Module 12

Building Java GUIs : Swing

Ref: Creating a GUI With JFC/Swing

Objectives

- Describe the **JFC *Swing*** technology *MFC*
- Define ***Swing***
- Identify the Swing packages
- Describe the GUI building blocks: containers, components, and layout managers
- Examine top-level, general-purpose, and special-purpose properties of container
- Examine components
- Examine layout managers
- Build a GUI using Swing components

JFC And Swing

What Are the Java Foundation Classes (JFC)?

Java Foundation Classes are a set of Graphical User Interface (GUI) support packages, including:

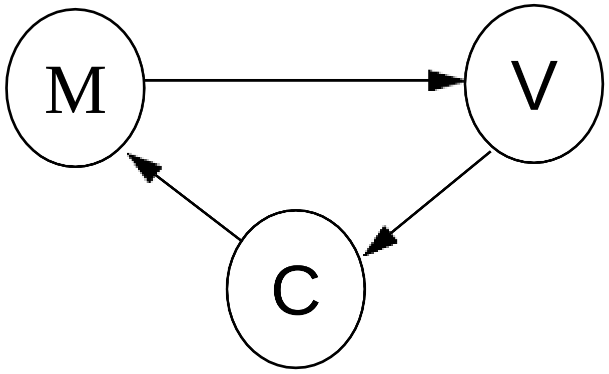
- Abstract Window Toolkit (AWT)
- The Swing component set
- 2D graphics
- Pluggable look-and-feel
- Accessibility
- Drag-and-drop
- Internationalization

What Is Swing?

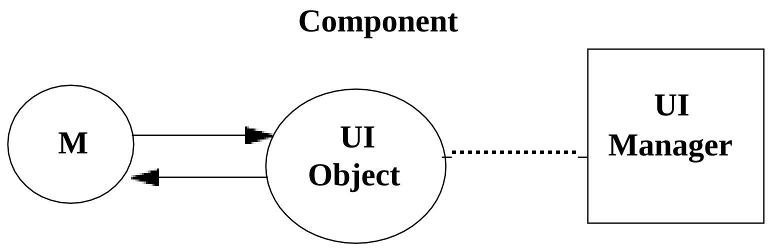
- An enhanced GUI component set
- Provides replacement components for those in the original AWT
- Has special features, such as a pluggable look-and feel

Swing Architecture

- Has its roots in the Model-View-Controller (MVC) architecture



- The Swing components follow Separable Model Architecture



MVC

- MVC is an idealized way of modeling a component as three separate parts:
  - The ***model*** that stores the data that defines the component
  - The ***view*** that creates the visual representation of the component from the data in the model
  - The ***controller*** that deals with user interaction with the component and modifies the model and/or the view in response to a user action as necessary
- MVC concept degenerates into the document/view architecture the Observable class and Observer interface. Sun calls it the **Separable Model architecture**

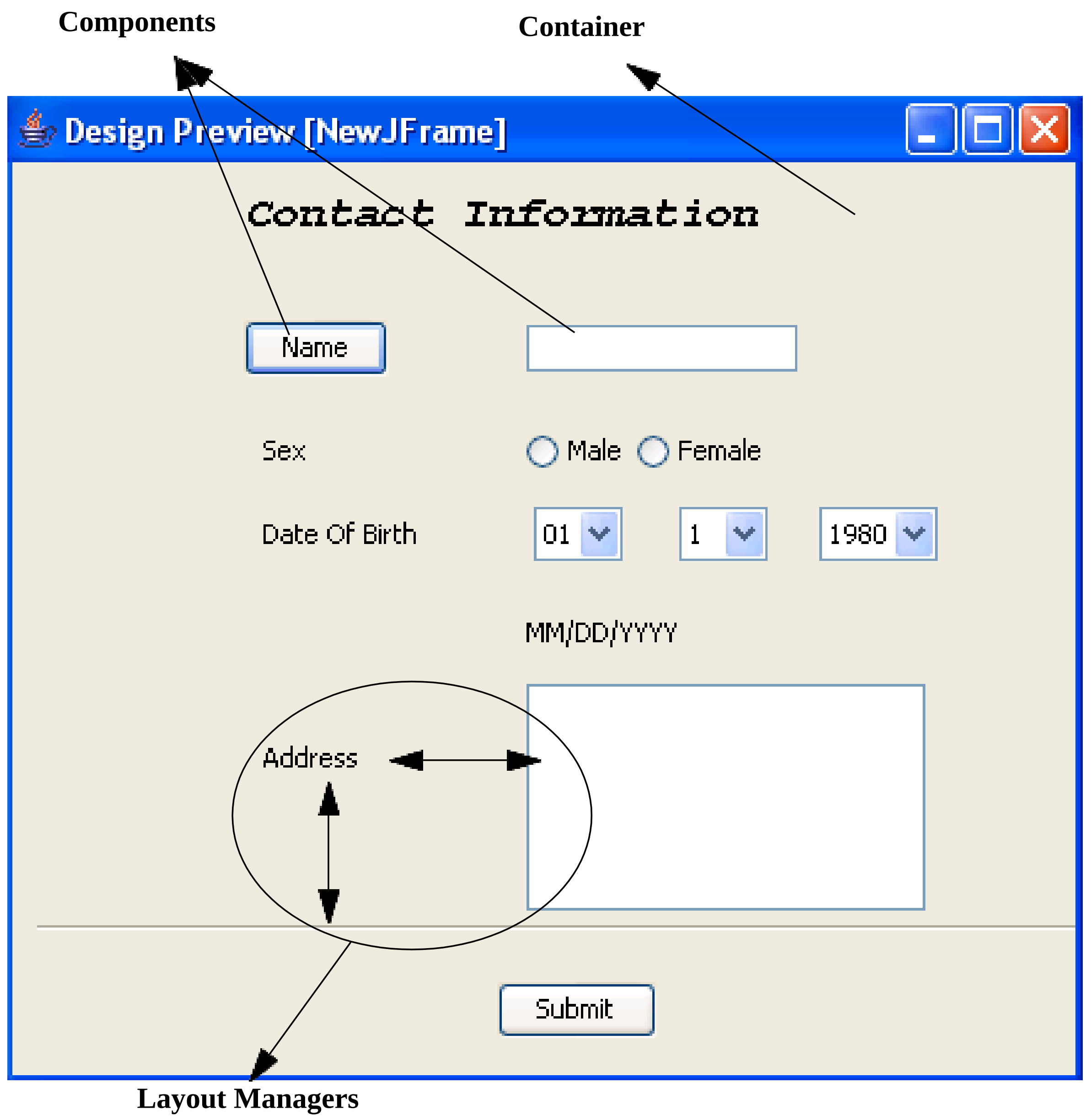
Swing Packages

Package Name	Package Name
`javax.swing`	`javax.swing.colorchooser`
`javax.swing.border`	`javax.swing.filechooser`
`javax.swing.event`	`javax.swing.table`
`javax.swing.undo`	`javax.swing.tree`
`javax.swing.plaf`	`javax.swing.text`
`javax.swing.plaf.basic`	`javax.swing.text.html`
`javax.swing.plaf.metal`	`javax.swing.text.html.parser`
`javax.swing.plaf.multi`	`javax.swing.text.rtf`
`javax.swing.plaf.synth`	`javax.swing.undo`

Examining the Composition of a Java Technology GUI

A Swing API-based GUI is composed of the following elements:

- Containers – Are on top of the GUI containment hierarchy.
- Components – Contain all the GUI components that are derived from the `JComponent` class.
- Layout Managers – Are responsible for laying out components in a container.


Components
Container
Design Preview [NewJFrame]
Contact Information
Name
Sex
Male
Female
Date Of Birth
01
1
1980
MM/DD/YYYY
Address
Submit
Layout Managers

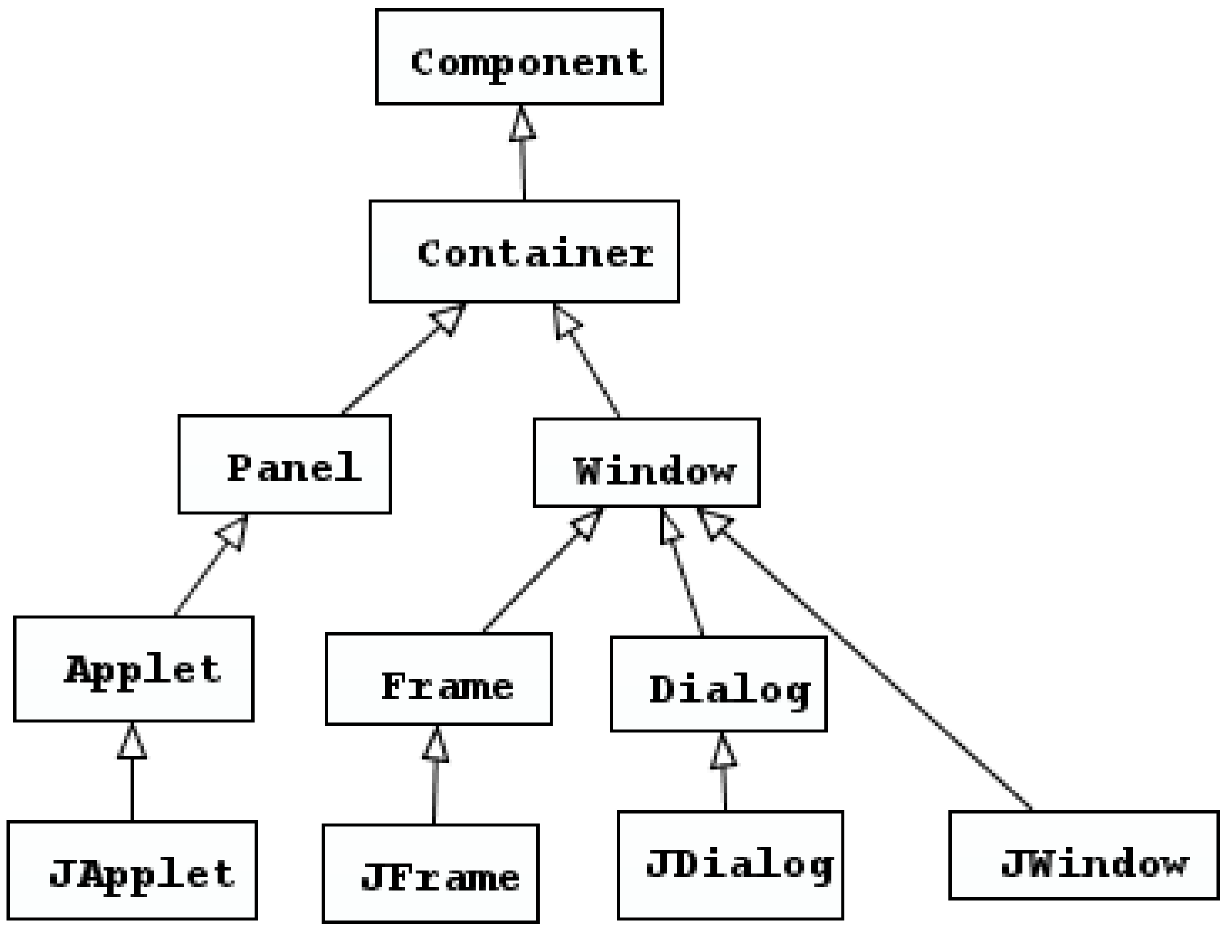
Swing Containers

Swing Containers

Swing containers can be classified into three main categories:

- Top-level containers:
  - `JFrame`, `JWindow`, and `JDialog`
- General-purpose containers:
  - `JPanel`, `JScrollPane`,`JToolBar`,`JSplitPane`, and `JTabbedPane`
- Special-purpose containers:
  - `JInternalFrame` and `JLayeredPane`

Top-Level Containers

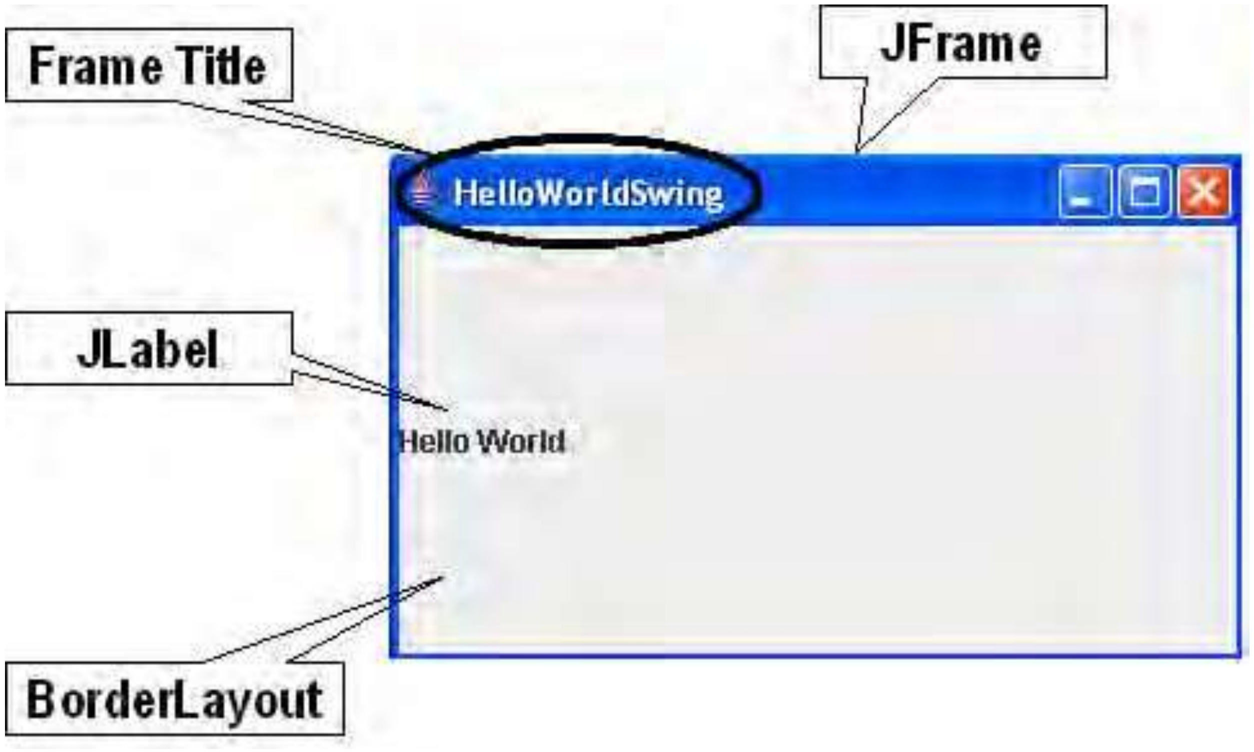


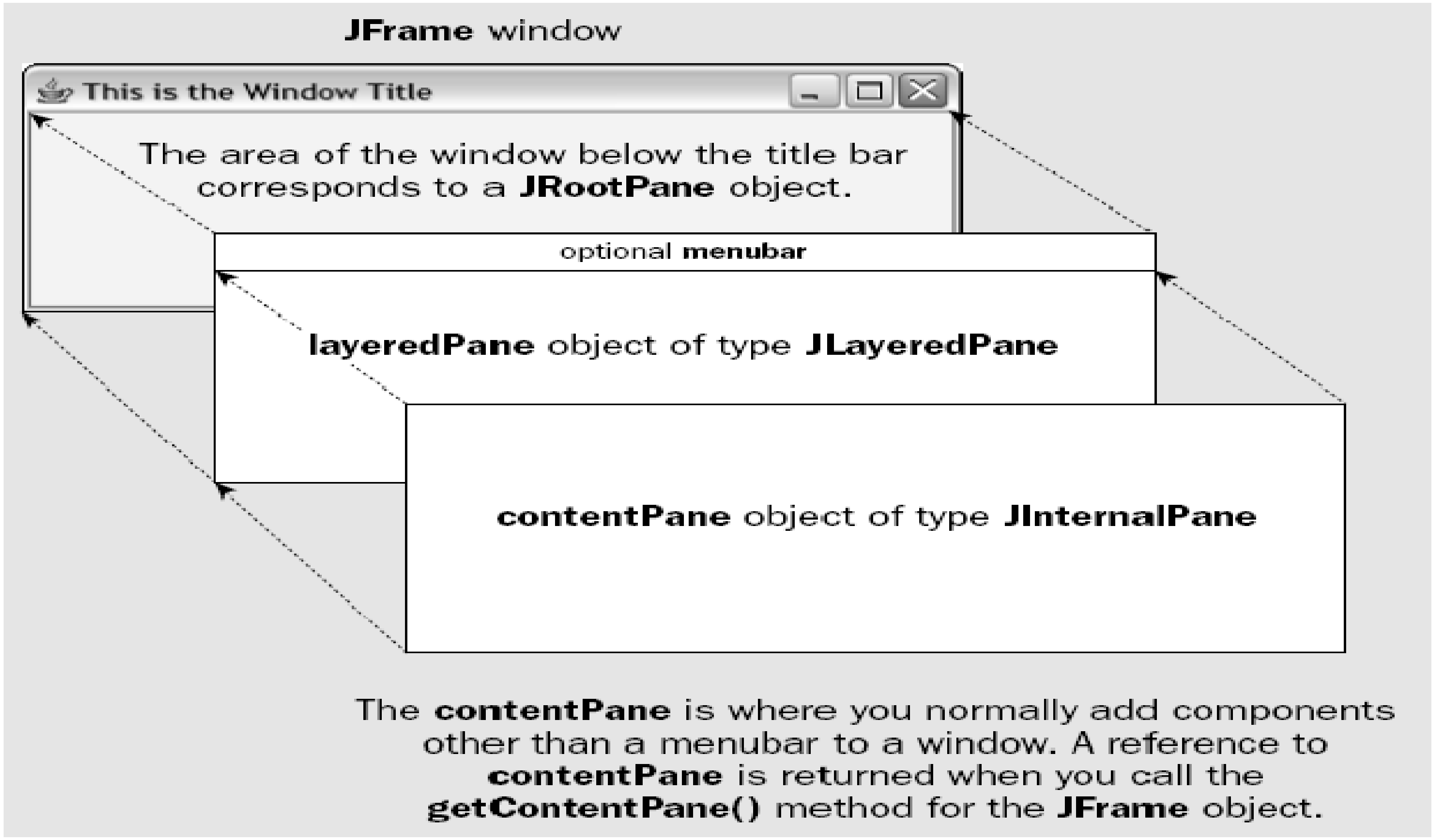
GUI Construction

- Programmatic
- GUI builder tool

Programmatic Construction

The output generated from the program




JFrame window
This is the Window Title
The area of the window below the title bar corresponds to a **JRootPane** object.
optional **menubar**
layeredPane object of type **JLayeredPane**
contentPane object of type **JInternalPane**
The **contentPane** is where you normally add components other than a menubar to a window. A reference to **contentPane** is returned when you call the **getContentPane()** method for the **JFrame** object.

Key Methods

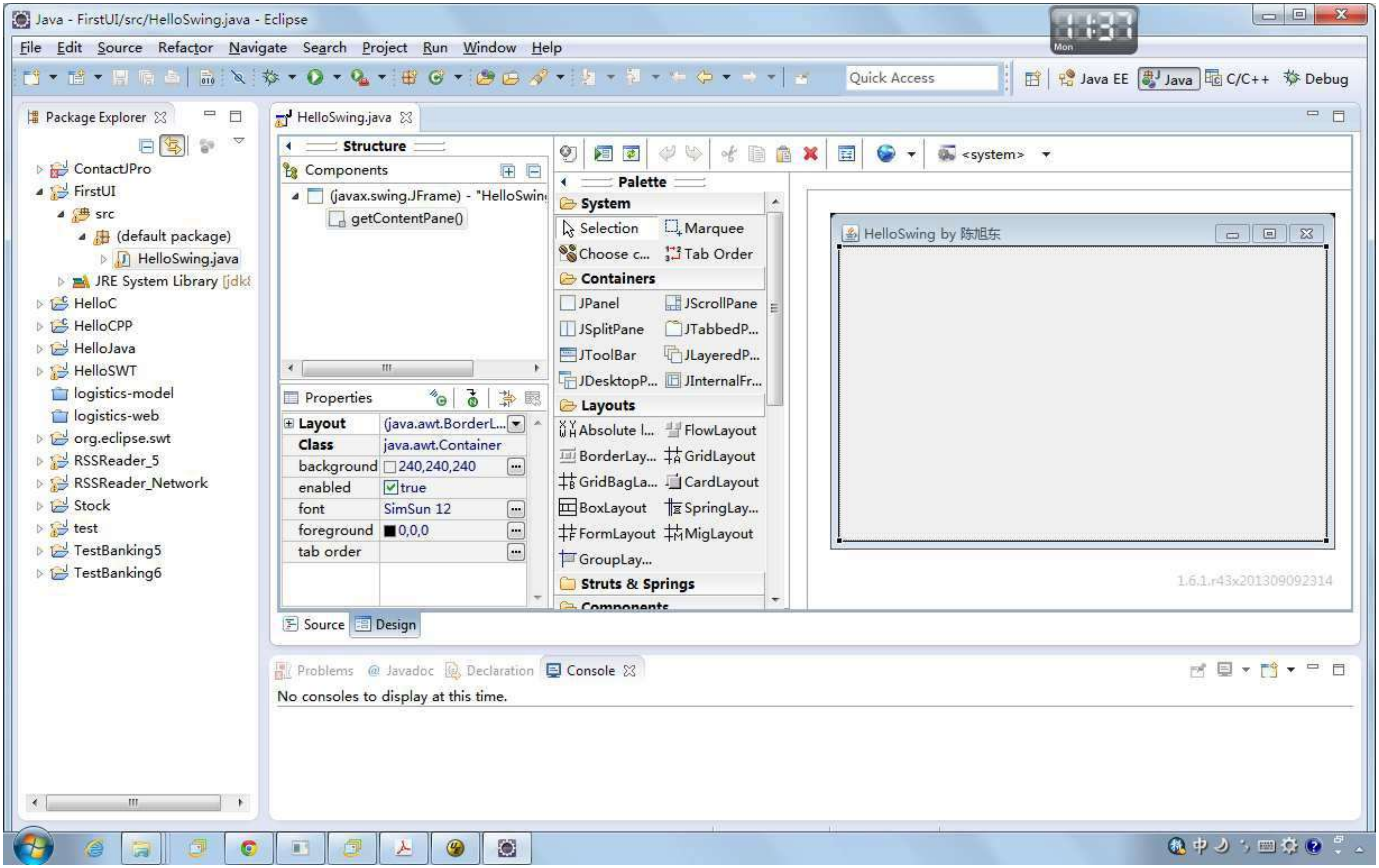
Methods for setting up the `JFrame` and adding `JLabel`:

- `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)` –Creates the program to exit when the close button is clicked.
- `setVisible(true)`–Makes the `JFrame` visible.
- `add(label)`–`JLabel` is added to the content pane not to the `JFrame` directly.

Key Methods

- The SwingUtilities class:
  - SwingUtilites.invokeLater(new Runnable())
- Tasks:
  - Executing GUI application code, such as rendering
  - Handling GUI events
  - Handling time consuming (background) processes

Object-Oriented Programming and Design



Layout	(java.awt.BorderL...
Class	java.awt.Container
background	240,240,240
enabled	✓ true
font	SimSun 12
foreground	■ 0,0,0
tab order	

```
No consoles to display at this time.
```

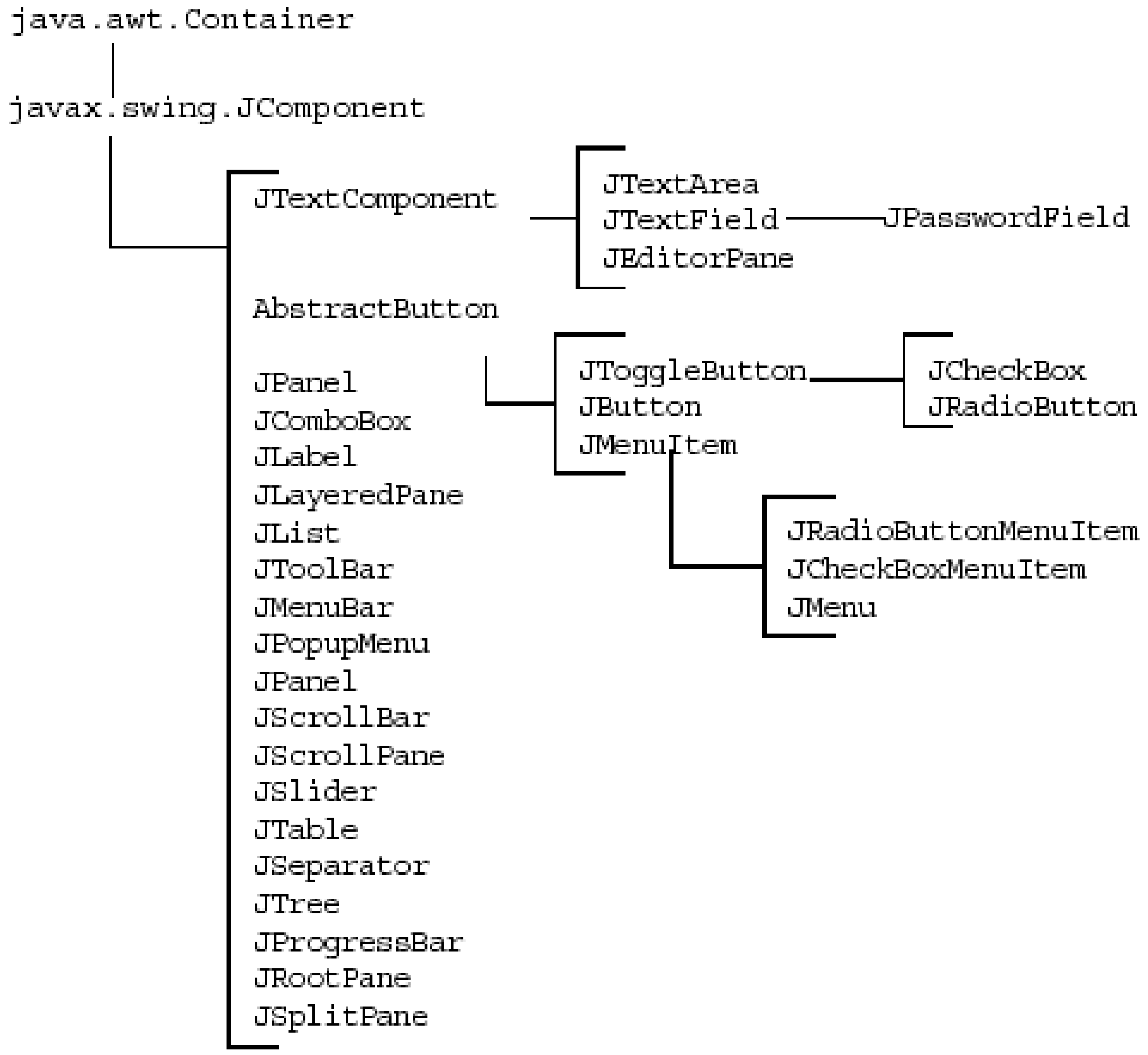
Swing Components

Swing Components

Swing components can be broadly classified as:

- Buttons
- Text components
- Uneditable information display components
- Menus
- Formatted display components
- Other basic controls

Swing Component Hierarchy



Swing Component Properties

Common component properties:

- All the Swing components share some common properties because they all extend `JComponent`.

Component-specific properties:

- Each component defines more specific properties.

Common Component Properties

Property	**Methods**

Border	`Border getBorder()`
	`void setBorder(Border b)`

Background and foreground color	`void setBackground(Color bg)`
	`void setForeground(Color bg)`

Font	`void setFont(Font f)`
Opaque	`void setOpaque(boolean isOpaque)`

Maximum and minimum size	`void setMaximumSize(Dimension d)`
	`void setMinimumSize(Dimension d)`

Alignment	`void setAlignmentX(float ax)`
	`void setAlignmentY(float ay)`

Preferred size	`void setPreferredSize(Dimension ps)`

Component-Specific Properties

The following shows properties specific to `JComboBox`.

Properties	**Methods**
Maximum row count	`void setMaximumRowCount(int count)`
Model	`void setModal(ComboBoxModel cbm)`
Selected index	`int getSelectedIndex()`
Selected Item	`Object getSelectedItem()`
Item count	`int getItemCount()`
Renderer	`void setRenderer(ListCellRenderer ar)`
Editable	`void setEditable(boolean flag)`

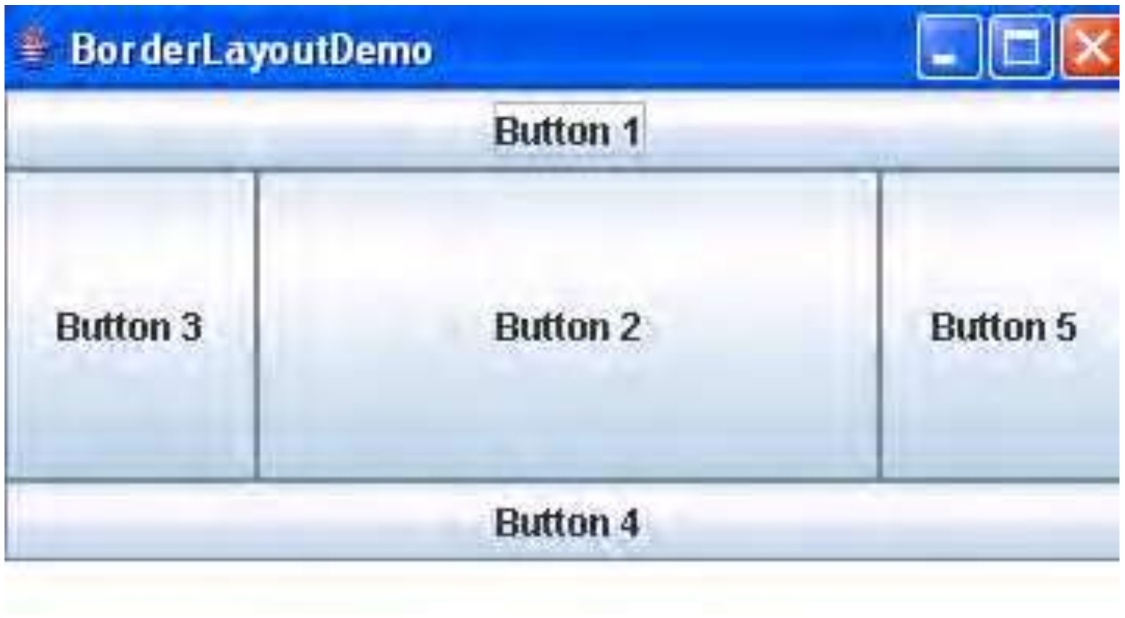
Layout manager

Layout Managers

- Handle problems caused by:
  - GUI resizing by user
  - Operating system differences in fonts
  - Locale-specific text layout requirements
- Layout manager classes:
  - `BorderLayout`
  - `FlowLayout`
  - `BoxLayout`
  - `CardLayout`
  - `GridLayout`
  - `GridBagLayout`

The `BorderLayout` Manager

The `BorderLayout` manager places components in top, bottom, left, right and center locations.



`BorderLayout` Example

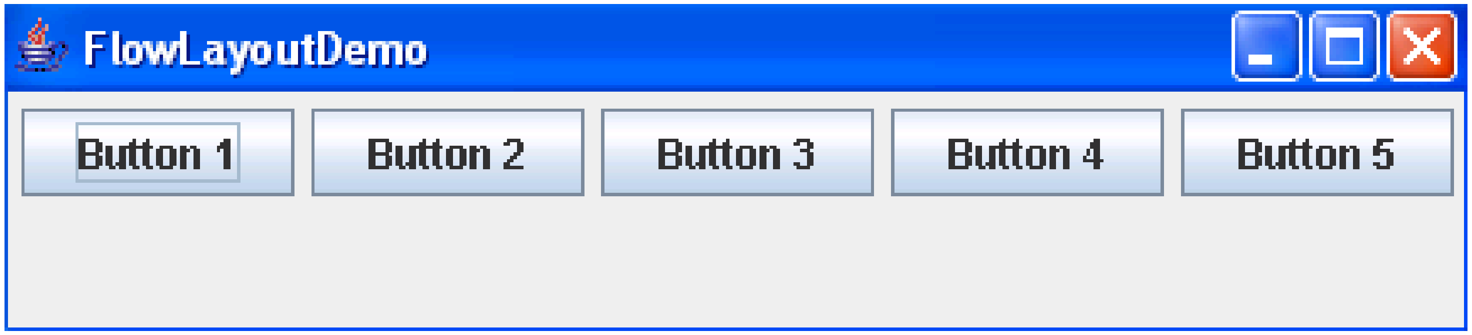
```
import java.awt.*;
import javax.swing.*;

public class BorderExample {
  private JFrame f;
  private JButton bn, bs, bw, be, bc;

  public BorderExample() {
    f = new JFrame("Border Layout");
    bn = new JButton("Button 1");
    bc = new JButton("Button 2");
    bw = new JButton("Button 3");
    bs = new JButton("Button 4");
    be = new JButton("Button 5");
  }

```

BorderLayout Example

```
  public void launchFrame()
    f.add(bn, { BorderLayout.NORTH)
    f.add(bs, ;
    f.add(bw, BorderLayout.SOUTH);
    f.add(be, BorderLayout.WEST);
    f.add(bc, BorderLayout.EAST);
    f.setSize(400,200);
    f.setVisible(true);
  }

  public static void main(String args[]) {
    BorderExample guiWindow2 = new BorderExample();
    guiWindow2.launchFrame();
  }
}

```

The `FlowLayout` Manager

The `FlowLayout` manager places components in a row, and if the row fills, components are placed in the next row.



`FlowLayout` Example

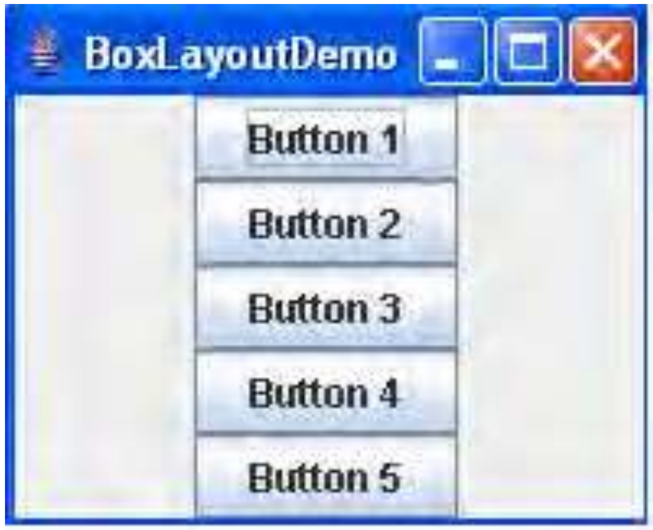
```
import javax.swing.*;
import java.awt.*;

public class LayoutExample {
    private JFrame f;
    private JButton b1;
    private JButton b2;
    private JButton b3;
    private JButton b4;
    private JButton b5;

    public LayoutExample() {
        f = new JFrame("GUI example");
        b1 = new JButton("Button 1");
        b2 = new JButton("Button 2");
        b3 = new JButton("Button 3");
        b4 = new JButton("Button 4");
        b5 = new JButton("Button 5");
    }
```

`FlowLayout` Example

```

    public void launchFrame()
        {  f.setLayout(new
        FlowLayout()); f.add(b1);
        f.add(b2);
        f.add(b3);
        f.add(b4);
        f.add(b5);
        f.pack();
        f.setVisible(true);
    }

    public static void main(String args[]) {
        LayoutExample guiWindow = new LayoutExample();
        guiWindow.launchFrame();
    }

} // end of LayoutExample class
```

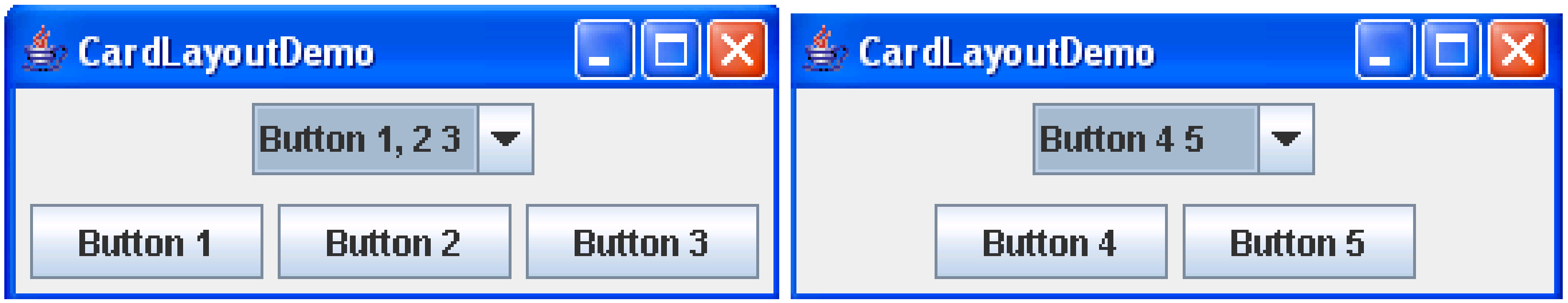
The `BoxLayout` Manager

The `BoxLayout` manager adds components from left to right, and from top to bottom in a single row of column.



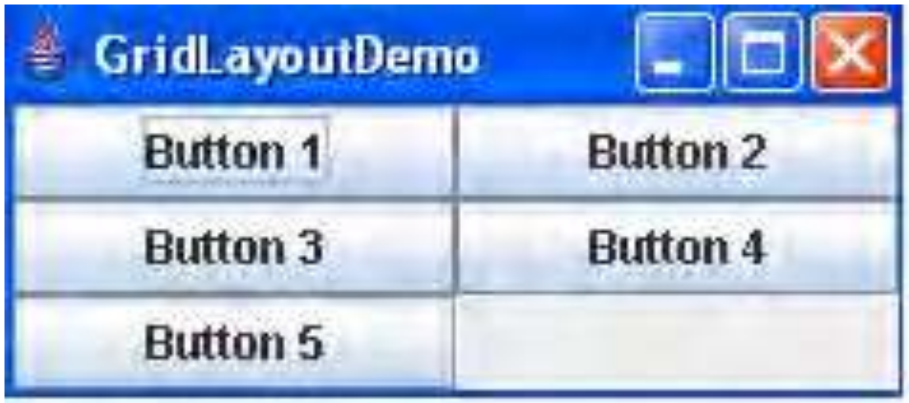
The `CardLayout` Manager

The `CardLayout` manager places the components in different cards. Cards are usually controlled by a combo box.



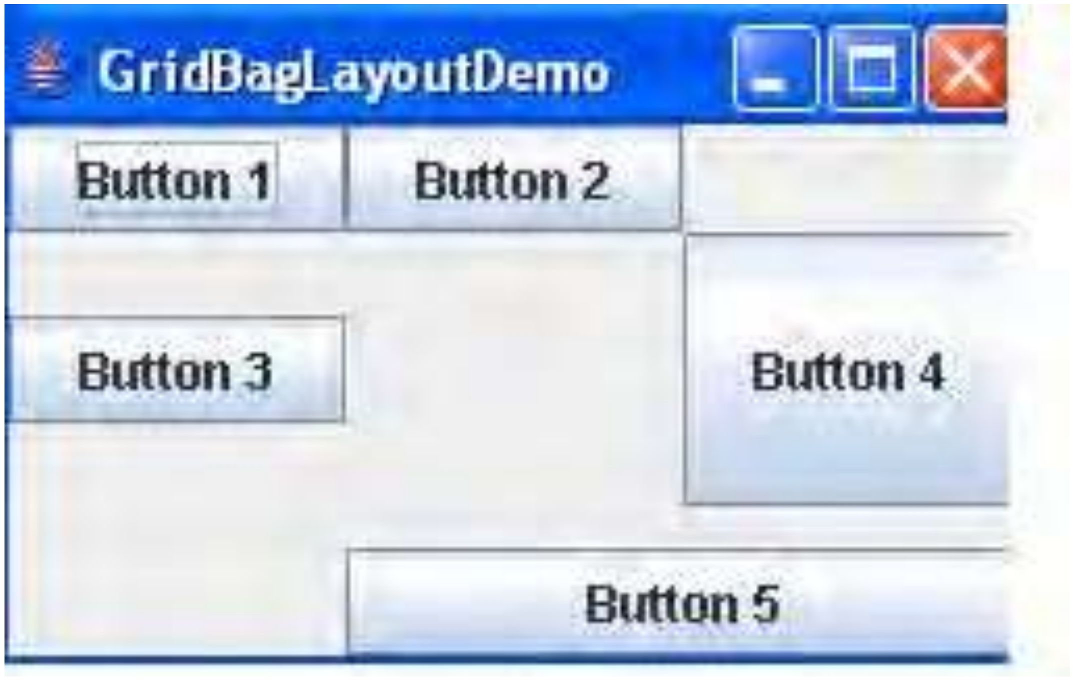
The `GridLayout` Manager

The `GridLayout` manager places components in rows and columns in the form of a grid.



The `GridBagLayout` Manager

The `GridBagLayout` manager arranges components in rows and columns, similar to a grid layout, but provides a wide variety of options for resizing and positioning the components.



Module 13

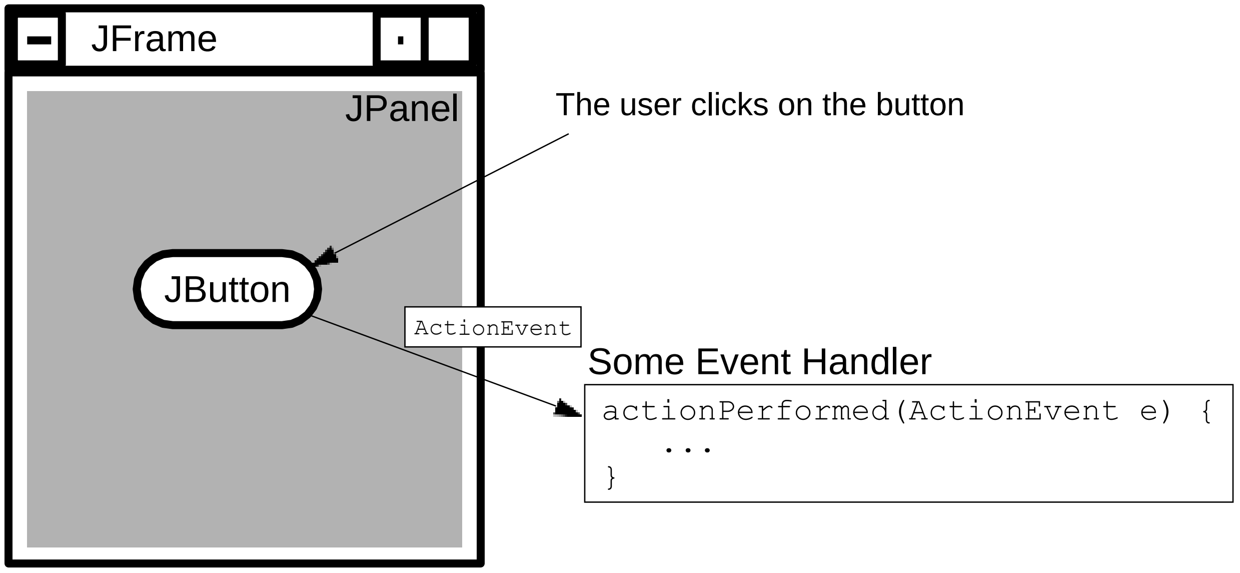
Handling GUI-Generated Events

Objectives

- Define events and event handling
- Examine the Java SE event model
- Describe GUI behavior
- Determine the user action that originated an event
- Develop event listeners

What Is an Event?

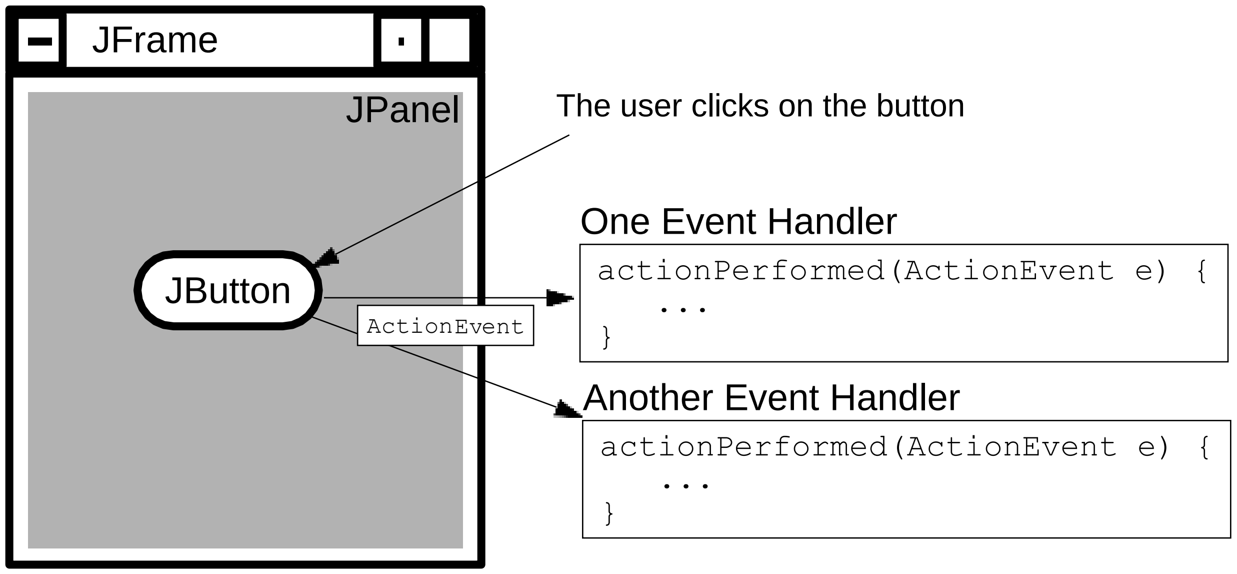
- Events – Objects that describe what happened
- Event sources – The generator of an event
- Event handlers – Object with a method that receives an event object, deciphers it, and processes the user's interaction



```
actionPerformed(ActionEvent e) {
   ...
}
```

Delegation Model

- An event can be sent to many event handlers.



```
actionPerformed(ActionEvent e) {
    ...
}
```

```
actionPerformed(ActionEvent e) {
    ...
}
```

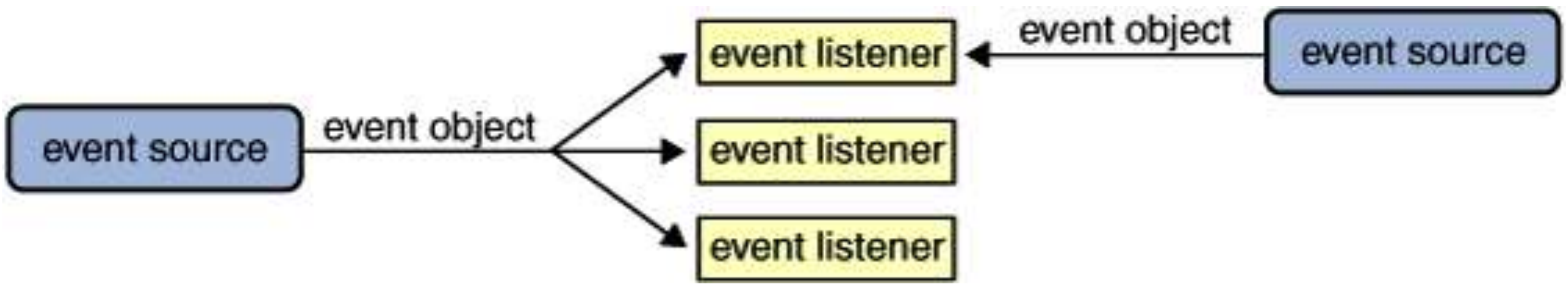
- Event handlers register with components when they are interested in events generated by that component.
    - *addxxxListener()* methods

Delegation Model

- Client objects (handlers) register with a GUI component that they want to observe.
- GUI components trigger only the handlers for the type of event that has occurred.
- Most components can trigger more than one type of event.
- The delegation model distributes the work among multiple classes.

Event Source

- Multiple listeners can register to be notified of events of a particular type from a particular source.
- Also, the same listener can listen to notifications from different objects.



A Listener Example

```
import java.awt.*;
import javax.swing.*;
public class TestButton {
  private JFrame f;
  private JButton b;

  public TestButton() {
    f = new JFrame("Test");
    b = new JButton("Press Me!");
    b.setActionCommand("ButtonPressed");
  }

  public void launchFrame()
    { b.addActionListener(new
    ButtonHandler());
    f.add(b,BorderLayout.CENTER);
    f.pack();
    f.setVisible(true);
```

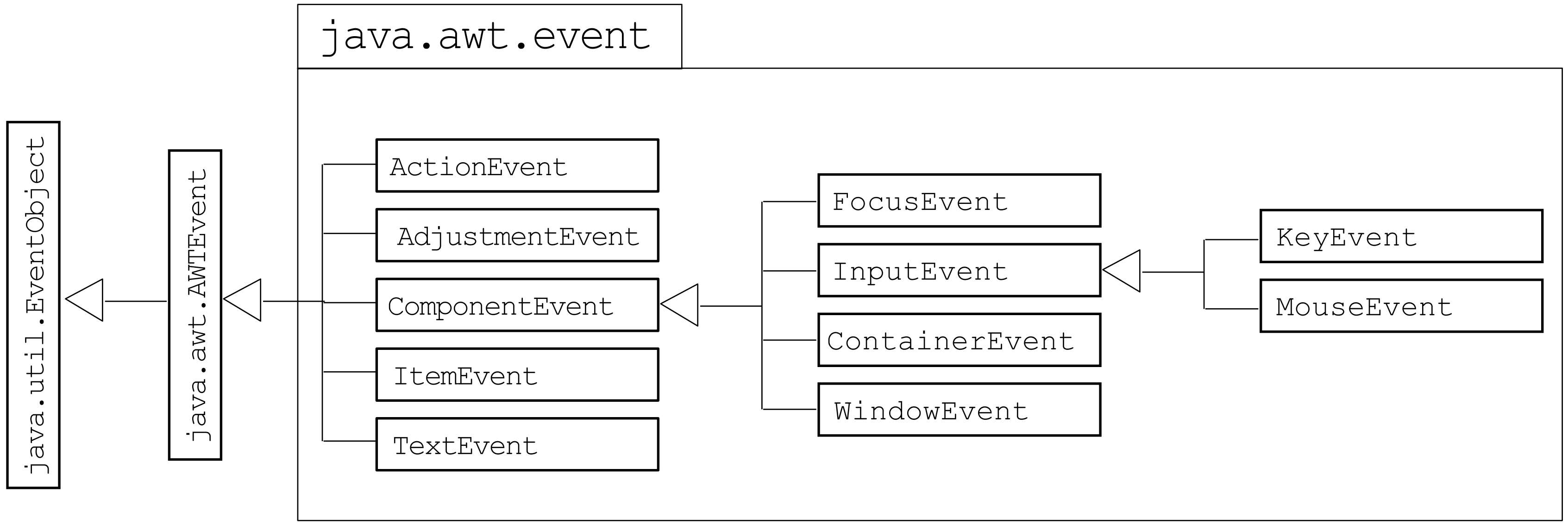
A Listener Example

```

  public static void main(String args[])
    { TestButton guiApp = new
    TestButton(); guiApp.launchFrame();
  }
}
```

Code for the event listener looks like the following:

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

public class ButtonHandler implements ActionListener {
  public void actionPerformed(ActionEvent e)
    { System.out.println("Action occurred");
    System.out.println("Button’s command is: "
                           + e.getActionCommand());
  }
}
```

Event Categories



Method Categories and Interfaces

Category	Interface Name	Methods
Action	`ActionListener`	`actionPerformed(ActionEvent)`
Item	`ItemListener`	`itemStateChanged(ItemEvent)`

Mouse	`MouseListener`	`mousePressed(MouseEvent)`
		`mouseReleased(MouseEvent)`
		`mouseEntered(MouseEvent)`
		`mouseExited(MouseEvent)`
		`mouseClicked(MouseEvent)`

Mouse motion	`MouseMotionListener`	`mouseDragged(MouseEvent)`
		`mouseMoved(MouseEvent)`

Key	`KeyListener`	`keyPressed(KeyEvent)`
		`keyReleased(KeyEvent)`
		`keyTyped(KeyEvent)`

Method Categories and Interfaces

Category	Interface Name	Methods

Focus	`FocusListener`	`focusGained(FocusEvent)`
		`focusLost(FocusEvent)`

Adjustment	`AdjustmentListener`	`adjustmentValueChanged (AdjustmentEvent)`

Component	`ComponentListener`	`componentMoved(ComponentEvent)`
		`componentHidden(ComponentEvent)`
		`componentResized(ComponentEvent)`
		`componentShown(ComponentEvent)`

Method Categories and Interfaces

Category	Interface Name	Methods

Window	`WindowListener`	`windowClosing(WindowEvent)`
		`windowOpened(WindowEvent)`
		`windowIconified(WindowEvent)`
		`windowDeiconified(WindowEvent)`
		`windowClosed(WindowEvent)`
		`windowActivated(WindowEvent)`
		`windowDeactivated(WindowEvent)`

Container	`ContainerListener`	`componentAdded(ContainerEvent)`
		`componentRemoved`
		`(ContainerEvent)`

Window state	`WindowStateListener`	`windowStateChanged(WindowEvent e)`

Window focus	`WindowFocusListener`	`windowGainedFocus(WindowEvent e)`
		`windowLostFocus(WindowEvent e)`

Method Categories and Interfaces

Category	Interface Name	Methods
Mouse wheel	`MouseWheelListener`	`mouseWheelMoved (MouseWheelEvent e)`
Input methods	`InputMethodListener`	`caretPositionChanged (InputMethodEvent e)` `inputMethodTextChnaged (InputMethodEvent e)`
Hierarchy	`HierarchyListener`	`hierarchyChanged (HierarchyEvent e)`
Hierarchy bounds	`HierarchyBoundsListener`	`ancestorMoved(HierarchyEvent e)` `ancestorResized(HierarchyEvent e)`
AWT	`AWTEventListener`	`eventDispatched(AWTEvent e)`
Text	`TextListener`	`textValueChanged(TextEvent)`

Complex Example

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class TwoListener
       implements MouseMotionListener, MouseListener {
  private JFrame f;
  private JTextField tf;

  public TwoListener() {
    f = new JFrame("Two listeners example");
    tf = new JTextField(30);
  }
```

Complex Example

```java

  public void launchFrame() {
    JLabel label = new JLabel("Click and drag the mouse");
    // Add components to the frame
    f.add(label, BorderLayout.NORTH);
    f.add(tf, BorderLayout.SOUTH);
    // Add this object as a listener
    f.addMouseMotionListener(this);
    f.addMouseListener(this);
    // Size the frame and make it visible
    f.setSize(300, 200);
    f.setVisible(true);
  }
```

Complex Example

```

    // These are MouseMotionListener events
    public void mouseDragged(MouseEvent e)  {
      String s =  "Mouse dragging:  X = " + e.getX()
                  + " Y = " + e.getY();
      tf.setText(s);
    }

    public void mouseEntered(MouseEvent e)  {
      String s = "The mouse entered";
      tf.setText(s);
    }

    public void mouseExited(MouseEvent e)
      { String s = "The mouse has left the
      building"; tf.setText(s);
    }
```

Complex Example

```

  // Unused MouseMotionListener method.
  // All methods of a listener must be present in the
  // class even if they are not used.
  public void mouseMoved(MouseEvent e) { }

  // Unused MouseListener methods.
  public void mousePressed(MouseEvent e) { }
  public void mouseClicked(MouseEvent e) { }
  public void mouseReleased(MouseEvent e) { }

  public static void main(String args[]) {
    TwoListener two = new TwoListener();
    two.launchFrame();
  }
}
```

Event Adapters

- The listener classes that you define can extend adapter classes and override only the methods that you need.
- An example is:

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

public class MouseClickHandler extends MouseAdapter {

  // We just need the mouseClick handler, so we use
  // an adapter to avoid having to write all the
  // event handler methods

  public void mouseClicked(MouseEvent e) {
    // Do stuff with the mouse click...
  }
}
```

Event Handling Using Inner Classes

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class TestInner {
  private JFrame f;
  private JTextField tf; // used by inner class

  public TestInner() {
    f = new JFrame("Inner classes example");
    tf = new JTextField(30);
  }

  class MyMouseMotionListener extends MouseMotionAdapter {
      public void mouseDragged(MouseEvent e) {
        String s = "Mouse dragging:  X = "+ e.getX()
                    + " Y = " + e.getY();
        tf.setText(s);
      }
  }
```

Event Handling Using Inner Classes

```

  public void launchFrame() {
    JLabel label = new JLabel("Click and drag the mouse");
    // Add components to the frame
    f.add(label, BorderLayout.NORTH);
    f.add(tf, BorderLayout.SOUTH);
    // Add a listener that uses an Inner class
    f.addMouseMotionListener(new MyMouseMotionListener());
    f.addMouseListener(new MouseClickHandler());
    // Size the frame and make it visible
    f.setSize(300, 200);
    f.setVisible(true);
  }

  public static void main(String args[])
    { TestInner obj = new TestInner();
    obj.launchFrame();
  }
}
```

Event Handling Using Anonymous Classes

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

public class TestAnonymous {
  private JFrame f;
  private JTextField tf;

  public TestAnonymous() {
    f = new JFrame("Anonymous classes example");
    tf = new JTextField(30);
  }

  public static void main(String args[])
    { TestAnonymous obj = new
    TestAnonymous(); obj.launchFrame();
  }

```

Event Handling Using Anonymous Classes

```
  public void launchFrame() {
    JLabel label = new JLabel("Click and drag the mouse");
    // Add components to the frame
    f.add(label, BorderLayout.NORTH);
    f.add(tf, BorderLayout.SOUTH);
    // Add a listener that uses an anonymous class
    f.addMouseMotionListener(new MouseMotionAdapter() {
      public void mouseDragged(MouseEvent e) {
        String s = "Mouse dragging:  X = "+ e.getX()
                    + " Y = " + e.getY();
        tf.setText(s);
      }
    }); // <- note the closing parenthesis
    f.addMouseListener(new MouseClickHandler()); // Not shown
    // Size the frame and make it visible
    f.setSize(300, 200);
    f.setVisible(true);
  }
}
```

Lambda Expressions*

Since JDK 8

lambda expression

- A *lambda expression* is a block of code that you can pass around so it can be executed later, once or multiple times.
- The **Syntax** of Lambda Expressions:
  parameters, the -> arrow, and an expression
- Java is a strongly typed language, lambda expression is simply a block of code, together with the specification of any variables that must be passed to the code.
  (String first, String second)-> first.length() - second.length()

lambda expression

If the code carries out a computation that doesn't fit in a single expression, write it exactly like you would have written a method: enclosed in {} and with explicit return statements.

```
(String first, String second) ->
{
  if (first.length() < second.length()) return -1;
  else if (first.length() > second.length()) return 1;
  else return 0;
}
```

lambda expression

- If a lambda expression has no parameters, you still supply empty parentheses
  () -> { for (int i = 100; i >= 0; i--) System.out.println(i); }
- If the parameter types of a lambda expression can be inferred, you can omit them.
  Comparator<String> comp =
  (first, second) -> first.length() - second.length();
- If a method has a single parameter with inferred type, you can even omit the parentheses
  ActionListener listener =
  event ->System.out.println("The time is " + new Date()");
  //Instead of (event) -> . . . or (ActionEvent event) -> . . .

Functional Interfaces

- if an interface has a single abstract method , such an interface is called a **functional interface**, such as *ActionListener* or *Comparator*. Lambdas are compatible with these interfaces.
- It is best to think of a lambda expression as a function, not an object, and it can be passed to a functional interface.
- It can be much more efficient than using traditional inner classes.
- NOTE : You can’t assign a lambda expression to a variable of type Object—Object is not a functional interface.

Module 14

GUI-Based Applications

Objectives

- Describe how to construct a menu bar, menu, and menu items in a Java GUI
- Understand how to change the color and font of a component
- Look and Feel
  UIManager.setLookAndFeel(”javax.swing.plaf.metal.MetalLookAndFeel”);
- Swing Components
- Swing Dialogs, JColorChooser, JFileChooser
- Drawing
- Printing
- Misc

How to Create a Menu

1. Create a JMenuBar object, and set it into a menu container, such as a JFrame.
2. Create one or more JMenu objects, and add them to the menu bar object.
3. Create one or more JMenuItem objects, and add them to the menu object.
   a. Text menu item
   b. Icon menu item
   c. Icon with text menu item
   d. radio button menu item: JRadioButtonMenuItem
   e. check box menu items : JCheckBoxMenuItem
4. Sub menu is a menu added to the parent menu

   swing/MenuDemo.java

Colors

- Colors:

```
setForeground()
setBackground()
```

- Example:

```
Color purple = new Color(255, 0, 255);
JButton b = new JButton("Purple");
b.setBackground(purple);
```

Fonts

- You can use the `setFont()` method to specify the font used for displaying text
- `Dialog`, `DialogInput`, `Serif`, and `SansSerif` are valid font names
- Example:

```
Font font = new Font("TimesRoman", Font.PLAIN, 14);
```

- Use the `GraphicsEnvironment` class to retrieve the set of all available fonts:

```
GraphicsEnvironment ge =
  GraphicsEnvironment.getLocalGraphicsEnvironment();
Font[] fonts = ge.getAllFonts();
```

java.awt.Toolkit

- The `Toolkit` class is an abstract superclass of all actual implementations of the Abstract Window Toolkit
- Subclasses of `Toolkit` are used to bind the various components to particular native toolkit implementations
- Useful methods:

```
getDefaultToolkit
getImage(String filename)
getScreenResolution
getScreenSize
getPrintJob
```

Printing

- The follow code fragment prints a `Frame`:

```
1   Frame f = new Frame("Print test");
2   Toolkit toolkit = frame.getToolkit();
3   PrintJob job = toolkit.getPrintJob(frame, "Test Printing", null);
4   Graphics g = job.getGraphics();
5   frame.printComponents(g);
6   g.dispose();
7   job.end();
```

1. Obtain graphics object (Line 4).
2. Draw on the graphics object (Line 5).
3. Send the graphics object to printer (Line 6).
4. End the print job (Line 7).

TestPrinting.java

look and feel

- Each look-and-feel is defined by a class the following look-and-feel classes is distributed with the JDK:
  - – Metal look-and-feel : javax.swing.plaf.metal.MetalLookAndFeel
  - – Motif look-and-feel : com.sun.java.swing.plaf.motif.MotifLookAndFeel
  - – Windows : com.sun.java.swing.plaf.windows.WindowsLookAndFeel
  - – WindowsClassic : com.sun.java.swing.plaf.windows.WindowsClassicLookAndFeel

- To programatically specify a L&F, use the **`UIManager.setLookAndFeel()`** method with the fully qualified name of the appropriate subclass of LookAndFeel as its argument.

```
// Set cross-platform Java L&F (also called "Metal")
UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
```

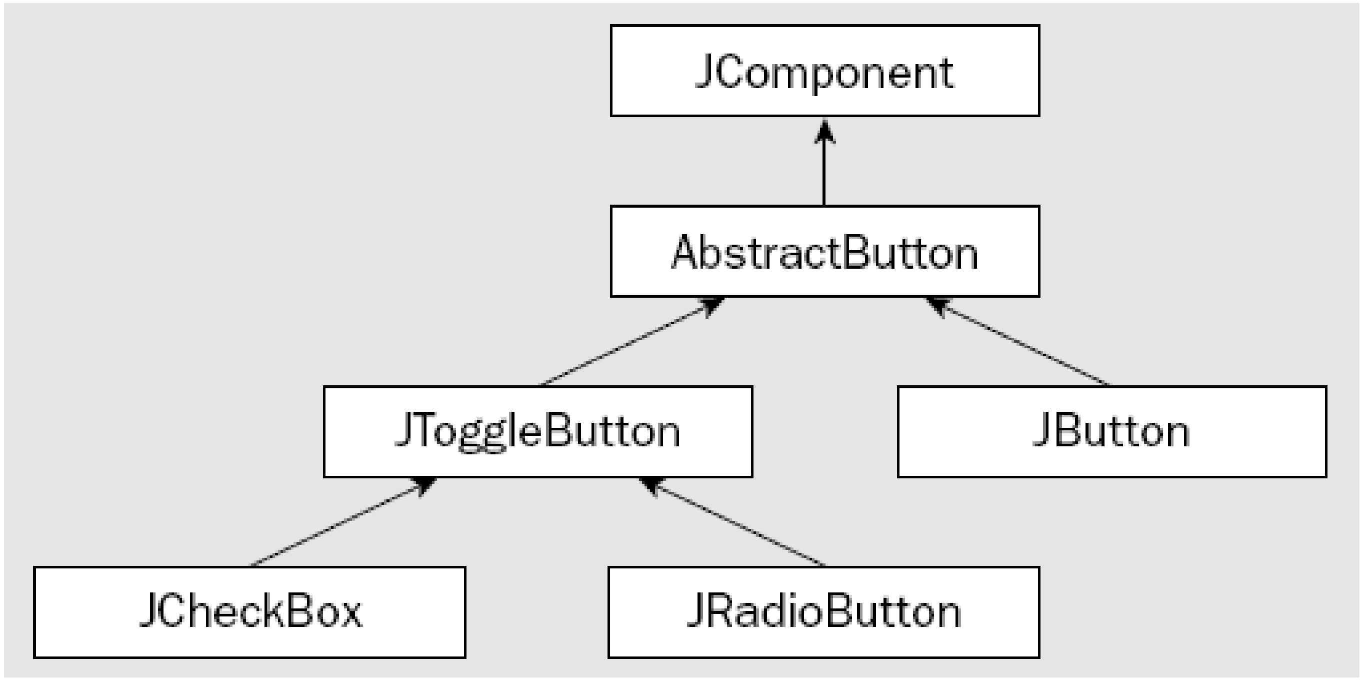
or

```
UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
```

Swing Components

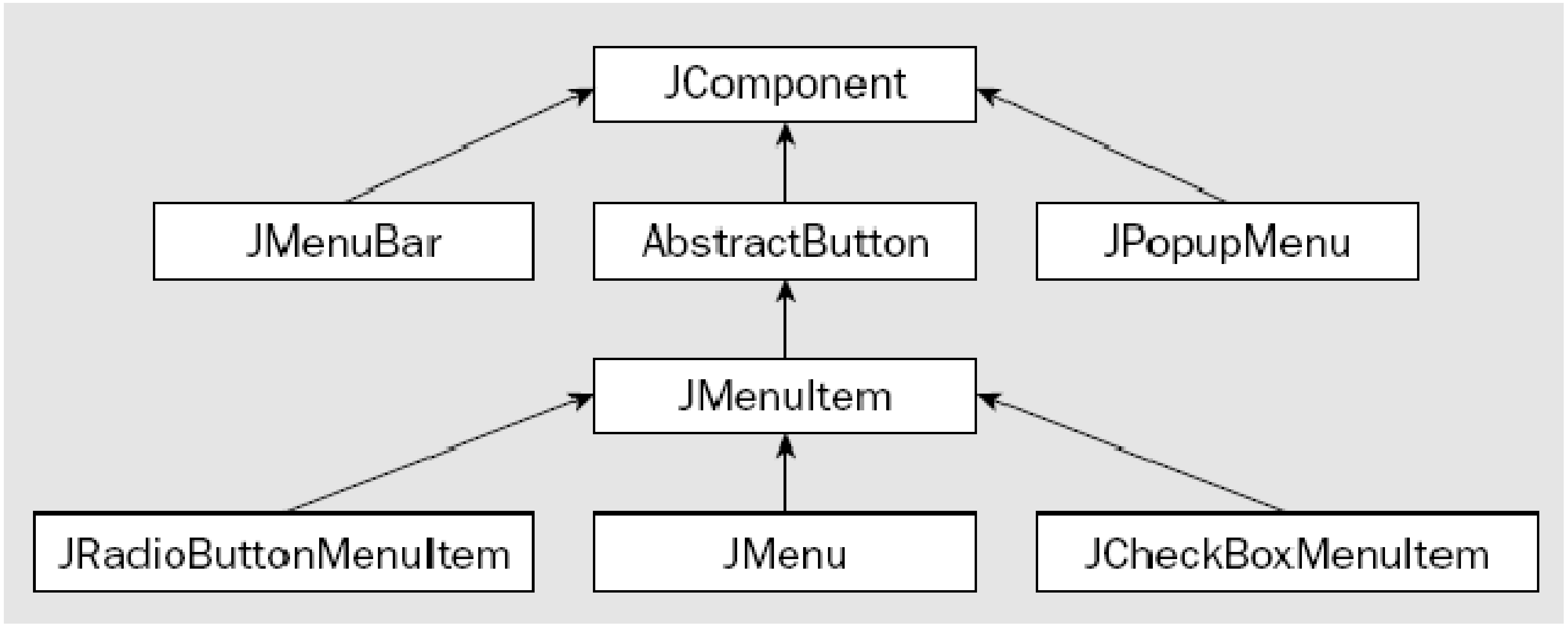
Swing Components

- Button



Swing Components

- Me u



Menu: Swing/Menu/MenuDemo.java

PopupMenu: Swing/Menu/PopupMenuDemo.java

Text Components

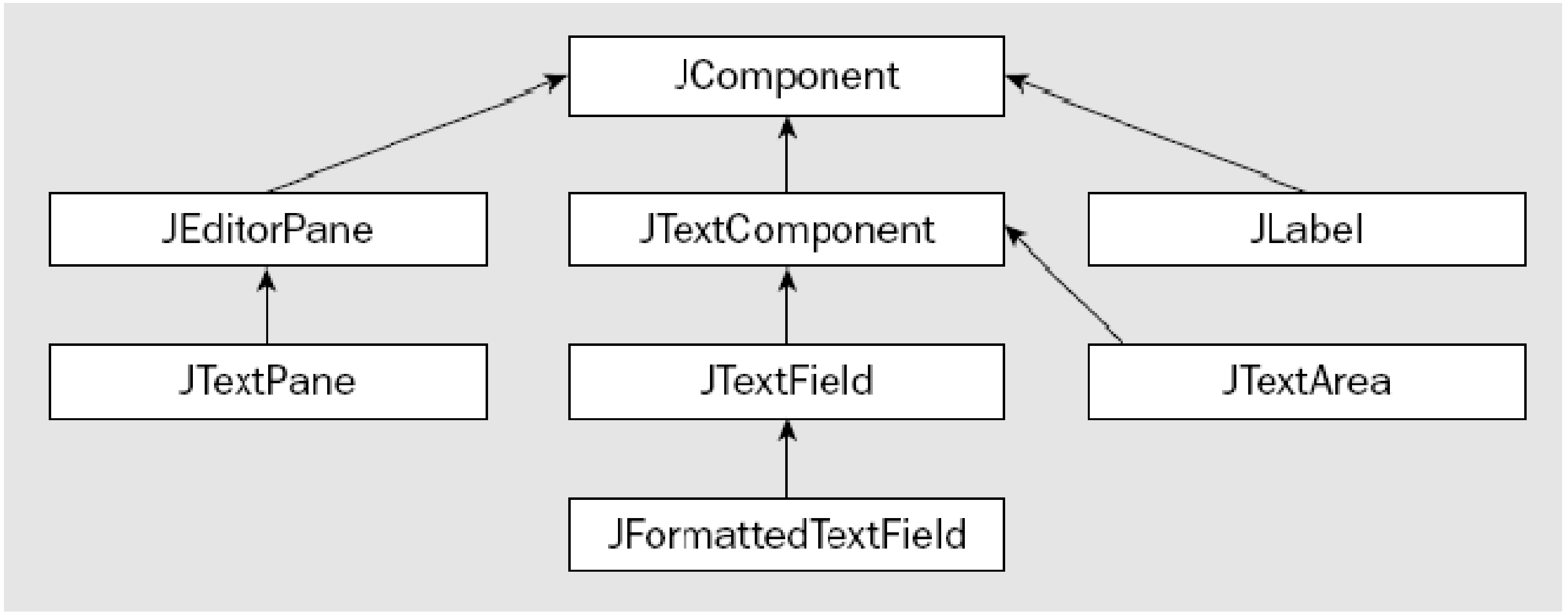
Swing text components can be broadly divided into three categories.

- Text controls – `JTextField`, `JPasswordField` (for user input)
- Plain text areas – `JTextArea` (displays text in plain text, also for multi-line user input
- Styled text areas – `JEditorPane`, `JTextPane` (displays formatted text)

Swing Components

- Text Components

  Swing/TextSamplerDemo.java Swing/TextSamplerDemoHelp.html



- JList and JTable
  - **Swing/ListDemo.java**
  - Swing/TableDemo.java

Swing Components

- ToolBar
  - javax.swing.JToolBar
  - **Example:**
    - Swing/Toolbar
- Icon
  - javax.swing.ImageIcon
  - Example: A Button with an Icon
    - **Swing/Icon**
- Tooltips
  - javax.swing.Action.SHORT_DESCRIPTION
  - javax.swing.AbstractAction.putValue()
  - Example: Implementing Tooltips
    - **Swing/Tooltips**

Dialogs

- A dialog is a window that is displayed within the context of another window—its parent.
- The javax.swing.JDialog defines dialogs, and a JDialo**g** object is a specialized sort of Window.
- You can create either a **modal dialog** or a **non-modal dialog**.(new JDialog(…))
  - A modal dialog inhibits the operation of any other windows in the application until you close the dialog. The dialog window retains the focus as long as it is displayed, and operation of the application cannot continue.
  - A non-modal dialog can be left on the screen for as long as you want, since it doesn’t block interaction with other windows in the application.
- ***A Simple Modal Dialog***
  - Swing/TestAboutDialog.java

Instant Dialogs

- javax.swing.JOptionPane class defines a number of static methods that will create and display standard modal dialogs
  - showMessageDialog() :
    - displays a simple, one-button dialog.
  - showOptionDialog() :
    - displays a customized dialog, can display a variety of buttons with customized button text, and can contain a standard text message or a collection of components.
  - showConfirmDialog() and showInputDialog() are used less often
- Example:

  \Swing\DialogDemo

Choosing Custom Colors

- javax.swing.JColorChooser enable any color to be chosen.
  - Swing/ColorChooserDemo.java

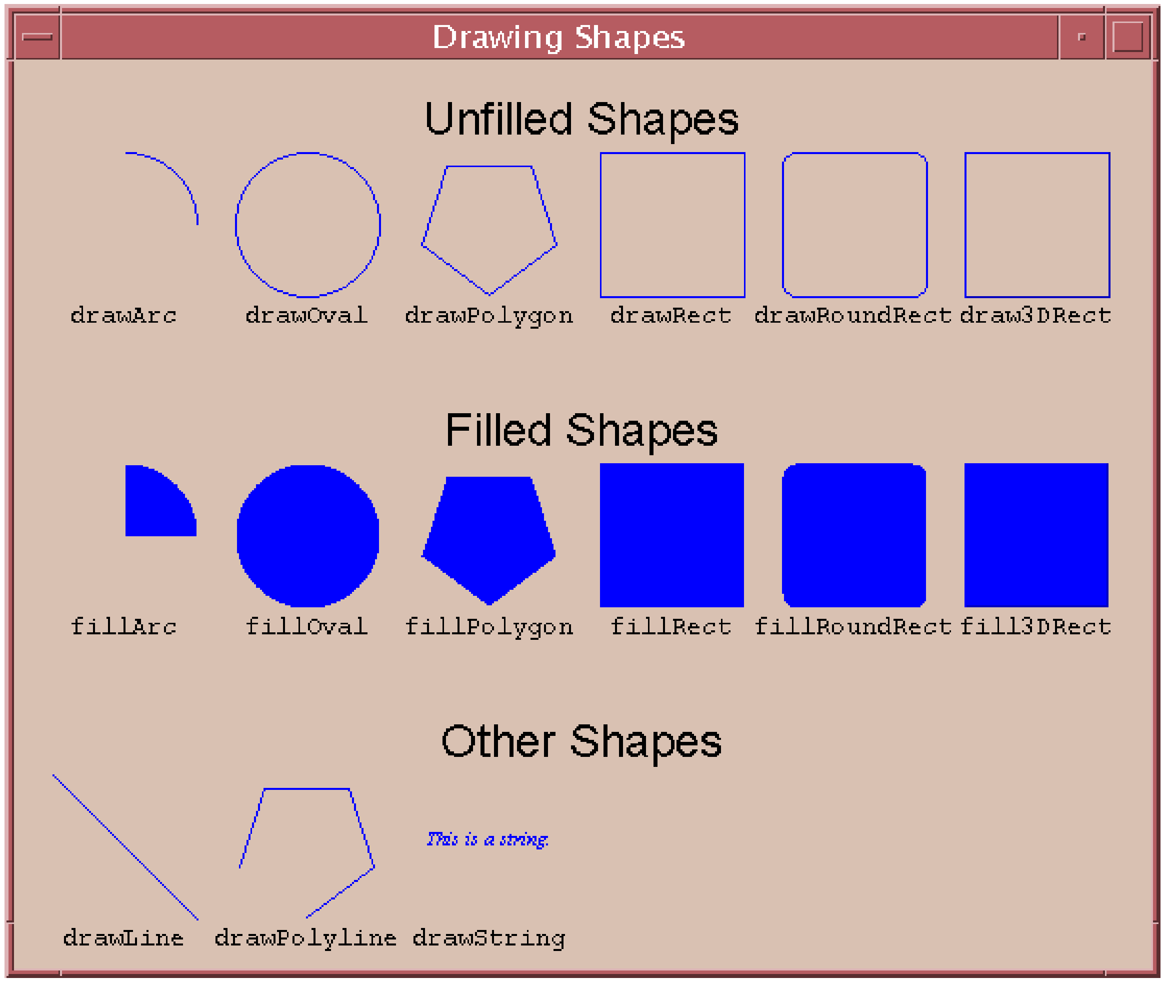
Using a File Chooser

- File choosers provide a GUI for navigating the file system, and then either choosing a file or directory from a list or entering the name of a file or directory.
  - To display a file chooser, **you** u**sually use the** JFileChooser API to show a modal dialog containing the file chooser.
  - Another way to present a file chooser is to add an instance of JFileChooser to a container.
- Example - open/save dialog:
  - TextEditor/TextEditor.java

Drawing in Components

- You can draw in any `Component` (although AWT provides the `Canvas` class just for this purpose)
- Typically, you would create a subclass of `Canvas` and override the `paint` method
- The `paint` method is called every time the component is shown (for example, if another window was overlapping the component and then removed)
- Every component has a `Graphics` object
- The `Graphics` class implements many drawing methods

Drawing With the Graphics Object



Misc

- Desktop
  - Desktop/*DesktopDemo.java*
- SystemTray
  - SystemTray/*TrayIconDemo.java*
- Splash Example
  - Spalsh/*SplashDemo.java*
- Image Processing
  - Image/*ImageTest.java*

Other GUI

- Eclipse SWT(Standard Widget Toolkit): IBM
  - For desktop Applications
  - Simpler than Swing
  - Feeling as the same of platform
  - import org.eclipse.swt.widget.*;
- JavaFX: Oracle
  - https://openjfx.io/index.html
- Dart/Flutter: Google Android
- HTML5: html,css,javascript
- Java Applet
- Flex - Flash Application, last to 2020.12
  - MXML + Scripts
  - Can use external Java Classes

Summary

- JFC And Swing
- Containers, Components and Layout manager
- Event Delegation Model
- Event Handling - Event Listener, Event Adapters
- Create a Menu
- Colors, Fonts, look and feel
- Swing Components
- Dialogs
- Drawing, Printing, and Misc