Java – GUI Programming
(Layout and Button)
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

• 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 that is used to display a GUI-based Java application.

- A frame is 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 to be displayed.
Labels

- A *label* is a GUI component that displays a line of text

- 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

- See [Authority.java](#)

- This program is not interactive, but the frame can be repositioned and resized
import java.awt.*;
import javax.swing.*;
public class Authority {
    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));
        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);
    }
}
Running Authority.class
Nested Panels

• 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 {
  // 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 root panel
  JPanel rootPanel = new JPanel();
  rootPanel.add (subPanel1);
  rootPanel.add (subPanel2);

  // Create and set the JFrame
  JFrame frame = new JFrame ("Nested Panels");
  frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
  frame.add (rootPanel);
  frame.pack ();
  frame.setVisible (true);
}
// 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);
NestedPanels.java - Sample Execution

• The following is a sample execution of NestedPanels.class
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.
Smiling Face Example

- The `SmilingFace` program draws a face by defining the `paintComponent` method of a panel.
- See `SmilingFace.java`.
- See `SmilingFacePanel.java`.
- The main method of the `SmilingFace` class instantiates a `SmilingFacePanel` and displays it.
- The `SmilingFacePanel` class is derived from the `JPanel` class using inheritance.
import javax.swing.JFrame;

public class SmilingFace
{
    public static void main (String[] args)
    {
        JFrame frame = new JFrame ("Smiling Face");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        SmilingFacePanel panel = new SmilingFacePanel();

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

public class SmilingFacePanel extends JPanel
{
    private final int BASEX = 120, BASEY = 60; // base point for head

    public SmilingFacePanel ()
    {
        setBackground (Color.blue);
        setPreferredSize (new Dimension(320, 200));
        setFont (new Font("Arial", Font.BOLD, 16));
    }
}
package DrawFace;

public class Face {

    public void paintComponent(Graphics page) {
        super.paintComponent(page);

        page.setColor(Color.yellow);
        page.fillOval(BASEX, BASEY, 80, 80);  // head
        page.fillOval(BASEX-5, BASEY+20, 90, 40);  // ears

        page.setColor(Color.black);
        page.drawOval(BASEX+20, BASEY+30, 15, 7);  // eyes
        page.drawOval(BASEX+45, BASEY+30, 15, 7);
        page.fillOval(BASEX+25, BASEY+31, 5, 5);   // pupils
        page.fillOval(BASEX+50, BASEY+31, 5, 5);

        page.drawArc(BASEX+20, BASEY+25, 15, 7, 0, 180);  // eyebrows
        page.drawArc(BASEX+45, BASEY+25, 15, 7, 0, 180);

        page.drawArc(BASEX+35, BASEY+40, 15, 10, 180, 180);  // nose
        page.drawArc(BASEX+20, BASEY+50, 40, 15, 180, 180);  // mouth
    }
}

------------------------------------------------------------------
//  Draws a face.
//  Dimensions: 80x80 (head), 90x40 (ears)
//  Positions: BASEX, BASEY
------------------------------------------------------------------
page.setColor (Color.white);
page.drawString ("Always remember that you are unique!",
             BASEX-105, BASEY-15);
page.drawString ("Just like everyone else.", BASEX-45, BASEY+105);
SmilingFace.java - Sample Execution

- The following is a sample execution of SmilingFace.class
Smiling Face Example

• Every Swing component has a `paintComponent` method

• The `paintComponent` method accepts a `Graphics` object that represents the graphics context for the panel

• We define the `paintComponent` method to draw the face with appropriate calls to the `Graphics` methods

• Note the difference between drawing on a panel and adding other GUI components to a panel
Splat Example

• The Splat example is structured a bit differently

• It 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
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);
    }
}
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);
    }
}
public void paintComponent (Graphics page) {
    super.paintComponent(page);
    circle1.draw(page);
    circle2.draw(page);
    circle3.draw(page);
    circle4.draw(page);
    circle5.draw(page);
}
import java.awt.*;

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

    public Circle (int size, Color shade, int upperX, int upperY)
    {
        diameter = size;
        color = shade;
        x = upperX;
        y = upperY;
    }
}
public void draw (Graphics page)
{
    page.setColor (color);
    page.fillOval (x, y, diameter, diameter);
}

public void setDiameter (int size)
{
    diameter = size;
}

public void setColor (Color shade)
{
    color = shade;
}
// 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;
}
public Color getColor ()
{
    return color;
}

public int getX ()
{
    return x;
}

public int getY ()
{
    return y;
}
Splat.java - Sample Execution

• The following is a sample execution of Splat.class
Layout Managers

• A layout manager is an object that determines the way that components are arranged in a container

• There are several predefined layout managers defined in the Java standard class library:

  - Flow Layout
  - Border Layout
  - Card Layout
  - Grid Layout
  - GridBag Layout
  - Box Layout
  - Overlay Layout

  Defined in the AWT

  Defined in Swing
Layout Managers

• Every container has a default layout manager, but we can explicitly set the layout manager as well

• Each layout manager has its own particular rules governing how the components will be arranged

• Some layout managers pay attention to a component's preferred size or alignment, while others do not

• A layout manager attempts to adjust the layout as components are added and as containers are resized
Layout Managers

• We can use the `setLayout` method of a container to change its layout manager

```java
JPanel panel = new JPanel();
panel.setLayout(new BorderLayout());
```

• The following example uses a *tabbed pane*, a container which permits one of several panes to be selected

• See [LayoutDemo.java](#)
• See [IntroPanel.java](#)
import javax.swing.*;

public class LayoutDemo
{
    public static void main (String[] args)
    {
        JFrame frame = new JFrame ("Layout Manager Demo");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        JTabbedPane tp = new JTabbedPane();
        tp.addTab ("Intro", new IntroPanel());
        tp.addTab ("Flow", new FlowPanel());
        tp.addTab ("Border", new BorderPanel());
        tp.addTab ("Grid", new GridPanel());
        tp.addTab ("Box", new BoxPanel());
    }
}
frame.getContentPane().add(tp);
frame.pack();
frame.setVisible(true);
}
//***************************************************************
// IntroPanel.java
//
// Represents the introduction panel for the LayoutDemo program.
//***************************************************************

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

public class IntroPanel extends JPanel
{
    //-----------------------------------------------------------------
    // Sets up this panel with two labels.
    //-----------------------------------------------------------------
    public IntroPanel()
    {
        setBackground (Color.green);

        JLabel l1 = new JLabel ("Layout Manager Demonstration");
        JLabel l2 = new JLabel ("Choose a tab to see an example of " +
                                 "a layout manager.");

        add (l1);
        add (l2);
    }
}
LayoutDemo.java - Sample Execution

• The following is a sample execution of LayoutDemo.class
Flow Layout

- *Flow layout* puts as many components as possible on a row, then moves to the next row
- Rows are created as needed to accommodate all of the components
- Components are displayed in the order they are added to the container
- Each row of components is centered horizontally in the window by default, but could also be aligned left or right
- Also, the horizontal and vertical gaps between the components can be explicitly set
- See [FlowPanel.java](#)
import java.awt.*;
import javax.swing.*;

public class FlowPanel extends JPanel
{
    public FlowPanel()
    {
        setLayout (new FlowLayout());
        setBackground (Color.green);

        JButton b1 = new JButton("BUTTON 1");
        JButton b2 = new JButton("BUTTON 2");
        JButton b3 = new JButton("BUTTON 3");
JButton b4 = new JButton ("BUTTON 4");
JButton b5 = new JButton ("BUTTON 5");

add (b1);
add (b2);
add (b3);
add (b4);
add (b5);
FlowPanel.java - Sample Execution

• The following is a sample execution of FlowPanel.class
Border Layout

- A border layout defines five areas into which components can be added.
Border Layout

- Each area displays one component (which could be a container such as a JPanel)
- Each of the four outer areas enlarges as needed to accommodate the component added to it
- If nothing is added to the outer areas, they take up no space and other areas expand to fill the void
- The center area expands to fill space as needed
- See BorderPanel.java
import java.awt.*;
import javax.swing.*;

public class BorderPanel extends JPanel
{
    // Sets up this panel with a button in each area of a border
    // layout to show how it affects their position, shape, and size.
    public BorderPanel()
    {
        setLayout (new BorderLayout());
        setBackground (Color.green);

        JButton b1 = new JButton ("BUTTON 1");
        JButton b2 = new JButton ("BUTTON 2");
        JButton b3 = new JButton ("BUTTON 3");
JButton b4 = new JButton ("BUTTON 4");
JButton b5 = new JButton ("BUTTON 5");

    add (b1, BorderLayout.CENTER);
    add (b2, BorderLayout.NORTH);
    add (b3, BorderLayout.SOUTH);
    add (b4, BorderLayout.EAST);
    add (b5, BorderLayout.WEST);
BorderPanel.java - Sample Execution

- The following is a sample execution of BorderPanel.class
Grid Layout

• A *grid layout* presents a container’s components in a rectangular grid of rows and columns

• One component is placed in each cell of the grid, and all cells have the same size

• As components are added to the container, they fill the grid from left-to-right and top-to-bottom (by default)

• The size of each cell is determined by the overall size of the container

• See [GridPanel.java](GridPanel.java)
import java.awt.*;
import javax.swing.*;

public class GridPanel extends JPanel
{
    // Sets up this panel with some buttons to show how grid layout affects their position, shape, and size.
    public GridPanel()
    {
        setLayout (new GridLayout (2, 3));
        setBackground (Color.green);

        JButton b1 = new JButton ("BUTTON 1");
        JButton b2 = new JButton ("BUTTON 2");
        JButton b3 = new JButton ("BUTTON 3");
JButton b4 = new JButton ("BUTTON 4");
JButton b5 = new JButton ("BUTTON 5");

add (b1);
add (b2);
add (b3);
add (b4);
add (b5);
GridPanel.java - Sample Execution

- The following is a sample execution of GridPanel.class
Box Layout

• A box layout organizes components horizontally (in one row) or vertically (in one column)

• Components are placed top-to-bottom or left-to-right in the order in which they are added to the container

• By combining multiple containers using box layout, many different configurations can be created

• Multiple containers with box layouts are often preferred to one container that uses the more complicated gridbag layout manager

• The details of Box Layout can be found in the textbook
Graphical User Interfaces

• A Graphical User Interface (GUI) in Java is created with at least three kinds of objects:
  – components
  – events
  – listeners

• We've previously discussed components, which 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 clicked
  – a keyboard key is pressed
  – a timer expires

• Events often correspond to user actions, but not always
Events and Listeners

• The Java standard class library contains several classes that represent typical events

• Components, such as a graphical button, generate (or fire) an event when it occurs

• A listener object "waits" for an event to occur and responds accordingly

• We can design listener objects to take whatever actions are appropriate when an event occurs
Events and Listeners

A component object may generate 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

• Generally we use components and events that are predefined by classes in the Java class library

• Therefore, 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 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 a component that allows the user to initiate an action by pressing a graphical button using the mouse.
- 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 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());

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);
   }
}
PushCounter.java - Sample Execution

• The following is a sample execution of PushCounter.class
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 is 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 class library contains interfaces for many types of events.
Push Counter Example

• The PushCounterPanel constructor:
  – instantiated 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
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 component generates an action event when the enter key is pressed

• See Fahrenheit.java
• See FahrenheitPanel.java
Fahrenheit.java

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);
    }
}
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());

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();

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

    resultLabel.setText (Integer.toString (celsiusTemp));
}
}
Fahrenheit.java - Sample Execution

• The following is a sample execution of Fahrenheit.class
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
- The `actionPerformed` method computes the conversion and updates the result label