GUI Components

Object Oriented Programming
2016375 - 5

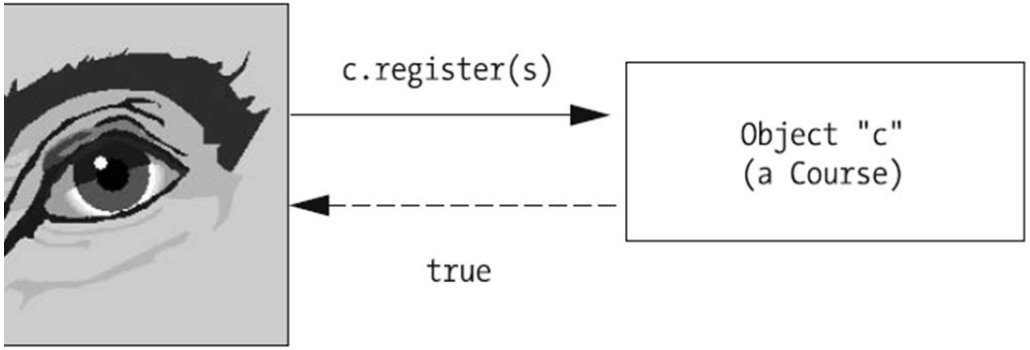
Camilo López

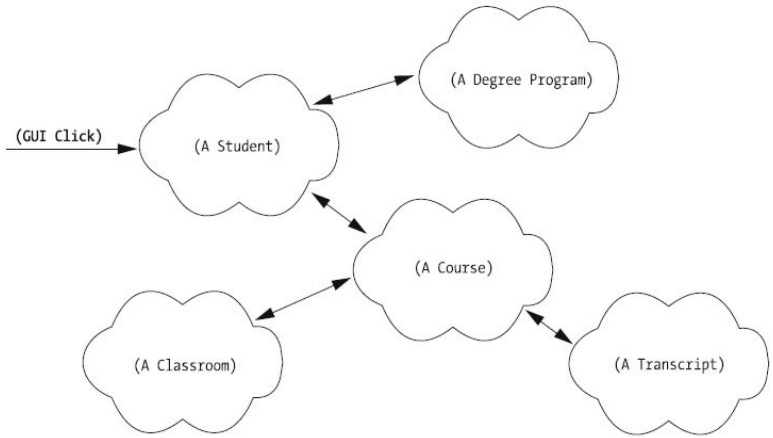
Outline

- Introduction
- Model-View Separation
- Choosing the set of components
- Simple GUI-based I/O

Introduction

- What have we done so far?





There's a communication between the objects in order to comply with a request

Introduction

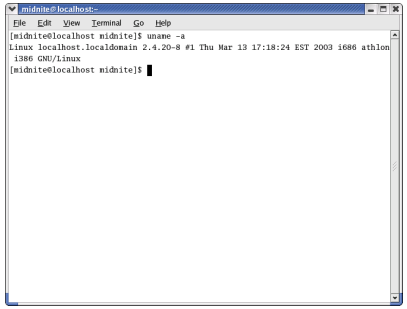
- What have we done so far?

Which are the triggering events?
How do we interact with the system?

Introduction

- What have we done so far?

Files




```
package comp;

public class MainClass {

    /**
     * @param args
     */
    public static void main(String[] args) {
        ComplexNumber c1 = new ComplexNumber(2,1);
        ComplexNumber c2 = new ComplexNumber(2,2);

        ComplexOperations co = new ComplexOperations();
        System.out.println(co.addition(c1, c2));
        System.out.println(co.subtraction(c1, c2));
        System.out.println(co.multiplication(c1, c2));
    }
}
```

Modifying Client Code

Which are the triggering events?
How do we interact with the system?

Introduction

- A **graphical user interface (GUI)** presents a user-friendly mechanism for interacting with an application.

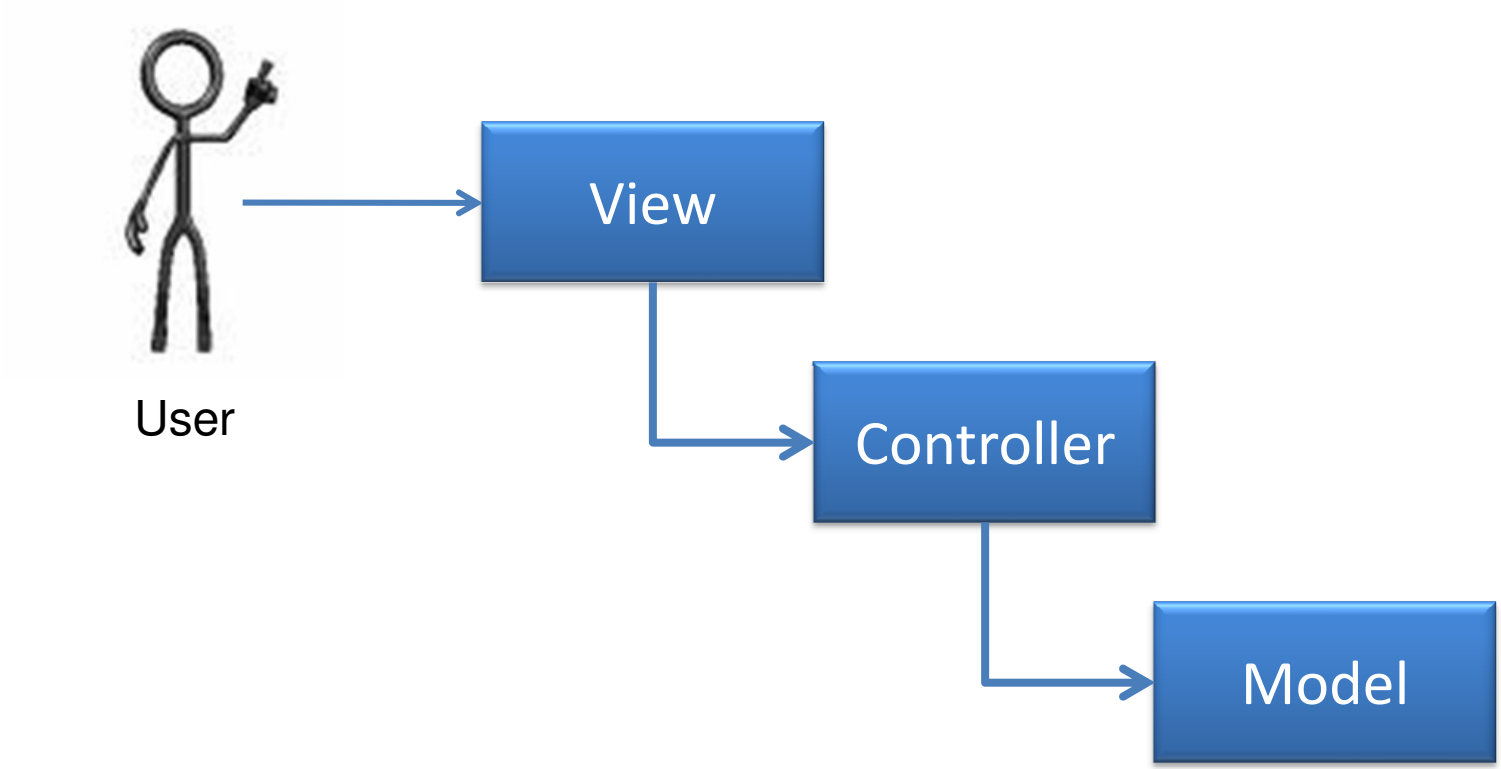
- a GUI enables us to ***create*** and ***interact with*** model objects. For instance:
  - Instantiating objects
  - Invoking their methods
  - Changing their state

Introduction

- A **GUI component is an object** with which the user interacts via the mouse, the keyboard or another form of input, such as voice recognition.
  - Are declared as classes
  - Are instantiated via the new keyword, invoking an appropriate constructor
  - Have attributes and methods
  - Communicate via messages
  - Are requested to provide services by invoking their methods via dot notation
  - Participate in inheritance hierarchies
  - Are referenced by reference variables whenever we wish to maintain named “handles” on them

Model–View Separation

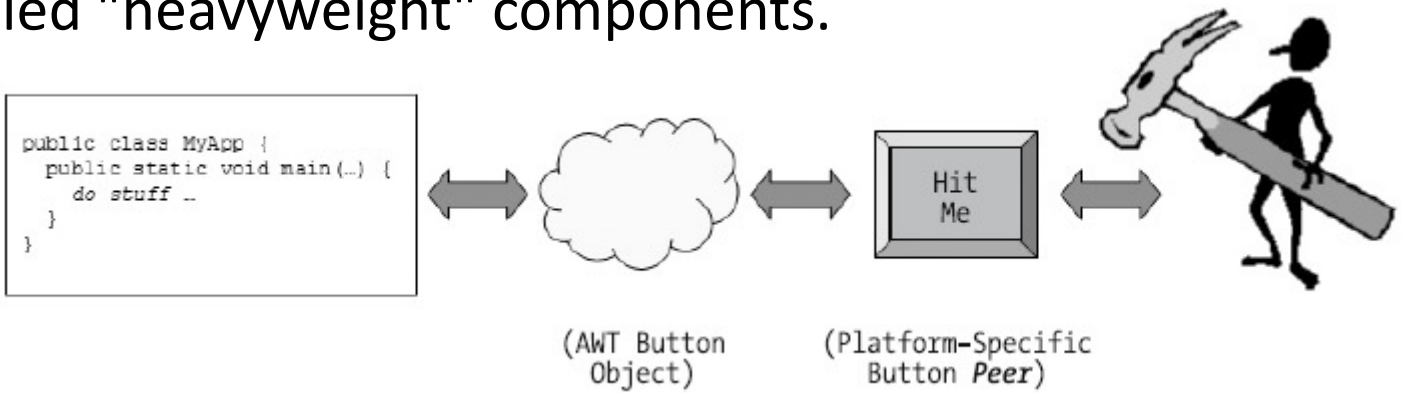
Model–View–Controller (MVC) design pattern



Choosing the set of components

AWT vs Swing

- AWT stands for Abstract Window Toolkit
- It is a portable GUI library for stand-alone applications and/or applets.
- Provides the connection between your application and the **native GUI.**
- Its components depend on native code counterparts (called peers) to handle their functionality. This is why they are often called "heavyweight" components.



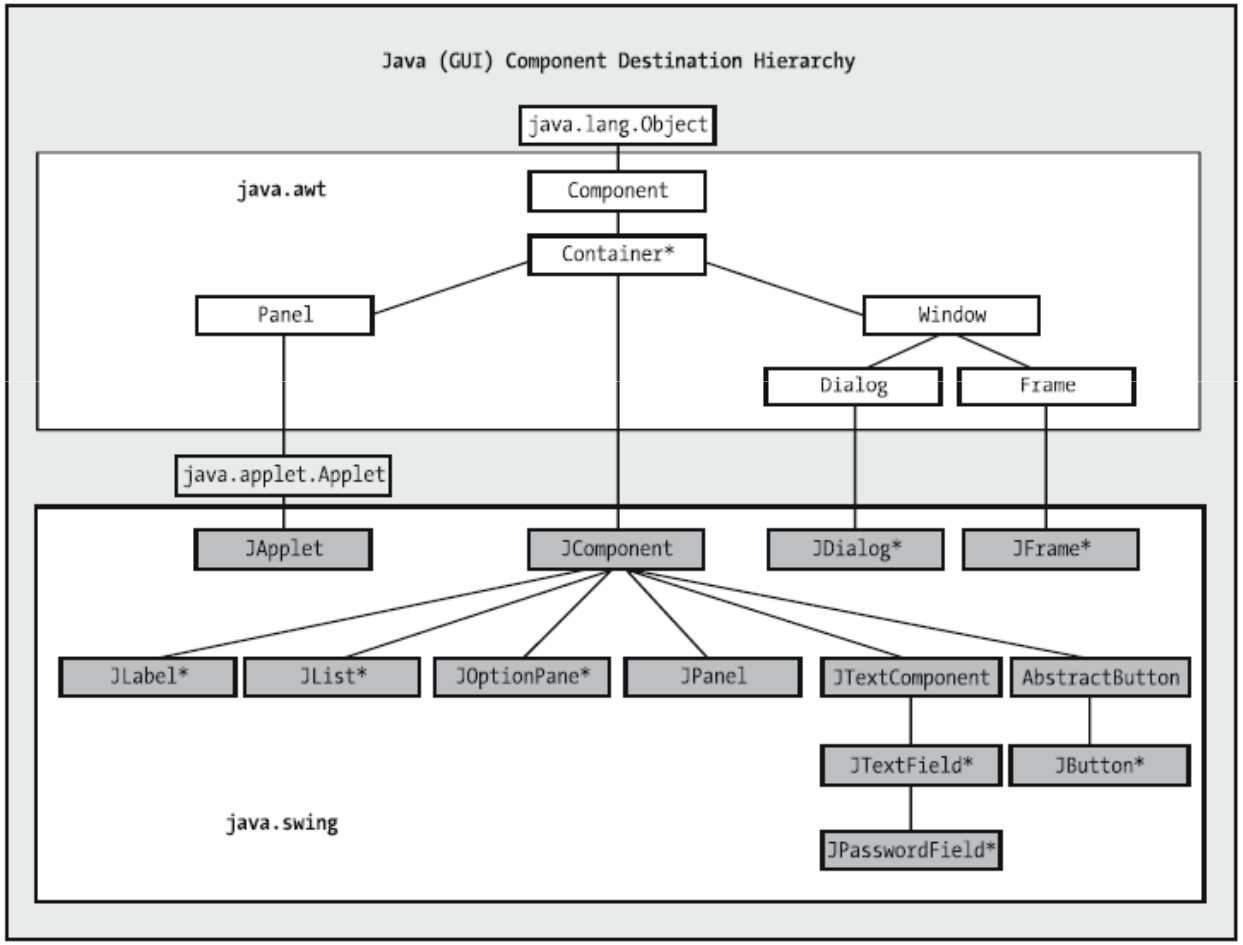
Choosing the set of components

AWT vs Swing

- Swing implements a set of GUI components that build on AWT technology and provide a pluggable look and feel.
- Swing is implemented entirely in the Java programming language, and is based on the JDK 1.1 Lightweight UI Framework.
- Its components do not depend on peers to handle their functionality. This is why they are often called "lightweight" components.
- Most AWT components named *"xxx"* have Swing counterparts named *"Jxxx".*

Choosing the set of components

AWT vs Swing



Simple GUI-based I/O

JOptionPane

```
import javax.swing.JOptionPane;

public class Addition{
    public static void main( String args[] ){
        String fNumber = JOptionPane.showInputDialog("Enter first integer");
        String sNumber = JOptionPane.showInputDialog("Enter second integer");

        int number1 = Integer.parseInt(fNumber );
        int number2 = Integer.parseInt(sNumber );

        int sum = number1 + number2;

        JOptionPane.showMessageDialog( null, "The sum is " + sum,
            "Sum of Two Integers“, JOptionPane.PLAIN_MESSAGE );
    }
}
```

yes, it’s javax

Simple GUI-based I/O

JOptionPane

- one-line calls to one of the static showXxxDialog methods shown below:

Method Nameº	**Description**
showConfirmDialog	Asks a confirming question, like yes/no/cancel.
showInputDialog	Prompt for some input.
showMessageDialog	Tell the user about something that has happened.
showOptionDialog	The Grand Unification of the above three.

For more info:

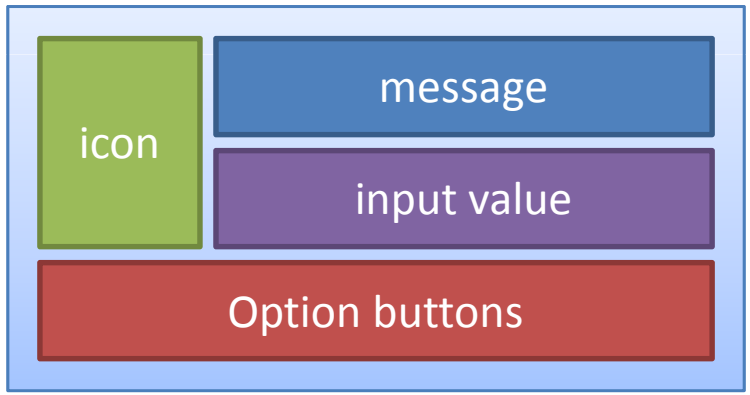
http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/JOptionPane.html

Simple GUI-based I/O

JOptionPane

- Constructor

```
JOptionPane(Object message, int messageType, int optionType, Icon icon,
Object[] options, Object initialValue)
```

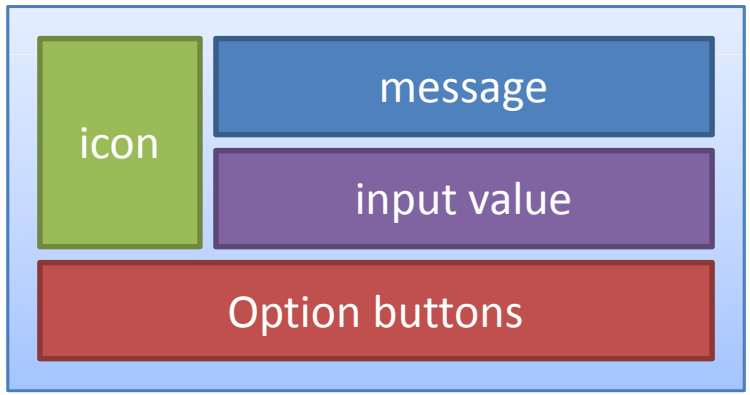


Simple GUI-based I/O

JOptionPane

- Constructor

```
JOptionPane(Object message, int messageType, int optionType, Icon icon,
Object[] options, Object initialValue)
```



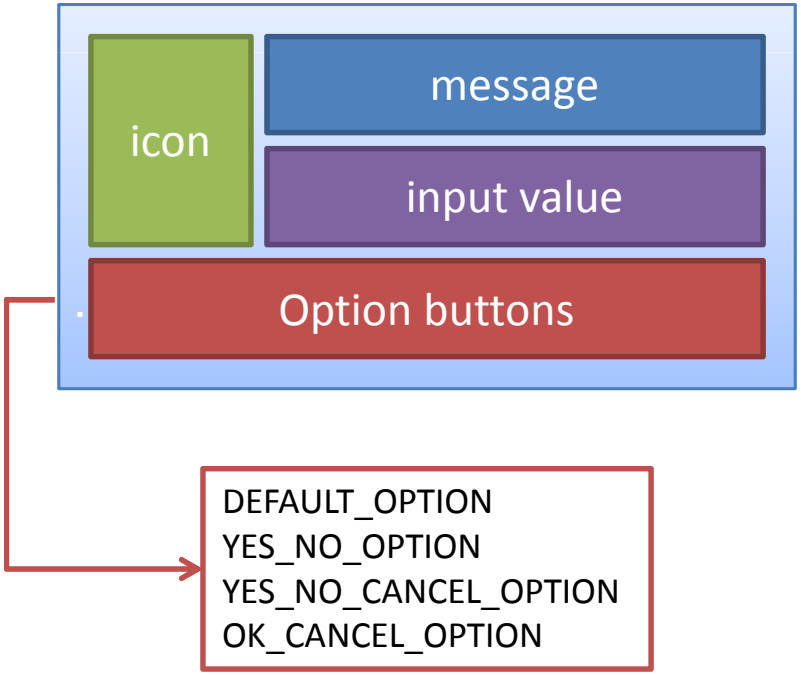
This refers to a custom icon

Simple GUI-based I/O

JOptionPane

- Constructor

```
JOptionPane(Object message, int messageType, int optionType, Icon icon,
Object[] options, Object initialValue)
```

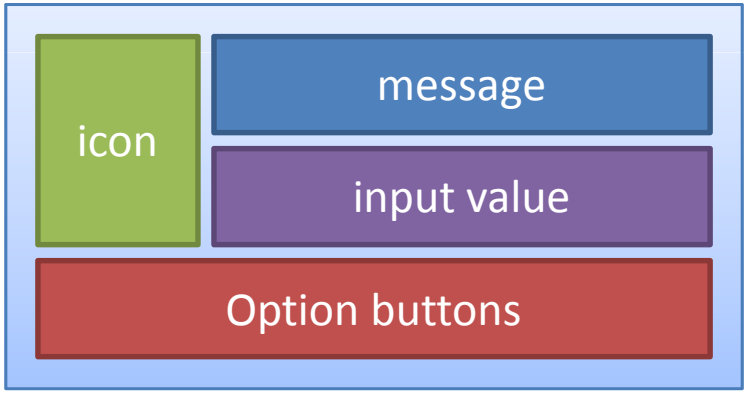


Simple GUI-based I/O

JOptionPane

- Constructor

```
JOptionPane(Object message, int messageType, int optionType, Icon icon,
Object[] options, Object initialValue)
```



Customize buttons

```
Object[] options = {"Yes, please", "No, thanks“};
```

Simple GUI-based I/O

JOptionPane – Message Dialog Constants

- All message dialog types except PLAIN_MESSAGE display an icon to the left of the message.
- These icons provide a visual indication of the message's importance to the user.

Message dialog type	Icon	Description
ERROR_MESSAGE		A dialog that indicates an error to the user.
INFORMATION_MESSAGE		A dialog with an informational message to the user.
WARNING_MESSAGE		A dialog warning the user of a potential problem.
QUESTION_MESSAGE		A dialog that poses a question to the user. This dialog normally requires a response, such as clicking a **Yes** or a **No** button.
PLAIN_MESSAGE	no icon	A dialog that contains a message, but no icon.

Simple GUI-based I/O

JOptionPane

```
import javax.swing.JOptionPane;

public class Addition{
    public static void main( String args[] ){
        String fNumber = JOptionPane.showInputDialog("Enter first integer");
        String sNumber = JOptionPane.showInputDialog("Enter second integer");

        int number1 = Integer.parseInt(fNumber );
        int number2 = Integer.parseInt(sNumber );

        int sum = number1 + number2;

        JOptionPane.showMessageDialog( null, "The sum is " + sum,
            "Sum of Two Integers“, JOptionPane.PLAIN_MESSAGE );
    }
}
```

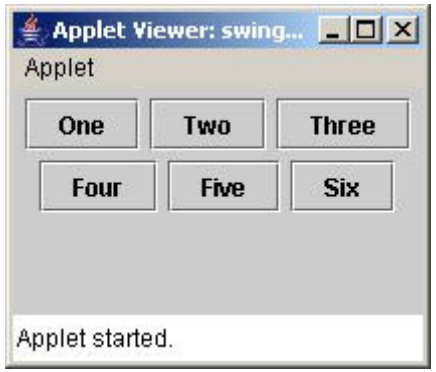
Message Dialog Constants

Displaying Text and Images in a Window

JLabel

```
import java.awt.FlowLayout; // specifies how components are arranged
import javax.swing.JFrame; // provides basic window features
import javax.swing.JLabel; // displays text and images
import javax.swing.SwingConstants; // common constants used with Swing
import javax.swing.Icon; // interface used to manipulate images
import javax.swing.ImageIcon; // loads images
```

java.awt.FlowLayout arranges components from left-to-right and top-to-bottom, centering components horizontally with a five pixel gap between them.



Displaying Text and Images in a Window

JLabel

```
import …

public class LabelFrame extends JFrame{
    private JLabel label1;

    public LabelFrame(){
        super( "Testing JLabel" );
        setLayout( new FlowLayout() ); // set frame layout

        label1 = new JLabel( "Label with text" );
        label1.setToolTipText( "This is label1" );
        add( label1 );
    }
}
```

LabelFrame Components

Displaying Text and Images in a Window

JLabel

```
import …

public class LabelFrame extends JFrame{
    private JLabel label1;

    public LabelFrame(){
        super( "Testing JLabel" );
        setLayout( new FlowLayout() ); // set frame layout

        label1 = new JLabel( "Label with text" );
        label1.setToolTipText( "This is label1" );
        add( label1 );
    }
}
```

JLabel constructor with a string argument

Displaying Text and Images in a Window

JLabel

```
import …

public class LabelFrame extends JFrame{
    private JLabel label1;

    public LabelFrame(){
        super( "Testing JLabel" );
        setLayout( new FlowLayout() ); // set frame layout

        label1 = new JLabel( "Label with text" );
        label1.setToolTipText( "This is label1" );
        add( label1 );
    }
}
```

add label1 to Jframe

Displaying Text and Images in a Window

JLabel

```
import …

public class LabelFrame extends JFrame{
    private JLabel label2;

    public LabelFrame(){
        super( "Testing JLabel" );
        setLayout( new FlowLayout() ); // set frame layout

        Icon img = new ImageIcon( "aireplanesafety.png" );
        label2 = new JLabel( "Label with text and icon", img,
            SwingConstants.LEFT );
        label2.setToolTipText( "This is label2" );
        add( label2 );
    }
}
```

JLabel constructor with string, Icon and alignment arguments

Displaying Text and Images in a Window

JLabel

```
import …

public class LabelFrame extends JFrame{
    private JLabel label3;

    public LabelFrame(){
        super( "Testing JLabel" );
        setLayout( new FlowLayout() ); // set frame layout

        label3 = new JLabel();
        label3.setText( "Label with icon and text at bottom" );
        label3.setIcon( img );
        label3.setHorizontalTextPosition( SwingConstants.CENTER );
        label3.setVerticalTextPosition( SwingConstants.BOTTOM );
        add( label3 );
    }
}
```

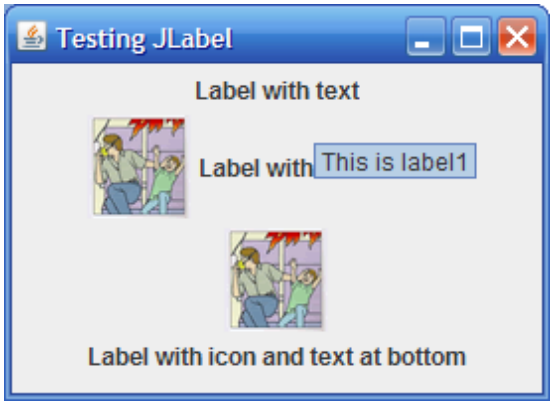
JLabel constructor with no arguments

Displaying Text and Images in a Window

JLabel

```
import javax.swing.JFrame;

public class LabelTest{
    public static void main( String args[] ){
        LabelFrame labelFrame = new LabelFrame();
        labelFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        labelFrame.setSize( 275, 200 );
        labelFrame.setVisible( true );
    }
}
```

By default, closing a window simply hides the window. We would like the application to terminate when the user closes the LabelFrame window.



Java Event Handling

- GUI events are generated when the user interacts with a component on the GUI.
- ***events are objects!***
- When a GUI component is created, it ***automatically*** has the ability to generate events whenever a user interacts with it
- What we ***do need to explicitly deal with***, however, is programming how the GUI should react to the (subset of) events that we are interested in.

Java Event Handling

Handling Events - Two Steps

- ***Step 1:***
  Design and instantiate a special type of object called a ***listener*** that is capable of "hearing" and responding to a particular type of event as generated by a particular component.

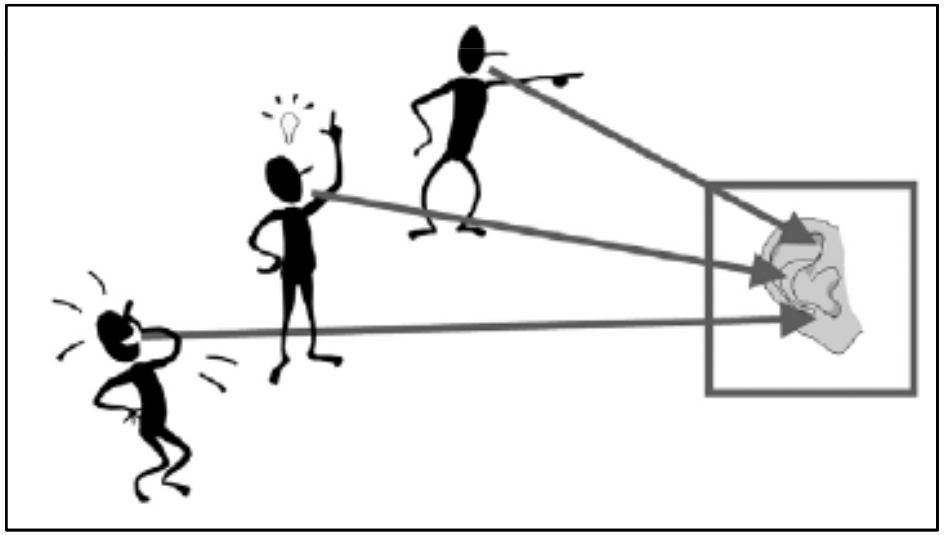
  program its methods with whatever behavior that we want it to react with when it hears such an event.

- For example,
  to respond to clicks of a JButton, which generate ActionEvents, we'd create an ActionListener, a type of listener that is capable of listening for and responding to ActionEvents specifically.

Java Event Handling

Handling Events - Two Steps

- ***Step 2:***
  Register the listener object that we've created with the specific Component object(s) that we want that listener to listen to.



A single listener can be registered to listen to one or more component objects

Java Event Handling

Events

- Whenever a user interacts with a component on a Java GUI, the component generates ***numerous*** events of various types. if the **cursor is moved over a JButton**; the **Jbutton is clicked**; and the **cursor is then moved off the button**, the following events are generated:

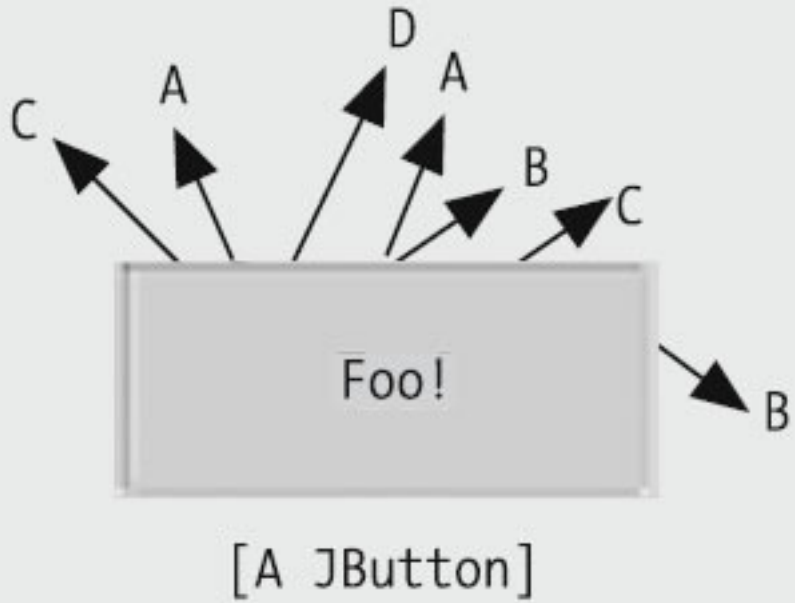
Java Event Handling

Events

- Whenever a user interacts with a component on a Java GUI, the component generates ***numerous*** events of various types. if the **cursor is moved over a JButton**; the **Jbutton is clicked**; and the **cursor is then moved off the button**, the following events are generated:
  - **A "mouse entered" event** is generated when the cursor first enters the boundaries of the JButton.
  - Numerous **"mouse moved" events** are generated as the cursor moves from pixel to pixel within the boundaries of the JButton.
  - **A "mouse pressed" event** is generated when the mouse button is depressed while the cursor is over the JButton.
  - Three events—a **"mouse released" event, a "mouse clicked" event, and an "action" event**—are generated when the mouse button is released.

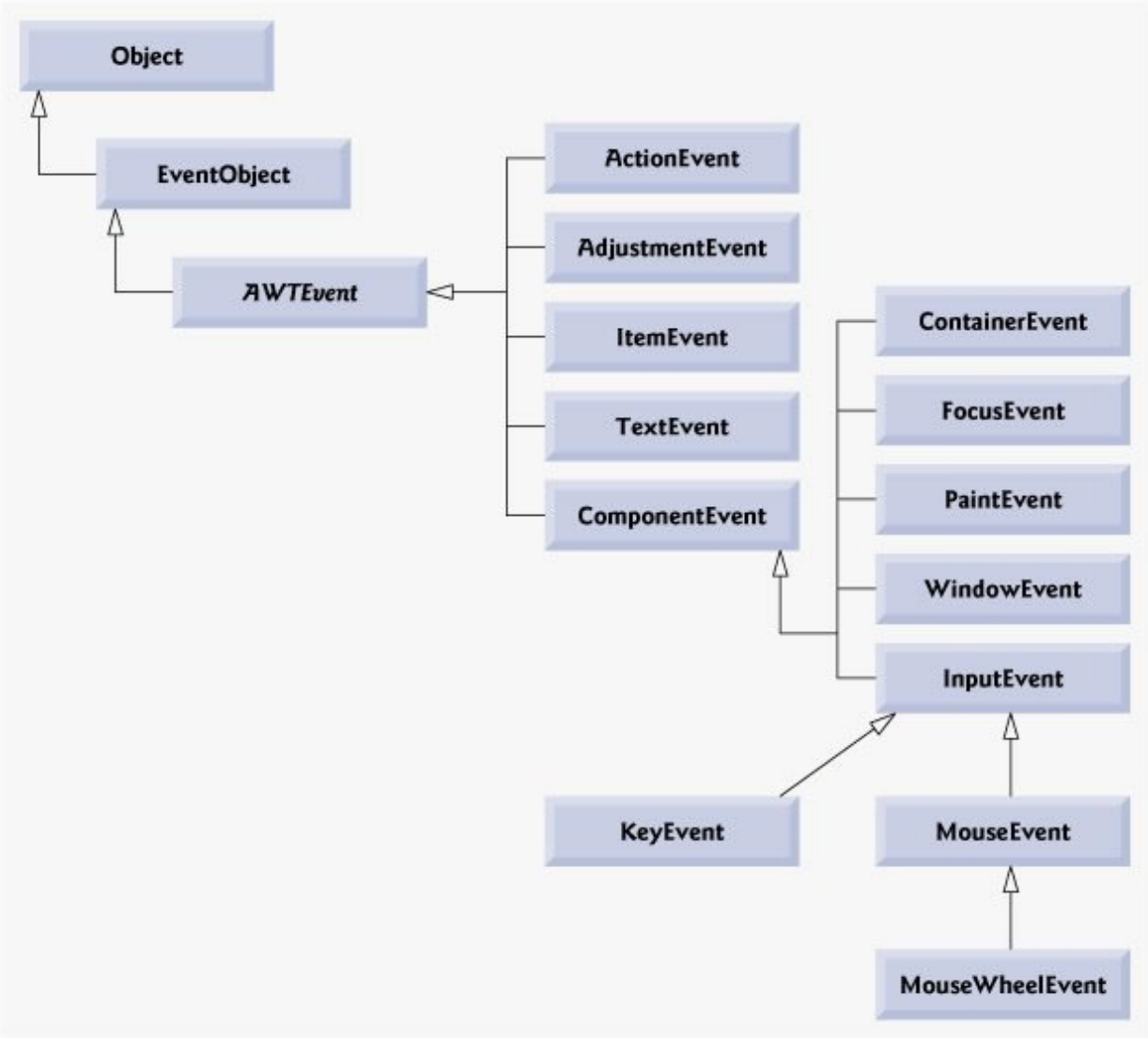
Java Event Handling

When a user interacts with a GUI component such as a JButton, various types of events are generated.

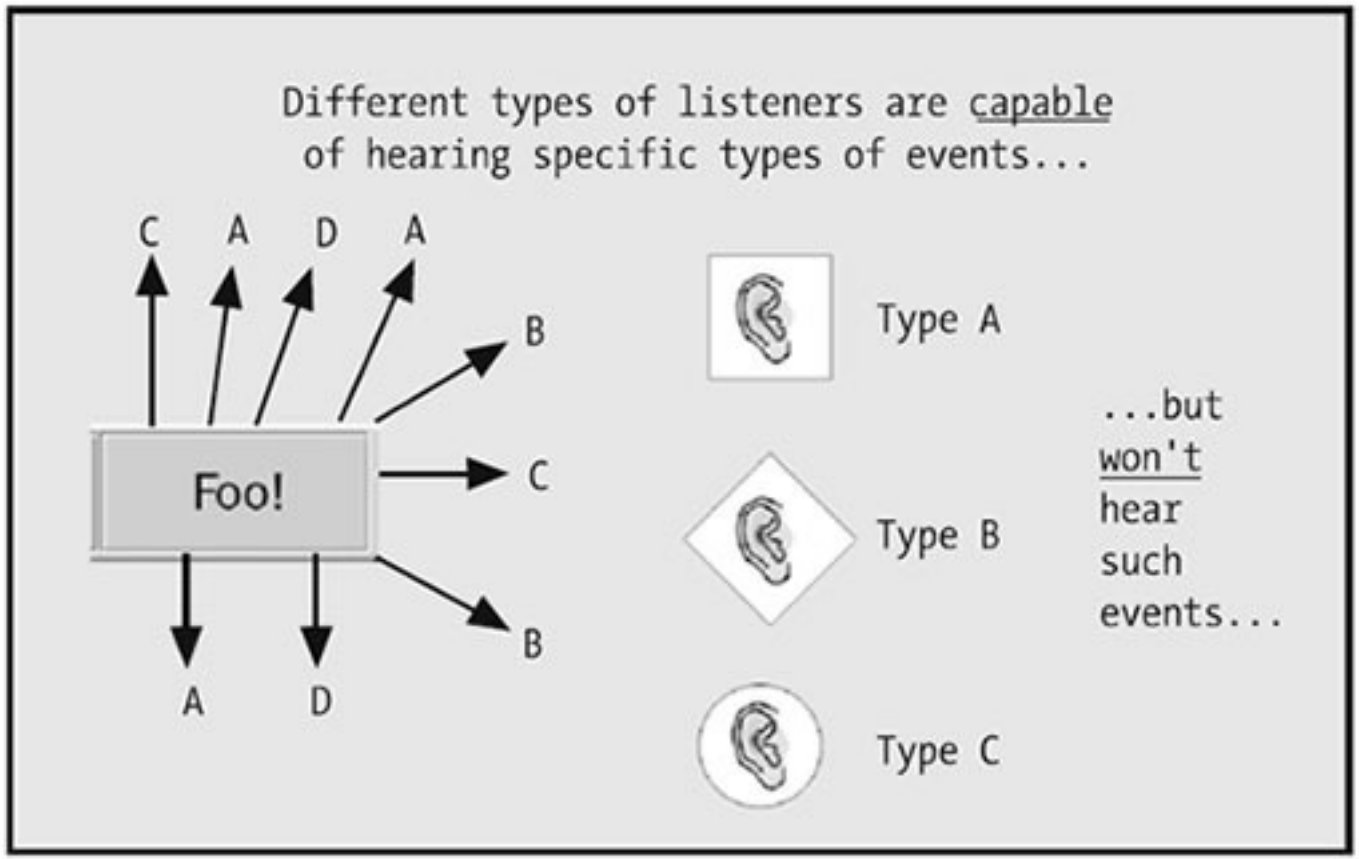


If no listeners have been created these events are not 'heard' or responded to, however.

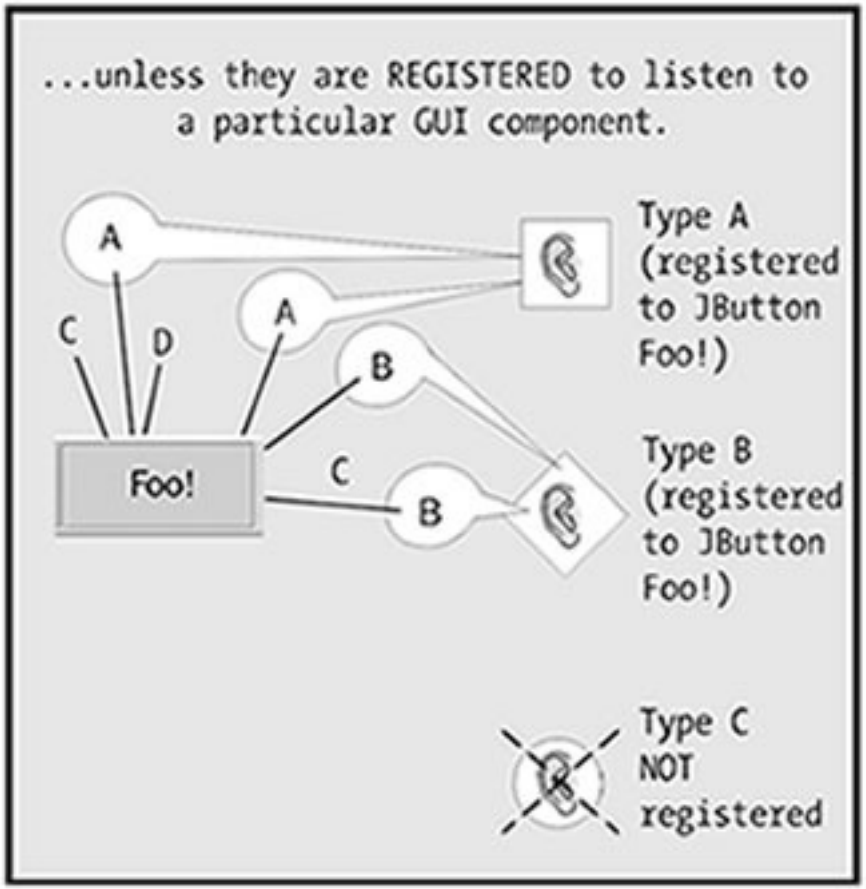
Java Event Handling



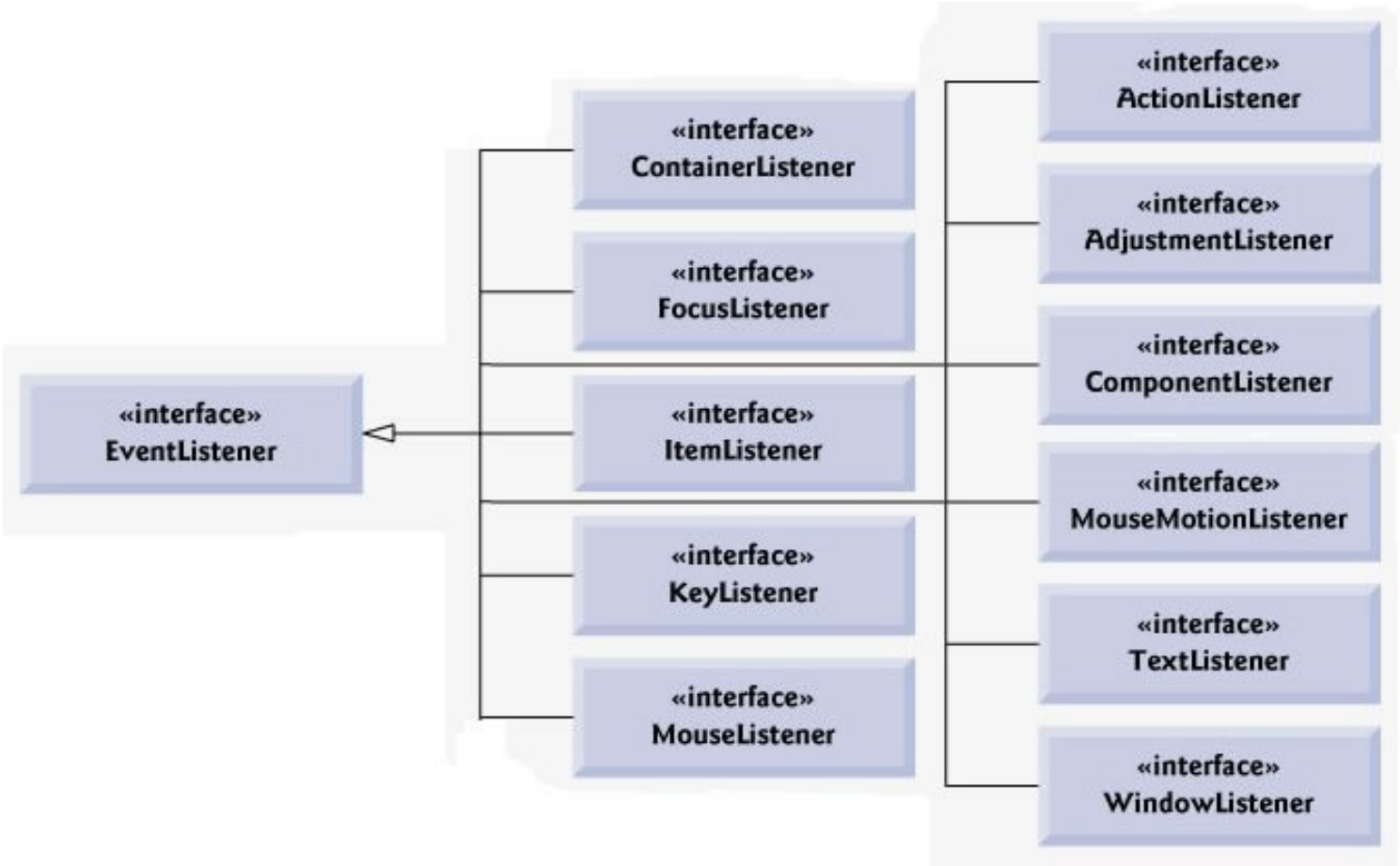
Java Event Handling



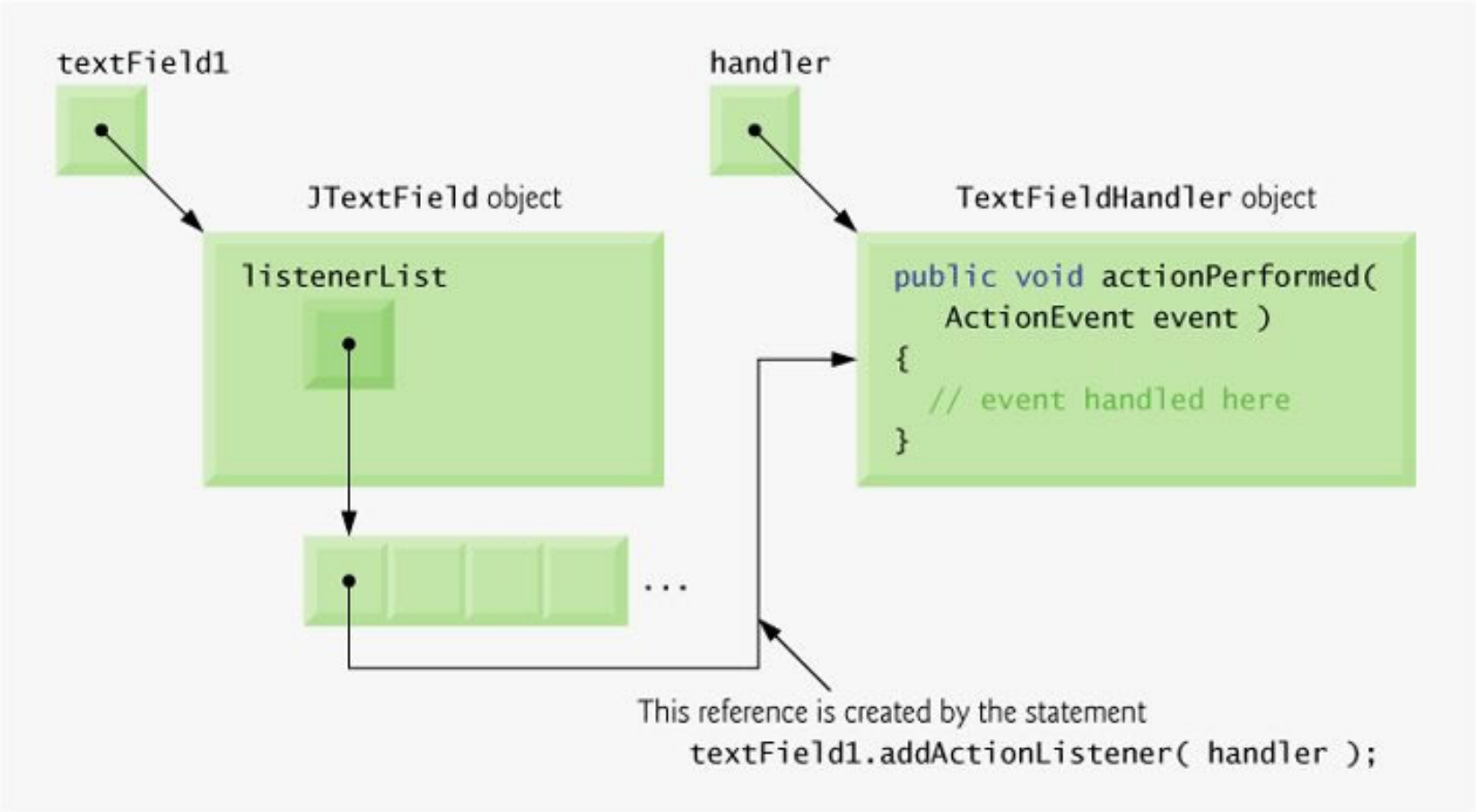
Java Event Handling



Java Event Handling



Java Event Handling



Java Event Handling

Summarizing

3 parts of the event-handling mechanism:

- **Event Source** → GUI component
- **Event Object** → Encapsulates info about the event
- **Event Listener** → Is notified by the event source when an event occurs
  → One of its methods executes to respond to the event

Java Event Handling

Using a Nested Class to Implement an Event Handler

- Inner Classes

```
public class OuterClass {
    // Declare the outer class's attributes -- details omitted.

    class InnerClass {
        // Declare the inner class's attributes and methods ...
    } // end of inner class

    public void someMethodOfOuterClass() {
        InnerClass x = new InnerClass();
        // etc.
    }

    // Declare other methods of the outer class ... details omitted.
}
```

Declare an inner class within the BODY of the OUTER class.

Java Event Handling

Using a Nested Class to Implement an Event Handler

- Inner Classes

```
public class OuterClass {
    // Declare the outer class's attributes -- details omitted.

    class InnerClass {
        // Declare the inner class's attributes and methods ...
    } // end of inner class

    public void someMethodOfOuterClass() {
        InnerClass x = new InnerClass();
        // etc.
    }

    // Declare other methods of the outer class ... details omitted.
}
```

An inner class is allowed to directly access its top-level class's variables and methods, even if they are private.

Java Event Handling

Using a Nested Class to Implement an Event Handler

- Inner Classes

```
public class OuterClass {
    // Declare the outer class's attributes -- details omitted.

    class InnerClass {
        // Declare the inner class's attributes and methods ...
    } // end of inner class

    public void someMethodOfOuterClass() {
        InnerClass x = new InnerClass();
        // etc.
    }

    // Declare other methods of the outer class ... details omitted.
}
```

We may instantiate an object of type InnerClass within any of OuterClass's methods.

Java Event Handling

Using a Nested Class to Implement an Event Handler

- Inner Classes

```
public class OuterClass {
    // Declare the outer class's attributes -- details omitted.

    class InnerClass {
        // Declare the inner class's attributes and methods ...
    } // end of inner class

    public void someMethodOfOuterClass() {
        InnerClass x = new InnerClass();
        // etc.
    }

    // Declare other methods of the outer class ... details omitted.
}
```

try to reference the symbol "InnerClass" from anywhere else in our application

compiler ERROR!!!

Text Fields and Event Handling

JTextField - JPasswordField

- Inner Classes

```
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.JPasswordField;
import javax.swing.JOptionPane;
```

The event we want to handle

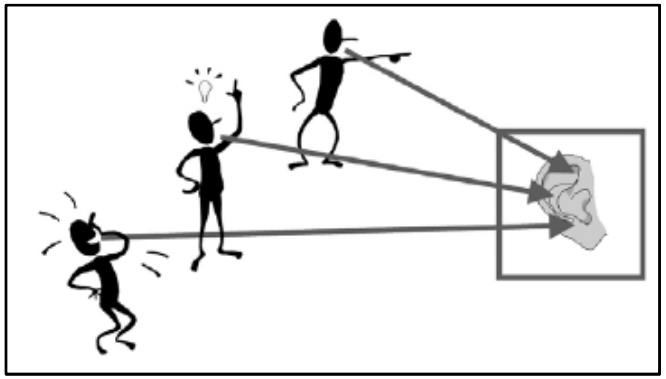
Text Fields and Event Handling

JTextField - JPasswordField

- Inner Classes

```
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.JPasswordField;
import javax.swing.JOptionPane;
```

The responsible for "hearing" and responding to a particular type of event



Text Fields and Event Handling

JTextField - JPasswordField

```
import…
public class TextFieldFrame extends JFrame{
    private JTextField textField1;
    public TextFieldFrame(){
        super( "Testing JTextField and JPasswordField" );
        setLayout( new FlowLayout() );

        textField1 = new JTextField( 10 );
        add( textField1 );

        TextFieldHandler handler = new TextFieldHandler();
        textField1.addActionListener( handler );
    }

    private class TextFieldHandler implements ActionListener{…}
}
```

The inner class is private because it will be used only to create event handlers for the text fields in top-level class TextFieldFrame

Text Fields and Event Handling

JTextField - JPasswordField

```
import…
public class TextFieldFrame extends JFrame{
    private JTextField textField1;
    public TextFieldFrame(){
        super( "Testing JTextField and JPasswordField" );
        setLayout( new FlowLayout() );

        textField1 = new JTextField( 10 );
        add( textField1 );

        TextFieldHandler handler = new TextFieldHandler();
        textField1.addActionListener( handler );
    }

    private class TextFieldHandler implements ActionListener{…}
}
```

This method will be called automatically when the user presses Enter in any of the GUI's text fields.

Text Fields and Event Handling

JTextField - JPasswordField

```
    private class TextFieldHandler implements ActionListener{
        public void actionPerformed( ActionEvent event ){
            String string = ""; // declare string to display
            if ( event.getSource() == textField1 ){
                string = String.format("textField1: %s", event.getActionCommand());
            }

            //…
            else if ( event.getSource() == passwordField ){
                string = String.format( "passwordField: %s",
                    new String( passwordField.getPassword() ) );
            }

            JOptionPane.showMessageDialog( null, string );
        }
    } // end private inner class TextFieldHandler
} // end class TextFieldFrame
```

The event we want to handle

Text Fields and Event Handling

JTextField - JPasswordField

```
private JTextField textField2; // text field constructed with text
private JTextField textField3; // text field with text and size
private JPasswordField passwordField; // password field with text

textField2 = new JTextField( "Enter text here" );
add( textField2 ); // add textField2 to JFrame

textField3 = new JTextField( "Uneditable text field", 21 );
textField3.setEditable( false ); // disable editing
add( textField3 ); // add textField3 to JFrame

passwordField = new JPasswordField( "Hidden text" );
add( passwordField ); // add passwordField to Jframe

TextFieldHandler handler = new TextFieldHandler();
textField2.addActionListener( handler );
textField3.addActionListener( handler );
passwordField.addActionListener( handler );
```

Buttons and Event Handling

JButtons

```
import…
public class ButtonFrame extends JFrame{
    private JButton plainJButton;
    private JButton fancyJButton;

    public ButtonFrame(){
        super( "Testing Buttons" );
        setLayout( new FlowLayout() );

        plainJButton = new JButton( "Plain Button" );
        add( plainJButton ); // add plainJButton to JFrame

        Icon btn1 = new ImageIcon( “btn1.gif“ );
        Icon btn2 = new ImageIcon("btn2.gif"  );
        fancyJButton = new JButton( "Fancy Button", btn1 );
        fancyJButton.setRolloverIcon( btn2 );
        add( fancyJButton );
```

Buttons and Event Handling

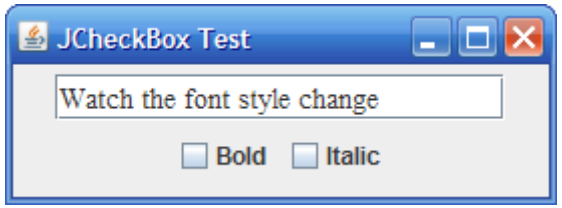
JButtons

```
        ButtonHandler handler = new ButtonHandler();
        fancyJButton.addActionListener( handler );
        plainJButton.addActionListener( handler );
    }

    private class ButtonHandler implements ActionListener{
        public void actionPerformed( ActionEvent event ){
            JOptionPane.showMessageDialog( ButtonFrame.this, String.format(
                "You pressed: %s", event.getActionCommand() ) );
        }
    }
}
```

Buttons that Maintain State

JCheckBox - JRadioButton

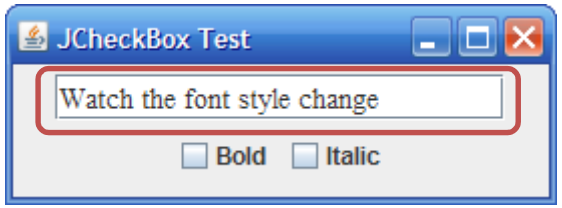
```
import…
public class ButtonFrame extends JFrame{
    private JTextField textField;
    private JCheckBox boldJCheckBox;
    private JCheckBox italicJCheckBox;

    public CheckBoxFrame(){
        super( "JCheckBox Test" );
        setLayout( new FlowLayout() ); // set frame layout

    // set up JTextField and set its font

    // set up the checkBoxes

    // register listeners for JCheckBoxes

} // end CheckBoxFrame constructor
```



Buttons that Maintain State

JCheckBox - JRadioButton

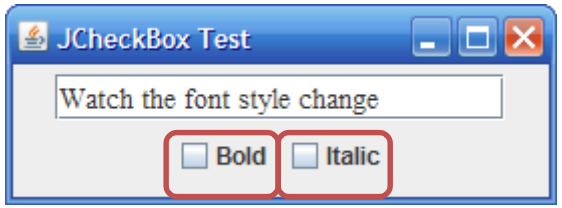
```
import…
public class ButtonFrame extends JFrame{
    private JTextField textField;
    private JCheckBox boldJCheckBox;
    private JCheckBox italicJCheckBox;

    public CheckBoxFrame(){
        super( "JCheckBox Test" );
        setLayout( new FlowLayout() ); // set frame layout

    // set up JTextField and set its font

    // set up the checkBoxes

    // register listeners for JCheckBoxes

} // end CheckBoxFrame constructor
```



Buttons that Maintain State

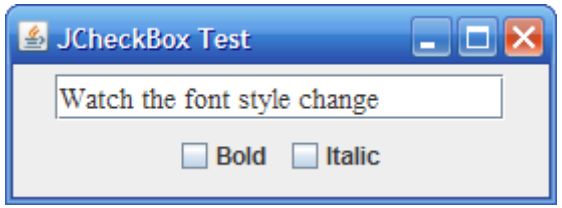
JCheckBox - JRadioButton

```
import…
public class ButtonFrame extends JFrame{
    private JTextField textField;
    private JCheckBox boldJCheckBox;
    private JCheckBox italicJCheckBox;

    public CheckBoxFrame(){
        super( "JCheckBox Test" );
        setLayout( new FlowLayout() ); // set frame layout

    // set up JTextField and set its font

    // set up the checkBoxes

    // register listeners for JCheckBoxes

} // end CheckBoxFrame constructor
```



Buttons that Maintain State

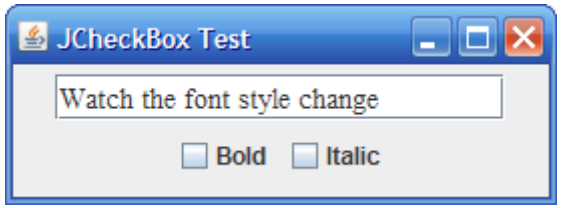
JCheckBox - JRadioButton




```
import…
public class ButtonFrame extends JFrame{
    private JTextField textField;
    private JCheckBox boldJCheckBox;
    private JCheckBox italicJCheckBox;

    public CheckBoxFrame(){
        super( "JCheckBox Test" );
        setLayout( new FlowLayout() ); // set frame layout

    // set up JTextField and set its font
        textField = new JTextField( "Watch the font style change", 20 );
        textField.setFont( new Font( "Serif", Font.PLAIN, 14 ) );
        add( textField ); // add textField to JFrame

    // set up the checkBoxes

    // register listeners for JCheckBoxes
} // end CheckBoxFrame constructor
```

Buttons that Maintain State

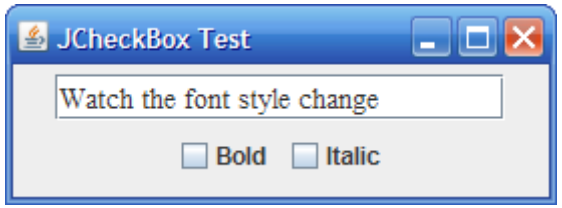
JCheckBox - JRadioButton




```
import…
public class ButtonFrame extends JFrame{
    private JTextField textField;
    private JCheckBox boldJCheckBox;
    private JCheckBox italicJCheckBox;

    public CheckBoxFrame(){
        super( "JCheckBox Test" );
        setLayout( new FlowLayout() ); // set frame layout

    // set up JTextField and set its font

    // set up the checkBoxes
        boldJCheckBox = new JCheckBox( "Bold" ); // create bold checkbox
        italicJCheckBox = new JCheckBox( "Italic" ); // create italic
        add( boldJCheckBox ); add( italicJCheckBox );

    // register listeners for JCheckBoxes
} // end CheckBoxFrame constructor
```

Buttons that Maintain State

JCheckBox - JRadioButton




```
import…
public class ButtonFrame extends JFrame{
    private JTextField textField;
    private JCheckBox boldJCheckBox;
    private JCheckBox italicJCheckBox;

    public CheckBoxFrame(){
        super( "JCheckBox Test" );
        setLayout( new FlowLayout() ); // set frame layout

    // set up JTextField and set its font

    // set up the checkBoxes

    // register listeners for JCheckBoxes
        CheckBoxHandler handler = new CheckBoxHandler();
        boldJCheckBox.addItemListener( handler );
        italicJCheckBox.addItemListener( handler );
    } // end CheckBoxFrame constructor
```

Buttons that Maintain State

CheckBoxHandler

```
import…
public class ButtonFrame extends JFrame{
    //attributes
    public CheckBoxFrame(){…}

    private class CheckBoxHandler implements ItemListener{
        private int valBold = Font.PLAIN; // controls bold font style
        private int valItalic = Font.PLAIN; // controls italic font style

        public void itemStateChanged( ItemEvent event ){
            if ( event.getSource() == boldJCheckBox )
                valBold = boldJCheckBox.isSelected() ? Font.BOLD : Font.PLAIN;
            if ( event.getSource() == italicJCheckBox )
                valItalic = italicJCheckBox.isSelected() ? Font.ITALIC : Font.PLAIN;
            textField.setFont(new Font( "Serif", valBold + valItalic, 14 ) );
        } // end method itemStateChanged
    } // end private inner class CheckBoxHandler
```

respond to checkBox events

Buttons that Maintain State

CheckBoxHandler

```
import…
public class ButtonFrame extends JFrame{
    //attributes
    public CheckBoxFrame(){…}

    private class CheckBoxHandler implements ItemListener{
        private int valBold = Font.PLAIN; // controls bold font style
        private int valItalic = Font.PLAIN; // controls italic font style

        public void itemStateChanged( ItemEvent event ){
            if ( event.getSource() == boldJCheckBox )
                valBold = boldJCheckBox.isSelected() ? Font.BOLD : Font.PLAIN;
            if ( event.getSource() == italicJCheckBox )
                valItalic = italicJCheckBox.isSelected() ? Font.ITALIC : Font.PLAIN;
            textField.setFont(new Font( "Serif", valBold + valItalic, 14 ) );
        } // end method itemStateChanged
    } // end private inner class CheckBoxHandler
```

process bold checkBox events

Buttons that Maintain State

CheckBoxHandler

```
import…
public class ButtonFrame extends JFrame{
    //attributes
    public CheckBoxFrame(){…}

    private class CheckBoxHandler implements ItemListener{
        private int valBold = Font.PLAIN; // controls bold font style
        private int valItalic = Font.PLAIN; // controls italic font style

        public void itemStateChanged( ItemEvent event ){
            if ( event.getSource() == boldJCheckBox )
                valBold = boldJCheckBox.isSelected() ? Font.BOLD : Font.PLAIN;
            if ( event.getSource() == italicJCheckBox )
                valItalic = italicJCheckBox.isSelected() ? Font.ITALIC : Font.PLAIN;
            textField.setFont(new Font( "Serif", valBold + valItalic, 14 ) );
        } // end method itemStateChanged
    } // end private inner class CheckBoxHandler
```

process italic checkBox events

Buttons that Maintain State

CheckBoxHandler

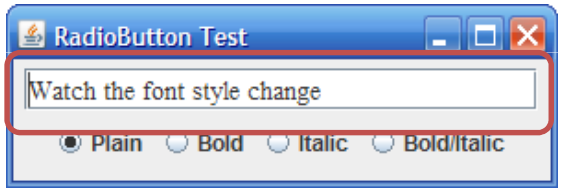
```
import…
public class ButtonFrame extends JFrame{
    //attributes
    public CheckBoxFrame(){…}

    private class CheckBoxHandler implements ItemListener{
        private int valBold = Font.PLAIN; // controls bold font style
        private int valItalic = Font.PLAIN; // controls italic font style

        public void itemStateChanged( ItemEvent event ){
            if ( event.getSource() == boldJCheckBox )
                valBold = boldJCheckBox.isSelected() ? Font.BOLD : Font.PLAIN;
            if ( event.getSource() == italicJCheckBox )
                valItalic = italicJCheckBox.isSelected() ? Font.ITALIC : Font.PLAIN;
            textField.setFont(new Font( "Serif", valBold + valItalic, 14 ) );
        } // end method itemStateChanged
    } // end private inner class CheckBoxHandler
```

set text field font

Buttons that Maintain State

JCheckBox - JRadioButton

```
import…
public class ButtonFrame extends JFrame{
    //attributes
    private Font plainFont;
    private Font boldFont;
    private Font italicFont;
    private Font boldItalicFont;
    private JTextField textField;
    private JRadioButton plainJRadioButton;
    private JRadioButton boldJRadioButton;
    private JRadioButton italicJRadioButton;
    private JRadioButton boldItalicJRadioButton;
    private ButtonGroup radioGroup;
    private ButtonGroup radioGroup1;

    public RadioButtonFrame(){…}

    private class RadioButtonHandler implements ItemListener{…}
}
```



Buttons that Maintain State

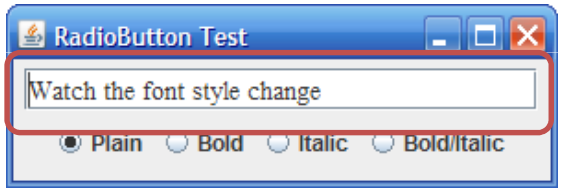
JCheckBox - JRadioButton

```
import…
public class ButtonFrame extends JFrame{
    //attributes
    private Font plainFont;
    private Font boldFont;
    private Font italicFont;
    private Font boldItalicFont;
    private JTextField textField;
    private JRadioButton plainJRadioButton;
    private JRadioButton boldJRadioButton;
    private JRadioButton italicJRadioButton;
    private JRadioButton boldItalicJRadioButton;
    private ButtonGroup radioGroup;
    private ButtonGroup radioGroup1;

    public RadioButtonFrame(){…}

    private class RadioButtonHandler implements ItemListener{…}
}
```

Buttons that Maintain State

JCheckBox - JRadioButton

```
import…
public class ButtonFrame extends JFrame{
    //attributes
    private Font plainFont;
    private Font boldFont;
    private Font italicFont;
    private Font boldItalicFont;
    private JTextField textField;
    private JRadioButton plainJRadioButton;
    private JRadioButton boldJRadioButton;
    private JRadioButton italicJRadioButton;
    private JRadioButton boldItalicJRadioButton;
    private ButtonGroup radioGroup;
    private ButtonGroup radioGroup1;

    public RadioButtonFrame(){…}

    private class RadioButtonHandler implements ItemListener{…}
}
```



logical relationship between JRadioButtons

Buttons that Maintain State

JCheckBox - JRadioButton

```
import…
public class ButtonFrame extends JFrame{
    //attributes
    public RadioButtonFrame(){
        // super() - set frame layout.
        //add text field

        // create radio buttons
        plainJRadioButton = new JRadioButton( "Plain", true );
        add( plainJRadioButton ); // add plain button to Jframe

        // create logical relationship between JRadioButtons
        radioGroup = new ButtonGroup(); // create ButtonGroup
        radioGroup.add( plainJRadioButton ); // add plain to group
        radioGroup.add( boldJRadioButton ); // add bold to group

        // create font objects
        // register events for JRadioButtons
    }
```

Buttons that Maintain State

JCheckBox - JRadioButton

```
import…
public class ButtonFrame extends JFrame{
    //attributes
    public RadioButtonFrame(){
        // super() - set frame layout.
        //add text field

        // create radio buttons
        plainJRadioButton = new JRadioButton( "Plain", true );
        add( plainJRadioButton ); // add plain button to Jframe

         // create logical relationship between JRadioButtons
        radioGroup = new ButtonGroup(); // create ButtonGroup
        radioGroup.add( plainJRadioButton ); // add plain to group
        radioGroup.add( boldJRadioButton ); // add bold to group

        // create font objects
        // register events for JRadioButtons
    }
```

Buttons that Maintain State

JCheckBox - JRadioButton

```
import…
public class ButtonFrame extends JFrame{
    //attributes
    public RadioButtonFrame(){
        // super() - set frame layout.
        //add text field
        // create radio buttons
        // create logical relationship between JRadioButtons

        // create font objects
        plainFont = new Font( "Serif", Font.PLAIN, 14 );
        boldFont = new Font( "Serif", Font.BOLD, 14 );
        italicFont = new Font( "Serif", Font.ITALIC, 14 );
        boldItalicFont = new Font( "Serif", Font.BOLD + Font.ITALIC, 14 );
        textField.setFont( plainFont ); // set initial font to plain

        // register events for JRadioButtons
       plainJRadioButton.addItemListener(new RadioButtonHandler(plainFont) );
}
```

Buttons that Maintain State

JCheckBox - JRadioButton

```
import…
public class ButtonFrame extends JFrame{
    //attributes
    public RadioButtonFrame(){
        // super() - set frame layout.
        //add text field
        // create radio buttons
        // create logical relationship between JRadioButtons

        // create font objects
        plainFont = new Font( "Serif", Font.PLAIN, 14 );
        boldFont = new Font( "Serif", Font.BOLD, 14 );
        italicFont = new Font( "Serif", Font.ITALIC, 14 );
        boldItalicFont = new Font( "Serif", Font.BOLD + Font.ITALIC, 14 );
        textField.setFont( plainFont ); // set initial font to plain

        // register events for JRadioButtons
       plainJRadioButton.addItemListener(new RadioButtonHandler(plainFont) );
}
```

Buttons that Maintain State

RadioButtonHandler

```
import…
public class ButtonFrame extends JFrame{
    //attributes
    public CheckBoxFrame(){…}

    private class RadioButtonHandler implements ItemListener{
        private Font font; // font associated with this listener

        public RadioButtonHandler( Font f ){
            font = f;
        } // end constructor RadioButtonHandler

        public void itemStateChanged( ItemEvent event ){
            textField.setFont( font );
        } // end method itemStateChanged
    }
```

respond to RadioButton events

Buttons that Maintain State

RadioButtonHandler

```
import…
public class ButtonFrame extends JFrame{
    //attributes
    public CheckBoxFrame(){…}

    private class RadioButtonHandler implements ItemListener{
        private Font font; // font associated with this listener

        public RadioButtonHandler( Font f ){
            font = f;
        } // end constructor RadioButtonHandler

        public void itemStateChanged( ItemEvent event ){
            textField.setFont( font );
        } // end method itemStateChanged
    }
```

set the font of this listener

Buttons that Maintain State

RadioButtonHandler

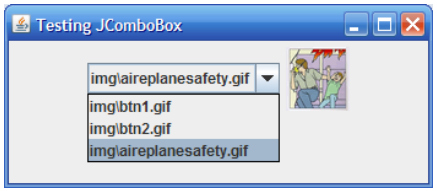
```
import…
public class ButtonFrame extends JFrame{
    //attributes
    public CheckBoxFrame(){…}

    private class RadioButtonHandler implements ItemListener{
        private Font font; // font associated with this listener

        public RadioButtonHandler( Font f ){
            font = f;
        } // end constructor RadioButtonHandler

        public void itemStateChanged( ItemEvent event ){
            textField.setFont( font );
        } // end method itemStateChanged
    }
```

handle radio button events by setting the font of textField

Buttons that Maintain State

JCheckBox - JRadioButton

```
import…
public class ButtonFrame extends JFrame{
    //attributes
    private JComboBox imagesJComboBox;
    private JLabel label;

    private String names[] =
    { "img\\btn1.gif", "img\\btn2.gif",  "img\\aireplanesafety.gif" };
    private Icon icons[] = {
        new ImageIcon( names[ 0 ] ),
        new ImageIcon( names[ 1 ] ),
        new ImageIcon( names[ 2 ] ) };

    public ComboBoxFrame(){…}
}
```



Buttons that Maintain State

JCheckBox - JRadioButton

```
import…
public class ButtonFrame extends JFrame{
    //attributes

    public ComboBoxFrame(){
        // super() - set frame layout.

        imagesJComboBox = new JComboBox( names );
        imagesJComboBox.setMaximumRowCount( 3 );
        imagesJComboBox.addItemListener(
            new ItemListener(){…}
        ); // end call to addItemListener

        add( imagesJComboBox ); // add combobox to JFrame
        label = new JLabel( icons[ 0 ] ); // display first icon
        add( label );
    }
```

anonymous inner class

Buttons that Maintain State

JCheckBox - JRadioButton

```
import…
public class ButtonFrame extends JFrame{
    //attributes

    public ComboBoxFrame(){
        // super() - set frame layout.
        //attributes
        imagesJComboBox.addItemListener(
        new ItemListener(){
            public void itemStateChanged( ItemEvent event ) {
                if ( event.getStateChange() == ItemEvent.SELECTED )
                    label.setIcon(icons[ imagesJComboBox.getSelectedIndex() ] );
            }
        } // end anonymous inner class
        ); // end call to addItemListener
        // set up
    }
```

handle JComboBox event

Buttons that Maintain State

JCheckBox - JRadioButton

```
import…
public class ButtonFrame extends JFrame{
    //attributes

    public ComboBoxFrame(){
        // super() - set frame layout.
        //attributes
        imagesJComboBox.addItemListener(
        new ItemListener(){
            public void itemStateChanged( ItemEvent event ) {
                if ( event.getStateChange() == ItemEvent.SELECTED )
                    label.setIcon(icons[ imagesJComboBox.getSelectedIndex() ] );
            }
        } // end anonymous inner class
        ); // end call to addItemListener
        // set up
    }
```

determine whether item selected

What else…

- JList
- Mouse Event Handling
- JPanel
- Key Event Handling
- Opening Files with JFileChooser

Check the source code

Download it from here

References

- J. Barker, *Beginning Java Objects: From Concepts To Code, Second Edition*, Apress, 2005.
- H.M. Deitel and P.J. Deitel, Java How to Program: Early Objects Version, Prentice Hall, 2009.