Graphical User Interfaces

Introduction

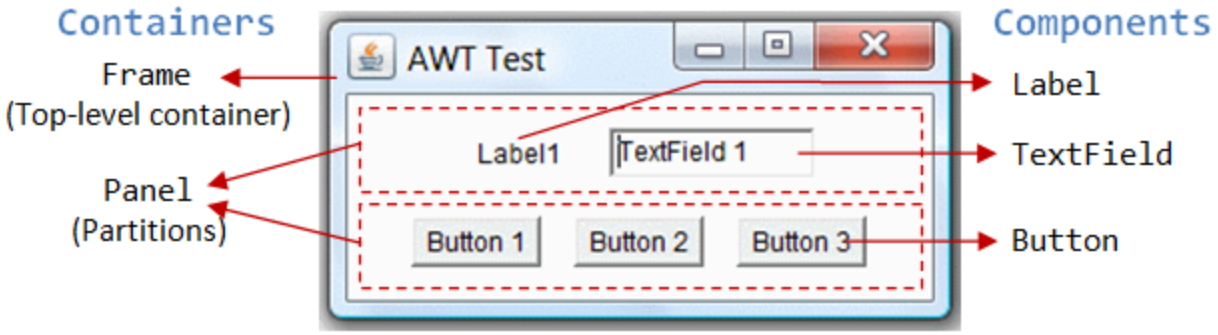
- Graphical user interface (GUI)
- Java 1.0
  - *AWT* (Abstract Windowing Toolkit)
  - Most of the AWT components have become obsolete and should be replaced by newer Swing components
- Java 2 (JDK 1.2)
  - Java Foundation Classes (JFC)
  - *Swing* - part of JFC
  - Java 2D, Accessibility, Internationalization, and Pluggable Look-and-Feel Support APIs
- Others have also provided Graphics APIs that work with Java
  - Eclipse's Standard Widget Toolkit (SWT) (used in Eclipse)
  - Google Web Toolkit (GWT) (used in Android)
  - etc.

AWT Packages

- **AWT is huge! It consists of 12 packages**
  - **Only 2 packages are commonly-used**
- **java.awt package contains the core AWT graphics classes**
  - **GUI Component classes: *Button*, *TextField*, *Label*, etc.**
  - **GUI Container classes: *Frame*, *Panel*, *Dialog*, *ScrollPane*, etc.**
  - **Layout managers: *FlowLayout*, *BorderLayout*, *GridLayout*, etc.**
  - **Custom graphics classes: *Graphics*, *Color*, *Font*, etc.**
- **java.awt.event package supports event handling**
  - **Event classes: *ActionEvent*, *MouseEvent*, *KeyEvent* and *WindowEvent*, etc.**
  - **Event Listener Interfaces: *ActionListener*, *MouseListener*, *KeyListener*, *WindowListener*, etc.**
  - **Event Listener Adapter classes: *MouseAdapter*, *KeyAdapter*, *WindowAdapter*, etc.**

Containers and Components

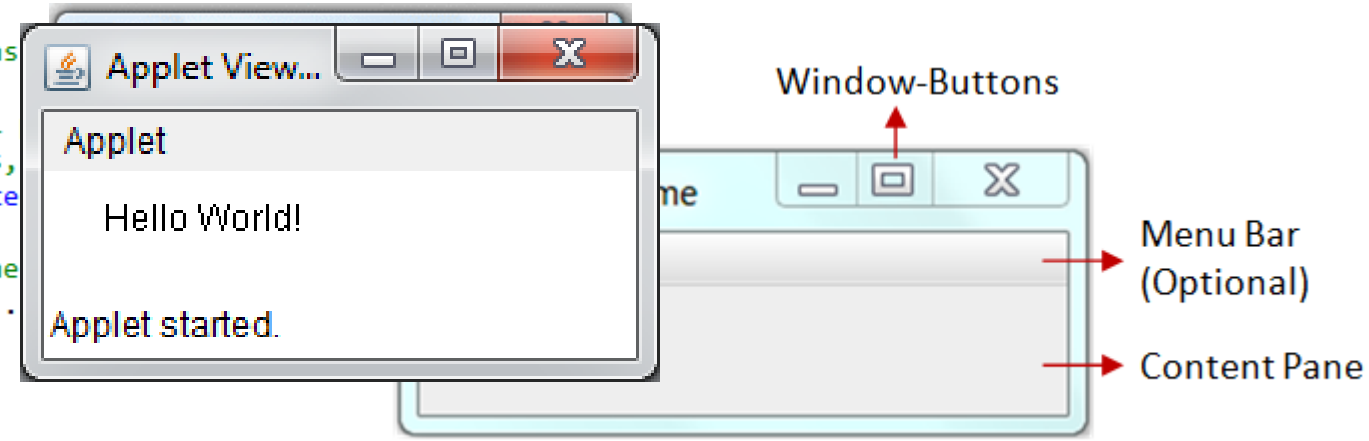
- Two types of GUI elements
- *Component*: Components are elementary GUI entities
  - Button, Label, TextField, etc.
- *Container*: Containers are used to hold components in a specific layout (such as flow or grid)
  - Containers: Frame, Panel and Applet
  - Layout: flow, grid, etc.
  - A container can also hold sub-containers



AWT Container Classes

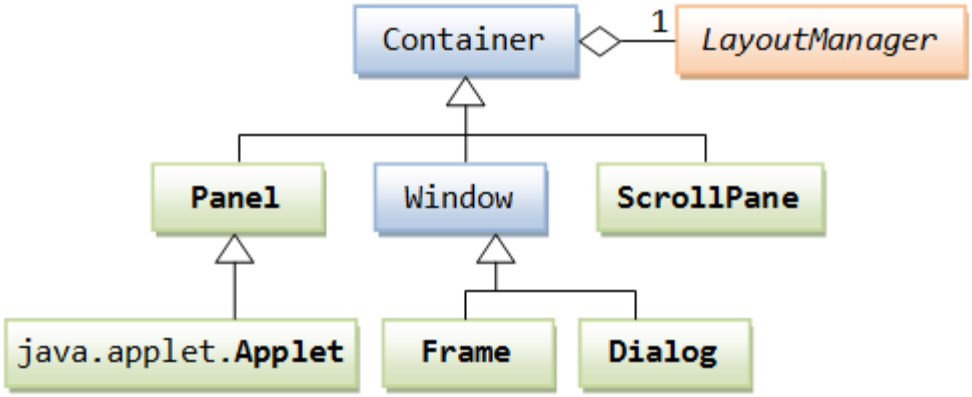
- ***Top-Level Containers*: *Frame*, *Dialog* and *Applet***
  - **A *Frame* provides the "main window" for the GUI application**
  - **An AWT *Dialog* is a "pop-up window" used for interacting with the users**
  - **An AWT *Applet* is the top-level container for an applet**

```
1  // Using Frame class in package java.awt
2  import java.awt.Frame;
3
4  // A GUI program is written as
5  // - the top-level container
6  // This subclass inherits all
7  // e.g., title, icon, buttons,
8  public class MyGUIProgram exte
9
10    // Constructor to setup the
11    public MyGUIProgram() { ...
12
13    // Other methods
14    ......
15    ......
16
17    // The entry main() method
18    public static void main(String[] args) {
19       // Invoke the constructor (to setup the GUI)
20       // by allocating an instance
21       new MyGUIProgram();
22    }
23 }
```



AWT Container Classes (Cont.)

- **Secondary Containers: *Panel* and *ScrollPane***
  - ***Panel*: a rectangular box under a higher-level container, used to layout a set of related GUI components in pattern such as grid or flow**
  - ***ScrollPane*: provides automatic horizontal and/or vertical scrolling for a single child component**
  - **Others**
- **Hierarchy of the AWT Container Classes**

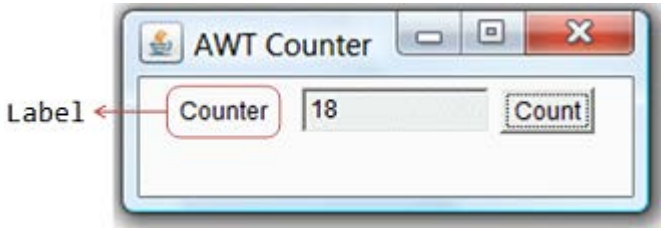


AWT Component Classes

- AWT provides many ready-made and reusable GUI components
  - Button, TextField, Label, Checkbox, CheckboxGroup (radio buttons), List, and Choice

AWT GUI Component: java.awt.Label

- A java.awt.Label provides a text description message



- **Constructor**
  - *public Label(String strLabel, int alignment)*;
    - Alignment: Label.LEFT, Label.RIGHT, and Label.CENTER
  - *public Label(String strLabel)*;
    - Text string in default of left-aligned
  - *public Label()*;
    - Set the label text via the setText() method later
- **Public Methods**
  - *public String getText()*;
  - *public void setText(String strLabel)*;
  - *public int getAlignment()*;
  - *public void setAlignment(int alignment)*;

Constructing a Component and Adding the Component into a Container

- **Three steps are necessary to create and place a GUI component:**
  - **Declare the component with an identifier (name)**
  - **Construct the component by invoking an appropriate constructor via the new operator**
  - **Identify the container (such as *Frame* or *Panel*) designed to hold this component**
- **Example**

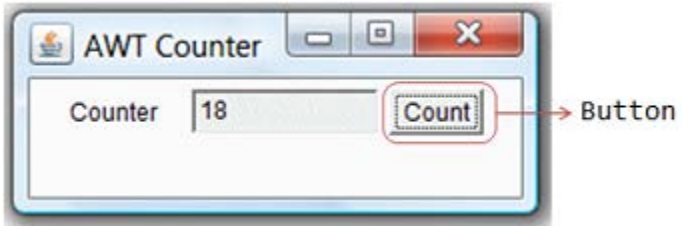
```
1 Label lblInput;                       // Declare an Label instance called lblInput
2 lblInput = new Label("Enter ID");     // Construct by invoking a constructor via the new operator
3 add(lblInput);                        // this.add(lblInput) - "this" is typically a subclass of Frame
4 lblInput.setText("Enter password"); // Modify the Label's text string
5 lblInput.getText();                   // Retrieve the Label's text string
```

- **An Anonymous Instance**

```
1 // Allocate an anonymous Label instance. "this" container adds the instance into itself.
2 // You CANNOT reference an anonymous instance to carry out further operations.
3 add(new Label("Enter Name: ", Label.RIGHT));
4
5 // Same as
6 Label lblXxx = new Label("Enter Name: ", Label.RIGHT)); // lblXxx assigned by compiler
7 add(lblXxx);
```

AWT GUI Component: java.awt.Button

- A java.awt.Button is a GUI component that triggers a certain programmed action upon clicking



- **Constructor**
  - *public Button(String buttonLabel)*;
  - *public Button()*;
- **Public Methods**
  - *public String getLabel()*;
  - *public void setLabel(String buttonLabel)*;
  - *public void setEnable(boolean enable)*;
- **Note:** the latest Swing's JButton
  - Replace getLabel()/setLabel() with getText()/setText() to be consistent with all the components

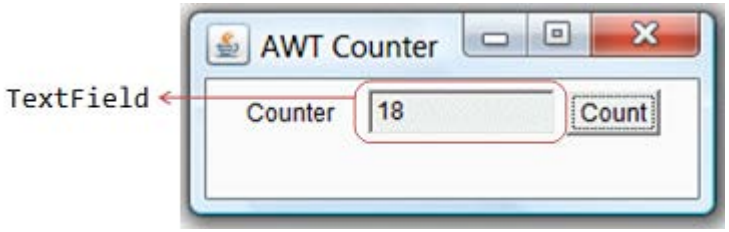
AWT GUI Component: java.awt.Button (Cont.)

- **Event**
  - **Clicking a button fires a so-called ActionEvent and triggers a certain programmed action**
- **Example**

```
1  Button btnColor = new Button("Red"); // Declare and allocate a Button instance called btnColor
2  add(btnColor);                       // "this" Container adds the Button
3  ...
4  btnColor.setLabel("green");          // Change the button's label
5  btnColor.getLabel();                 // Read the button's label
6  ...
7  add(Button("Blue"));    // Create an anonymous Button. It CANNOT be referenced later
```

AWT GUI Component: java.awt.TextField

- **A java.awt.TextField is *single-line text box* for users to enter texts**



- **Constructor**
  - ***public TextField(String strInitialText, int columns)***;
  - ***public TextField(String strInitialText)***;
  - ***public TextField(int columns)***;
- **Public Methods**
  - ***public String getText()***;
  - ***public void setText(String strText)***;
  - ***public void setEnable(boolean editable)***;

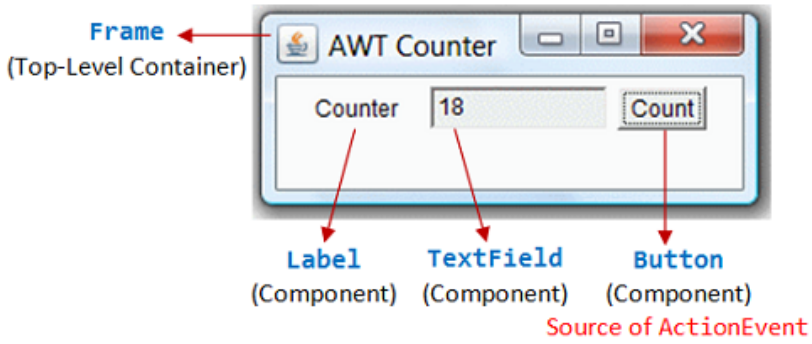
AWT GUI Component: java.awt.TextField (Cont.)

- **Event**
  - **Hitting the "ENTER" key on a TextField fires a ActionEvent, and triggers a certain programmed action**
- **Example**

```
1  TextField tfInput = new TextField(30); // Declare and allocate an TextField instance called tfInput
2  add(tfInput);                          // "this" Container adds the TextField
3  TextField tfResult = new TextField();  // Declare and allocate an TextField instance called tfResult
4  tfResult.setEditable(false) ;          // Set to read-only
5  add(tfResult);                         // "this" Container adds the TextField
6  ......
7  // Read an int from TextField "tfInput", square it, and display on "tfResult".
8  // getText() returns a String, need to convert to int
9  int number = Integer.parseInt(tfInput.getText());
10 number *= number;
11 // setText() requires a String, need to convert the int number to String.
12 tfResult.setText(number + "");
```

Example: AWTCounter

```java
import java.awt.*;        // Using AWT container and component classes
import java.awt.event.*;  // Using AWT event classes and listener interfaces

// An AWT program inherits from the top-level container java.awt.Frame
public class AWTCounter extends Frame implements ActionListener {
   private Label lblCount;    // Declare component Label
   private TextField tfCount; // Declare component TextField
   private Button btnCount;   // Declare component Button
   private int count = 0;     // Counter's value
   /** Constructor to setup GUI components and event handling */
   public AWTCounter () {
      setLayout(new FlowLayout());
      // "super" Frame sets its layout to FlowLayout, which arranges the components
      //  from left-to-right, and flow to next row from top-to-bottom.
      lblCount = new Label("Counter");  // construct Label
      add(lblCount);                    // "super" Frame adds Label
      tfCount = new TextField("0", 10); // construct TextField
      tfCount.setEditable(false);       // set to read-only
      add(tfCount);                     // "super" Frame adds tfCount
      btnCount = new Button("Count");   // construct Button
      add(btnCount);                    // "super" Frame adds Button
      btnCount.addActionListener(this);
      // Clicking Button source fires ActionEvent
      // btnCount registers this instance as ActionEvent listener
      setTitle("AWT Counter");  // "super" Frame sets title
      setSize(250, 100);        // "super" Frame sets initial window size
      setVisible(true);         // "super" Frame shows
   }
   /** The entry main() method */
   public static void main(String[] args) {
      // Invoke the constructor to setup the GUI, by allocating an instance
      AWTCounter app = new AWTCounter();
   }
   /** ActionEvent handler - Called back upon button-click. */
   @Override
   public void actionPerformed(ActionEvent evt) {
      ++count; // increase the counter value
      // Display the counter value on the TextField tfCount
      tfCount.setText(count + ""); // convert int to String
   }
}
```

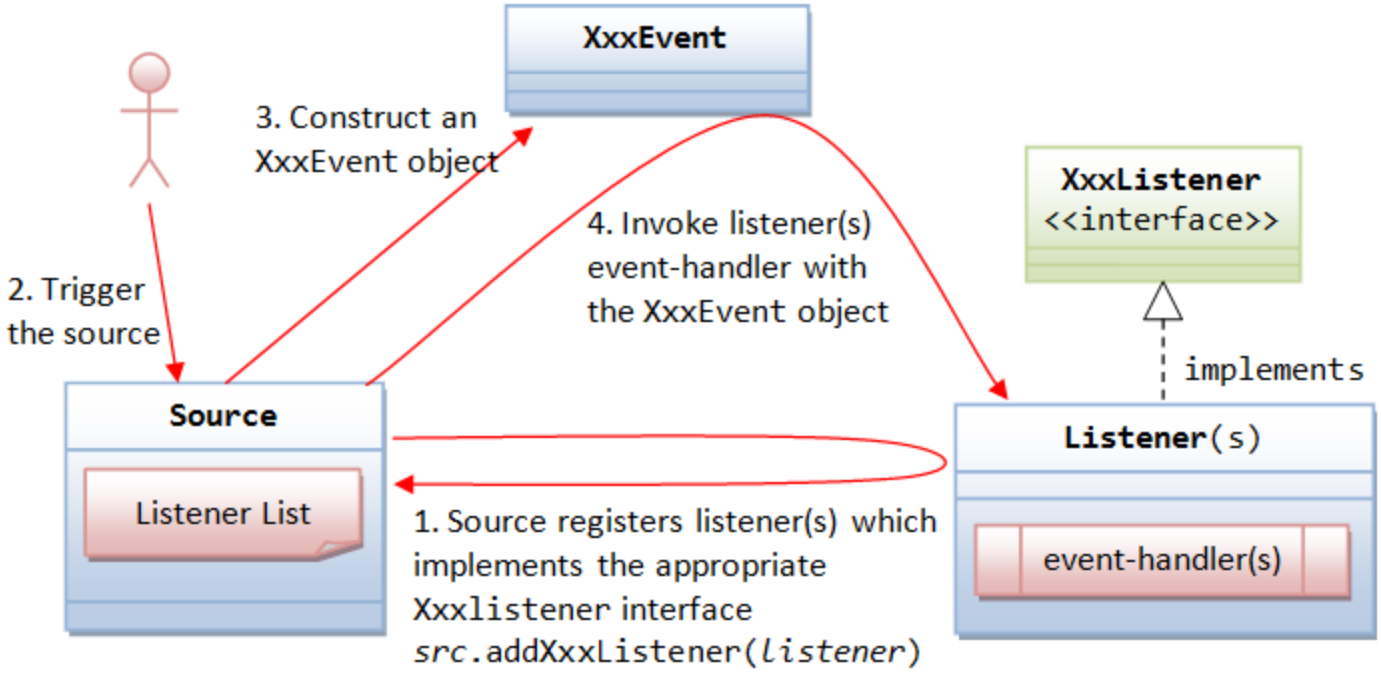


AWT Event-Handling

- **"Event-Driven" programming model for event-handling**
  - **The AWT's event-handling classes are kept in package java.awt.event**
- **Event-driven programming**
  - **When an event has been fired in response to an user input (such as clicking a mouse button or hitting the ENTER key), a piece of event-handling codes is executed**
  - **Unlike the procedural model, where codes are executed in a sequential manner**
- **Three objects are involved in the event-handling: *a source*, *listener(s)* and *an event object***
  - **The listener(s) "subscribes" to a source's event**
  - **The source "publishes" the event to all its subscribers upon activation**
  - **Known as *subscribe-publish* or *observable-observer* design pattern**

AWT Event-Handling (Cont.)

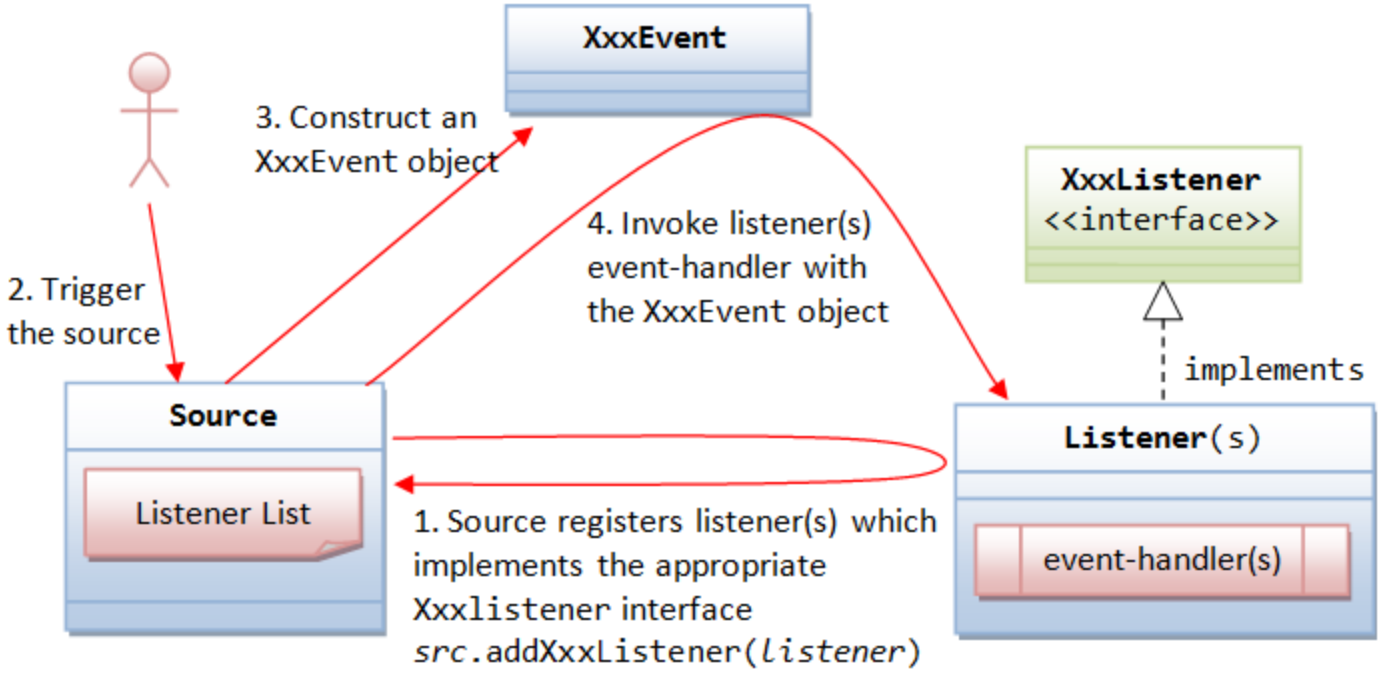
- 1.The *source* object registers its *listener(s)* for a certain type of *event*
  - How the *source* and *listener* understand each other?
    - The answer is via an agreed-upon interface
  - For example, if a *source* is capable of firing an event called XxxEvent (e.g., MouseEvent) involving various operational modes (e.g., mouse-clicked)



AWT Event-Handling (Cont.)

- **Declare an interface called XxxListener (e.g., MouseListener) containing the names of the handler methods**

```
1  // A MouseListener interface, which declares the signature of the handlers
2  //  for the various operational modes.
3  public interface MouseListener {
4     public void mousePressed(MouseEvent evt);  // Called back upon mouse-button pressed
5     public void mouseReleased(MouseEvent evt); // Called back upon mouse-button released
6     public void mouseClicked(MouseEvent evt);  // Called back upon mouse-button clicked (pressed and released)
7     public void mouseEntered(MouseEvent evt);  // Called back when mouse pointer entered the component
8     public void mouseExited(MouseEvent evt);   // Called back when mouse pointer exited the component
9  }
```



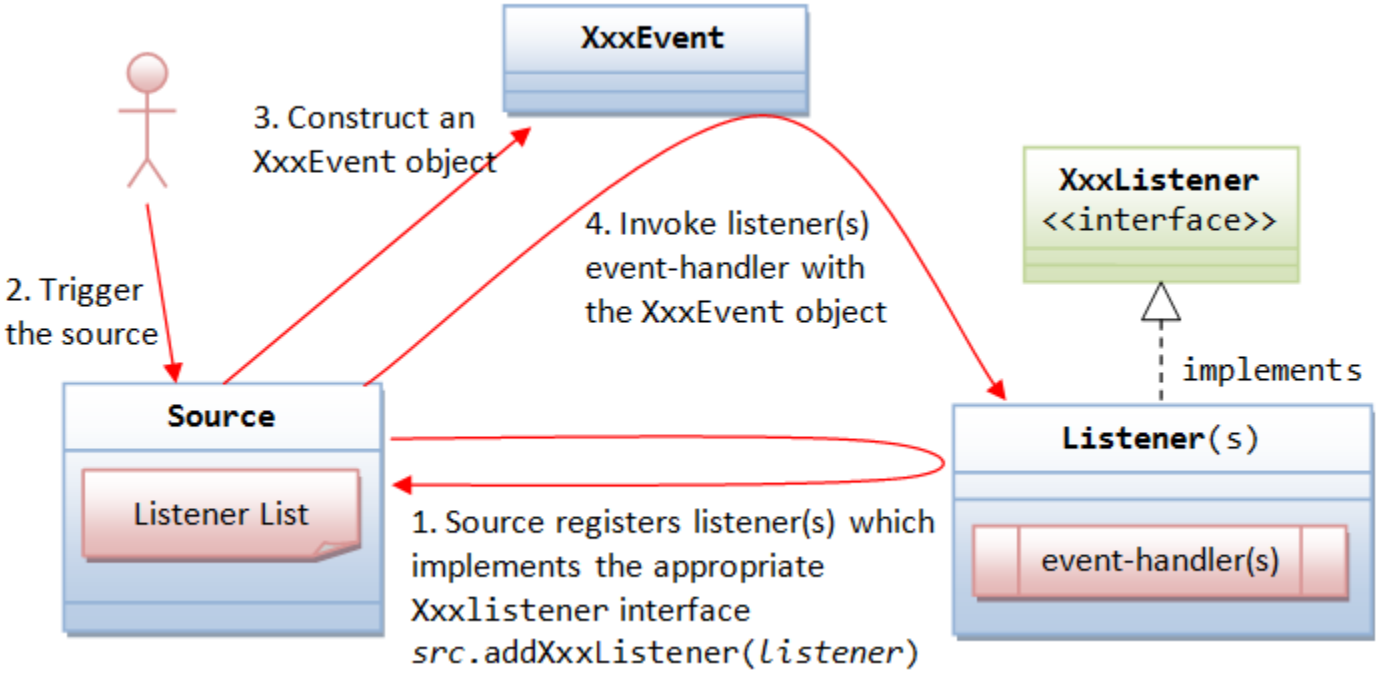
AWT Event-Handling (Cont.)

- **All the listeners interested in the XxxEvent (e.g., MouseEvent) must implement the XxxListener (e.g., MouseListener) interface**

```
1  // An example of MouseListener,  which provides implementation to the handler methods
2  class MyMouseListener implement MouseListener {
3     @Override
4     public void mousePressed(MouseEvent e)  {
5        System.out.println("Mouse-button pressed!");
6     }
7     @Override
8     public void mouseReleased(MouseEvent e) {
9        System.out.println("Mouse-button released!");
10    }
11    @Override
12    public void mouseClicked(MouseEvent e)  {
13       System.out.println("Mouse-button clicked (pressed and released)!");
14    }
15    @Override
16    public void mouseEntered(MouseEvent e)  {
17       System.out.println("Mouse-pointer entered the source component!");
18    }
19    @Override
20    public void mouseExited(MouseEvent e)   {
21       System.out.println("Mouse exited-pointer the source component!");
22    }
23 }
```

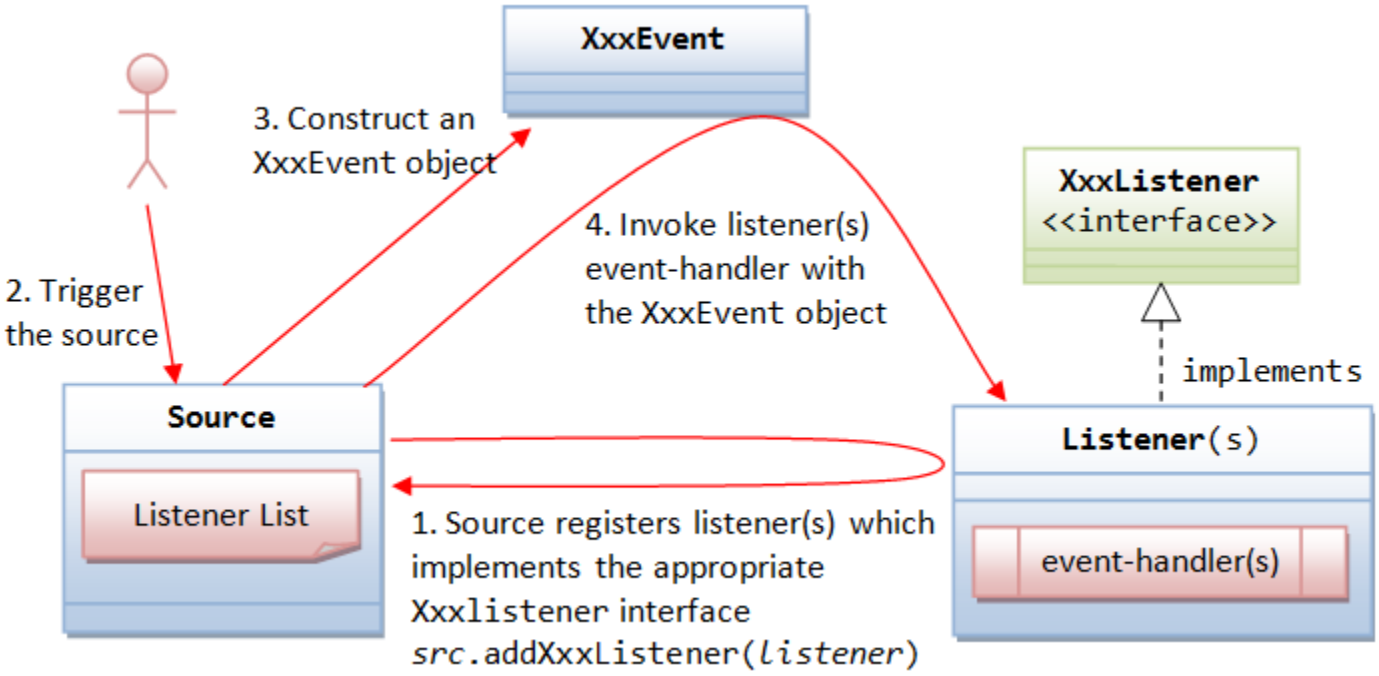
AWT Event-Handling (Cont.)

- In the source, a list of listener object(s), and define two methods
  - addXxxListener() and removeXxxListener() to add and remove a listener from this list
- The signature of the methods are:
  - public void addXxxListener(XxxListener l);
  - public void removeXxxListener(XxxListener l);



AWT Event-Handling (Cont.)

- 2.The source is triggered by a user
- 3.The source create an XxxEvent object, which encapsulates the necessary information about the activation
- 4.Finally, for each of the listeners in the listener list, the source invokes the appropriate handler on the listener(s), which provides the programmed response



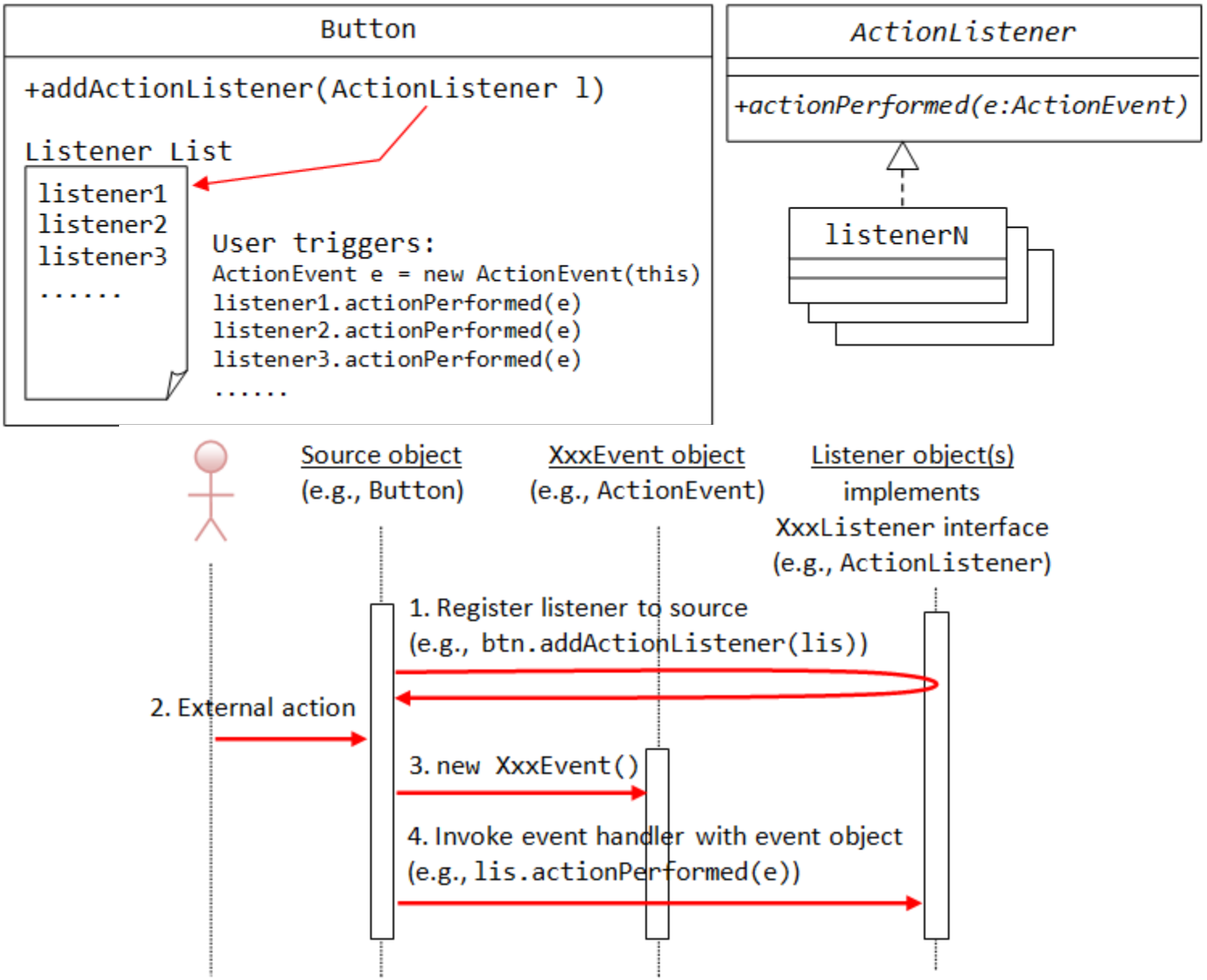
Revisit Example: AWTCounter

```java
import java.awt.*;        // Using AWT container and component classes
import java.awt.event.*;  // Using AWT event classes and listener interfaces

// An AWT program inherits from the top-level container java.awt.Frame
public class AWTCounter extends Frame implements ActionListener {
   private Label lblCount;    // Declare component Label
   private TextField tfCount; // Declare component TextField
   private Button btnCount;   // Declare component Button
   private int count = 0;     // Counter's value
   /** Constructor to setup GUI components and event handling */
   public AWTCounter () {
      setLayout(new FlowLayout());
      // "super" Frame sets its layout to FlowLayout, which arranges the components
      //  from left-to-right, and flow to next row from top-to-bottom.
      lblCount = new Label("Counter");  // construct Label
      add(lblCount);                    // "super" Frame adds Label
      tfCount = new TextField("0", 10); // construct TextField
      tfCount.setEditable(false);       // set to read-only
      add(tfCount);                     // "super" Frame adds tfCount
      btnCount = new Button("Count");   // construct Button
      add(btnCount);                    // "super" Frame adds Button
      btnCount.addActionListener(this);
      // Clicking Button source fires ActionEvent
      // btnCount registers this instance as ActionEvent listener
      setTitle("AWT Counter");  // "super" Frame sets title
      setSize(250, 100);        // "super" Frame sets initial window size
      setVisible(true);         // "super" Frame shows
   }
   /** The entry main() method */
   public static void main(String[] args) {
      // Invoke the constructor to setup the GUI, by allocating an instance
      AWTCounter app = new AWTCounter();
   }
   /** ActionEvent handler - Called back upon button-click. */
   @Override
   public void actionPerformed(ActionEvent evt) {
      ++count; // increase the counter value
      // Display the counter value on the TextField tfCount
      tfCount.setText(count + ""); // convert int to String
   }
}
```

```java
public interface ActionListener {
   public void actionPerformed(ActionEvent e);
   // Called back upon button-click (on Button),
   // enter-key pressed (on TextField)
}
```

- Identify btnCount (***Button***) as the ***source*** object
- Clicking Button fires an ***ActionEvent*** to all its ActionEvent listener(s)
- The listener(s) is required to implement ***ActionListener*** interface

Revisit Example: AWTCounter (Cont.)



Revisit Example: AWTCounter

```java
import java.awt.*;        // Using AWT container and component classes
import java.awt.event.*;  // Using AWT event classes and listener interfaces

// An AWT program inherits from the top-level container java.awt.Frame
public class AWTCounter extends Frame implements ActionListener {
   private Label lblCount;    // Declare component Label
   private TextField tfCount; // Declare component TextField
   private Button btnCount;   // Declare component Button
   private int count = 0;     // Counter's value
   /** Constructor to setup GUI components and event handling */
   public AWTCounter () {
      setLayout(new FlowLayout());
      // "super" Frame sets its layout to FlowLayout, which arranges the components
      //  from left-to-right, and flow to next row from top-to-bottom.
      lblCount = new Label("Counter");  // construct Label
      add(lblCount);                    // "super" Frame adds Label
      tfCount = new TextField("0", 10); // construct TextField
      tfCount.setEditable(false);       // set to read-only
      add(tfCount);                     // "super" Frame adds tfCount
      btnCount = new Button("Count");   // construct Button
      add(btnCount);                    // "super" Frame adds Button
      btnCount.addActionListener(this);
      // Clicking Button source fires ActionEvent
      // btnCount registers this instance as ActionEvent listener
      setTitle("AWT Counter");  // "super" Frame sets title
      setSize(250, 100);        // "super" Frame sets initial window size
      setVisible(true);         // "super" Frame shows
   }
   /** The entry main() method */
   public static void main(String[] args) {
      // Invoke the constructor to setup the GUI, by allocating an instance
      AWTCounter app = new AWTCounter();
   }
   /** ActionEvent handler - Called back upon button-click. */
   @Override
   public void actionPerformed(ActionEvent evt) {
      ++count; // increase the counter value
      // Display the counter value on the TextField tfCount
      tfCount.setText(count + ""); // convert int to String
   }
}
```

```java
public interface ActionListener {
   public void actionPerformed(ActionEvent e);
   // Called back upon button-click (on Button),
   // enter-key pressed (on TextField)
}
```

- Identify btnCount (*Button*) as the *source* object
- Clicking Button fires an *ActionEvent* to all its ActionEvent listener(s)
- The listener(s) is required to implement *ActionListener* interface

An Inner Class as Event Listener

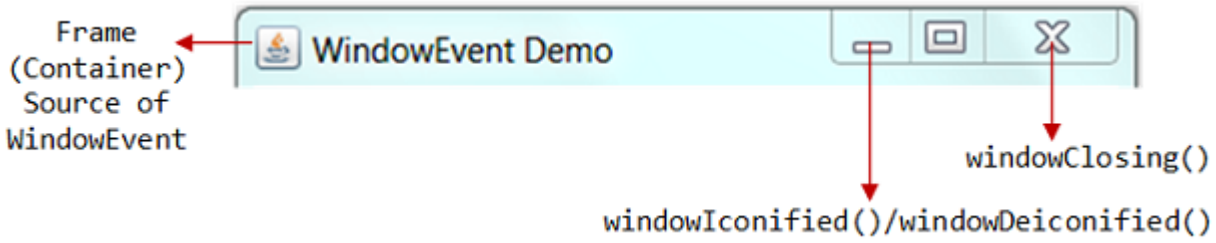
```
1  import java.awt.*;
2  import java.awt.event.*;
3  // An AWT GUI program inherits from the top-level container java.awt.Frame
4  public class AWTCounterNamedInnerClass extends Frame {
5     // This class is NOT a ActionListener, hence, it does not implement ActionListener
6     // The event-handler actionPerformed() needs to access these "private" variables
7     private TextField tfCount;
8     private int count = 0;
9     /** Constructor to setup the GUI */
10    public AWTCounterNamedInnerClass () {
11       setLayout(new FlowLayout());  // "super" Frame sets to FlowLayout
12       add(new Label("Counter"));    // anonymous instance of Label
13       tfCount = new TextField("0", 10);
14       tfCount.setEditable(false);   // read-only
15       add(tfCount);                 // "super" Frame adds tfCount
16
17       Button btnCount = new Button("Count");
18       add(btnCount);                // "super" Frame adds btnCount
19
20       // Construct an anonymous instance of BtnCountListener (a named inner class).
21       // btnCount adds this instance as a ActionListener.
22       btnCount.addActionListener(new BtnCountListener());
23
24       setTitle("AWT Counter");
25       setSize(250, 100);
26       setVisible(true);
27    }
28    /** The entry main method */
29    public static void main(String[] args) {
30       new AWTCounterNamedInnerClass(); // Let the constructor do the job
31    }
32    /**
33     * BtnCountListener is a "named inner class" used as ActionListener.
34     * This inner class can access private variables of the outer class.
35     */
36    private class BtnCountListener implements ActionListener {
37       @Override
38       public void actionPerformed(ActionEvent e) {
39          ++count;
40          tfCount.setText(count + "");
41       }
42    }
43 }
```

An Anonymous Inner Class as Event Listener

```
1  import java.awt.*;
2  import java.awt.event.*;
3  // An AWT GUI program inherits from the top-level container java.awt.Frame
4  public class AWTCounterAnonymousInnerClass extends Frame {
5     // This class is NOT a ActionListener, hence, it does not implement ActionListener
6     // The event-handler actionPerformed() needs to access these private variables
7     private TextField tfCount;
8     private int count = 0;
9     /** Constructor to setup the GUI */
10    public AWTCounterAnonymousInnerClass () {
11       setLayout(new FlowLayout());  // "super" Frame sets to FlowLayout
12       add(new Label("Counter"));    // an anonymous instance of Label
13       tfCount = new TextField("0", 10);
14       tfCount.setEditable(false);   // read-only
15       add(tfCount);                 // "super" Frame adds tfCount
16
17       Button btnCount = new Button("Count");
18       add(btnCount);                // "super" Frame adds btnCount
19
20       // Construct an anonymous instance of an anonymous class.
21       // btnCount adds this instance as a ActionListener.
22       btnCount.addActionListener(new ActionListener() {
23          @Override
24          public void actionPerformed(ActionEvent e) {
25             ++count;
26             tfCount.setText(count + "");
27          }
28       });
29
30       setTitle("AWT Counter");
31       setSize(250, 100);
32       setVisible(true);
33    }
34
35    /** The entry main method */
36    public static void main(String[] args) {
37       new AWTCounterAnonymousInnerClass(); // Let the constructor do the job
38    }
39 }
```

WindowEvent and *WindowListener* Interface

- **A *WindowEvent* is fired (to all its *WindowEvent* listeners) when a window (e.g., Frame) has been opened/closed, activated/deactivated, iconified/deiconified via the 3 buttons at the top-right corner or other means. The source of *WindowEvent* shall be a top-level window-container such as Frame.**



WindowEvent and *WindowListener* Interface

- A ***WindowEvent*** listener must implement ***WindowListener*** interface, which declares 7 abstract event-handling methods, as follows

```
1  public void windowClosing(WindowEvent evt)
2     // Called-back when the user attempts to close the window by clicking the window close button.
3     // This is the most-frequently used handler.
4  public void windowOpened(WindowEvent evt)
5     // Called-back the first time a window is made visible.
6  public void windowClosed(WindowEvent evt)
7     // Called-back when a window has been closed as the result of calling dispose on the window.
8  public void windowActivated(WindowEvent evt)
9     // Called-back when the Window is set to be the active Window.
10 public void windowDeactivated(WindowEvent evt)
11    // Called-back when a Window is no longer the active Window.
12 public void windowIconified(WindowEvent evt)
13    // Called-back when a window is changed from a normal to a minimized state.
14 public void windowDeiconified(WindowEvent evt)
15    // Called-back when a window is changed from a minimized to a normal state.
```

Event Listener's Adapter Classes

- A WindowEvent listener is required to implement the WindowListener interface, which declares 7 abstract methods
- Although we are only interested in windowClosing(), we need to provide an empty body to the other 6 methods in order to compile the program

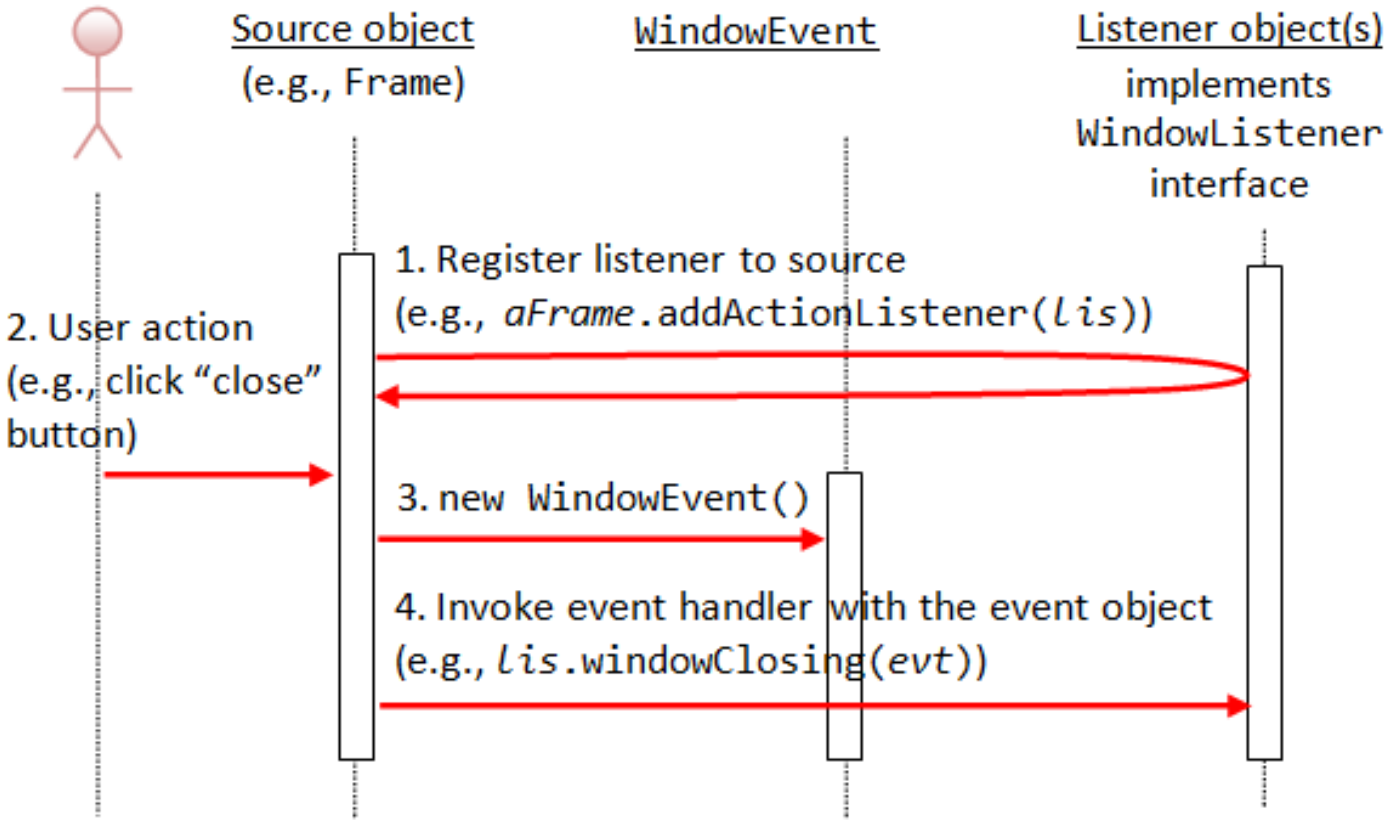
```
27        // Allocate an anonymous instance of an anonymous inner class
28        // that implements WindowListener.
29        // "this" Frame adds the instance as WindowEvent listener.
30        addWindowListener(new WindowListener() {
31           @Override
32           public void windowClosing(WindowEvent e) {
33              System.exit(0);  // terminate the program
34           }
35           // Need to provide an empty body for compilation
36           @Override public void windowOpened(WindowEvent e) { }
37           @Override public void windowClosed(WindowEvent e) { }
38           @Override public void windowIconified(WindowEvent e) { }
39           @Override public void windowDeiconified(WindowEvent e) { }
40           @Override public void windowActivated(WindowEvent e) { }
41           @Override public void windowDeactivated(WindowEvent e) { }
42        });
```

Event Listener‘s Adapter Classes (Cont.)

- An adapter class called *WindowAdapter* is therefore provided, which implements the WindowListener interface and provides default implementations to all the 7 abstract methods
- You can override only methods of interest and leave the rest to their default implementation

```
27        // Allocate an anonymous instance of an anonymous inner class
28        // that extends WindowAdapter.
29        // "this" Frame adds the instance as WindowEvent listener.
30        addWindowListener(new WindowAdapter() {
31           @Override
32           public void windowClosing(WindowEvent e) {
33              System.exit(0);  // Terminate the program
34           }
35        });
```

Event Listener‘s Adapter Classes (Cont.)



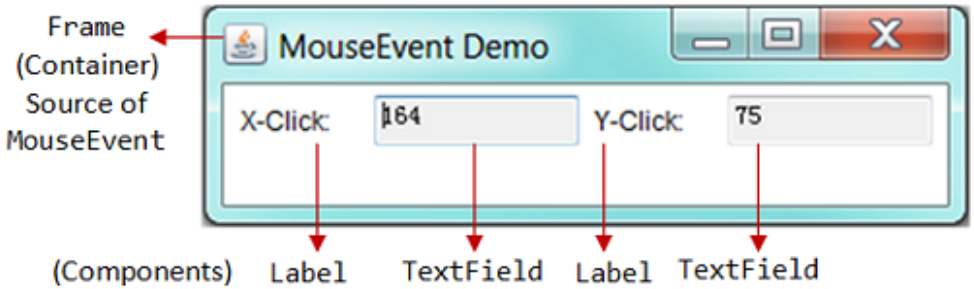
MouseEvent and *MouseListener* Interface

- A ***MouseEvent*** is fired to all its registered listeners, when you press, release, or click (press followed by release) a mouse-button (left or right button) at the source object; or position the mouse-pointer at (enter) and away (exit) from the source object.
- A ***MouseEvent*** listener must implement the ***MouseListener*** interface, which declares the following five abstract methods:

```
1 public void mouseClicked(MouseEvent evt)
2    // Called-back when the mouse-button has been clicked (pressed followed by released) on the source.
3 public void mousePressed(MouseEvent evt)
4 public void mouseReleased(MouseEvent evt)
5    // Called-back when a mouse-button has been pressed/released on the source.
6    // A mouse-click invokes mousePressed(), mouseReleased() and mouseClicked().
7 public void mouseEntered(MouseEvent evt)
8 public void mouseExited(MouseEvent evt)
9    // Called-back when the mouse-pointer has entered/exited the source.
```

MouseEvent and *MouseListener* Interface

```
1  import java.awt.*;
2  import java.awt.event.*;
3
4  public class MouseEventDemo extends Frame {
5     private TextField tfMouseX; // to display mouse-click-x
6     private TextField tfMouseY; // to display mouse-click-y
7
8     // Constructor - Setup the UI components and event handlers
9     public MouseEventDemo() {
10       setLayout(new FlowLayout()); // "super" frame sets its layout to FlowLayout
11       // Label (anonymous)
12       add(new Label("X-Click: ")); // "super" frame adds Label component
13       // TextField
14       tfMouseX = new TextField(10); // 10 columns
15       tfMouseX.setEditable(false);  // read-only
16       add(tfMouseX);                // "super" frame adds TextField component
17       // Label (anonymous)
18       add(new Label("Y-Click: ")); // "super" frame adds Label component
19       // TextField
20       tfMouseY = new TextField(10);
21       tfMouseY.setEditable(false);  // read-only
22       add(tfMouseY);                // "super" frame adds TextField component
23
24       addMouseListener(new MouseListener{
25             /* MouseEvent handlers */
26          // Called back upon mouse clicked
27          @Override
28          public void mouseClicked(MouseEvent evt) {
29             tfMouseX.setText(evt.getX() + "");
30             tfMouseY.setText(evt.getY() + "");
31          }
32
33          // Not used - need to provide an empty body to compile.
34          @Override public void mousePressed(MouseEvent evt) { }
35          @Override public void mouseReleased(MouseEvent evt) { }
36          @Override public void mouseEntered(MouseEvent evt) { }
37          @Override public void mouseExited(MouseEvent evt) { }
38       });
39          // "super" frame (source) fires the MouseEvent.
40          // "super" frame adds "this" object as a MouseEvent listener.
41       setTitle("MouseEvent Demo"); // "super" Frame sets title
42       setSize(350, 100);           // "super" Frame sets initial size
43       setVisible(true);            // "super" Frame shows
44    }
45    public static void main(String[] args) {
46       new MouseEventDemo();  // Let the constructor do the job
47    }
48 }
```



Layout Managers and Panel

- A container has a so-called *layout manager* to arrange its components
- AWT provides the following layout managers
  - *FlowLayout*, *GridLayout*, *BorderLayout*, *GridBagLayout*, *BoxLayout*, *CardLayout*, etc.
- A container has a *setLayout()* method to set its layout manager
  - public void setLayout(LayoutManager mgr)
- To set up the layout of a Container (such as Frame, Panel), you have to:
  - Construct an instance of the chosen layout object, via new and constructor, e.g., *new FlowLayout()*
  - Invoke the *setLayout()* method of the Container, with the layout object created as the argument
  - Place the GUI components into the Container using the *add()* method in the correct order; or into the correct zones

Layout Managers and Panel

- **For example**

```
1  // Allocate a Panel (container)
2  Panel p = new Panel();
3  // Allocate a new Layout object. The Panel container sets to this layout.
4  p.setLayout(new FlowLayout());
5  // The Panel container adds components in the proper order.
6  p.add(new JLabel("One"));
7  p.add(new JLabel("Two"));
8  p.add(new JLabel("Three"));
9  ......
```

- **Container's getLayout()**

```
1  Panel awtPanel = new Panel();
2  System.out.println(awtPanel.getLayout());
3        // java.awt.FlowLayout[hgap=5,vgap=5,align=center]
```

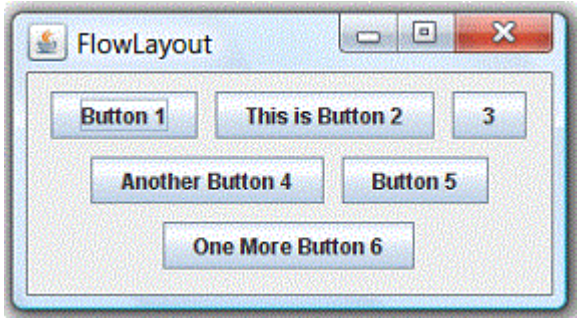
- **Panel's Initial Layout**
  - **Panel provides a constructor to set its initial layout manager**

```
1  public void Panel (LayoutManager layout)
2     // Construct a Panel in the given layout
3     // By default, Panel (and JPanel) has FlowLayout
4
5  // For example, create a Panel in BorderLayout
6  Panel mainPanel = new Panel(new BorderLayout());
```

java.awt.FlowLayout

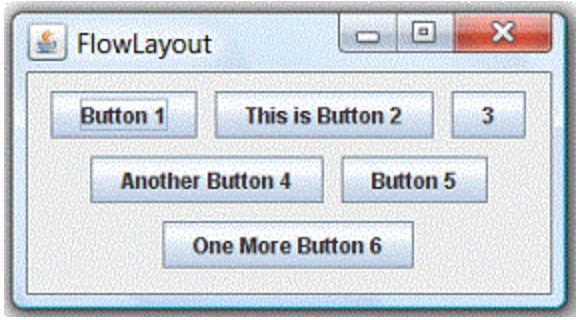
- Components are arranged from **left-to-right** inside the container in the order that they are added
  - When one row is filled, a new row will be started
  - The actual appearance depends on the width of the display window

- **Constructor**
  - *public FlowLayout()*;
  - *public FlowLayout(int align)*;
  - *public FlowLayout(int align, int hgap, int vgap)*;



java.awt.FlowLayout (Cont.)

```
1  import java.awt.*;
2  import java.awt.event.*;
3
4  // An AWT GUI program inherits the top-level container java.awt.Frame
5  public class AWTFlowLayoutDemo extends Frame {
6     private Button btn1, btn2, btn3, btn4, btn5, btn6;
7
8     /** Constructor to setup GUI components */
9     public AWTFlowLayoutDemo () {
10       setLayout(new FlowLayout());
11          // "this" Frame sets layout to FlowLayout, which arranges the components
12          // from left-to-right, and flow from top-to-bottom.
13
14       btn1 = new Button("Button 1");
15       add(btn1);
16       btn2 = new Button("This is Button 2");
17       add(btn2);
18       btn3 = new Button("3");
19       add(btn3);
20       btn4 = new Button("Another Button 4");
21       add(btn4);
22       btn5 = new Button("Button 5");
23       add(btn5);
24       btn6 = new Button("One More Button 6");
25       add(btn6);
26
27       setTitle("FlowLayout Demo"); // "this" Frame sets title
28       setSize(280, 150);       // "this" Frame sets initial size
29       setVisible(true);        // "this" Frame shows
30    }
31
32    /** The entry main() method */
33    public static void main(String[] args) {
34       new AWTFlowLayoutDemo();  // Let the constructor do the job
35    }
36 }
```



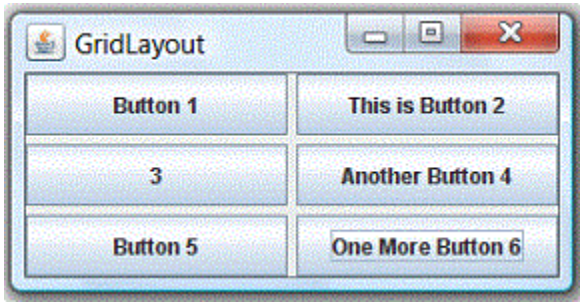
java.awt.GridLayout

- Components are arranged in a grid (matrix) of rows and columns inside the Container
- Components are added in a left-to-right, top-to-bottom manner in the order they are added

- **Constructor**
  - *public GridLayout(int rows, int columns)*;
  - *public GridLayout(int rows, int columns, int hgap, int vgap)*;
    - By default: rows=1, cols=0, hgap=0, vgap=0

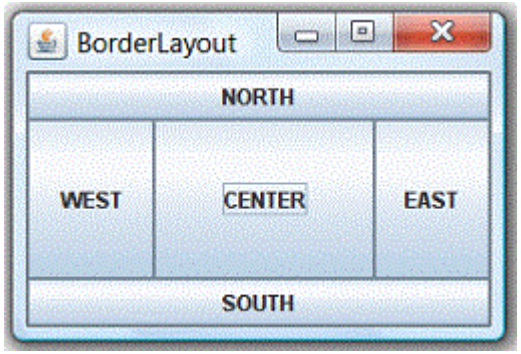
java.awt.GridLayout (Cont.)

```
1  import java.awt.*;
2  import java.awt.event.*;
3
4  // An AWT GUI program inherits the top-level container java.awt.Frame
5  public class AWTGridLayoutDemo extends Frame {
6     private Button btn1, btn2, btn3, btn4, btn5, btn6;
7
8     /** Constructor to setup GUI components */
9     public AWTGridLayoutDemo () {
10       setLayout(new GridLayout(3, 2, 3, 3));
11          // "this" Frame sets layout to 3x2 GridLayout, horizontal and verical gaps of 3 pixels
12
13       // The components are added from left-to-right, top-to-bottom
14       btn1 = new Button("Button 1");
15       add(btn1);
16       btn2 = new Button("This is Button 2");
17       add(btn2);
18       btn3 = new Button("3");
19       add(btn3);
20       btn4 = new Button("Another Button 4");
21       add(btn4);
22       btn5 = new Button("Button 5");
23       add(btn5);
24       btn6 = new Button("One More Button 6");
25       add(btn6);
26
27       setTitle("GridLayout Demo"); // "this" Frame sets title
28       setSize(280, 150);       // "this" Frame sets initial size
29       setVisible(true);        // "this" Frame shows
30    }
31
32    /** The entry main() method */
33    public static void main(String[] args) {
34       new AWTGridLayoutDemo();  // Let the constructor do the job
35    }
36 }
```



java.awt.BorderLayout

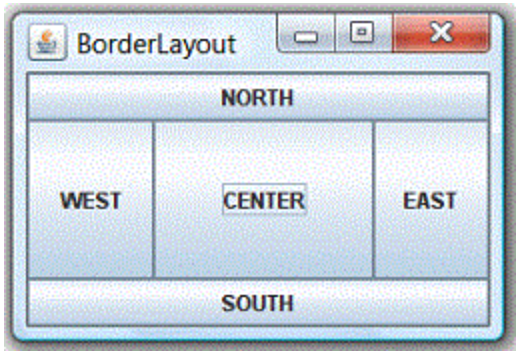
- The container is divided into 5 zones
  - EAST, WEST, SOUTH, NORTH, and CENTER
- Components are added using method aContainer.***add***(acomponent, aZone)
  - azone is either BorderLayout.NORTH (or PAGE_START), BorderLayout.SOUTH (or PAGE_END), BorderLayout.WEST (or LINE_START), BorderLayout.EAST (or LINE_END), or BorderLayout.CENTER



- **Constructor**
  - ***public BorderLayout()***;
  - ***public BorderLayout(int hgap, int vgap)***;
    - By default: hgap=0, vgap=0

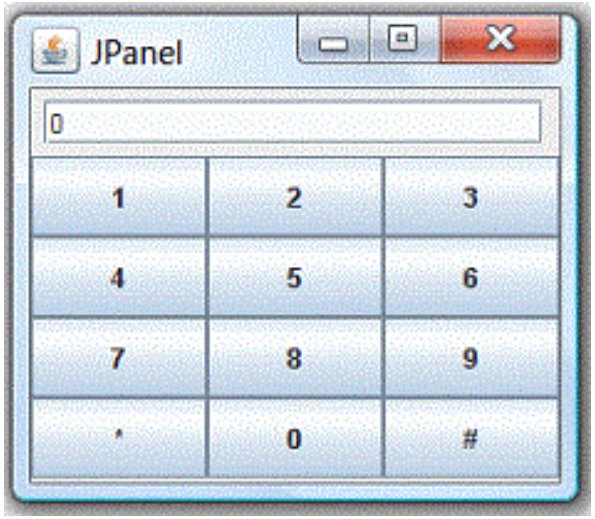
java.awt.GridLayout (Cont.)

```
1  import java.awt.*;
2  import java.awt.event.*;
3
4  // An AWT GUI program inherits the top-level container java.awt.Frame
5  public class AWTBorderLayoutDemo extends Frame {
6     private Button btnNorth, btnSouth, btnCenter, btnEast, btnWest;
7
8     /** Constructor to setup GUI components */
9     public AWTBorderLayoutDemo () {
10       setLayout(new BorderLayout(3, 3));
11          // "this" Frame sets layout to BorderLayout,
12          // horizontal and vertical gaps of 3 pixels
13
14       // The components are added to the specified zone
15       btnNorth = new Button("NORTH");
16       add(btnNorth, BorderLayout.NORTH);
17       btnSouth = new Button("SOUTH");
18       add(btnSouth, BorderLayout.SOUTH);
19       btnCenter = new Button("CENTER");
20       add(btnCenter, BorderLayout.CENTER);
21       btnEast = new Button("EAST");
22       add(btnEast, BorderLayout.EAST);
23       btnWest = new Button("WEST");
24       add(btnWest, BorderLayout.WEST);
25
26       setTitle("BorderLayout Demo"); // "this" Frame sets title
27       setSize(280, 150);        // "this" Frame sets initial size
28       setVisible(true);         // "this" Frame shows
29    }
30
31    /** The entry main() method */
32    public static void main(String[] args) {
33       new AWTBorderLayoutDemo();  // Let the constructor do the job
34    }
35 }
```



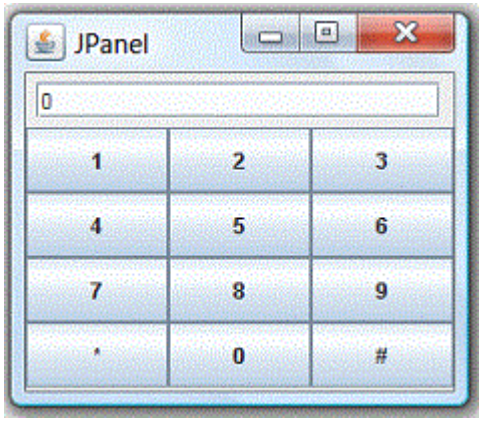
Using Panels as Sub-Container to Organize Components

- An AWT Panel is a rectangular pane
  - Used as sub-container to organized a group of related components in a specific layout (e.g., FlowLayout, BorderLayout)
- Panels are secondary containers
  - Can be added into a top-level container (such as Frame), or another Panel



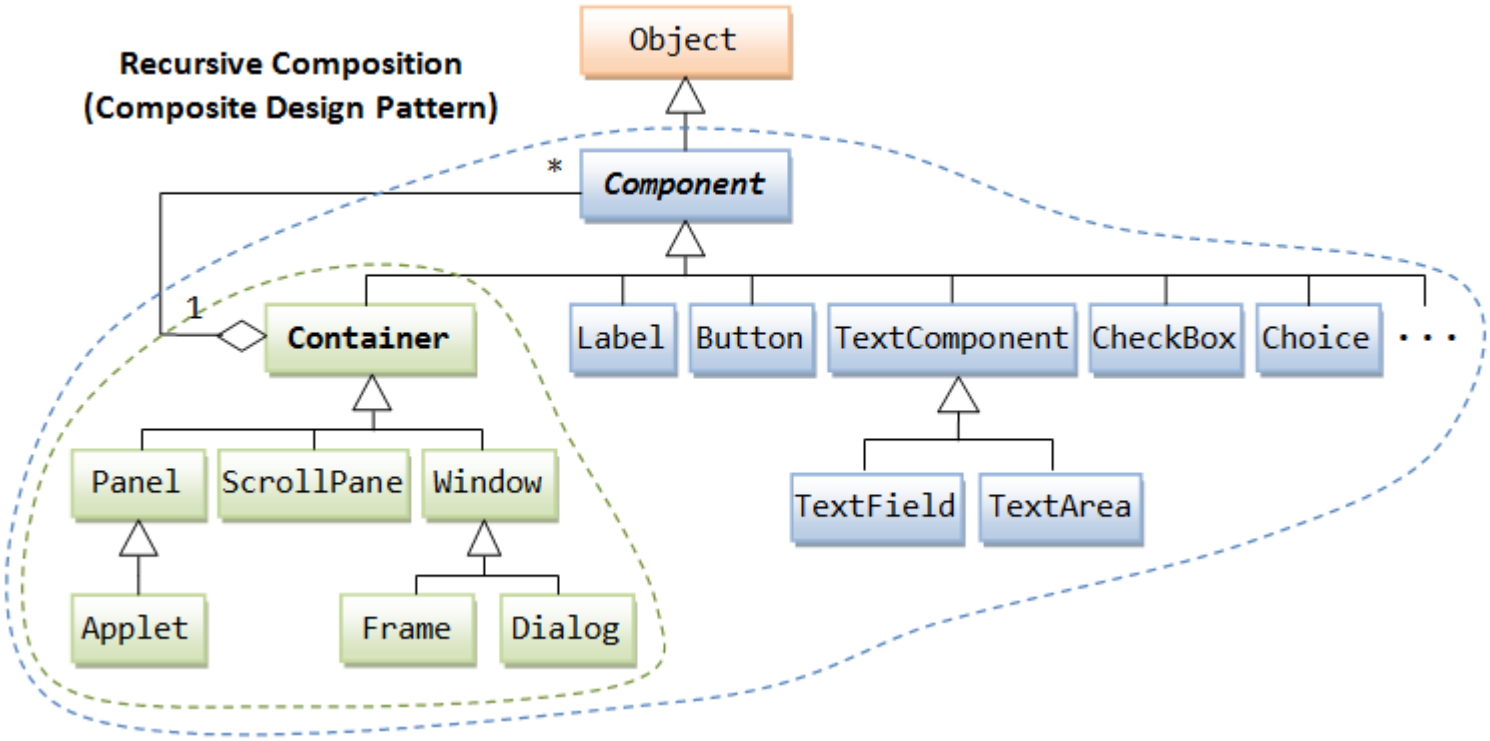
Using Panels as Sub-Container to Organize Components (Cont.)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  // An AWT GUI program inherits the top-level container java.awt.Frame
4  public class AWTPanelDemo extends Frame {
5     private Button[] btnNumbers = new Button[10];  // Array of 10 numeric buttons
6     private Button btnHash, btnStar;
7     private TextField tfDisplay;
8
9     /** Constructor to setup GUI components */
10    public AWTPanelDemo () {
11       // Set up display panel
12       Panel panelDisplay = new Panel(new FlowLayout());
13       tfDisplay = new TextField("0", 20);
14       panelDisplay.add(tfDisplay);
15       // Set up button panel
16       Panel panelButtons = new Panel(new GridLayout(4, 3));
17       btnNumbers[1] = new Button("1");
18       panelButtons.add(btnNumbers[1]);
19       ......
20       btnNumbers[9] = new Button("9");
21       panelButtons.add(btnNumbers[9]);
22          // Can use a loop for the above statements!
23       btnStar = new Button("*");
24       panelButtons.add(btnStar);
25       btnNumbers[0] = new Button("0");
26       panelButtons.add(btnNumbers[0]);
27       btnHash = new Button("#");
28       panelButtons.add(btnHash);
29
30       setLayout(new BorderLayout());  // "this" Frame sets to BorderLayout
31       add(panelDisplay, BorderLayout.NORTH);
32       add(panelButtons, BorderLayout.CENTER);
33
34       setTitle("BorderLayout Demo"); // "this" Frame sets title
35       setSize(200, 200);         // "this" Frame sets initial size
36       setVisible(true);          // "this" Frame shows
37    }
38    /** The entry main() method */
39    public static void main(String[] args) {
40       new AWTPanelDemo();  // Let the constructor do the job
41    }
42 }
```



Composite Design Pattern

- Two groups of classes in the AWT hierarchy: containers and components
  - A container holds components
  - A container (e.g., Frame and Panel) can also hold sub-containers

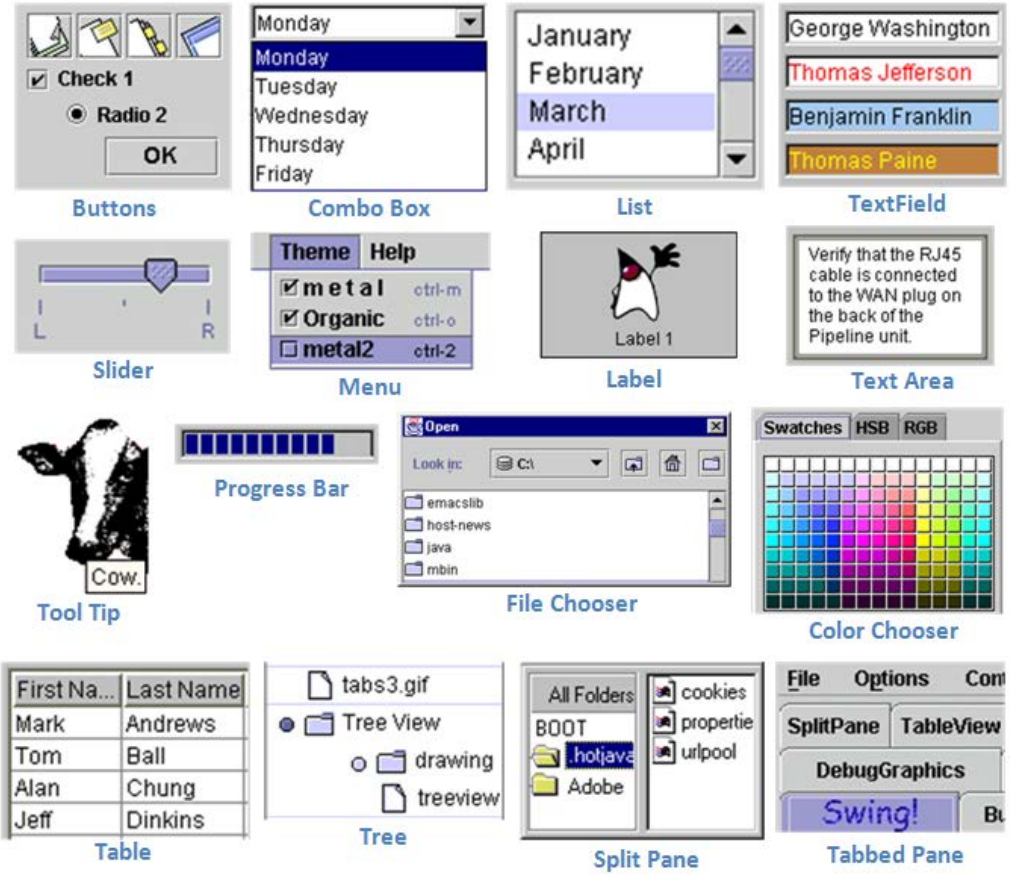


Swing

- Swing is part of the so-called "Java Foundation Classes (JFC)" (have you heard of MFC?)
  - Introduced in 1997 after the release of JDK 1.1
  - JFC was subsequently included as an integral part of JDK since JDK 1.2
- JFC consists of:
  - Swing API: for advanced graphical programming
  - Accessibility API: provides assistive technology for the disabled
  - Java 2D API: for high quality 2D graphics and images
  - Pluggable look and feel supports
  - Drag-and-drop support between Java and native applications

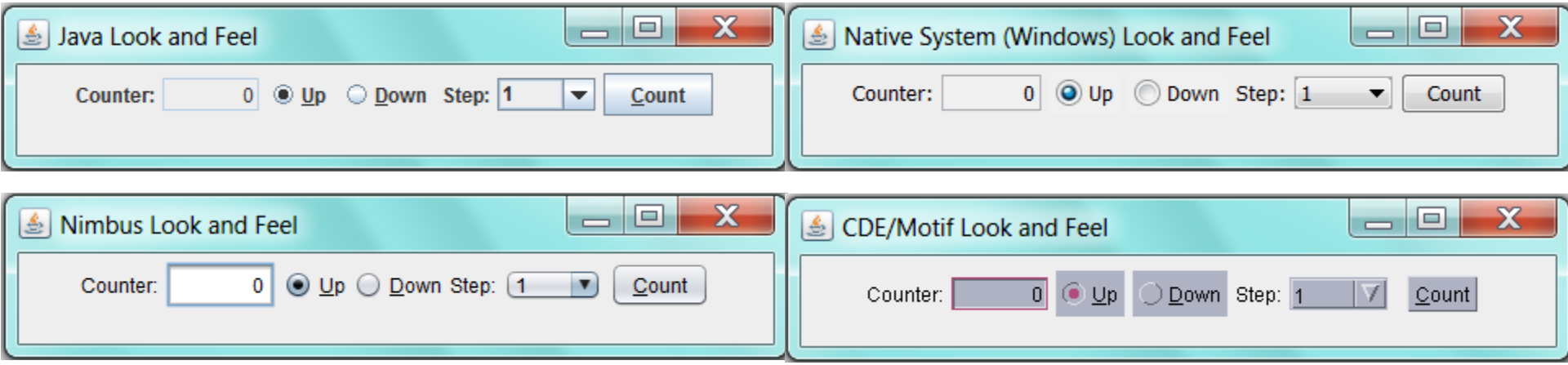
Swing's Features

- Swing is huge (consists of 18 API packages as in JDK 1.8) and has great depth
- Swing provides a huge and comprehensive collection of reusable GUI components



Swing‘s Features (Cont.)

- Swing is written in pure Java (except a few classes) and therefore is 100% *portable*
- Swing components are *lightweight*
- Swing components support *pluggable look-and-feel*



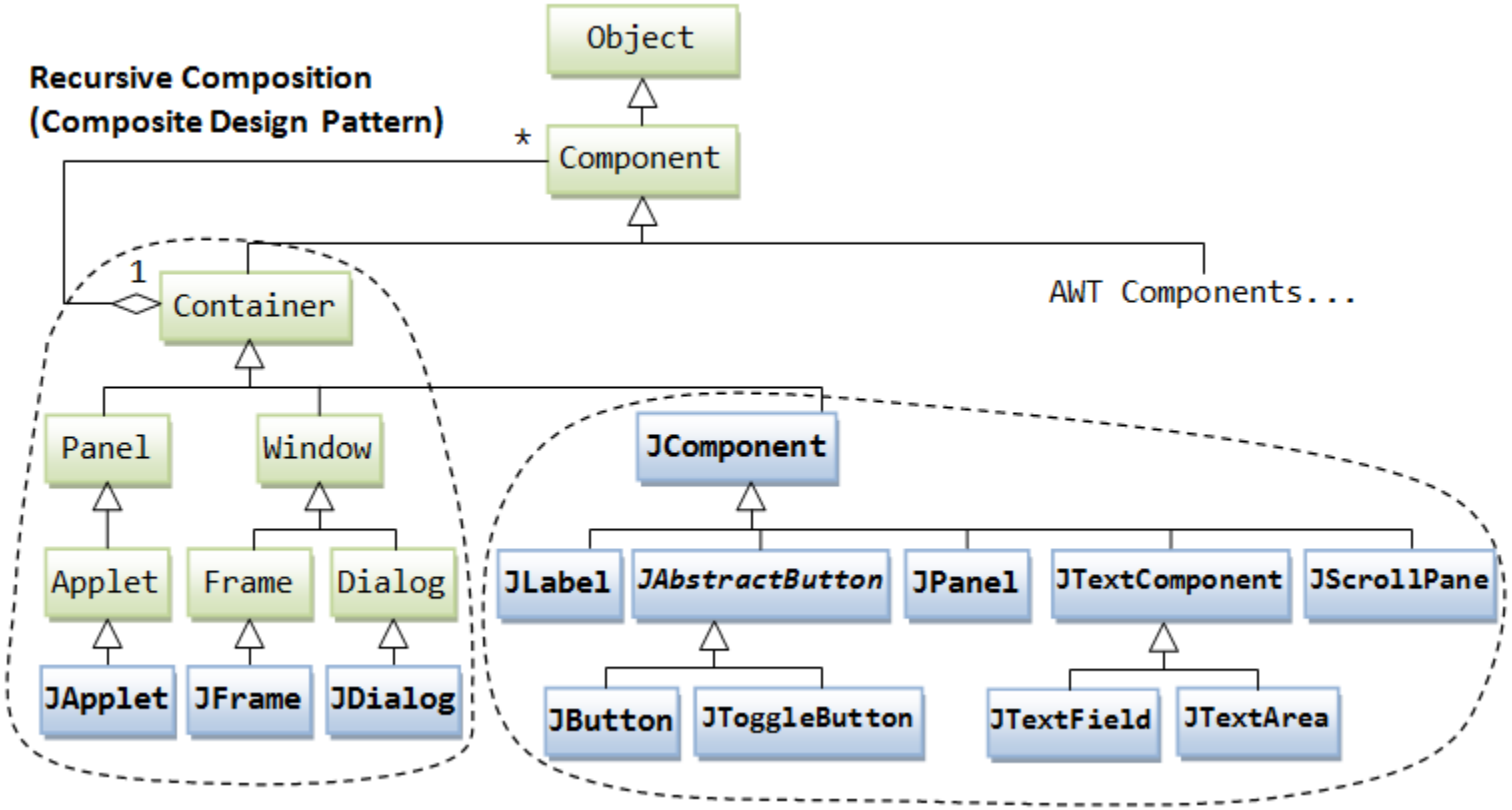
- Swing supports *mouse-less operation*
- Swing components support "*tool-tips*“
- Swing components are *JavaBeans* – a Component-based Model used in Visual Programming

Swing‘s Features (Cont.)

- Swing application uses AWT event-handling classes
  - Swing event: javax.swing.event, but not frequently used
- Swing application uses AWT's layout manager
  - Add new layout managers, such as Springs, Struts, and BoxLayout
- Swing implements double-buffering and automatic repaint batching for smoother screen repaint
- Swing introduces JLayeredPane and JInternalFrame for creating Multiple Document Interface (MDI) applications
- Swing supports floating toolbars (in JToolBar), splitter control, "undo "
- Others - check the Swing website

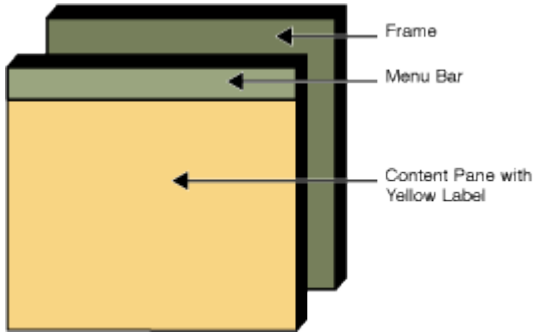
Using Swing API

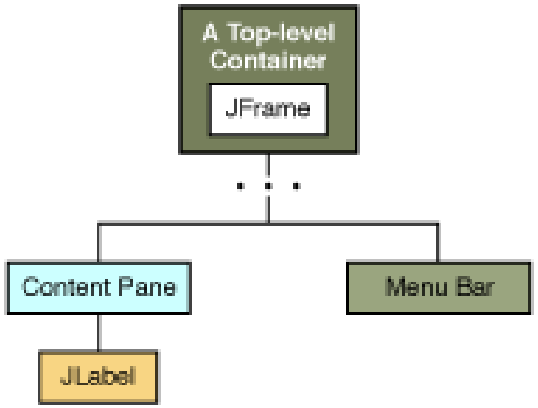
- If you understood the AWT programming, switching over to ***Swing*** is straight-forward
- Swing component classes begin with a prefix "J "
  - E.g., JButton, JTextField, JLabel, JPanel, JFrame, or JApplet
- Two groups of classes: containers and components



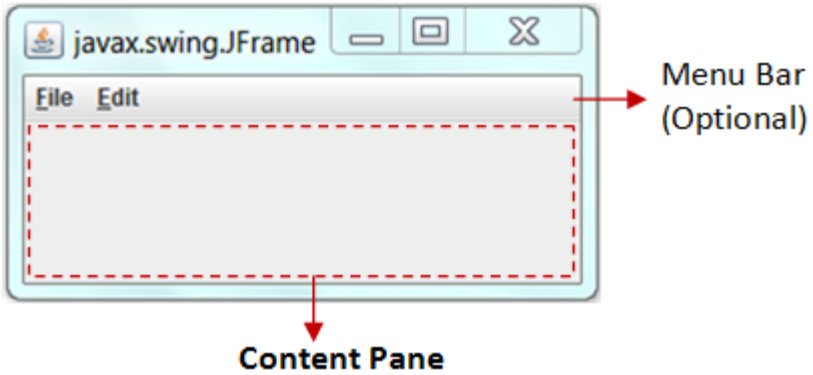
The Content-Pane of Swing's Top-Level Container

- The JComponents must be added onto the so-called *content-pane* of the top-level container (e.g., JFrame)
  - JComponents are *lightweight* components





javax.swing.JFrame




```
Container cp = aJFrame.getContentPane();
aJFrame.setContentPane(aPanel);
```

The Content-Pane of Swing's Top-Level Container

- **Get the content-pane via *getContentPane()* from a top-level container, and add components onto it**

```
1  public class TestGetContentPane extends JFrame {
2     // Constructor
3     public TestGetContentPane() {
4        // Get the content-pane of this JFrame, which is a java.awt.Container
5        // All operations, such as setLayout() and add() operate on the content-pane
6        Container cp = this.getContentPane();
7        cp.setLayout(new FlowLayout());
8        cp.add(new JLabel("Hello, world!"));
9        cp.add(new JButton("Button"));
10       ......
11    }
12    .......
13 }
```

- **Set the content-pane to a JPanel via JFrame's *setContentPane()***

```
1  public class TestSetContentPane extends JFrame {
2     // Constructor
3     public TestSetContentPane() {
4        // The "main" JPanel holds all the GUI components
5        JPanel mainPanel = new JPanel(new FlowLayout());
6        mainPanel.add(new JLabel("Hello, world!"));
7        mainPanel.add(new JButton("Button"));
8
9        // Set the content-pane of this JFrame to the main JPanel
10       this.setContentPane(mainPanel);
11       ......
12    }
13    .......
14 }
```

The Content-Pane of Swing's Top-Level Container

- **Notes: If a component is added directly into a JFrame, it is added into the content-pane of JFrame instead, i.e.,**

```
1  // "this" is a JFrame
2  add(new JLabel("add to JFrame directly"));
3  // is executed as
4  getContentPane().add(new JLabel("add to JFrame directly"));
```

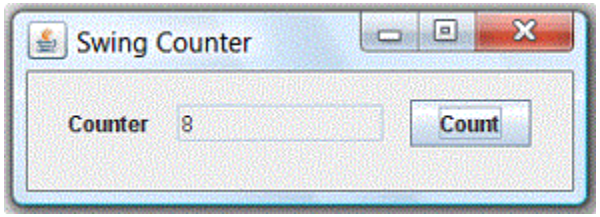
The Content-Pane of Swing's Top-Level Container

- **Event-Handling in Swing**
  - Swing uses the AWT event-handling classes (in package java.awt.event)
  - Swing introduces a few new event-handling classes (in package javax.swing.event) but they are not frequently used
- **Writing Swing Applications**
  - Use the Swing components with prefix "J" in package javax.swing, e.g., JFrame, JButton, JTextField, JLabel, etc.
  - A top-level container (such as JFrame or JApplet) is needed. The JComponents shall be added onto the content-pane of the top-level container.
    - Retrieve a reference to the content-pane by invoking method getContentPane() from the top-level container
    - Set the content-pane to the main JPanel created in your program
  - Swing applications uses AWT event-handling classes, e.g., ActionEvent/ActionListener, MouseEvent/MouseListener, etc.
  - Run the constructor in the Event Dispatcher Thread (instead of Main thread) for thread safety, as shown in the following program template

Swing Program Template

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  // A Swing GUI application inherits from top-level container javax.swing.JFrame
5  public class ..... extends JFrame {
6      // private variables
7     // ......
8
9     /** Constructor to setup the GUI components */
10    public ......() {
11       Container cp = this.getContentPane();
12
13       // Content-pane sets layout
14       cp.setLayout(new ....Layout());
15
16       // Allocate the GUI components
17       // .....
18
19       // Content-pane adds components
20       cp.add(....);
21
22       // Source object adds listener
23       // .....
24
25       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
26          // Exit the program when the close-window button clicked
27       setTitle("......");  // "this" JFrame sets title
28       setSize(300, 150);   // "this" JFrame sets initial size (or pack())
29       setVisible(true);    // show it
30    }
31
32    /** The entry main() method */
33    public static void main(String[] args) {
34       // Run GUI codes in Event-Dispatching thread for thread-safety
35       SwingUtilities.invokeLater(new Runnable() {
36          @Override
37          public void run() {
38             new ......();  // Let the constructor do the job
39          }
40       });
41    }
42 }
```

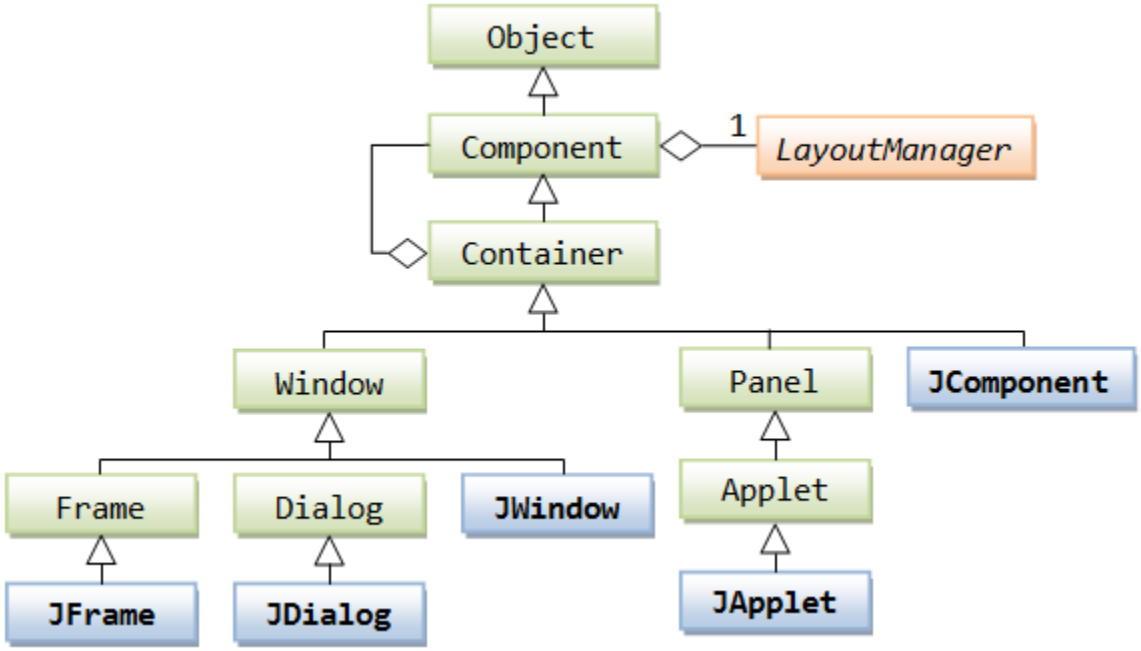
Swing Example: SwingCounter




```java
import java.awt.*;       // Using AWT containers and components
import java.awt.event.*; // Using AWT events and listener interfaces
import javax.swing.*;    // Using Swing components and containers

// A Swing GUI application inherits from top-level container
// javax.swing.JFrame
public class SwingCounter extends JFrame {
   private JTextField tfCount;
   // Use Swing's JTextField instead of AWT's TextField
   private int count = 0;

   /** Constructor to setup the GUI */
   public SwingCounter () {
      // Retrieve the content-pane of the top-level container JFrame
      // All operations done on the content-pane
      Container cp = getContentPane();
      cp.setLayout(new FlowLayout());

      cp.add(new JLabel("Counter"));
      tfCount = new JTextField("0", 10);
      tfCount.setEditable(false);
      cp.add(tfCount);

      JButton btnCount = new JButton("Count");
      cp.add(btnCount);

      // Allocate an anonymous instance of an anonymous inner class
      // that implements ActionListener as ActionEvent listener
      btnCount.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
            ++count;
            tfCount.setText(count + "");
         }
      });

      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      // Exit program if close-window button clicked
      setTitle("Swing Counter"); // "this" JFrame sets title
      setSize(300, 100);         // "this" JFrame sets initial size
      setVisible(true);          // "this" JFrame shows
   }

   /** The entry main() method */
   public static void main(String[] args) {
      // Run the GUI construction in the Event-Dispatching thread
      // for thread-safety
      SwingUtilities.invokeLater(new Runnable() {
         @Override
         public void run() {
            new SwingCounter(); // Let the constructor do the job
         }
      });
   }
}
```

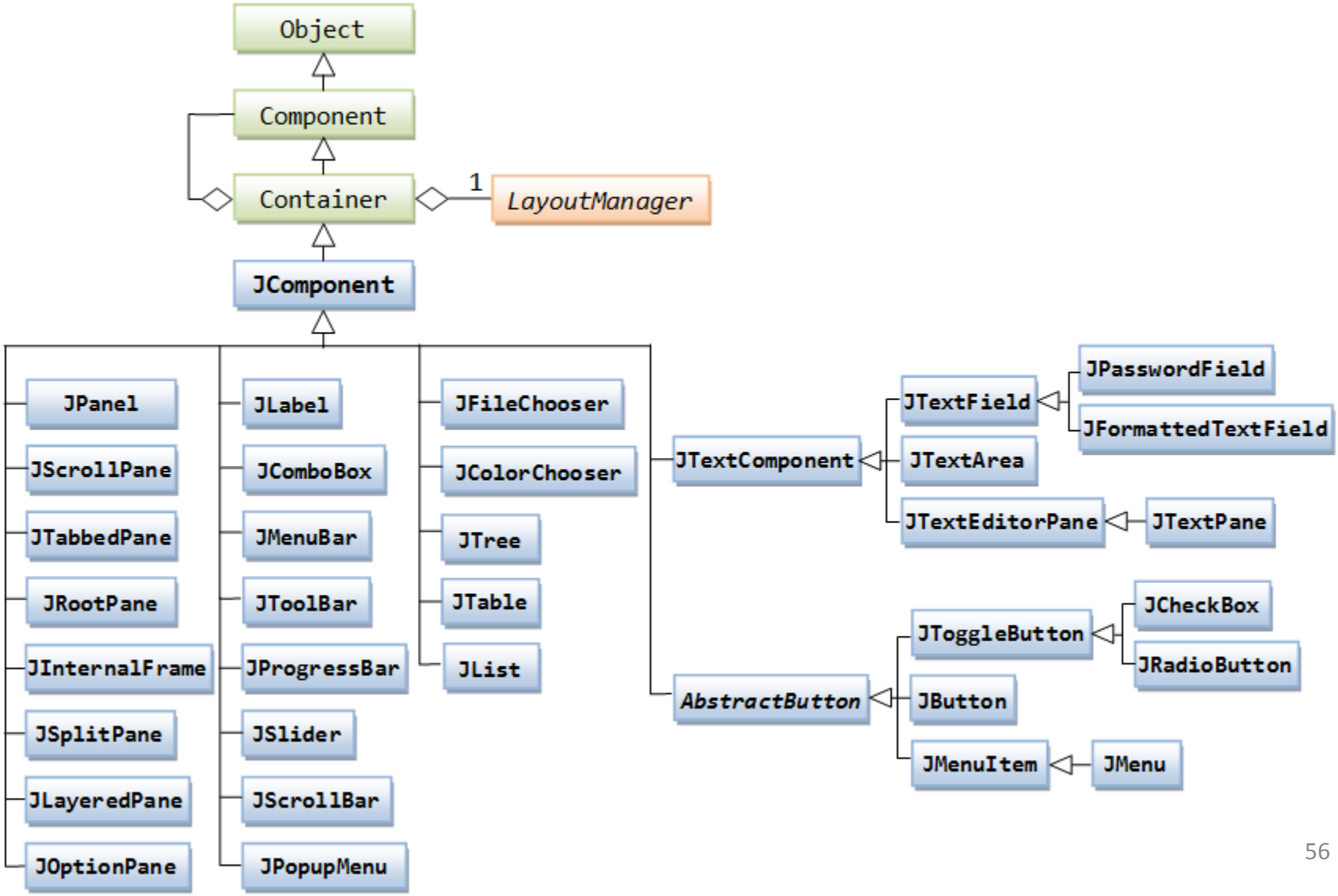
Class Hierarchy of Swing's Top-level Containers

- JFrame, JDialog, JApplet
- These top-level Swing containers are heavyweight, that rely on the underlying windowing subsystem of the native platform



Class Hierarchy of Swing's JComponents

- JComponent and its descendants are lightweight components



谢谢

zhenling@seu.edu.cn