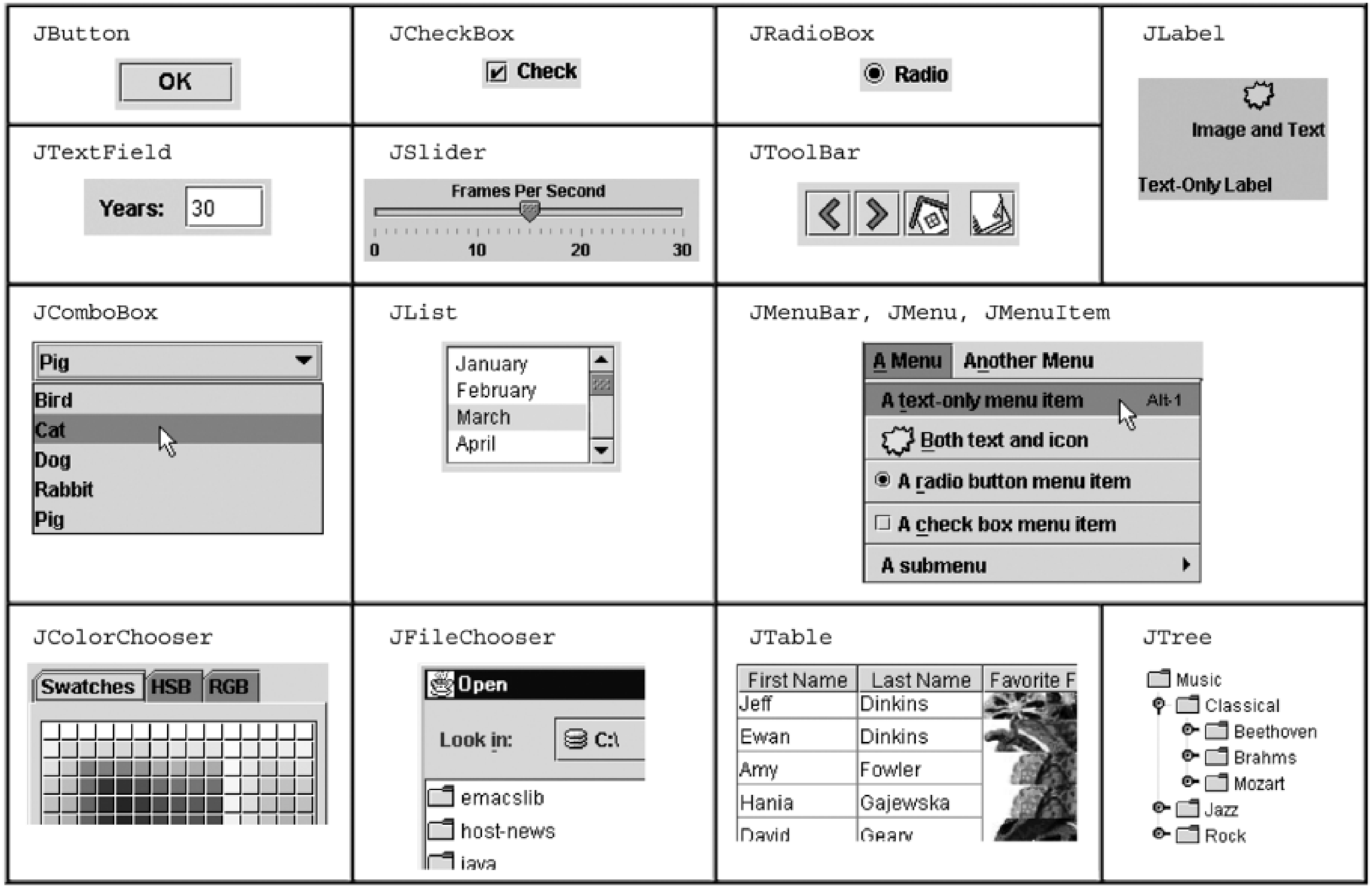
Swing GUI Components

Components



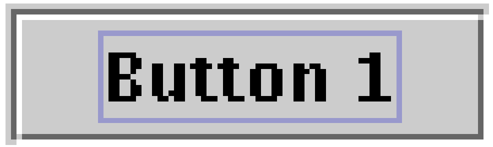
First Name	Last Name	Favorite F
Jeff	Dinkins	
Ewan	Dinkins	
Amy	Fowler	
Hania	Gajewska	
David	Geary	

Component properties

- Each has a `get` (or `is`) accessor and a `set` modifier method.
- examples: `getColor`, `setFont`, `setEnabled`, `isVisible`

name	type	description
background	`Color`	background color behind component
border	`Border`	border line around component
enabled	`boolean`	whether it can be interacted with
focusable	`boolean`	whether key text can be typed on it
font	`Font`	font used for text in component
foreground	`Color`	foreground color of component
height, width	`int`	component's current size in pixels
visible	`boolean`	whether component can be seen
tooltip text	`String`	text shown when hovering mouse
size, minimum / maximum / preferred size	`Dimension`	various sizes, size limits, or desired sizes that the component may take

JButton



a clickable region for causing actions to occur

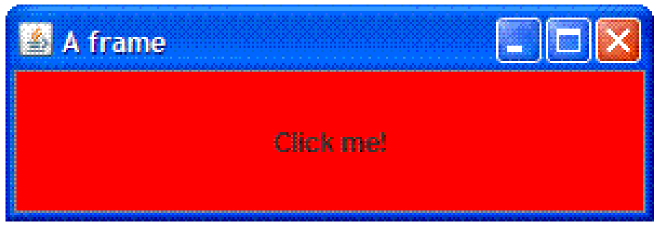
- `public JButton(String text)`
  Creates a new button with the given string as its text.

- `public String getText()`
  Returns the text showing on the button.

- `public void setText(String text)`
  Sets button's text to be the given string.

Simple JButton Example

```
import java.awt.*;        // Where is the other button?
import javax.swing.*;

public class ButtonExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(new Dimension(300, 100));
        frame.setTitle("A frame");

        JButton button1 = new JButton();
        button1.setText("I'm a button.");
        button1.setBackground(Color.BLUE);
        frame.add(button1);

        JButton button2 = new JButton();
        button2.setText("Click me!");
        button2.setBackground(Color.RED);
        frame.add(button2);

        frame.setVisible(true);
    }
```



JLabel

a string of text displayed on screen in a graphical program. Labels often give information or describe other components

- Can display:
  - Single line of read-only text
  - Image
  - Text and image
  - `setToolTipText` method (from `JComponent`) specifies text that appears when user moves cursor over `JLabel`

- `public JLabel(String text)`
  Creates a new label with the given string as its text.
- `public JLabel(ImageIcon picture)`

Constructs a label which displays the image picture.

- `public String getText()`
  Returns the text showing on the label.

- `public void setText(String text)`
  Sets label's text to be the given string.

Jlabel (cont.)

- `public void setText(String text, int alignment)`
  - replaces the text currently displayed in the label by text.
  - It also sets the alignment of the text, left, right or centre.
  - The possible values of alignment can be found in the class SwingConstants, e.g. use SwingConstants.CENTER to centre the text in the label.

- `public void setForeground(Color c)`

  sets the text colour to c.
- `public void setBackground(Color c)`

  sets the background colour to c.
- `public void setOpaque(boolean b)`

  makes the label transparent if b is false and nontransparent if b is true.

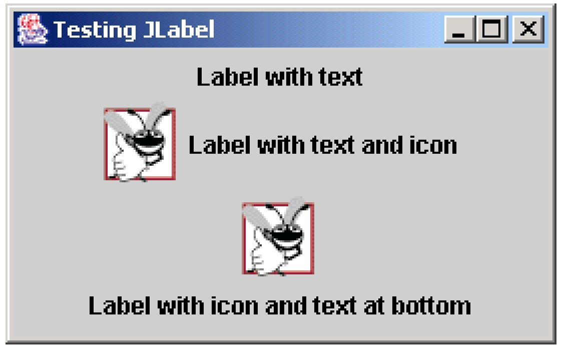
Simple JLabel Example

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

public class LabelTest extends JFrame {
     private JLabel label1, label2, label3;

      // set up GUI
      public LabelTest()
      {
         super( "Testing JLabel" );
         // get content pane and set its layout
         Container container = getContentPane();
         container.setLayout( new FlowLayout() );

         // JLabel constructor with a string argument
         label1 = new JLabel( "Label with text" );
         label1.setToolTipText( "This is label1" );
         container.add( label1 );
```

Simple JLabel Example (cont.)

```
// JLabel constructor with string, Icon and alignment arguments
Icon bug = new ImageIcon( "bug1.gif" );
label2 = new JLabel( "Label with text and icon", bug,
SwingConstants.LEFT );
label2.setToolTipText( "This is label2" );
container.add( label2 );

// JLabel constructor no arguments
label3 = new JLabel();
label3.setText( "Label with icon and text at bottom" );
label3.setIcon( bug );
label3.setHorizontalTextPosition( SwingConstants.CENTER );
label3.setVerticalTextPosition( SwingConstants.BOTTOM );
label3.setToolTipText( "This is label3" );
container.add( label3 );
```

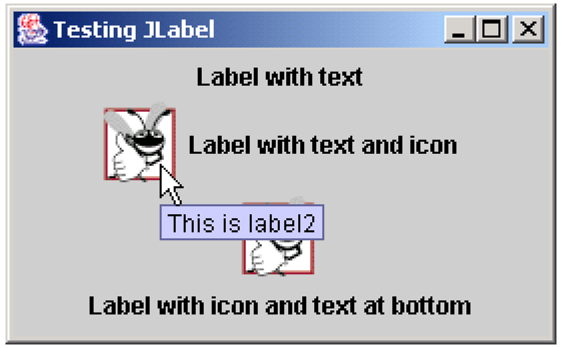
Simple JLabel Example (cont.)


Testing JLabel

Label with text

Label with text and icon

Label with icon and text at bottom


Testing JLabel

Label with text

Label with text and icon

This is label2

Label with icon and text at bottom

```
      setSize( 275, 170 );
      setVisible( true );

   } // end constructor

   public static void main( String args[] )
   {
      LabelTest application = new LabelTest();
      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
   }

} // end class LabelTest
```

JTextField, JPasswordField

an input control for typing text values
(field = single line; area = multi-line)

- `public JTextField(int columns)`
  - Single-line area in which user can enter text
  - setEditable(false), user cannot modify textfield
- `public class JPasswordField extends JTextField`
  - Hides characters that user enters

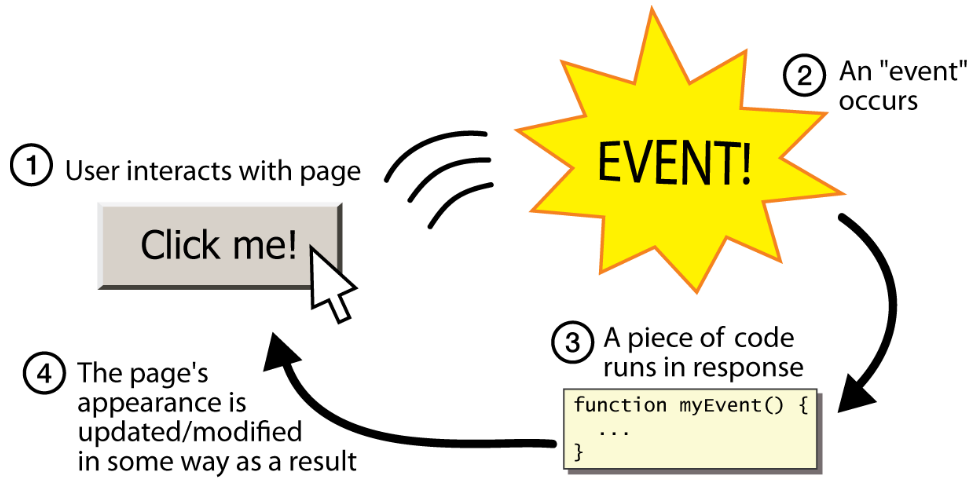
- `public String getText()`
  Returns the text currently in the field.

- `public void setText(String text)`
  Sets field's text to be the given string.

Graphical events

- **event**: An object that represents a user's interaction with a GUI component; can be "handled" to create interactive components.

- **listener**: An object that waits for events and responds to them.
  - To handle an event, attach a *listener* to a component.
  - The listener will be notified when the event occurs (e.g. button click).

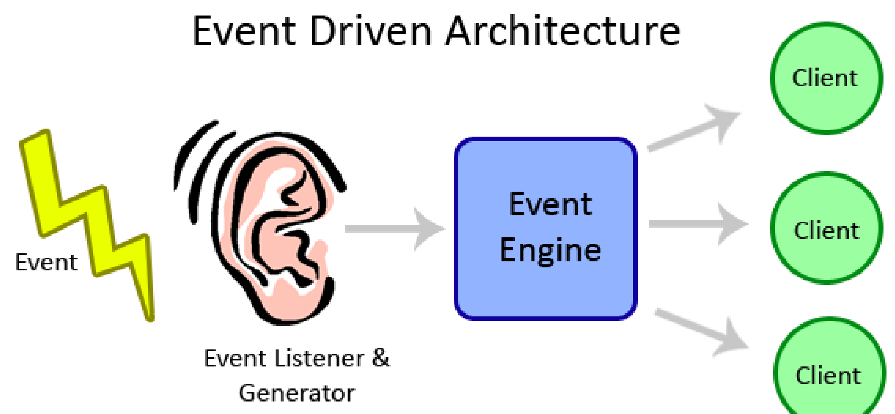


① User interacts with page
Click me!
EVENT!
② An "event" occurs
③ A piece of code runs in response
④ The page's appearance is updated/modified in some way as a result

```
function myEvent() {
  ...
}
```

Event-driven programming

- **event-driven programming**: A style of coding where a program's overall flow of execution is dictated by events.
  - Rather than a central "main" method that drives execution, the program loads and waits for user input events.
  - As each event occurs, the program runs particular code to respond.
  - The overall flow of what code is executed is determined by the series of events that occur, not a pre-determined order.



Event hierarchy

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

- `EventObject`
  - `AWTEvent` (AWT)
    - **`ActionEvent`**
    - `TextEvent`
    - `ComponentEvent`
      - `FocusEvent`
      - `WindowEvent`
      - `InputEvent`
        - `KeyEvent`
        - `MouseEvent`

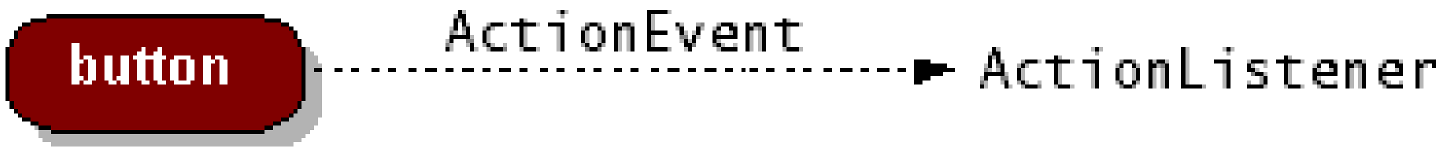
- `EventListener`
  - `AWTEventListener`
  - **`ActionListener`**
  - `TextListener`
  - `ComponentListener`
  - `FocusListener`
  - `WindowListener`

  - `KeyListener`
  - `MouseListener`

Action events

- **action event**: An action that has occurred on a GUI component.
  - The most common, general event type in Swing. Caused by:
    - button or menu clicks,
    - check box checking / unchecking,
    - pressing Enter in a text field, ...

  - Represented by a class named `ActionEvent`
  - Handled by objects that implement interface `ActionListener`



Implementing a listener

```
public class name implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        code to handle the event;
    }
}
```

- `JButton` and other graphical components have this method:
  - `public void addActionListener(ActionListener al)` Attaches the given listener to be notified of clicks and events that occur on this component.

GUI event example

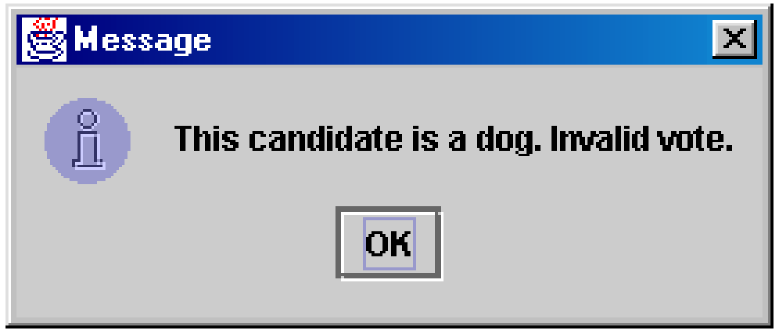
```java
public class MyGUI {
    private JFrame frame;
    private JButton doubleText;
    private JTextField textfield;

    public MyGUI() {
        ...
        doubleText.addActionListener(new DoubleTextListener());
    }
    ...

    // When button is clicked, doubles the field's text.
    private class DoubleTextListener implements ActionListener {
        public void actionPerformed(ActionEvent event) {
            String text = textfield.getText();
            textfield.setText(text + text);
        }
    }
}
```

JOptionPane

- `JOptionPane.showMessageDialog(`**parent**`,` **message**`);`

```
import javax.swing.*;

JOptionPane.showMessageDialog(null,
    "This candidate is a dog. Invalid vote.");
```

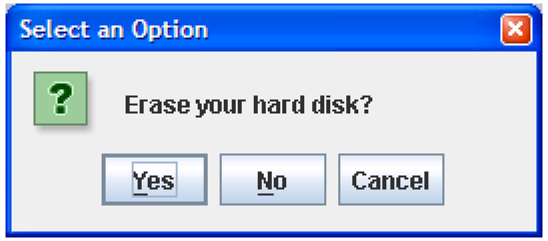
- Advantages:
  - Simple; looks better than console.
- Disadvantages:
  - Created with static methods; not object-oriented.
  - Not powerful (just simple dialog boxes).



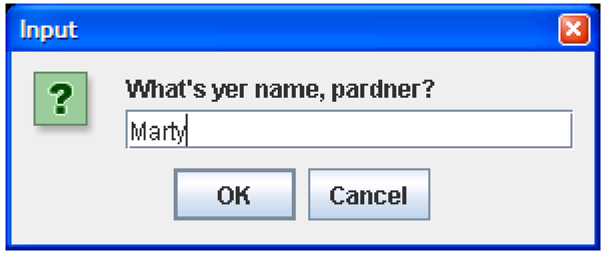
This candidate is a dog. Invalid vote.

More JOptionPane

- `JOptionPane.showConfirmDialog(`**parent, message**`)`
  - Displays a message and list of choices Yes, No, Cancel.
  - Returns an `int` such as `JOptionPane.YES_OPTION` or `NO_OPTION` to indicate what button was pressed.



- `JOptionPane.showInputDialog(`**parent, message**`)`
  - Displays a message and text field for input.
  - Returns the value typed as a String (or `null` if user presses Cancel).



Simple JButton Example

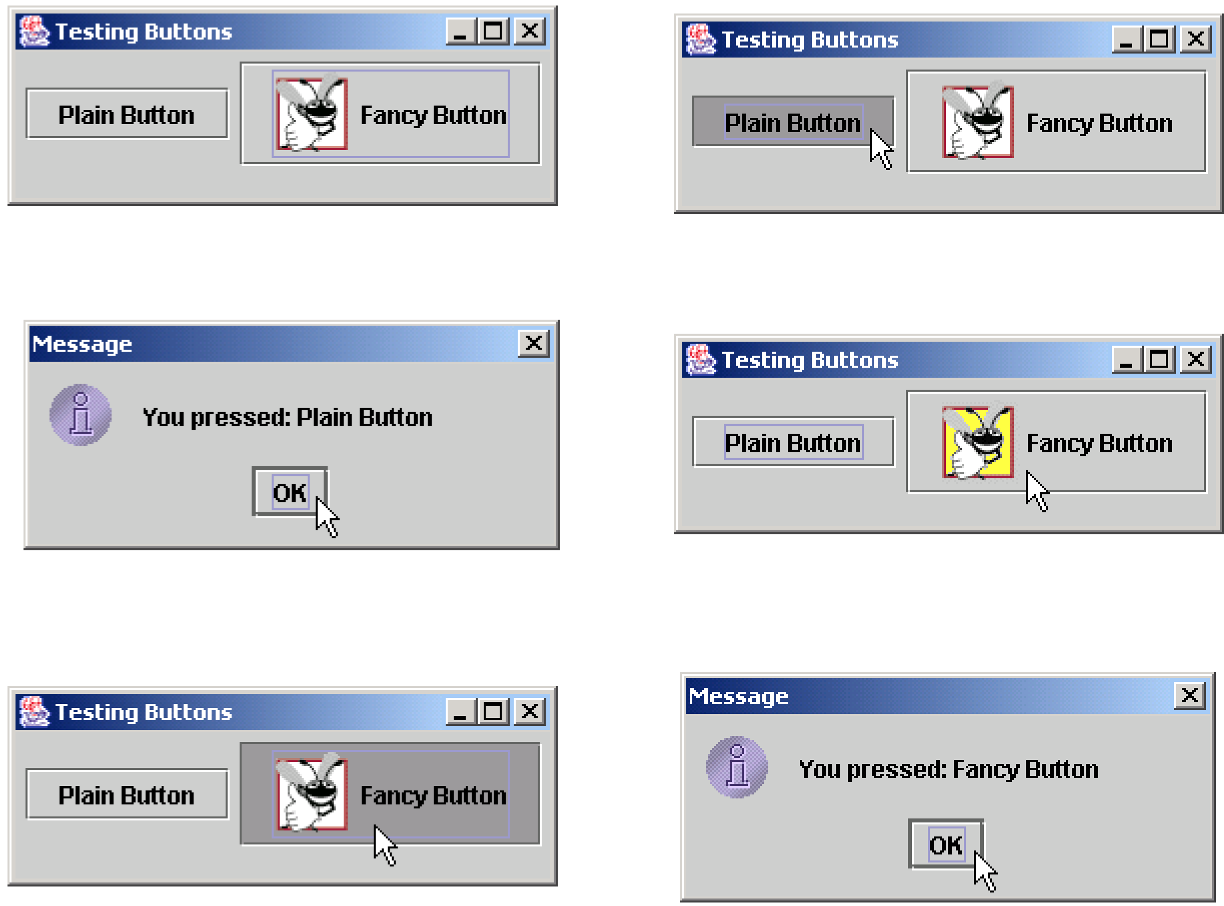
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ButtonTest extends JFrame {
   private JButton plainButton, fancyButton;

   // set up GUI
   public ButtonTest()
   {
      super( "Testing Buttons" );

      // get content pane and set its layout
      Container container = getContentPane();
      container.setLayout( new FlowLayout() );

      // create buttons
      plainButton = new JButton( "Plain Button" );
      container.add( plainButton );
```

Simple JButton Example (cont.)

```
      Icon bug1 = new ImageIcon( "bug1.gif" );
      Icon bug2 = new ImageIcon( "bug2.gif" );
      fancyButton = new JButton( "Fancy Button", bug1 );
      fancyButton.setRolloverIcon( bug2 );
      container.add( fancyButton );

     // create an instance of inner class ButtonHandler
     // to use for button event handling

    ButtonHandler handler = new ButtonHandler();
    fancyButton.addActionListener( handler );
    plainButton.addActionListener( handler );

    setSize( 275, 100 );
    setVisible( true );

} // end ButtonTest constructor
```

Simple JButton Example (cont.)

```
    public static void main( String args[] )
   {
      ButtonTest application = new ButtonTest();
      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
   }

   // inner class for button event handling
    private class ButtonHandler implements ActionListener {

     // handle button event
     public void actionPerformed( ActionEvent event )
     {
        JOptionPane.showMessageDialog( ButtonTest.this,
           "You pressed: " + event.getActionCommand() );
     }

   } // end private inner class ButtonHandler

} // end class ButtonTest
```



You pressed: Plain Button

You pressed: Fancy Button

Simple JTextField Example

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

  public class TextFieldTest extends JFrame {
    private JTextField textField1, textField2, textField3;
    private JPasswordField passwordField;

    // set up GUI
    public TextFieldTest()
    {
        super( "Testing JTextField and JPasswordField" );

        Container container = getContentPane();
        container.setLayout( new FlowLayout() );

        // construct textfield with default sizing
        textField1 = new JTextField( 10 );
        container.add( textField1 );
```

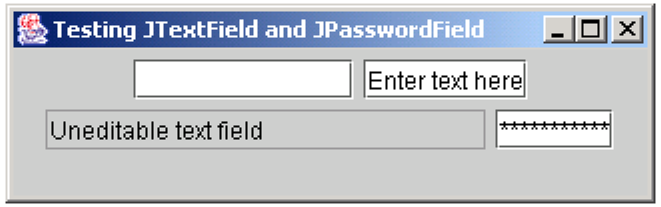
Simple JTextField Example (cont.)

```
        // construct textfield with default text
        textField2 = new JTextField( "Enter text here" );
        container.add( textField2 );
        // construct textfield with default text,
        textField3 = new JTextField( "Uneditable text field", 20 );
        textField3.setEditable( false );
        container.add( textField3 );

       // construct passwordfield with default text
        passwordField = new JPasswordField( "Hidden text" );
        container.add( passwordField );

       // register event handlers
        TextFieldHandler handler = new TextFieldHandler();
        textField1.addActionListener( handler );
        textField2.addActionListener( handler );
        textField3.addActionListener( handler );
        passwordField.addActionListener( handler );

        setSize( 325, 100 );
        setVisible( true );
     } // end constructor TextFieldTest
public static void main( String args[] ){
     TextFieldTest application = new TextFieldTest();
     application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
```

Simple JTextField Example (cont.)

```
// private inner class for event handling
      private class TextFieldHandler implements ActionListener {
         // process textfield events
         public void actionPerformed( ActionEvent event )
         {
             String string = "";
             // user pressed Enter in JTextField textField1
             if ( event.getSource() == textField1 )
                string = "textField1: " + event.getActionCommand();
                 // user pressed Enter in JTextField textField2
             else if ( event.getSource() == textField2 )
                string = "textField2: " + event.getActionCommand();
                 // user pressed Enter in JTextField textField3
             else if ( event.getSource() == textField3 )
                string = "textField3: " + event.getActionCommand();
                 // user pressed Enter in JTextField passwordField
             else if ( event.getSource() == passwordField ) {
                string = "passwordField: " +
                new String( passwordField.getPassword() );
             }
             JOptionPane.showMessageDialog( null, string );
             } // end method actionPerformed

     } // end private inner class TextFieldHandler
```

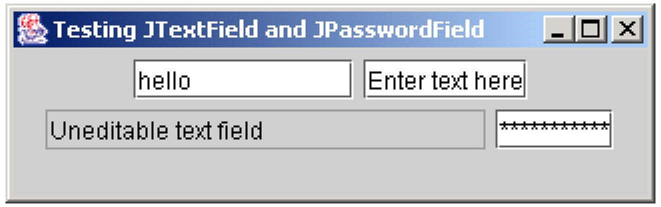
Simple JTextField Example (cont.)


Testing JTextField and JPasswordField

Enter text here

Uneditable text field

****

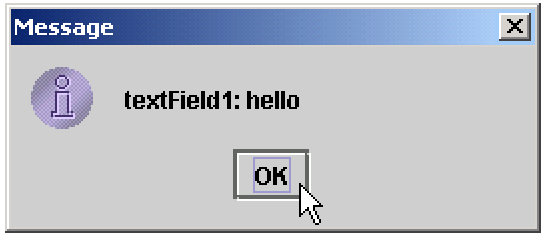

Testing JTextField and JPasswordField

hello

Enter text here

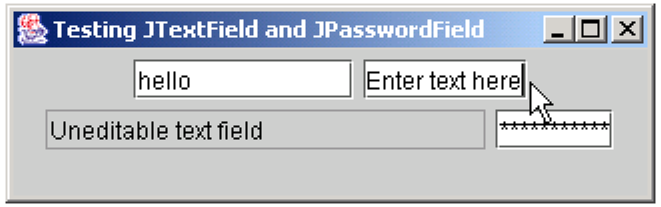
Uneditable text field

****


Message

textField1: hello

OK

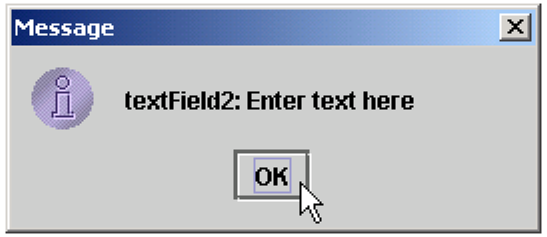

Testing JTextField and JPasswordField

hello

Enter text here

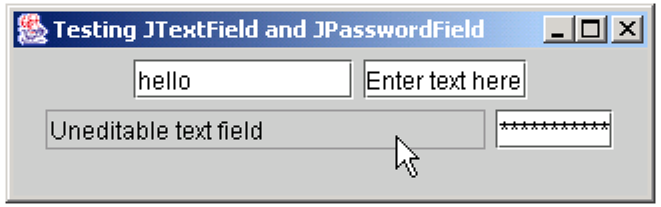
Uneditable text field

****


Message

textField2: Enter text here

OK

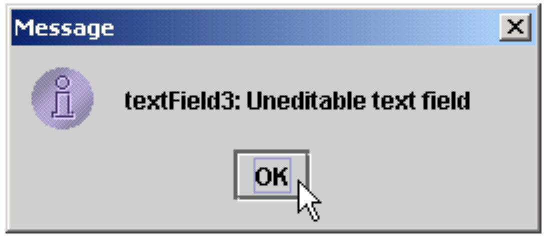

Testing JTextField and JPasswordField

hello

Enter text here

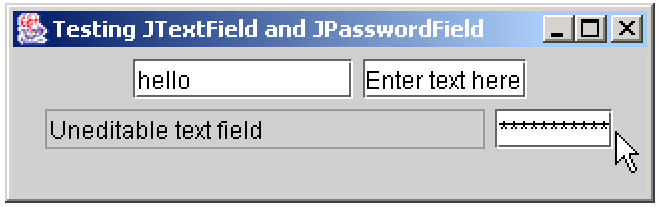
Uneditable text field

****


Message

textField3: Uneditable text field

OK

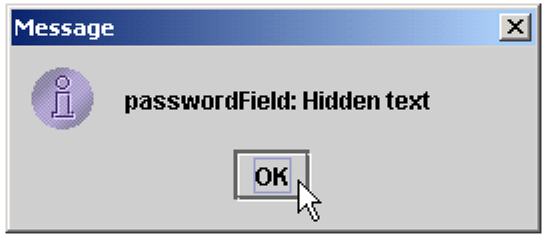

Testing JTextField and JPasswordField

hello

Enter text here

Uneditable text field

****


Message

passwordField: Hidden text

OK