Java GUI Components

Alark Joshi

Graphical Applications

- Except for the applets seen in Chapter 2, the example programs we've explored thus far have been text-based
- They are called *command-line applications*, which interact with the user using simple text prompts
- Let's examine some Java applications that have graphical components
- These components will serve as a foundation to programs that have true graphical user interfaces (GUIs)

GUI Components

- A *GUI component* is an object that represents a screen element such as a button or a text field
- GUI-related classes are defined primarily in the `java.awt` and the `javax.swing` packages
- The *Abstract Windowing Toolkit* (AWT) was the original Java GUI package
- The *Swing* package provides additional and more versatile components
- Both packages are needed to create a Java GUI-based program

GUI Containers

- A *GUI container* is a component that is used to hold and organize other components
- A *frame* is a container displayed as a separate window with a title bar
- It can be repositioned and resized on the screen as needed
- A *panel* is a container that cannot be displayed on its own but is used to organize other components
- A panel must be added to another container (like a frame or another panel) to be displayed

GUI Containers

- A GUI container can be classified as either heavyweight or lightweight
- A *heavyweight container* is one that is managed by the underlying operating system
- A *lightweight container* is managed by the Java program itself
- Occasionally this distinction is important
- A frame is a heavyweight container and a panel is a lightweight container

Labels

- A *label* is a GUI component that displays a line of text and/or an image
- Labels are usually used to display information or identify other components in the interface
- Let's look at a program that organizes two labels in a panel and displays that panel in a frame
- This program is not interactive, but the frame can be repositioned and resized
- See `Authority.java`

```java
//********************************************************************
//  Authority.java       Author: Lewis/Loftus
//
//  Demonstrates the use of frames, panels, and labels.
//********************************************************************

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

public class Authority
{
   //-----------------------------------------------------------------
   //  Displays some words of wisdom.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Authority");

      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      JPanel primary = new JPanel();
      primary.setBackground (Color.yellow);
      primary.setPreferredSize (new Dimension(250, 75));
```

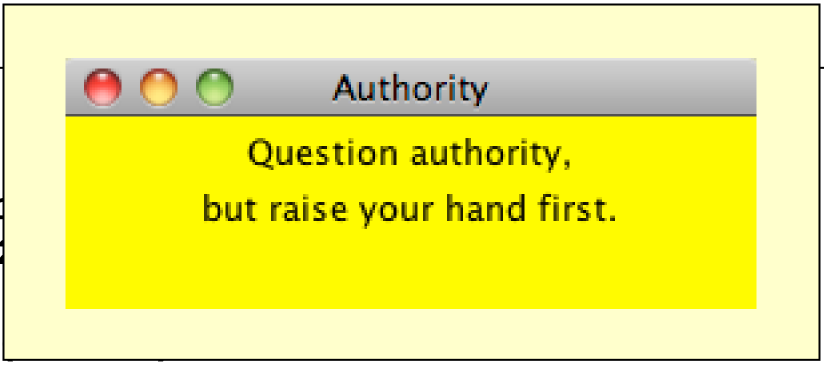
continued

continued

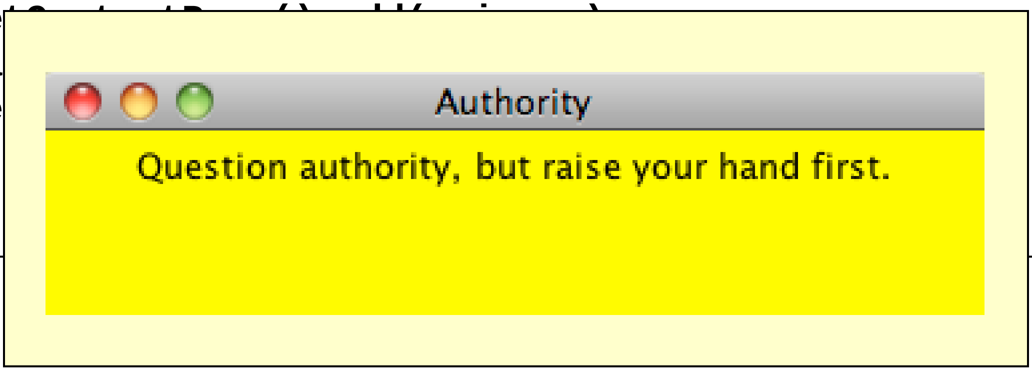
```
      JLabel label1 = new JLabel ("Question authority,");
      JLabel label2 = new JLabel ("but raise your hand first.");

      primary.add (label1);
      primary.add (label2);

      frame.getContentPane().add(primary);
      frame.pack();
      frame.setVisible(true);
   }
}
```

continued

```
      JLabel label1
      JLabel label2                                                      );
                                                                         first.");

      primary.add
      primary.add (label2);

      frame.ge
      frame.pa
      frame.se
   }
}
```



Question authority,
but raise your hand first.



Question authority, but raise your hand first.

Nested Panels

- Containers that contain other components make up the *containment hierarchy* of an interface
- This hierarchy can be as intricate as needed to create the visual effect desired
- The following example nests two panels inside a third panel – note the effect this has as the frame is resized
- See `NestedPanels.java`

```java
//********************************************************************
//  NestedPanels.java       Author: Lewis/Loftus
//
//  Demonstrates a basic component hierarchy.
//********************************************************************

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

public class NestedPanels
{
   //-----------------------------------------------------------------
   //  Presents two colored panels nested within a third.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Nested Panels");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      // Set up first subpanel
      JPanel subPanel1 = new JPanel();
      subPanel1.setPreferredSize (new Dimension(150, 100));
      subPanel1.setBackground (Color.green);
      JLabel label1 = new JLabel ("One");
      subPanel1.add (label1);
```

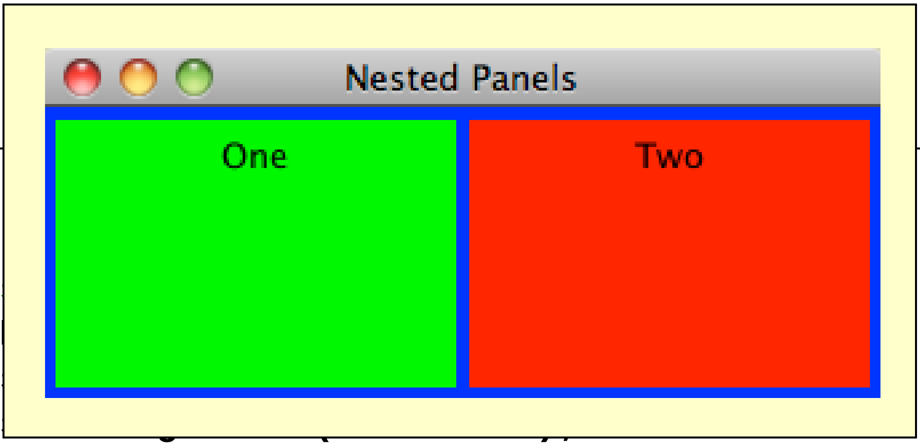
continued

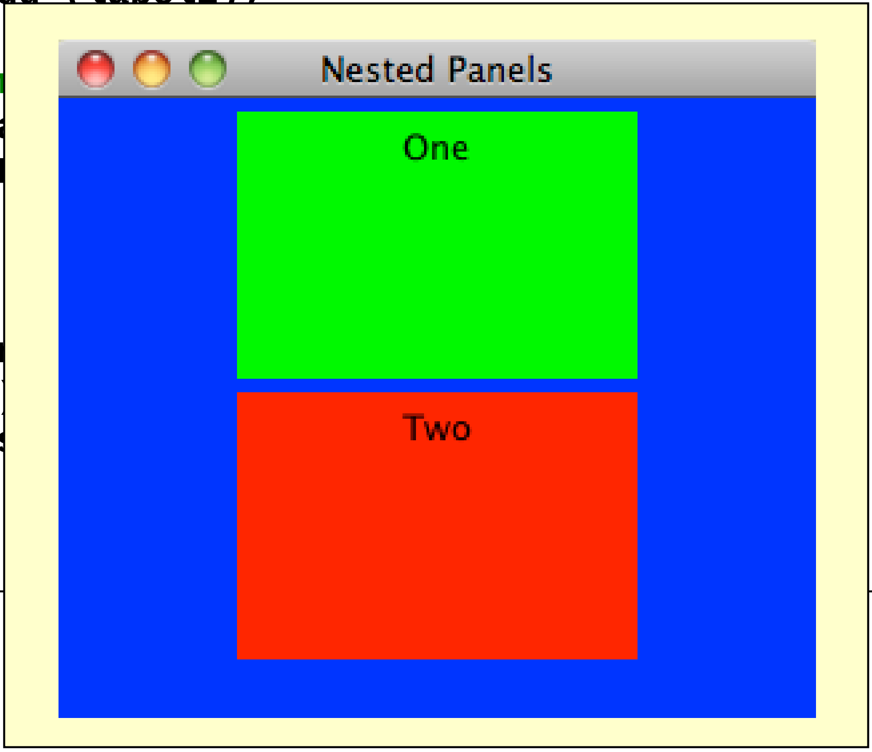
continued

```
      // Set up second subpanel
      JPanel subPanel2 = new JPanel();
      subPanel2.setPreferredSize (new Dimension(150, 100));
      subPanel2.setBackground (Color.red);
      JLabel label2 = new JLabel ("Two");
      subPanel2.add (label2);

      // Set up primary panel
      JPanel primary = new JPanel();
      primary.setBackground (Color.blue);
      primary.add (subPanel1);
      primary.add (subPanel2);

      frame.getContentPane().add(primary);
      frame.pack();
      frame.setVisible(true);
   }
}
```

continued

```
      // Set up
      JPanel sub
      subPanel2.                                                         );
      subPanel2.
      JLabel label2 = new JLabel ("Two");
      subPanel2.add (label2);

      // Set up pr
      JPanel prima
      primary.set
      primary.add
      primary.add

      frame.getCo
      frame.pack(
      frame.setVi
   }
}
```





Images

- Images can be displayed in a Java program in various ways
- As we've seen, a `JLabel` object can be used to display a line of text
- It can also be used to display an image
- That is, a label can be composed of text, an image, or both at the same time

Images

- The `ImageIcon` class is used to represent the image that is stored in a label
- If text is also included, the position of the text relative to the image can be set explicitly
- The alignment of the text and image within the label can be set as well
- See `LabelDemo.java`

```
//********************************************************************
//  LabelDemo.java       Author: Lewis/Loftus
//
//  Demonstrates the use of image icons in labels.
//********************************************************************

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

public class LabelDemo
{
   //-----------------------------------------------------------------
   //  Creates and displays the primary application frame.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Label Demo");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      ImageIcon icon = new ImageIcon ("devil.gif");

      JLabel label1, label2, label3;

      label1 = new JLabel ("Devil Left", icon, SwingConstants.CENTER);
```

continued

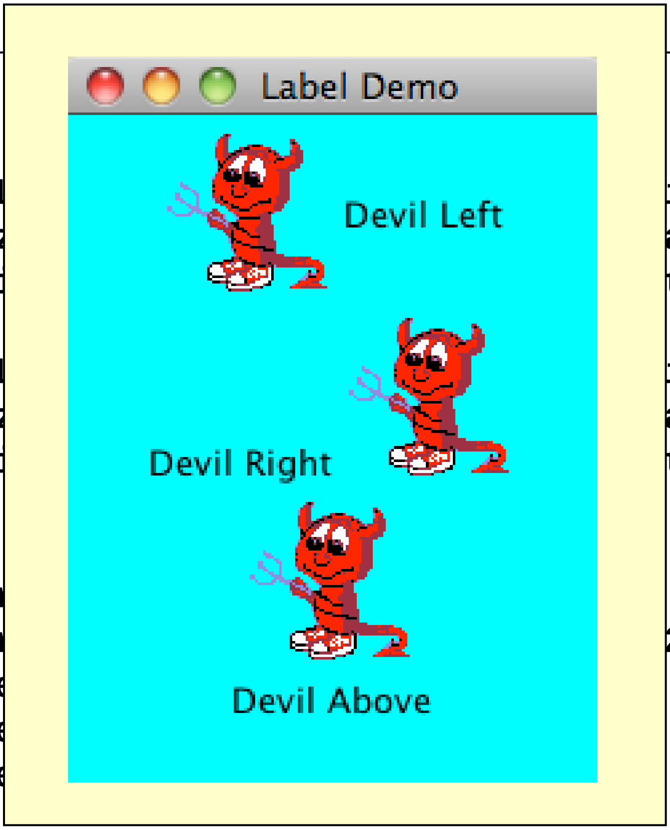
continued

```
      label2 = new JLabel ("Devil Right", icon, SwingConstants.CENTER);
      label2.setHorizontalTextPosition (SwingConstants.LEFT);
      label2.setVerticalTextPosition (SwingConstants.BOTTOM);

      label3 = new JLabel ("Devil Above", icon, SwingConstants.CENTER);
      label3.setHorizontalTextPosition (SwingConstants.CENTER);
      label3.setVerticalTextPosition (SwingConstants.BOTTOM);

      JPanel panel = new JPanel();
      panel.setBackground (Color.cyan);
      panel.setPreferredSize (new Dimension (200, 250));
      panel.add (label1);
      panel.add (label2);
      panel.add (label3);

      frame.getContentPane().add(panel);
      frame.pack();
      frame.setVisible(true);
   }
}
```

continued

```
      label2 = new JL
      label2.setHoriz
      label2.setVerti

      label3 = new JL
      label3.setHoriz
      label3.setVerti

      JPanel panel = 
      panel.setBackgr
      panel.setPrefer
      panel.add (labe
      panel.add (labe
      panel.add (labe
```

```
ingConstants.CENTER);
ants.LEFT);
ts.BOTTOM);

ingConstants.CENTER);
ants.CENTER);
ts.BOTTOM);



250));
```



```
      frame.getContentPane().add(panel);
      frame.pack();
      frame.setVisible(true);
   }
}
```

Graphical Objects

- Some objects contain information that determines how the object should be represented visually
- Most GUI components are graphical objects
- We can have some effect on how components get drawn
- We did this in Chapter 2 when we defined the `paint` method of an applet
- Let's look at some other examples of graphical objects

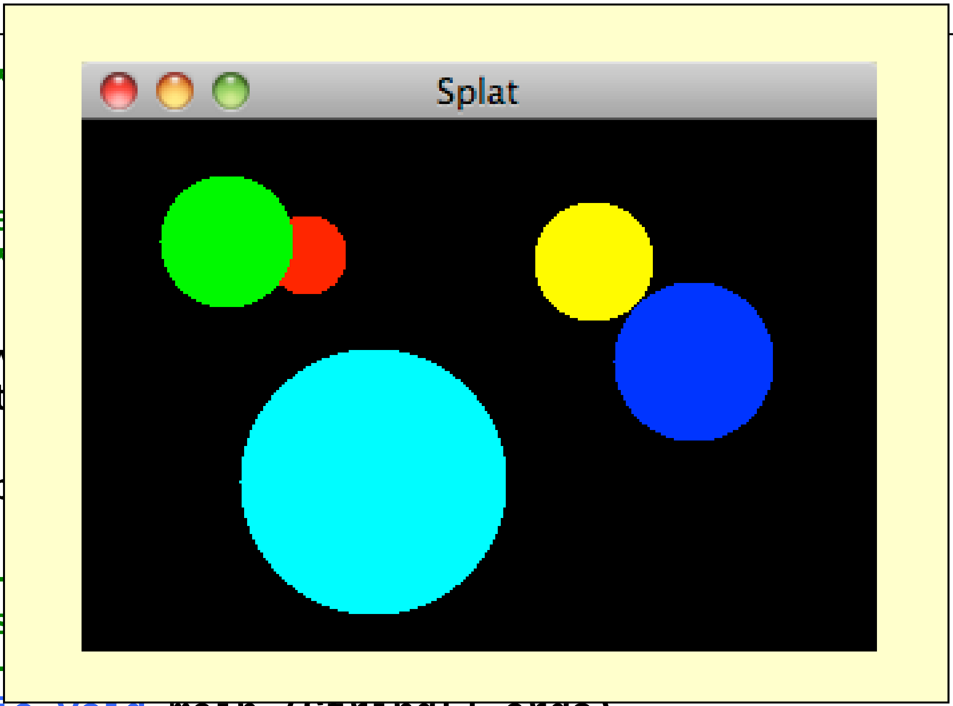
Splat Example

- The `Splat` example draws a set of colored circles on a panel, but each circle is represented as a separate object that maintains its own graphical information

- The `paintComponent` method of the panel "asks" each circle to draw itself

- See `Splat.java`
- See `SplatPanel.java`
- See `Circle.java`

```java
//********************************************************************
//  Splat.java       Author: Lewis/Loftus
//
//  Demonstrates the use of graphical objects.
//********************************************************************

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

public class Splat
{
   //-----------------------------------------------------------------
   //  Presents a collection of circles.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Splat");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add(new SplatPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```



```
//************                                       *************
//  Splat.java
//
//  Demonstrate
//************                                       *************

import javax.sw
import java.awt

public class Sp
{
   //----------                                      -------------
   //  Presents
   //----------                                      -------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Splat");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add(new SplatPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```

```java
//********************************************************************
//  SplatPanel.java       Author: Lewis/Loftus
//
//  Demonstrates the use of graphical objects.
//********************************************************************

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

public class SplatPanel extends JPanel
{
   private Circle circle1, circle2, circle3, circle4, circle5;

   //-----------------------------------------------------------------
   //  Constructor: Creates five Circle objects.
   //-----------------------------------------------------------------
   public SplatPanel()
   {
      circle1 = new Circle (30, Color.red, 70, 35);
      circle2 = new Circle (50, Color.green, 30, 20);
      circle3 = new Circle (100, Color.cyan, 60, 85);
      circle4 = new Circle (45, Color.yellow, 170, 30);
      circle5 = new Circle (60, Color.blue, 200, 60);

      setPreferredSize (new Dimension(300, 200));
      setBackground (Color.black);
   }
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Draws this panel by requesting that each circle draw itself.
   //-----------------------------------------------------------------
   public void paintComponent (Graphics page)
   {
      super.paintComponent(page);

      circle1.draw(page);
      circle2.draw(page);
      circle3.draw(page);
      circle4.draw(page);
      circle5.draw(page);
   }
}
```

```
//********************************************************************
//  Circle.java       Author: Lewis/Loftus
//
//  Represents a circle with a particular position, size, and color.
//********************************************************************

import java.awt.*;

public class Circle
{
   private int diameter, x, y;
   private Color color;

   //-----------------------------------------------------------------
   //  Constructor: Sets up this circle with the specified values.
   //-----------------------------------------------------------------
   public Circle (int size, Color shade, int upperX, int upperY)
   {
      diameter = size;
      color = shade;
      x = upperX;
      y = upperY;
   }
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Draws this circle in the specified graphics context.
   //-----------------------------------------------------------------
   public void draw (Graphics page)
   {
      page.setColor (color);
      page.fillOval (x, y, diameter, diameter);
   }

   //-----------------------------------------------------------------
   //  Diameter mutator.
   //-----------------------------------------------------------------
   public void setDiameter (int size)
   {
      diameter = size;
   }

   //-----------------------------------------------------------------
   //  Color mutator.
   //-----------------------------------------------------------------
   public void setColor (Color shade)
   {
      color = shade;
   }
```

continue

continue

```
   //-----------------------------------------------------------------
   //  X mutator.
   //-----------------------------------------------------------------
   public void setX (int upperX)
   {
      x = upperX;
   }

   //-----------------------------------------------------------------
   //  Y mutator.
   //-----------------------------------------------------------------
   public void setY (int upperY)
   {
      y = upperY;
   }

   //-----------------------------------------------------------------
   //  Diameter accessor.
   //-----------------------------------------------------------------
   public int getDiameter ()
   {
      return diameter;
   }
```

continue

continue

```
   //----------------------------------------------------------------
   //  Color accessor.
   //----------------------------------------------------------------
   public Color getColor ()
   {
      return color;
   }

   //----------------------------------------------------------------
   //  X accessor.
   //----------------------------------------------------------------
   public int getX ()
   {
      return x;
   }

   //----------------------------------------------------------------
   //  Y accessor.
   //----------------------------------------------------------------
   public int getY ()
   {
      return y;
   }
}
```

Graphical User Interfaces

- A Graphical User Interface (GUI) in Java is created with at least three kinds of objects:
  - components, events, and listeners
- *Components* are objects that represent screen elements:
  - labels, buttons, text fields, menus, etc.
- Some components are *containers* that hold and organize other components:
  - frames, panels, applets, dialog boxes

Events

- An *event* is an object that represents some activity to which we may want to respond

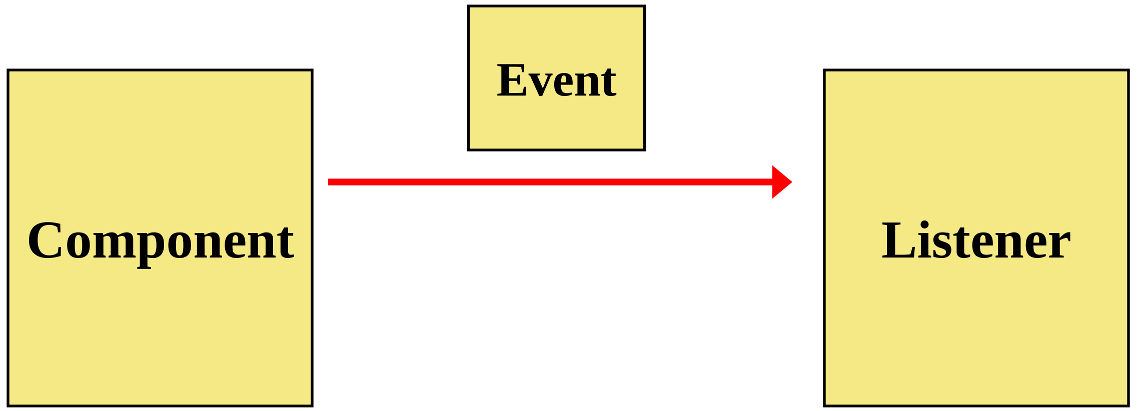
- For example, we may want our program to perform some action when the following occurs:
  - the mouse is moved
  - the mouse is dragged
  - a mouse button is clicked
  - a graphical button is pressed
  - a keyboard key is pressed
  - a timer expires

"If a tree falls in a forest and no one is around to hear it, does it make a sound?”

Events and Listeners

- The Java API contains several classes that represent typical events
- Components, such as a graphical button, generate (or fire) an event when it occurs
- We set up a *listener* object to respond to an event when it occurs
- We can design listener objects to take whatever actions are appropriate when an event occurs

Events and Listeners



A component object generates an event

A corresponding listener object is designed to respond to the event

When the event occurs, the component calls the appropriate method of the listener, passing an object that describes the event

GUI Development

- To create a Java program that uses a GUI we must:

  1. instantiate and set up the necessary components
  2. implement listener classes for any events we care about
  3. establish the relationship between listeners and the components that generate the corresponding events

- Let's now explore some new components and see how this all comes together

Buttons

- A *push button* is defined by the `JButton` class

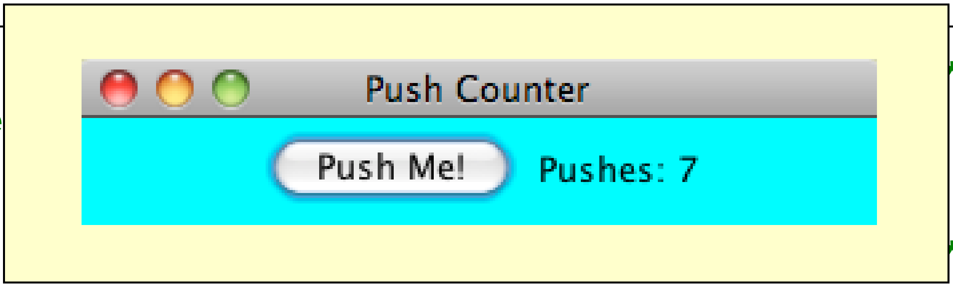
- It generates an *action event*

- The `PushCounter` example displays a push button that increments a counter each time it is pushed

- See `PushCounter.java`
- See `PushCounterPanel.java`

```java
//********************************************************************
//  PushCounter.java       Authors: Lewis/Loftus
//
//  Demonstrates a graphical user interface and an event listener.
//********************************************************************

import javax.swing.JFrame;

public class PushCounter
{
   //-----------------------------------------------------------------
   //  Creates the main program frame.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Push Counter");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add(new PushCounterPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```

```
//************                                   *************
//  PushCounte
//
//  Demonstrat                                   listener.
//************                                   *************

import javax.swing.JFrame;

public class PushCounter
{
   //-----------------------------------------------------------------
   //  Creates the main program frame.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Push Counter");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add(new PushCounterPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```



```
//********************************************************************
//  PushCounterPanel.java       Authors: Lewis/Loftus
//
//  Demonstrates a graphical user interface and an event listener.
//********************************************************************

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

public class PushCounterPanel extends JPanel
{
   private int count;
   private JButton push;
   private JLabel label;

   //-----------------------------------------------------------------
   //  Constructor: Sets up the GUI.
   //-----------------------------------------------------------------
   public PushCounterPanel ()
   {
      count = 0;

      push = new JButton ("Push Me!");
      push.addActionListener (new ButtonListener());
```

continue

continue

```
      label = new JLabel ("Pushes: " + count);

      add (push);
      add (label);

      setPreferredSize (new Dimension(300, 40));
      setBackground (Color.cyan);
   }

   //*****************************************************************
   //  Represents a listener for button push (action) events.
   //*****************************************************************
   private class ButtonListener implements ActionListener
   {
      //--------------------------------------------------------------
      //  Updates the counter and label when the button is pushed.
      //--------------------------------------------------------------
      public void actionPerformed (ActionEvent event)
      {
         count++;
         label.setText("Pushes: " + count);
      }
   }
}
```

Push Counter Example

- The components of the GUI are the button, a label to display the counter, a panel to organize the components, and the main frame
- The `PushCounterPanel` class represents the panel used to display the button and label
- The `PushCounterPanel` class is derived from `JPanel` using inheritance
- The constructor of `PushCounterPanel` sets up the elements of the GUI and initializes the counter to zero

Push Counter Example

- The `ButtonListener` class is the listener for the action event generated by the button
- It is implemented as an *inner class*, which means it is defined within the body of another class
- That facilitates the communication between the listener and the GUI components
- Inner classes should only be used in situations where there is an intimate relationship between the two classes and the inner class is not needed in any other context

Push Counter Example

- Listener classes are written by implementing a *listener interface*
- The `ButtonListener` class implements the `ActionListener` interface
- An interface is a list of methods that the implementing class must define
- The only method in the `ActionListener` interface is the `actionPerformed` method
- The Java API contains interfaces for many types of events

Push Counter Example

- The `PushCounterPanel` constructor:
  - instantiates the `ButtonListener` object
  - establishes the relationship between the button and the listener by the call to `addActionListener`
- When the user presses the button, the button component creates an `ActionEvent` object and calls the `actionPerformed` method of the listener
- The `actionPerformed` method increments the counter and resets the text of the label

Quick Check

Which object in the Push Counter example generated the event?

What did it do then?

Quick Check

Which object in the Push Counter example generated the event?

The button component generated the event.

What did it do then?

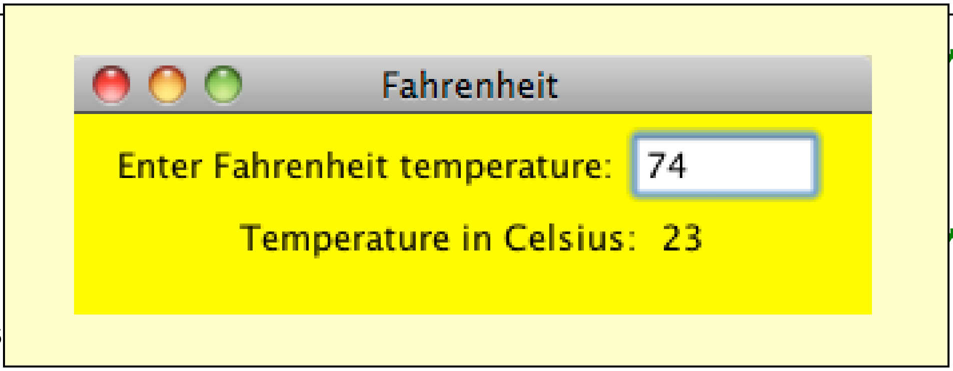
It called the actionPerformed method of the listener object that had been registered with it.

Text Fields

- Let's look at another GUI example that uses another type of component
- A *text field* allows the user to enter one line of input
- If the cursor is in the text field, the text field object generates an action event when the enter key is pressed
- See `Fahrenheit.java`
- See `FahrenheitPanel.java`

```java
//********************************************************************
//  Fahrenheit.java       Author: Lewis/Loftus
//
//  Demonstrates the use of text fields.
//********************************************************************

import javax.swing.JFrame;

public class Fahrenheit
{
   //-----------------------------------------------------------------
   //  Creates and displays the temperature converter GUI.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Fahrenheit");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      FahrenheitPanel panel = new FahrenheitPanel();

      frame.getContentPane().add(panel);
      frame.pack();
      frame.setVisible(true);
   }
}
```

```
//************
//  Fahrenheit
//
//  Demonstrat
//************
```



Enter Fahrenheit temperature: 74

Temperature in Celsius: 23

```
**************

**************

import javax.s

public class Fahrenheit
{
   //-----------------------------------------------------------------
   //  Creates and displays the temperature converter GUI.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Fahrenheit");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      FahrenheitPanel panel = new FahrenheitPanel();

      frame.getContentPane().add(panel);
      frame.pack();
      frame.setVisible(true);
   }
}
```

```
//********************************************************************
//  FahrenheitPanel.java       Author: Lewis/Loftus
//
//  Demonstrates the use of text fields.
//********************************************************************

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

public class FahrenheitPanel extends JPanel
{
   private JLabel inputLabel, outputLabel, resultLabel;
   private JTextField fahrenheit;

   //-----------------------------------------------------------------
   //  Constructor: Sets up the main GUI components.
   //-----------------------------------------------------------------
   public FahrenheitPanel()
   {
      inputLabel = new JLabel ("Enter Fahrenheit temperature:");
      outputLabel = new JLabel ("Temperature in Celsius: ");
      resultLabel = new JLabel ("---");

      fahrenheit = new JTextField (5);
      fahrenheit.addActionListener (new TempListener());
```

continue

continue

```
      add (inputLabel);
      add (fahrenheit);
      add (outputLabel);
      add (resultLabel);

      setPreferredSize (new Dimension(300, 75));
      setBackground (Color.yellow);
   }

   //*****************************************************************
   //  Represents an action listener for the temperature input field.
   //*****************************************************************
   private class TempListener implements ActionListener
   {
      //--------------------------------------------------------------
      //  Performs the conversion when the enter key is pressed in
      //  the text field.
      //--------------------------------------------------------------
      public void actionPerformed (ActionEvent event)
      {
         int fahrenheitTemp, celsiusTemp;

         String text = fahrenheit.getText();
```

continue

continue

```
         fahrenheitTemp = Integer.parseInt (text);
         celsiusTemp = (fahrenheitTemp-32) * 5/9;

         resultLabel.setText (Integer.toString (celsiusTemp));
      }
   }
}
```

Fahrenheit Example

- Like the `PushCounter` example, the GUI is set up in a separate panel class
- The `TempListener` inner class defines the listener for the action event generated by the text field
- The `FahrenheitPanel` constructor instantiates the listener and adds it to the text field
- When the user types a temperature and presses enter, the text field generates the action event and calls the `actionPerformed` method of the listener

Exercise

- Modify the Fahrenheit example to take two numbers from the user and display the sum of the inputs