Java Programming

Graphical User Interface

(GUI-3)

Introduction

- Jframe
- JOptionPane
- Jlabel
- JTextField
- Jbutton
- JCkeckBox
- JRadioButton
- JComboBox
- JList
- JPanel
- JTextArea
- Jmenu
- JMenuBar
- JDeskPane
- JInternalFrame
- JTabbedPane
- JPopupMenu
- JToolBar
- JSlider
- JTree
- JTable

Using Menus with Frames

- Menus are an integral part of GUIs.
- Allow the user to perform actions without unnecessarily cluttering a GUI with extra components.
- In Swing GUIs, menus can be attached only to objects of the classes that provide method `setJMenuBar`.
  - Two such classes are `JFrame` and `JApplet`.
- The classes used to declare menus are `JMenuBar`, `JMenu`, `JMenuItem`, `JCheckBoxMenuItem` and class `JRadioButtonMenuItem`.

Using Menus with Frames

- Class `JMenuBar` (a subclass of `JComponent`) manage a menu bar, which is a container for menus.

- Class `JMenu` (a subclass of `javax.swing.JMenuItem`)—menus.
  - Menus contain menu items and are added to menu bars or to other menus as submenus.

- Class `JMenuItem` (a subclass of `javax.swing.AbstractButton`)—menu items.
  - A menu item causes an action event when clicked.
  - Can also be a submenu that provides more menu items from which the user can select.

Using Menus with Frames

- Class `JCheckBoxMenuItem` (a subclass of `javax.swing.JMenuItem`)—menu items that can be toggled on or off.

- Class `JRadioButtonMenuItem` (a subclass of `javax.swing.JMenuItem`)—menu items that can be toggled on or off like `JCheckBoxMenuItem`s.
  - When multiple `JRadioButtonMenuItem`s are maintained as part of a `ButtonGroup`, only one item in the group can be selected at a given time.

- Mnemonics can provide quick access to a menu or menu item from the keyboard.
  - Can be used with all subclasses of `javax.swing.AbstractButton`.

- `JMenu` method `setMnemonic` (inherited from class `AbstractButton`) indicates the mnemonic for a menu.

Using Menus with Frames

```java
// Fig. 25.5: MenuFrame.java
// Demonstrating menus.
import java.awt.Color;
import java.awt.Font;
import java.awt.BorderLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import javax.swing.JFrame;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JLabel;
import javax.swing.SwingConstants;
import javax.swing.ButtonGroup;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JMenuBar;

```

Fig. 25.5 | JMenus and mnemonics. (Part 1 of 10.)

Using Menus with Frames

```java
public class MenuFrame extends JFrame
{
   private final Color[] colorValues =
      { Color.BLACK, Color.BLUE, Color.RED, Color.GREEN };
   private JRadioButtonMenuItem[] colorItems; // color menu items
   private JRadioButtonMenuItem[] fonts; // font menu items
   private JCheckBoxMenuItem[] styleItems; // font style menu items
   private JLabel displayJLabel; // displays sample text
   private ButtonGroup fontButtonGroup; // manages font menu items
   private ButtonGroup colorButtonGroup; // manages color menu items
   private int style; // used to create style for font

```

Fig. 25.5 | JMenus and mnemonics. (Part 2 of 10.)

Using Menus with Frames

```java
   // no-argument constructor set up GUI
   public MenuFrame()
   {
      super( "Using JMenus" );

      JMenu fileMenu = new JMenu( "File" ); // create file menu
      fileMenu.setMnemonic( 'F' ); // set mnemonic to F

      // create About... menu item
      JMenuItem aboutItem = new JMenuItem( "About..." );
      aboutItem.setMnemonic( 'A' ); // set mnemonic to A
      fileMenu.add( aboutItem ); // add about item to file menu
      aboutItem.addActionListener(

         new ActionListener() // anonymous inner class
         {
            // display message dialog when user selects About...
            public void actionPerformed( ActionEvent event )
            {
               JOptionPane.showMessageDialog( MenuFrame.this,
                  "This is an example\nof using menus",
                  "About", JOptionPane.PLAIN_MESSAGE );
            } // end method actionPerformed
         } // end anonymous inner class
      ); // end call to addActionListener
```

Fig. 25.5 | JMenus and mnemonics. (Part 3 of 10.)

Using Menus with Frames

```java

      JMenuItem exitItem = new JMenuItem( "Exit" ); // create exit item
      exitItem.setMnemonic( 'x' ); // set mnemonic to x
      fileMenu.add( exitItem ); // add exit item to file menu
      exitItem.addActionListener(

         new ActionListener() // anonymous inner class
         {
            // terminate application when user clicks exitItem
            public void actionPerformed( ActionEvent event )
            {
               System.exit( 0 ); // exit application
            } // end method actionPerformed
         } // end anonymous inner class
      ); // end call to addActionListener

      JMenuBar bar = new JMenuBar(); // create menu bar
      setJMenuBar( bar ); // add menu bar to application
      bar.add( fileMenu ); // add file menu to menu bar

      JMenu formatMenu = new JMenu( "Format" ); // create format menu
      formatMenu.setMnemonic( 'r' ); // set mnemonic to r
```

Fig. 25.5 | JMenus and mnemonics. (Part 4 of 10.)

Using Menus with Frames

```
      // array listing string colors
      String[] colors = { "Black", "Blue", "Red", "Green" };

      JMenu colorMenu = new JMenu( "Color" ); // create color menu
      colorMenu.setMnemonic( 'C' ); // set mnemonic to C

      // create radio button menu items for colors
      colorItems = new JRadioButtonMenuItem[ colors.length ];
      colorButtonGroup = new ButtonGroup(); // manages colors
      ItemHandler itemHandler = new ItemHandler(); // handler for colors

      // create color radio button menu items
      for ( int count = 0; count < colors.length; count++ )
      {
         colorItems[ count ] =
            new JRadioButtonMenuItem( colors[ count ] ); // create item
         colorMenu.add( colorItems[ count ] ); // add item to color menu
         colorButtonGroup.add( colorItems[ count ] ); // add to group
         colorItems[ count ].addActionListener( itemHandler );
      } // end for

```

Fig. 25.5 | JMenus and mnemonics. (Part 5 of 10.)

Using Menus with Frames

```java
      colorItems[ 0 ].setSelected( true ); // select first Color item

      formatMenu.add( colorMenu ); // add color menu to format menu
      formatMenu.addSeparator(); // add separator in menu

      // array listing font names
      String[] fontNames = { "Serif", "Monospaced", "SansSerif" };
      JMenu fontMenu = new JMenu( "Font" ); // create font menu
      fontMenu.setMnemonic( 'n' ); // set mnemonic to n

      // create radio button menu items for font names
      fonts = new JRadioButtonMenuItem[ fontNames.length ];
      fontButtonGroup = new ButtonGroup(); // manages font names

      // create Font radio button menu items
      for ( int count = 0; count < fonts.length; count++ )
      {
         fonts[ count ] = new JRadioButtonMenuItem( fontNames[ count ] );
         fontMenu.add( fonts[ count ] ); // add font to font menu
         fontButtonGroup.add( fonts[ count ] ); // add to button group
         fonts[ count ].addActionListener( itemHandler ); // add handler
      } // end for

```

Fig. 25.5 | JMenus and mnemonics. (Part 6 of 10.)

Using Menus with Frames

```
      fonts[ 0 ].setSelected( true ); // select first Font menu item
      fontMenu.addSeparator(); // add separator bar to font menu

      String[] styleNames = { "Bold", "Italic" }; // names of styles
      styleItems = new JCheckBoxMenuItem[ styleNames.length ];
      StyleHandler styleHandler = new StyleHandler(); // style handler

      // create style checkbox menu items
      for ( int count = 0; count < styleNames.length; count++ )
      {
         styleItems[ count ] =
            new JCheckBoxMenuItem( styleNames[ count ] ); // for style
         fontMenu.add( styleItems[ count ] ); // add to font menu
         styleItems[ count ].addItemListener( styleHandler ); // handler
      } // end for

      formatMenu.add( fontMenu ); // add Font menu to Format menu
      bar.add( formatMenu ); // add Format menu to menu bar

```

Fig. 25.5 | JMenus and mnemonics. (Part 7 of 10.)

Using Menus with Frames

```
      // set up label to display text
      displayJLabel = new JLabel( "Sample Text", SwingConstants.CENTER );
      displayJLabel.setForeground( colorValues[ 0 ] );
      displayJLabel.setFont( new Font( "Serif", Font.PLAIN, 72 ) );

      getContentPane().setBackground( Color.CYAN ); // set background
      add( displayJLabel, BorderLayout.CENTER ); // add displayJLabel
   } // end MenuFrame constructor

   // inner class to handle action events from menu items
   private class ItemHandler implements ActionListener
   {
      // process color and font selections
      public void actionPerformed( ActionEvent event )
      {
         // process color selection
         for ( int count = 0; count < colorItems.length; count++ )
         {
            if ( colorItems[ count ].isSelected() )
            {
               displayJLabel.setForeground( colorValues[ count ] );
               break;
            } // end if
         } // end for
```

Fig. 25.5 | JMenus and mnemonics. (Part 8 of 10.)

Using Menus with Frames

```

         // process font selection
         for ( int count = 0; count < fonts.length; count++ )
         {
            if ( event.getSource() == fonts[ count ] )
            {
               displayJLabel.setFont(
                  new Font( fonts[ count ].getText(), style, 72 ) );
            } // end if
         } // end for

         repaint(); // redraw application
      } // end method actionPerformed
   } // end class ItemHandler

   // inner class to handle item events from checkbox menu items
   private class StyleHandler implements ItemListener
   {
      // process font style selections
      public void itemStateChanged( ItemEvent e )
      {
         String name = displayJLabel.getFont().getName(); // current Font
         Font font; // new font based on user selections

```

Fig. 25.5 | JMenus and mnemonics. (Part 9 of 10.)

Using Menus with Frames

```
         // determine which items are checked and create Font
         if ( styleItems[ 0 ].isSelected() &&
              styleItems[ 1 ].isSelected() )
            font = new Font( name, Font.BOLD + Font.ITALIC, 72 );
         else if ( styleItems[ 0 ].isSelected() )
            font = new Font( name, Font.BOLD, 72 );
         else if ( styleItems[ 1 ].isSelected() )
            font = new Font( name, Font.ITALIC, 72 );
         else
            font = new Font( name, Font.PLAIN, 72 );

         displayJLabel.setFont( font );
         repaint(); // redraw application
      } // end method itemStateChanged
   } // end class StyleHandler
} // end class MenuFrame
```

Fig. 25.5 | JMenus and mnemonics. (Part 10 of 10.)

Using Menus with Frames

```java
// Fig. 25.6: MenuTest.java
// Testing MenuFrame.
import javax.swing.JFrame;

public class MenuTest
{
   public static void main( String[] args )
   {
      MenuFrame menuFrame = new MenuFrame(); // create MenuFrame
      menuFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      menuFrame.setSize( 500, 200 ); // set frame size
      menuFrame.setVisible( true ); // display frame
   } // end main
} // end class MenuTest
```

Fig. 25.6 | Test class for MenuFrame. (Part 1 of 2.)

Using Menus with Frames

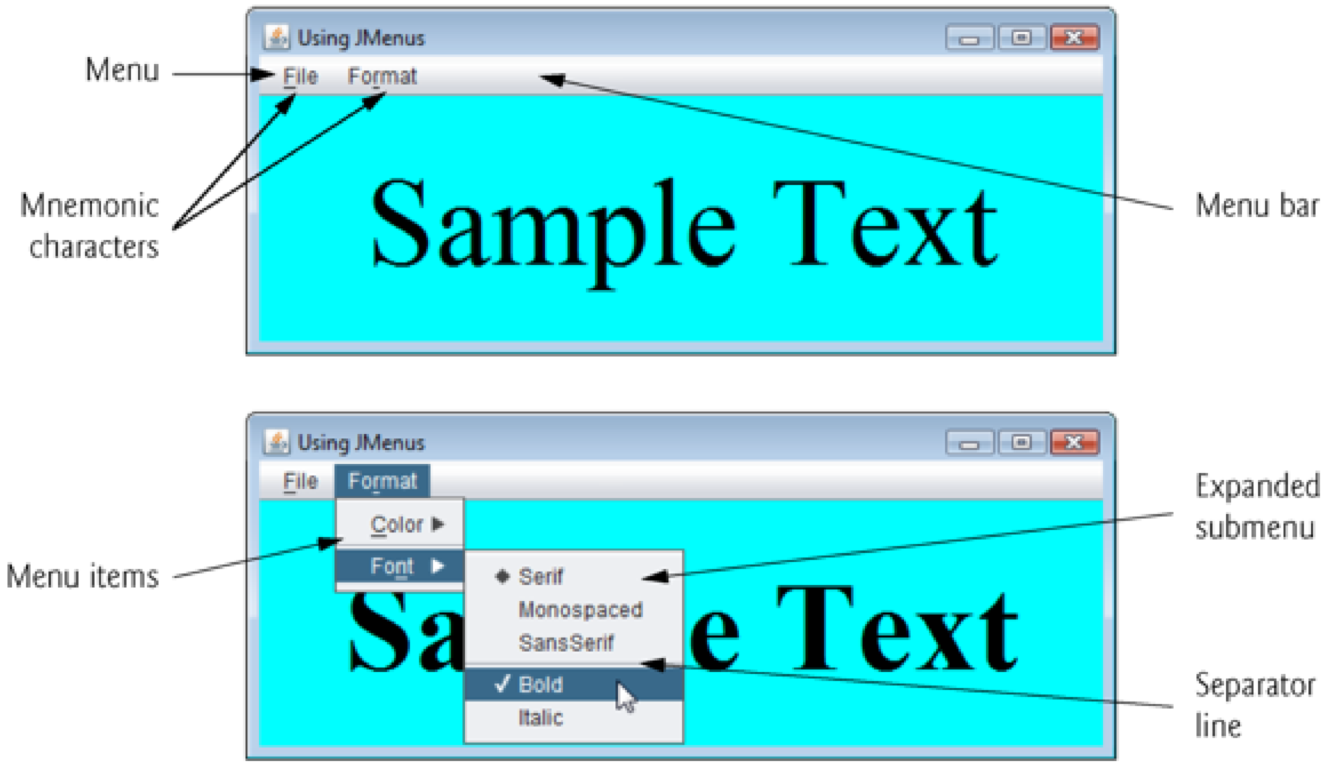


Fig. 25.6 | Test class for `MenuFrame`. (Part 2 of 2.)

JDesktopPane and JInternalFrame

- Multiple-document interface (MDI)
  - a main window (called the parent window) containing other windows (called child windows), to manage several open documents that are being processed in parallel.

- Swing's `JDesktopPane` and `JInternalFrame` classes implement multiple-document interfaces.

JDesktopPane and JInternalFrame

```
// Fig. 25.11: DesktopFrame.java
// Demonstrating JDesktopPane.
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.util.Random;
import javax.swing.JFrame;
import javax.swing.JDesktopPane;
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JInternalFrame;
import javax.swing.JPanel;
import javax.swing.ImageIcon;

public class DesktopFrame extends JFrame
{
   private JDesktopPane theDesktop;

```

Fig. 25.11 | Multiple-document interface. (Part 1 of 5.)

JDesktopPane and JInternalFrame

```
    // set up GUI
    public DesktopFrame()
    {
       super( "Using a JDesktopPane" );

       JMenuBar bar = new JMenuBar(); // create menu bar
       JMenu addMenu = new JMenu( "Add" ); // create Add menu
       JMenuItem newFrame = new JMenuItem( "Internal Frame" );

       addMenu.add( newFrame ); // add new frame item to Add menu
       bar.add( addMenu ); // add Add menu to menu bar
       setJMenuBar( bar ); // set menu bar for this application

       theDesktop = new JDesktopPane(); // create desktop pane
       add( theDesktop ); // add desktop pane to frame

       // set up listener for newFrame menu item
       newFrame.addActionListener(

```

Fig. 25.11 | Multiple-document interface. (Part 2 of 5.)

JDesktopPane and JInternalFrame

```
            new ActionListener() // anonymous inner class
            {
               // display new internal window
               public void actionPerformed( ActionEvent event )
               {
                  // create internal frame
                  JInternalFrame frame = new JInternalFrame(
                     "Internal Frame", true, true, true, true );

                  MyJPanel panel = new MyJPanel(); // create new panel
                  frame.add( panel, BorderLayout.CENTER ); // add panel
                  frame.pack(); // set internal frame to size of contents

                  theDesktop.add( frame ); // attach internal frame
                  frame.setVisible( true ); // show internal frame
               } // end method actionPerformed
            } // end anonymous inner class
         ); // end call to addActionListener
      } // end DesktopFrame constructor
   } // end class DesktopFrame
```

Fig. 25.11 | Multiple-document interface. (Part 3 of 5.)

JDesktopPane and JInternalFrame

```
// class to display an ImageIcon on a panel
class MyJPanel extends JPanel
{
   private static Random generator = new Random();
   private ImageIcon picture; // image to be displayed
   private final static String[] images = { "yellowflowers.png",
      "purpleflowers.png", "redflowers.png", "redflowers2.png",
      "lavenderflowers.png" };

   // load image
   public MyJPanel()
   {
      int randomNumber = generator.nextInt( images.length );
      picture = new ImageIcon( images[ randomNumber ] ); // set icon
   } // end MyJPanel constructor

   // display imageIcon on panel
   public void paintComponent( Graphics g )
   {
      super.paintComponent( g );
      picture.paintIcon( this, g, 0, 0 ); // display icon
   } // end method paintComponent

```

Fig. 25.11 | Multiple-document interface. (Part 4 of 5.)

JDesktopPane and JInternalFrame

```java
    // return image dimensions
    public Dimension getPreferredSize()
    {
       return new Dimension( picture.getIconWidth(),
          picture.getIconHeight() );
    } // end method getPreferredSize
 } // end class MyJPanel
```

Fig. 25.11 | Multiple-document interface. (Part 5 of 5.)

JDesktopPane and JInternalFrame

```
// Fig. 25.12: DesktopTest.java
// Demonstrating JDesktopPane.
import javax.swing.JFrame;

public class DesktopTest
{
   public static void main( String[] args )
   {
      DesktopFrame desktopFrame = new DesktopFrame();
      desktopFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      desktopFrame.setSize( 600, 480 ); // set frame size
      desktopFrame.setVisible( true ); // display frame
   } // end main
} // end class DesktopTest
```

Fig. 25.12 | Test class for DeskTopFrame. (Part 1 of 3.)

JDesktopPane and JInternalFrame

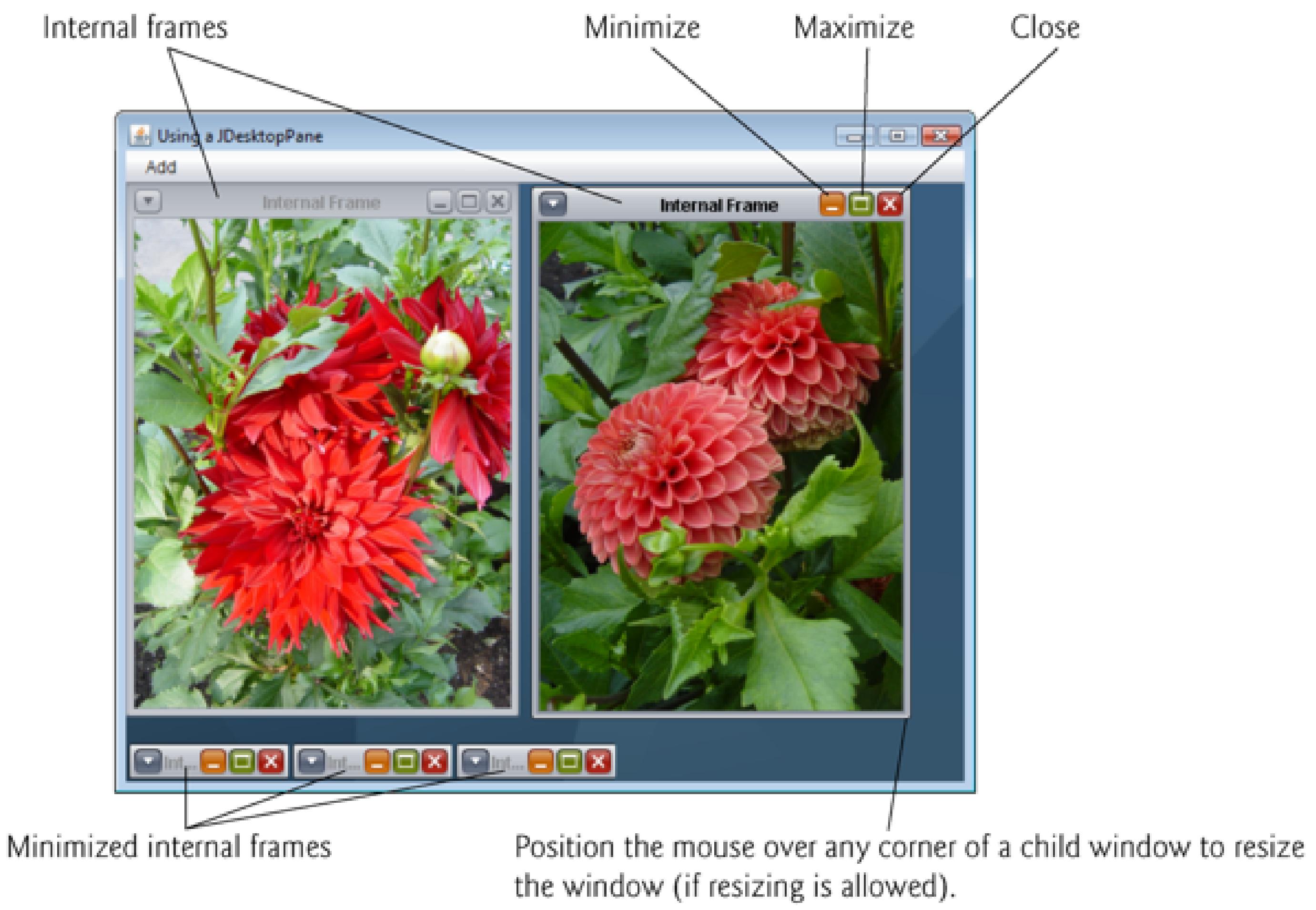

Internal frames
Minimize
Maximize
Close
Using a JDesktopPane
Add
Internal Frame
Internal Frame
Minimized internal frames
Position the mouse over any corner of a child window to resize the window (if resizing is allowed).

Fig. 25.12 | Test class for `DeskTopFrame`. (Part 2 of 3.)

JDesktopPane and JInternalFrame

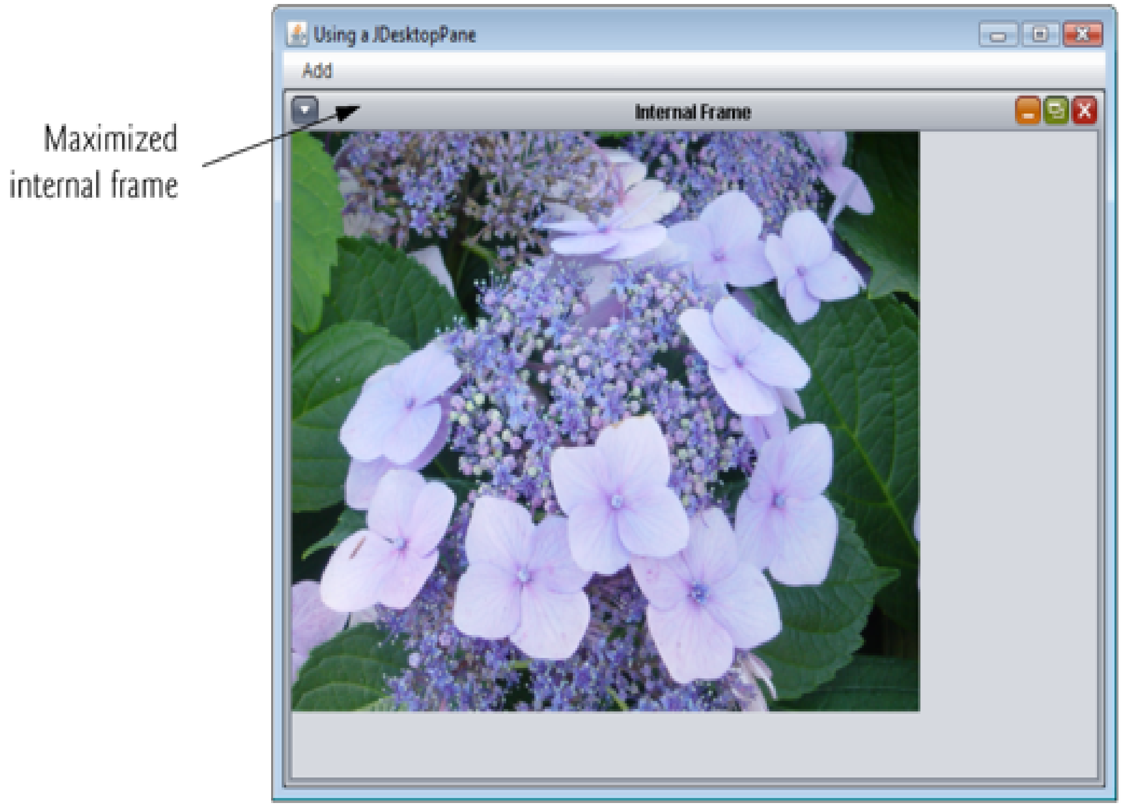


Fig. 25.12 | Test class for `DeskTopFrame`. (Part 3 of 3.)

JTabbedPane

- A `JTabbedPane` arranges GUI components into layers, of which only one is visible at a time.
- Users access each layer by clicking a tab.
- The tabs appear at the top by default but also can be positioned at the left, right or bottom of the `JTabbedPane`.
- Any component can be placed on a tab.
    - If the component is a container, such as a panel, it can use any layout manager to lay out several components on the tab.
- Class `JTabbedPane` is a subclass of `JComponent`.

JTabbedPane

```java
// Fig. 25.13: JTabbedPaneFrame.java
// Demonstrating JTabbedPane.
import java.awt.BorderLayout;
import java.awt.Color;
import javax.swing.JFrame;
import javax.swing.JTabbedPane;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.SwingConstants;

public class JTabbedPaneFrame extends JFrame
{
   // set up GUI
   public JTabbedPaneFrame()
   {
      super( "JTabbedPane Demo " );

      JTabbedPane tabbedPane = new JTabbedPane(); // create JTabbedPane

```

Fig. 25.13 | JTabbedPane used to organize GUI components. (Part 1 of 3.)

JTabbedPane

```
      // set up panel1 and add it to JTabbedPane
      JLabel label1 = new JLabel( "panel one", SwingConstants.CENTER );
      JPanel panel1 = new JPanel(); // create first panel
      panel1.add( label1 ); // add label to panel
      tabbedPane.addTab( "Tab One", null, panel1, "First Panel" );

      // set up panel2 and add it to JTabbedPane
      JLabel label2 = new JLabel( "panel two", SwingConstants.CENTER );
      JPanel panel2 = new JPanel(); // create second panel
      panel2.setBackground( Color.YELLOW ); // set background to yellow
      panel2.add( label2 ); // add label to panel
      tabbedPane.addTab( "Tab Two", null, panel2, "Second Panel" );

```

Fig. 25.13 | JTabbedPane used to organize GUI components. (Part 2 of 3.)

JTabbedPane

```java
      // set up panel3 and add it to JTabbedPane
      JLabel label3 = new JLabel( "panel three" );
      JPanel panel3 = new JPanel(); // create third panel
      panel3.setLayout( new BorderLayout() ); // use borderlayout
      panel3.add( new JButton( "North" ), BorderLayout.NORTH );
      panel3.add( new JButton( "West" ), BorderLayout.WEST );
      panel3.add( new JButton( "East" ), BorderLayout.EAST );
      panel3.add( new JButton( "South" ), BorderLayout.SOUTH );
      panel3.add( label3, BorderLayout.CENTER );
      tabbedPane.addTab( "Tab Three", null, panel3, "Third Panel" );

      add( tabbedPane ); // add JTabbedPane to frame
   } // end JTabbedPaneFrame constructor
} // end class JTabbedPaneFrame
```

Fig. 25.13 | JTabbedPane used to organize GUI components. (Part 3 of 3.)

JTabbedPane

```java
// Fig. 25.14: JTabbedPaneDemo.java
// Demonstrating JTabbedPane.
import javax.swing.JFrame;

public class JTabbedPaneDemo
{
   public static void main( String[] args )
   {
      JTabbedPaneFrame tabbedPaneFrame = new JTabbedPaneFrame();
      tabbedPaneFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      tabbedPaneFrame.setSize( 250, 200 ); // set frame size
      tabbedPaneFrame.setVisible( true ); // display frame
   } // end main
} // end class JTabbedPaneDemo
```

Fig. 25.14 | Test class for `JTabbedPaneFrame`. (Part 1 of 2.)

JTabbedPane

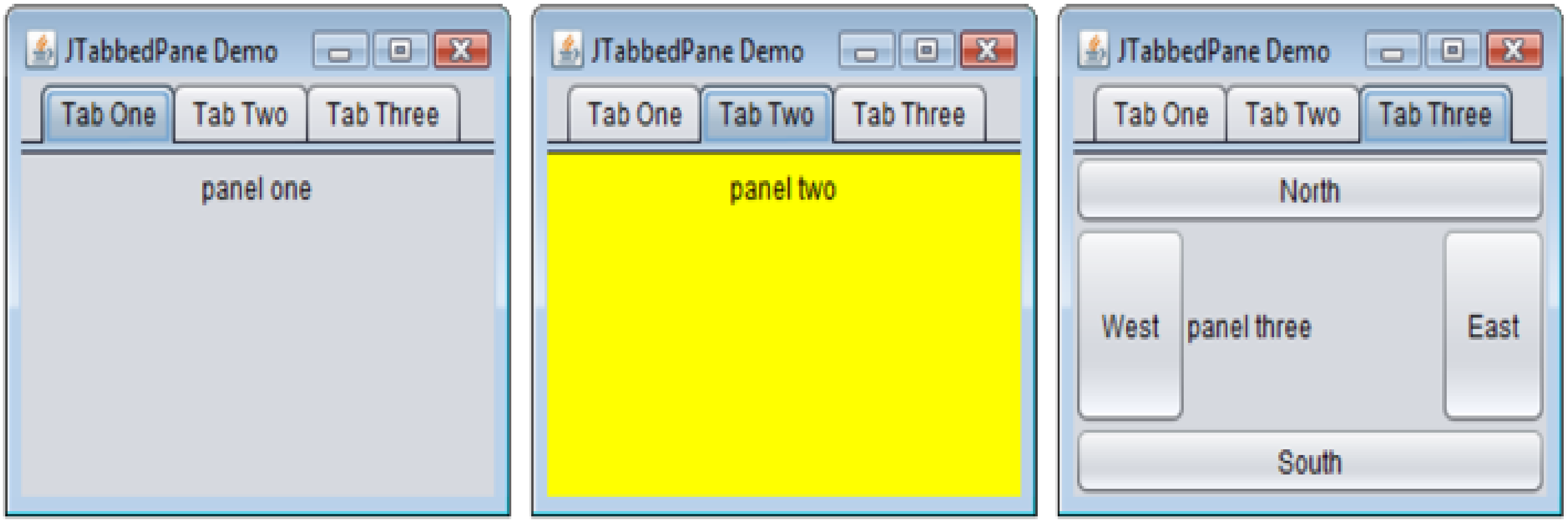
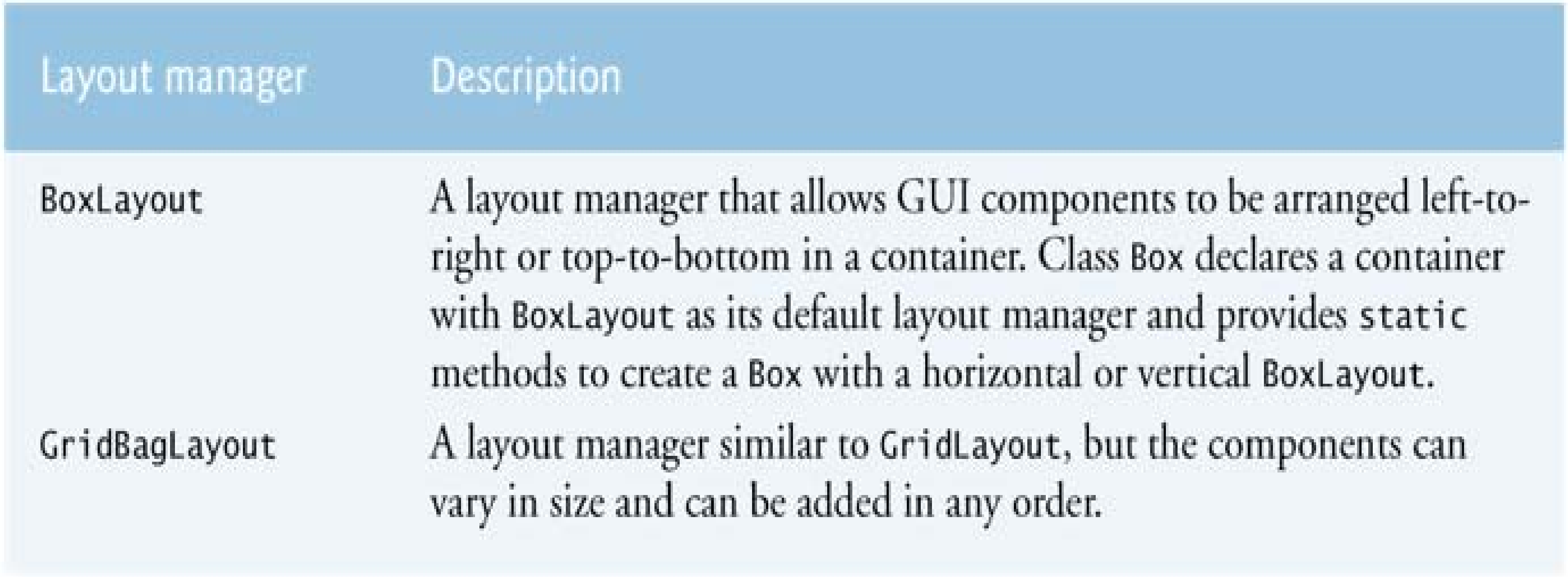


Fig. 25.14 | Test class for `JTabbedPaneFrame`. (Part 2 of 2.)

Layout Managers: BoxLayout and GridBagLayout



Layout manager	Description
`BoxLayout`	A layout manager that allows GUI components to be arranged left-to-right or top-to-bottom in a container. Class `Box` declares a container with `BoxLayout` as its default layout manager and provides `static` methods to create a `Box` with a horizontal or vertical `BoxLayout`.
`GridBagLayout`	A layout manager similar to `GridLayout`, but the components can vary in size and can be added in any order.

Fig. 25.15 | Additional layout managers.

Layout Managers: BoxLayout and GridBagLayout

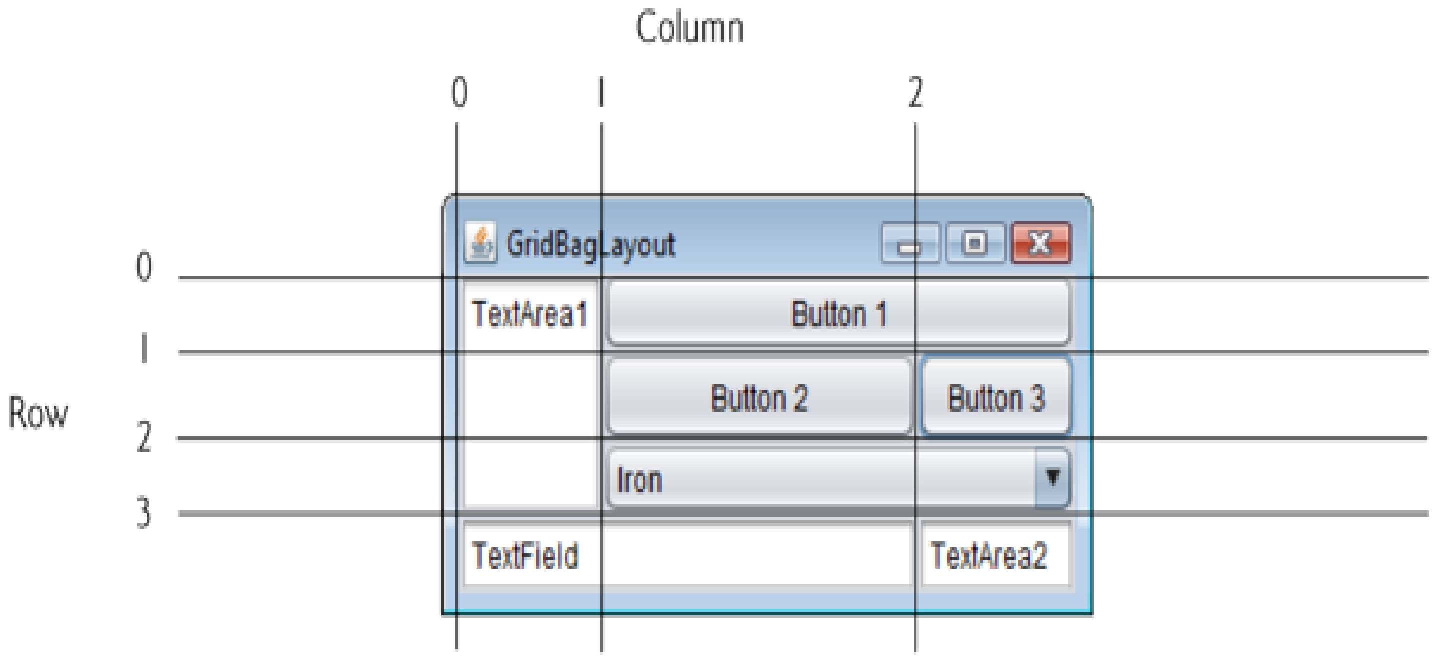


Fig. 25.18 | Designing a GUI that will use `GridBagLayout`.