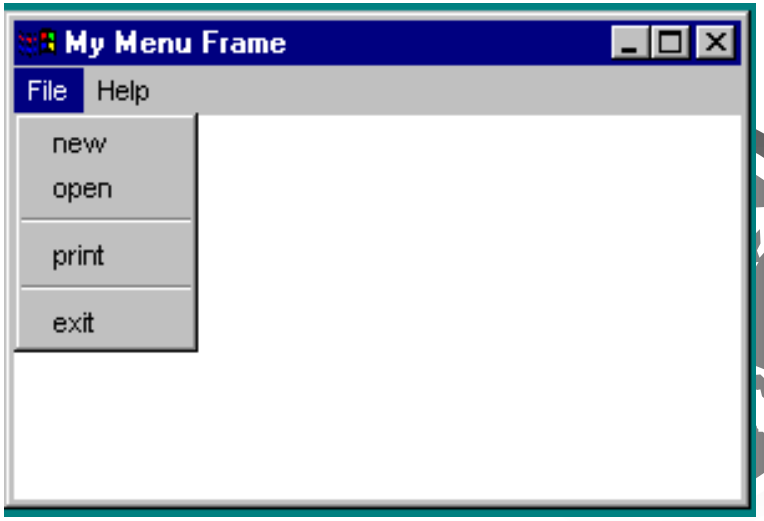
21 Menus, Toolbars, and Dialogs

Chapter 34@8e

Menus

Java provides several classes—`JMenuBar`, `JMenu`, `JMenuItem`, `JCheckBoxMenuItem`, and `JRadioButtonMenuItem` —to implement menus in a frame.

A JFrame or JApplet can hold a *menu bar* to which the *pull-down menus* are attached. Menus consist of *menu items* that the user can select (or toggle on or off). Menu bars can be viewed as a structure to support menus.



The `JMenuBar` Class

A menu bar holds menus; the menu bar can only be added to a frame. Following is the code to create and add a `JMenuBar` to a frame:

```
JFrame f = new JFrame();
f.setSize(300, 200);
f.setVisible(true);
JMenuBar mb = new JMenuBar();
f.setJMenuBar(mb);
```

The JMenu Class

You attach menus onto a `JMenuBar`. The following code creates two menus, File and Help, and adds them to the `JMenuBar mb`:

```
JMenu fileMenu = new JMenu("File", false);
JMenu helpMenu = new JMenu("Help", true);
mb.add(fileMenu);
mb.add(helpMenu);
```

```
/**
 * Constructs a new <code>JMenu</code> with the supplied string as
 * its text and specified as a tear-off menu or not.
 *
 * @param s the text for the menu label
 * @param b can the menu be torn off (not yet implemented)
 */
public JMenu(String s, boolean b) {
    this(s);
}
```

如果布尔值为false，那么当释放鼠标按钮后，菜单项会消失；如果布尔值为true，那么当释放鼠标按钮后，菜单项仍将显示。这时的菜单称为 tearOff 菜单。

The `JMenuItem` Class

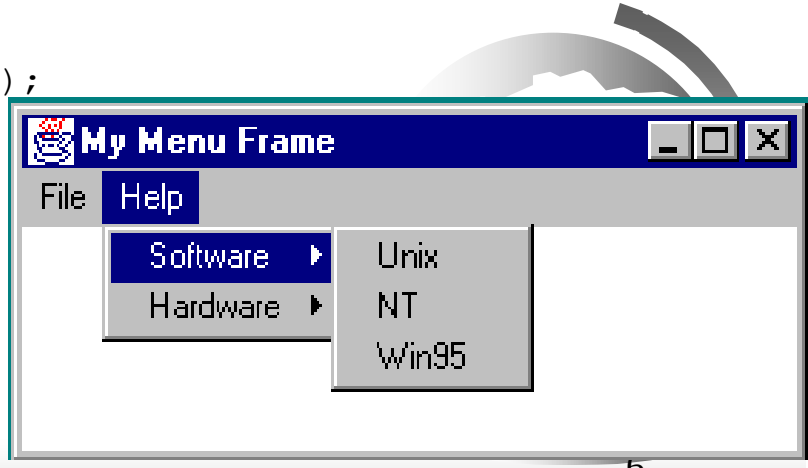
You add menu items on a menu. The following code adds menu items and item separators in menu `fileMenu`:

```
fileMenu.add(new JMenuItem("new"));
fileMenu.add(new JMenuItem("open"));
fileMenu.addSeparator();
fileMenu.add(new JMenuItem("print"));
fileMenu.add(new JMenuItem("exit"));
fileMenu.addSeparator();
```

Submenus

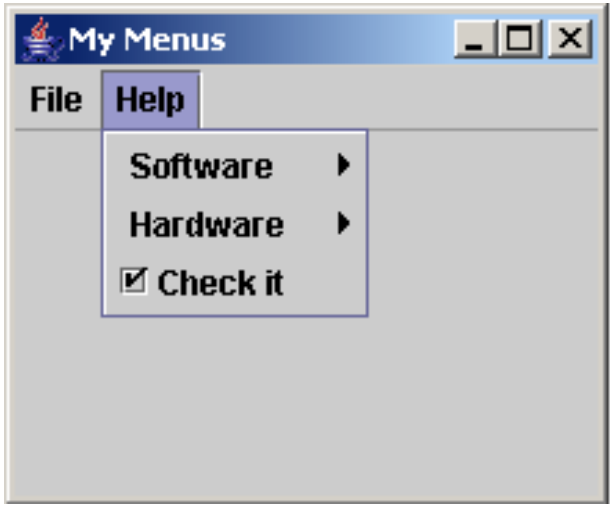
You can add submenus into menu items. The following code adds the submenus “Unix,” “NT,” and “Win95” into the menu item “Software.”

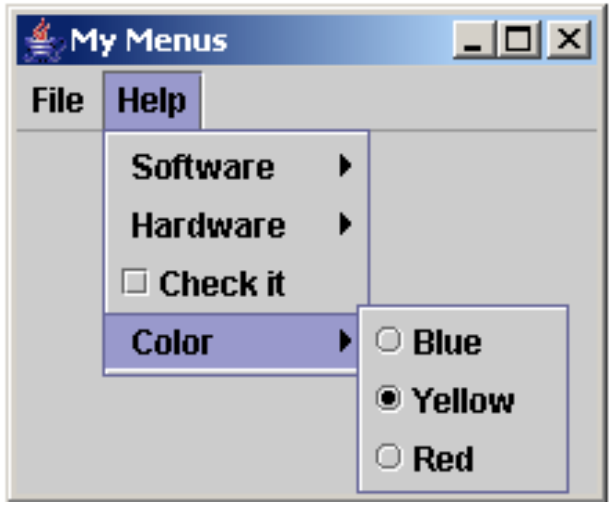
```
JMenu softwareHelpSubMenu = new JMenu("Software");

JMenu hardwareHelpSubMenu = new JMenu("Hardware");

helpMenu.add(softwareHelpSubMenu);

helpMenu.add(hardwareHelpSubMenu);

softwareHelpSubMenu.add(new JMenuItem("Unix"));

softwareHelpSubMenu.add(new JMenuItem("NT"));

softwareHelpSubMenu.add(new JMenuItem("Win95"));
```

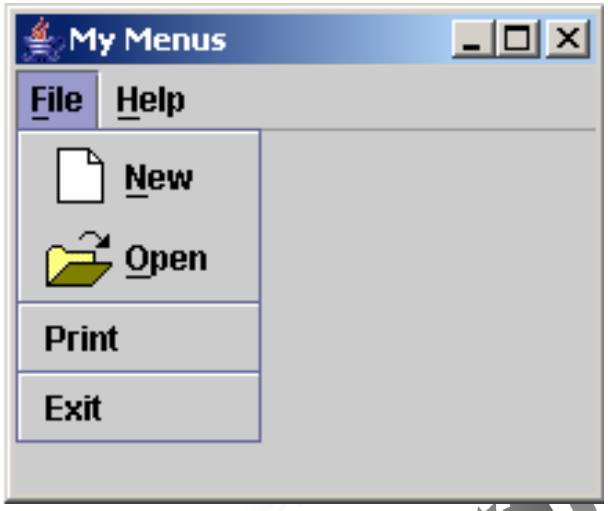


Check Box Menu Items

helpMenu.add(new JCheckBoxMenuItem("Check it"));







Radio Button Menu Items

```
JMenu colorHelpSubMenu = new JMenu("Color");
helpMenu.add(colorHelpSubMenu);

JRadioButtonMenuItem jrbmiBlue, jrbmiYellow, jrbmiRed;
colorHelpSubMenu.add(jrbmiBlue =  new JRadioButtonMenuItem("Blue"));
colorHelpSubMenu.add(jrbmiYellow =   new JRadioButtonMenuItem("Yellow"));
colorHelpSubMenu.add(jrbmiRed =   new JRadioButtonMenuItem("Red"));
```

```
ButtonGroup btg = new ButtonGroup();
btg.add(jrbmiBlue);
btg.add(jrbmiYellow);
btg.add(jrbmiRed);
```

Image Icons, Keyboard Mnemonics, and Keyboard Accelerators

```
JMenuItem jmiNew, jmiOpen;

fileMenu.add(jmiNew = new JMenuItem("New"));

fileMenu.add(jmiOpen = new JMenuItem("Open"));
```

```
jmiNew.setIcon(new ImageIcon("image/new.gif"));

jmiOpen.setIcon(new ImageIcon("image/open.gif"));
```

```
helpMenu.setMnemonic('H');

fileMenu.setMnemonic('F');

jmiNew.setMnemonic('N');

jmiOpen.setMnemonic('O');
```

setMnemonic 设置的是括号里的。只有在菜单显示的情况下才可以起作用。

setAccelerator 设置的是后面的 Ctrl + O 之类。

```
jmiOpen.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O, ActionEvent.CTRL_MASK));
```

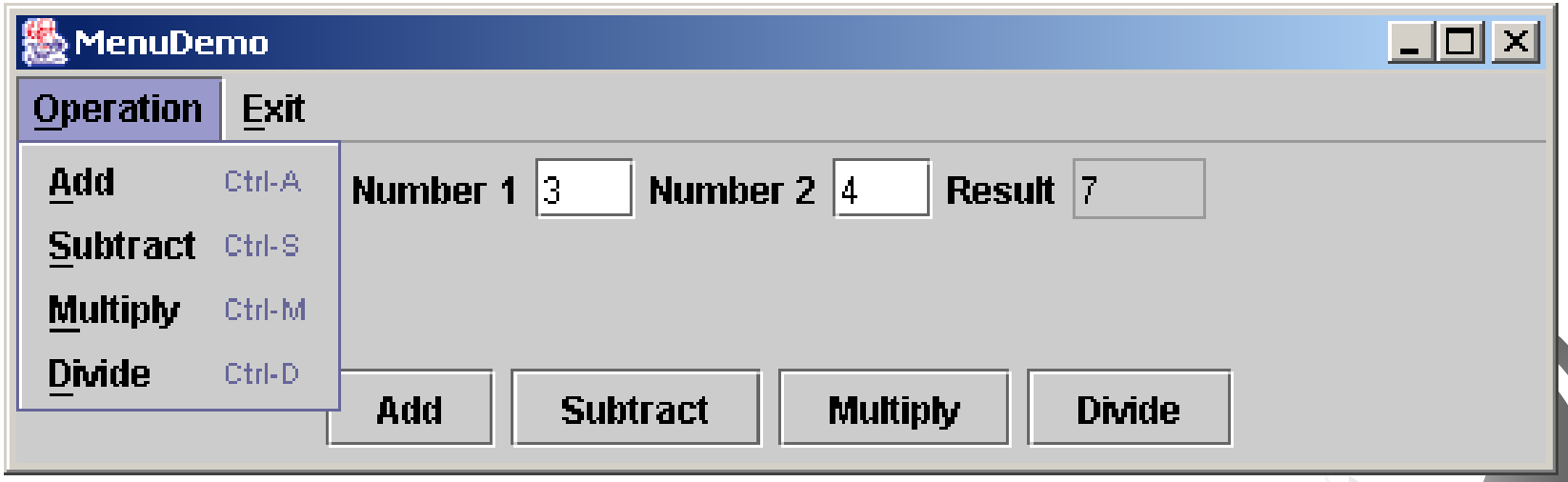
```
class MenuActionListener implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    System.out.println("Selected: " + e.getActionCommand());

  }
}

public class ContructMenuActionListener {
  public static void main(final String args[]) {
    JFrame frame = new JFrame("MenuSample Example");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JMenuBar menuBar = new JMenuBar();

    // File Menu, F - Mnemonic
    JMenu fileMenu = new JMenu("File");
    fileMenu.setMnemonic(KeyEvent.VK_F);
    menuBar.add(fileMenu);

    // File->New, N - Mnemonic
    JMenuItem newMenuItem = new JMenuItem("New");
    newMenuItem.addActionListener(new MenuActionListener());
    fileMenu.add(newMenuItem);

    frame.setJMenuBar(menuBar);
    frame.setSize(350, 250);
    frame.setVisible(true);
  }
}
```

Example: Using Menus

Objective: Create a user interface that performs arithmetic. The interface contains labels and text fields for Number 1, Number 2, and Result. The Result box displays the result of the arithmetic operation between Number 1 and Number 2.

Example: Using Menus

Problem: Create a user interface that performs arithmetic. The interface contains labels and text fields for Number 1, Number 2, and Result. The Result box displays the result of the arithmetic operation between Number 1 and Number 2.



MenuDemo

Run

Popup Menus

A popup menu, also known as a context menu, is like a regular menu, but does not have a menu bar and can float anywhere on the screen.

Creating a popup menu is similar to creating a regular menu. First, you create an instance of JPopupMenu, then you can add JMenuItem, JCheckBoxMenuItem, JRadioButtonMenuItem, and separators to the popup menu. For example, the following code creates a JPopupMenu and adds JMenuItems into it:

```
JPopupMenu jPopupMenu = new JPopupMenu();
jPopupMenu.add(new JMenuItem("New"));
jPopupMenu.add(new JMenuItem("Open"));
```

Displaying a Popup Menu

A regular menu is always attached to a menu bar using the setJMenuBar method, but a popup menu is associated with a parent component and is displayed using the show method in the JPopupMenu class. You specify the parent component and the location of the popup menu, using the coordinate system of the parent like this:

```
jPopupMenu.show(component, x, y);
```

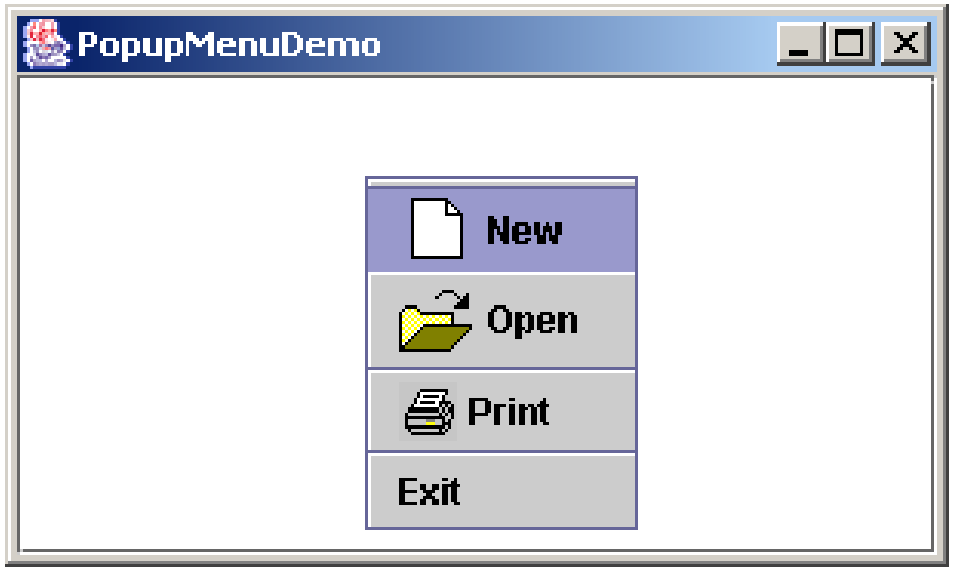
Popup Trigger

The popup menu usually contains the commands for an object. Customarily, you display a popup menu by pointing to the object and clicking a certain mouse button, the so-called *popup trigger*.

Popup triggers are system-dependent. In Windows, the popup menu is displayed when the right mouse button is released. In Motif, the popup menu is displayed when the third mouse button is pressed and held down.

Example: Using Popup Menus

Problem: The program creates a text area in a scroll pane. The popup menu is displayed when the mouse pointed to the text area triggers the popup menu.



PopupMenuDemo

Run

```
...
jTextArea1.addMouseListener(new MouseAdapter() {
  @Override
  public void mousePressed(MouseEvent e) { // For Motif
    showPopup(e);
  }

  @Override
  public void mouseReleased(MouseEvent e) { // For Windows
    showPopup(e);
  }
});
```

```
/** Display popup menu when triggered */
private void showPopup(java.awt.event.MouseEvent evt) {
  if (evt.isPopupTrigger())
    jPopupMenu1.show(evt.getComponent(), evt.getX(), evt.getY());
}
```

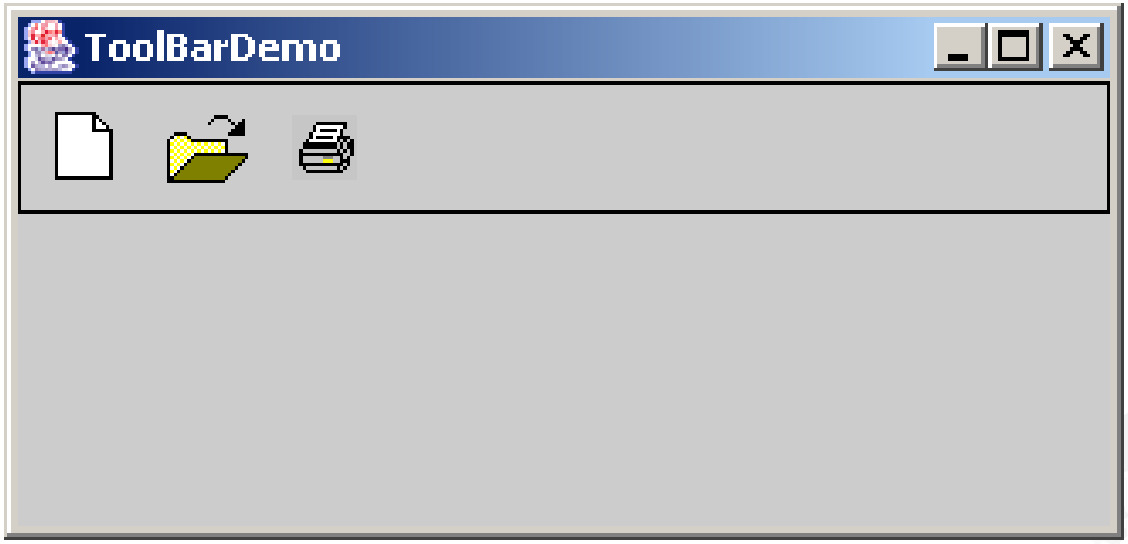
JToolBar

In user interfaces, a toolbar is often used to hold commands that also appear in the menus. Frequently used commands are placed in a toolbar for quick access. Clicking a command in the toolbar is faster than choosing it from the menu.

Swing provides the JToolBar class as the **container** to hold tool bar components. JToolBar uses BoxLayout to manage components by default. You can set a different layout manager if desired. The components usually appear as icons. Since icons are not components, they cannot be placed into a tool bar directly. Instead you may place buttons into the tool bar and set the icons on the buttons. An instance of JToolBar is like a regular container. Often it is placed in the north, west, or east of a container of BorderLayout.

Example: Using Tool Bars

Problem: Create a JToolBar that contains three buttons with the icons representing the commands New, Open, and Print icons.



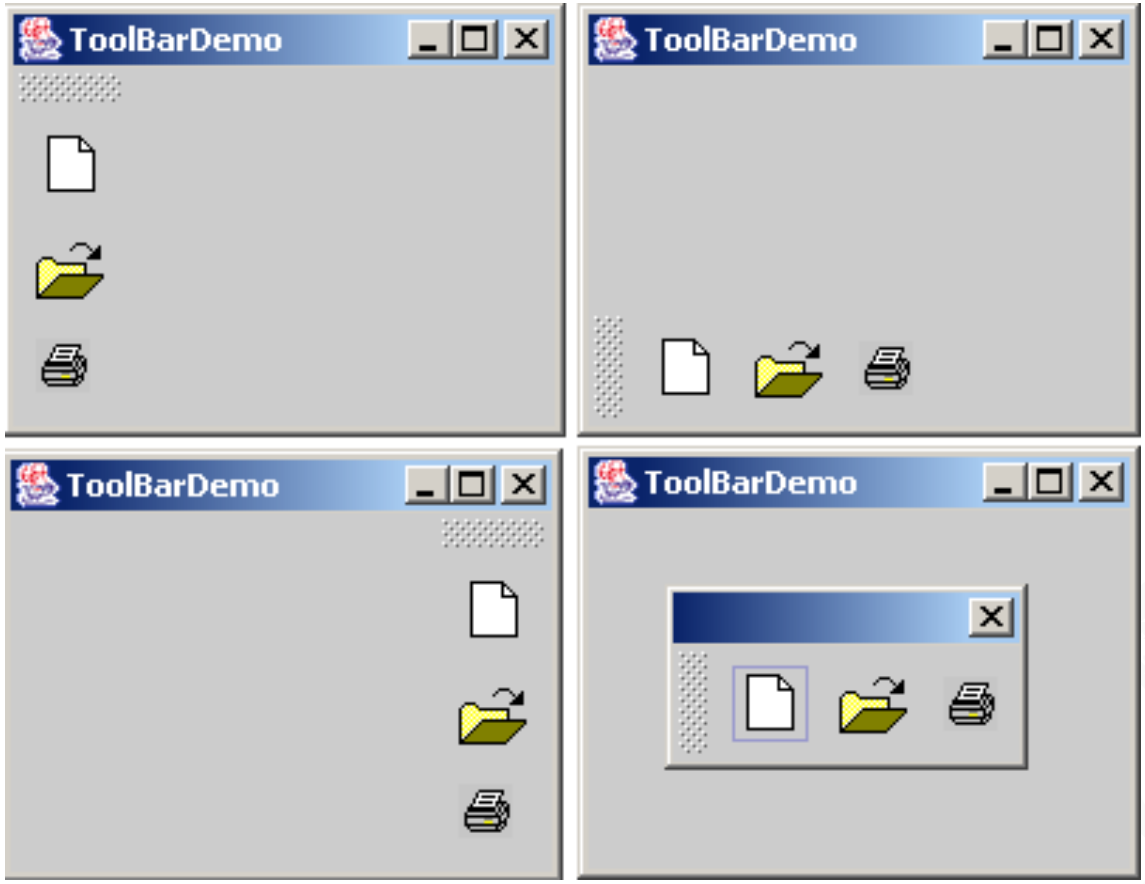
ToolBarDemo

Run

```
JToolBar jToolBar1 = new JToolBar("My Tool Bar");
jToolBar1.setFloatable(true);
jToolBar1.add(jbtNew);
jToolBar1.add(jbtOpen);
jToolBar1.add(jbPrint);

jbtNew.setToolTipText("New");
jbtOpen.setToolTipText("Open");
jbPrint.setToolTipText("Print");

jbtNew.setBorderPainted(false);
jbtOpen.setBorderPainted(false);
jbPrint.setBorderPainted(false);

add(jToolBar1, BorderLayout.NORTH);
```

Floatable Tool Bars

JToolBar may be floatable.

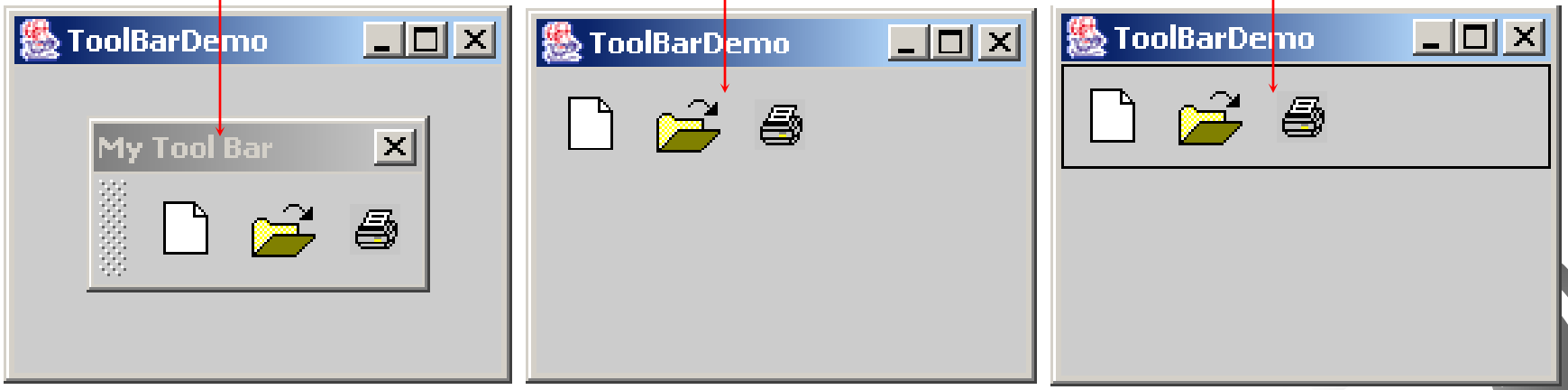


Tool Bars Title and Border

You can set a title for the floatable tool bar.

If you set floatable false, the floatable controller is not displayed

If you set floatable false, the floatable controller is not displayed



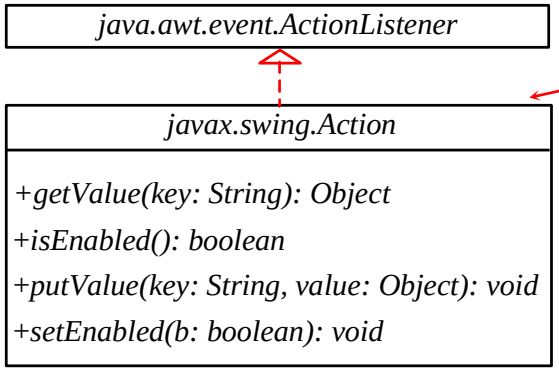
Processing Actions Using the Action Interface

Often menus and tool bars contain some common actions. For example, you can save a file by choosing *File*, *Save*, or by clicking the save button in the tool bar.

Swing provides the Action interface, which can be used to create **action objects** for processing actions. Using Action objects, common action processing can be centralized and separated from the other application code.

ActionListener, Action, and AbstractAction

The Action interface provides a useful extension to the ActionListener interface in cases where the same functionality may be accessed by several controls



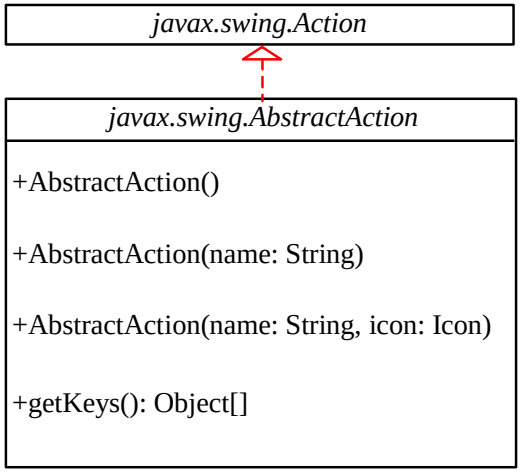
Gets one of this object's properties using the associated key.

Returns true if action is enabled.

Associates a key/value pair with the action.

Enables or disables the action.

AbstractAction class provides a default implementation for Action



Defines an Action object with a default description string and default icon.

Defines an Action object with the specified description string and a default icon.

Defines an Action object with the specified description string and the specified icon.

Returns an array of objects which are keys for which values have been set for this AbstractAction, or null if no keys have values set.

Creating and Using an Action instance

```
Action exitAction = new AbstractAction("Exit") {
  public void actionPerformed(ActionEvent e) {
    System.exit(0);
  }
};
```

Certain containers, such as JMenu and JToolBar, know how to add an Action object. When an Action object is added to such a container, the container automatically creates an appropriate component for the Action object, and registers a listener with the Action object. Here is an example of adding an Action object to a menu and a tool bar:

```
jMenu.add(exitAction);
jToolBar.add(exitAction);
```

根据Action对象中的name和icon属性，自动的创建按钮或者菜单项

Associating Action instances with Buttons

Several Swing components such as JButton, JRadioButton, and JCheckBox contain constructors to create instances from Action objects.

For example, you can create a JButton from an Action object, as follows:

```
JButton jbt = new JButton(exitAction);
```

Associating Action instances with Keystrokes

Action objects can also be used to respond to keystrokes. To associate actions with keystrokes, you need to create an instance of the KeyStroke class using the static getKeyStroke method, as follows:

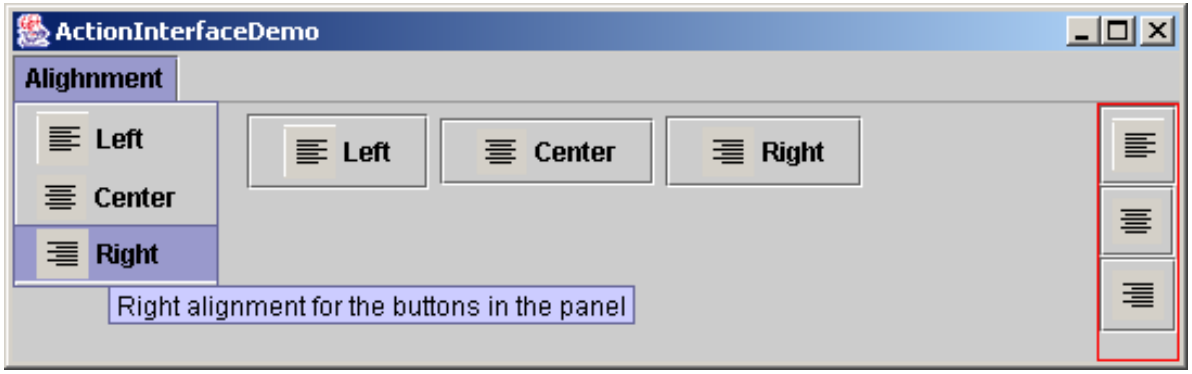
```
KeyStroke exitKey =
  KeyStroke.getKeyStroke(KeyEvent.VK_E, KeyEvent.CTRL_MASK);
```

You can now associate an action with the keystroke by registering it with an instance of JComponent. For example, the following code associates exitAction with exitKey, and registers this action with jPanel1.

```
jPanel1.registerKeyboardAction
  (exitAction, exitKey, Component.WHEN_IN_FOCUSED_WINDOW);
```

Example: Using Actions

Problem: Write a program that creates three menu items, Left, Center, and Right, three tool bar buttons, Left, Center, and Right, and three regular buttons, Left, Center, and Right in a panel. The panel that holds the buttons uses the FlowLayout. The action of the left, center, and right button sets the alignment of the FlowLayout to left, right, and center, respectively.



```
// Create actions
Action leftAction = new MyAction("Left", leftImageIcon,
  "Left alignment for the buttons in the panel",
  new Integer(KeyEvent.VK_L),
  KeyStroke.getKeyStroke(KeyEvent.VK_L, ActionEvent.CTRL_MASK));
Action centerAction = new MyAction("Center", centerImageIcon,
  "Center alignment for the buttons in the panel",
  new Integer(KeyEvent.VK_C),
  KeyStroke.getKeyStroke(KeyEvent.VK_C, ActionEvent.CTRL_MASK));
Action rightAction = new MyAction("Right", rightImageIcon,
  "Right alignment for the buttons in the panel",
  new Integer(KeyEvent.VK_R),
  KeyStroke.getKeyStroke(KeyEvent.VK_R, ActionEvent.CTRL_MASK));

// Create menus
JMenuBar jMenuBar1 = new JMenuBar();
JMenu jmenuAlignment = new JMenu("Alignment");
setJMenuBar(jMenuBar1);
jMenuBar1.add(jmenuAlignment);

// Add actions to the menu
jmenuAlignment.add(leftAction);
jmenuAlignment.add(centerAction);
jmenuAlignment.add(rightAction);

// Add actions to the toolbar
JToolBar jToolBar1 = new JToolBar(JToolBar.VERTICAL);
jToolBar1.setBorder(BorderFactory.createLineBorder(Color.red));
jToolBar1.add(leftAction);
jToolBar1.add(centerAction);
jToolBar1.add(rightAction);
```

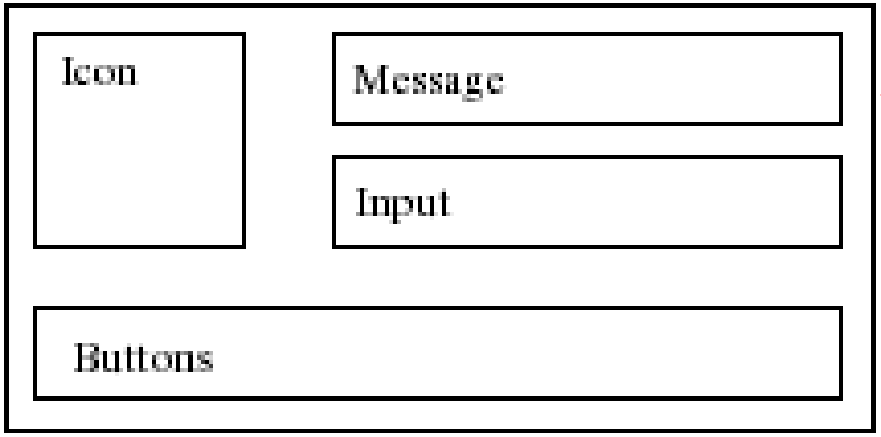
```
private class MyAction extends AbstractAction {
  String name;

  MyAction(String name, Icon icon) {
    super(name, icon);
    this.name = name;
  }

  MyAction(String name, Icon icon, String desc, Integer mnemonic,
      KeyStroke accelerator) {
    super(name, icon);
    putValue(Action.SHORT_DESCRIPTION, desc);
    putValue(Action.MNEMONIC_KEY, mnemonic);
    putValue(Action.ACCELERATOR_KEY, accelerator);
    this.name = name;
  }
```

JOptionPane Dialogs

A *dialog* is normally used as a temporary window to receive additional information from the user, or to provide notification that some event has occurred.

Java provides the JOptionPane class, which can be used to **create standard dialogs**.

You can also build custom dialogs by extending the JDialog class.



A JOptionPane dialog can display an icon, a message, an input, and option buttons.

Message Dialogs

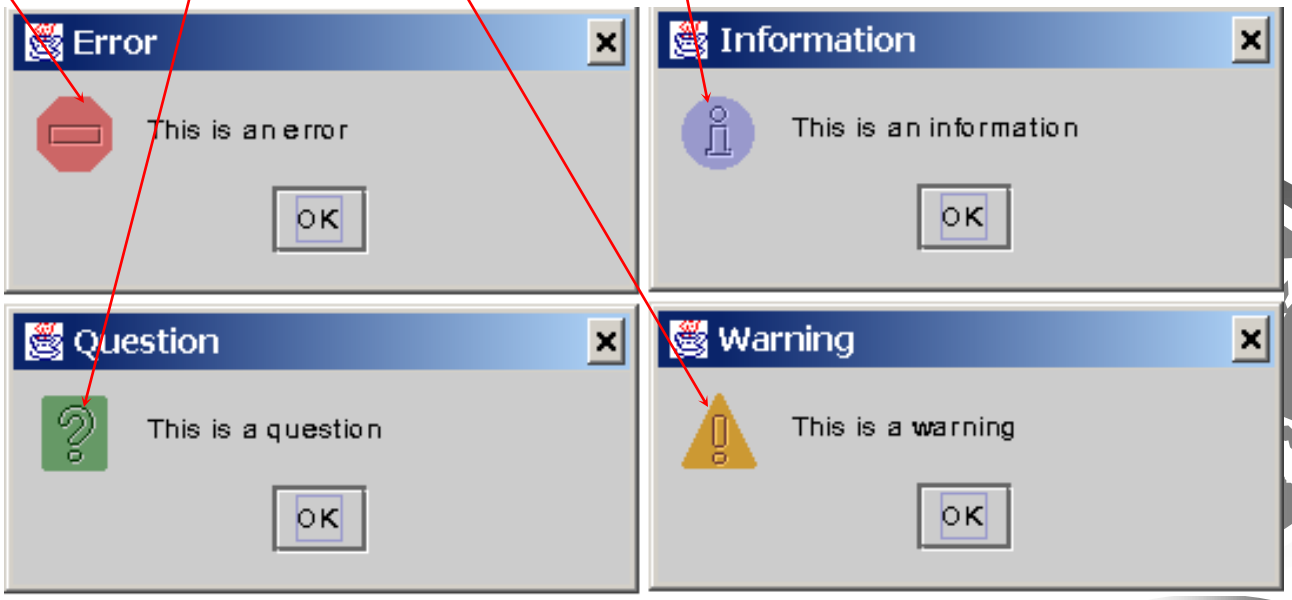
A message dialog box simply displays a message to alert the user and waits for the user to click the OK button to close the dialog.

```
JOptionPane.showMessageDialog(null, “This is an error",
  “Error", JOptionPane.ERROR_MESSAGE);
```

Message Types

The messageType is one of the following constants:

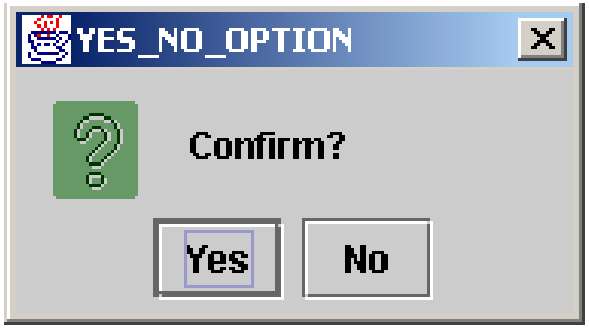
- JOptionPane.ERROR_MESSAGE
- JOptionPane.INFORMATION_MESSAGE
- JOptionPane.PLAIN_MESSAGE
- JOptionPane.WARNING_MESSAGE
- JOptionPane.QUESTION_MESSAGE

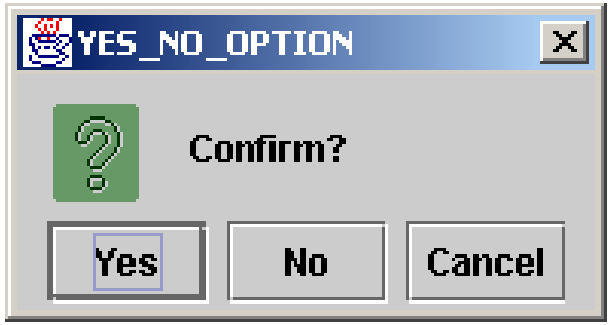


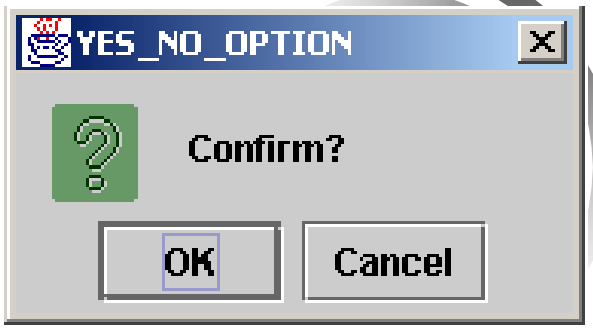
Confirmation Dialogs

A message dialog box displays a message and waits for the user to click the OK button to dismiss the dialog. The message dialog does not return any value.

A confirmation dialog asks a question and requires the user to respond with an appropriate button. The confirmation dialog returns a value that corresponds to a selected button.







创建确认对话框的方法如下：

```
01.  public static int showConfirmDialog(Component parentComponent,Object message,String
     title,int optionType,int messageType,Icon icon)
```

参数 parentComponent、message、title、messageType 和 icon 与 showMessageDialog() 方法中的参数的含义相同。其中，只有 parentComponent 和 message 参数是必需的，title 的默认值为“选择一个选项”。messageType 的默认值是 QUESTION_MESSAGE。optionType 参数用于控制在对话框上显示的按钮，可选值如下：

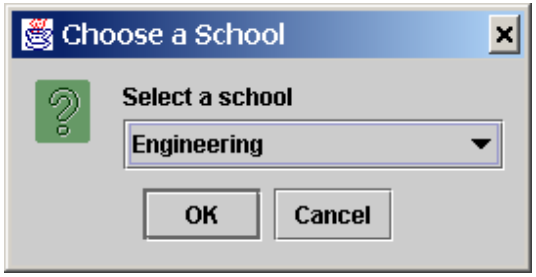
- 0 或 JOptionPane.YES_NO_OPTIION。
- 1 或 JOptionPane.YES_NO_CANCEL_0PTII0N。
- 2 或 JOptionPane.OK_CANCEL_OPTIION。

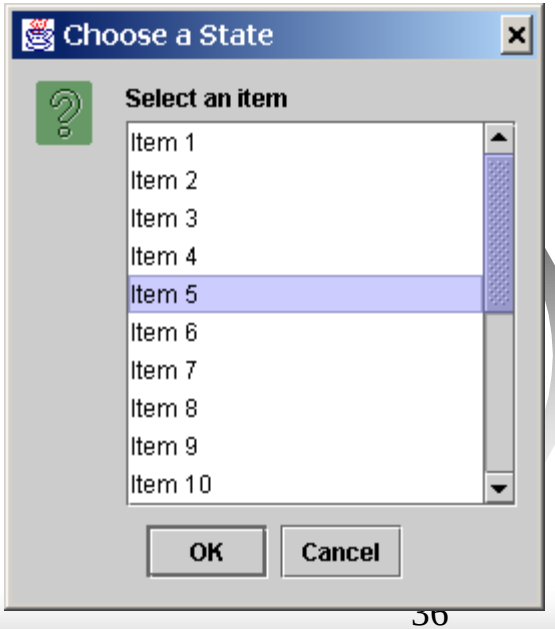
例如，使用 showCon&mDialog() 方法创建 3 个确认对话框，该方法中指定的参数个数和参数值都是不同的，语句如下：

```
01.  JOptionPane.showConfirmDialog(p,"确定要删除吗？","删除提示",0);
02.  JOptionPane.showConfirmDialog(p,"确定要删除吗？","删除提示",1,2);
03.  ImageIcon icon=new ImageIcon("F:\\pic\\n63.gif");
04.  JOptionPane.showConfirmDialog(p,"确定要删除吗？","删除提示",2,1,icon);
```

Input Dialogs

An *input dialog* box is used to receive input from the user. The input can be entered from a text field or selected from a combo box or a list. Selectable values can be specified in an array, and a particular value can be designated as the initial selected value.





String javax.swing.JOptionPane.showInputDialog(Component parentComponent, Object message, String title, int messageType) throws HeadlessException

Shows a dialog requesting input from the user parented to `parentComponent` with the dialog having the title `title` and message type `messageType`.

Parameters:

parentComponent the parent `Component` for the dialog
message the `Object` to display
title the `String` to display in the dialog title bar
messageType the type of message that is to be displayed: `ERROR_MESSAGE`, `INFORMATION_MESSAGE`, `WARNING_MESSAGE`, `QUESTION_MESSAGE`, or `PLAIN_MESSAGE`

Press 'F2' for focus

Object javax.swing.JOptionPane.showInputDialog(Component parentComponent, Object message, String title, int messageType, Icon icon, Object[] selectionValues, Object initialSelectionValue) throws HeadlessException

Prompts the user for input in a blocking dialog where the initial selection, possible selections, and all other options can be specified. The user will able to choose from `selectionValues`, where `null` implies the user can input whatever they wish, usually by means of a `JTextField`. `initialSelectionValue` is the initial value to prompt the user with. It is up to the UI to decide how best to represent the `selectionValues`, but usually a `JComboBox`, `JList`, or `JTextField` will be used.

Parameters:

parentComponent the parent `Component` for the dialog
message the `Object` to display
title the `String` to display in the dialog title bar
messageType the type of message to be displayed: `ERROR_MESSAGE`, `INFORMATION_MESSAGE`, `WARNING_MESSAGE`, `QUESTION_MESSAGE`, or `PLAIN_MESSAGE`
icon the `Icon` image to display
selectionValues an array of `Objects` that gives the possible selections
initialSelectionValue the value used to initialize the input field

Returns:

user's input, or `null` meaning the user canceled the input

Throws:

HeadlessException - if `GraphicsEnvironment.isHeadless` returns `true`

See Also:

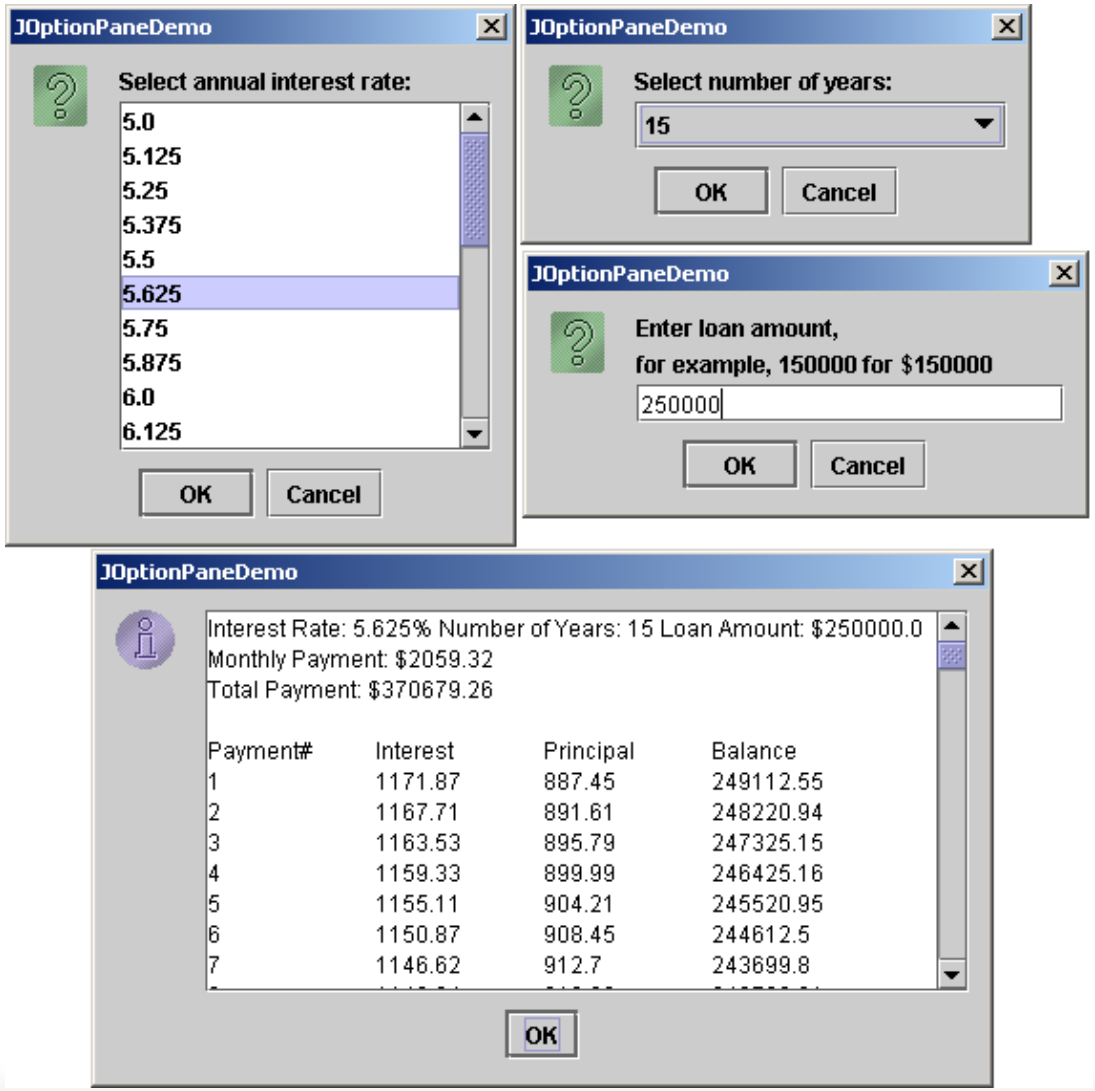
Option Dialogs

An *option dialog* allows you to create custom buttons.

Example: Creating JOptionPane Dialogs

Problem: This example demonstrates using standard dialogs. The program prompts the user to select the annual interest rate from a list in an input dialog, the number of years from a combo box in an input dialog, and the loan amount from an input dialog, and displays the loan payment schedule in a text area inside a JScrollPane in a message dialog.

Example: Creating JOptionPane Dialogs, cont.



JOptionPaneDemo

Run

```
// Create an array for annual interest rates
Object[] rateList = new Object[25];
int i = 0;
for (double rate = 5; rate <= 8; rate += 1.0 / 8)
  rateList[i++] = new Double(rate);

// Prompt the user to select an annual interest rate
Object annualInterstRateObject = JOptionPane.showInputDialog(
  null, "Select annual interest rate:", "JOptionPaneDemo",
  JOptionPane.QUESTION_MESSAGE, null, rateList, null);
double annualInterestRate =
  ((Double)annualInterstRateObject).doubleValue();
```

```
// Create an array for number of years
Object[] yearList = {new Integer(7), new Integer(15),
  new Integer(30)};

// Prompt the user to enter number of years
Object numberOfYearsObject = JOptionPane.showInputDialog(null,
  "Select number of years:", "JOptionPaneDemo",
  JOptionPane.QUESTION_MESSAGE, null, yearList, null);
int numberOfYears = ((Integer)numberOfYearsObject).intValue();
```

```
// Prompt the user to enter loan amount
String loanAmountString = JOptionPane.showInputDialog(null,
  "Enter loan amount,\nfor example, 150000 for $150000",
  "JOptionPaneDemo", JOptionPane.QUESTION_MESSAGE);
double loanAmount = Double.parseDouble(loanAmountString);
```

```
// Display the header
output += "\nPayment#\tInterest\tPrincipal\tBalance\n";

for (i = 1; i <= numberOfYears * 12; i++) {
  interest = (int)(monthlyInterestRate * balance * 100) / 100.0;
  principal = (int)((monthlyPayment - interest) * 100) / 100.0;
  balance = (int)((balance - principal) * 100) / 100.0;
  output += i + "\t" + interest + "\t" + principal + "\t" +
    balance + "\n";
}

// Display monthly payment and total payment
JScrollPane jsp = new JScrollPane(new JTextArea(output));
jsp.setPreferredSize(new java.awt.Dimension(400, 200));
JOptionPane.showMessageDialog(null, jsp,
  "JOptionPaneDemo", JOptionPane.INFORMATION_MESSAGE, null);
```

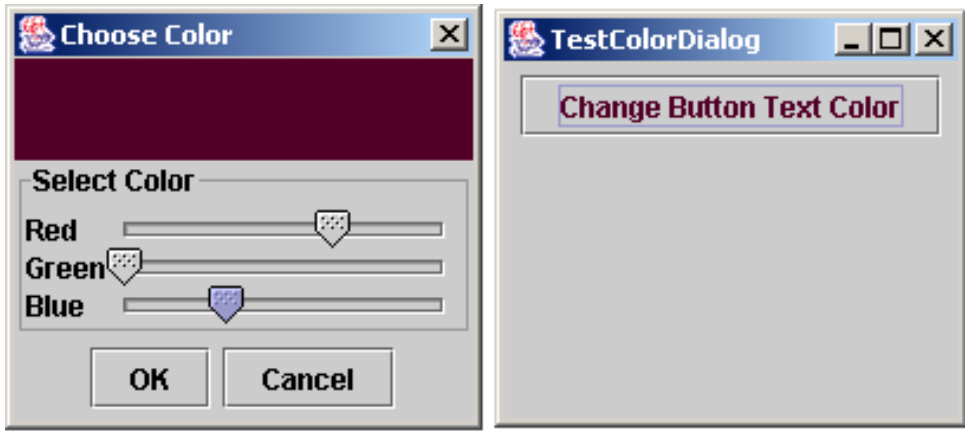
Creating Custom Dialogs

In Swing, the JDialog class can be extended to create custom dialogs.

JDialog is a subclass of java.awt.Dialog fitted with an instance of JRootPane. As with JFrame, components are added to the contentPane of JDialog. Creating a custom dialog usually involves laying out user interface components in the dialog, adding buttons for dismissing the dialog, and installing listeners that respond to button actions.

Example: Creating Custom Dialogs

Problem: Create a custom dialog box for choosing colors, as shown in Figure 34.18 (a). Use this dialog to choose the color for the foreground for the button, as shown in Figure 34.18 (b).



```
public class ColorDialog extends JDialog {
  // Declare color component values and selected color
  private int redValue, greenValue, blueValue;
  private Color color = null;

  // Create sliders
  private JSlider jslRed = new JSlider(0, 128);
  private JSlider jslGreen = new JSlider(0, 128);
  private JSlider jslBlue = new JSlider(0, 128);

  // Create two buttons
  private JButton jbtOK = new JButton("OK");
  private JButton jbtCancel = new JButton("Cancel");

  // Create a panel to display the selected color
  private JPanel jpSelectedColor = new JPanel();

  public ColorDialog() {
    this(null, true);
```

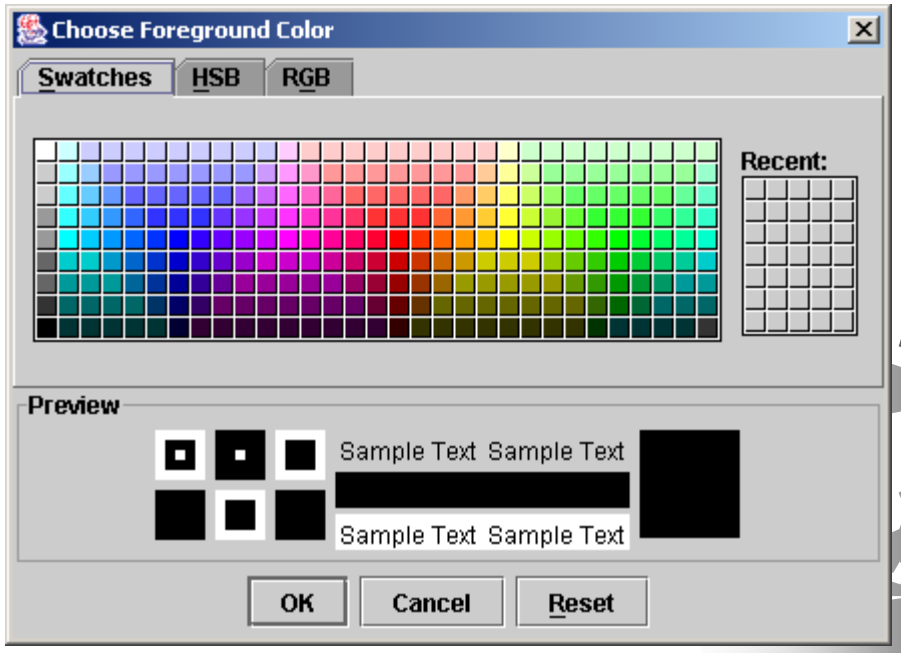
```
jbtOK.addActionListener(new ActionListener() {
  @Override
  public void actionPerformed(ActionEvent e) {
    setVisible(false);
  }
});

jbtCancel.addActionListener(new ActionListener() {
  @Override
  public void actionPerformed(ActionEvent e) {
    color = null;
    setVisible(false);
  }
});
```

```
jslRed.addChangeListener(new ChangeListener() {
  @Override
  public void stateChanged(ChangeEvent e) {
    redValue = jslRed.getValue();
    color = new Color(redValue, greenValue, blueValue);
    jpSelectedColor.setBackground(color);
  }
});
```

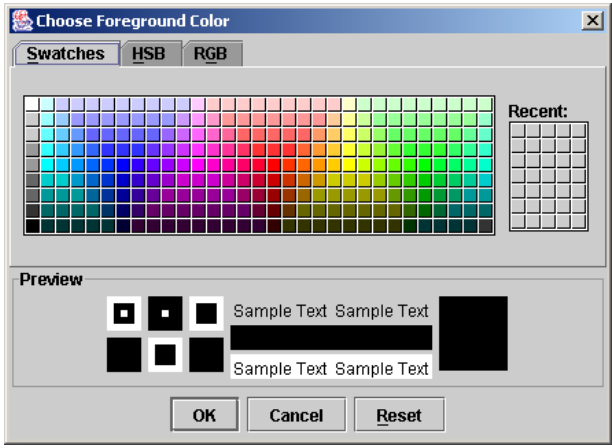
```
private ColorDialog colorDialog1 = new ColorDialog();
private JButton jbtChangeColor = new JButton("Choose color");

public TestColorDialog() {
  setLayout(new java.awt.FlowLayout());
  jbtChangeColor.setText("Change Button Text Color");
  jbtChangeColor.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
      colorDialog1.setVisible(true);

      if (colorDialog1.getColor() != null)
        jbtChangeColor.setForeground(colorDialog1.getColor());
    }
  });
  add(jbtChangeColor);
}
```

JColorChooser

Color dialogs are commonly used in GUI programming. Swing provides a convenient and versatile color dialog named javax.swing.JColorChooser. Like JOptionPane, JColorChooser is a lightweight component inherited from JComponent. It can be added to any container.



Using JColorChooser



To create a JColorChooser, use

```
new JColorChooser();
```

To display a JColorChooser dialog box, use

```
public static Color showDialog(Component parentComponent,
    String title,  Color initialColor)
```

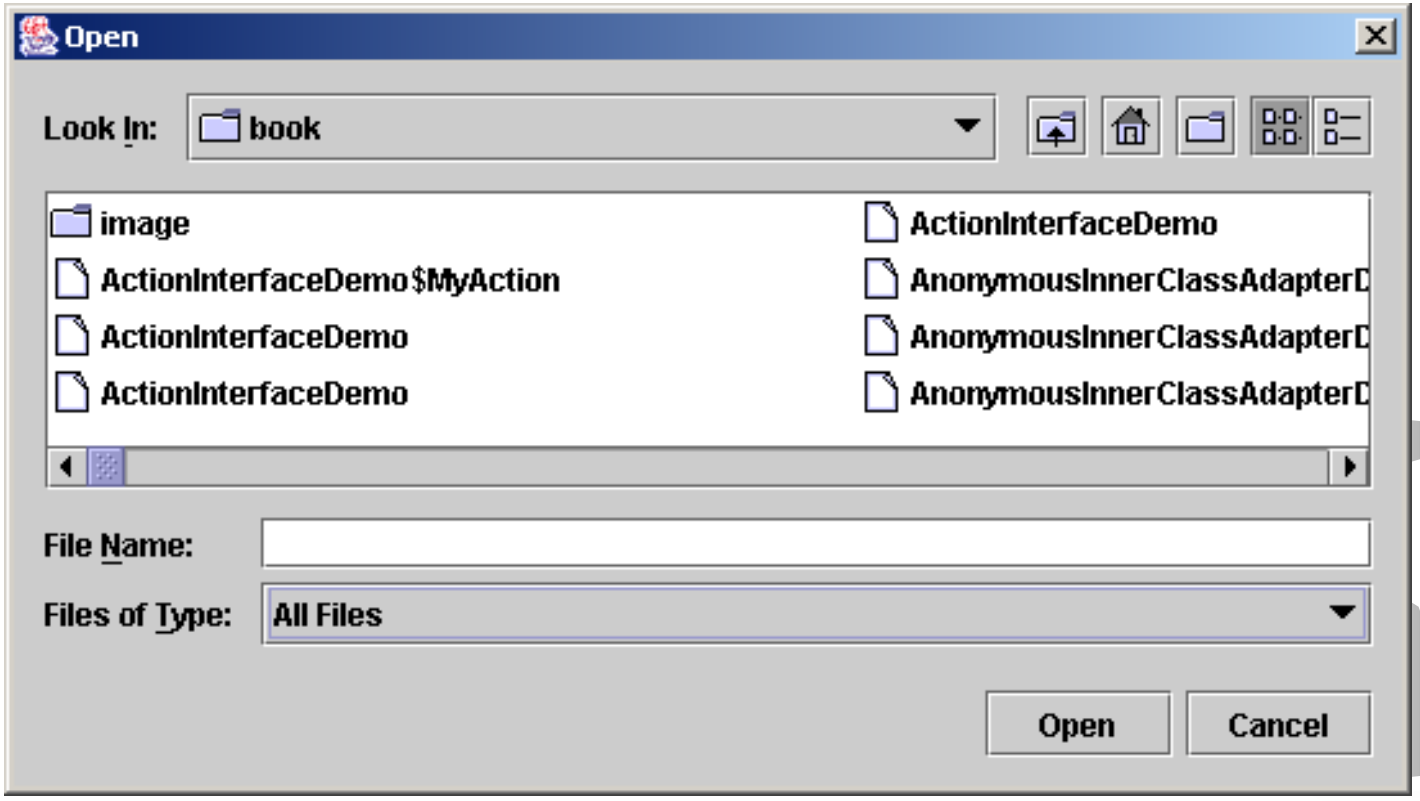
This method creates an instance of JDialog with three buttons, OK, Cancel, and Reset, to hold a JColorChooser object, as shown in Figure 34.27. The method displays a modal dialog. If the user clicks the *OK* button, the method dismisses the dialog and returns the selected color. If the user clicks the *Cancel* button or closes the dialog, the method dismisses the dialog and returns null.

JColorChooserDemo

```
public class JColorChooserDemo extends javax.swing.JApplet {
  public void init() {
    Color color = JColorChooser.showDialog(this, "Choose a color",
      Color.YELLOW);
  }

  public static void main(String[] args) {
    JColorChooserDemo applet = new JColorChooserDemo();
    JFrame frame = new JFrame();
    //EXIT_ON_CLOSE == 3
    frame.setDefaultCloseOperation(3);
    frame.setTitle("TestColorDialog");
    frame.getContentPane().add(applet, BorderLayout.CENTER);
    applet.init();
    applet.start();
    frame.setSize(400,320);
    Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
    frame.setLocation((d.width - frame.getSize().width) / 2,
      (d.height - frame.getSize().height) / 2);
    frame.setVisible(true);
  }
}
```

JFileChooser

Swing provides the javax.swing.JFileChooser class that displays a dialog box from which the user can navigate through the file system and select files for loading or saving.



Using JFileChooser

Creating a JFileChooser: Using JFileChooser's no-arg constructor.

Displaying an Open File Dialog:

The file dialog box can appear in two types: open and save. The open type is for opening a file, and the save type is for storing a file. To create an open file dialog, use the following method:

public int showOpenDialog(Component parent)

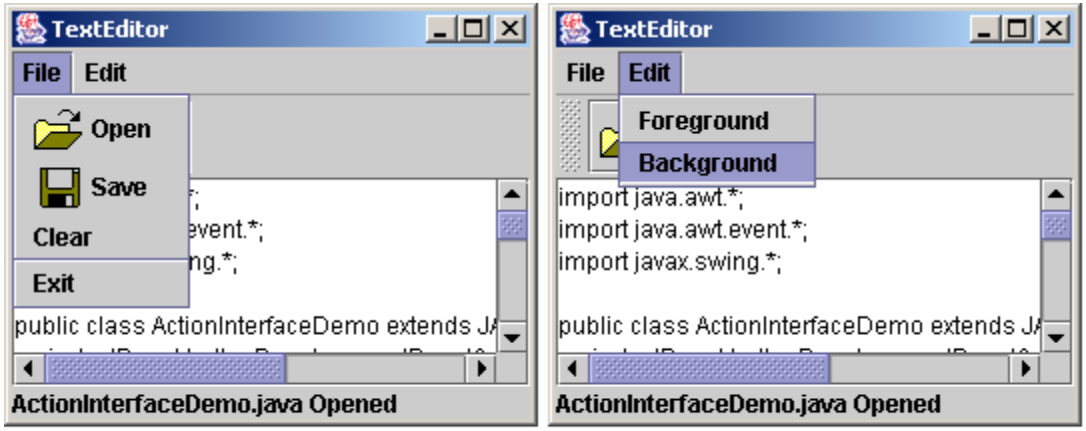
This method creates a dialog box that contains an instance of JFileChooser for opening a file. The method returns an int value, either APPROVE_OPTION or CANCEL_OPTION, which indicates whether the OK button or the Cancel button was clicked.

Displaying a Save File Dialog:

public int showSaveDialog(Component parent)

Example: Creating a Text Editor

Problem: This example uses Swing menus, tool bar, file chooser, and color chooser to create a simple text editor, as shown in Figure 34.23, which allows the user to open and save text files, clear text, and change the color and font of the text.



Creating Internal Frames

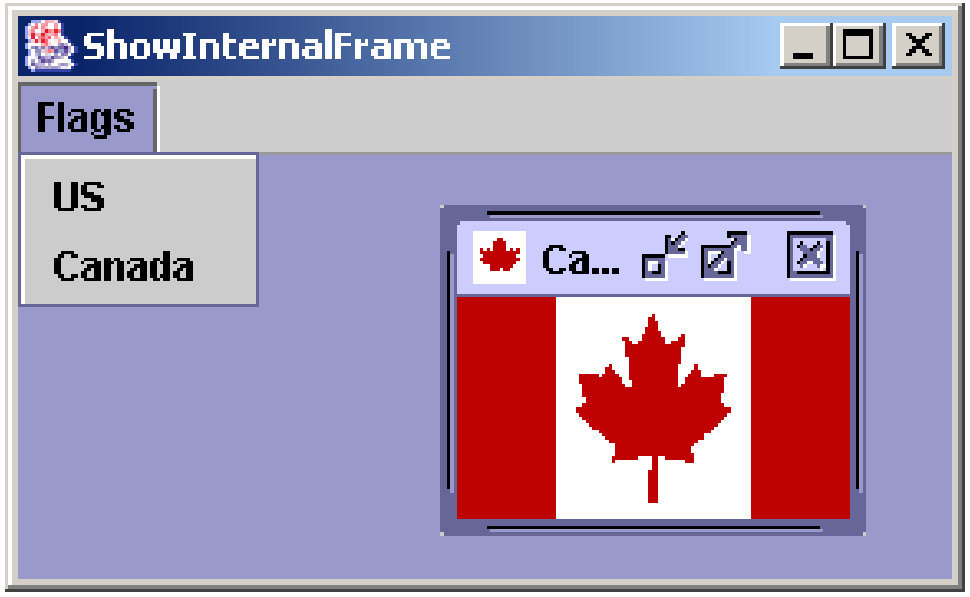
The JInternalFrame class is almost the same as the external JFrame class.

The components are added to the internal frame in the same way as they are added to the external frame.

The internal frame can have menus, the title, the Close icon, the Minimize icon, and the Maximize icon just like the external frame.

Example: Creating Internal Frames

Problem: This example creates internal frames to display flags in an applet. You can select flags from the Flags menu. Clicking a menu item causes a flag to be displayed in an internal frame.



```
// Create JDesktopPane to hold the internal frame
private JDesktopPane desktop = new JDesktopPane();
private JInternalFrame internalFrame =
  new JInternalFrame("US", true, true, true, true);
```

```
desktop.add(internalFrame);

this.setSize(new Dimension(400, 300));
this.getContentPane().add(desktop, BorderLayout.CENTER);

jlblImage.setIcon(USIcon);
internalFrame.setFrameIcon(USIcon);

internalFrame.add(jlblImage);
internalFrame.setLocation(20, 20);
internalFrame.setSize(100, 100);
internalFrame.setVisible(true);
```