Chapter 8 Creating a GUI with JFC/Swing

Introduction

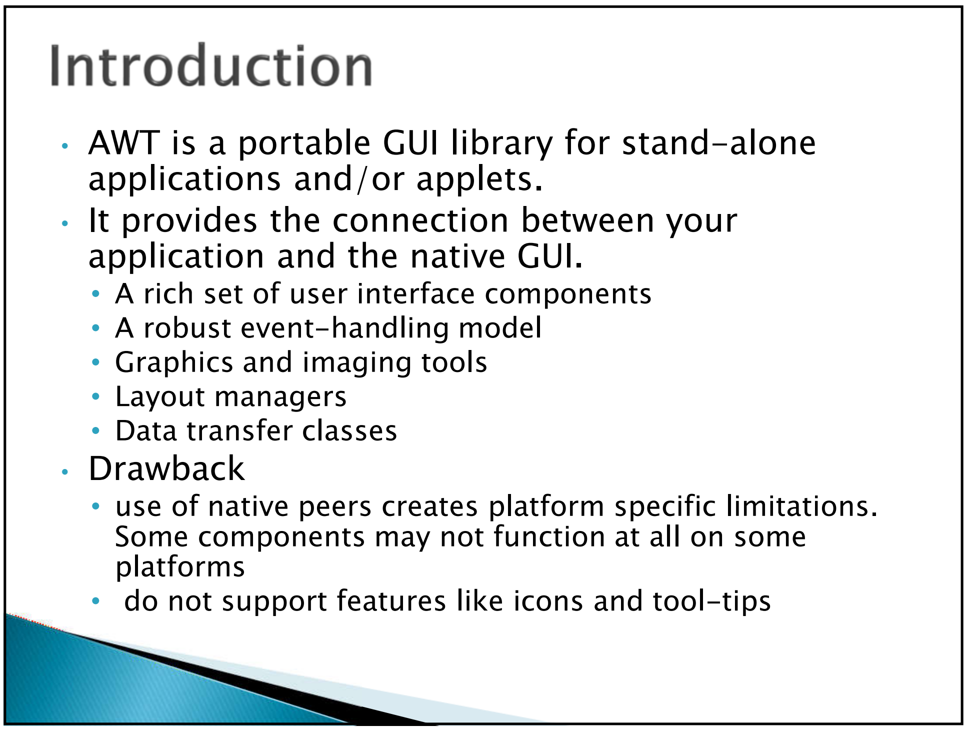
- The main task of a GUI program design is to create and arrange a number of components such as buttons, and register event process codes to handle the interaction events such as clicking on a button

Introduction

- There are two basic sets of components called the Abstract Window Toolkit (AWT) and Swing. Both of these groups of components are part of the Java Foundation Classes (JFC).

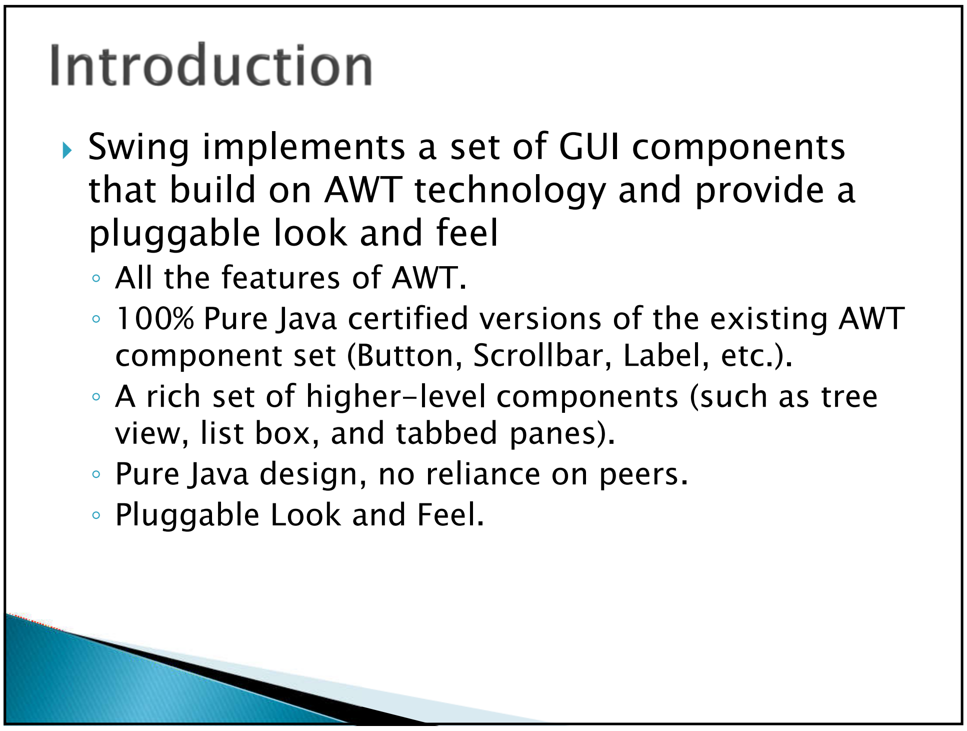
Introduction

- There are two basic sets of components called the Abstract Window Toolkit (AWT) and Swing. Both of these groups of components are part of the Java Foundation Classes (JFC).



Introduction

- AWT is a portable GUI library for stand-alone applications and/or applets.
- It provides the connection between your application and the native GUI.
  - A rich set of user interface components
  - A robust event-handling model
  - Graphics and imaging tools
  - Layout managers
  - Data transfer classes
- Drawback
  - use of native peers creates platform specific limitations. Some components may not function at all on some platforms
  - do not support features like icons and tool-tips

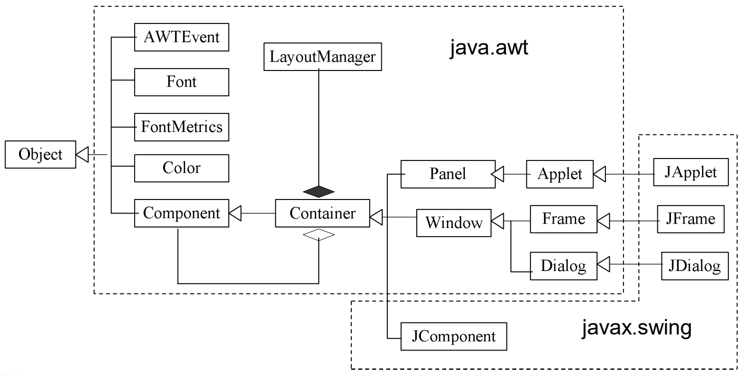


Introduction

- Swing implements a set of GUI components that build on AWT technology and provide a pluggable look and feel
  - All the features of AWT.
  - 100% Pure Java certified versions of the existing AWT component set (Button, Scrollbar, Label, etc.).
  - A rich set of higher-level components (such as tree view, list box, and tabbed panes).
  - Pure Java design, no reliance on peers.
  - Pluggable Look and Feel.

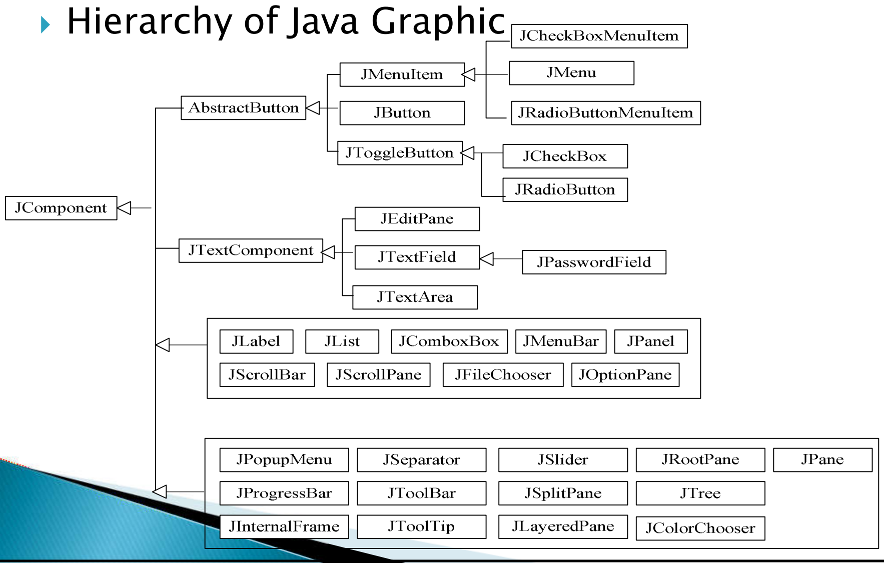
Introduction

- Hierarchy of Java Graphic



Introduction

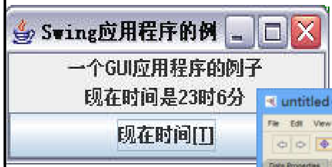
- Hierarchy of Java Graphic

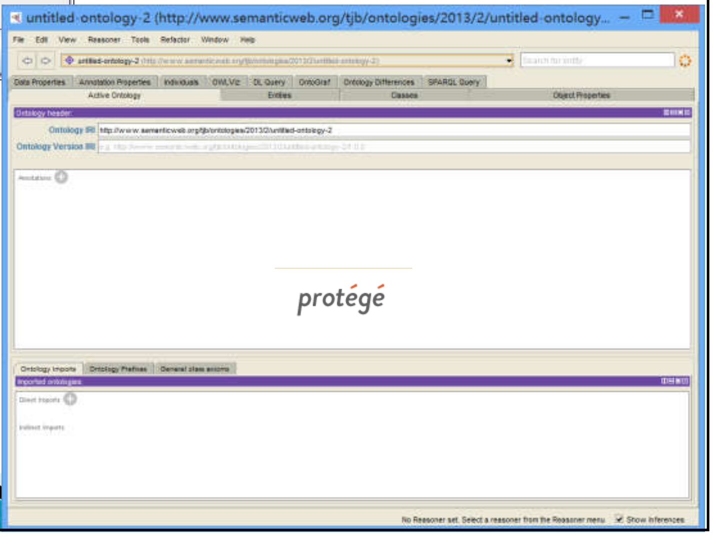


Using Swing APIs and Layout managers

- The javax.swing.JFrame is a built-in class to represent windows with title, border, optional menu bar and user-specified components.
- It can be moved, resized, iconified and closed.

AWT/Swing Applications

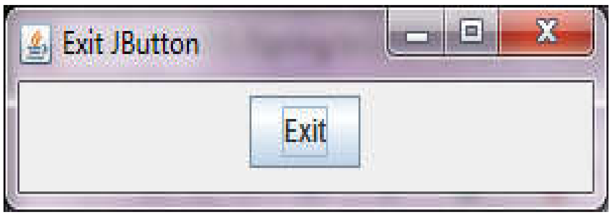

Swing应用程序的例
一个GUI应用程序的例子
现在时间是23时6分

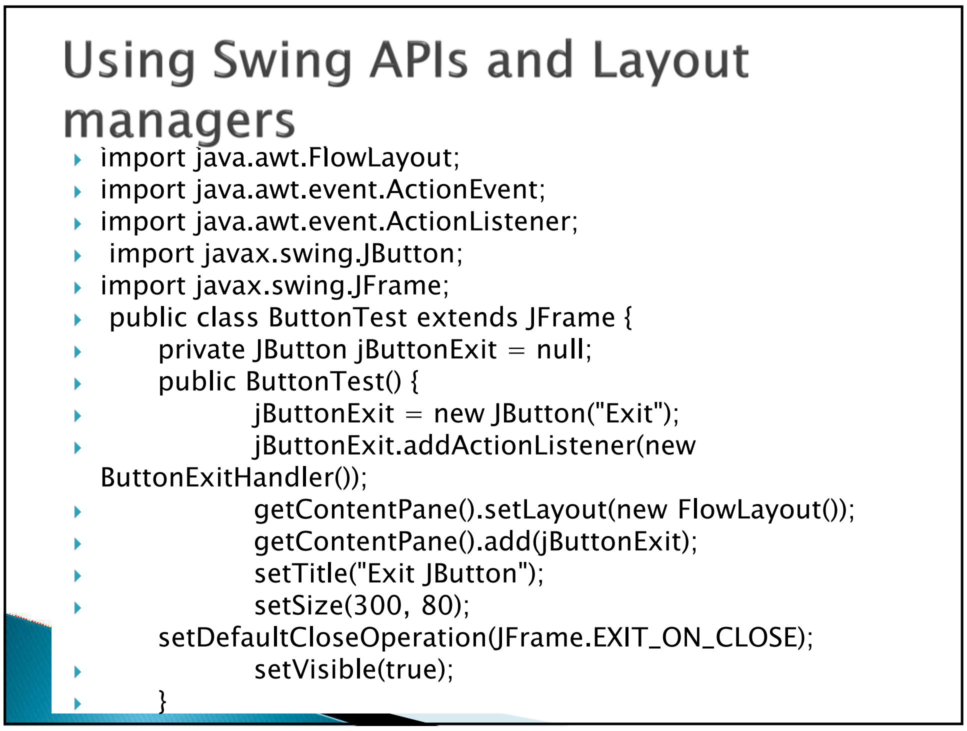

untitled-ontology-2 (http://www.semanticweb.org/tjb/ontologies/2013/2/untitled-ontology...

Using Swing APIs and Layout managers

- You generally create a GUI program following the steps below:
  - Step 1: Construct a top-level container which is usually a subclass of the JFrame;
  - Step 2: Add components to the contentPane of the top-level container and arrange them in a particular layout;
  - Step 3: Design event handling classes as appropriate to your application and register their instances to corresponding components. This step is optional;
  - Step 4: Set the title of the top-level container and set the size of the container;
  - Step 5: Set the default close operation. When the user clicks the close button, your program will usually terminate.
  - Step 6: Make the container visible.

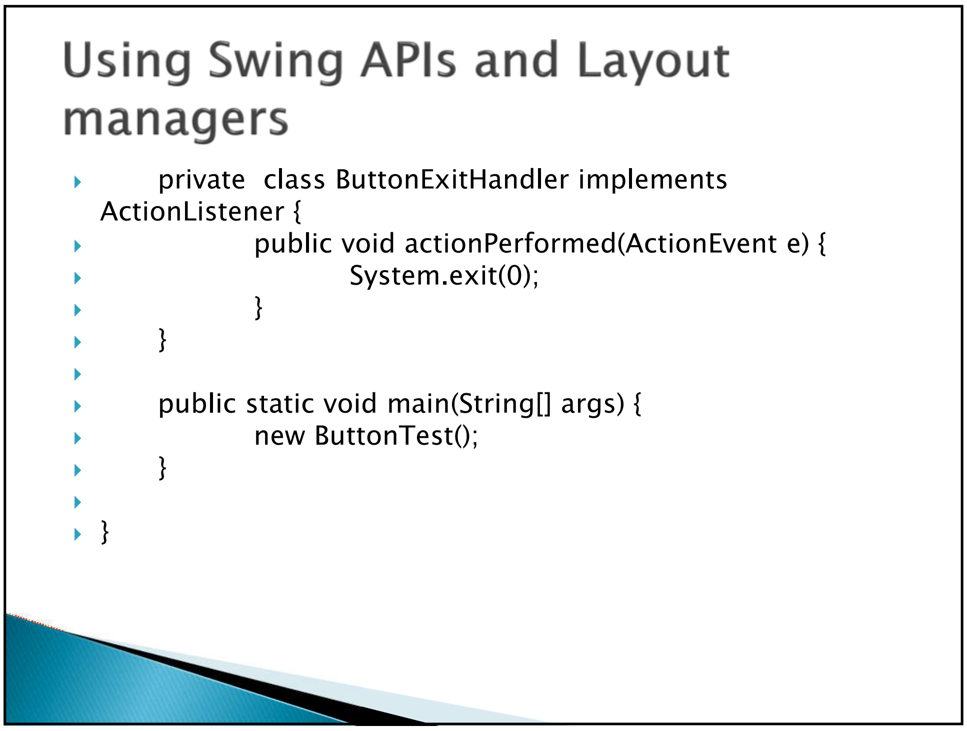
Using Swing APIs and Layout managers





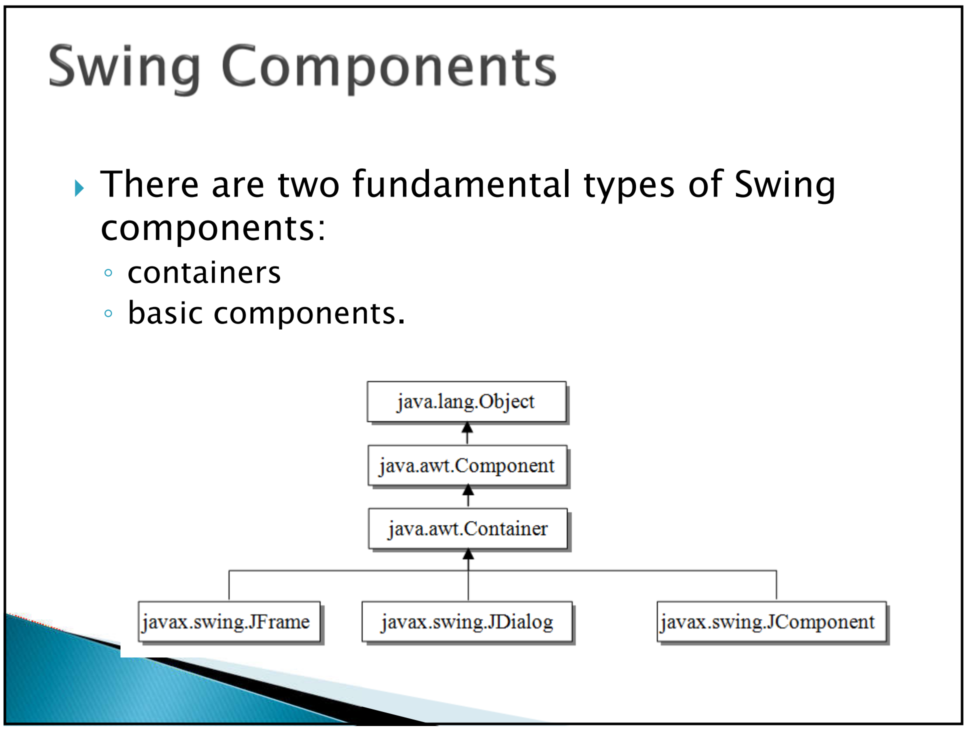
Using Swing APIs and Layout managers

```
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
 import javax.swing.JButton;
import javax.swing.JFrame;
 public class ButtonTest extends JFrame {
      private JButton jButtonExit = null;
      public ButtonTest() {
                 jButtonExit = new JButton("Exit");
                 jButtonExit.addActionListener(new
ButtonExitHandler());
                 getContentPane().setLayout(new FlowLayout());
                 getContentPane().add(jButtonExit);
                 setTitle("Exit JButton");
                 setSize(300, 80);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                 setVisible(true);
      }
```



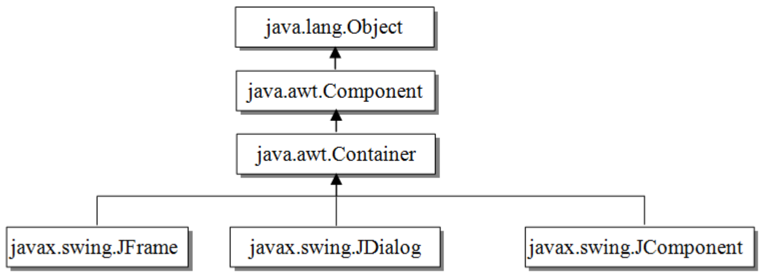
Using Swing APIs and Layout managers

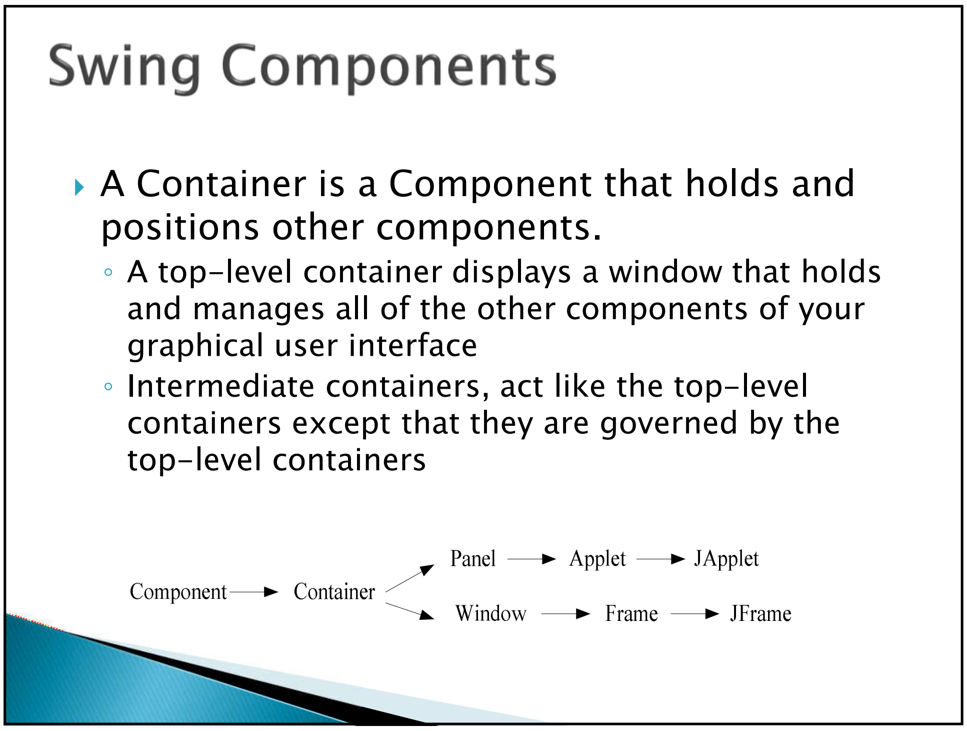
```
      private  class ButtonExitHandler implements
ActionListener {
                 public void actionPerformed(ActionEvent e) {
                            System.exit(0);
                 }
      }

      public static void main(String[] args) {
                 new ButtonTest();
      }

}
```



Swing Components

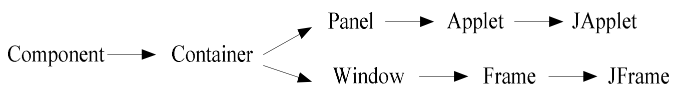
- There are two fundamental types of Swing components:
  - containers
  - basic components.





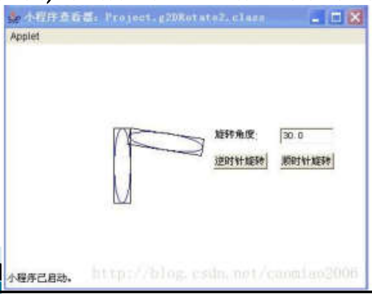
Swing Components

- A Container is a Component that holds and positions other components.
  - A top-level container displays a window that holds and manages all of the other components of your graphical user interface
  - Intermediate containers, act like the top-level containers except that they are governed by the top-level containers



Swing Components

- There are three types of top-level containers:
  - windows (e.g. JFrame) that has a title bar with close and iconifying buttons,
  - dialogs (e.g. JOptionPane, ProgressMonitor, JDialog, JFileChooser, JColorChooser) that are children of windows
  - applets (JApplet) that run in a web browser.

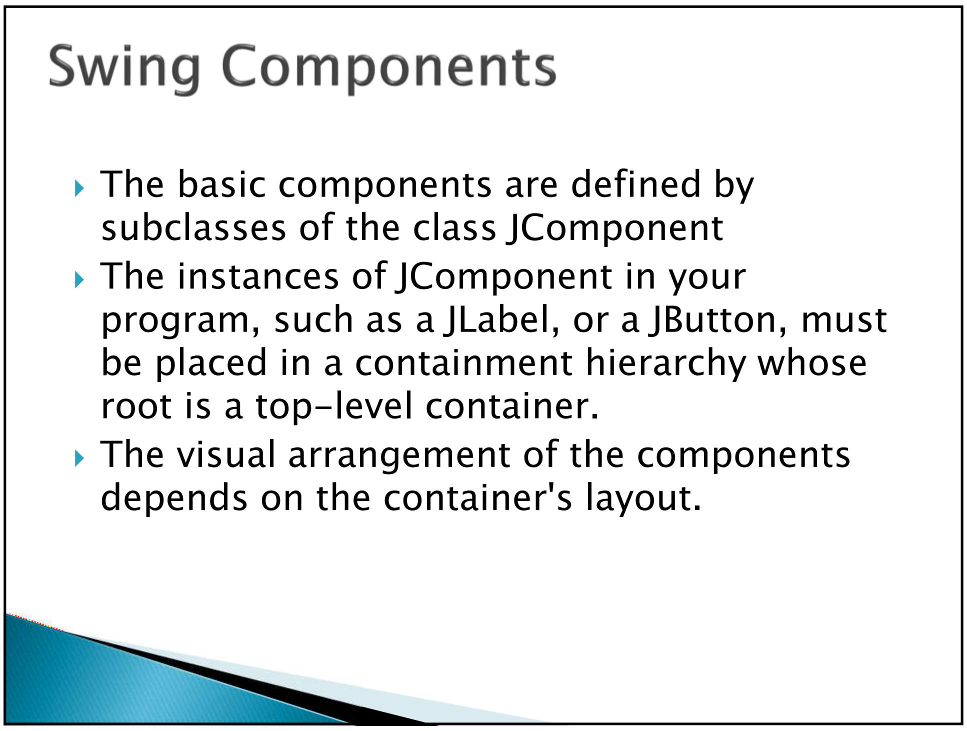

小程序查看器: Project.g2DRotate2.class
Applet
旋转角度:
30.0
逆时针旋转
顺时针旋转
小程序已启动。
http://blog.csdn.net/canmiao2006

Example: a simple Frame

```
import javax.swing.JFrame;

public class MyFirstFrame extends JFrame { //inherit JFrame
    public static void main(String args[]) {
        MyFirstFrame frame = new MyFirstFrame();
        frame.setVisible(true); // set frame visible
    }
    public MyFirstFrame() {
        super();
        setTitle("My first JFrame"); // set title
        setBounds(100, 100, 500, 375); // set position and size
        getContentPane().setLayout(null); // set layout
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);//set close opration
    }
}
```

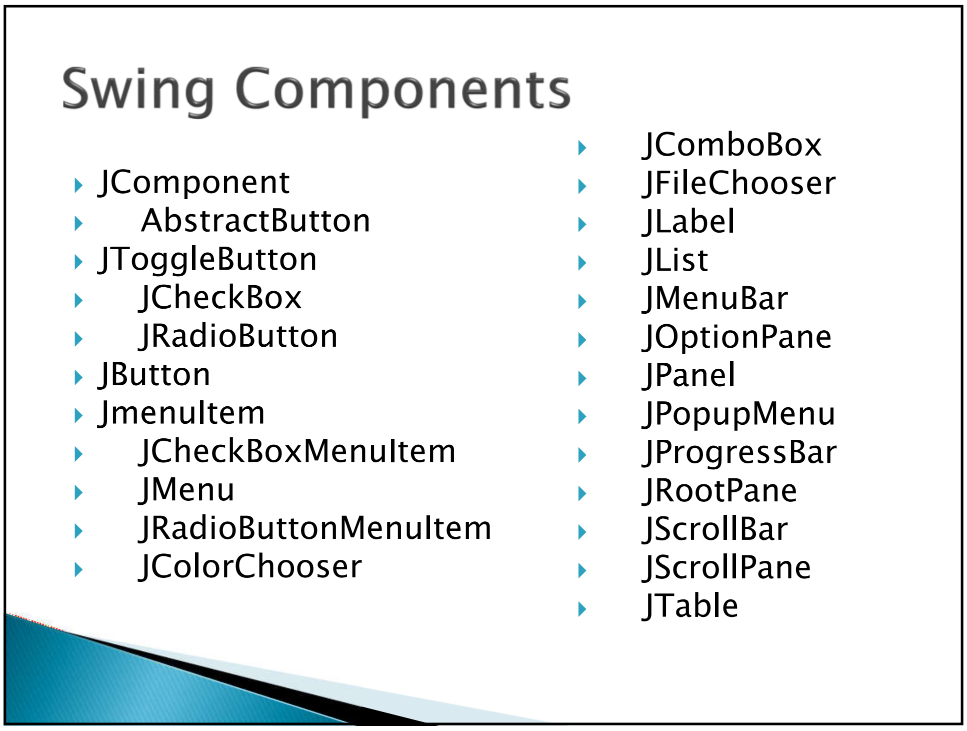
Constant	Value	Operation
HIDE_ON_CLOSE	1	Hide the window
DO_NOTHING_ON_CLOSE	0	Do nothing
DISPOSE_ON_CLOSE	2	Dispose current window
EXIT_ON_CLOSE	3	Exit the whole app

18



Swing Components

- The basic components are defined by subclasses of the class JComponent
- The instances of JComponent in your program, such as a JLabel, or a JButton, must be placed in a containment hierarchy whose root is a top-level container.
- The visual arrangement of the components depends on the container's layout.



Swing Components

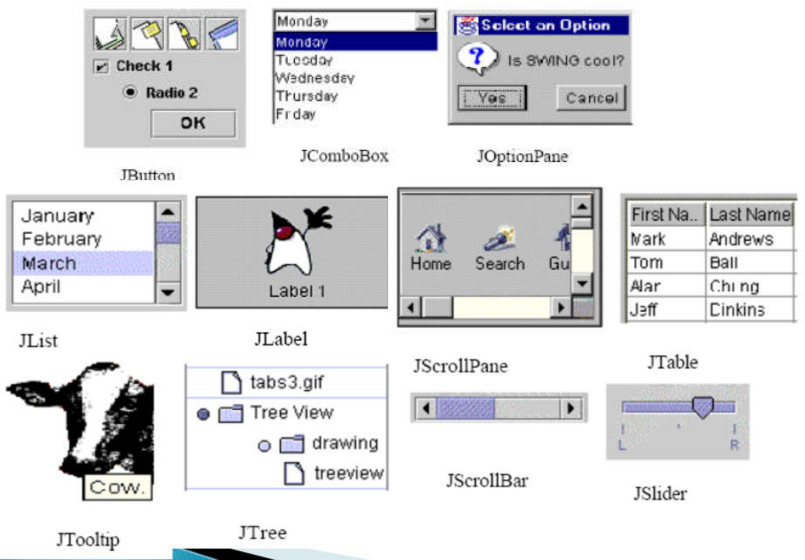
- JComponent
- AbstractButton
- JToggleButton
- JCheckBox
- JRadioButton
- JButton
- JmenuItem
- JCheckBoxMenuItem
- JMenu
- JRadioButtonMenuItem
- JColorChooser
- JComboBox
- JFileChooser
- JLabel
- JList
- JMenuBar
- JOptionPane
- JPanel
- JPopupMenu
- JProgressBar
- JRootPane
- JScrollBar
- JScrollPane
- JTable

Swing Components

- JTextComponent
- JEditorPane
  - JTextPane
- JTextArea
- JTextField
  - JFormattedTextField
  - JPasswordField
  - JToolBar
  - JToolTip

- public void setForeground (Color fg);
- public void setBackground (Color bg);
- public void setLocation (Point p);
- public void setSize (Dimension d);
- public void setFont (Font f);

Swing Components



22

Basic Components

- A JLabel is just a single line of text.
  - JLabel label = new JLabel("Name: ", JLabel.LEFT);
- You can change the text displayed in a label by the setText() method, and use setFont(Font font) to change the font, use setHorizontalAlignment(int alignment) to change the text align orientation

Constant	Value	Orientation
LEFT	2	Left
CENTER	0	Center
RIGHT	4	Right

- final JLabel label = new JLabel();
- label.setBounds(0, 0, 492, 341);
- label.setText("Welcome to Swing!");
- label.setFont(new Font("", Font.BOLD, 22));
- label.setHorizontalAlignment(JLabel.CENTER);
- getContentPane().add(label);

24

Basic Components

- A JTextField is a field where the user can edit text.
- The JTextField contains a single line of editable text.
- a JTextArea can display multiple lines.

```
final JLabel label = new JLabel();
label.setText("姓名：");
label.setBounds(10, 10, 46, 15);
getContentPane().add(label);

JTextField textField = new JTextField();
textField.setHorizontalAlignment(JTextField.CENTER
);
textField.setFont(new Font("", Font.BOLD, 12));
textField.setBounds(62, 7, 120, 21);
getContentPane().add(textField);
```

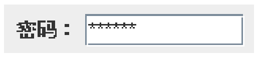
26

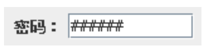
Basic Components

- setText() Substitute newText for current contents
- getText() Return a copy of the current contents
- getSelectedText()   Return the selected text
- select()   Change the selection;
- selectAll()   Select the entire text
- getSelectionStart()   Get starting point of selection, characters starts from zero
- getSelectionEnd()   Get end point of selection
- setEditable()   Specify whether or not the text in the component can be edited by the user

Basic Components

- JPasswordField is a subclass of JTextField, which is identical except that it masks the characters that it contains with asterisks
- Use setEchoChar(char c) to set display character





Basic Components

- A JTextArea is a multi-line area that displays plain text
- The specific methods for JTextArea class are
  - append()
    - Adds the specified text at the end of the current contents;
  - insert()
    - Inserts the specified text, starting at specified position;
  - replaceRange()
    - Replaces the text from the specified position start to position end;
  - setLineWrap()
    - Sets whether it can automatically wrap to the next line or not. The default value is false.

- JTextArea textArea = **new JTextArea();** textArea.setBounds(20, 20, 200, 200);
- textArea.setColumns(15);
- textArea.setRows(3);
- textArea.setLineWrap(**true);** getContentPane().add(textArea);

30

Basic Components

- A JTextArea does not have scroll bars by default, but scroll bars can be added easily by putting the text area in a scroll pane:
  - JScrollPane scroller = new JScrollPane(new JTextArea());
- A JScrollPane is a component that provides scrolling for another component.
- The horizontal and/or vertical scroll bars will appear automatically.
- Several Swing components, including the JTextArea, are designed specifically to work with JScrollPane.

- JTextArea textArea = **new JTextArea();** textArea.setColumns(15);
- textArea.setRows(3);
- textArea.setLineWrap(**true**);
- **final JScrollPane scrollPane = new JScrollPane();** scrollPane.setViewportView(textArea);
- Dimension dime = textArea.getPreferredSize(); scrollPane.setBounds(62, 5, dime.width, dime.height); getContentPane().add(scrollPane);

32

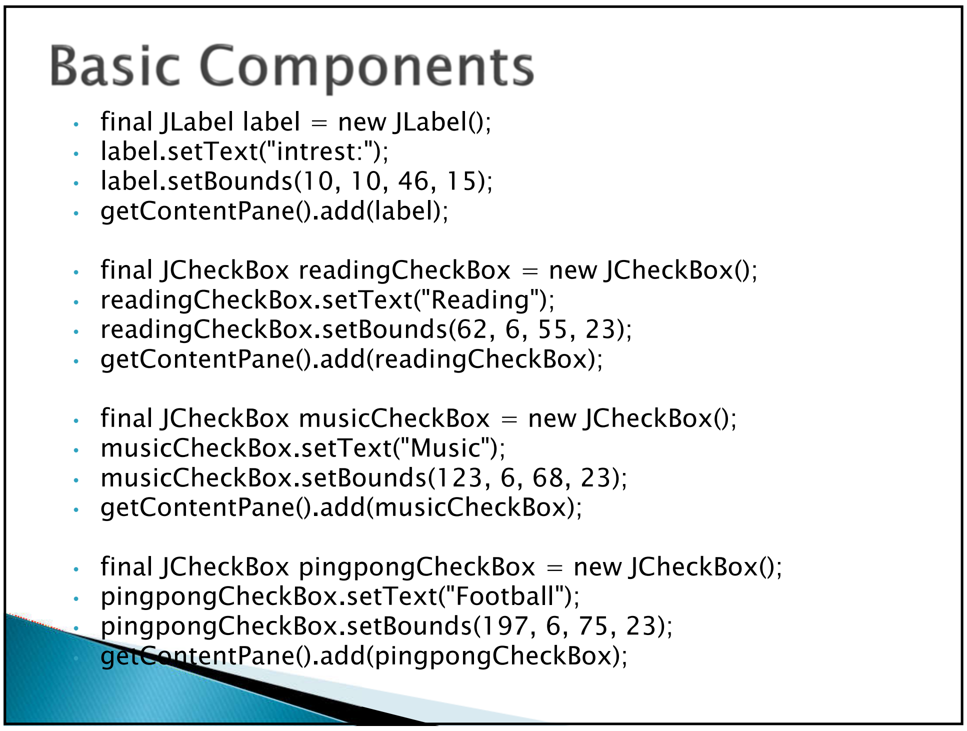
Basic Components

- Buttons can be configured, and to some degree controlled, by Actions.
- The JButton generates an ActionEvent when the user clicks on a button.

```
JButton button = new JButton();
button.setBounds(50, 50, 200, 23);
button.setText("Button1");
getContentPane().add(button);
```

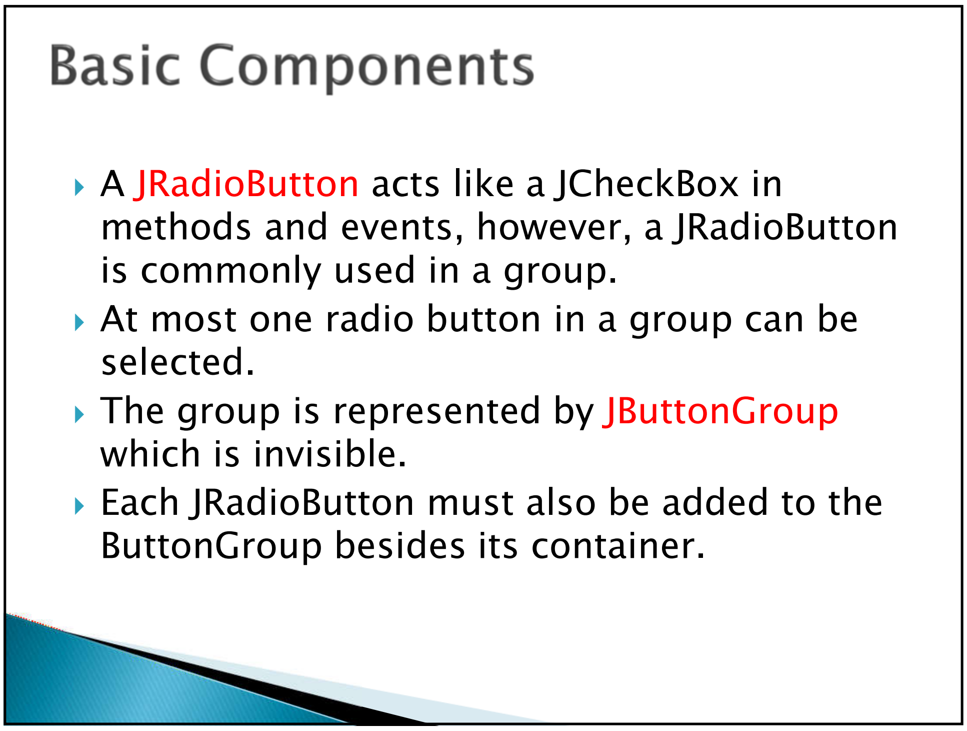
Basic Components

- A JCheckBox is a component that has two states: selected or unselected.
- The current state of a checkbox is set by its setSelected() method and is determined by its isSelected() method.
- A Checkbox generates an ActionEvent when the user clicks it.
- However, there is no ActionEvent generated if you change the state by the setSelected() method.



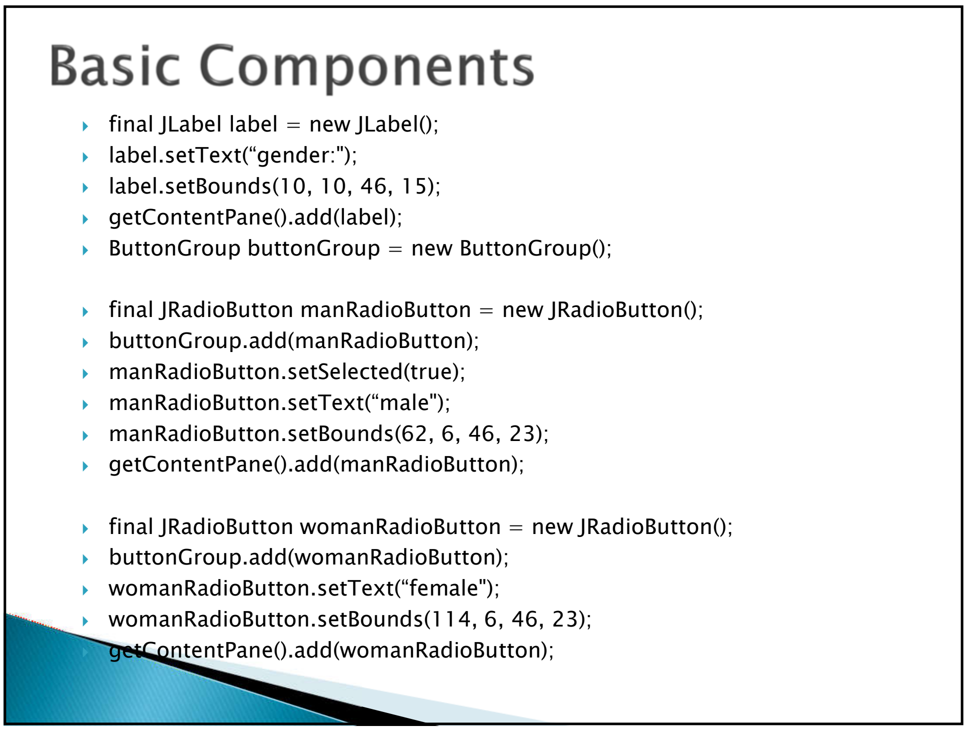
Basic Components

```
final JLabel label = new JLabel();
label.setText("intrest:");
label.setBounds(10, 10, 46, 15);
getContentPane().add(label);

final JCheckBox readingCheckBox = new JCheckBox();
readingCheckBox.setText("Reading");
readingCheckBox.setBounds(62, 6, 55, 23);
getContentPane().add(readingCheckBox);

final JCheckBox musicCheckBox = new JCheckBox();
musicCheckBox.setText("Music");
musicCheckBox.setBounds(123, 6, 68, 23);
getContentPane().add(musicCheckBox);

final JCheckBox pingpongCheckBox = new JCheckBox();
pingpongCheckBox.setText("Football");
pingpongCheckBox.setBounds(197, 6, 75, 23);
getContentPane().add(pingpongCheckBox);
```



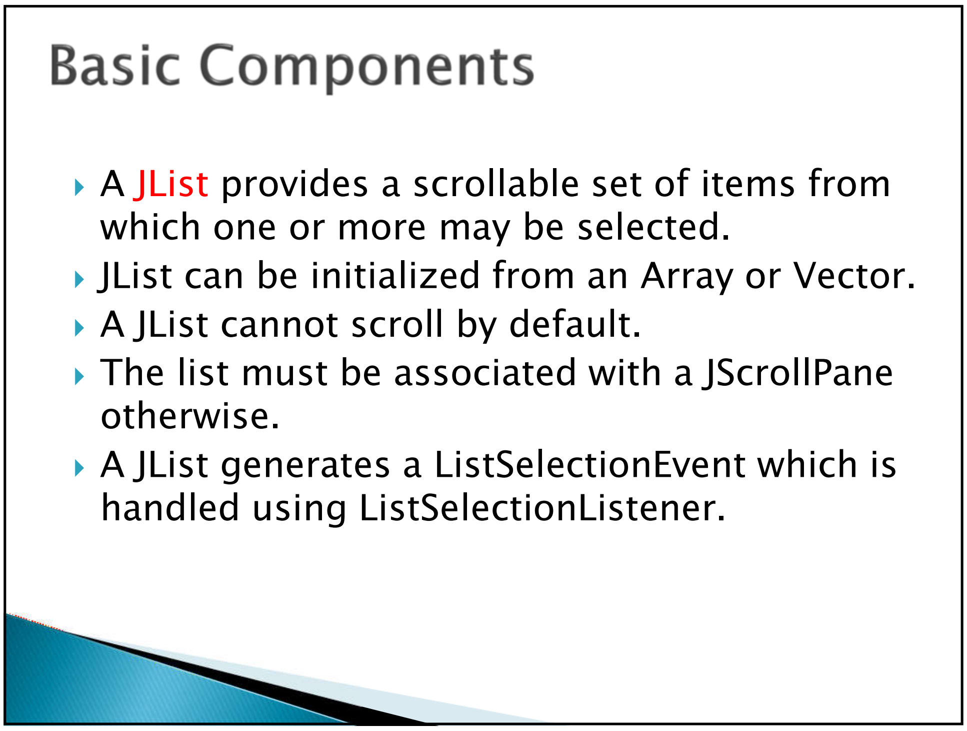
Basic Components

- A JRadioButton acts like a JCheckBox in methods and events, however, a JRadioButton is commonly used in a group.
- At most one radio button in a group can be selected.
- The group is represented by JButtonGroup which is invisible.
- Each JRadioButton must also be added to the ButtonGroup besides its container.



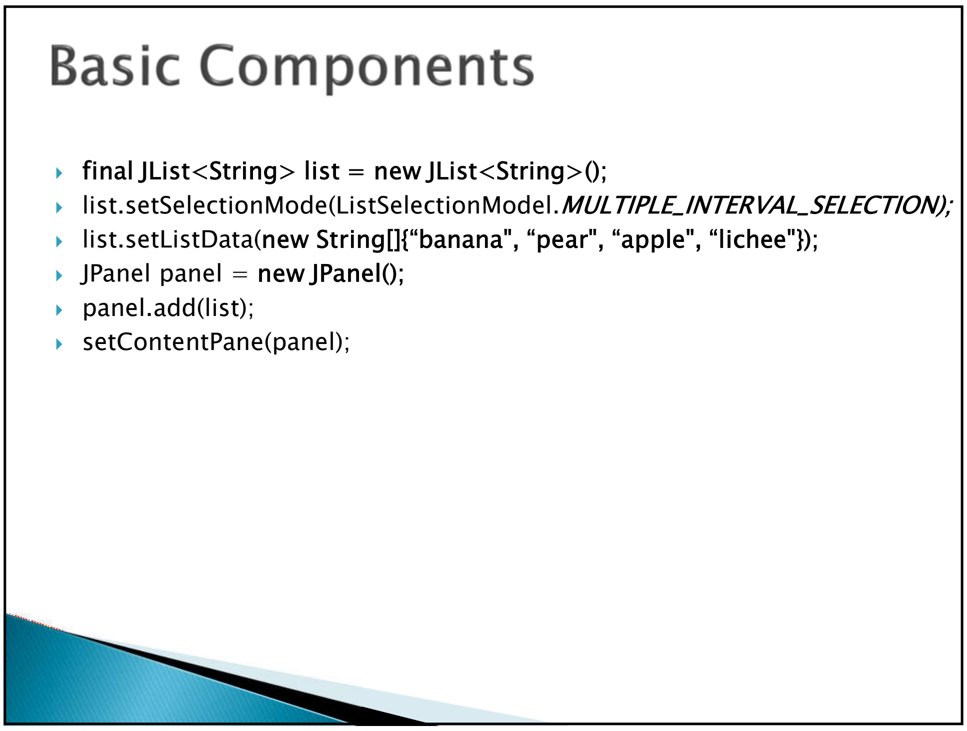
Basic Components

```
final JLabel label = new JLabel();
label.setText(“gender:");
label.setBounds(10, 10, 46, 15);
getContentPane().add(label);
ButtonGroup buttonGroup = new ButtonGroup();

final JRadioButton manRadioButton = new JRadioButton();
buttonGroup.add(manRadioButton);
manRadioButton.setSelected(true);
manRadioButton.setText(“male");
manRadioButton.setBounds(62, 6, 46, 23);
getContentPane().add(manRadioButton);

final JRadioButton womanRadioButton = new JRadioButton();
buttonGroup.add(womanRadioButton);
womanRadioButton.setText(“female");
womanRadioButton.setBounds(114, 6, 46, 23);
getContentPane().add(womanRadioButton);
```



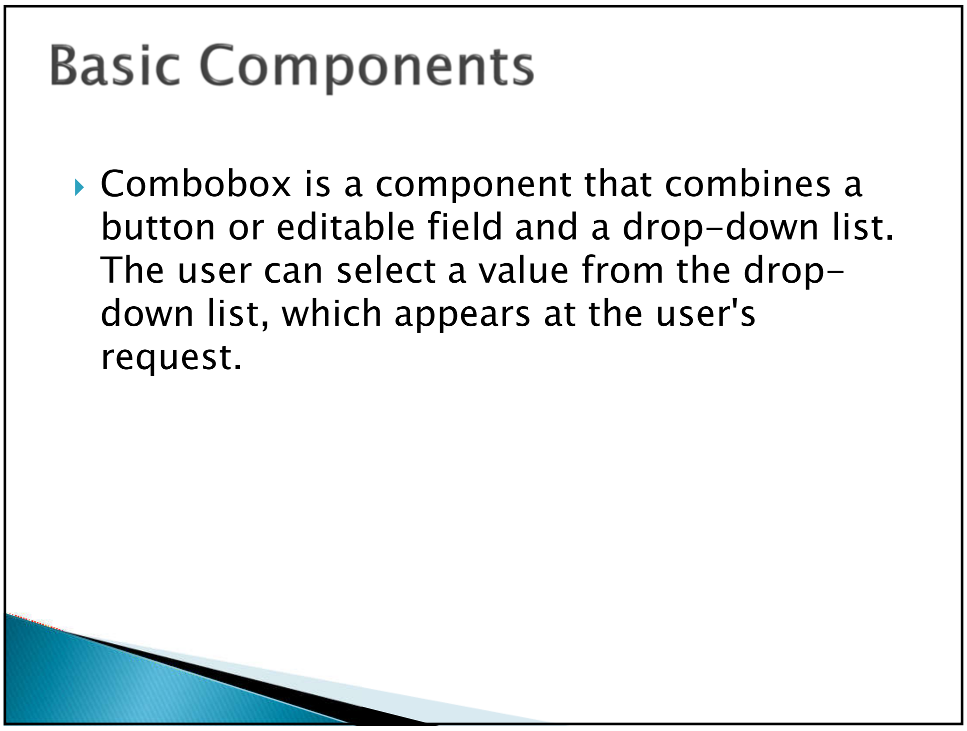
Basic Components

- A JList provides a scrollable set of items from which one or more may be selected.
- JList can be initialized from an Array or Vector.
- A JList cannot scroll by default.
- The list must be associated with a JScrollPane otherwise.
- A JList generates a ListSelectionEvent which is handled using ListSelectionListener.



Basic Components

```
final JList<String> list = new JList<String>();
list.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
list.setListData(new String[]{“banana", “pear", “apple", “lichee"});
JPanel panel = new JPanel();
panel.add(list);
setContentPane(panel);
```



Basic Components

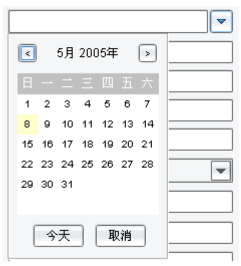
- Combobox is a component that combines a button or editable field and a drop-down list. The user can select a value from the drop-down list, which appears at the user's request.

Basic Components

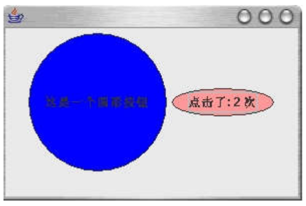
- **final JLabel label = new JLabel();**
- label.setText("degree:");
- label.setBounds(10, 10, 46, 15);
- getContentPane().add(label);
- String[] schoolAges = {"bachelor", "master", "doctor" };

- JComboBox comboBox = new JComboBox(schoolAges);
- comboBox.setEditable(**true**);
- comboBox.setMaximumRowCount(3);
- comboBox.insertItemAt("junior", 0);
- comboBox.setSelectedItem("bachelor ");
- comboBox.setBounds(62, 7, 104, 21);
- getContentPane().add(comboBox);

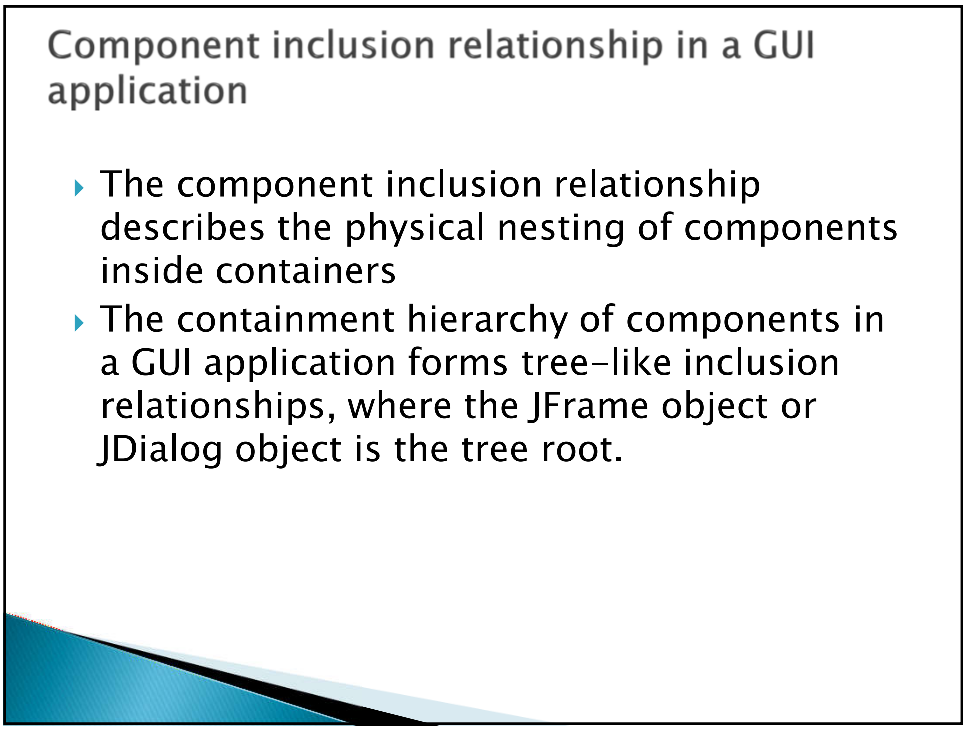
Self-defined components



DatePicker

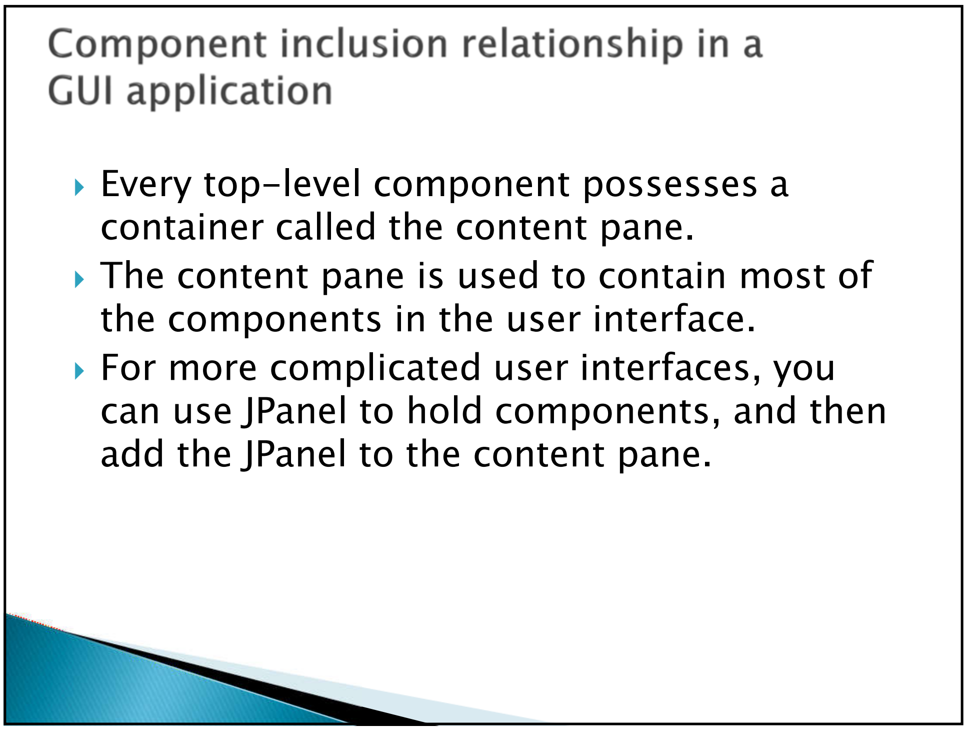


RoundButton



Component inclusion relationship in a GUI application

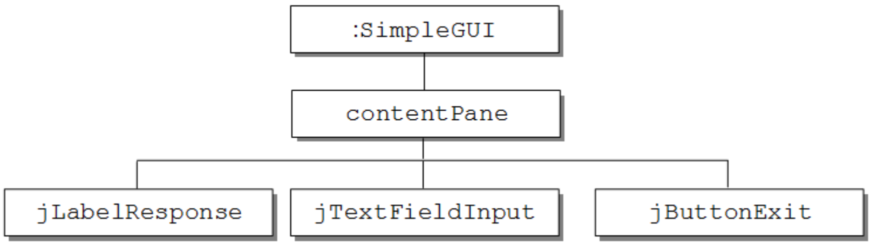
- The component inclusion relationship describes the physical nesting of components inside containers
- The containment hierarchy of components in a GUI application forms tree-like inclusion relationships, where the JFrame object or JDialog object is the tree root.

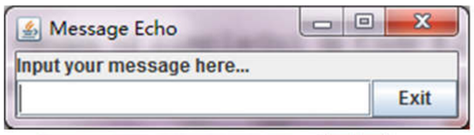


Component inclusion relationship in a GUI application

- Every top-level component possesses a container called the content pane.
- The content pane is used to contain most of the components in the user interface.
- For more complicated user interfaces, you can use JPanel to hold components, and then add the JPanel to the content pane.

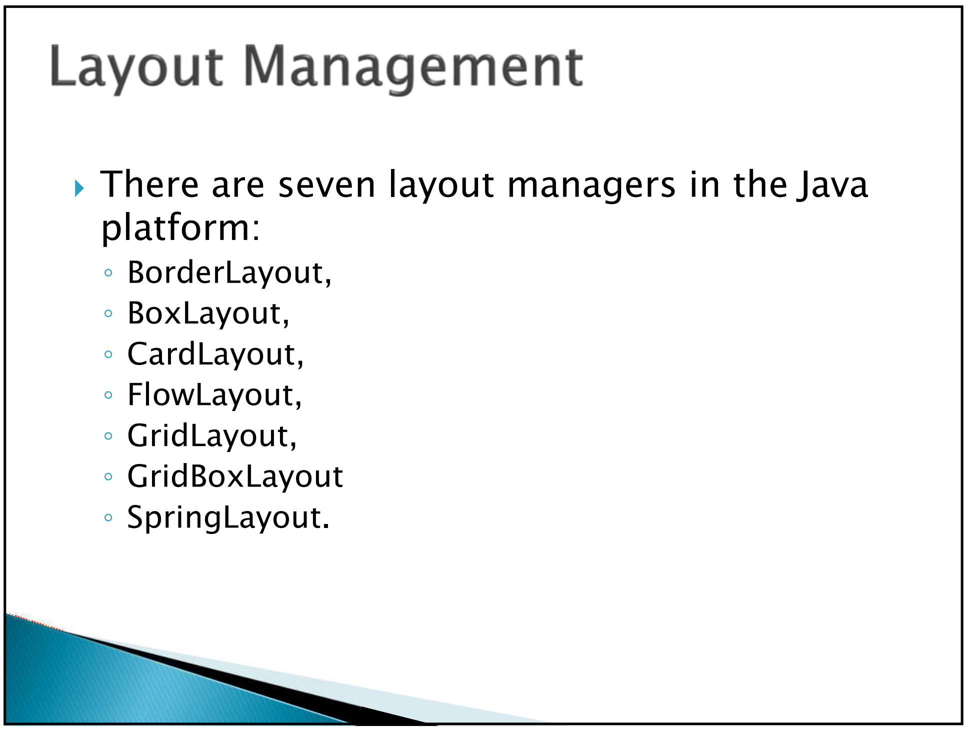
Component inclusion relationship in a GUI application





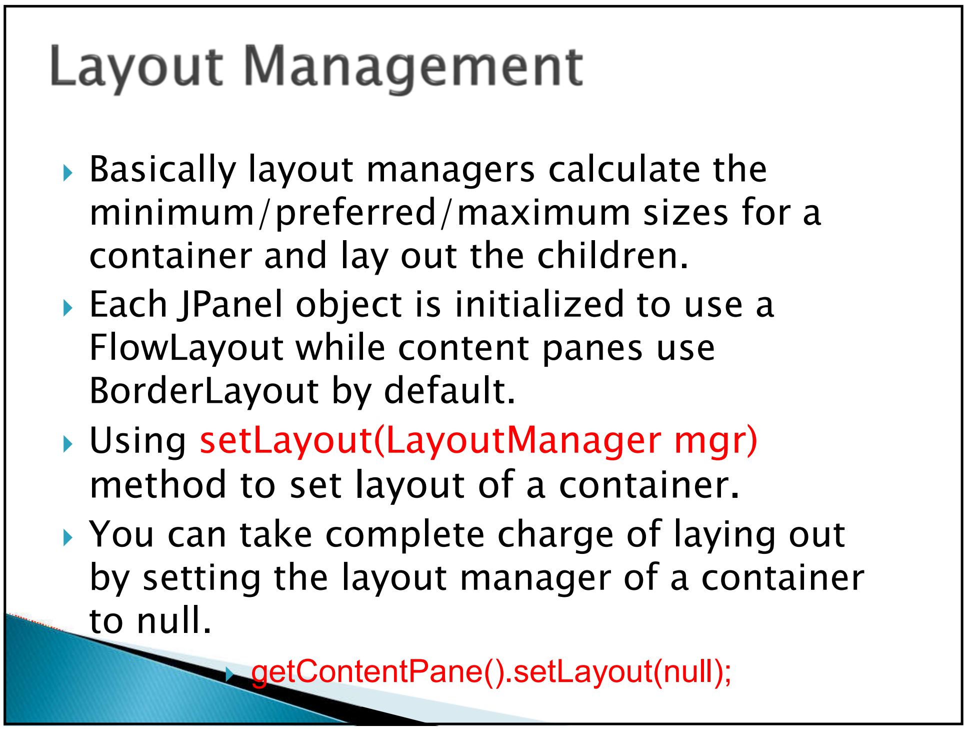
Layout Management

- The sizes and positions of the components in a container are controlled by a layout manager.
- Every container, including JPanel, has a default layout.



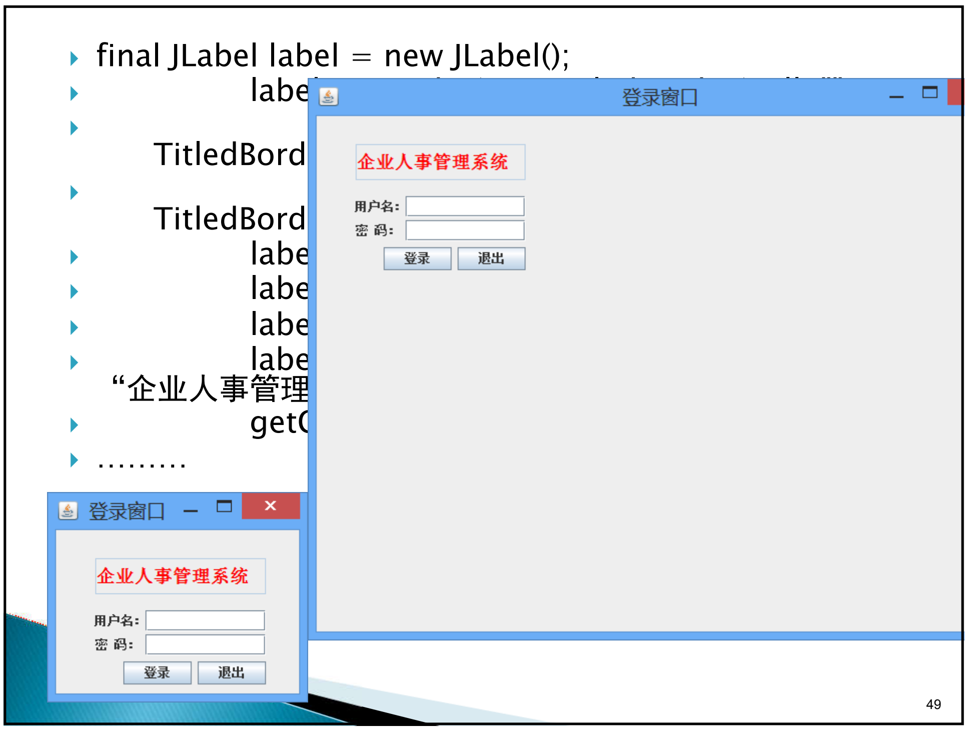
Layout Management

- There are seven layout managers in the Java platform:
  - BorderLayout,
  - BoxLayout,
  - CardLayout,
  - FlowLayout,
  - GridLayout,
  - GridBoxLayout
  - SpringLayout.



Layout Management

- Basically layout managers calculate the minimum/preferred/maximum sizes for a container and lay out the children.
- Each JPanel object is initialized to use a FlowLayout while content panes use BorderLayout by default.
- Using setLayout(LayoutManager mgr) method to set layout of a container.
- You can take complete charge of laying out by setting the layout manager of a container to null.
  - getContentPane().setLayout(null);



- final JLabel label = new JLabel();
- labe
- TitledBord
- TitledBord
- labe
- labe
- labe
- labe
  “企业人事管理
- get(
- ....

登录窗口

企业人事管理系统

用户名:

密 码:

登录 退出

登录窗口

企业人事管理系统

用户名:

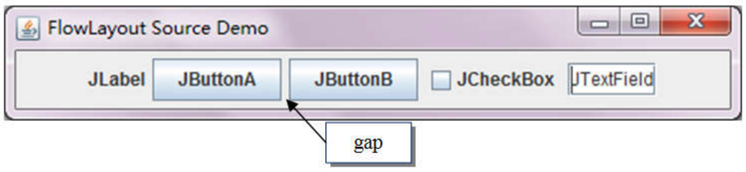
密 码:

登录 退出

49

Layout Management

- A flow layout arranges components in a directional flow, much like lines of text in a paragraph.
- Constructors in FlowLayout
  - FlowLayout()
  - FlowLayout(int align)
  - FlowLayout(int align, int hgap, int vgap)



- final FlowLayout flowLayout = new FlowLayout();
- flowLayout.setHgap(10);
- flowLayout.setVgap(10);
- flowLayout.setAlignment(FlowLayout.*LEFT);*
- getContentPane().setLayout(flowLayout);

- final JButton aButton = new JButton();
- aButton.setText("按钮 A");
- getContentPane().add(aButton);

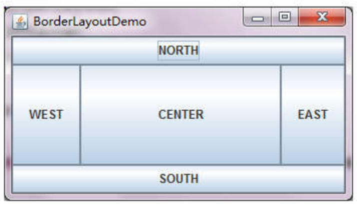
- final JButton bButton = new JButton();
- bButton.setText("按钮 B");
- getContentPane().add(bButton);

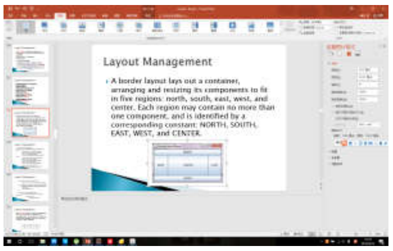
- final JButton cButton = new JButton();
- cButton.setText("按钮 C");
- getContentPane().add(cButton);

51

Layout Management

- A border layout lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center. Each region may contain no more than one component, and is identified by a corresponding constant: NORTH, SOUTH, EAST, WEST, and CENTER.




Layout Management
A border layout lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center. Each region may contain no more than one component, and is identified by a corresponding constant: NORTH, SOUTH, EAST, WEST, and CENTER.

```
final BorderLayout borderLayout = new BorderLayout();
borderLayout.setHgap(10);
borderLayout.setVgap(10);
Container panel = getContentPane();
panel.setLayout(borderLayout);

final JButton aButton = new JButton();
aButton.setText("A");
getContentPane().add(aButton,BorderLayout.NORTH);

final JButton bButton = new JButton();
bButton.setText(" B");
getContentPane().add(bButton,BorderLayout.CENTER);

final JButton cButton = new JButton();
cButton.setText("C");
getContentPane().add(cButton,BorderLayout.WEST);
```

Layout Management

- The GridLayout class is a layout manager that lays out a container's components in a rectangular grid. The container is divided into equal-sized rectangles, and one component is placed in each rectangle.

1#	2#	3#
4#	5#	6#

Layout Management

- Constructors in GridLayout class
  - GridLayout() single row single line
  - GridLayout(int rows, int cols)
  - GridLayout(int rows, int cols, int hgap, int vgap)

```
final GridLayout gridLayout = new GridLayout(3,3);
      getContentPane().setLayout(gridLayout);

      for(int i = 0;i<9;i++){
            getContentPane().add(new JButton(i+""));
      }
```

55

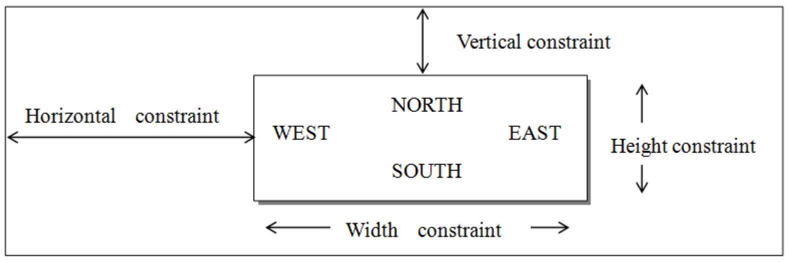
Layout Management

- A layout manager that allows multiple components to be laid out either vertically or horizontally. The components will not wrap so, for example, a vertical arrangement of components will stay vertically arranged when the frame is resized.

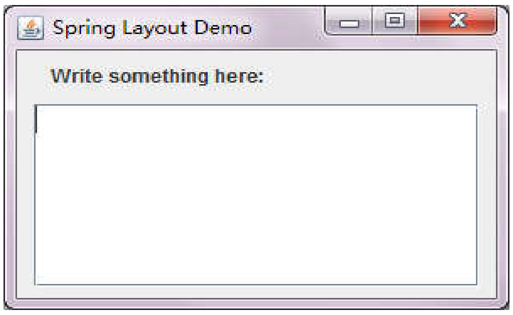
56

```
Box hBox = Box.createHorizontalBox();
Box vBox = Box.createVerticalBox();

hBox.add(new JButton("1"));
hBox.add(Box.createHorizontalGlue());
hBox.add(new JButton("2"));
hBox.add(Box.createHorizontalStrut(10));
hBox.add(new JButton("3"));

vBox.add(new JButton("4"));
vBox.add(Box.createVerticalGlue());
vBox.add(new JButton("5"));
vBox.add(Box.createVerticalStrut(10));
vBox.add(new JButton("6"));

getContentPane().add(hBox);
getContentPane().add(vBox);
```

Layout Management

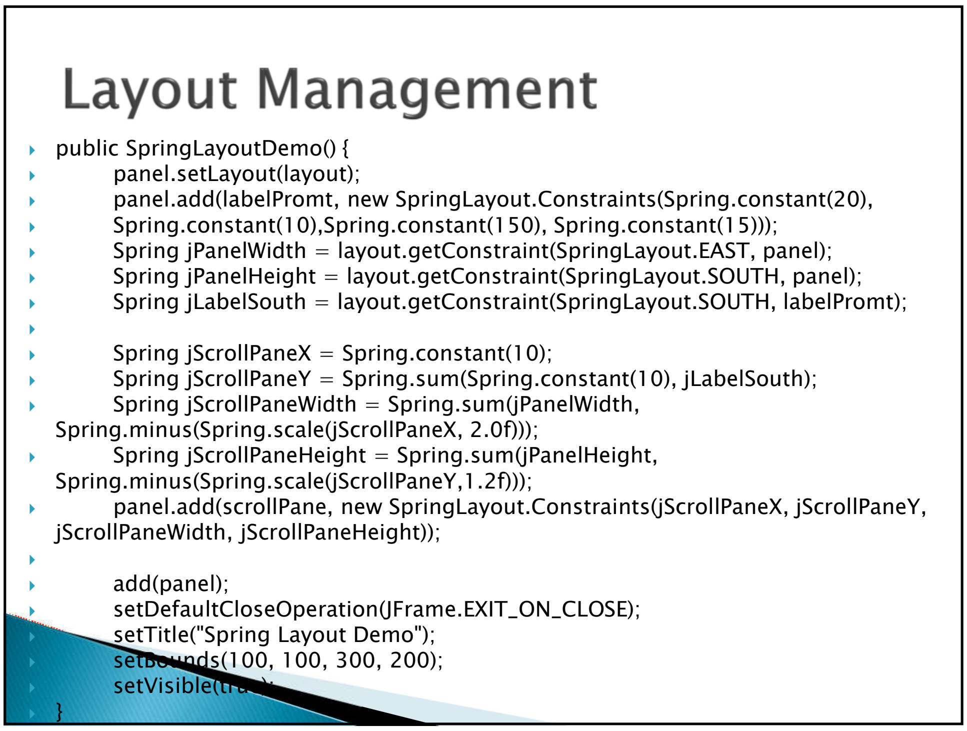


Layout Management



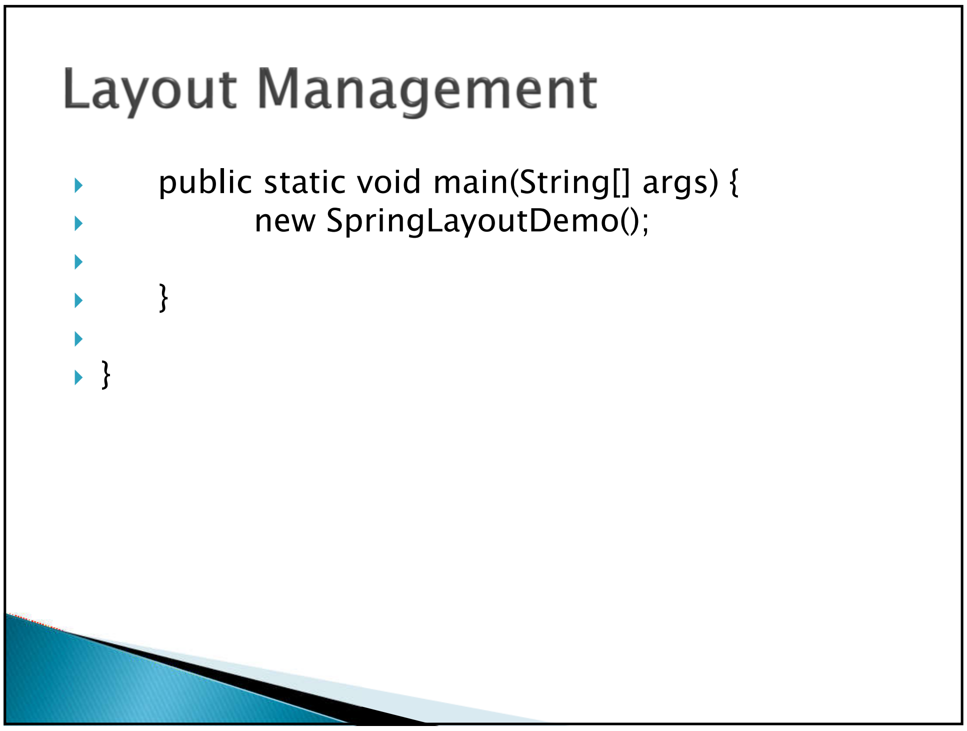
Layout Management

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.Spring;
import javax.swing.SpringLayout;

public class SpringLayoutDemo extends JFrame {

    private JPanel panel = new JPanel();
    private JLabel labelPromt = new JLabel("Write something
here:");
    private JTextArea textArea = new JTextArea();
    private JScrollPane scrollPane = new
JScrollPane(textArea);
    private SpringLayout layout = new SpringLayout();
```



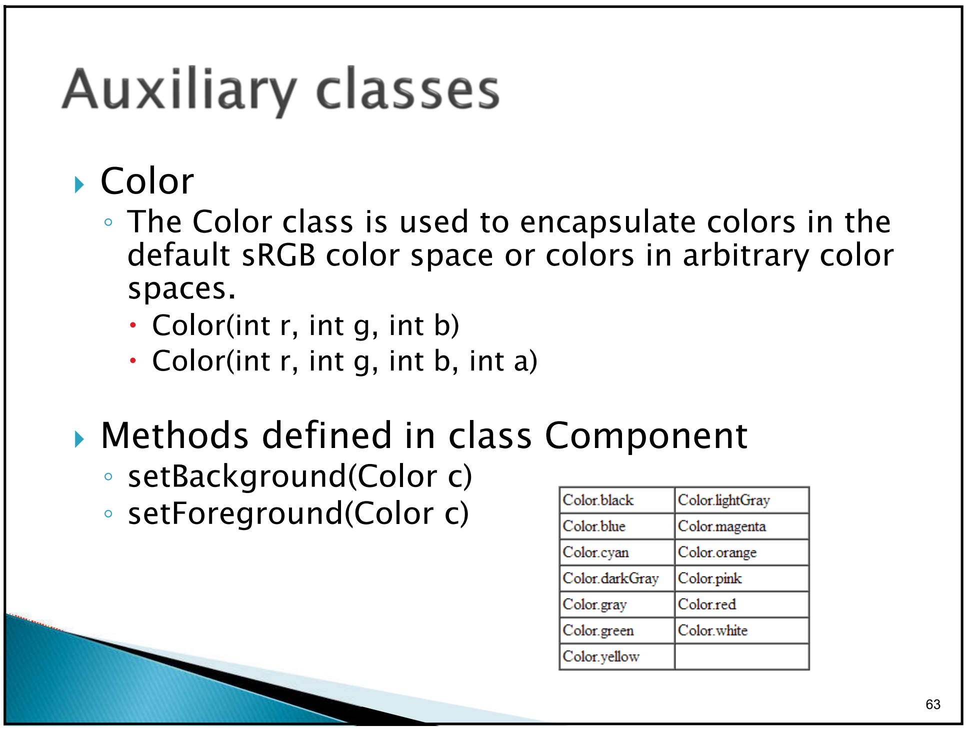
Layout Management

```
public SpringLayoutDemo() {
        panel.setLayout(layout);
        panel.add(labelPromt, new SpringLayout.Constraints(Spring.constant(20),
        Spring.constant(10),Spring.constant(150), Spring.constant(15)));
        Spring jPanelWidth = layout.getConstraint(SpringLayout.EAST, panel);
        Spring jPanelHeight = layout.getConstraint(SpringLayout.SOUTH, panel);
        Spring jLabelSouth = layout.getConstraint(SpringLayout.SOUTH, labelPromt);

        Spring jScrollPaneX = Spring.constant(10);
        Spring jScrollPaneY = Spring.sum(Spring.constant(10), jLabelSouth);
        Spring jScrollPaneWidth = Spring.sum(jPanelWidth,
Spring.minus(Spring.scale(jScrollPaneX, 2.0f)));
        Spring jScrollPaneHeight = Spring.sum(jPanelHeight,
Spring.minus(Spring.scale(jScrollPaneY,1.2f)));
        panel.add(scrollPane, new SpringLayout.Constraints(jScrollPaneX, jScrollPaneY,
jScrollPaneWidth, jScrollPaneHeight));

        add(panel);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setTitle("Spring Layout Demo");
        setBounds(100, 100, 300, 200);
        setVisible(true);
}
```



Layout Management

```
        public static void main(String[] args) {
                new SpringLayoutDemo();

        }

}
```

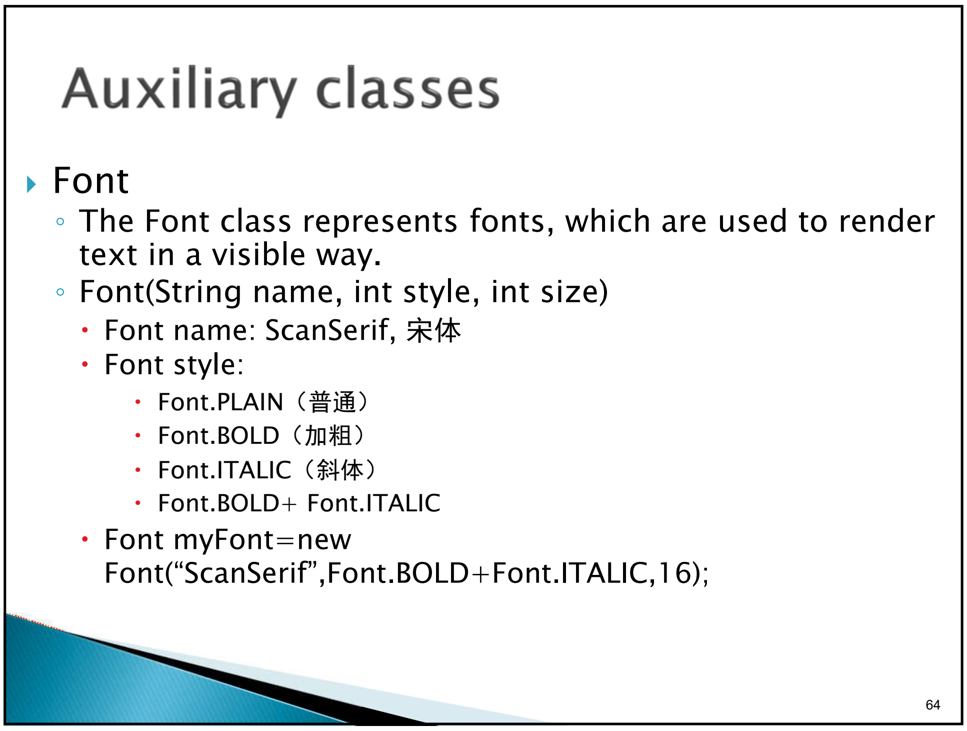


Auxiliary classes

- Color
  - The Color class is used to encapsulate colors in the default sRGB color space or colors in arbitrary color spaces.
    - Color(int r, int g, int b)
    - Color(int r, int g, int b, int a)
- Methods defined in class Component
  - setBackground(Color c)
  - setForeground(Color c)

Color.black	Color.lightGray
Color.blue	Color.magenta
Color.cyan	Color.orange
Color.darkGray	Color.pink
Color.gray	Color.red
Color.green	Color.white
Color.yellow	

63



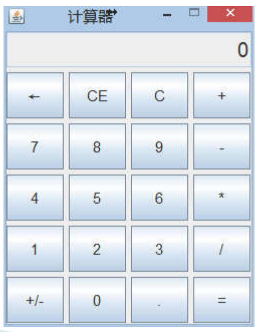
Auxiliary classes

- Font
  - The Font class represents fonts, which are used to render text in a visible way.
  - Font(String name, int style, int size)
    - Font name: ScanSerif, 宋体
    - Font style:
      - Font.PLAIN（普通）
      - Font.BOLD（加粗）
      - Font.ITALIC（斜体）
      - Font.BOLD+ Font.ITALIC
    - Font myFont=new Font(“ScanSerif”,Font.BOLD+Font.ITALIC,16);

64

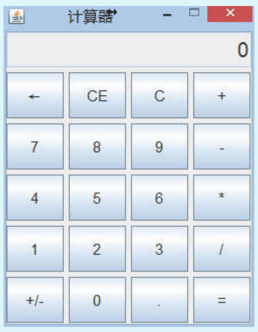
JPanel

- The JPanel class provides general-purpose containers for lightweight components.
  - It allows for better organization
  - It allows nest within each other and etc.



65

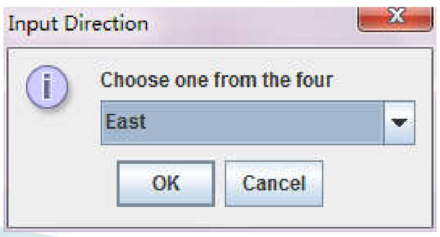
```
JTextField screen = new JTextField("0");
screen.setHorizontalAlignment(JTextField.RIGHT);
screen.setEditable(false);

JPanel panel = new JPanel();
GridLayout gl = new GridLayout(5, 4, 5, 5);
panel.setLayout(gl);

for (int i = 0; i < 20; i++) {
    JButton button = new JButton();
    button.setText(""+i);
    panel.add(button);
}

setTitle("Caculator");
setBounds(100, 100, 250, 320);
BorderLayout layout = new BorderLayout(5,5);
getContentPane().setLayout(layout);
getContentPane().add(screen, BorderLayout.NORTH);
getContentPane().add(panel, BorderLayout.CENTER);
```



66

Dialogs

- A dialog window is a window that used to input data, modify data, change the application settings etc.
- This kind of window can be moved, but cannot be resized, iconified, maximized or minimized normally.



Dialogs

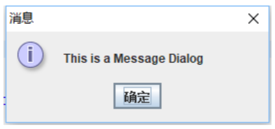
- A typical dialog window consists of five items
  - a title string("Input Direction"),
  - a descriptive message to be placed in the dialog("Choose one from the four"),
  - message type(information here, which is a default icon symbolizes message type) ,
  - options(an array of Strings which consist of "East","West","South" and "North"),
  - initial value ("East")
  - option type(OK and Cancel button here).

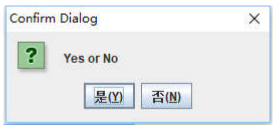
Dialogs

- three basic types:
  - a "message" dialog
  - a "confirm" dialog
  - an "input" dialog
  - an "option" dialog

Dialogs

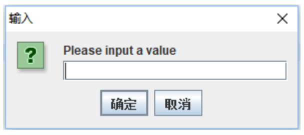
```
JOptionPane.showMessageDialog(null, "This is a Message Dialog");

int confirmResult = JOptionPane.showConfirmDialog(null, "Yes or No",
"Confirm Dialog",JOptionPane.YES_NO_OPTION);
System.out.println("Result of confirm dialog is " + confirmResult);
```

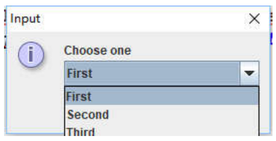




Dialogs

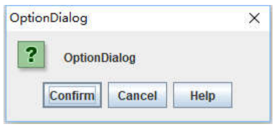
```
String inputValue = JOptionPane.showInputDialog("Please input a
value");
System.out.println("Input is " + inputValue);

Object[] possibleValues = { "First", "Second", "Third" };
Object selectedValue = JOptionPane.showInputDialog(null, "Choose
one", "Input", JOptionPane.INFORMATION_MESSAGE,
null, possibleValues, possibleValues[0]);
System.out.println("you have selected " + selectedValue);
```





Dialogs

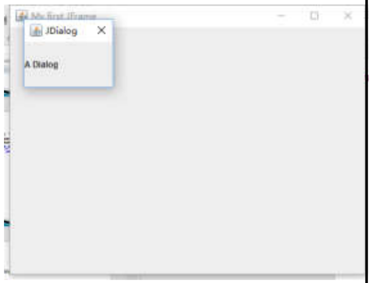
```
Object[] options = { "Confirm", "Cancel", "Help" };
int response = JOptionPane.showOptionDialog(this, "OptionDialog",
"OptionDialog", JOptionPane.YES_OPTION,
JOptionPane.QUESTION_MESSAGE, null, options, options[0]);
System.out.println("you have selected " + response);
```



Dialogs

- A dialog window can be either modal or modeless.
- When a modal dialog pops out, the user will not be able to interact with its parent until it is closed. In other words, a modal dialogs blocks its parent window.
- Modeless dialogs do not block their parents.

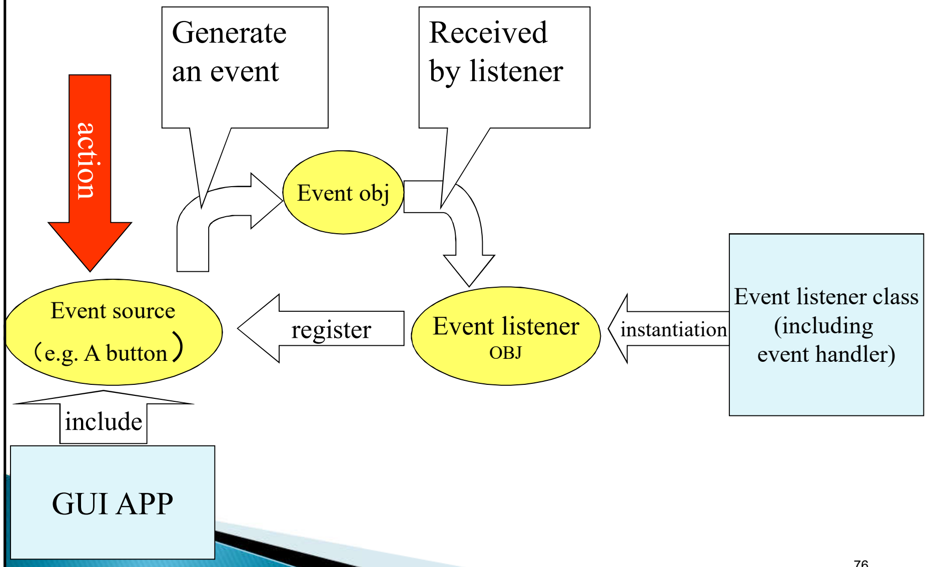
```
public class MyFirstFrame extends JFrame { //inherit JFrame
    public static void main(String args[]) {
        new MyFirstFrame();

    }
    public MyFirstFrame() {
        super();
        setTitle("My first JFrame"); // set title
        setBounds(100, 100, 500, 375); // set position and size
        getContentPane().setLayout(null); // set layout
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);//set close opration
        setVisible(true);
        new MyJDialog(this).setVisible(true);
    }

    class MyJDialog extends JDialog{
        public MyJDialog(MyFirstFrame frame){
            super(frame,"JDialog",true);
            Container container=getContentPane();
            container.add(new JLabel("A Dialog"));
            setBounds(120,120,100,100);
        }
    }
}
```



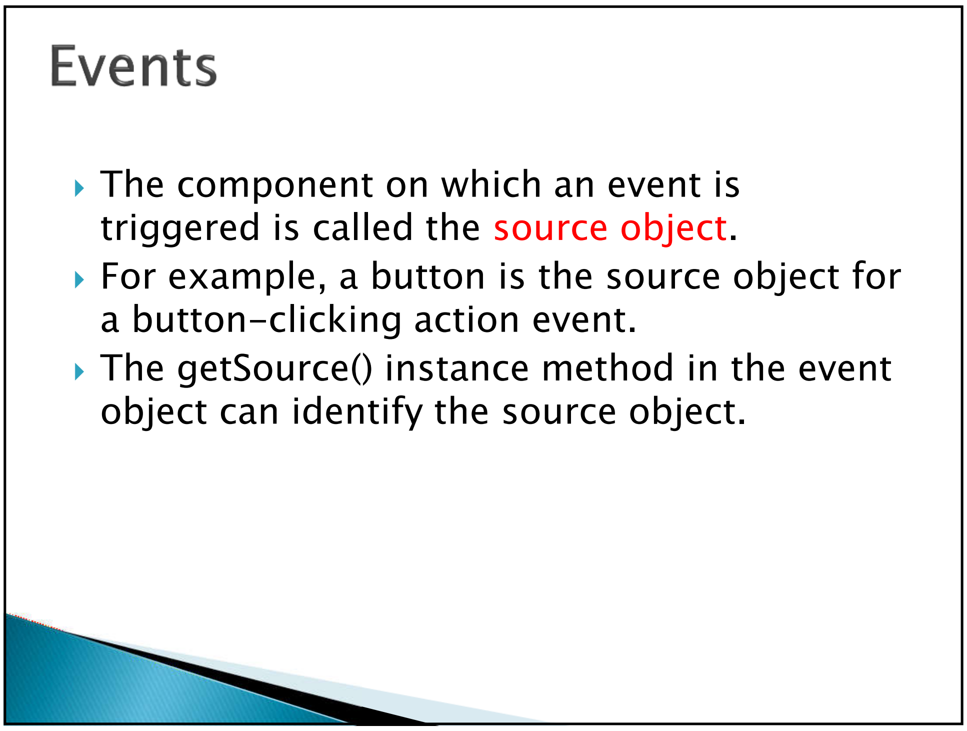
Events

- A user interaction causes an event.
- For example,
  - clicking a button,
  - typing in a text field,
  - selecting an item from a menu,
  - closing a window or moving the mouse are typical events.
- The event is an object and generated by the GUI system.
- An event object is sent from a single source object to one or more registered listeners which are delegated to handle the event

Event Handling

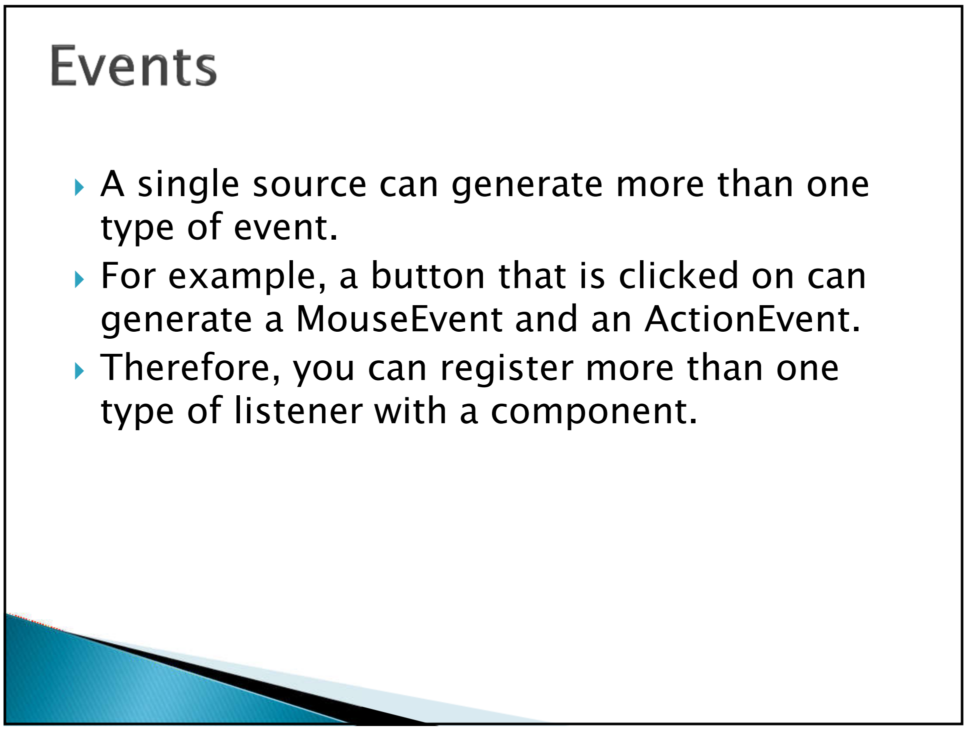


76



Events

- The component on which an event is triggered is called the source object.
- For example, a button is the source object for a button-clicking action event.
- The getSource() instance method in the event object can identify the source object.



Events

- A single source can generate more than one type of event.
- For example, a button that is clicked on can generate a MouseEvent and an ActionEvent.
- Therefore, you can register more than one type of listener with a component.

Events

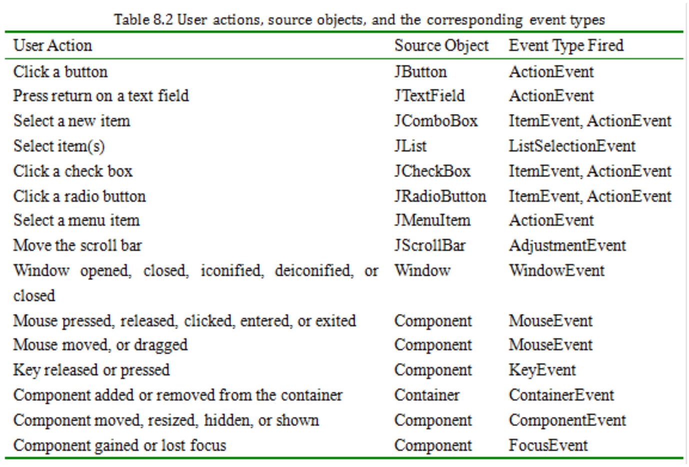
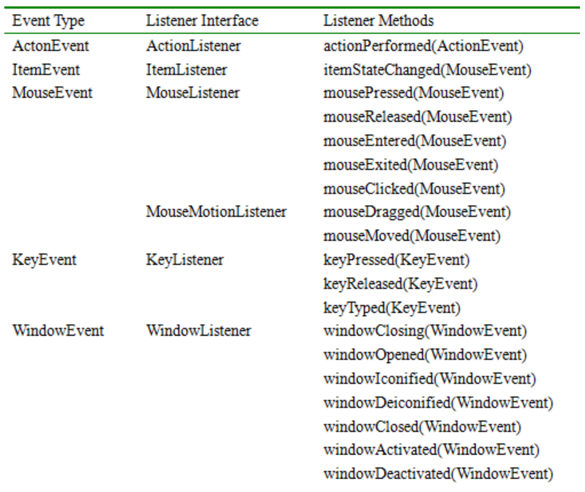


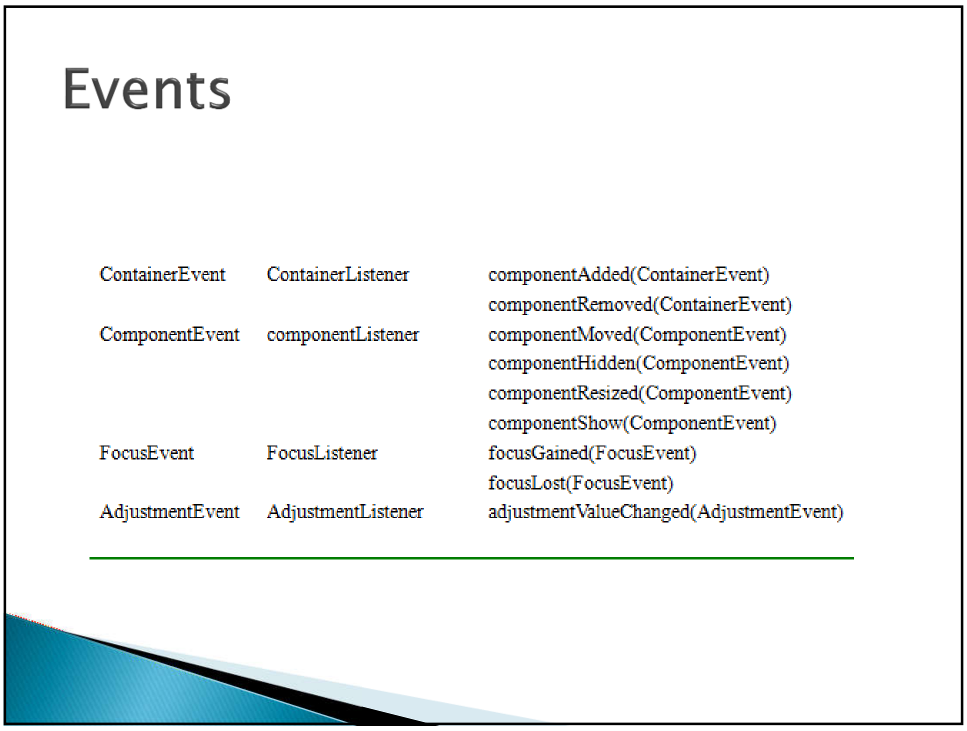
Table 8.2 User actions, source objects, and the corresponding event types

User Action	Source Object	Event Type Fired
Click a button	JButton	ActionEvent
Press return on a text field	JTextField	ActionEvent
Select a new item	JComboBox	ItemEvent, ActionEvent
Select item(s)	JList	ListSelectionEvent
Click a check box	JCheckBox	ItemEvent, ActionEvent
Click a radio button	JRadioButton	ItemEvent, ActionEvent
Select a menu item	JMenuItem	ActionEvent
Move the scroll bar	JScrollBar	AdjustmentEvent
Window opened, closed, iconified, deiconified, or closed	Window	WindowEvent
Mouse pressed, released, clicked, entered, or exited	Component	MouseEvent
Mouse moved, or dragged	Component	MouseEvent
Key released or pressed	Component	KeyEvent
Component added or removed from the container	Container	ContainerEvent
Component moved, resized, hidden, or shown	Component	ComponentEvent
Component gained or lost focus	Component	FocusEvent

Events

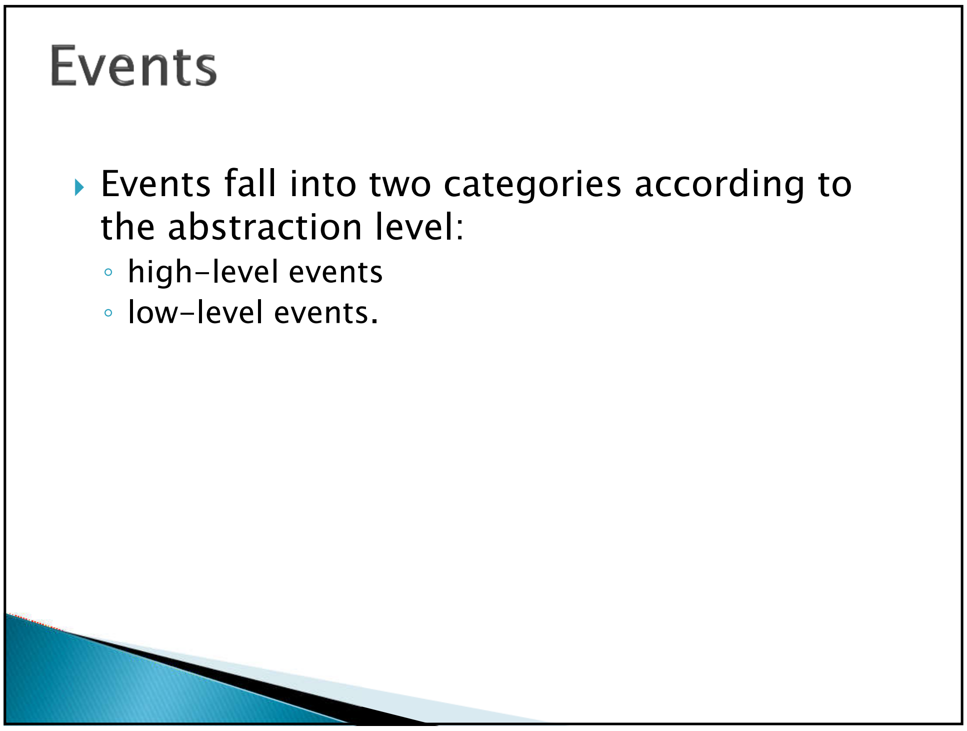


Event Type	Listener Interface	Listener Methods
ActonEvent	ActionListener	actionPerformed(ActionEvent)
ItemEvent	ItemListener	itemStateChanged(MouseEvent)
MouseEvent	MouseListener	mousePressed(MouseEvent)
		mouseReleased(MouseEvent)
		mouseEntered(MouseEvent)
		mouseExited(MouseEvent)
		mouseClicked(MouseEvent)
	MouseMotionListener	mouseDragged(MouseEvent)
		mouseMoved(MouseEvent)
KeyEvent	KeyListener	keyPressed(KeyEvent)
		keyReleased(KeyEvent)
		keyTyped(KeyEvent)
WindowEvent	WindowListener	windowClosing(WindowEvent)
		windowOpened(WindowEvent)
		windowIconified(WindowEvent)
		windowDeiconified(WindowEvent)
		windowClosed(WindowEvent)
		windowActivated(WindowEvent)
		windowDeactivated(WindowEvent)



Events

ContainerEvent	ContainerListener	componentAdded(ContainerEvent)
		componentRemoved(ContainerEvent)
ComponentEvent	componentListener	componentMoved(ComponentEvent)
		componentHidden(ComponentEvent)
		componentResized(ComponentEvent)
		componentShow(ComponentEvent)
FocusEvent	FocusListener	focusGained(FocusEvent)
		focusLost(FocusEvent)
AdjustmentEvent	AdjustmentListener	adjustmentValueChanged(AdjustmentEvent)

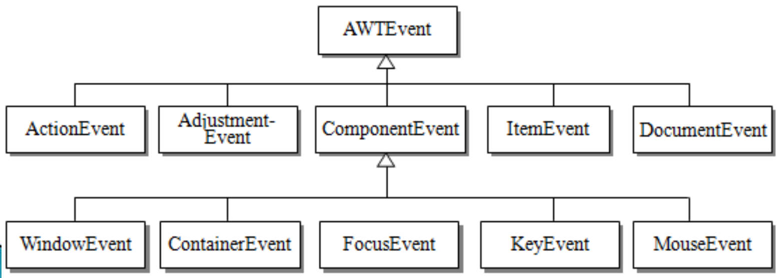


Events

- Events fall into two categories according to the abstraction level:
  - high-level events
  - low-level events.

Events

- The high level events are classified as
  - action events,
  - adjustment events,
  - item events,
  - document events.



Events

- Action events appear when component-sensitive action has taken place, for example, clicking a button, pressing Enter key in a JTextField.
- Adjustment events are only specific for a scrollbar when the scroll box has been moved.
- Item events happen when the user selects has selected a checkbox or list item.
- Document events come out when content in a JTextComponent has changed.

Events

- The low level events are classified into six kinds:
  - window events
  - container events
  - component events
  - focus events
  - key events
  - mouse events.

Events

- Window events occur when a window has been opened, closed, iconified, de-iconified, activated, or deactivated;
- Container events take place when a component has been added to or removed from a container;
- Component events emerge when a component has been hidden, shown, resized, or moved;
- Focus events are triggered when a component has gained or lost focus;
- Key events arise when keyboard key has been pressed or released;
- Mouse events are generated when a mouse button has been pressed or released, or mouse has been moved.

Events

- There are three different ways to implement a listener.
- For example, we create a button that closes the frame when the button is clicked, and use actionPerformed() declared in the ActionListener interface to make the system call System.exit(0) to close the frame.

Listener Class

```
class MyActionListner implements ActionListener {

    public void actionPerformed(ActionEvent e) {//handler
    System.out.println(“exiting");
    System.exit(0);
    }
}
```

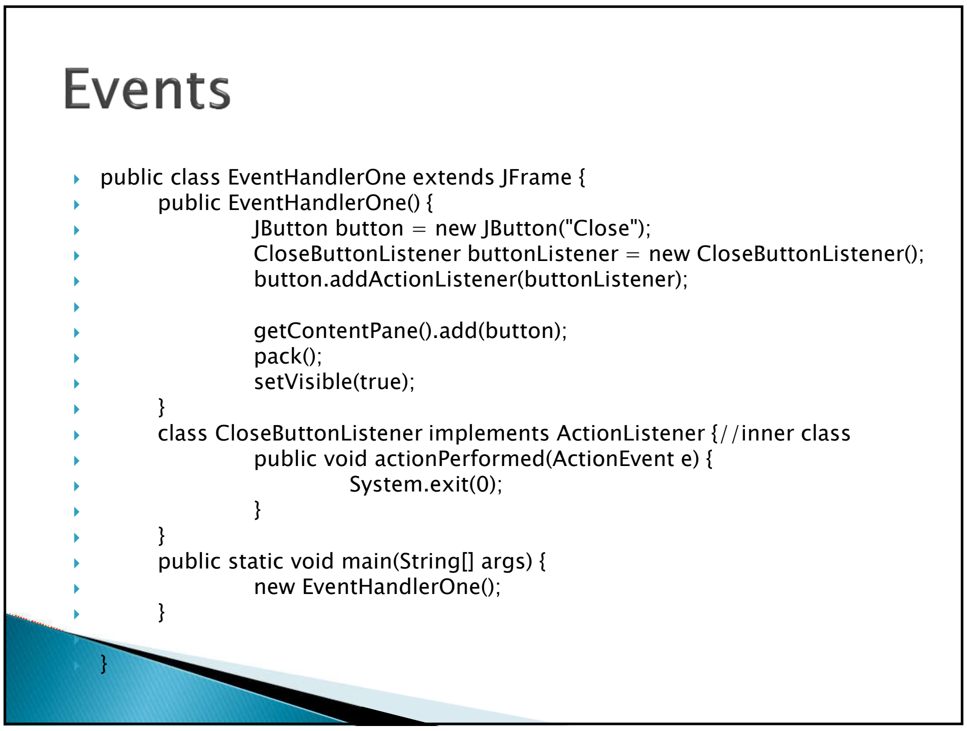
88

GUI Application

```
public class EventTest extends JFrame{
    EventTest(){
        super();
        setTitle(“Event handling");
        setSize(500,500);
        JButton btn = new JButton("Exit");
        MyActionListner listner
                = new MyActionListner();
        btn.addActionListener(listner);
        getContentPane().add(btn);
}

    public static void main(String [] args){
        EventTest fr=new EventTest();
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        fr.setVisible(true);
    }
}
```

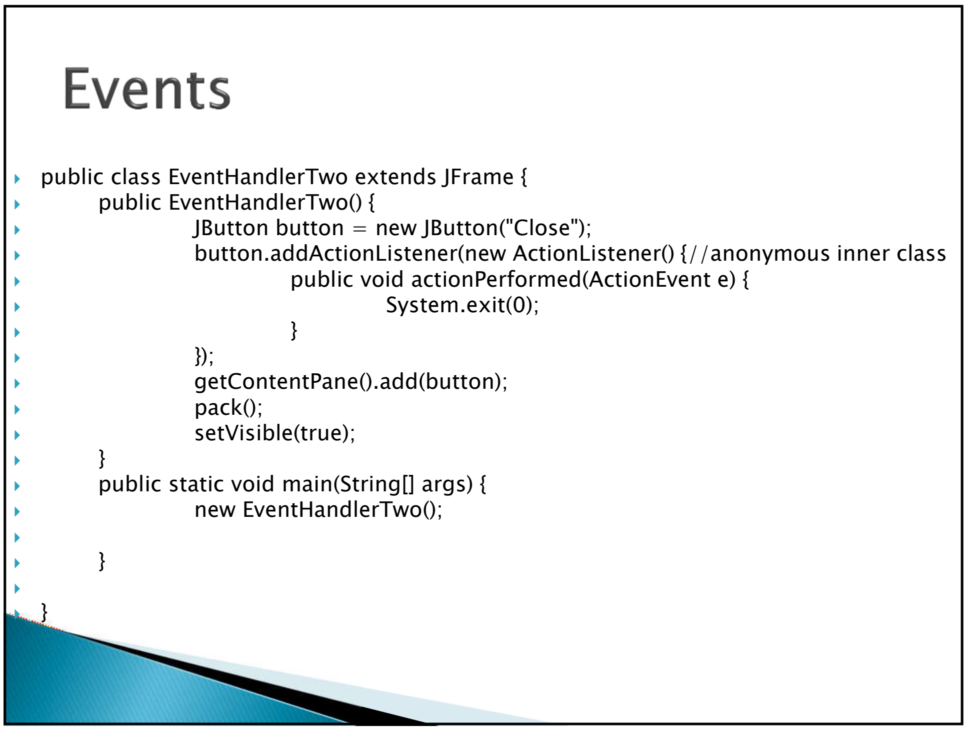
Events

- Inner class
  - An inner class, or nested class, is a class defined as a member of another class.



Events

```
public class EventHandlerOne extends JFrame {
        public EventHandlerOne() {
                JButton button = new JButton("Close");
                CloseButtonListener buttonListener = new CloseButtonListener();
                button.addActionListener(buttonListener);

                getContentPane().add(button);
                pack();
                setVisible(true);
        }
        class CloseButtonListener implements ActionListener {//inner class
                public void actionPerformed(ActionEvent e) {
                        System.exit(0);
                }
        }
        public static void main(String[] args) {
                new EventHandlerOne();
        }

}
```



Events

```
public class EventHandlerTwo extends JFrame {
        public EventHandlerTwo() {
                JButton button = new JButton("Close");
                button.addActionListener(new ActionListener() {//anonymous inner class
                        public void actionPerformed(ActionEvent e) {
                                System.exit(0);
                        }
                });
                getContentPane().add(button);
                pack();
                setVisible(true);
        }
        public static void main(String[] args) {
                new EventHandlerTwo();

        }

}
```

Events

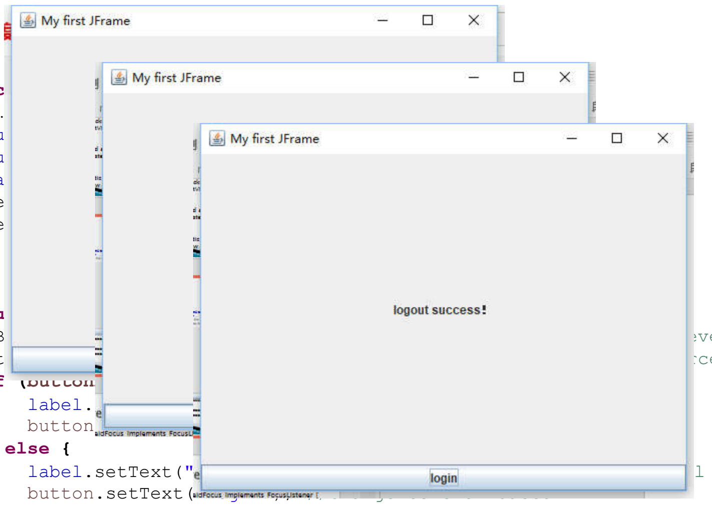
- The third way is to implement the ActionListener interface by the top-level container.
- The actionPerformed() method is provided to satisfy the interface implementation as a member method in the top-level container.
- Next, it uses the addActionListener() method to register the listener for the button.

Events

```
public class EventHandlerThree extends JFrame implements
ActionListener {
        public EventHandlerThree() {
                JButton button = new JButton("Close");
                button.addActionListener(this);

                getContentPane().add(button);
                pack();
                setVisible(true);
        }

        public void actionPerformed(ActionEvent e) {
                System.exit(0);
        }

        public static void main(String[] args) {
                new EventHandlerThree();
        }
}
```

Events

```
final
final

public
    ....
    su
    su
    la
    ge
    ge

}
class
    pu
    JB                                                              event
    St                                                              ce
    if (button
        label.
        button
    } else {
        label.setText("e                                            l
        button.setText(
    }
}
```



Focus Events

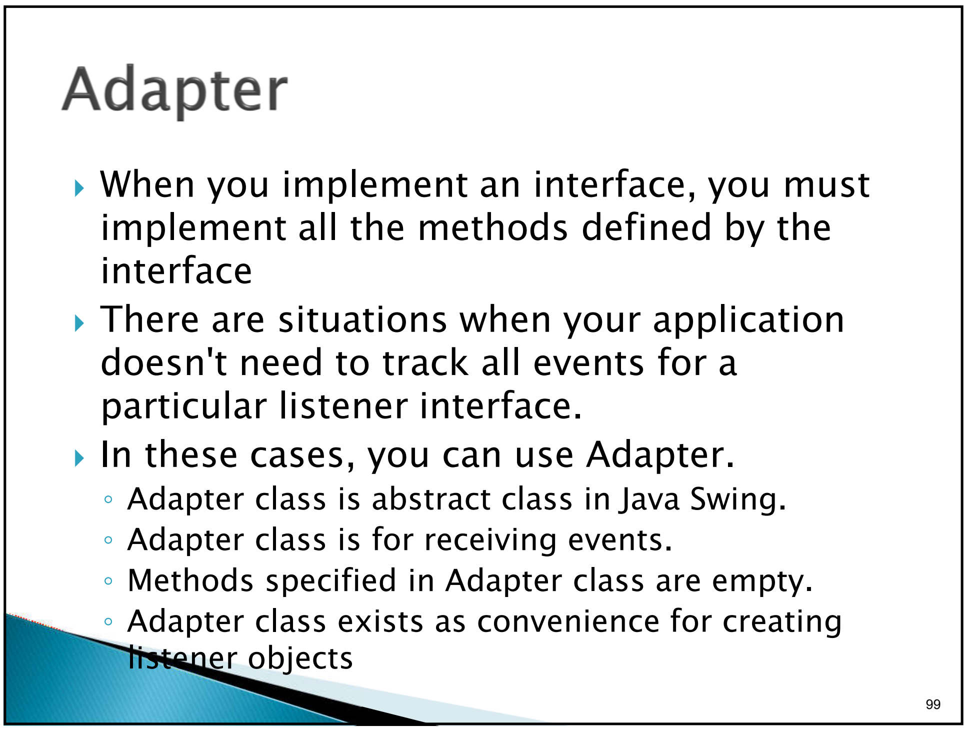
- class TextFieldFocus implements FocusListener {
-     public void focusGained(FocusEvent e) {
-         textField.setText("");
-     }
-     public void focusLost(FocusEvent e) {
-         textField.setText("2015-4-12");
-     }
- }

Mouse Events

MouseListener

```
public interface MouseListener extends EventListener {
    public void mouseEntered(MouseEvent e);
    public void mousePressed(MouseEvent e);
    public void mouseReleased(MouseEvent e);
    public void mouseClicked(MouseEvent e);
    public void mouseExited(MouseEvent e);}
```

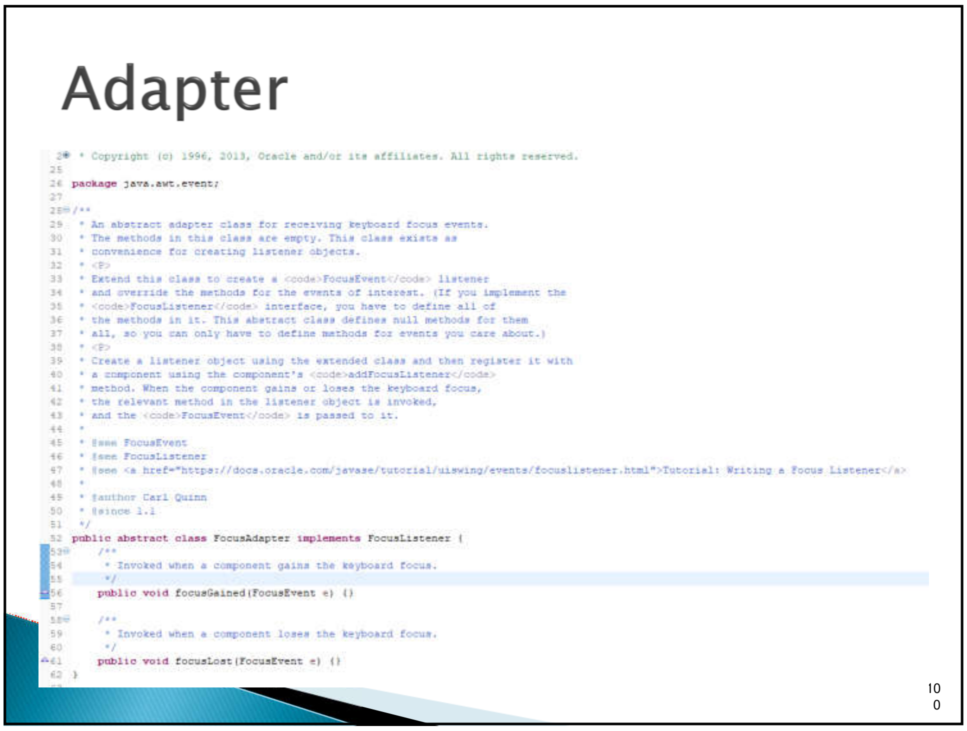
```
label.addMouseListener(new MouseListener() {
        public void mouseEntered(MouseEvent e) {
                System.out.println(“entring");
        }
        public void mousePressed(MouseEvent e) {
                System.out.println(“pressed");
                int i = e.getButton();
                if (i == MouseEvent.BUTTON1)
                        System.out.println("left key");
                if (i == MouseEvent.BUTTON2)
                        System.out.println("wheel");
                if (i == MouseEvent.BUTTON3)
                        System.out.println("right key");
        }
                    ……
});
```



Adapter

- When you implement an interface, you must implement all the methods defined by the interface
- There are situations when your application doesn't need to track all events for a particular listener interface.
- In these cases, you can use Adapter.
  - Adapter class is abstract class in Java Swing.
  - Adapter class is for receiving events.
  - Methods specified in Adapter class are empty.
  - Adapter class exists as convenience for creating listener objects

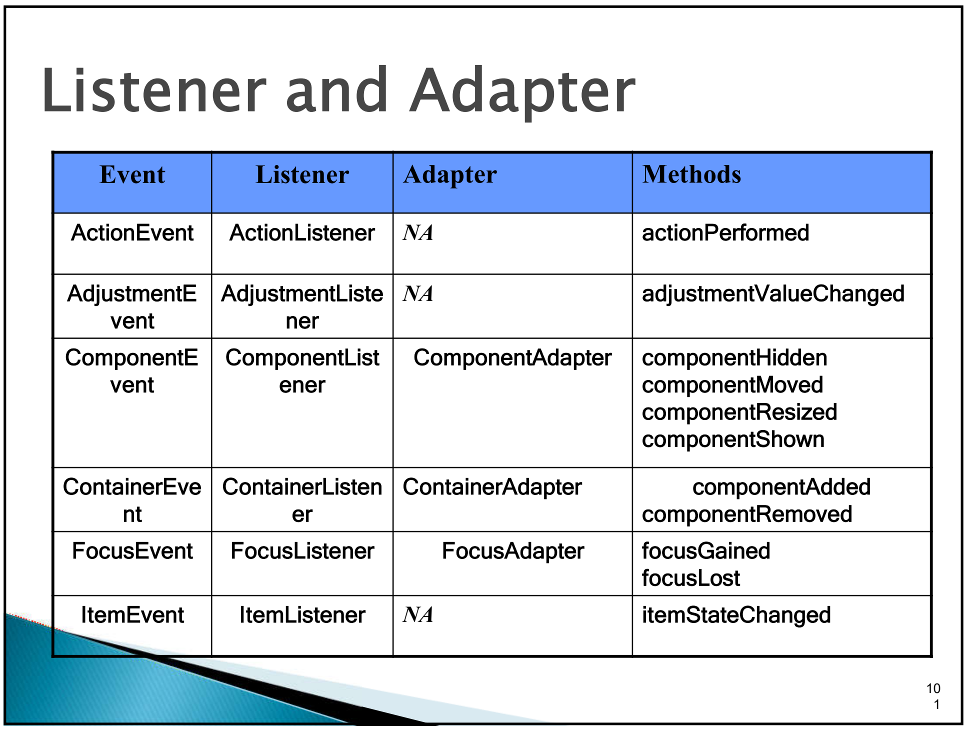
99



Adapter

```java
 * Copyright (c) 1996, 2013, Oracle and/or its affiliates. All rights reserved.

package java.awt.event;

/**
 * An abstract adapter class for receiving keyboard focus events.
 * The methods in this class are empty. This class exists as
 * convenience for creating listener objects.
 * <P>
 * Extend this class to create a <code>FocusEvent</code> listener
 * and override the methods for the events of interest. (If you implement the
 * <code>FocusListener</code> interface, you have to define all of
 * the methods in it. This abstract class defines null methods for them
 * all, so you can only have to define methods for events you care about.)
 * <P>
 * Create a listener object using the extended class and then register it with
 * a component using the component's <code>addFocusListener</code>
 * method. When the component gains or loses the keyboard focus,
 * the relevant method in the listener object is invoked,
 * and the <code>FocusEvent</code> is passed to it.
 *
 * @see FocusEvent
 * @see FocusListener
 * @see <a href="https://docs.oracle.com/javase/tutorial/uiswing/events/focuslistener.html">Tutorial: Writing a Focus Listener</a>
 *
 * @author Carl Quinn
 * @since 1.1
 */
public abstract class FocusAdapter implements FocusListener {
    /**
     * Invoked when a component gains the keyboard focus.
     */
    public void focusGained(FocusEvent e) {}

    /**
     * Invoked when a component loses the keyboard focus.
     */
    public void focusLost(FocusEvent e) {}
}
```

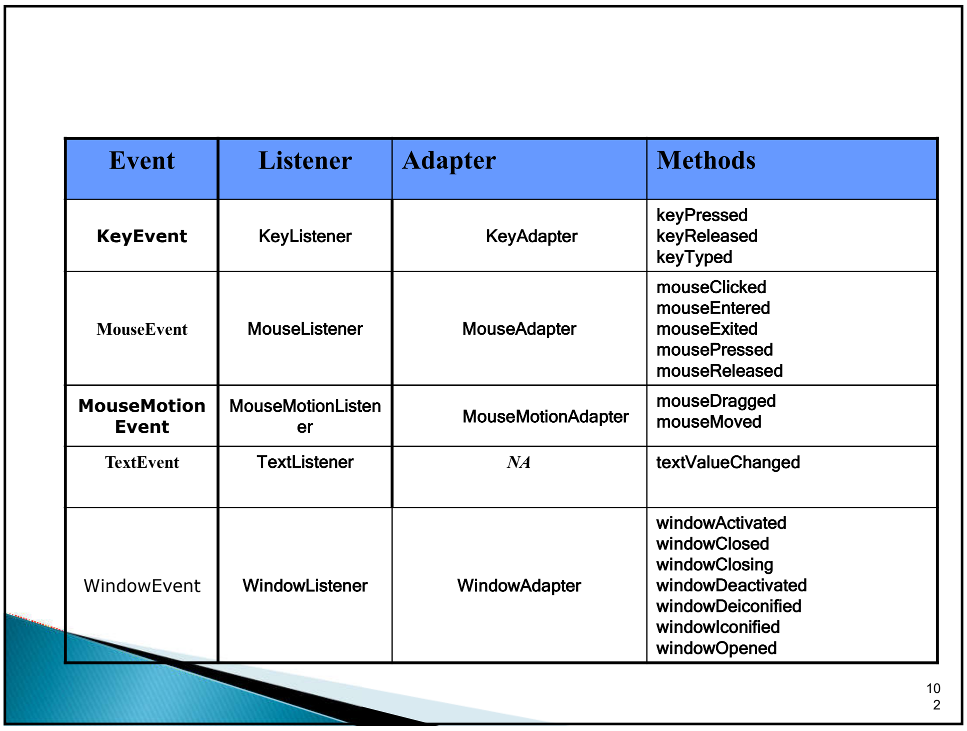
10
0



Listener and Adapter

Event	Listener	Adapter	Methods
ActionEvent	ActionListener	*NA*	actionPerformed
AdjustmentEvent	AdjustmentListener	*NA*	adjustmentValueChanged
ComponentEvent	ComponentListener	ComponentAdapter	componentHidden componentMoved componentResized componentShown
ContainerEvent	ContainerListener	ContainerAdapter	componentAdded componentRemoved
FocusEvent	FocusListener	FocusAdapter	focusGained focusLost
ItemEvent	ItemListener	*NA*	itemStateChanged

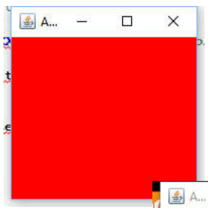
101

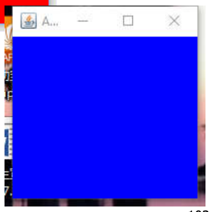


Event	Listener	Adapter	Methods
KeyEvent	KeyListener	KeyAdapter	keyPressed keyReleased keyTyped
MouseEvent	MouseListener	MouseAdapter	mouseClicked mouseEntered mouseExited mousePressed mouseReleased
MouseMotion Event	MouseMotionListener	MouseMotionAdapter	mouseDragged mouseMoved
TextEvent	TextListener	*NA*	textValueChanged
WindowEvent	WindowListener	WindowAdapter	windowActivated windowClosed windowClosing windowDeactivated windowDeiconified windowIconified windowOpened

102

```
public class AdapterFrame extends JFrame {


    public static void main(String args[]) {
        AdapterFrame frame = new AdapterFrame();
        frame.setVisible(true); // set frame visible
    }
    public AdapterFrame() {
        super();
        setTitle("Adapter JFrame"); // set title
        setBounds(100, 100, 200, 200); // set position and size
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);//set close opration

        this.addWindowListener(new WinListener(this));

    }
    class WinListener extends WindowAdapter {
        AdapterFrame frame;

    WinListener(AdapterFrame frame){
        this.frame = frame;
    }

    public void windowActivated(WindowEvent e){
        frame.getContentPane().setBackground(Color.RED);
    }
    public void windowDeactivated(WindowEvent e){
        frame.getContentPane().setBackground(Color.BLUE);
    }
    }

}
```





103