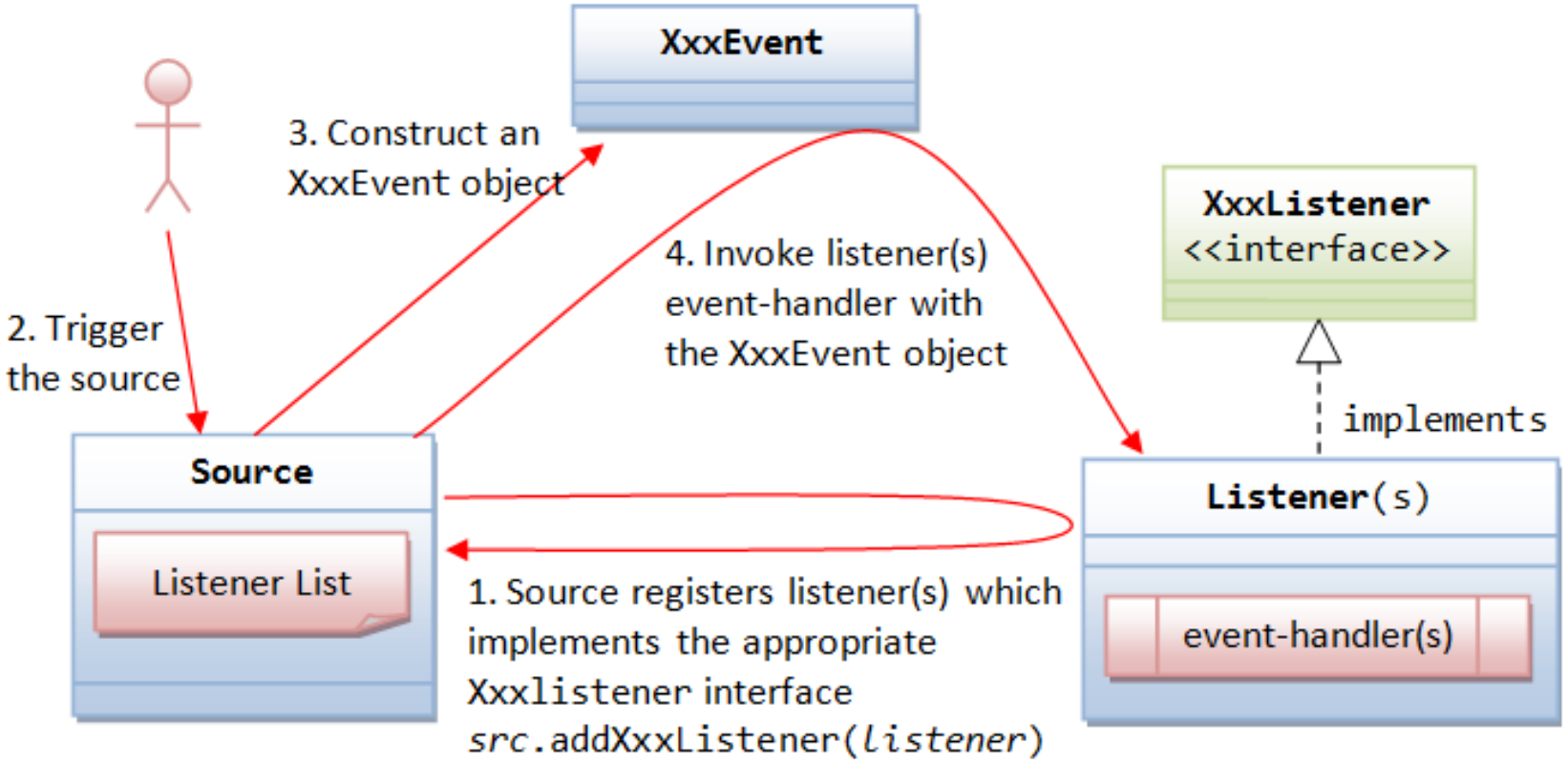
COP 3330, Spring 2013

GUIs – IV (Recap- Detailed Discussion on AWT and Swing API)

Instructor : Arup Ghosh
04-03-13

School of Electrical Engineering and Computer Science
University of Central Florida

AWT Event-Handling



AWT Event-Handling

- In event-driven programming, a piece of event-handling codes is executed (or called back) when an event has been fired in response to an user input (such as clicking a mouse button or hitting the ENTER key). This is unlike the procedural model, where codes are executed in a sequential manner.

- The AWT's event-handling classes are kept in package java.awt.event.

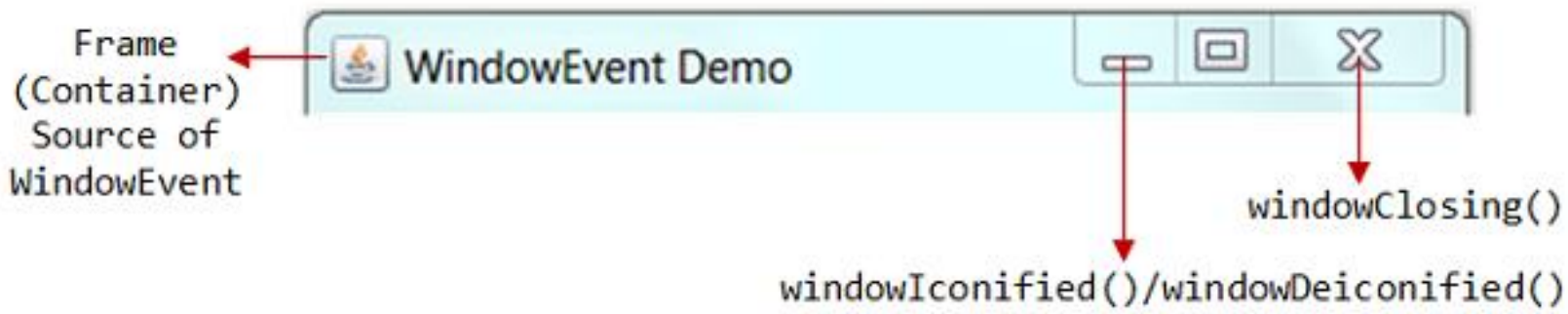
AWT Event-Handling

- Three objects are involved in the event-handling: a *source*, a *listenser*(s) and an *event* object.
- The *source* object (such as Button and Textfield) interacts with the user. Upon triggered, it creates an *event* object. This *event* object will be messaged to all the *registered listener* object(s), and an appropriate event-handler method of the listener(s) is called-back to provide the response. In other words, *triggering a source fires an event to all its listeners*.

WindowEvent and WindowListener Interface

- A WindowEvent is fired (to all its WindowListensers) 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 a WindowEvent shall be a top-level window-container such as Frame.
- A WindowEvent listener must implement WindowListener interface, which declares abstract event-handling methods. Among them, the windowClosing(), which is called back upon clicking the window-close button, is the most commonly-used.

WindowEvent and WindowListener

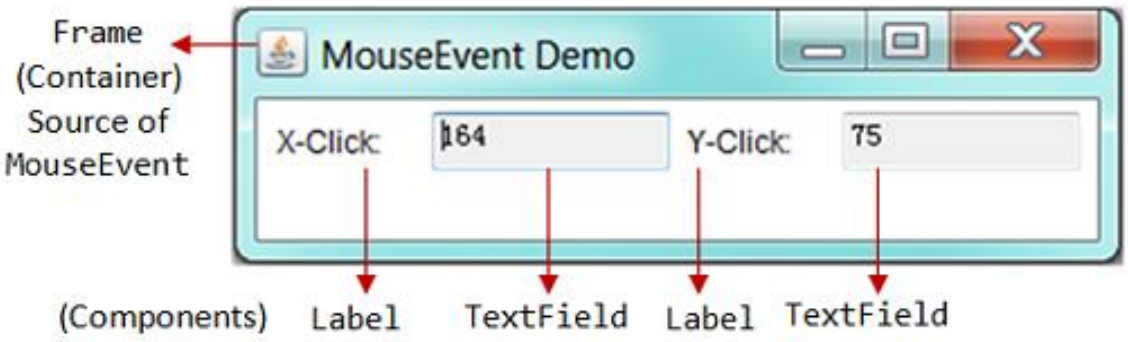


Example 2: WindowEvent Demo

- Implement Example 1 with WindowEvent and WindowListener

MouseEvent and MouseListener Interface

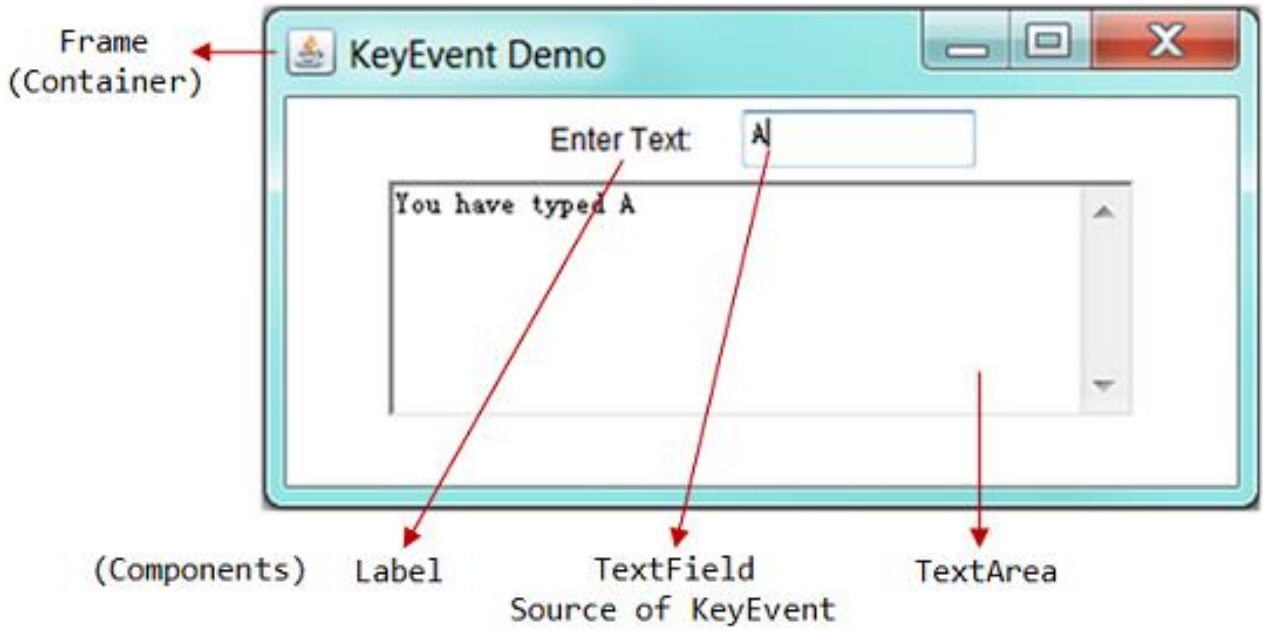
- A MouseEvent is fired to all its registered listeners, when you press, release, or click (press followed by release) a mouse-button (left or right button) at the source object; or position the mouse-pointer at (enter) and away (exit) from the source object.
- A MouseEvent listener must implement the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



KeyEvent and KeyListener Interface

- A KeyEvent is fired (to all its registered KeyListeners) when you pressed, released, and typed (pressed followed by released) a key on the source object. A KeyEvent listener must implement KeyListenerinterface, which declares three abstract methods:
  - public void keyTyped(KeyEvent e) // Called-back when a key has been typed (pressed and released).
  - public void keyPressed(KeyEvent e)
  - public void keyReleased(KeyEvent e) // Called-back when a key has been pressed/released.

Example 3: KeyEventDemo



Event Listener's Adapter class

- Refer to the WindowEventDemo, 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. This is tedious.

- 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 then derive a subclass from WindowAdapter and override only methods of interest and leave the rest to their default implementation.

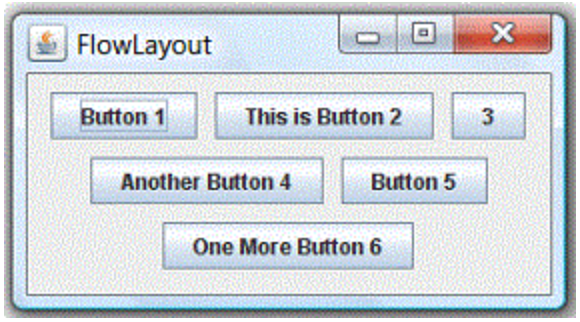
Other Event-Listener Adapter Classes

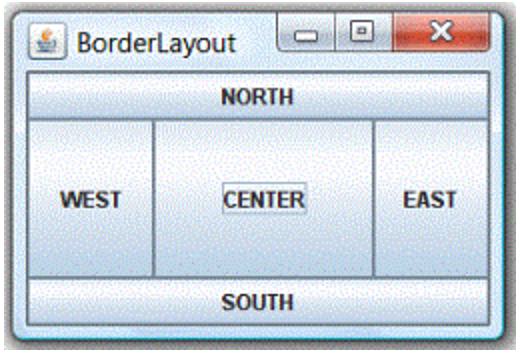
- Similarly, adapter classes such as MouseAdapter, MouseMotionAdapter, KeyAdapter, FocusAdapter are available for MouseListener, MouseMotionListener, KeyListener, and FocusListener, respectively.

- There is no ActionAdapter for ActionListener, because there is only one abstract method (i.e. actionPerformed()) declared in the ActionListener interface. This method has to be overridden and there is no need for an adapter.

Layout Managers

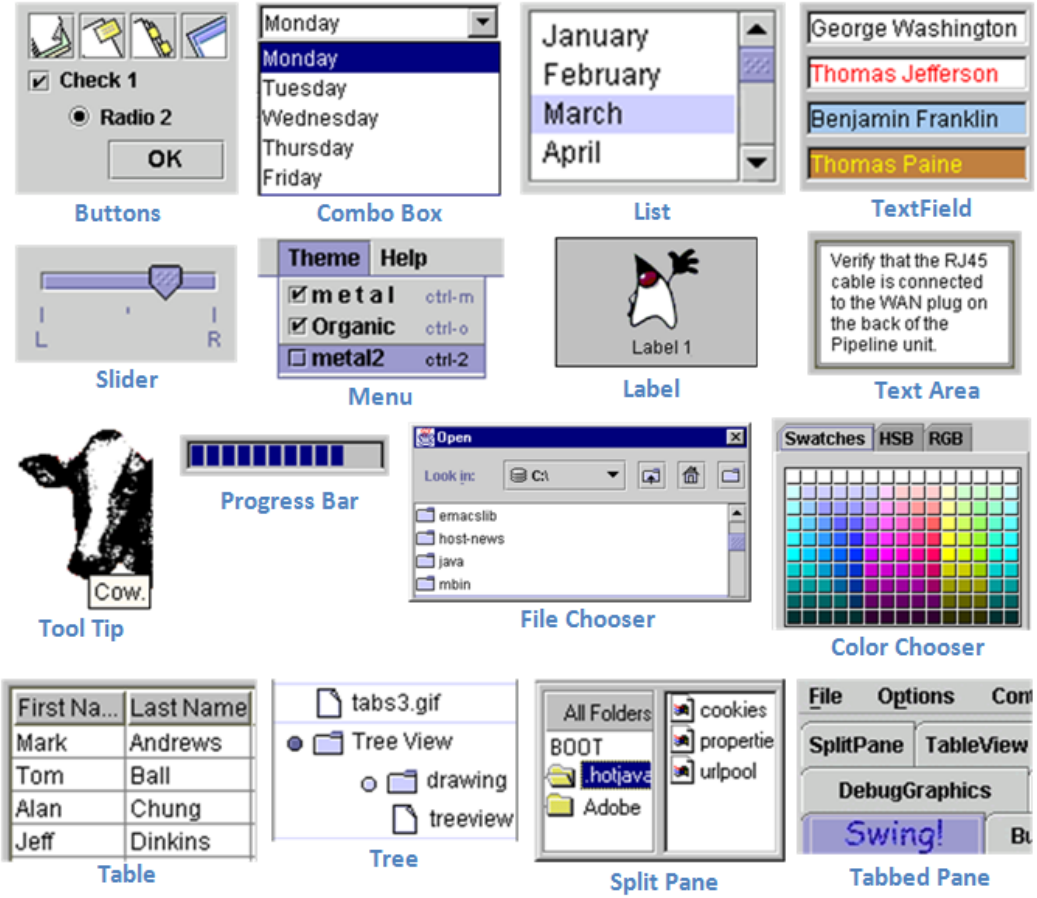
- A container has a so-called *layout manager* to arrange its components. The layout managers provide a level of abstraction to map your user interface on all windowing systems, so that the layout can be *platform-independent*.





Swing

- Swing is part of the so-called "Java Foundation Classes (JFC)" (have you heard of MFC?), which was introduced in 1997 after the release of JDK 1.1.

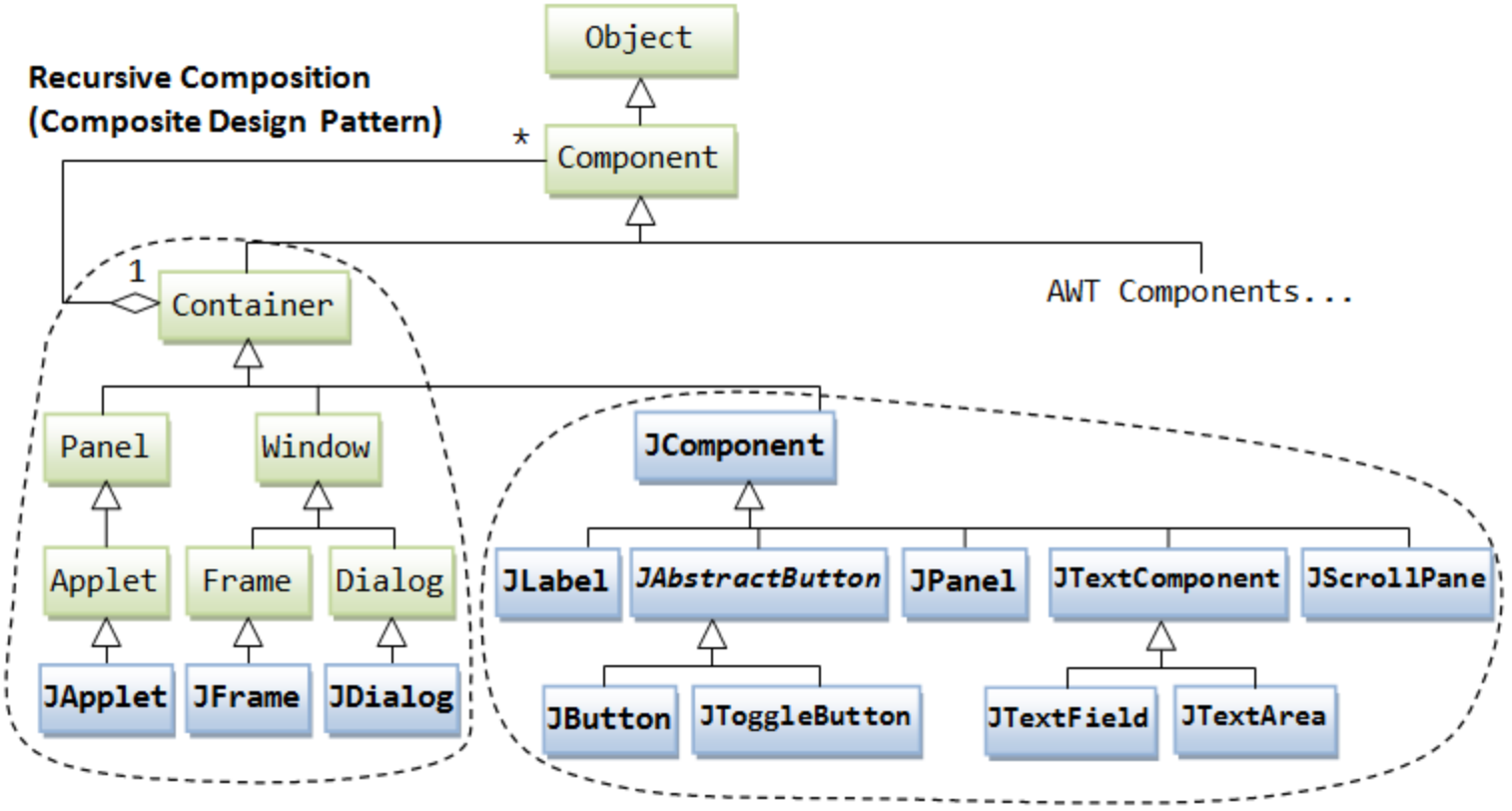


Swing

- Swing components are *lightweight*. The AWT components are *heavyweight* (in terms of system resource utilization).
- Each AWT component has its own opaque native display, and always displays on top of the lightweight components.

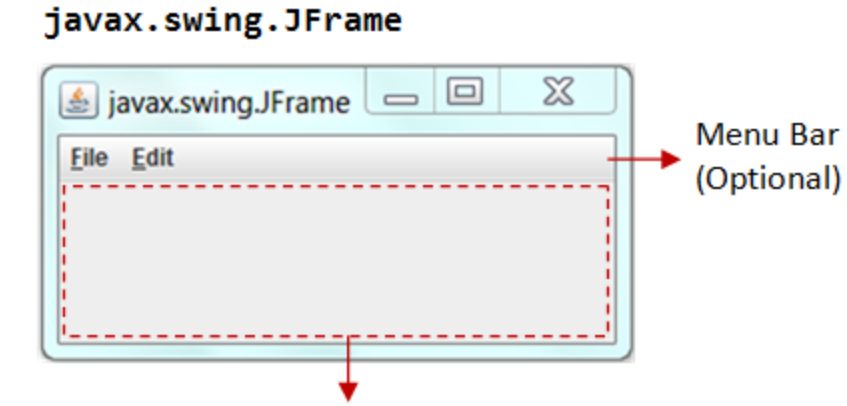
Swing Component Classes

- Compared with the AWT classes (in package java.awt), Swing component classes (in package javax.swing) begin with a prefix "J", e.g., JButton, JTextField, JLabel, JPanel, JFrame, or JApplet.



Swing's Top-Level and Secondary Containers

- Just like AWT application, a Swing application requires a *top-level container*. There are three top-level containers in Swing:
  - JFrame, JDialog, JApplet
- Similarly to AWT, there are *secondary containers* (such as JPanel) which can be used to group and layout relevant components.



The Content-Pane of Swing's Top Level Container

- However, unlike AWT, the JComponents shall not be added onto the top-level container (e.g., JFrame, JApplet) directly because they are lightweight components.
- The JComponents must be added onto the so-called *content-pane* of the top-level container.
- Content-pane is in fact a java.awt.Container that can be used to group and layout components.

- You could:
  - get the content-pane via getContentPane() from a top-level container, and add components onto it.
  - set the content-pane to a JPanel (the main panel created in your application which holds all your GUI components) via JFrame's setContentPane().

Event-Handing 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.

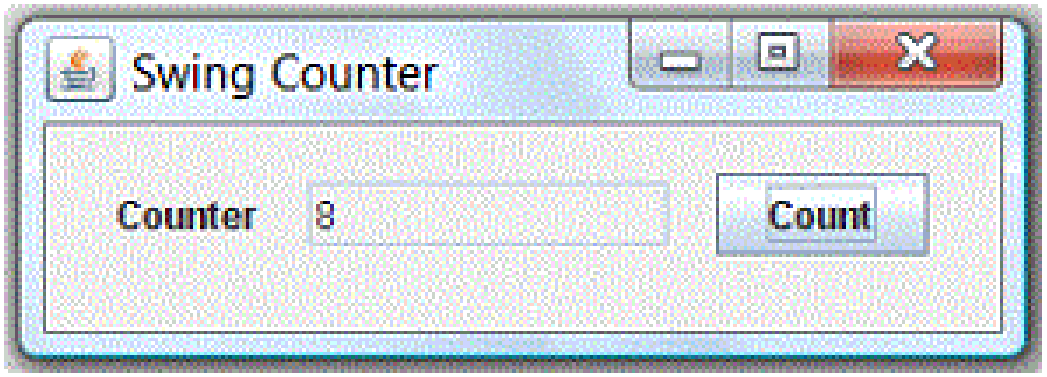
Writing Swing Application

- Use the Swing components with prefix "J" in package javax.swing.

- A top-level container (such as JFrame or JApplet) is needed.
  The JComponents cannot be added directly onto the top-level container. They shall be added onto the *content-pane* of the top-level container. You can retrieve a reference to the content-pane by invoking method getContentPane() from the top-level container, or set the content-pane to the main JPanel created in your program.

Swing Example 1: SwingCounter

- Convert the earlier AWT application example into Swing. Compare the two source files and note the changes (which are highlighted). The display is shown below. Note the differences in *look and feel* between the AWT GUI components and Swing's.



Using Visual GUI Builder - NetBeans/Eclipse

- If you have a complicated layout for your GUI application, you should use a GUI Builder, such as NetBeans or Eclipse to layout your GUI components in a drag-and-drop manner, similar to the popular visual languages such as Visual Basic and Dephi.