Java GUI Programming
Part 1

Chapter 12 and 22

P. Deitel , H. Deitel - Java How To Program, 10th Edition

Edited by Ehsan Edalat

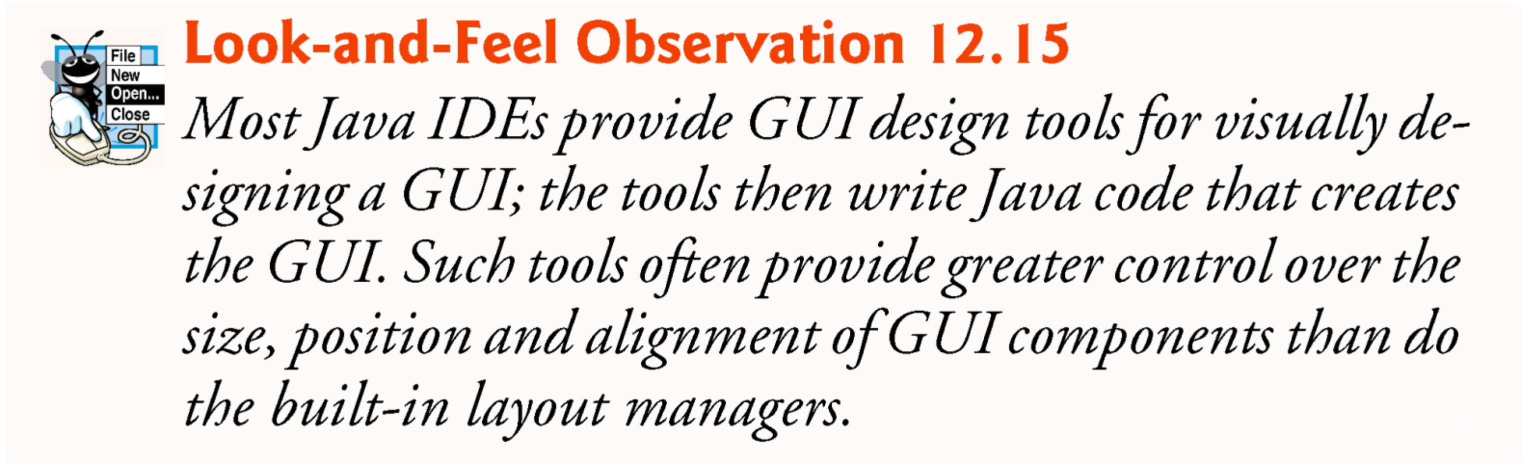
12.1 Introduction

- A graphical user interface (GUI) presents a user-friendly mechanism for interacting with an application.
  - Pronounced “GOO-ee”
  - Gives an application a distinctive “look-and-feel.”
  - Consistent, intuitive user-interface components give users a sense of familiarity
  - Learn new applications more quickly and use them more productively.

12.1 Introduction (cont.)

- Built from GUI components.
  - Sometimes called *controls* or *widgets*—*short* for window gadgets.
- User *interacts* via the mouse, the keyboard or another form of input, such as voice recognition.
- IDEs
  - Provide GUI design tools to specify a component's *size*, *location* and other attributes in a visual manner by using the mouse, keyboard and drag-and-drop.
  - Generate the GUI code for you.
  - Greatly simplify creating GUIs, but each IDE has different capabilities and generates different code.

12.1 Introduction (cont.)

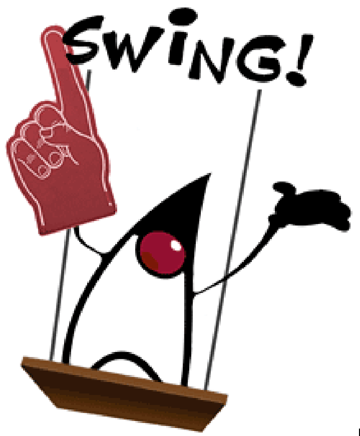


Look-and-Feel Observation 12.15

Most Java IDEs provide GUI design tools for visually designing a GUI; the tools then write Java code that creates the GUI. Such tools often provide greater control over the size, position and alignment of GUI components than do the built-in layout managers.

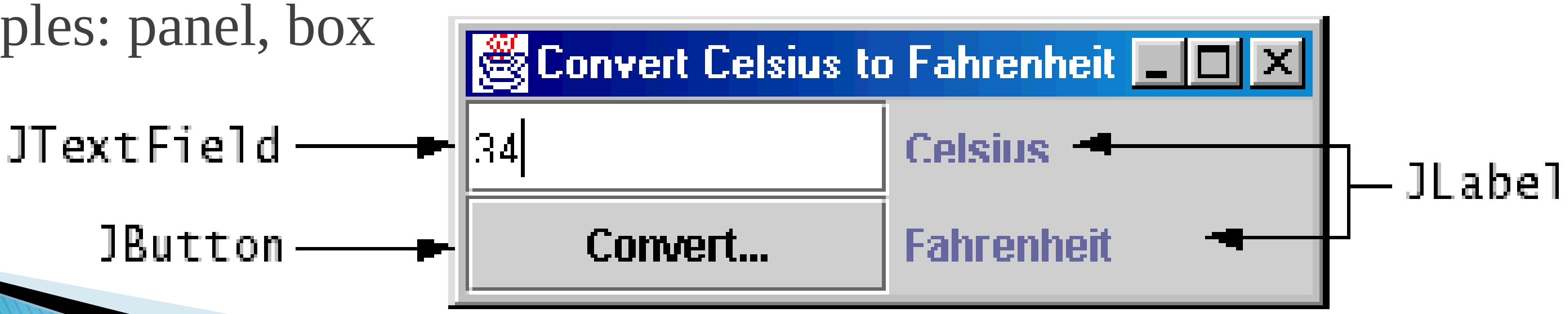
Java GUI History

- **Abstract Window Toolkit** (**AWT**): Sun's initial effort to create a set of cross-platform GUI classes. *(JDK 1.0 - 1.1)*
  - Maps general Java code to each operating system's real GUI system.
  - *Problems:* clunky to use.
- **Swing**: A newer GUI library written from the ground up that allows much more powerful graphics and GUI construction. *(JDK 1.2+)*
  - Paints GUI controls itself pixel-by-pixel rather than handing off to OS.
  - *Benefits:* Features; compatibility; OO design.
  - *Problem:* Both exist in Java now; easy to get them mixed up; still have to use both in various places.
- JavaFX, new approach in GUI programming!



GUI terminology

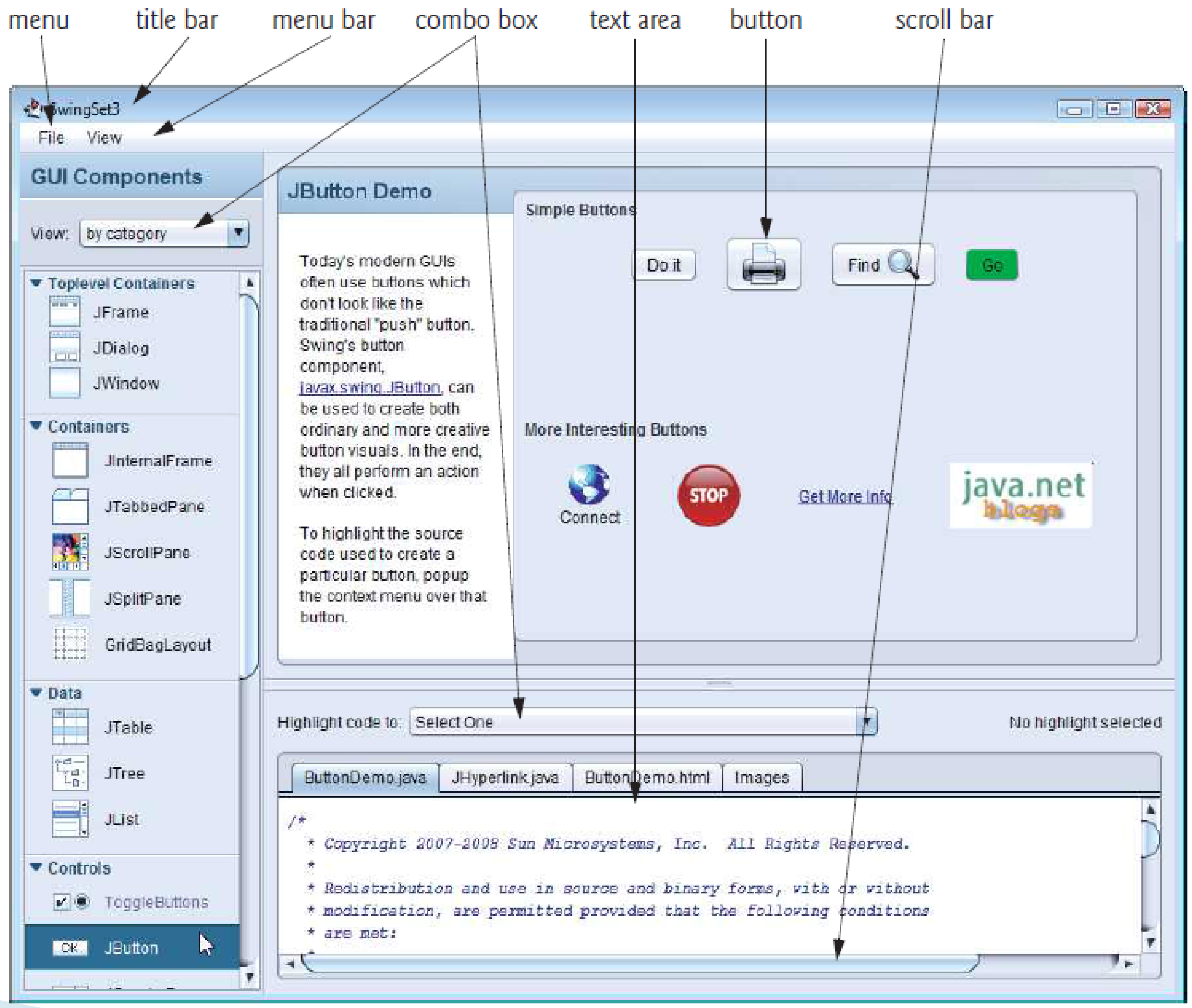
- **window**: A first-class citizen of the graphical desktop.
  - Also called a *top-level container*.
  - examples: frame, dialog box, applet
- **component**: A GUI widget that resides in a window.
  - Also called *controls* in many other languages.
  - examples: button, text box, label
- **container**: A logical grouping for storing components.
  - examples: panel, box



12.1 Introduction (cont.)

- Example of a GUI: SwingSet3 application (Fig. 12.1)
  `http://www.oracle.com/technetwork/java/javase/downloads/index.html`
- title bar at top contains the window's title.
- menu bar contains menus (File and View).
- In the top-right region of the window is a set of buttons
  - Typically, users press buttons to perform tasks.
- In the GUI Components area of the window is a combo box;
  - User can click the down arrow at the right side of the box to select from a list of items.

12.1 Introduction (cont.)

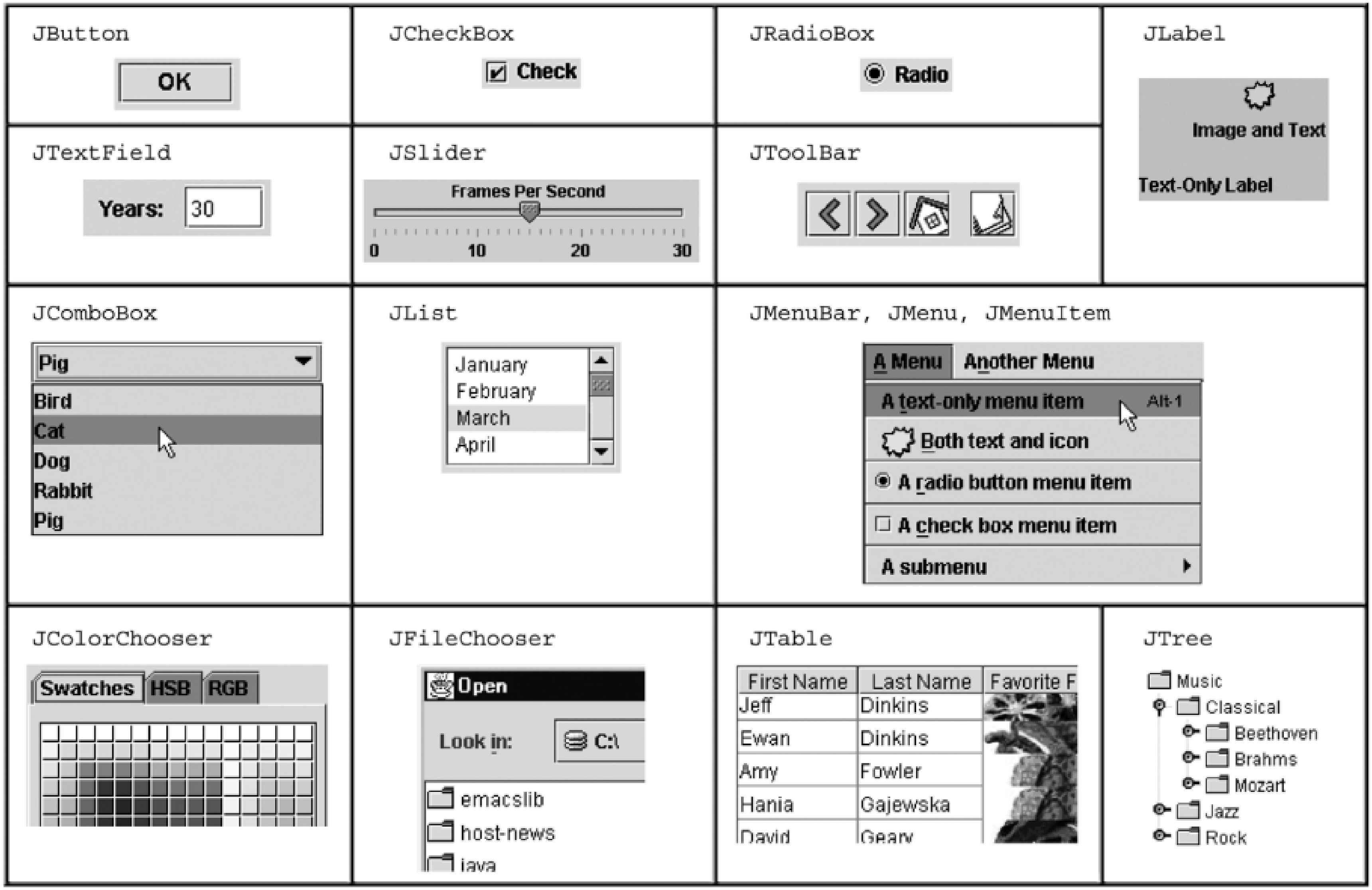


Today's modern GUIs often use buttons which don't look like the traditional "push" button. Swing's button component, javax.swing.JButton, can be used to create both ordinary and more creative button visuals. In the end, they all perform an action when clicked.

To highlight the source code used to create a particular button, popup the context menu over that button.

```
/*
 * Copyright 2007-2008 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
```

Components



First Name	Last Name	Favorite F
Jeff	Dinkins	
Ewan	Dinkins	
Amy	Fowler	
Hania	Gajewska	
David	Geary	

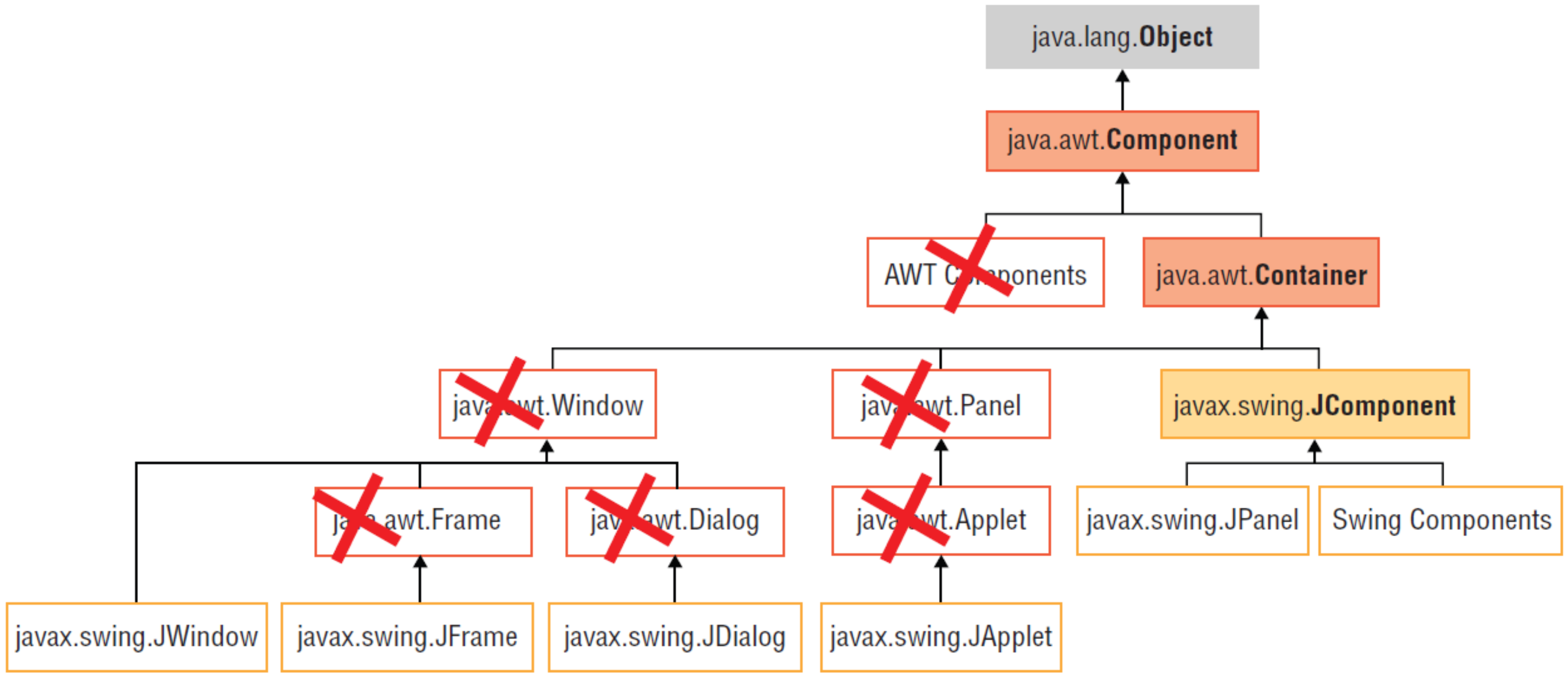
Swing inheritance hierarchy

```
import java.awt.*;
import javax.swing.*;
```

- `Component` (AWT)
  - `Window`
    - `Frame`
      - **`JFrame`** (Swing)
      - **`JDialog`**
  - `Container`
    - **`JComponent`** (Swing)

JButton	**JColorChooser**	**JFileChooser**
JComboBox	**JLabel**	**JList**
JMenuBar	**JOptionPane**	**JPanel**
JPopupMenu	**JProgressBar**	**JScrollbar**
JScrollPane	**JSlider**	**JSpinner**
JSplitPane	**JTabbedPane**	**JTable**
JToolbar	**JTree**	**JTextArea**
JTextField	...	

Swing inheritance hierarchy



Component properties

- Each has a get (or is) accessor and a set modifier method.
- examples: getColor, setFont, setEnabled, isVisible

name	type	description
background	`Color`	background color behind component
border	`Border`	border line around component
enabled	`boolean`	whether it can be interacted with
focusable	`boolean`	whether key text can be typed on it
font	`Font`	font used for text in component
foreground	`Color`	foreground color of component
height, width	`int`	component's current size in pixels
visible	`boolean`	whether component can be seen
tooltip text	`String`	text shown when hovering mouse
size, minimum / maximum / preferred size	`Dimension`	various sizes, size limits, or desired sizes that the component may take

12.2 Java's Nimbus Look-and-Feel

- Swing has a cross-platform look-and-feel known as Nimbus.
- We've configured our systems to use Nimbus as the default look-and-feel.

12.2 Java's Nimbus Look-and-Feel (cont.)

- Three ways to use Nimbus:
  - Set it as the default for all Java applications that run on your computer.
  - Set it as the look-and-feel when you launch an application by passing a command-line argument to the `java` command.
  - Set it as the look-and-feel programatically in your application (Section 22.6).

12.2 Java's Nimbus Look-and-Feel (cont.)

- To set Nimbus as the default for all Java applications:
  - Create a text file named `swing.properties` in the `lib` folder of both your JDK installation folder and your JRE installation folder.
  - Place the following line of code in the file:

```
swing.defaultlaf=
  com.sun.java.swing.plaf.nimbus.
  NimbusLookAndFeel
```

- In addition to the standalone JRE, there is a JRE nested in your JDK's installation folder. If you are using an IDE that depends on the JDK (e.g., NetBeans), you may also need to place the `swing.properties` file in the nested `jre` folder's `lib` folder.

12.2 Java’s Nimbus Look-and-Feel (cont.)

- To select Nimbus on an application-by-application basis:
  - Place the following command-line argument after the `java` command and before the application’s name when you run the application:

```
-Dswing.defaultlaf=
   com.sun.java.swing.plaf.nimbus.NimbusLookAnd
   Feel
```

12.3 Simple GUI-Based Input/Output with `JOptionPane`

- Most applications use windows or dialog boxes (also called dialogs) to interact with the user.
- JOptionPane (package `javax.swing`) provides prebuilt dialog boxes for input and output
  - Displayed via `static JOptionPane` methods.
- Figure 12.2 uses two input dialogs to obtain integers from the user and a message dialog to display the sum of the integers the user enters.
- Sample Code … .

12.3 Simple GUI-Based Input/Output with `JOptionPane`

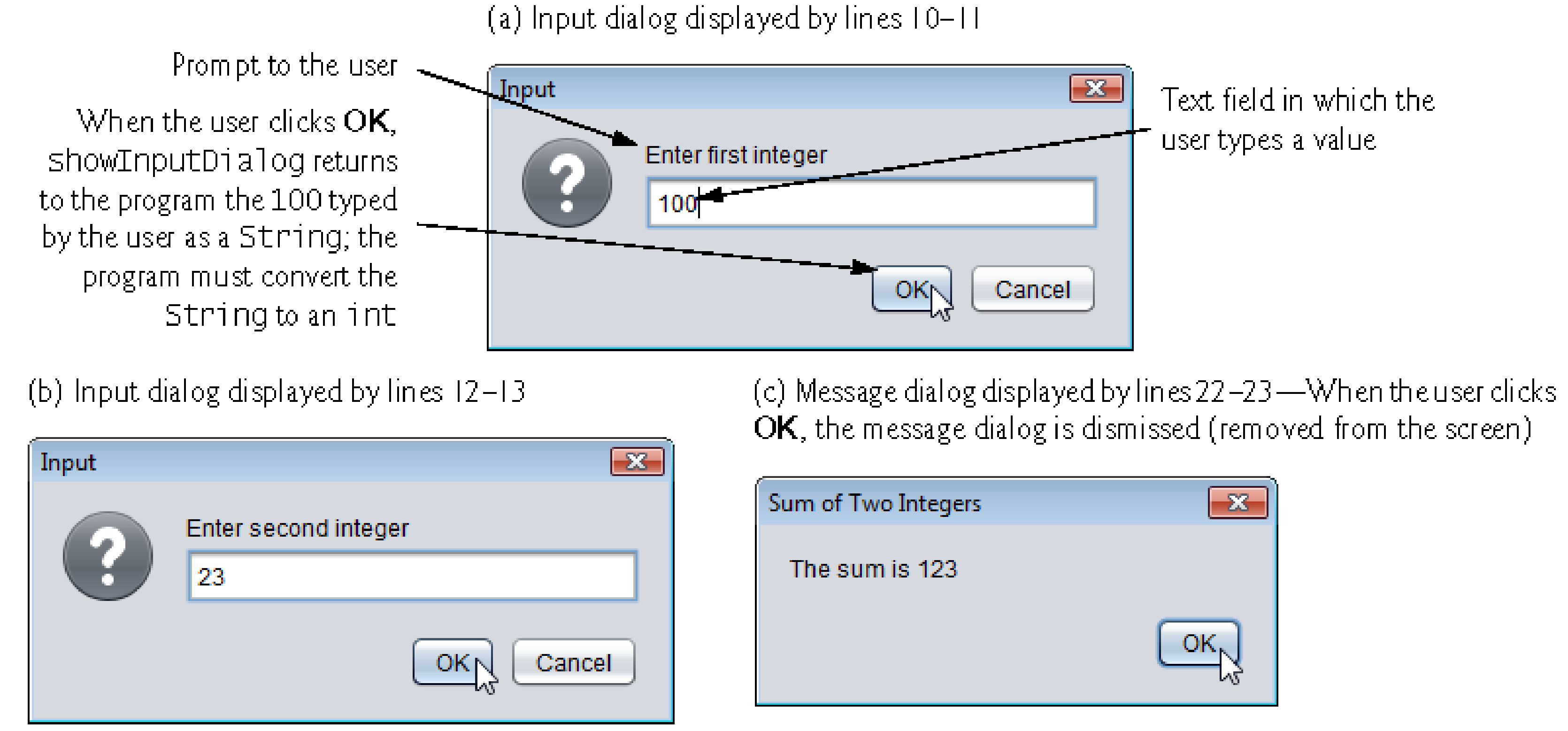
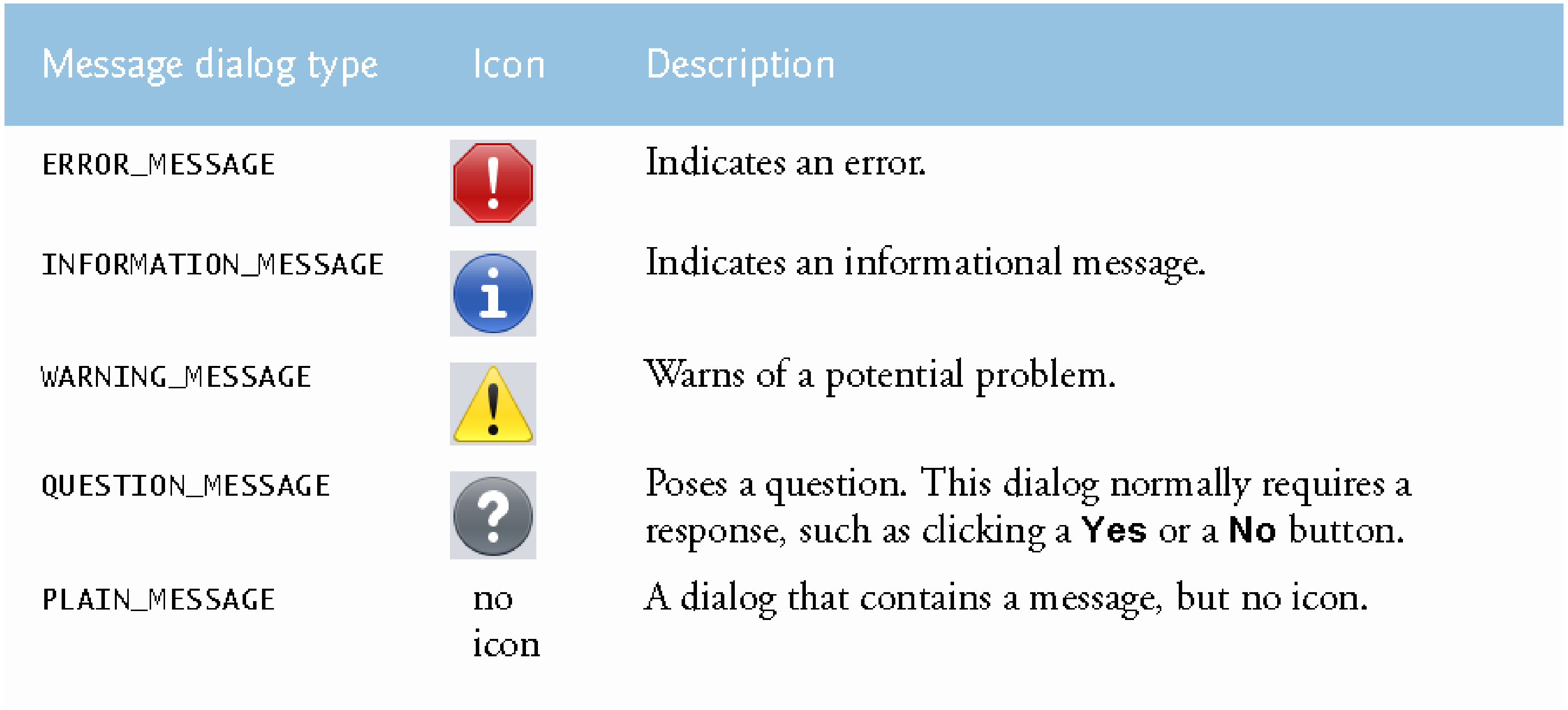

(a) Input dialog displayed by lines 10–11
Prompt to the user
When the user clicks **OK**, `showInputDialog` returns to the program the 100 typed by the user as a `String`; the program must convert the `String` to an `int`
Text field in which the user types a value
Input
Enter first integer
100
OK
Cancel
(b) Input dialog displayed by lines 12–13
Input
Enter second integer
23
OK
Cancel
(c) Message dialog displayed by lines 22–23—When the user clicks **OK**, the message dialog is dismissed (removed from the screen)
Sum of Two Integers
The sum is 123
OK

Fig. 12.2 | Addition program that uses **`JOptionPane`** for input and output. (Part 3 of 3.)

12.3 Simple GUI-Based Input/Output with `JOptionPane`



Message dialog type	Icon	Description
`ERROR_MESSAGE`	🛑	Indicates an error.
`INFORMATION_MESSAGE`	ℹ	Indicates an informational message.
`WARNING_MESSAGE`	⚠	Warns of a potential problem.
`QUESTION_MESSAGE`	❓	Poses a question. This dialog normally requires a response, such as clicking a **Yes** or a **No** button.
`PLAIN_MESSAGE`	no icon	A dialog that contains a message, but no icon.

Fig. 12.3 | `JOptionPane` `static` constants for message dialogs.

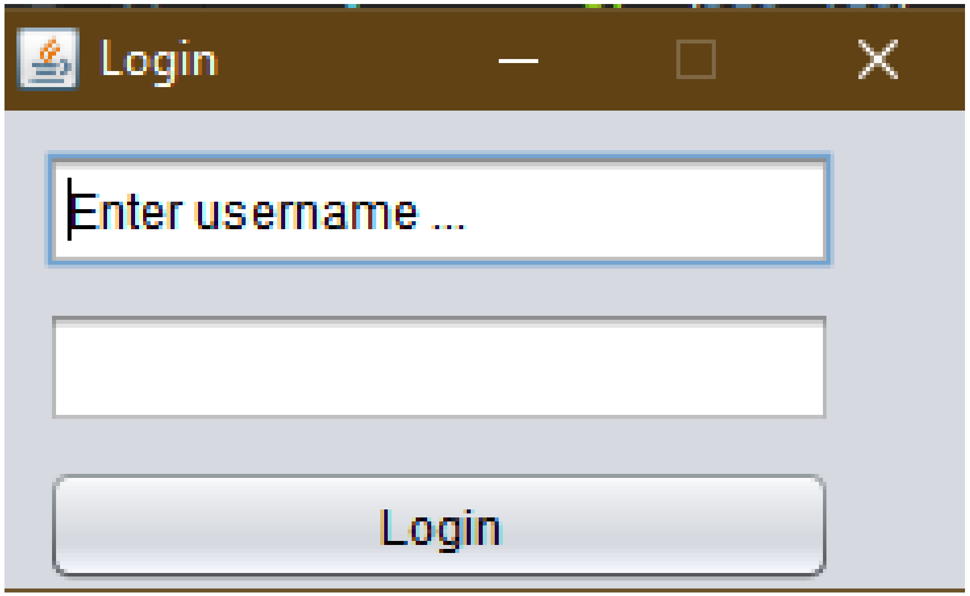
12.18 Introduction to Layout Managers

- Layout managers arrange GUI components in a container for presentation purposes
- Can use for basic layout capabilities
- Enable you to concentrate on the basic look-and-feel—the layout manager handles the layout details.
- Layout managers implement interface LayoutManager (in package `java.awt`).
- `Container`'s `setLayout` method takes an object that implements the `LayoutManager` interface as an argument.

12.18 Introduction to Layout Managers (cont.)

- There are two ways for you to arrange components in a GUI:
  - Absolute positioning
    - Greatest level of control.
    - Set `Container`'s layout to `null`.
    - Specify the absolute position of each GUI component with respect to the upper-left corner of the `Container` by using `Component` methods `setSize` and `setLocation` or `setBounds`.
    - Must specify each GUI component's size.

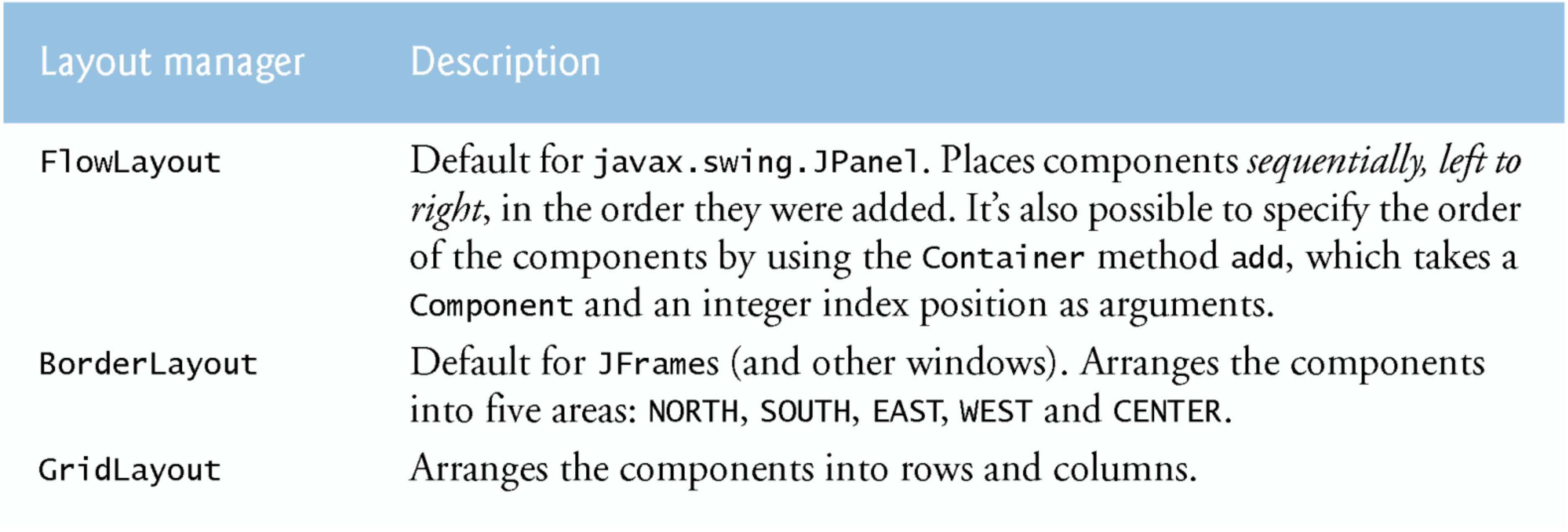
LoginFrame-First Attempt (Absolute Locations)



12.18 Introduction to Layout Managers (cont.)

- Layout managers
  - Simpler and faster than absolute positioning.
  - Makes your GUIs more resizable.
  - Lose some control over the size and the precise positioning of each component.

12.18 Introduction to Layout Managers (cont.)



Layout manager	Description
`FlowLayout`	Default for `javax.swing.JPanel`. Places components *sequentially, left to right*, in the order they were added. It's also possible to specify the order of the components by using the `Container` method `add`, which takes a `Component` and an integer index position as arguments.
`BorderLayout`	Default for `JFrames` (and other windows). Arranges the components into five areas: NORTH, SOUTH, EAST, WEST and CENTER.
`GridLayout`	Arranges the components into rows and columns.

Fig. 12.38 | Layout managers.

12.18.1 `FlowLayout`

- `FlowLayout` is the *simplest* layout manager.
- GUI components placed from left to right in the order in which they are added to the container.
- When the edge of the container is reached, components continue to display on the next line.
- `FlowLayout` allows GUI components to be *left aligned*, *centered* (the default) and *right aligned*.

12.18.1 `FlowLayout` (cont.)

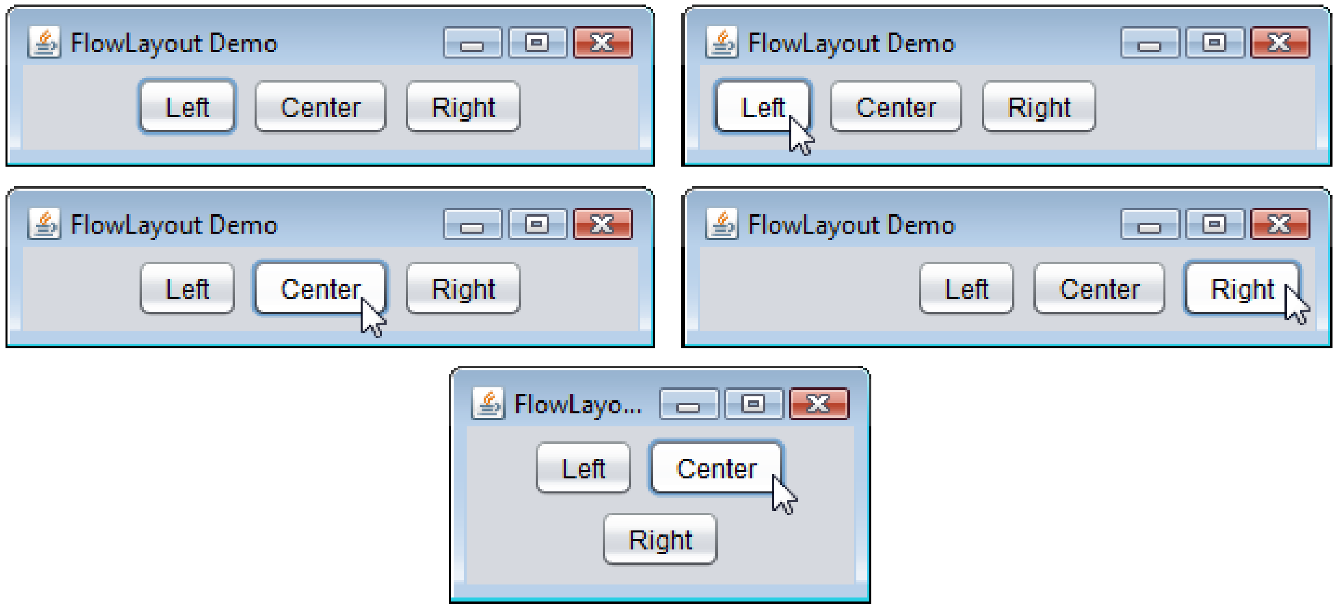


Fig. 12.40 | Testing `FlowLayoutFrame`. (Part 2 of 2.)

12.18.1 `FlowLayout` (cont.)

- `FlowLayout` method setAlignment changes the alignment for the `FlowLayout`.
  - FlowLayout.LEFT
  - FlowLayout.CENTER
  - FlowLayout.RIGHT

12.18.2 BorderLayout

- `BorderLayout`
  - the default layout manager for a `Jframe`
  - arranges components into five regions: `NORTH`, `SOUTH`, `EAST`, `WEST` and `CENTER`.
  - `NORTH` corresponds to the top of the container.
- `BorderLayout` limits a `Container` to at most five components—one in each region.
  - The component placed in each region can be a container to which other components are attached.

12.18.2 `BorderLayout` (cont.)

- `BorderLayout` constructor arguments specify the number of pixels between components that are arranged horizontally (horizontal gap space) and between components that are arranged vertically (vertical gap space), respectively.
  - The default is one pixel of gap space horizontally and vertically.

12.18.2 BorderLayout (cont.)

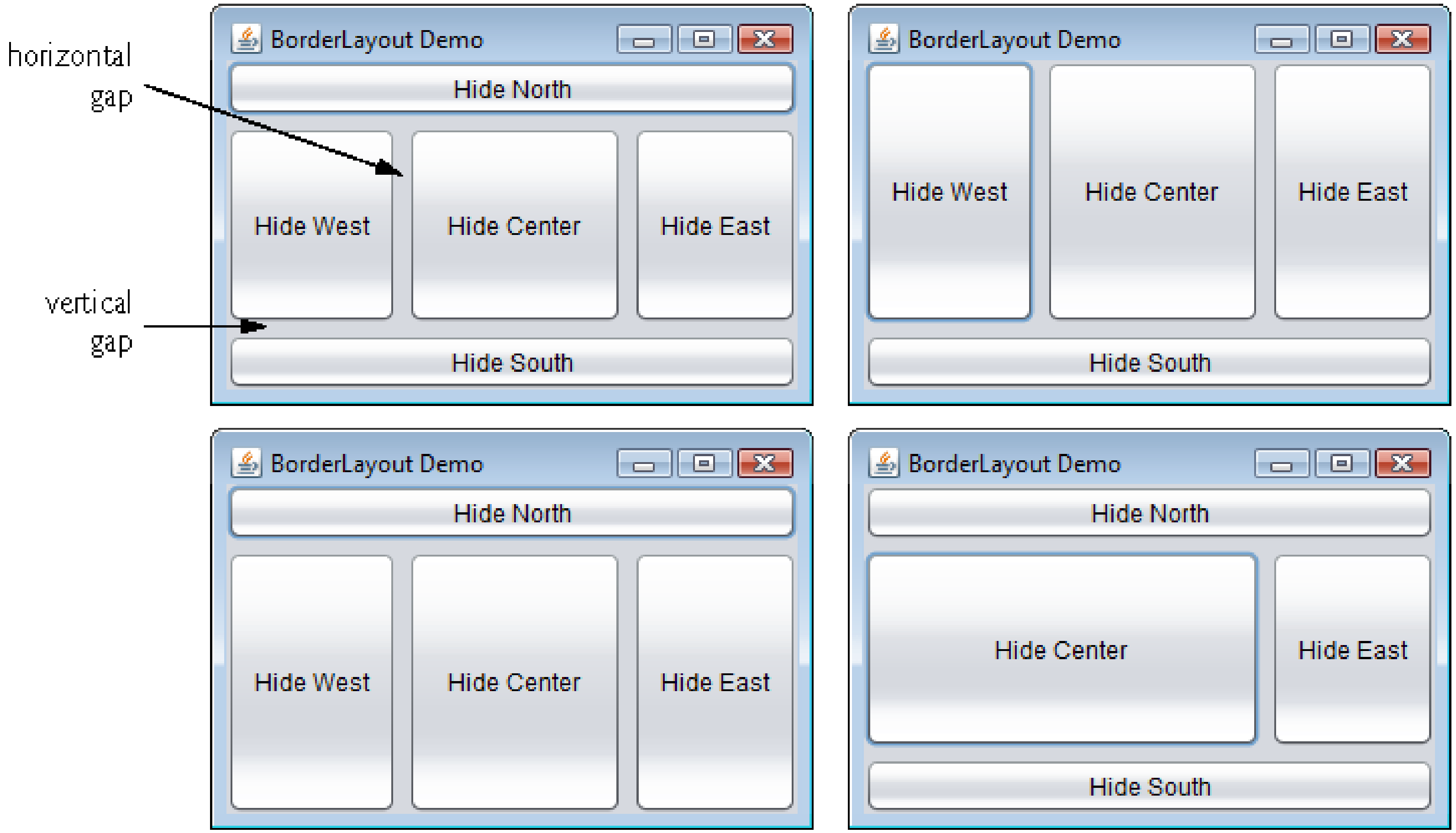
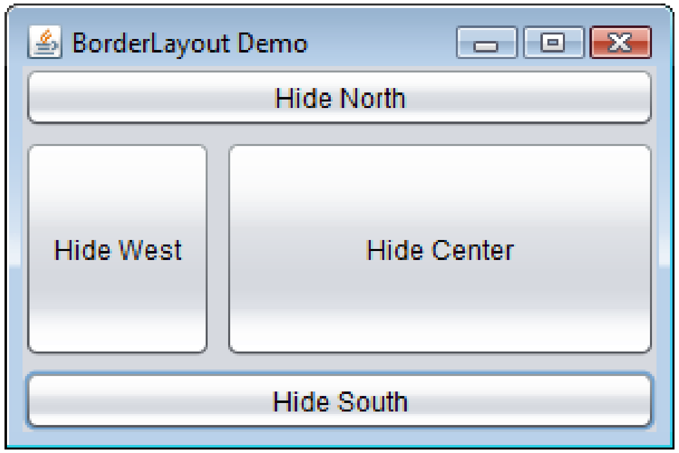


Fig. 12.42 | Testing `BorderLayoutFrame`. (Part 2 of 3.)

12.18.2 BorderLayout (cont.)



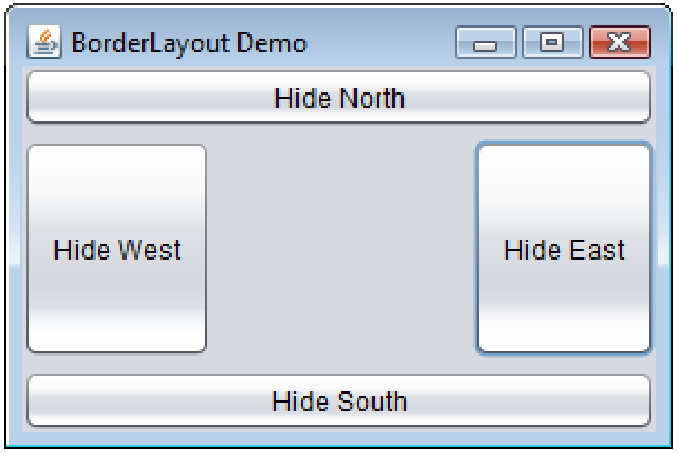
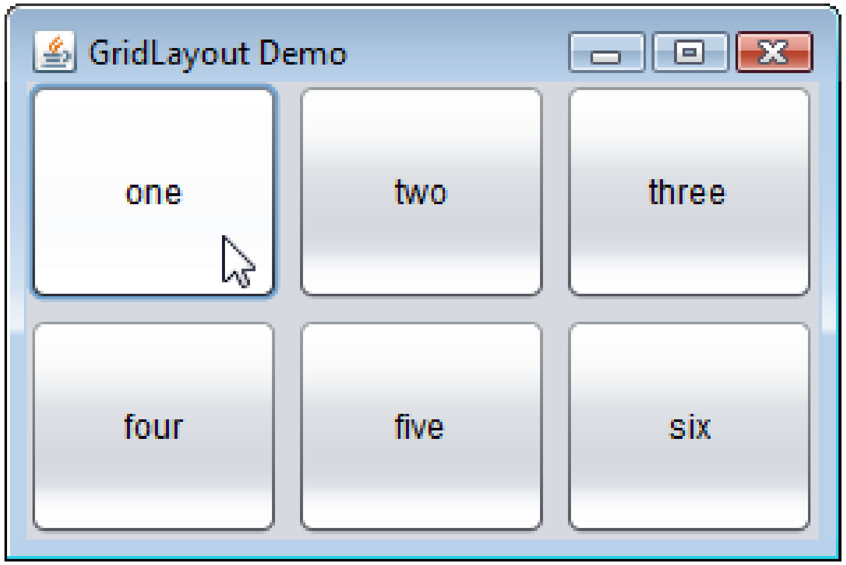


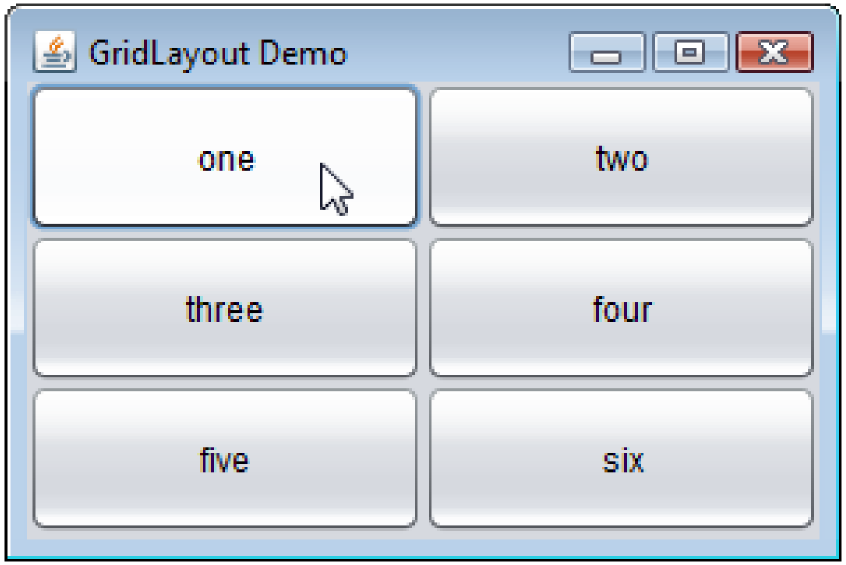
Fig. 12.42 | Testing **BorderLayoutFrame**. (Part 3 of 3.)

12.18.3 `GridLayout`

- `GridLayout` divides the container into a *grid* of *rows* and *columns*.
  - Implements interface `LayoutManager`.
  - Every `Component` has the same width and height.
  - Components are added starting at the top-left cell of the grid and proceeding left to right until the row is full. Then the process continues left to right on the next row of the grid, and so on.

12.18.3 GridLayout



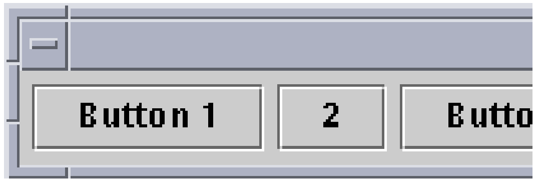


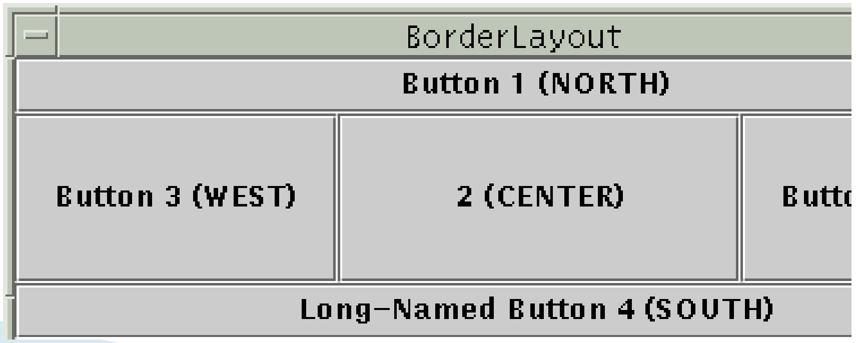
Preferred sizes

- Swing component objects each have a certain size they would "like" to be: Just large enough to fit their contents (text, icons, etc.).
  - This is called the *preferred size* of the component.

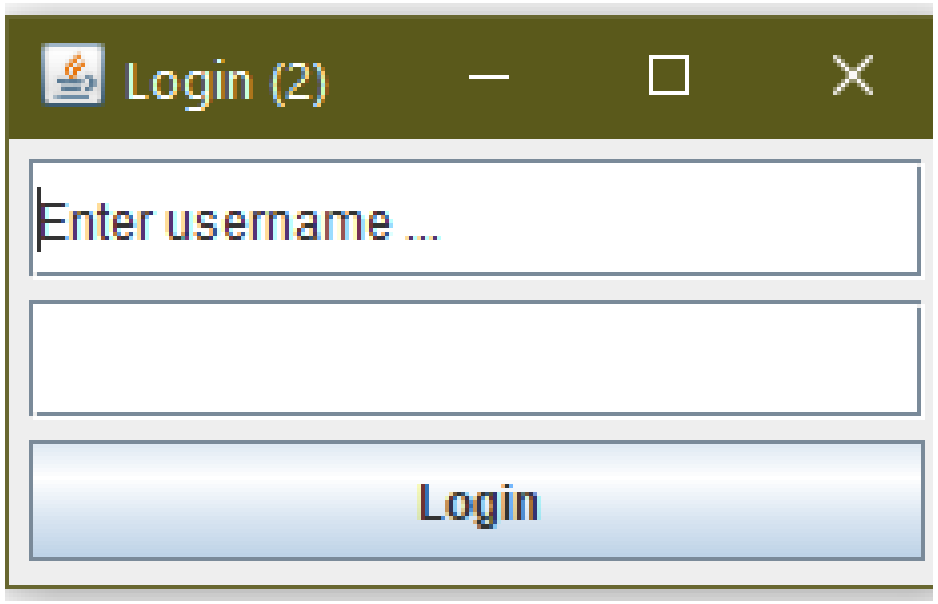
  - Some types of layout managers (e.g. FlowLayout) choose to size the components inside them to the preferred size.
  - Others (e.g. BorderLayout, GridLayout) disregard the preferred size and use some other scheme to size the components.

Buttons at preferred size: *Not preferred size:*





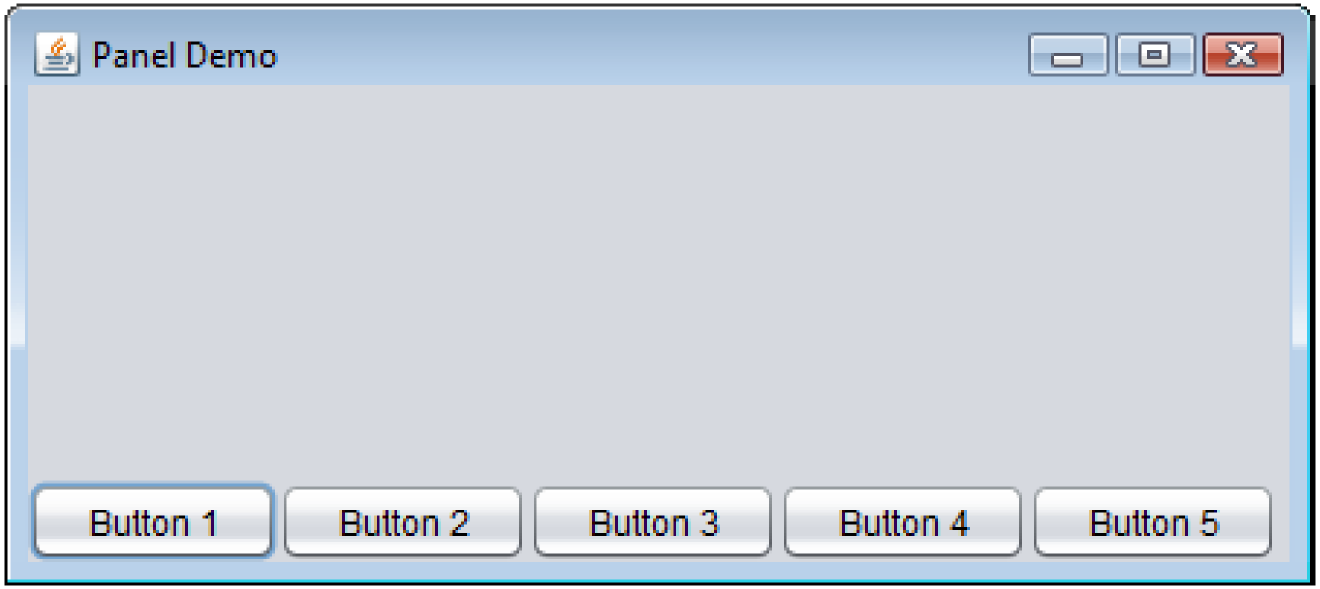
LoginFrame-WithLayout



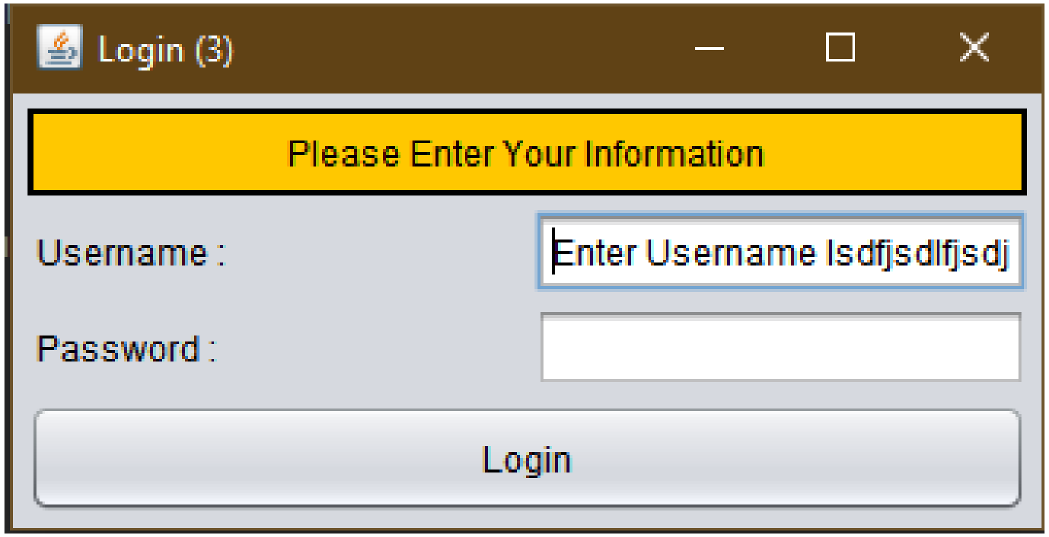
12.19 Using Panels to Manage More Complex Layouts

- Complex GUIs often require that each component be placed in an exact location.
  - Often consist of multiple panels, with each panel's components arranged in a specific layout.
- Class `JPanel` extends `JComponent` and `JComponent` extends class `Container`, so every `JPanel` is a `Container`.
- Every `JPanel` may have components, including other panels, attached to it with `Container` method `add`.
- `JPanel` can be used to create a more complex layout in which several components are in a specific area of another container.

12.19 Using Panels to Manage More Complex Layouts



LoginFrame-MultiLayout



Other Layouts

- BoxLayout.
- CardLayout.
- GridBagLayout.
- GroupLayout.
- SpringLayout.