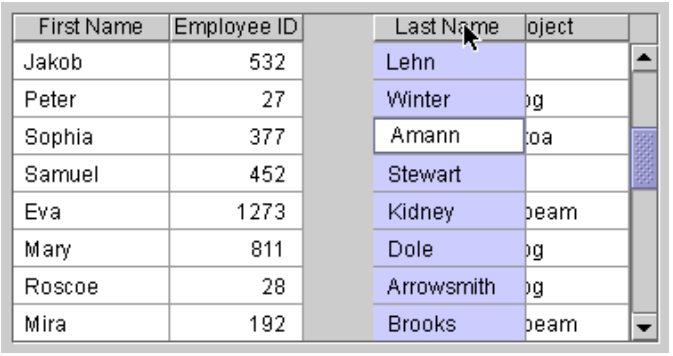
 When resizing a table vertically, make sure that it always displays a whole number of lines.

Table Scrolling Unlike a list, a table is not automatically provided with a scroll bar when its contents exceeds its visible area. You can provide a scroll bar for your table by placing it inside a scroll pane. A table has column headers only when it is in a scroll pane. For information on scroll panes, see "Scroll Panes" on page 99.

Column Reordering You can allow users to rearrange the columns in the table. When users drag the column header to the right or left, the entire column moves. Releasing the mouse button places the column at the new location.

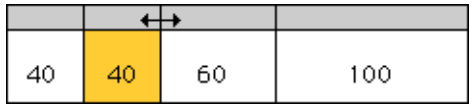
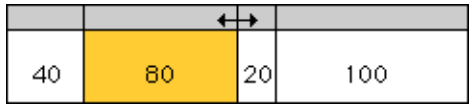
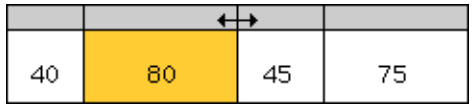
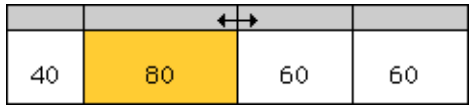
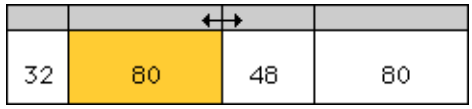
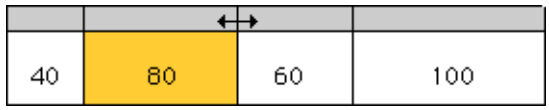
The following figure shows the Last Name column being dragged to the right. The column is selected (although that is not a requirement of moving a column).

FIGURE 152 Reordering Columns by Dragging the Column Header



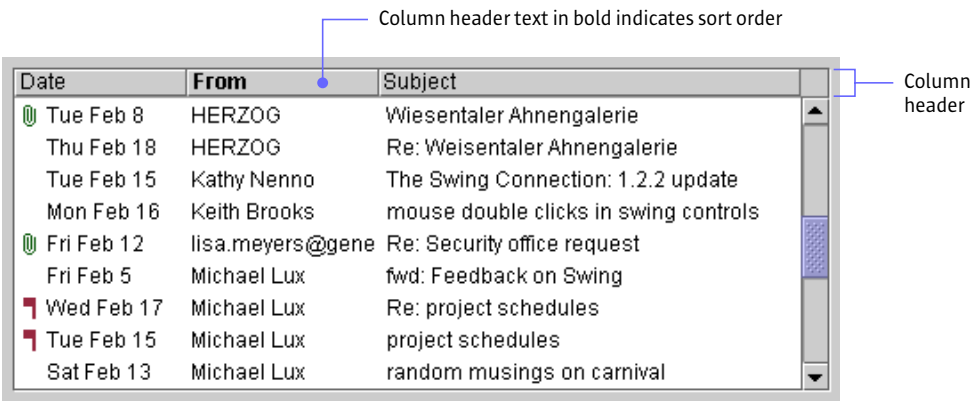
Column Resizing You can enable users to resize the columns in the table. Users drag the right border of the column header to the right to make the column wider, and to the left to make the column narrower. When the user resizes a column, you must decide whether to change the width of the entire table or adjust the other columns so the overall width is preserved. The five resize options are described and illustrated in the following table.

TABLE 12 Table Resize Options

Option	Description	Illustration
	The original table. The double arrow shows the west resize pointer before the columns are resized.	
Resize next	Resizes the columns on either side of the border being moved. One column becomes bigger, while the other becomes smaller.	
Resize subsequent	Resizes the column whose border was moved and all columns to its right. This option is the default option.	
Resize last	Resizes the column whose border was moved and the last (right-most) column.	
Resize all	Resizes all other columns, distributing the remaining space proportionately.	
Resize off	Resizes the column whose border was moved, and makes the table wider or narrower to adjust the space added or removed from the column. This is the only option that changes the overall width of the table.	

Row Sorting You can give users the ability to sort the rows in the table by clicking the column headers. An email application, which displays message headers in a table, is well suited for row sorting, as shown in the following figure. The messages can be sorted by date, sender, or subject. The header of the “From” column is in bold to indicate the messages are sorted by sender.

FIGURE 153 Row Sorting in an Email Application



Provide a visual indicator for the table column that currently determines the sort order. For example, put the column header text in bold.

If your application has a menu bar, provide row sorting as a set of menu items as well (for example, include “Sort by Sender” on the View menu).

Row sorting is not included with the table component. However, the JFC contains sample code that developers can use to implement row sorting. See the website `http://java.sun.com/docs/books/tutorial/uiswing/components/table.html#sorting` for more information.

Selection Models When designing a table, you must decide which objects (cells, rows, or columns) users can select. Although the JFC provides 24 models for selecting objects in tables, only nine are recommended for use in the Java look and feel:

- No selection
- Single cell
- Single range of cells
- Single row
- Single range of rows
- Multiple ranges of rows

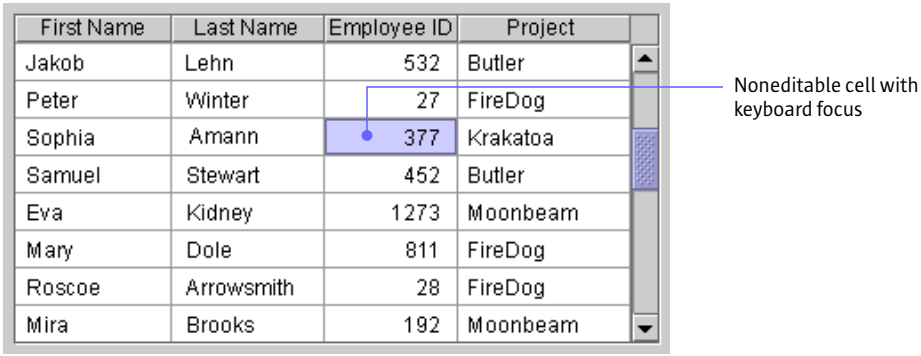
- Single column
- Single range of columns
- Multiple ranges of columns

No Selection You can turn off selection in the table. When users click in a cell, nothing is selected.

SIngle Cell You can enable users to select a cell by clicking it. The cell gets keyboard focus, which is indicated by an inner border. Any previous selection is deselected.

In the following figure, the cell 377 is selected and has keyboard focus. The cell cannot be edited, as indicated by the primary 3 background color.

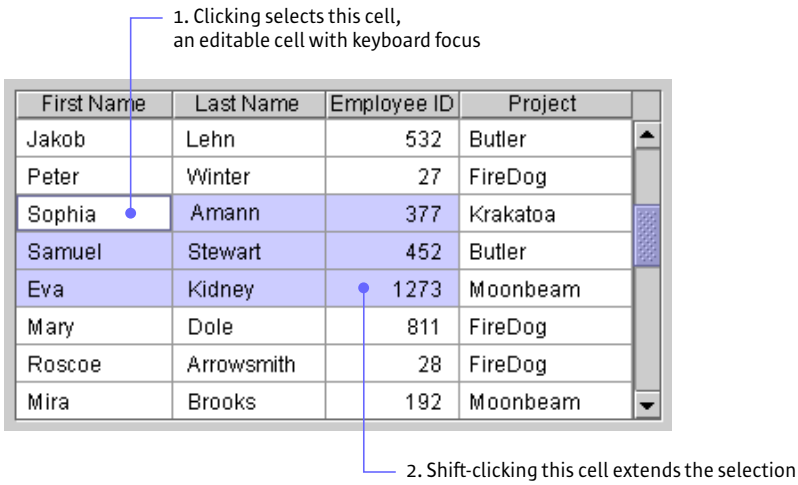
FIGURE 154 Single Cell Selection



Range of Cells You can enable users to select a single cell or a rectangular range of cells. Users select a cell by clicking it. The cell gets keyboard focus and becomes the anchor point of the selection. Users extend the selection by moving the pointer to a new cell and Shift-clicking. Users can also select a range of cells by dragging through the range.

In the following figure, the user selected the range by clicking Sophia, then Shift-clicking 1273. The cell containing Sophia is editable, as indicated by its white background.

FIGURE 155 Range of Cells

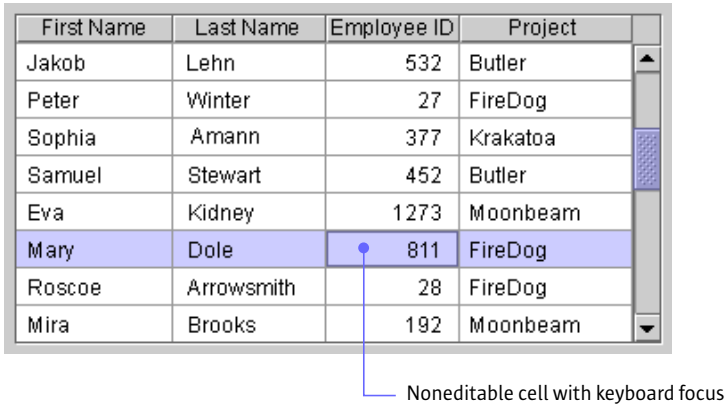


In range selection, the selection always extends from the cell with the anchor point to the cell where the user Shift-clicked. If users move the pointer within the selection and Shift-click, the selection becomes smaller.

Single Row You can enable users to select an entire row by clicking any cell in the row. The selected cell gets keyboard focus, which is indicated by an inner border. Any previous selection is deselected. Single row selection is well suited for tables that have multicolumn lists.

In the following figure, the user clicked the cell 811. This cell is not editable, as indicated by its background color.

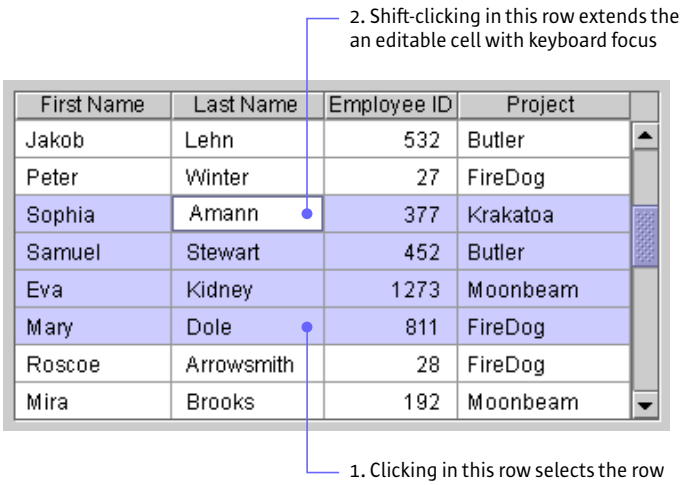
FIGURE 156 Single Row Selection



Single Range of Rows You can enable users to select one row or a range of rows. Users select a row by clicking any cell in the row. The cell that was clicked gets keyboard focus and becomes the anchor point of the selection. Users extend the selection by moving the pointer to a new row and Shift-clicking. Users can also select a range of rows by dragging through the range.

In the following figure, the user clicked Amann, then Shift-clicked Dole. The cell Amann is editable, as indicated by its white background.

FIGURE 157 Range of Rows



In range selection, the selection always extends from the row with the anchor point to the row where the user Shift-clicked. If users Shift-click within an existing selection, the selection becomes smaller.

An email application might use the singe range selection in a table that displays the message headers. Users can select one message header or a range of message headers (which is especially useful for deleting messages).

Multiple Ranges of Rows You can enable users to select a single row, a range of rows, or multiple row ranges (also known as discontinuous, discontiguous, or disjoint ranges). Users select a single row by clicking any cell in the row and extend the selection by Shift-clicking. To start another range, users Control-click any cell in the row. The cell gets keyboard focus and becomes the anchor point of the range. The selection of the row toggles as follows:

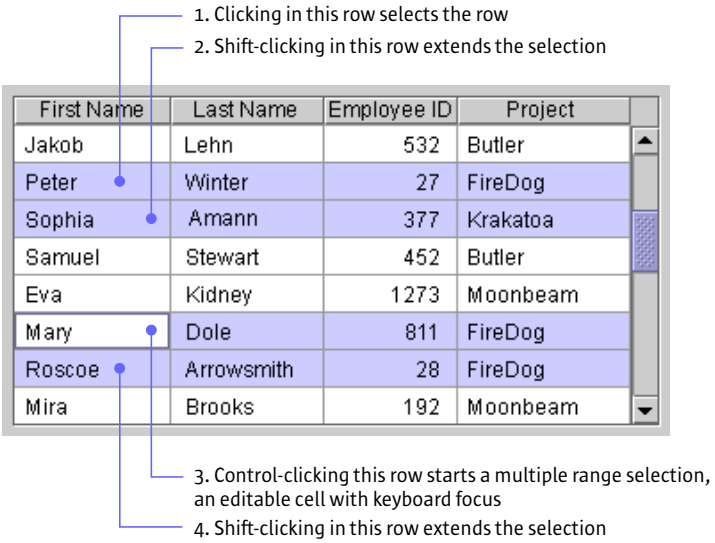
- If the row is not already selected, it is selected. A subsequent Shift-click selects all rows from the anchor point to the row where the user Shift-clicked.

- If the row is within an existing selection, the row is deselected. A subsequent Shift-click deselects all rows from the anchor point to the row where the user Shift-clicked.

Users can also select another range by dragging through the range while holding down the Control key.

In the following figure, the user selected the first range by clicking Peter, and then Shift-clicking Sophia. The user made another range by Control-clicking Mary, and then Shift-clicking Roscoe. The cell Mary has keyboard focus and is editable.

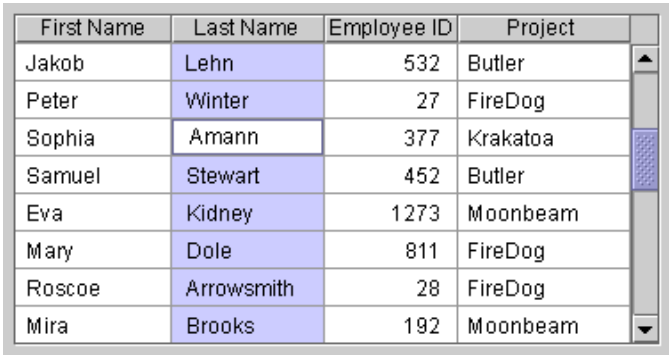
FIGURE 158 Multiple Row Ranges



Single Column Only You can enable users to select an entire column by clicking any cell in the column. The selected cell gets keyboard focus, which is indicated by an inner border. Any previous selection is deselected.

In the following figure, the user clicked Amann in the Last Name column. The white background indicates that the cell can be edited.

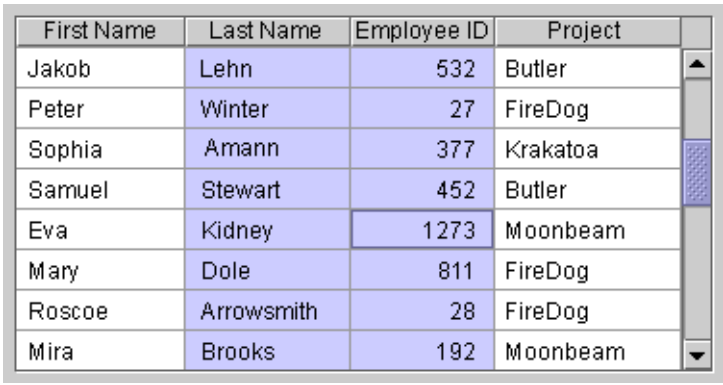
FIGURE 159 Single Column Selection



Single Range of Columns You can enable users to select one column or a range of columns. Users select a column by clicking any cell in the column. The cell that was clicked gets keyboard focus and becomes the anchor point of the selection. Users extend the selection by moving the pointer to a new column and Shift-clicking. Users can also select a range of columns by dragging through the range.

In the following figure, the user clicked 1273, and then Shift-clicked Lehn. The cell containing 1273 cannot be edited, as indicated by its background color.

FIGURE 160 Range of Columns



In range selection, the selection always extends from the column with the anchor point to the column where the user Shift-clicked. If users Shift-click within an existing selection, the selection becomes smaller.

Multiple Ranges of Columns You can enable users to select a single column, a range of columns, or multiple column ranges (also known as discontinuous, discontiguous, or disjoint ranges). Users select a single column by clicking any cell in the column and extend the selection by Shift-clicking. To start another range, users Control-click any cell in the column. The cell gets keyboard focus and becomes the anchor point of the range. The selection of the column toggles as follows:

- If the column is not already selected, it is selected. A subsequent Shift-click selects all columns from the anchor point to the column where the user Shift-clicked.
- If the column is within an existing selection, the column is deselected. A subsequent Shift-click deselects all columns from the anchor point to the column where the user Shift-clicked.

Users can also select another range by dragging through the range while holding down the Control key.

In the following figure, the user clicked Jakob, and then Shift-clicked Lehn. The user selected another range by Control-clicking Krakatoa, which has keyboard focus and can be edited, as indicated by its white background.

FIGURE 161 Multiple Column Ranges

First Name	Last Name	Employee ID	Project
Jakob	Lehn	532	Butler
Peter	Winter	27	FireDog
Sophia	Amann	377	Krakatoa
Samuel	Stewart	452	Butler
Eva	Kidney	1273	Moonbeam
Mary	Dole	811	FireDog
Roscoe	Arrowsmith	28	FireDog
Mira	Brooks	192	Moonbeam

Tree Views

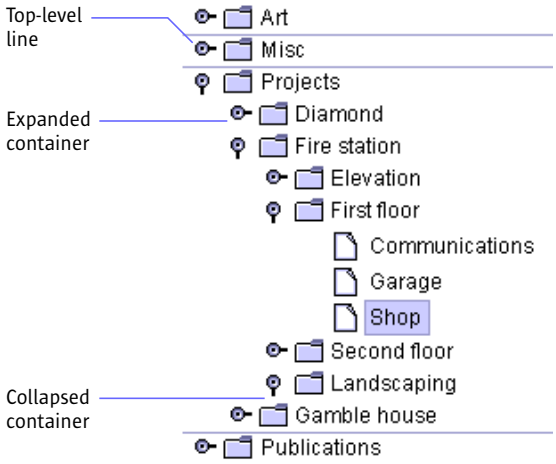
A **tree view** represents a set of hierarchical data in the form of an outline, which users can expand and collapse. Tree views are useful for displaying data such as the folders in a file system or the table of contents in a help system.

A tree view is made up of nodes. The top-level node, from which all other nodes branch, is the root node. Nodes that might have subnodes are called “containers”. All other nodes are called “leaves.” The default graphic for a container is a folder icon and the default graphic for a leaf is a file icon. Each node also has a label, which describes its contents.

Turners appear next to each container in the tree view. The turner points right when the container is collapsed and down when the container is expanded. By default, turners are not displayed for the highest level nodes.

In the following figure, the nodes “Projects,” “Fire station,” “First floor,” and “Landscaping” are expanded containers; all the other containers are collapsed. “Landscaping” is a container without subnodes. “Communications,” “Garage,” and “Shop” are leaves. The turner, container, and leaf graphics are the default graphics provided in the Java look and feel.

FIGURE 162 Tree View with Top-level Lines



Users click the right-pointing turner to expand a container so that its branch of containers and leaves are visible in the tree view. The turner rotates to point downward. Clicking the downward-pointing turner collapses a container so that its branch is no longer visible.

Users can edit the data in a node. A tree view is provided with a text editor that allows users to insert, cut, copy, and paste text. For more info on editing the text in a node, see "Editable Text Fields" on page 162. For the keyboard operations that are appropriate for tree views, see "Tree Views" on page 197.

In most cases, display the second level of the hierarchy as your highest level. Your outline will be easier to use if you do not display the root node.

Display turners for all nodes in the tree, including the nodes at the highest-level.

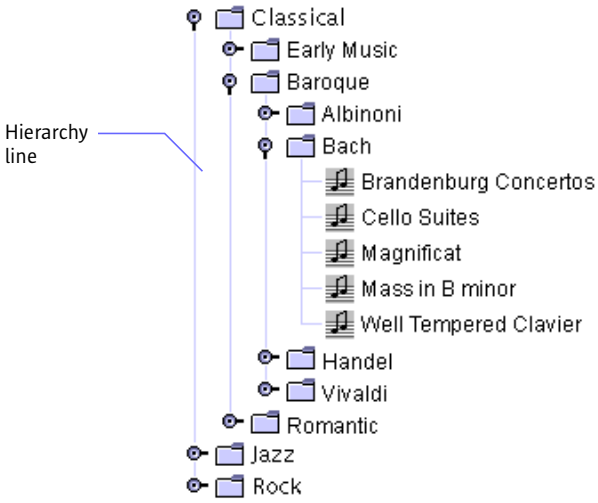
Setting `rootVisible` to false turns off the display of the root node.

Setting `showsRootHandles` to true turns on the display of turners for the highest level nodes

Lines in Tree Views

You have three options for including lines in the tree view. The first option is not to include any lines. The second option is to draw lines that separate the top-level nodes, as shown in the preceding figure. The third option is to draw lines that define the hierarchical relationships of the nodes, as shown in the following figure.

FIGURE 163 Tree View with Hierarchy Lines



Setting the client property `JTree.lineSyle` to `None` displays no lines, to `Horizontal` displays top-level lines, to `Angled` displays hierarchy lines.

Graphics in Tree Views You can substitute your own graphics for the container and leaf node graphics. For example, if your hierarchy represents the nodes and servers in a network, you might include graphic representations of the nodes and servers. You might also use separate graphics to show when a container is expanded and when it is collapsed. In the preceding figure, a custom music graphic is used for the leaf nodes.

A: KEYBOARD NAVIGATION, ACTIVATION, AND SELECTION

This appendix defines the keyboard operations that enable users to navigate through, activate, or select the JFC components described in this book. Navigating means to move the input focus from one user interface component to another. Activating refers to operating the component. Selecting means to designate where the next action will take place. For an overview of these concepts, see "Keyboard Navigation and Activation" on page 83.

In general, navigating between components follows these rules.

- **Tab or Ctrl-Tab.** Moves keyboard focus to the next component or to the first member of the next group of components. Use Ctrl-Tab when the component itself accepts tabs, as in text fields, tables, and tabbed panes.
- **Shift-Tab.** Moves keyboard focus to the previous component or to the first component in the previous group of components.
- **Arrow keys.** Moves keyboard focus within the individual components of a group of components, for example, within menu items in a menu or within radio buttons in a group of radio buttons.

This appendix presents the keyboard operations in a series of tables, arranged alphabetically by component. The left column of each table describes an action (for example, move focus to the left) and the right column describes its keyboard operation (for example, left arrow key). Some actions list more than one keyboard operation. For example, both Home and Ctrl-Home move to the beginning of a list. In this case, implement the keyboard operation appropriate for your environment.

 Ensure that your application can be operated solely from the keyboard. Unplug your mouse to verify this when you test the application.

 Some of the keyboard operations described in the following tables may be temporarily incomplete or not implemented. However, you should reserve these key sequences for future versions of JFC and the JDK.

Checkboxes

The following table lists the keyboard operations for checkboxes. For more information on this component, see "Checkboxes" on page 148.

TABLE 13 Keyboard Operations for Checkboxes

Action	Keyboard Operation
Navigates within checkbox group	Arrow keys
Selects or deselects checkbox	Spacebar

Combo Boxes

The following table lists the keyboard operations for combo boxes. For details on this component, see "Combo Boxes" on page 153.

TABLE 14 Keyboard Operations for Combo Boxes

Action	Keyboard Operation
Posts associated list	Spacebar, down arrow, Alt-down arrow
Closes associated list	Escape
Chooses highlighted item and closes list	Enter or Return, spacebar
Moves highlight within list	Up arrow, down arrow

Command Buttons

The following table lists the keyboard operations for command buttons. For more information on this component, see "Command Buttons" on page 142.

TABLE 15 Keyboard Operations for Command Buttons

Action	Keyboard Operation
Activates command button	Spacebar
Activates default button (does not require keyboard focus)	Enter or Return
Activates Cancel button (does not require keyboard focus)	Escape

Desktop Panes and Internal Frames

The following table lists the keyboard operations for desktop panes and internal frames. For details on internal frames, see "Internal Frames" on page 105. For a discussion of minimized windows for internal frames, see "Minimized Internal Frames" on page 106. For a discussion of MDI applications, see "Working With Multiple Document Interfaces" on page 105.

TABLE 16 Keyboard Operations for Desktop Panes and Internal Frames

Action	Keyboard Operation
Opens internal frame	Ctrl-F5
Closes internal frame	Ctrl-F4
Moves internal frame	Ctrl-F7
Resizes internal frame	Ctrl-F8
Minimizes internal frame	Ctrl-F9
Navigates first between open internal frames, then among minimized internal frames	Ctrl-Esc, Ctrl-Tab, Shift-Esc, Shift-Tab
Opens minimized internal frame that has keyboard focus	Ctrl-F5, Enter, Return
Navigates among associated windows on the desktop pane	Ctrl-F6, Shift-Ctrl–F6
Navigates between associated windows when an internal frame creates a secondary window	Ctrl-F6, Shift-Ctrl-F6
Displays desktop contextual menu	Ctrl-spacebar

Dialog Boxes

The following table lists the keyboard operations for dialog boxes, alert boxes, and utility windows. See Chapter 8, "Dialog Boxes," for a comprehensive treatment of these components.

TABLE 17 Keyboard Operations for Dialog Boxes

Action	Keyboard Operation
Navigates into dialog box	Alt-F6
Navigates out of dialog box	Alt-F6
Activates Cancel button	Escape
Activates default command button	Enter or Return, spacebar

HTML Panes

HTML panes use the navigation, selection, and activation sequences described in Table 27 on page 194, plus the two listed here. For details on the appearance and behavior of this component, see "HTML Editor Kits" on page 167.

TABLE 18 Keyboard Operations for HTML Panes

Action	Keyboard Operation
Navigates to link and other focusable elements	Tab, Shift-Tab, Ctrl-Tab, Shift-Ctrl-Tab
Activates link	Enter or Return, spacebar

Lists

The actions listed in the following table assume multiple selection. For more information on selection, see "Clicking and Selecting Objects" on page 77. See "Lists" on page 169 for details on the appearance and behavior of lists.

TABLE 19 Keyboard Operations for Lists

Action	Keyboard Operation
Moves focus up one row or line	Up arrow
Moves focus down one row or line	Down arrow
Moves focus up one view minus one line, giving focus to the first line in the view	Page Up
Moves focus down one view minus one line, giving focus to the first line in the view	Page Down
Moves to beginning of list	Home, Ctrl-Home
Moves to end of list	End, Ctrl-End
Selects all items in the list	Ctrl-A, Ctrl-/
Deselects all	Ctrl-/
Makes a selection (and deselects any previous selection)	Spacebar
Toggles selection (and does not affect previous selections)	Ctrl-spacebar
Extends selection	Shift-spacebar
Extends selection down one item	Shift-down arrow
Extends selection up one item	Shift-up arrow
Extends selection to beginning of list	Shift-Home
Extends selection to the end of list	Shift-End
Extends selection up one view	Shift-PgUp
Extends selection down one view	Shift-PgDn

Menus

The keyboard operations in this table appy to menu bars, menus, drop-down menus, submenus, contextual menus, menu items, radio button menu items and checkbox menu itemss. For a discussion of menus, see Chapter 9.

TABLE 20 Keyboard Operations for Menus

Action	Keyboard Operation
Posts the current menu	Enter or Return, spacebar, arrow keys
Dismisses a menu without taking action and returns focus to the last component that had focus	Escape
Moves focus to the menu bar and posts first menu	F10
Navigates within menus	Arrow keys
Navigates between titles in the menu bar	Arrow keys
Activates a menu item	Enter or Return, spacebar (dismisses menu and goes to last window item with focus)
Displays contextual menu	Shift-F10
Dismisses contextual menu	Escape
Navigates within contextual menu	Arrow keys
Activates highlighted item in contextual menu and dismisses menu	Enter or Return, spacebar

Radio Buttons

The following table lists the keyboard operations for radio buttons. For a discussion of the appearance and behavior of this component, see “Radio Buttons” on page 151.

TABLE 21 Keyboard Operations for Radio Buttons

Action	Keyboard Operation
Selects radio button	Spacebar

Scrollbars

Keyboard navigation in the JFC does not apply to the scrollbar; it applies to the component tp which the scrollbar is attached, for instance, a formatted text pane. For a discussion of the appearance and behavior of scrollbars, see "Scrollbars" on page 100.

TABLE 22 Keyboard Operations for Scrollbars

Action	Keyboard Operation
Navigates within a scrollbar	Arrow keys
Moves up one view	Page Up
Moves down one view	Page Down
Moves to beginning of data	Ctrl-Home
Moves to end of data	Ctrl-End
Moves right one view minus one line	Ctrl-PgDn
Moves left one view	Ctrl-Pg Up

Sliders

The following table lists the keyboard operations for sliders. For details on this component, see "Sliders" on page 156.

TABLE 23 Keyboard Operations for Sliders

Action	Keyboard Operation
Changes value of slider	Arrow keys
Moves to the left/top value	Home
Moves to the right/bottom value	End
Jumps in the left/top direction (approximately 20% of the scale)	Page Up, Ctrl-PgUp
Jumps in the right/bottom direction (approximately 20% of the scale)	Page Down, Ctrl-PgDn

Split Panes

The following table lists the keyboard operations for split panes. After users enter a split pane, pressing Tab cycles the focus to the components within the split pane. For a description of the appearance and behavior of this component, see "Split Panes" on page 103.

TABLE 24 Keyboard Operations for Split Panes

Action	Keyboard Operation
Navigates between split panes and gives focus to the last element that had focus	Tab, F6
Gives focus to the splitter bar	F8
Changes size and moves the splitter appropriately	Arrow keys, Home, End

Tabbed Panes

The following table lists the keyboard operations for tabbed panes. For a description of the appearance and behavior of this component, see "Tabbed Panes" on page 101. When a tabbed pane initially gets focus, the focus goes to one of the tabs, and not to one of the content panes.

TABLE 25 Keyboard Operations for Tabbed Panes

Action	Keyboard Operation
Navigates through the tabs	Arrow keys
Moves from the tab to its associated content pane	Ctrl-down arrow
Moves from the content pane to its associated tab	Ctrl-up arrow
Moves to next or previous content pane	Ctrl-PgDn or Ctrl-PgUp

Tables

The following table lists the keyboard operations for tables. For a description of the appearance and behavior of this component, see"Tables" on page 171.

TABLE 26 Keyboard Operations for Tables

Action	Keyboard Operations
Moves focus one cell up	Shift-Return
Moves focus one cell down	Return
Moves focus one cell to the left	Shift-Tab
Moves focus one cell to the right	Tab
Deselects current selection and moves focus one cell up	Up arrow
Deselects current selection and moves focus one cell down	Down arrow
Scrolls up one view and gives focus to the first visible cell in the current column	Page Up
Scrolls down one view and gives focus to the first visible cell in the current column	Page Down
Scrolls left one view and gives focus to the first visible cell in the current row	Ctrl-PgUp
Scrolls right one view and gives focus to the first visible cell in the current row	Ctrl-PgDn
Moves focus and view to first cell in the current row	Home
Moves focus and view to last cell in the current row	End
Moves focus and view to first cell in current column	Ctrl-Home
Moves focus and view to last cell in current column	Ctrl-End
Allows editing in a cell without overwriting the information	F2
Resets the cell to the state it was in before it was edited	Esc
Selects the entire table	Ctrl-A

TABLE 26 Keyboard Operations for Tables *(Continued)*

Action	Keyboard Operations
Extends the selection up one row	Shift-up arrow
Extends the selection down one row	Shift-down arrow
Extends the selection one column to the left	Shift-left arrow
Extends the selection one column to the right	Shift-down arrow
Extends the selection to the beginning of the row	Shift-Home
Extends the selection to the end of the row	Shift-End
Extends the selection to the beginning of the column	Ctrl-Shift-Home
Extends the selection to the end of the colulmn	Ctrl-Shift-End
Extends the selection up one view	Shift-PgUp
Extends the selection down one view	Shift-PgDn
Extends the selection left one view	Ctrl-Shift-PgUp
Extends the selection right one view	Ctrl-Shift-PgDn

Text Areas and Formatted Text Panes

The following table lists the keyboard operations for text areas and formatted text panes. For details on the appearance and behavior of these components, see "Text Areas" on page 165 and "Styled Text Editor Kits" on page 166.

TABLE 27 Keyboard Operations for Text Areas and Formatted Text Panes

Action	Keyboard Operation
Moves insertion point up one line	Up arrow
Moves insertion point down one line	Down arrow
Moves insertion point left one component or character	Left arrow
Moves insertion point right one component or character	Right arrow
Moves up one view	Page Up

TABLE 27 Keyboard Operations for Text Areas and Formatted Text Panes *(Continued)*

Action	Keyboard Operation
Moves down one view	Page Down
Moves left one view	Ctrl-Page Up
Moves right one view	Ctrl-Page Down
Moves to beginning of line	Home
Moves to end of row or line	End
Moves to beginning of data	Ctrl-Home
Moves to end of data	Ctrl-End
Moves to next word	Ctrl-right arrow
Moves to previous word	Ctrl-left arrow
Selects all	Ctrl-A, Ctrl-/
Deselects all	Ctrl-\
Extends selection up	Shift-up arrow
Extends selection down	Shift-down arrow
Extends selection left	Shift-left arrow
Extends selection right	Shift-right arrow
Extends selection up one view	Shift-PgUp
Extends selection down one view	Shift-PgDown
Extends selection to the left one view	Ctrl-Shift-PgUp
Extends selection to the right one view	Ctrl-Shift-PgDn
Extends selection to beginning of line	Shift-Home
Extends selection to end of line	Shift-End
Extends selection to beginning of data	Ctrl-Shift-Home
Extends selection to end of data	Ctrl-Shift-End
Extend selections to next word	Ctrl-Shift-right arrow
Extends selection to previous word	Ctrl-Shift-left arrow

Text Fields

The following table lists the keyboard operations for text fields. For details on this component, see "Text Fields" on page 162.

TABLE 28 Keyboard Operations for Text Fields

Action	Keyboard Operation
Moves insertion point one character to the right	Right arrow
Moves insertion point one character to the left.	Left arrow
Moves insertion point to the beginning of the next word	Ctrl-right arrow
Moves to the beginning of the previous word	Ctrl-left arrow
Moves insertion point to the beginning of the field	Home
Moves insertion point to the end of the field	End
Submits text entry	Enter or Return
Extends selection to the beginning of the line	Shift-Home
Extends selection to the end of line	Shift-End
Extends selection one character to the left	Shift-left arrow
Extends selection one character to the right	Shift-right arrow
Extends selection to the next word	Shift-Ctrl-right arrow
Extends selection to the previous word	Shift-Ctrl-left arrow

Toggle Buttons

The following table lists the keyboard operations for toggle buttons. For details on this component, see "Toggle Buttons" on page 147.

TABLE 29 Keyboard Operations for Toggle Buttons

Action	Keyboard Operation
Toggles button on or off	Spacebar

Toolbars

The following table lists the keyboard operations for toolbars. For details on the appearance and behavior of this component, see "Toolbars" on page 134.

TABLE 30 Keyboard Operations for Toolbars

Action	Keyboard Operation
Navigates within toolbar	Arrow keys
Activates toolbar	Enter, Spacebar

Tool Tips

The following table lists the keyboard operations for tool tips. For details on this component, see "Tool Tips" on page 138.

TABLE 31 Keyboard Operations for Toootips

Action	Keyboard Operation
Display tool tip	Ctrl-F1
Remove tool tip	Esc, Ctrl-F1

Tree Views

The following table lists the keyboard operations for tree views. For details on the appearance and behavior of this component, see "Tree Views" on page 182.

TABLE 32 Keyboard Operations for Tree Views

Action	Keyboard Operation
Expands current node	Right arrow
Contracts current node	Left arrow
Moves focus up one node	Up arrow
Moves focus down one node	Down arrow
Moves focus to the first node in tree	Home
Moves focus to the last node in the tree	End
Moves up one view	Page Up
Moves down one view	Page Down

TABLE 32 Keyboard Operations for Tree Views *(Continued)*

Action	Keyboard Operation
Moves left one view	Ctrl-PgUp
Moves right one view	Ctrl-PgDn
Selects all nodes in the tree view	Ctrl-A, Ctrl-/
Deselects all	Ctrl-\
Extends selection down	Shift-down arrow
Extends selection up	Shift-up arrow
Extends selection to beginning of tree	Shift-Home
Extends selection to end of tree	Shift-End
Extends selection up one view	Shift-PgUp
Extends selection down one view	Shift-PgDn
Extends selection right one view	Ctrl-Shift-PgDn
Extends selection left one view	Ctrl-Shift-PgUp

GLOSSARY

Abstract Window Toolkit

The class library that provides the standard API for building GUIs for Java programs. The Abstract Window Toolkit (AWT) includes imaging tools, data transfer classes, GUI components, containers for GUI components, an event system for handling system and user events among parts of the AWT, and layout managers for managing the size and position of GUI components in platform-independent designs. The GUI components in the AWT are implemented as native-platform versions of the components, and they have largely been supplanted by the Swing components. See also **Java Foundation Classes**, **Swing classes**.

accessibility

The degree to which software can be comfortably used by a wide variety of people, including those who require assistive technologies like screen magnifiers or voice recognition. An accessible JFC application employs the Java Accessibility API, enables its users to select an appropriate look and feel, and provides keyboard operations for all actions that can be carried out by use of the mouse. See also **Java Accessibility API**, **Java Accessibility Utilities**, **keyboard operations**.

alert box

A dialog box used by an application to convey a message or warning or to gather information from the user. Four standard alert boxes (Question, Info, Error, and Warning) are supplied for JFC applications. Alert boxes are created using the `JOptionPane` component. See also **dialog box**.

applet

A program, written in the Java language, that a user can interact with in a web browser. See also **application**.

application

A program that combines all the functions necessary for a user to accomplish a particular set of tasks (for instance, word processing or inventory tracking). Unless stated otherwise, this book uses “application” to refer to both applets and standalone applications. See also **applet**.

assistive technology Hardware or software that helps people with disabilities use a computer (or provides alternative means of use to all users). Examples include pointing devices other than the mouse, audio or text-only browsers, and screen readers that translate the contents of the screen into Braille, voice output, or audible cues.

AWT See **Abstract Window Toolkit**.

bit depth The amount of information (in bits) per pixel used to represent a graphic object. A bit depth of 8 supports up to 256 colors; a bit depth of 24 can accommodate up to 16, 777, 216 colors.

browser An application that enables users to view, navigate through, and interact with HTML documents and applets. Also called a "web browser."

button A collective term for the various controls whose on-screen appearance typically simulates a push button or a radio button. The user clicks buttons to specify commands or set options. See also **checkbox**, **command button**, **radio button**, **toggle button**, **toolbar button**.

checkbox A control, with associated text, that a user clicks to select or deselect an option. A check mark in the checkbox graphic indicates that the option is selected. Checkboxes are created using the `JCheckBox` component. See also **radio button**.

checkbox menu item A menu item that appears with a checkbox next to it to represent an on or off setting. A check mark in the checkbox graphic indicates that the value associated with that menu item is selected. Checkbox menu items are created using the `JCheckBoxMenuItem` component. See also **menu item**.

color chooser A component that enables a user to select a color. Color choosers are created using the `JColorChooser` component. See also **HSB**, **palette window**, **RGB**, **utility window**.

combo box A component with a drop-down arrow that the user clicks to display a list of options. Noneditable combo boxes (sometimes called "list boxes") have a list from which the user can select one item. Editable combo boxes offer a text field as well as a list of options. The user can make a selection by typing a value in the text field or by selecting an item from the list. Combo boxes are created using the `JComboBox` component.

command button A button with a rectangular border that contains text, a graphic, or both. A user clicks a command button to specify a command that initiates an action. Command buttons are created using the `JButton` component. See also **button**, **toggle button**, **toolbar button**.

component A piece of code or, by extension, the interface element implemented by that code. See also **Swing classes**.

container A component (like an applet, window, pane, or internal frame) that holds other components.

contextual menu A menu that is displayed when a user presses mouse button 2 while the pointer is over an object or area associated with that menu. A contextual menu offers only menu items that are applicable to the object or region at the location of the pointer. Sometimes called a "pop-up menu." Contextual menus are created using the `JPopupMenu` component. See also **menu**.

control An interface element that a user can manipulate to perform an action, select an option, or set a value. Examples include buttons, sliders, and combo boxes. (A progress bar, by contrast, is not considered a control because a user cannot change its setting.)

cross-platform Pertaining to heterogeneous computing environments. For example, a cross-platform application is one that has a single code base for multiple operating systems.

cursor See **pointer**.

default button The command button that the application activates if a user presses Enter or Return. Default buttons in Java look and feel applications have a heavier border than other command buttons. See also **command button**.

designer A professional who specifies the way that users will interact with an application, chooses the interface components, and lays them out in a set of views. The designer might or might not be the same person as the developer who writes the application code.

desktop pane A container, a sort of virtual desktop, for an MDI application. Desktop panes are created using the `JDesktopPane` component. See also **internal frame**, **MDI**.

dialog box A secondary window displayed by an application to gather information from users or to inform them of a condition. A dialog box can contain panes, lists, buttons, and other controls. Dialog boxes are created using the `JDialog` component. See also **alert box**, **color chooser**, **file chooser**, **palette window**, **secondary window**, **utility window**.

dithering Creating the illusion of additional shades or colors in a displayed graphic by varying the density, colors, and patterns of on-screen dots.

drag To move the mouse while holding down a mouse button. See also **drag and drop**.

drag and drop To drag an interface element in order to move, copy, or link it to a new location. See also **drag**.

drop-down arrow The triangle indicator that a user clicks to view more options than are visible on screen—such as the list attached to a combo box or the options provided by some toolbar buttons.

drop-down menu A menu that appears when a user chooses a menu title in the menu bar. Drop-down menus are created using the `JMenu` component. See also **menu**, **menu bar**.

editor pane A component that supports a variety of plug-in editors. The Java look and feel can display formatted, HTML, and RTF text. Editor panes are created using the `JEditorPane` component.

file chooser A component that enables a user to save or select files (for example, after choosing Open or Save As in the File menu). File choosers typically contain a list of files and directories, text fields for typing file names, and Open or Save As buttons. File choosers are created using the `JFileChooser` component.

flush 3D style In the Java look and feel, the effect created by rendering on-screen graphics whose surfaces appear to be in the same plane as the surrounding canvas.

GIF Graphics Interchange Format. An 8-bit graphics format developed by CompuServe and commonly used on the World Wide Web. GIF files are limited to 256 colors, and they compress without loss of information. The GIF format is typically used for graphics in the Java look and feel. See also **bit depth**, **JPEG**.

HSB For "hue, saturation, brightness." In computer graphics, a color model that represents colors as the addition of black to other colors. In the HSB color model, hue refers to a color's pigment (its light frequency), saturation is the amount of pigment in the hue, and brightness is the amount of black in the color. See also **RGB**.

icon An on-screen graphic representing an interface element that a user can select or manipulate, like an application, document, or disk.

insertion point The place, usually indicated by a blinking bar, where typed text or a dragged or pasted selection will appear. See also **pointer**.

internal frame A container used in MDI applications to create windows that a user cannot drag outside of the desktop pane. In an MDI application that uses the Java look and feel, internal frames have a window border, title bar, and standard window controls with the Java look and feel. Internal frames are created using the `JInternalFrame` component. See also **desktop pane**, **MDI**.

internationalization The process of preparing software that is suitable for the global marketplace, taking into account wide variations in regions, languages, and cultures. Internationalization usually requires the separation of component text from code to ease the process of translation. See also **localization**.

Java 2D API A programming interface (part of the Java Foundation Classes in JDK 1.2) that provides an advanced two-dimensional imaging model for complex shapes, text, and images. Features include enhanced font and color support and a single, comprehensive rendering model. See also **Java Foundation Classes**.

Java Accessibility API A programming interface (part of the Java Foundation Classes) that enables assistive technologies to interact and communicate with JFC components. A Java application that fully supports the Java Accessibility API is compatible with such technologies as screen readers and screen magnifiers. See also **accessibility**, **assistive technology**, **Java Accessibility Utilities**, **Java Foundation Classes**.

Java Accessibility Utilities A set of classes (provided in JDK 1.2) for use by the vendors who create assistive technologies or automated tool tests. See also **accessibility**, **assistive technology**, **Java Accessibility API**, **Java Foundation Classes**.

Java Development Kit Software that includes the APIs and tools that developers need to write, compile, debug, and run Java applets and applications. Also called "JDK."

Java Foundation Classes A product that includes the Swing classes, pluggable look and feel designs, and the Java Accessibility API (all implemented without native code and compatible with JDK 1.1). In JDK 1.2, the Java Foundation Classes (JFC) also include the Java 2D API, drag and drop, and other enhancements. See also **Abstract Window Toolkit**, **pluggable look and feel architecture**, **Swing classes**.

Java look and feel The default appearance and behavior for JFC applications, designed for cross-platform use. The Java look and feel works in the same way on any platform that supports the Java Foundation Classes. See also **Java Foundation Classes**, **pluggable look and feel architecture**.

JDK See **Java Development Kit**.

JFC See **Java Foundation Classes**.

JFC application An application built with the Java Foundation Classes. See also **Java Foundation Classes**.

JPEG A graphics format developed by the Joint Photographic Experts Group. The JPEG format is frequently used for photographs and other complex images that benefit from a larger color palette than a GIF image can provide. JPEG compression is "lossy"; decompressed images are not identical to uncompressed images. See also **GIF**.

keyboard focus The active window or component, where the user's next keystrokes will take effect. Sometimes called the "input focus."

keyboard operations A collective term for keyboard shortcuts, mnemonics, and other forms of navigation and activation that utilize the keyboard instead of the mouse. See also **keyboard shortcut**, **mnemonic**.

keyboard shortcut A keystroke combination (usually a modifier key and a character key, like Control-C, or a function key, like F1) that activates a menu item from the keyboard even if the relevant menu is not currently displayed. See also **keyboard operations**, **mnemonic**.

label Static text that appears in the interface. For example, a label might identify a group of checkboxes. (The text that identifies each checkbox within the group, however, is specified in the individual checkbox component and is therefore not considered a label.) Labels are created using the `JLabel` component.

layout manager An object that assists the designer in determining the size and position of components within a container. Each container type has a default layout manager.

lightweight GUI component An interface element that is implemented entirely in Java code, without requiring the use of the equivalent component from a native toolbox. See also **Swing classes**.

list A set of choices from which a user can select one or more items. Items in a list can be text, graphics, or both. If the list of items is longer than the space allocated for the component, a vertical scrollbar appears. Lists are created using the `JList` component. See also **combo box**.

localization The process of customizing software for a particular locale. Localization usually involves translation and often requires changes to fonts, keyboard usage, and date and time formats. See also **internationalization**.

look and feel The appearance and behavior of a complete set of GUI components. See also **Java look and feel**.

MDI Multiple document interface. An interface that confines all of an application's primary windows inside its desktop pane. See also **desktop pane**.

menu A list of choices (menu items) logically grouped and displayed by an application so that a user need not memorize all available commands or options. Menus in the Java look and feel are "sticky"—that is, they remain posted on screen after the user clicks the menu title. Menus are created using the `JMenu` component. See also **contextual menu**, **drop-down menu**, **menu bar**, **menu item**, **submenu**.

menu bar The horizontal strip at the top of a window that contains the titles of the application's drop-down menus. Menu bars are created using the `JMenuBar` component. See also **drop-down menu**.

menu item A choice in a menu. Menu items (text or graphics) are typically commands or other options that a user can select. Menu items are created using the `JMenuItem` component. See also **checkbox menu item**, **radio button menu item**.

middle mouse button The central button on a three-button mouse (typically used in UNIX environments). The Java look and feel does not utilize the middle mouse button. See also **mouse button 2**.

MIME Multipurpose Internet Mail Extensions. An Internet standard for sending and receiving non-ASCII email attachments (including video, audio, and graphics). Web browsers also use MIME types to interpret and display files that are not written in HTML.

minimized window A reduced representation of an internal frame in an MDI application. Minimized windows look like horizontally oriented tags that appear at the lower-left corner of the desktop. The user can drag minimized windows to rearrange them.

mnemonic An underlined letter, typically in a menu title, menu item, or the text of a button or component. A mnemonic reminds the user how to activate the equivalent command by pressing the Alt key and the character key that corresponds to the underlined letter. See also **keyboard focus**, **keyboard operations**, **keyboard shortcut**.

modal dialog box In a JFC application, a dialog box that stops the user's interaction with the current application. Modal dialog boxes are created using the `JDialog` component. See also **dialog box**, **modeless dialog box**.

modeless dialog box In a JFC application, a dialog box whose presence does not prevent the user from interacting with the current application. Modeless dialog boxes are created using the `JDialog` component. See also **dialog box**, **modal dialog box**.

modifier key A key that does not produce a character but rather modifies the action of other keys—for example, the Control or the Shift key.

mouse button 1 The primary button on a mouse (the only button, for Macintosh users). By default, mouse button 1 is the leftmost button, though left-handed users might switch the button settings so that the rightmost button becomes mouse button 1. See also **middle mouse button**, **mouse button 2**.

mouse button 2 On a two-button or three-button mouse, the button that is used to display contextual menus. By default, mouse button 2 is the rightmost button on the mouse, though left-handed users might switch the settings so that the leftmost button becomes mouse button 2. See also **contextual menu**, **middle mouse button**, **mouse button 1**.

mouse-over feedback A change in the visual appearance of an interface element that occurs when the user moves the pointer over it—for example, the display of a button border when the pointer moves over a toolbar button.

multiple document interface See **MDI**.

Multiplexing look and feel A cross-platform look and feel that enables a designer to run more than one look and feel at a time—for example, the Java look and feel plus another look and feel customized for blind users. See also **look and feel**.

native code Code that refers to the methods of a specific operating system or is compiled for a specific processor.

palette window In an MDI application with the Java look and feel, a modeless window that displays a collection of tools, colors, or patterns. Palette windows float on top of document windows. User choices made in a palette window affect whichever primary window is active. Palette windows are created using the `JInternalFrame` component. See also **utility window**.

pane A collective term for scroll panes, split panes, and tabbed panes.

panel A container for organizing the contents of a window, dialog box, or applet. Panels are created using the `JPanel` component. See also **tabbed pane**.

password field A special text field in which the user types a password. The field displays a pound sign (#) for each typed character. Password fields are created using the `JPasswordField` component.

plain window A window with no title bar or window controls, typically used for splash screens. Plain windows are created using the `JWindow` component. See also **primary window**, **window controls**.

pluggable look and feel architecture An architecture that separates the implementation of interface elements from their presentation, enabling an application to dynamically choose how its interface elements interact with users. When a pluggable look and feel is used for an application, the designer can select from several look and feel designs.

plug-in editor An editor that can be plugged into the editor pane. The Java Foundation Classes supply plug-in editors for formatted, RTF, and HTML data.

pointer A small graphic that moves around the screen as the user manipulates the mouse (or another pointing device). Depending on its location and the active application, the pointer can assume various shapes, such as an arrowhead, crosshair, or clock. By moving the pointer and pressing mouse buttons, a user can select objects, set the insertion point, and activate windows. Sometimes referred to as the "cursor." See also **insertion point**.

preference A setting for an application or a tool. Users typically set preferences. See also **property**.

primary window A top-level window of an application, where the principal interaction with the user occurs. Primary windows always retain the look and feel of the user's native platform. Primary windows are created using the `JFrame` component. See also **dialog box**, **secondary window**.

progress bar An interface element that indicates one or more operations are in progress and shows the user what proportion of the operations has been completed. Progress bars are created using the `JProgressBar` component. See also **control**, **slider**.

property A characteristic of an object. Depending on the object, the user or the designer might set its properties. See also **preference**.

radio button A button that a user clicks to set an option. Unlike checkboxes, radio buttons are mutually exclusive—selecting one radio button deselects all other radio buttons in the group. Radio buttons are created using the `JRadioButton` component. See also **checkbox**.

radio button menu item A menu item that appears with a radio button next to it. Separators indicate which radio button menu items are in a group. Selecting one radio button menu item deselects all other radio button menu items in that group. Radio button menu items are created using the `JRadioButtonMenuItem` component.

RGB For "red, green, blue." In computer graphics, a color model that represents colors as percentages of red, green, and blue. See also **HSB**.

scroll arrow In a scrollbar, one of the two arrows that a user can click to move through displayed information in the corresponding direction (up or down in a vertical scrollbar, left or right in a horizontal scrollbar). See also **scrollbar**.

scrollbar A component that enables a user to control what portion of a document or list (or similar information) is visible on screen. A scrollbar consists of a vertical or horizontal bar, a scroll box that moves through the channel of the scrollbar, and two scroll arrows. Scrollbars are created using the `JScrollBar` component. See also **scroll arrow**, **scroll box**, **scroll pane**.

scroll box A box that a user can drag in the channel of a scrollbar to cause scrolling in the corresponding direction. The scroll box's position in the scrollbar indicates the user's location in the list, window, or pane. In the Java look and feel, the scroll box's size indicates what proportion of the total information is currently visible on screen. A large scroll box, for example, indicates that the user can peruse the contents with just a few clicks in the scrollbar. See also **scrollbar**.

scroll pane A container that provides scrolling with optional vertical and horizontal scrollbars. Scroll panes are created using the `JScrollPane` component. See also **scrollbar**.

secondary window A modal or modeless window created from and dependent upon a primary window. Secondary windows set options or supply additional details about actions and objects in the primary window. Secondary windows are dismissed when their associated primary window is dismissed. Secondary windows are created using either the `JFrame` or the `JDialog` component. See also **dialog box**, **primary window**.

separator A line graphic that divides such things as menu items into logical groupings. Separators are created using the `JSeparator` component.

slider A control that enables the user to set a value in a range—for example, the RGB values for a color. Sliders are created using the `JSlider` component. See also **progress bar**.

split pane A container that enables the user to adjust the relative size of two adjacent panes. Split panes are created using the `JSplitPane` component.

submenu A menu that a user opens by choosing a menu item in a higher-level menu. Submenus are created using the `JMenu` component.

Swing classes A set of lightweight GUI components, featuring a pluggable look and feel, that are included in the Java Foundation Classes. The Swing classes supply code for interface elements such as windows, dialog boxes and choosers, panels and panes, menus, and controls. See also **Abstract Window Toolkit**, **Java Foundation Classes**, **lightweight GUI component**, **pluggable look and feel architecture**.

tabbed pane A container that enables the user to switch between several components (usually `JPanel` components) that appear to share the same space on screen. The user can view a particular panel by clicking its tab. Tabbed panes are created using the `JTabbedPane` component.

table A two-dimensional arrangement of data in rows and columns. Tables are created using the `JTable` component.

text area A multiline region for displaying (and sometimes editing) text. Text in such areas is restricted to a single font in one font style. Text areas are created using the `JTextArea` component. See also **editor pane**.

text field An area in which a user can type a single line of text. Text fields are created using the `JTextField` component. See also **password field**.

theme mechanism A feature that enables a designer to specify alternative colors and fonts across the entire Java look and feel. See also **Java look and feel**.

title bar The strip at the top of a window that contains its title and window controls. See also **window controls**.

toggle button A button that alternates between two states. For example, a user might click one toggle button in a toolbar to turn italics on or off. A single toggle button has checkbox behavior; a programmatically grouped set of toggle buttons can be given the mutually exclusive behavior of radio buttons. Toggle buttons are created using the `JToggleButton` component. See also **toolbar button**.

toolbar A collection of frequently used commands or options. Toolbars typically contain buttons, but other controls (such as text fields and combo boxes) can be placed in toolbars as well. Toolbars are created using the `JToolBar` component. See also **toolbar button**.

toolbar button A button that appears in a toolbar, typically a command or toggle button. Toolbar buttons are created using the `JButton` or `JToggleButton` component. See also **command button**, **toggle button**.

tool tip A short text string that appears on screen when a user moves the pointer over the interface element associated with that tip.

top-level container The highest-level container for a Java application or applet. The top-level containers are `JWindow`, `JFrame`, and `JDialog`.

tree view A representation of hierarchical data (for example, directory and file names) as a graphical outline. Clicking expands or collapses elements of the outline. Tree views are created using the `JTree` component.

typeface A set of characters in a particular design and style (typically, normal, bold, italic, or bold italic) and available within a specified range of sizes. Sometimes called a "font."

utility window In a non-MDI application with the Java look and feel, a modeless dialog box that typically displays a collection of tools, colors, fonts, or patterns. Unlike palette windows, utility windows do not float. User choices made in a utility window affect whichever primary window is active. A utility window is not dismissed when a primary window is dismissed. Utility windows are created using the `JDialog` component. See also **palette window**, **secondary window**.

window See **dialog box**, **palette window**, **plain window**, **primary window**, **secondary window**, **utility window**.

window controls Controls that affect the state of a window (for example, the Maximize button in Microsoft Windows title bars or the resize box in the lower-right corner of Macintosh windows).

INDEX

C

D

E

F

K

L

M

N

O

P

Q

R

S

T