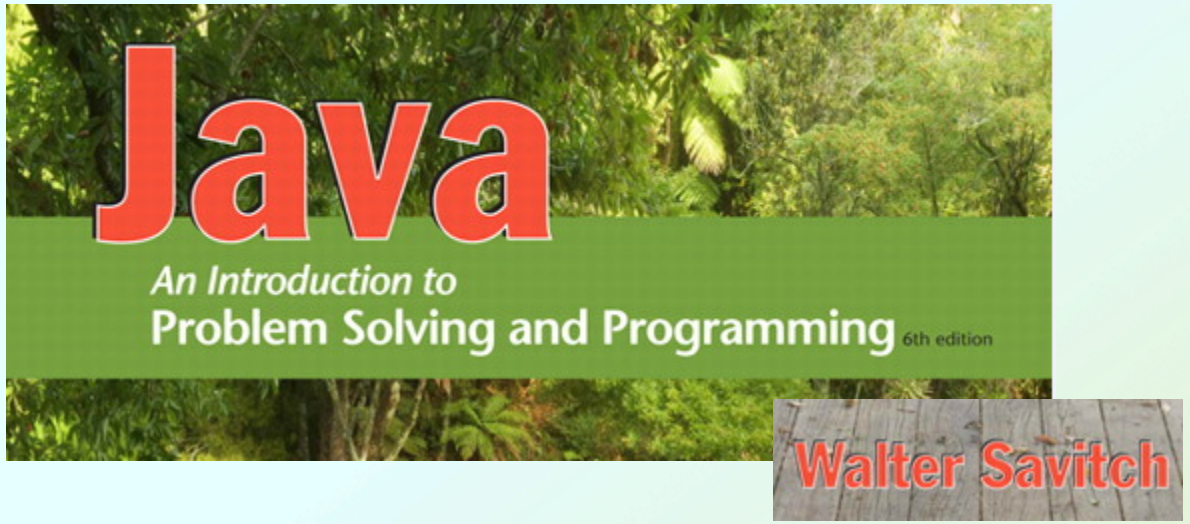


More Swing

Chapter 15

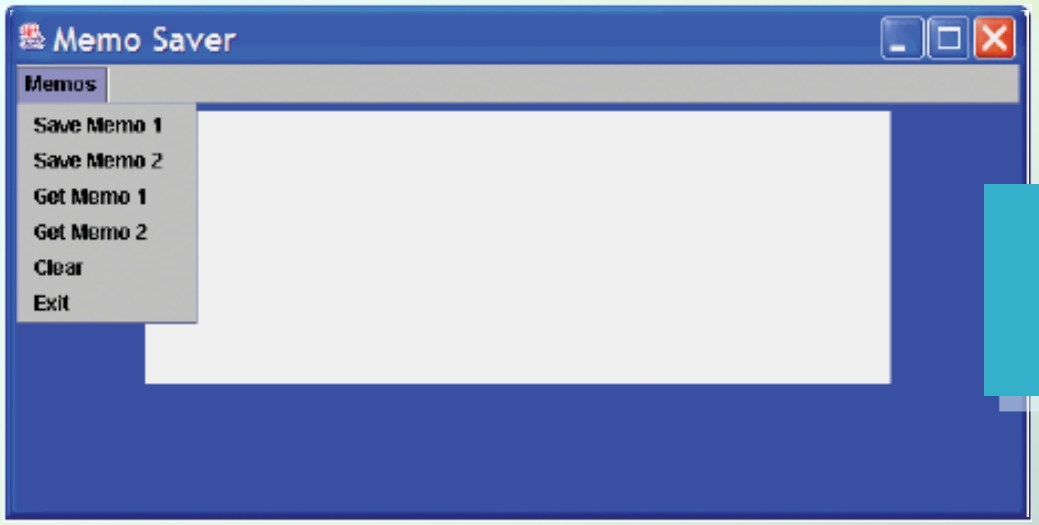
*JAVA: An Introduction to Problem Solving & Programming, 6*th Ed. By Walter Savitch
ISBN 0132162709

Objectives

- Add menus, icons, borders, and scroll bars to your GUIs
- Use the `BoxLayout` manager and the `Box` class
- Use inner classes and describe their advantages
- Use the `WindowListener` interface
- Create GUIs whose components change from visible to invisible and vice versa

A GUI with a Menu

- Menu items behave like buttons
- Note example program, listing 15.1
  `class MemoGUI`



Sample screen output

Menu Bars, Menus, and Menu Items

- A menu item is a **`JMenuItem`** object
- A menu is a **`JMenu`** object
  - Use the .add method to add a menu item to a menu
- A menu bar is a **`JMenuBar`** object and a container for menus
  - Add menu bar to a frame with **`setJMenuBar`**

Nested Menus

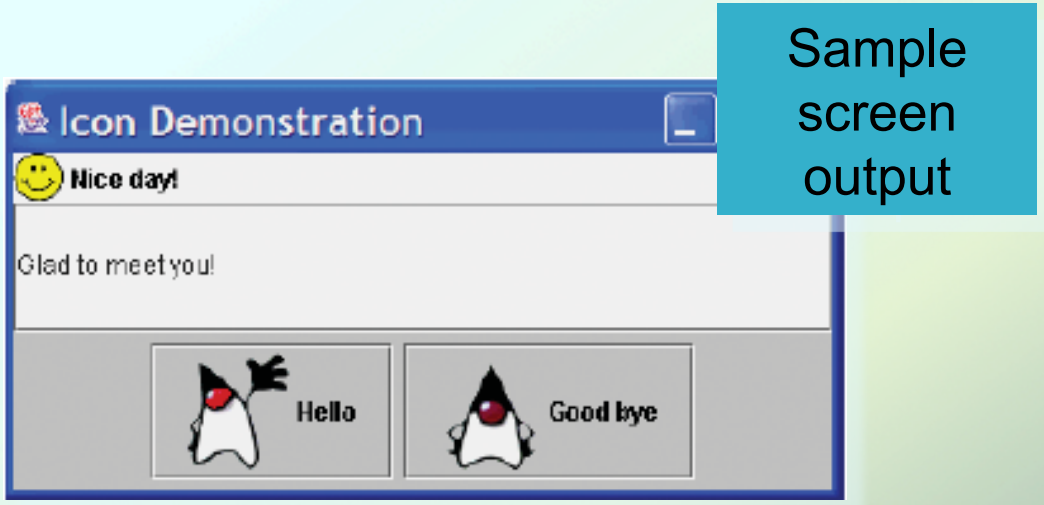
- Class **`JMenu`** is a descendant of the **`JMenuItem`** class
- Every menu is also a menu item object
- Thus a menu can be a menu *item* in another menu

Adding Icons

- An icon is
  - A picture
  - An object of **`ImageIcon`**
- An icon can be placed in
  - A label
  - A button
  - Both can also have text with the icon

Using Icons

- View sample program, listing 15.2
  `class IconDemo`



Swing Methods

- Figure 15.1a some methods in the classes **`JButton`** and **`JLabel`**

`public JButton()`
`public JLabel()`
Creates a button or label with no text or icon on it. (Typically, you will use `setText` or `setIcon` later to set these items.)

`public JButton(String text)`
`public JLabel(String text)`
Creates a button or label containing the given text.

`public JButton(ImageIcon picture)`
`public JLabel(ImageIcon picture)`
Creates a button or label containing the given icon.

`public JButton(String text, ImageIcon picture)`
`public JLabel(String text, ImageIcon picture, int horizontalAlignment)`
Creates a button or label containing both the given text and an icon whose horizontal alignment is specified by one of the constants `SwingConstants.LEFT`, `SwingConstants.CENTER`, `SwingConstants.RIGHT`, `SwingConstants.LEADING`, or `SwingConstants.TRAILING`.

`public void setText(String text)` Makes `text` the only text on the button or label.

Swing Methods

- Figure 15.1b some methods in the classes **`JButton`** and **`JLabel`**

```
public void setIcon(ImageIcon picture)
```
Makes `picture` the only icon on the button or label.

```
public void setMargin(Insets margin) //JButton only
```
Sets the size of the margin around the text and icon on the button. The following special case will work for most simple situations, where the `int` values give the number of pixels from the edge of the button to the text or icon:

```
public void setMargin(new Insets(int top, int left,
                                 int bottom, int right))
```

```
public void setPreferredSize(Dimension preferredSize)
```
Sets the preferred size of the button or label. The layout manager is not required to use this preferred size, as it is only a suggestion. The following special case will work for most simple situations, where the `int` values give the width and height in pixels:

```
public void setPreferredSize(new Dimension(int width, int height))
```

```
public void setMaximumSize(Dimension maximumSize)
```
Sets the maximum size of the button or label. The layout manager is not required to respect this maximum size, as it is only a suggestion. The following special case will work for most simple situations, where the `int` values give the width and height in pixels:

```
public void setMaximumSize(new Dimension(int width, int height))
```

Swing Methods

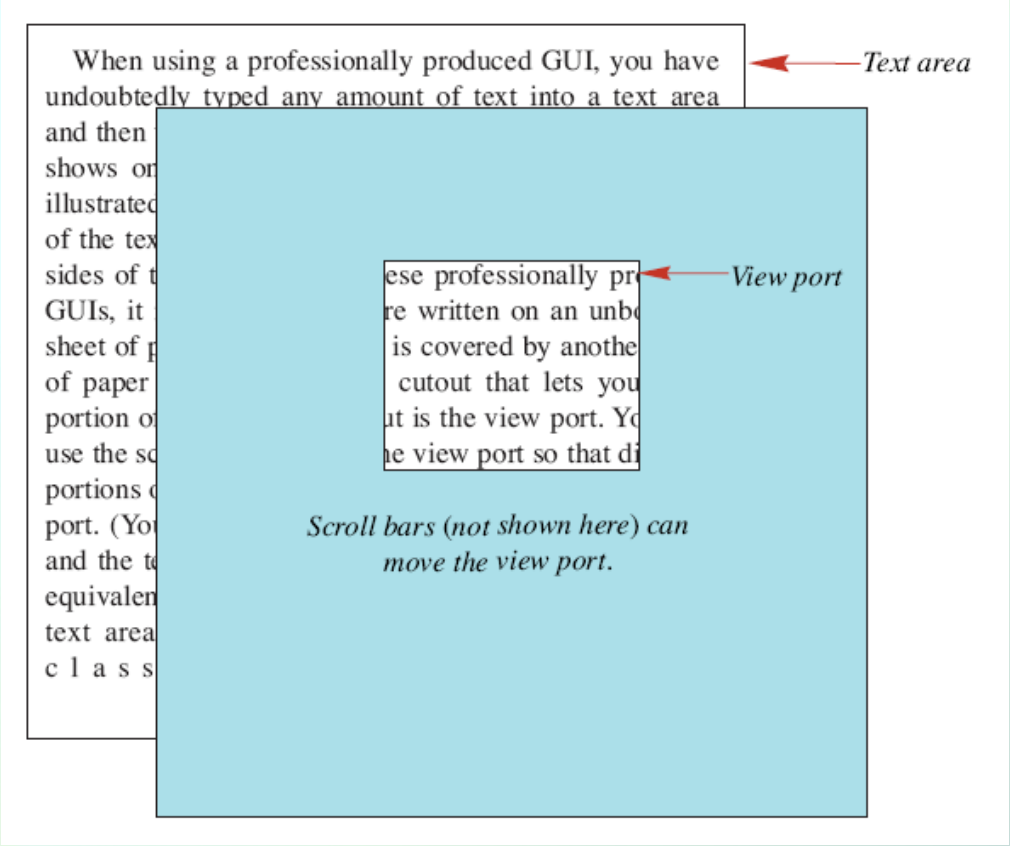
- Figure 15.1c some methods in the classes **`JButton`** and **`JLabel`**

`public void setMinimumSize(Dimension minimumSize)`
Sets the minimum size of the button or label. The layout manager is not required to respect this minimum size, as it is only a suggestion. The following special case will work for most simple situations, where the `int` values give the width and height in pixels:
`public void setMinimumSize(new Dimension(int width, int height))`

`public void setVerticalTextPosition(int textPosition)` Sets the vertical position of the text relative to the icon. This text position should be one of the constants `SwingConstants.TOP`, `SwingConstants.CENTER` (the default position), or `SwingConstants.BOTTOM`.
`public void setHorizontalTextPosition(int textPosition)` Sets the horizontal position of the text relative to the icon. This text position should be one of the constants `SwingConstants.RIGHT`, `SwingConstants.LEFT`, `SwingConstants.CENTER`, `SwingConstants.LEADING`, or `SwingConstants.TRAILING`.

Text Area Features

- Figure 15.2 View port for a text area



The `JScrollPane` Class for Scroll Bars

- A view port shows a portion of the class
- A scroll pane is a viewport with scrollbars
- Vertical and/or horizontal scrollbars may be specified

Example Program

- View code, listing 15.3

  `class ScrollBarDemo`

Sample screen output

JAVA: An Introduction to Problem Solving & Programming, 6th Ed. By Walter Savitch
ISBN 0132162709

JScrollPane methods

- Figure 15.3 Some methods and constants in **`class JScrolPane`**

`public JScrollPane(Component objectToBeScrolled)`
Creates a new scroll pane for the given object `objectToBeScrolled`. Note that this object need not be a text area, although that is the only type of argument we consider in this book.

`public void setHorizontalScrollBarPolicy(int policy)`
Sets the policy for showing the horizontal scroll bar. The argument `policy` should be a constant of the form

`JScrollPane.HORIZONTAL_SCROLLBAR_`*When*

where *When* is either `ALWAYS`, `NEVER`, or `AS_NEEDED`.
These constants are defined in the class `JScrollPane`. You should think of them as policies, not as the `int` values that they are. You need not be aware of their data type. The phrase "`AS_NEEDED`" means that the scroll bar is shown only when it is needed. This concept is explained more fully in the text. The meanings of the other policy constants are obvious from their names.

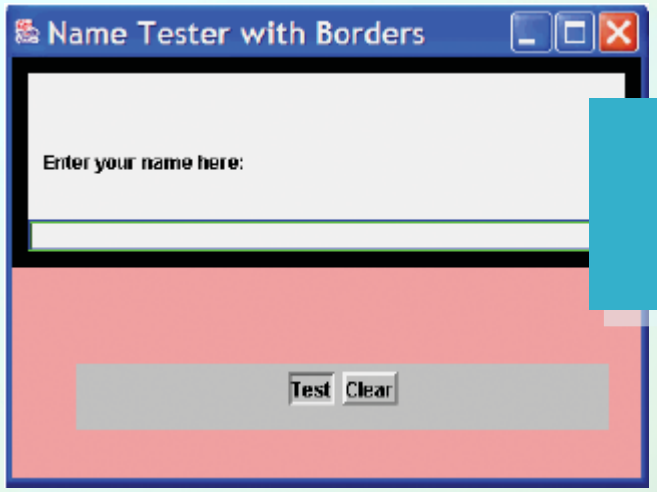
`public void setVerticalScrollBarPolicy(int policy)`
Sets the policy for showing the vertical scroll bar. The argument `policy` should be a constant of the form

`JScrollPane.VERTICAL_SCROLLBAR_`*When*

where *When* is either `ALWAYS`, `NEVER`, or `AS_NEEDED`. The same comment about the constants for a horizontal scroll bar are true here also.

Adding Borders

- View sample program, listing 15.4
  `class BorderDemo`



Sample screen output

Border Class Constructors

- Figure 15.4a, some constructors of various border classes

`public BevelBorder(int bevel)` Creates a bevel border that is either raised or lowered according to whether the argument `bevel` is the constant `BevelBorder.RAISED` or `BevelBorder.LOWERED`, respectively.
`public EmptyBorder(int top, int left, int bottom, int right)` Creates an empty border—which is essentially space around the component—whose size is specified in pixels by the given arguments.
`public EtchedBorder(int etch, Color highlightColor, Color shadowColor)` Creates an etched border, which is similar to a bevel border, except that you cannot set its size, and it has a highlight and shadow in specified colors. The argument `etch` should be one of the constants `EtchedBorder.RAISED` or `EtchedBorder.LOWERED`.
`public EtchedBorder(Color highlightColor, Color shadowColor)` Creates a new lowered etched border with the specified colors for its highlight and shadow.
`public LineBorder(Color theColor, int thickness)` Creates a new line border, which is a band around the component, having the specified color and thickness, given in pixels.

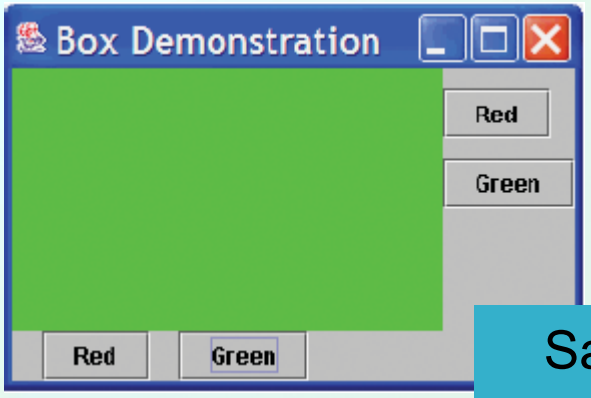
Border Class Constructors

- Figure 15.4b, some constructors of various border classes

`public LineBorder(Color color, int thickness, boolean roundedCorners)` Creates a line border having the specified color, thickness, and corner shape. The thickness is given in pixels. If `roundedCorners` is true, rounded corners are produced. If `roundedCorners` is false or omitted, right-angle corners are produced.
`public MatteBorder(int top, int left, int bottom, int right, Color theColor)` Creates a new `MatteBorder` object. A `MatteBorder` is similar to a `LineBorder`, but you can specify the size—in pixels—of each of the four sides.
`public MatteBorder(int top, int left, int bottom, int right, ImageIcon theIcon)` Creates a new matte border—which is like a wallpaper pattern—that is tiled with copies of `theIcon`. A sample of a program using this constructor is in the file `BorderDemoWithIcon.java` available on the Web.

`BoxLayout` Manager Class

- View demo program, listing 15.5
  `class BoxLayoutDemo`



Sample screen output

Struts and Glue

- A strut is an invisible component
  - A fixed length used to separate visible components
  - Note use of strut in listing 15.5
  - Struts can be either horizontal or vertical
- Glue is also an invisible component
  - Not used to stick things together
  - Rather, they are not rigid, more like "goo"
  - Glue components also separate visible items

Setting Spacing Between Components

- Methods used to set horizontal and vertical spacing

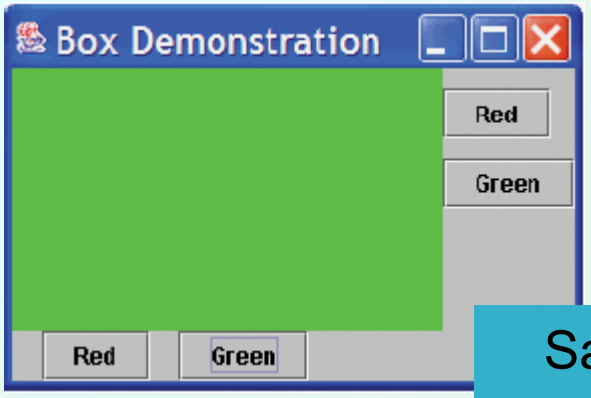
```
/**
 Sets the horizontal gap between components
 to hgap, which is expressed in pixels.
*/
public void setHgap(int hgap)

/**
 Sets the vertical gap between components
 to vgap, which is expressed in pixels.
*/
public void setVgap(int vgap)
```

Except for classes **`Box`** and **`BoxLayout`**

The Box Container Class

- View demo program, listing 15.6
  `class BoxClassDemo`



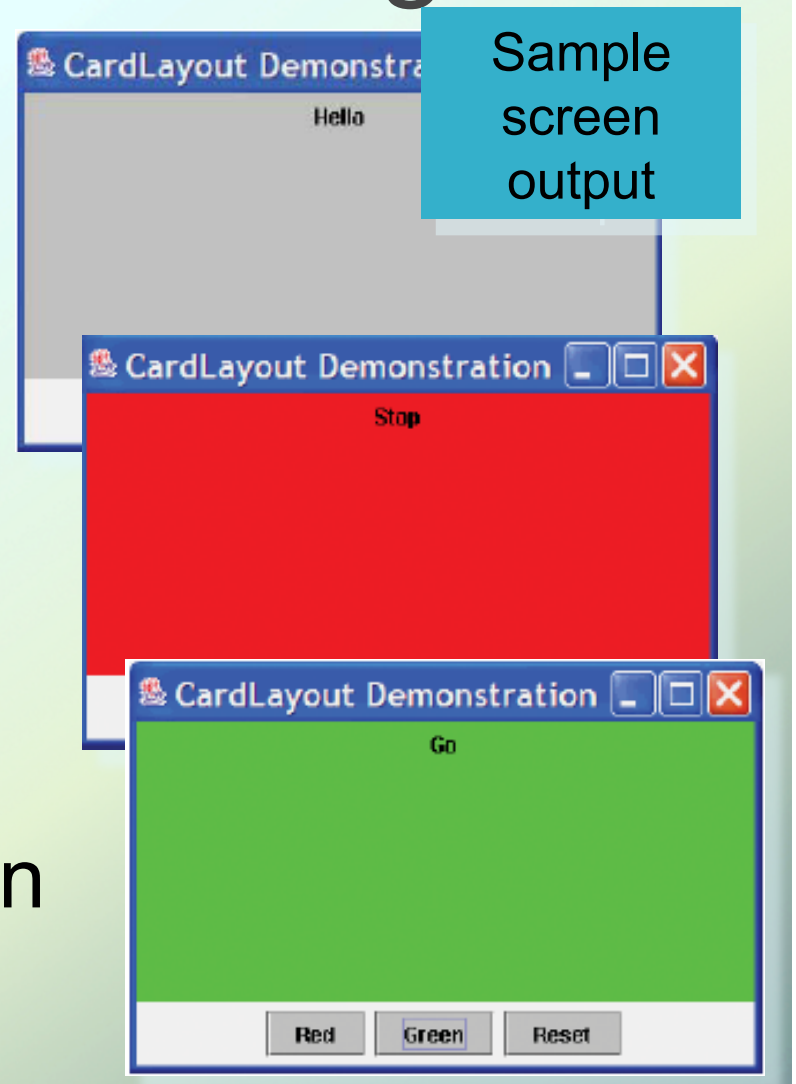
Sample screen output

The `CardLayout` Manager

- View demo program, listing 15.7
  `class CardLayoutDemo`
- The `CardLayout` manager creates a set of views that can be cycled
- Do not use an anonymous argument when using the `CardLayout` manager

The `CardLayout` Manager

- Initially or after reset button clicked
- After clicking red button
- After clicking green button



CardLayout Methods

- Figure 15.5 Some methods in the **`CardLayout`** manager class

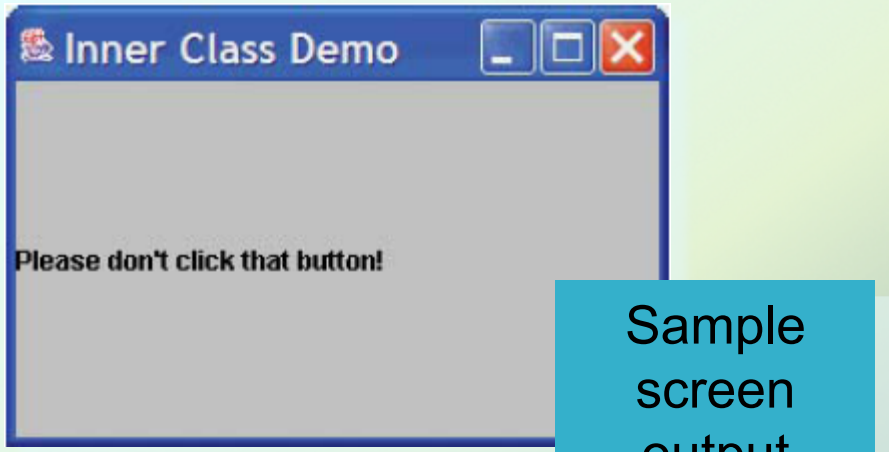
`public void first(Container theContainer)` Causes the first "card" in `theContainer` to be displayed.
`public void last(Container theContainer)` Causes the last "card" in `theContainer` to be displayed.
`public void next(Container theContainer)` Causes the next "card" in `theContainer` to be displayed. The next card is the one that was added after the currently displayed card was added. If the currently displayed card is the last card, the method displays the first card.
`public void previous(Container theContainer)` Causes the previous "card" in `theContainer` to be displayed. The previous card is the one that was added before the currently displayed card was added. If the currently displayed card is the first card, the method displays the last card.
`public void show(Container theContainer, String cardName)` Displays the "card" that was identified as `cardName` when it was added.

Inner Classes

- Classes defined within other classes
- Often used in Swing programs and classes
  - However not confined to that usage
- Advantages
  - Can be used to make outer class self contained
  - Methods of inner class have access to all elements of the outer class

Helping Classes

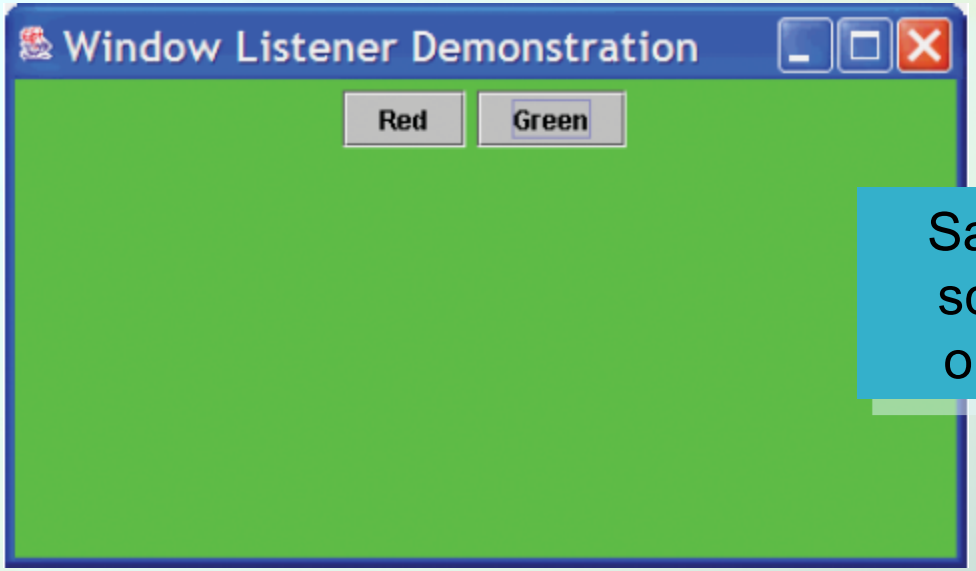
- View example, listing 15.8
  `class InnerClassDemo`



Sample screen output

The `WindowListener` Interface

- View demo, listing 15.9
  `class WindowListenerDemo`



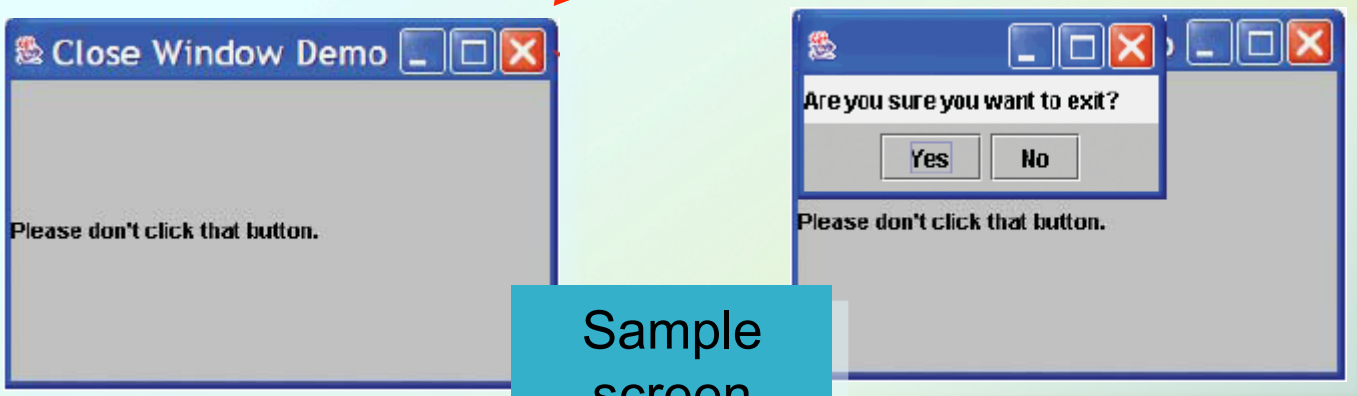
Sample screen output

Programming the Close-Window Button

- View example, listing 15.10
  `class CloseWindowDemo`
- By default a window's close-window button does not end the execution
- It is possible to program the behavior of a window's close window button

Programming the Close-Window Button

- Click the close-window button
  - The second window appears



Sample screen output

Interface `WindowConstants`

- Figure 15.6 Constants in the interface `WindowConstants`

The following constants are used with the method `setDefaultCloseOperation` to specify the default window-close operation:

`WindowConstants.DO_NOTHING_ON_CLOSE`
Tells the method to do nothing. The programmer should program any desired action in the definition of the `windowClosing` method of a registered `WindowListener` object.

`WindowConstants.HIDE_ON_CLOSE`
Tells the method to hide the frame after any registered `WindowListener` objects are invoked.

`WindowConstants.DISPOSE_ON_CLOSE`
Tells the method to hide and dispose of the frame after any registered `WindowListener` objects are invoked.

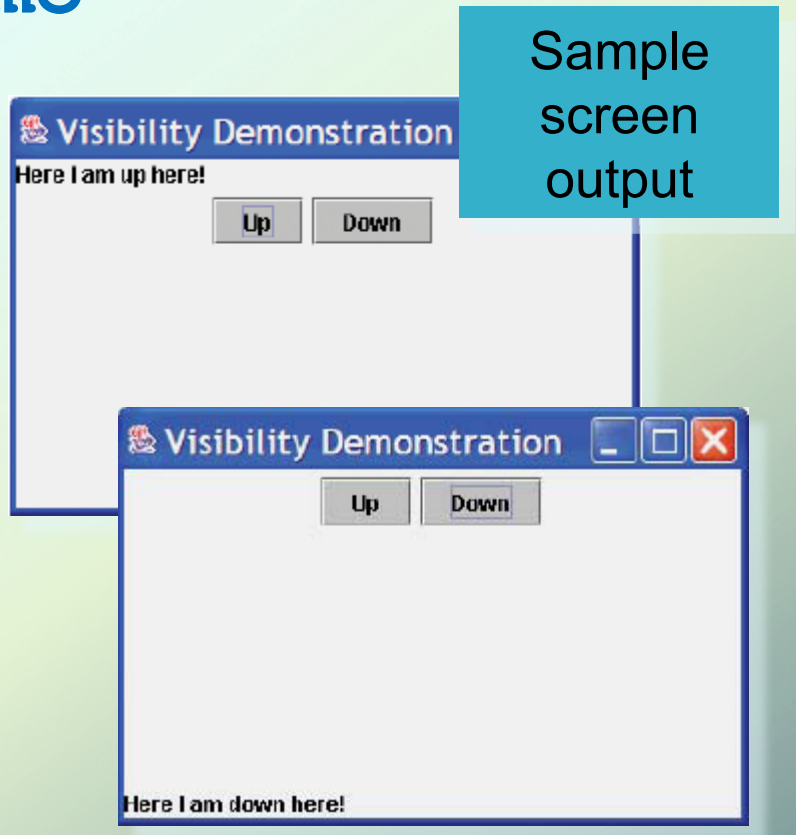
`WindowConstants.EXIT_ON_CLOSE`
Tells the method to exit the application by calling the method `System.exit`. Equivalent to the constant `JFrame.EXIT_ON_CLOSE`.

Components with Changing Visibility

- View sample program, listing 15.11
  `class VisibilityDemo`

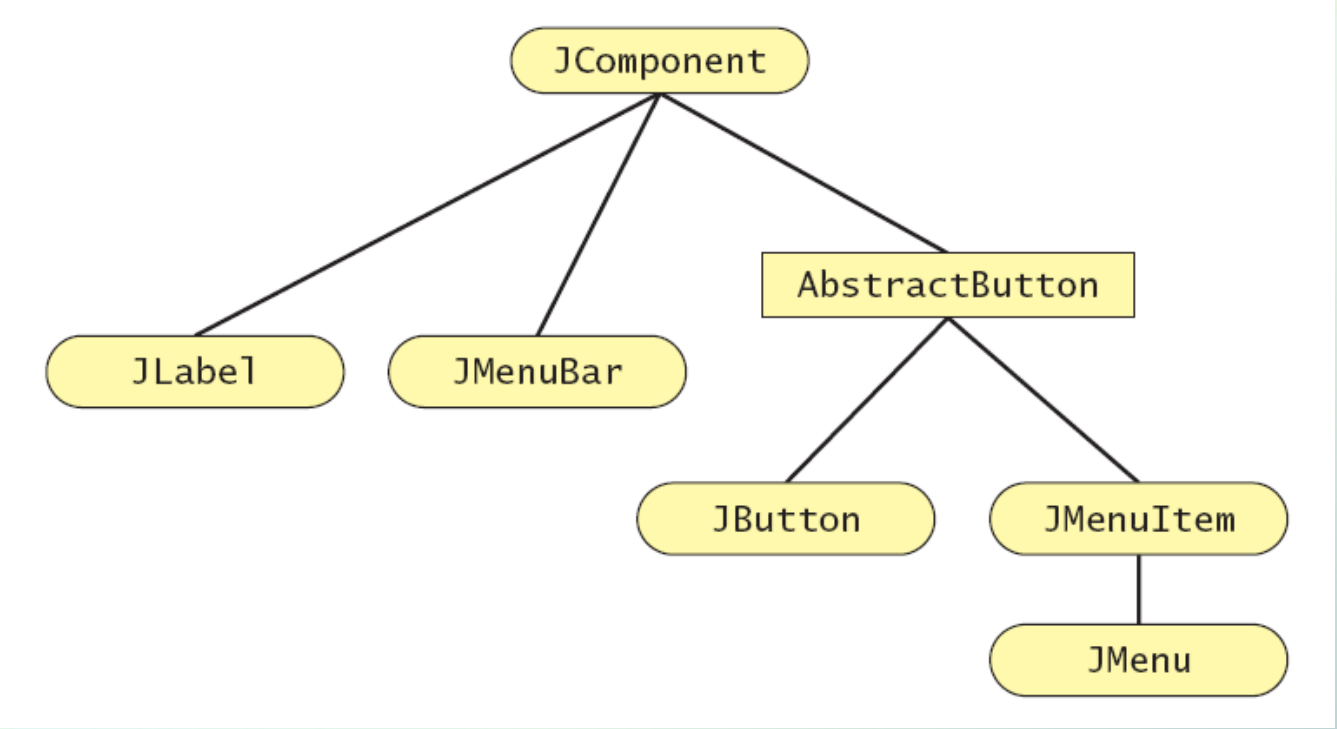
- After clicking the Up button

- After clicking the Down button



Buttons, Menus, Abstract Buttons

- Figure 15.7 A portion of the Swing Class Hierarchy



Buttons, Menus, Abstract Buttons

- Figure 15.8 Some methods inherited from **`AbstractButton`** by class **`JButton`** and **`JMenuItem`**

`public void addActionListener(ActionListener listener)`
`public void removeActionListener(ActionListener listener)`
Adds/removes the specified action listener from the button or menu item.

`public String getActionCommand()`
`public void setActionCommand(String actionCommand)`
Gets/sets the action command for the button or menu item.

`public Icon getIcon()`
`public void setIcon(Icon anIcon)`
Gets/sets the icon on the button or menu item.

`public String getText()`
`public void setText(String text)`
Gets/sets the text on the button or menu item.

Buttons, Menus, Abstract Buttons

- Figure 15.9 Some constructors of classes **`JButton`** and **`JMenuItem`**

`public JButton()`
`public JMenuItem()`
Creates a button (menu item) that has no text or icon.

`public JButton(Icon anIcon)`
`public JMenuItem(Icon anIcon)`
Creates a button (menu item) that has the given icon.

`public JButton(String text)`
`public JMenuItem(String text)`
Creates a button (menu item) that has the given text.

`public JButton(Icon anIcon, String text)`
`public JMenuItem(Icon anIcon, String text)`
Creates a button (menu item) that has the given icon and text.

Summary

- Classes **`JMenuItem`**, **`JMenu`**, and **`JMenuBar`** provide menu tools
- You can add icons to buttons, labels, menu items
- Menu bar can be added to a frame by using
  - **`setJmenu`**
  - **`add`**
- Buttons and menu items need registered action listeners

Summary

- Close-window button can be reprogrammed like any other button
  - May need to call `setDefaultCloseOperation`
- Changing visibility of a component requires call to method `validate`