Graphical User Interfaces in Java
Graphical Applications

• 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
GUI Components

- 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

• 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*
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
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 = new JLabel("Question authority,
but raise your hand first.");
JLabel label2 = new JLabel("but raise your hand first.");
primary.add(label1);
primary.add(label2);
frame.getContentPane().add(primary);
frame.pack();
frame.setVisible(true);
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`
import java.awt.*;  
import javax.swing.*;  

public class NestedPanels  
{
    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);  
    }
}
// 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

```java
// 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);
```
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
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);
}
```java
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);
    }
    }
```
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.
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
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

Copyright © 2014 Pearson Education, Inc.
GUI Development

• To create a Java program that uses a GUI we must:
  – instantiate and set up the necessary components
  – implement listener classes for any events we care about
  – 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`
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);
    }
}
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);
    }
}
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

- We discuss interfaces in more detail in Chapter 6
Push Counter Example

1. The **PushCounterPanel constructor**:
   - instantiates the `ButtonListener` object
   - establishes the relationship between the button and the listener by the call to `addActionListener`

2. When the user presses the button, the button component creates an `ActionEvent` object and calls the `actionPerformed` method of the listener

3. 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
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);
    }
}
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();
        panel.add(panel);
        frame.pack();
        frame.setVisible(true);
    }
}
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

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

    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());
    }

    public static void main(String[] args)
    {

    }

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

    class TempListener extends ActionListener
    {
        public void actionPerformed(ActionEvent e)
        {
            double fahrenheitTemp = Double.parseDouble(fahrenheit.getText());
            double celsiusTemp = (fahrenheitTemp - 32) * 5 / 9;
            outputLabel.setText("Temperature in Celsius:");
            resultLabel.setText(fahrenheitTemp + "");
        }
    }
}
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.
Determining Event Sources

• Recall that interactive GUIs require establishing a relationship between components and the listeners that respond to component events

• One listener object can be used to listen to two different components

• The source of the event can be determined by using the `getSource` method of the event passed to the listener

• See `LeftRight.java`
• See `LeftRightPanel.java`
import javax.swing.JFrame;

public class LeftRight {
    
    // Creates the main program frame.
    public static void main(String[] args) {
        JFrame frame = new JFrame("Left Right");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(new LeftRightPanel());
        frame.pack();
        frame.setVisible(true);
    }
}
import javax.swing.JFrame;

public class LeftRight {
    // Creates the main program frame.
    public static void main(String[] args) {
        JFrame frame = new JFrame("Left Right");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

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

        frame.pack();
        frame.setVisible(true);
    }
}
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class LeftRightPanel extends JPanel
{
    private JButton left, right;
    private JLabel label;
    private JPanel buttonPanel;

    continue
public LeftRightPanel()
{
    left = new JButton("Left");
    right = new JButton("Right");

    ButtonListener listener = new ButtonListener();
    left.addActionListener(listener);
    right.addActionListener(listener);

    label = new JLabel("Push a button");

    buttonPanel = new JPanel();
    buttonPanel.setPreferredSize(new Dimension(200, 40));
    buttonPanel.setBackground(Color.blue);
    buttonPanel.add(left);
    buttonPanel.add(right);

    setPreferredSize(new Dimension(200, 80));
    setBackground(Color.cyan);
    add(label);
    add(buttonPanel);
}
private class ButtonListener implements ActionListener {

    // Determines which button was pressed and sets the label text accordingly.

    public void actionPerformed(ActionEvent event) {
        if (event.getSource() == left) {
            label.setText("Left");
        } else {
            label.setText("Right");
        }
    }
}
Dialog Boxes

• A *dialog box* is a window that appears on top of any currently active window

• It may be used to:
  – convey information
  – confirm an action
  – allow the user to enter data
  – pick a color
  – choose a file

• A dialog box usually has a specific, solitary purpose, and the user interaction with it is brief
Dialog Boxes

• The `JOptionPane` class provides methods that simplify the creation of some types of dialog boxes

• See `EvenOdd.java`

• Specialized dialog boxes for choosing colors and files are covered in Chapter 9
import javax.swing.JOptionPane;

public class EvenOdd
{
    public static void main(String[] args)
    {
        String numStr, result;
        int num, again;

        continue
continue

do
{
    numStr = JOptionPane.showInputDialog("Enter an integer: ");
    num = Integer.parseInt(numStr);

    result = "That number is " + ((num%2 == 0) ? "even" : "odd");

    JOptionPane.showMessageDialog(null, result);
    again = JOptionPane.showConfirmDialog(null, "Do Another?");
} while (again == JOptionPane.YES_OPTION);
```java
continue
do {
    numStr = JOptionPane.showInputDialog("Enter an integer: ");
    num = Integer.parseInt(numStr);
    result = "That number is " + ((num%2 == 0) ? "even" : "odd");
    JOptionPane.showMessageDialog(null, result);
    again = JOptionPane.showConfirmDialog(null, "Do Another?");
} while (again == JOptionPane.YES_OPTION);
}
Check Boxes

• A check box is a button that can be toggled on or off

• It is represented by the JCheckBox class

• Unlike a push button, which generates an action event, a check box generates an item event whenever it changes state

• The ItemListener interface is used to define item event listeners

• A check box calls the itemStateChanged method of the listener when it is toggled
Check Boxes

• Let's examine a program that uses check boxes to determine the style of a label's text string

• It uses the `Font` class, which embodies a character font's:
  – family name (such as Times or Courier)
  – style (bold, italic, or both)
  – font size

• See `StyleOptions.java`
• See `StyleOptionsPanel.java`
import javax.swing.JFrame;

public class StyleOptions
{
    // ---
    //  Creates and presents the program frame.
    // ---
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("Style Options");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        StyleOptionsPanel panel = new StyleOptionsPanel();
        frame.getContentPane().add(panel);

        frame.pack();
        frame.setVisible(true);
    }
}
import javax.swing.JFrame;

public class StyleOptions {
    //-------------------------------
    // Creates and presents the program frame.
    //-------------------------------
    public static void main (String[] args) {
        JFrame frame = new JFrame ("Style Options");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        StyleOptionsPanel panel = new StyleOptionsPanel();
        frame.getContentPane().add(panel);
        frame.pack();
        frame.setVisible (true);
    }
}
public class StyleOptionsPanel extends JPanel {
  private JLabel saying;
  private JCheckBox bold, italic;
  
  continue
Sets up a panel with a label and some check boxes that control the style of the label's font.

```java
public StyleOptionsPanel()
{
    saying = new JLabel("Say it with style!");
    saying.setFont(new Font("Helvetica", Font.PLAIN, 36));

    bold = new JCheckBox("Bold");
    bold.setBackground(Color.cyan);
    italic = new JCheckBox("Italic");
    italic.setBackground(Color.cyan);

    StyleListener listener = new StyleListener();
    bold.addItemListener(listener);
    italic.addItemListener(listener);

    add(saying);
    add(bold);
    add(italic);

    setBackground(Color.cyan);
    setPreferredSize(new Dimension(300, 100));
}
```
continue

//*******************************************************************************
//  Represents the listener for both check boxes.
//*******************************************************************************
private class StyleListener implements ItemListener
{
  //-----------------------------------------------
  //  Updates the style of the label font style.
  //-----------------------------------------------
  public void itemStateChanged(ItemEvent event)
  {
    int style = Font.PLAIN;

    if (bold.isSelected())
      style = Font.BOLD;

    if (italic.isSelected())
      style += Font.ITALIC;

    saying.setFont(new Font("Helvetica", style, 36));
  }
}
Radio Buttons

- A group of *radio buttons* represents a set of mutually exclusive options – only one can be selected at any given time

- When a radio button from a group is selected, the button that is currently "on" in the group is automatically toggled off

- To define the group of radio buttons that will work together, each radio button is added to a *ButtonGroup* object

- A radio button generates an action event
Radio Buttons

- Let's look at a program that uses radio buttons to determine which line of text to display

- See `QuoteOptions.java`
- See `QuoteOptionsPanel.java`
import javax.swing.JFrame;

public class QuoteOptions
{
    //---
    //  Creates and presents the program frame.
    //---
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("Quote Options");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        QuoteOptionsPanel panel = new QuoteOptionsPanel();
        frame.getContentPane().add(panel);

        frame.pack();
        frame.setVisible(true);
    }
}
import javax.swing.JFrame;

public class QuoteOptions {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Quote Options");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        QuoteOptionsPanel panel = new QuoteOptionsPanel();
        frame.getContentPane().add(panel);
        frame.pack();
        frame.setVisible(true);
    }
}
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class QuoteOptionsPanel extends JPanel
{
    private JLabel quote;
    private JRadioButton comedy, philosophy, carpentry;
    private String comedyQuote, philosophyQuote, carpentryQuote;

    //-----------------------------------------------------------------
    // Sets up a panel with a label and a set of radio buttons
    // that control its text.
    //-----------------------------------------------------------------
    public QuoteOptionsPanel()
    {
        comedyQuote = "Take my wife, please.";
        philosophyQuote = "I think, therefore I am.";
        carpentryQuote = "Measure twice. Cut once.";

        quote = new JLabel(comedyQuote);
        quote.setFont(new Font("Helvetica", Font.BOLD, 24));
        quote.setLineWrap(true);
        add(quote);
        add(comedy);
        add(philosophy);
        add(carpentry);
    }

    public void radioActionPerformed(ActionEvent event)
    {
        String selectedQuote = "
        if (event.getSource() == comedy)
            selectedQuote = comedyQuote;
        else if (event.getSource() == philosophy)
            selectedQuote = philosophyQuote;
        else if (event.getSource() == carpentry)
            selectedQuote = carpentryQuote;

        quote.setText(selectedQuote);
        quote.setLineWrap(true);
        quote.setFont(new Font("Helvetica", Font.BOLD, 24));
    }
}
comedy = new JRadioButton("Comedy", true);
comedy.setBackground(Color.green);
philosophy = new JRadioButton("Philosophy");
philosophy.setBackground(Color.green);
carpentry = new JRadioButton("Carpentry");
carpentry.setBackground(Color.green);

ButtonGroup group = new ButtonGroup();
group.add(comedy);
group.add(philosophy);
group.add(carpentry);

QuoteListener listener = new QuoteListener();
comedy.addActionListener(listener);
philosophy.addActionListener(listener);
carpentry.addActionListener(listener);

add(quote);
add(comedy);
add(philosophy);
add(carpentry);

setBackground(Color.green);
setPreferredSize(new Dimension(300, 100));
}
continue

// *************************************************************
// Represents the listener for all radio buttons
// *************************************************************
private class QuoteListener implements ActionListener {
    // --------------------------------------------------------------
    // Sets the text of the label depending on which radio
    // button was pressed.
    // --------------------------------------------------------------
    public void actionPerformed(ActionEvent event) {
        Object source = event.getSource();

        if (source == comedy)
            quote.setText(comedyQuote);
        else
            if (source == philosophy)
                quote.setText(philosophyQuote);
            else
                quote.setText(carpentryQuote);
    }
}