GUI Programming

OUTLINE

- **Handling Events**
- **Swing and AWT class hierarchy**
- **GUI Code Example**

Using a GUI Component (Recap)

1. **Create it**
2. **Configure it**
3. **Add children (if container)**
4. **Add to parent (if not JFrame)**
5. **Listen to it**

the order IS important

It's not aabout looking pretty! Lt's all about Listening and Service!

- **The last time we were mostly talking about GUI creation.**
- **But graphical interface is just a front of event-oriented programming… a nice picture we all like. This picture is not an INTERFACE yet… actual communication is still missing.**

Why?

- **Conventional programming**
  - **Approach**
    - Start at main()
    - Continue until end of program or exit()
- **Event-driven programming**
  - **Unable to predict time & occurrence of event**
  - **Approach**
    - Start with main()
    - Build GUI
    - **Await events (& perform associated computation)**

Event-driven Paradigm in Java

- **During implementation**
  - **Implement event listeners for each event**
  - **Usually at least one event listener class per widget**
- **At run time**
  - **Register listener object with widget object**
  - **Java generates event object when events occur**
  - **Java then passes event object to event listener**

Event Handling Model

- **GUIs are event driven**
  - **Generate events when user interacts with GUI**
    - Mouse movements, mouse clicks, typing in a text field, etc.
  - **Event information stored in object that extends `AWTEvent`**

Event Handling Model

- **To process an event**
  - **Register an event listener**
    - Object from a class that implements an event-listener interface (from **`java.awt.event`** or **`javax.swing.event`**)
    - "Listens" for events
  - **Implement event handler**
    - Method that is called in response to an event
    - Each event handling interface has one or more event handling methods that must be defined

Event Handling Model

- **Delegation event model**
  - **Use of event listeners in event handling**
  - **Processing of event delegated to particular object (the listener) in the program**
- **When an event occurs**
  - **GUI component notifies its listeners**
    - Calls listener's event handling method
- **Example:**
  - ***Enter* pressed in a `JTextField`**
  - **Method `actionPerformed` called for registered listener**
    - And then actual responding to the event may start…

In other words…

- **a piece of code (i.e. event handler) is attached to a GUI component**
- **an event handler is called when an event (e.g. a mouse click) is activated / fired**

- **Why "Delegation Event Model"?**
  - **processing of an event is delegated to an object (the listener) in your program**

In other words…

- **event source: a GUI component that *generates / fires* an event**
- **event: a user interaction (e.g. a click on the button)**
- **event listener: an object that has implemented event handlers to *react* to an event**
- **IMPORTANT: event listeners must be registered with an event source in order to receive notification. If you miss that, not error message is generated!**

Registration of an event listener

- **write a class that implements an [event type]Listener interface**

- **create an instance of that class (i.e. an event listener)**

- **register the listener with a GUI component:**

add[event type]Listener (<an event listener>)

E.g.

```
JButton b = new JButton(…)

b.addActionListener(new
                    MyButtonListener());
```

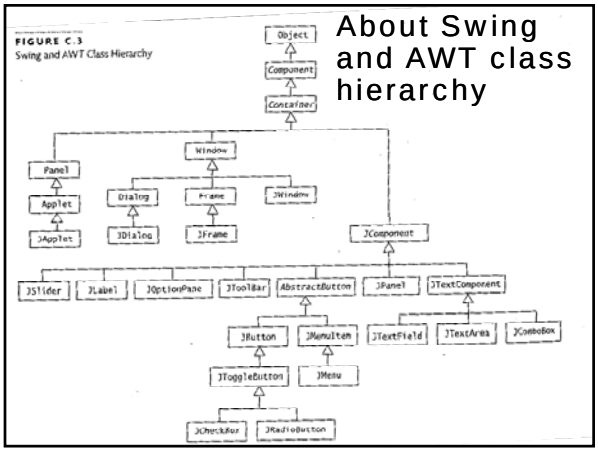
A listener implements ActionListener interface

- sole method:

```
void actionPerformed (ActionEvent e)
```

```
class MyButtonListener implements
   ActionListener {
   void actionPerformed(ActionEvent e) {
      your code to
      -- obtain the details of the event, and
      -- react appropriately
   }
}
```

About Swing and AWT class hierarchy

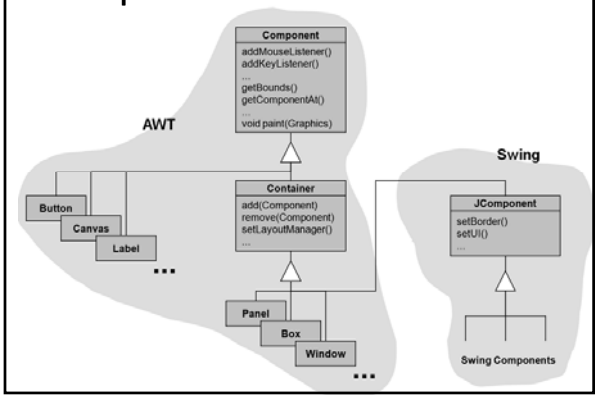


About Swing and AWT class hierarchy

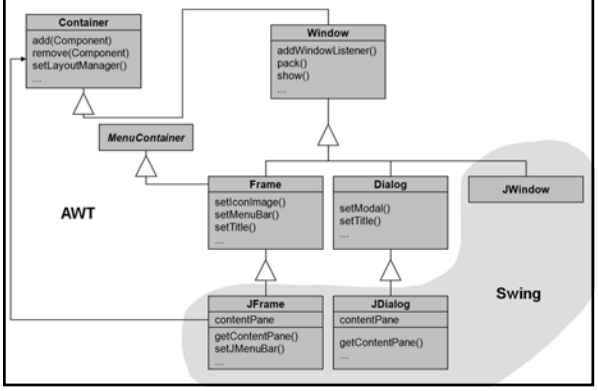
JAVA GUI

- **Basic Java GUI Components**
  - java.awt.Component, java.awt.Container, javax.swing.JComponent, …
- **Event handling**
  - packages java.awt.event and javax.swing.event
- **Creating UI layouts**
  - using and creating layout managers
  - java.awt.LayoutManager and java.awt.LayoutManager2
- **Using resources**
  - using resource bundles for different locales
  - java.util.ResourceBundle

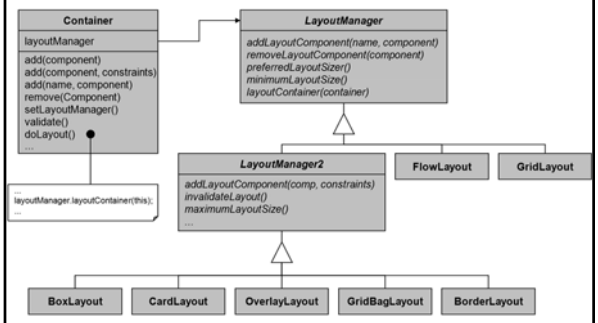
Components



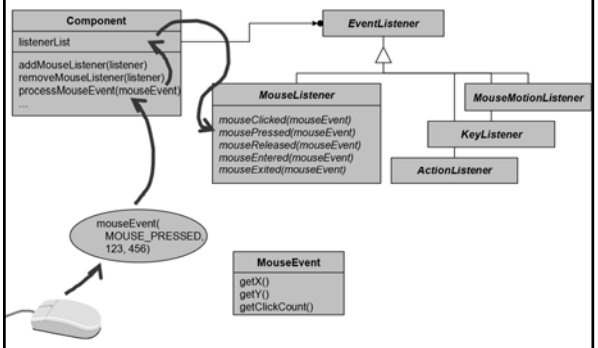
WINDOWS & FRAMES



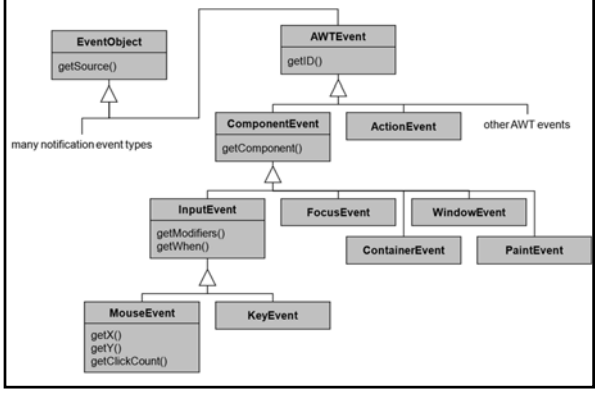
LAYOUT



EVENTS



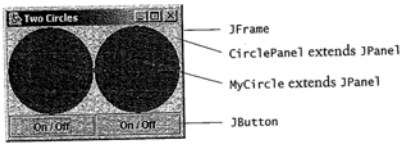
EVENTS – cont.

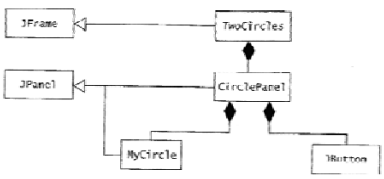


Summary: GUI Programming Steps in Java

- **declare a container and components**
- **add components to one or more containers using a layout manager**
- **register event listener(s) with the components**
- **create event listener method(s)**

Examples





TwoCircles.java

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

/** TwoCircles is a simple event-oriented application that
    displays two circles and two buttons. The buttons are
    placed below the circles. The buttons are labeled on/off.
    When a button is clicked, the state of the circle is toggled.
*/
public class TwoCircles extends JFrame {

    // Constructor
    /** Construct a TwoCircles object. Set the title and
        default close operation. Using a grid layout add
        two CirclePanel objects. Finally, pack the frame
        and set it visible.
    */
    public TwoCircles() {
        super("Two Circles");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        getContentPane().setLayout(new GridLayout(1,2));
        getContentPane().add(new CirclePanel(100));
        getContentPane().add(new CirclePanel(100));
        pack();
        setVisible(true);   // Show the JFrame.
    }

    // Main Method
    /** Instantiate a TwoCircles object.
        @param args Not used. */
    public static void main(String[] args) {
        TwoCircles tc = new TwoCircles();
    }
}
```

CirclePanel.java

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

/** A CirclePanel will contain a circle and a button. */
public class CirclePanel extends JPanel {

    // Data Fields
    /** The button object */
    JButton onOffButton;
    /** The Circle object */
    MyCircle theCircle;

    // Constructor
    /** Construct a CirclePanel object. */
    public CirclePanel(int size) {
        setLayout(new BorderLayout());
        theCircle = new MyCircle(size);
        onOffButton = new JButton("On / Off");
        onOffButton.addActionListener(new ToggleState());
        add(theCircle, BorderLayout.CENTER);
        add(onOffButton, BorderLayout.SOUTH);
    }

    // Inner Class
    /** The action listener for the button. */
    private class ToggleState implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            theCircle.toggleState();
        }
    }
}
```

MyCircle.java

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

/** Class MyCircle is a JPanel that consists of a circle enclosed
    in a square. The square is always displayed, but the circle can
    be turned on and off.
*/
public class MyCircle extends JPanel {

    // Data Fields
    /** The size */
    private int size;
    /** Display state for the circle */
    private boolean showCircle = true;
```

```
    // Constructors
    /** Construct a MyCircle object of the specified size.
        @param size The size of the circle in pixels
    */
    public MyCircle(int size) {
        this.size = size;
        // Encapsulate the object's dimensions in a Dimension obje
        Dimension dims = new Dimension(size, size);
        // Set the object's dimensions.
        setPreferredSize(dims);
        setMaximumSize(dims);
        setMinimumSize(dims);
    }

    /** Toggle the state of the circle. */
    public void toggleState() {
        showCircle = !showCircle;
        repaint();   // Calls paint to redraw the object.
    }

    /** Paint the component when it changes. This method is called
        by the Swing API.
        @param g The graphics object used for painting. */
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.GREEN);
        g.fillRect(0, 0, size, size);
        if (showCircle) {
            g.setColor(Color.RED);
            g.fillOval(0, 0, size, size);
        }
    }
}
```

Questions?