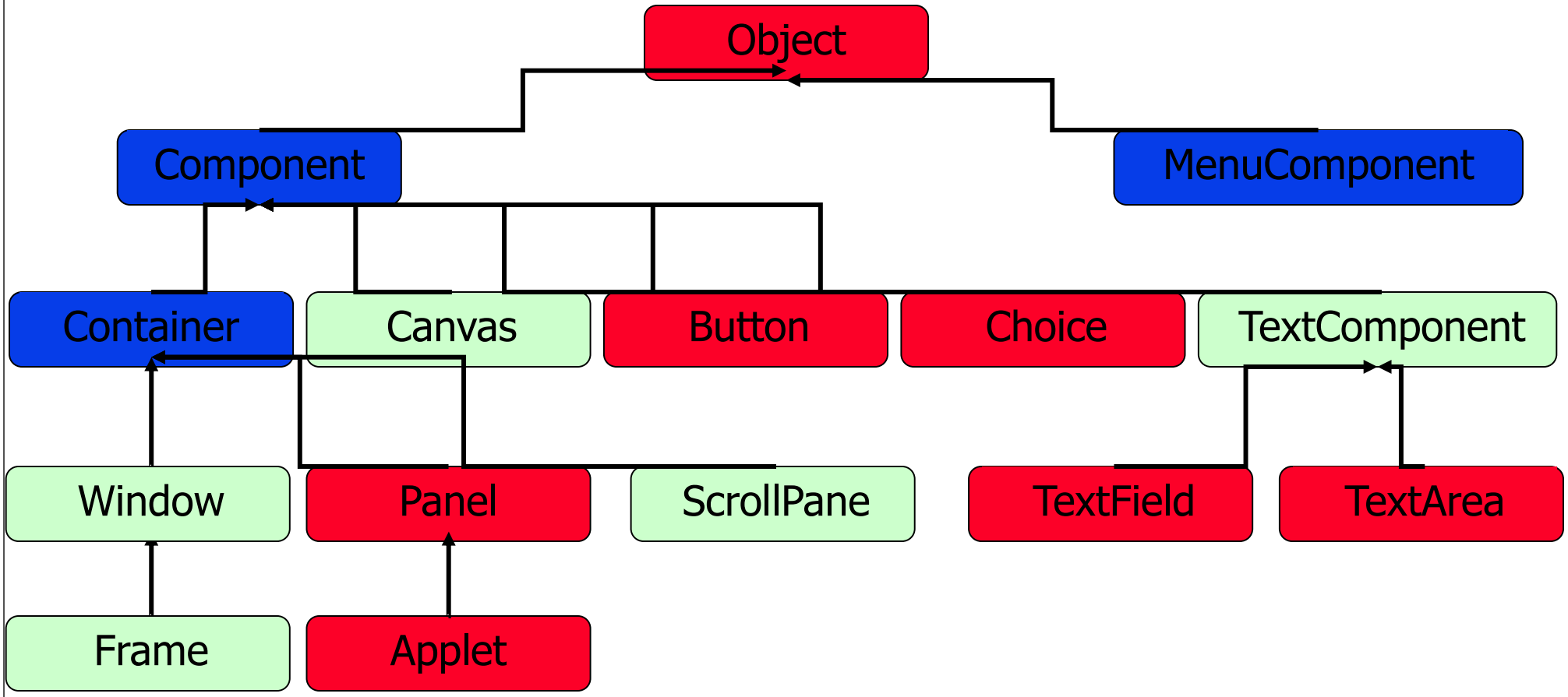
Java Graphical User Interfaces

The Abstract Windowing Toolkit (AWT)

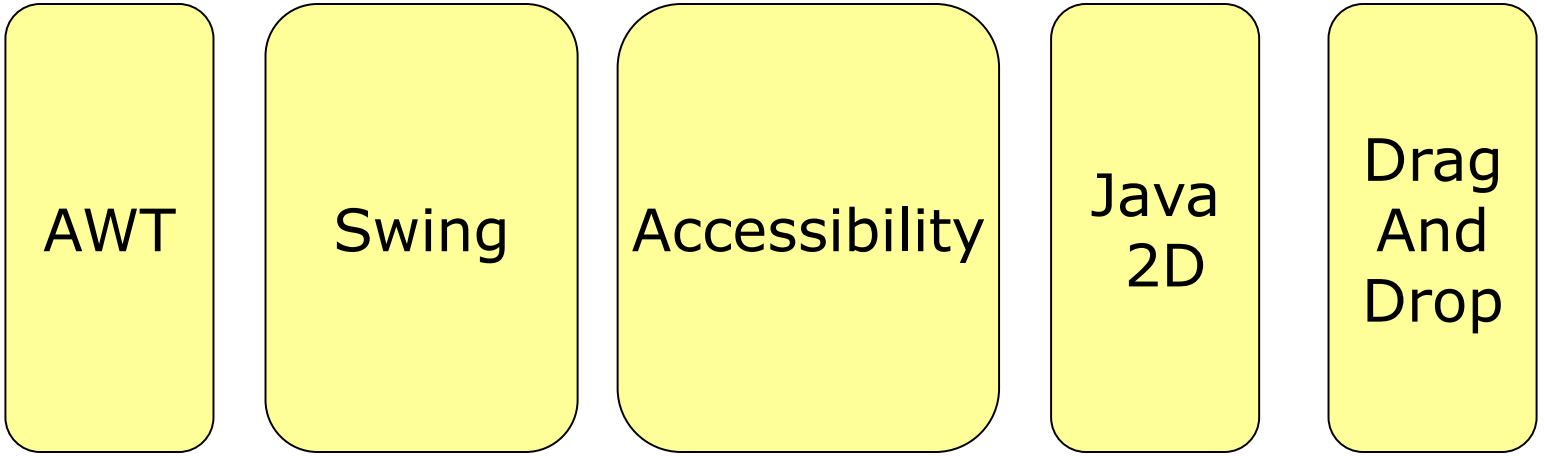
- Since Java was first released, its user interface facilities have been a significant weakness
  - The Abstract Windowing Toolkit (AWT) was part of the JDK from the beginning, but it really was not sufficient to support a complex user interface
- JDK 1.1 fixed a number of problems and, most notably, introduced a new event model
  - It did not make any major additions to the basic components

Part of The AWT Hierarchy



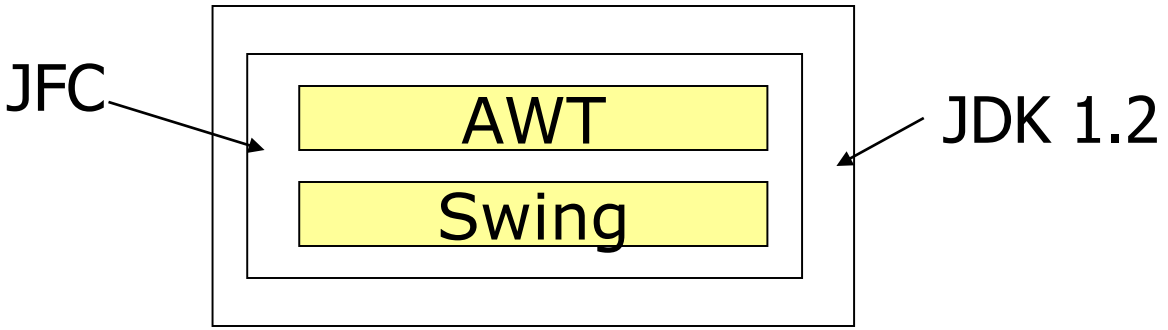
Java Foundation Classes (JFC)

- In April 1997, JavaSoft announced the Java Foundation Classes (JFC)
  - A major part of the JFC was a new set of user interface components called Swing



Swing

- The Swing classes are used to build graphic user interfaces.
  - Swing does not stand for anything
- Swing is built on top of the core 1.1 and 1.2 AWT libraries
- Swing improves AWT on three major issues
  - It (usually) does not rely on the platform's native components.
  - It supports "Pluggable Look-and-Feel" or PLAF
  - It is based on the Model-View-Controller (MVC) design pattern



Why is Swing so Hot?

- It uses lightweight components
- It uses a variant of the Model View Controller Architecture (MVC)
- It has Pluggable Look And Feel (PLAF)
- It uses the Delegation Event Model

Weighting Components

- Sun makes a distinction between *lightweight* and *heavyweight* components
  - *Lightweight* components are not dependent on native peers to render themselves: they are coded in Java
  - *Heavyweight* components are rendered by the host operating system: they are resources managed by the underlying window manager

Heavyweight Components

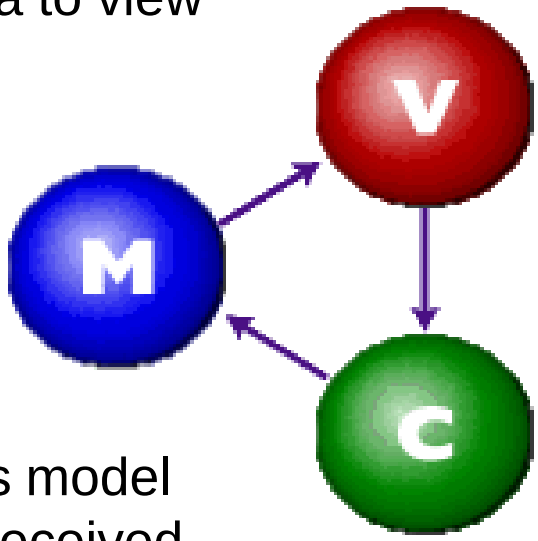
- *Heavyweight* components were unwieldy for two reasons
  - Equivalent components on different platforms do not necessarily act alike
  - The look and feel of each component was tied to the host operating system
- Almost all Swing components are *lightweight* except
  - `JApplet`
  - `JFrame`
  - `JDialog`
  - `JWindow`

Model View Controller

- Independent elements:
  - Model
    - state data for each component
    - different data for different models
  - View
    - how the component looks onscreen
  - Controller
    - dictates how the component reacts to events

MVC Communication

Model passes data to view for rendering

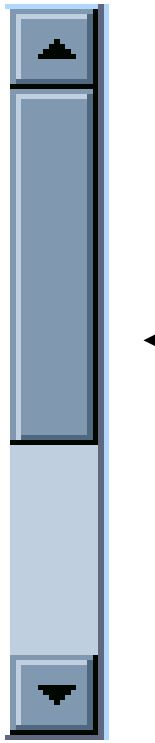


View determines which events are passed to controller

Controller updates model based on events received

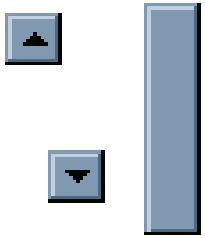
http://java.sun.com/products/jfc/tsc/articles/architecture/images/mvc_lg.gif

MVC Example



Model: Minimum=0
Maximum=100
Value=0

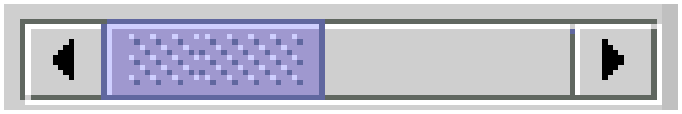
View:



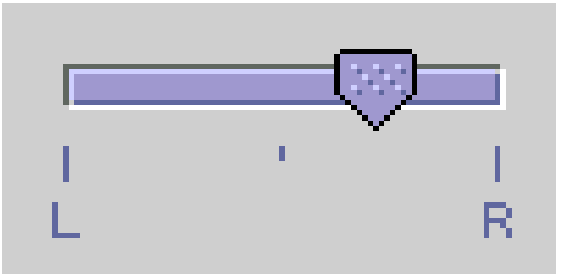
Controller:
-accept mouse click on end buttons
-accept mouse drag on thumb

MVC in Java

- Swing uses the model-delegate design, a similar architecture to MVC
- The View and Controller elements are combined into the UI delegate since Java handles most events in the AWT anyway
- Multiple views can be used with a single model
- Changes to a single Model can affect different views
- Components can share a model (`JScrollbar` and `JSlider` share the `BoundedRangeModel`)



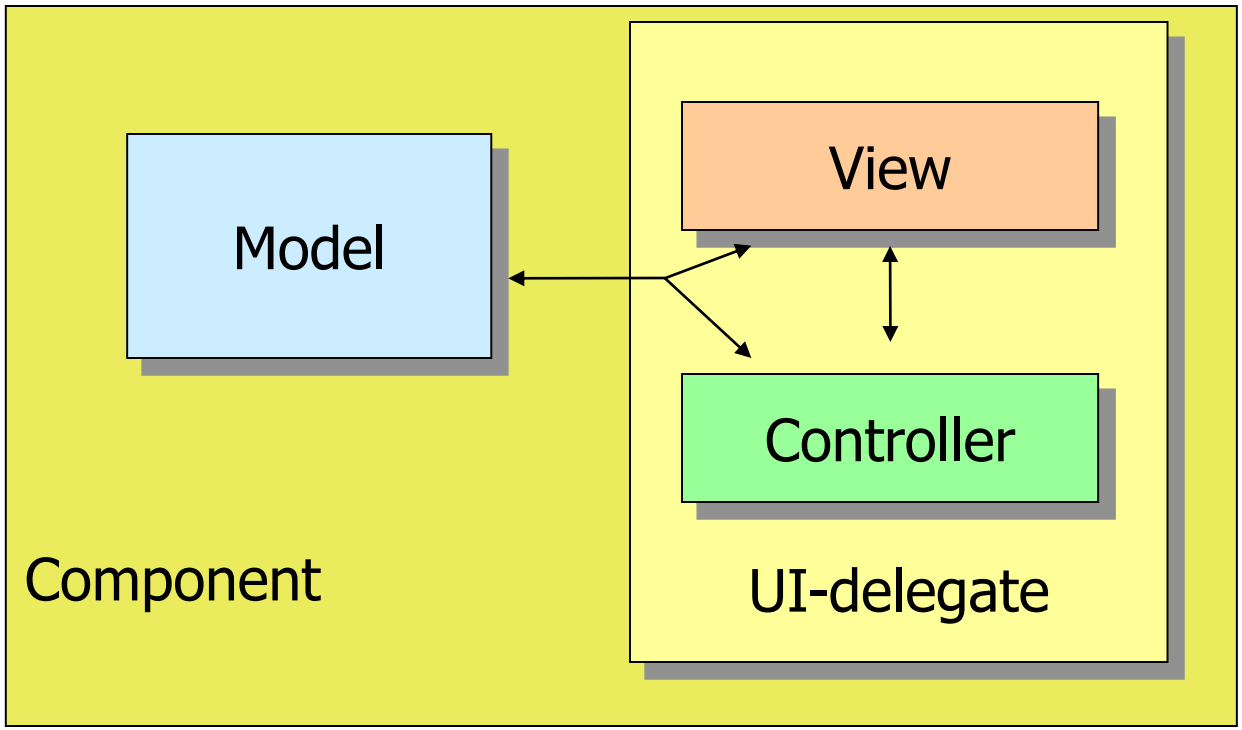
JScrollBar



JSlider

http://java.sun.com/products/jfc/tsc/articles/component_gallery/index.html

MVC in Java



MVC in Java (2)

Most components provide the model-defined API directly in the component class. The component can be manipulated without interacting with the model at all

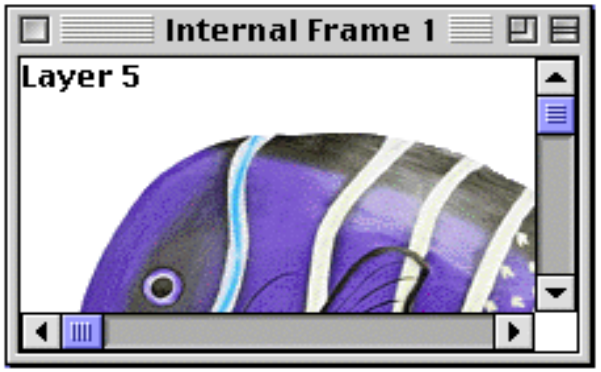
```
//example of method in JSlider class

public int getValue() {
        return getModel().getValue();
}

//so we can use the following and avoid the model

JSlider slider = new JSlider();
int value = slider.getValue();
```

PLAF Features in Swing

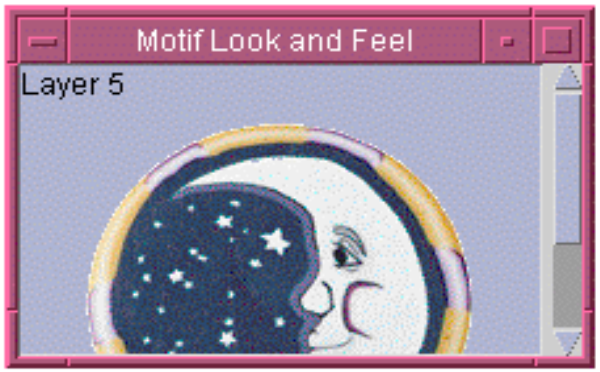
- Default Metal style
- Can emulate Motif, and Windows styles
- Supports Mac style through download
- New styles can be designed
- Can be reset at runtime

PLAF examples

Java Look and Feel



MacOS Look and Feel



Motif Look and Feel

Windows Look and Feel

http://java.sun.com/products/jfc/tsc/articles/jlf-design/index.html

PLAF Structure

- All components have an abstract UI delegate in the swing.plaf package (`Jbutton - ButtonUI`)
- UI delegate is accessed by `get/setUI()` method
- Each Look and Feel has a concrete class for each abstract UI delegate (`WindowsButtonUI`)
- communicate through UIManager class
  - `get/setLookAndFeel()`

GUI Packages

- AWT
  - java.awt
  - java.awt.color
  - java.awt.datatransfer
  - java.awt.event
  - java.awt.font
  - java.awt.geom
  - java.awt.image
  - ...
- Swing
  - javax.accessibility
  - javax.swing
  - javax.swing.colorchooser
  - javax.swing.event
  - javax.swing.filechooser
  - javax.swing.plaf
  - javax.swing.table
  - javax.swing.text.html
  - javax.swing.tree
  - ...

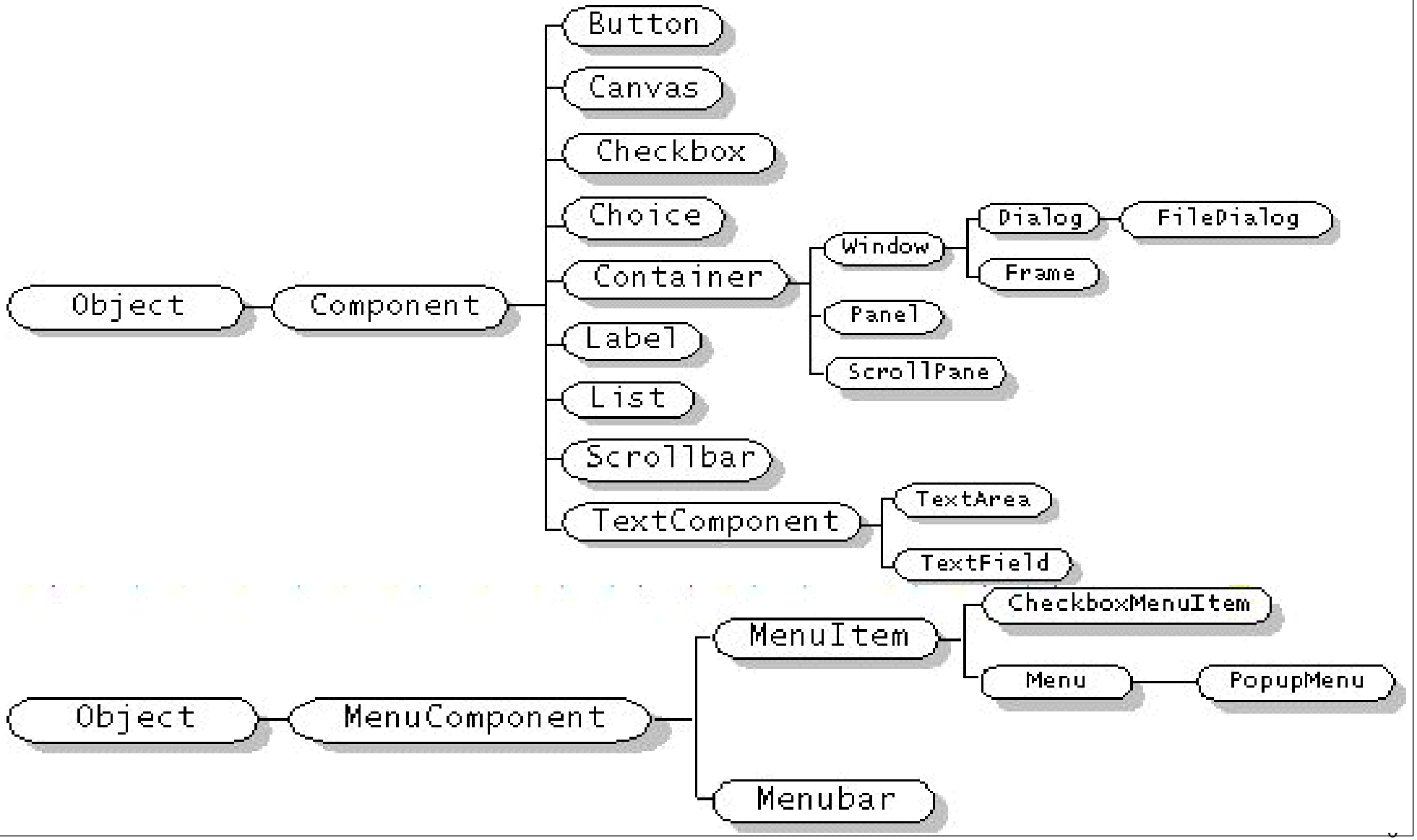
Components

- A graphical user interface consists of different graphic Component objects which are combined into a hierarchy using Container objects

- *Component* class
  - An abstract class for GUI components such as *Menus*, *Buttons*, *Labels*, *Lists*, etc.

- *Container* class
  - An abstract class that extends *Component*. Classes derived from *Container* -most notably *Panel*, *Applet*, *Window*, *Dialog*, *Frame*- can contain multiple components

Inheritance Hierarchies for Components



Additional Swing Features

- Using Swing:
  - A wide variety of components can be created (*tables*, *trees*, *sliders*, *progress bars*, *internal frame*, …)
  - Components can have *tooltips* placed over them
  - Arbitrary keyboard events can be bound to components
  - There is additional debugging support
  - There is support for parsing and displaying HTML-based information

Applets vs. Applications

- Using Swing it is possible to create two different types of GUI programs
  - *Standalone applications*
    - Programs are started from the command line
    - Code resides on the machine on which they are run
  - *Applets*
    - Programs run inside a web browser
    - Code is downloaded from a web server
    - JVM is contained inside the web browser
    - For security purposes applets are normally prevented from doing certain things (for example opening files)
- For now we will write *standalone* applications

Running Applets

```
<APPLET code="HelloWorld.class" width=300 height=200>
  <PARAM name="username" value="Fred">
  <PARAM name="age" value="34">
</APPLET>
```

- We name this file as Filename.html
- We run it buy appletviewer command
- An applet should always contains an "init" method

```
import java.applet.Applet;
import java.awt.*;
public class MyApplet extends Applet{
   public void init (){
      add (new Label ("This is a read-only string"));}}
```

JFrame

- A *JFrame* is a *window* with all of the adornments added.
- A *JFrame* provides the basic building block for screen-oriented applications
- To show or hide a *JFrame*, *setVisible()* is used. Moreover, in previous to 1.5 versions, *show()* can also be used.

```
JFrame win = new JFrame("title");
```

Creating a JFrame

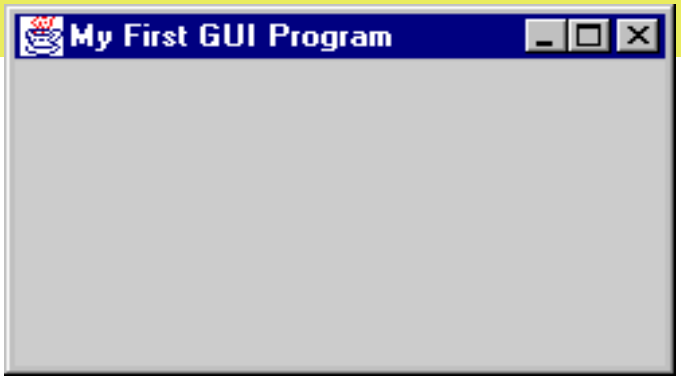
```
import javax.swing.*;

public class SwingFrame
{
   public static void main (String[] args)
   {
      JFrame win = new JFrame("My First GUI
                                  Program");
      win.setVisible(true);
   }
}
```

JFrame

- Sizing a `Frame`
  - You can specify the size
    - *Height* and *width* given in pixels
    - The size of a pixel will vary based on the resolution of the device on which the frame is rendered
  - The method *`pack()`* will set the size of the frame automatically based on the size of the components contained in the *`Content Pane`* (i.e. the Container that corresponds to the Frame)
    - Note that *pack()* does not look at the title bar
  - *setBounds()* or *setSize()* together with *setLocation()* can be used

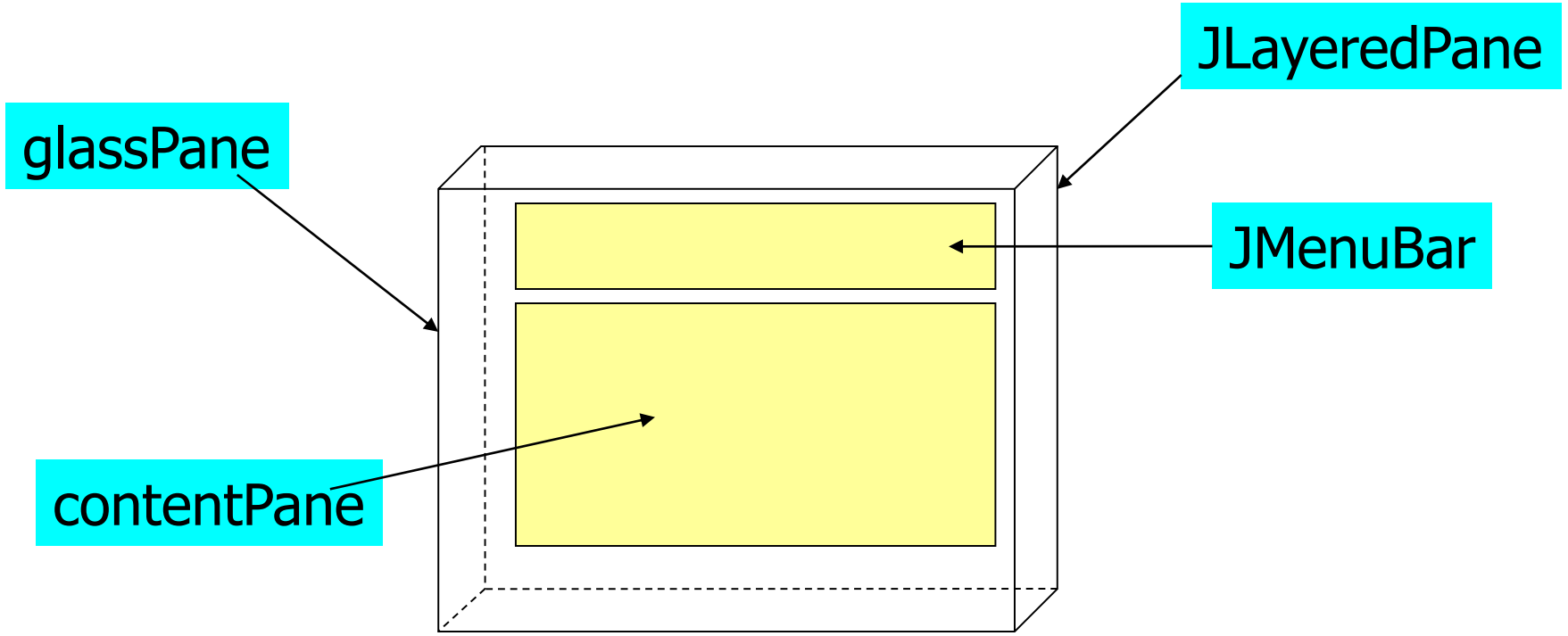
Creating a JFrame

```
import javax.swing.*;

public class SwingFrame
{
    public static void main (String[] args)
    {
        JFrame win = new JFrame("My First GUI Program");
        win.setSize(250, 150);
        win.setVisible(true);
    }
}
```



JFrame

- A *JFrame* contains a *JRootPane* as its only child

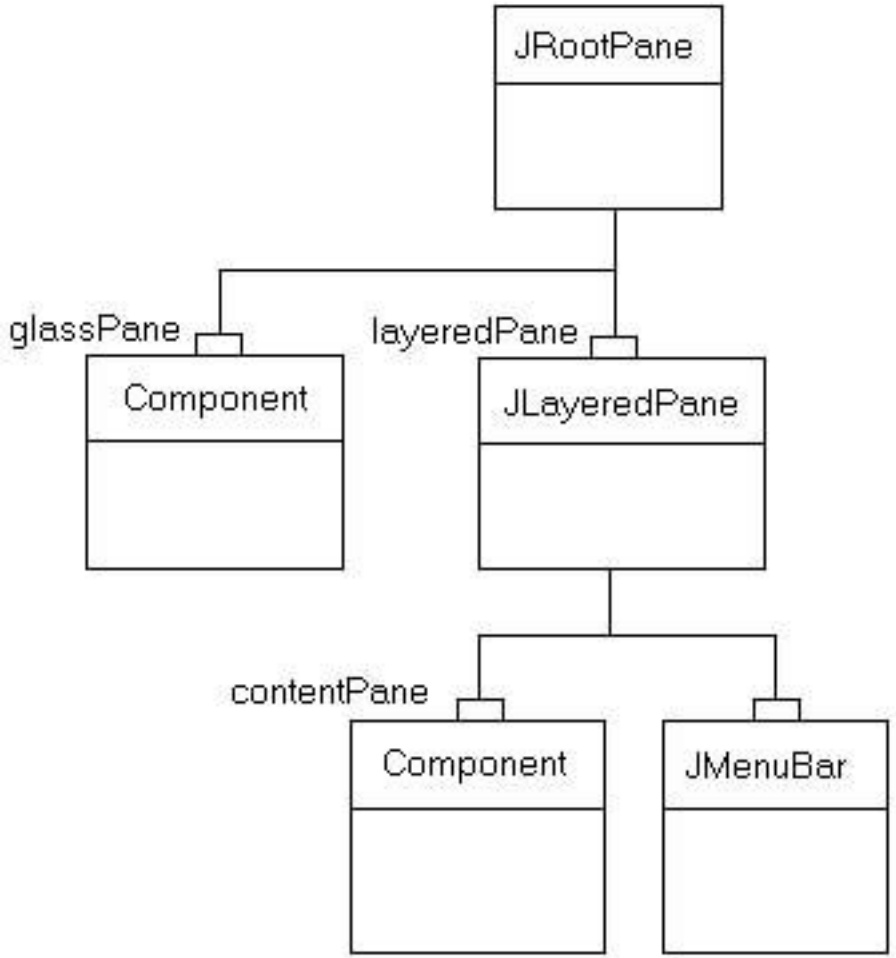


- A *JRootpane* is made up of a *glassPane*, an optional *JMenuBar*, and a *contentPane*

JFrame

- The *contentPane* provided by the root pane should, as a rule, contain all the non-menu components displayed by the *JFrame*
- The *JLayeredPane* manages the *menuBar* and the *contentPane*
- The *glassPane* sits over the top of everything, where it is in a position to intercept mouse movements
- Although the *menuBar* component is optional, the *JLayeredPane*, *contentPane*, and *glassPane* always exist
- To add components to the *JFrame* (other than the optional menu bar), you add the object to the *contentPane* of the *JRootPane*, like this: jframe.getContentPane().add(child);
- The JMenuBar is specified with: jframe.setJMenuBar(menubar);

JFrame

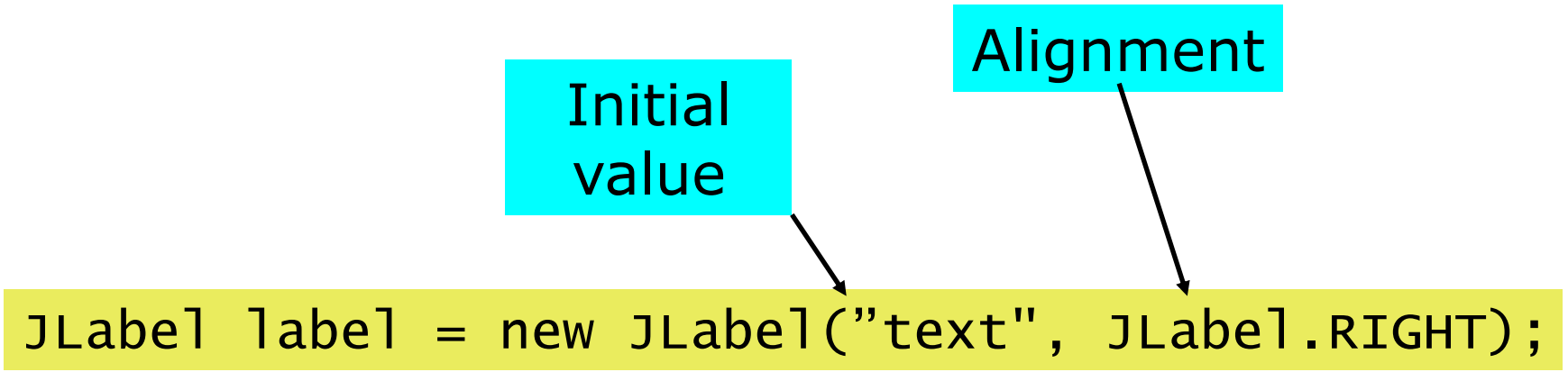

JRootPane
glassPane
layeredPane
Component
JLayeredPane
contentPane
Component
JMenuBar

Swing Components

- JComponent
  - JComboBox
  - JLabel
  - JList
  - JMenuBar
  - JPanel
  - JPopupMenu
  - JScrollBar
  - JScrollPane
  - JTable
  - JTree
  - JInternalFrame
  - JOptionPane
  - JProgressBar
  - JRootPane
  - JSeparator
  - JSlider
  - JSplitPane
  - JTabbedPane
  - JToolBar
  - JToolTip
  - Jviewport
  - JColorChooser
  - JTextComponent
  - …

JLabel

- `JLabel`s are components where you can put text into
- When creating a label you can specify the initial value and the alignment you wish to use within the label
- You can use `getText()` and `setText()` to get and change the label's text



- Like the majority of JComponents, JLabel has more than one constructors

Hello World

```
import javax.swing.*;

public class SwingFrame
{
   public static void main(String[] args)
   {
      JFrame win = new JFrame("My First GUI Program");
      JLabel label = new JLabel("Hello World");
      win.getContentPane().add(label);
      win.pack();
      win.setVisible(true);
   }
}
```

JButton

- `JButton` extends `JComponent`, displays a string and delivers an `ActionEvent` for each mouse click
- Normally buttons are displayed with a border
- In addition to text, `JButtons` can also display icons

Initial value

```
JButton button = new JButton("text");
```

JButton

```
import javax.swing.*;

public class SwingFrame
{
    public static void main(String[] args)
    {
        JFrame win = new JFrame("My First GUI Program");
        JButton button = new JButton("Click Me!!");
        win.getContentPane().add(button);
        win.pack();
        win.setVisible(true);
    }
}
```

Layout Manager

- `LayoutManager`
  - An interface that defines methods for positioning and sizing objects within a container
    - Java defines several default implementations of `LayoutManager`
- Geometrical placement in a *Container* is controlled by a `LayoutManager` object
- Containers may contain components
  - which means containers can contain containers!!
- All containers come equipped with a layout manager which positions and shapes (lays out) the container's components
- Much of the action in the AWT occurs between components, containers, and their layout managers

Layout Managers

- Layouts allow you to format components on the screen in a platform independent way
- The standard JDK provides several classes that implement the `LayoutManager` interface. Some of them are:
  - *`BorderLayout`*
  - *`CardLayout`*
  - *`FlowLayout`*
  - *`GridBagLayout`*
  - *`GridLayout`*
- Layout managers are defined in the AWT package

Changing the Layout

- To change the layout used in a container you first need to create the container
- Then the `setLayout()` method is invoked on the container
- The layout manager should be established before any components are added to the container

```
JPanel p = new JPanel() ;
p.setLayout(new FlowLayout());
```

FlowLayout

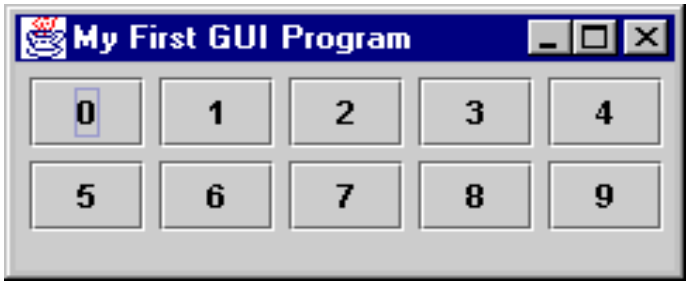
- `FlowLayout` is the default layout for the `JPanel` class
- When you add components to the screen, they flow left to right (centered) based on the order added and the width of the screen
- Very similar to word wrap and full justification on a word processor
- If the screen is resized, the components' flow will change based on the new width and height

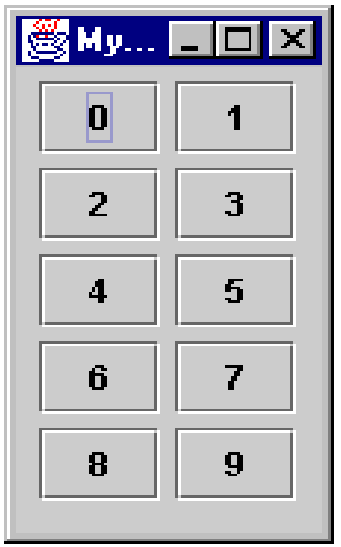
FlowLayout

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

public class SwingFrame
{
   public static void main(String args[])
   {
      JFrame win = new JFrame("My First GUI Program");
      win.getContentPane().setLayout(new FlowLayout());
      for (int i = 0; i < 10; i++)
         win.getContentPane().add(new
                  JButton(String.valueOf(i)));
      win.pack();
      win.setVisible(true);
   }
}
```

FlowLayout


My First GUI Program
0
1
2
3
4
5
6
7
8
9


My First GUI Program
0
1
2
3
4
5
6
7
8
9

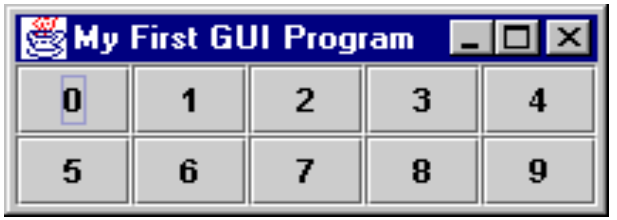

My...
0
1
2
3
4
5
6
7
8
9

GridLayout

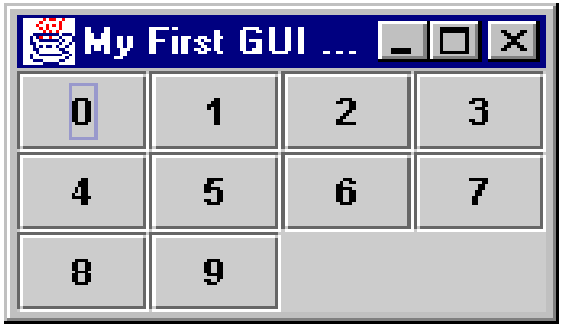
- The `GridLayout` is used for arranging components in rows and columns
  - If the number of rows is specified, the number of columns will be set to the number of components divided by the rows
  - If the number of columns is specified, the number of rows will be set to the number of components divided by the columns
  - Specifying the number of columns affects the layout only when the number of rows is set to zero
- The order in which you add components is relevant
  - The first component is placed at (0,0), the second at (0,1), ….
- The underlying components are resized to fill the row-column area if possible

GridLayout

gridLayout(4, 4)



gridLayout(2, 4)



gridLayout(0, 4)

gridLayout(10, 10)

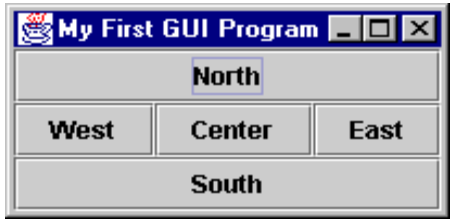
BorderLayout

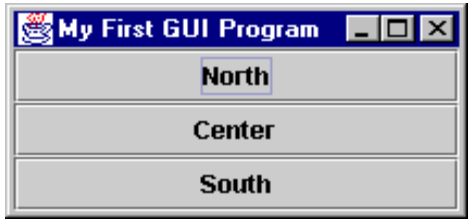
- `BorderLayout` provides 5 areas to hold components
  - These are named after the four different borders of the screen, North, South, East, West and Center
- When a Component is added to the layout, you must specify which area to place it in
  - The order in which components are added is not important
- The center area will always be resized to be as large as possible

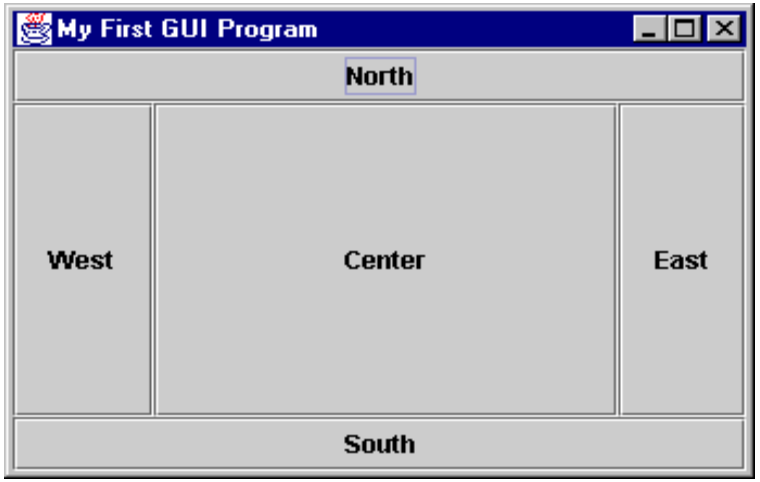
BorderLayout

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

public class SwingFrame
{
   public static void main(String[] args)
   {
      JFrame win = new JFrame("My First GUI Program");
      Container content = win.getContentPane();
      content.setLayout(new BorderLayout());
      content.add("North", new JButton("North"));
      content.add("South", new JButton("South"));
      content.add("East", new JButton("East"));
      content.add("West", new JButton("West") );
      content.add("Center", new JButton("Center"));
      win.pack();
      win.setVisible(true);
   }
}
```

BorderLayout







GridBagLayout

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

public class GridBagEx1 extends JApplet
{
   protected void makebutton(String name,
  GridBagLayout gridbag, GridBagConstraints c)
   {
      JButton button = new JButton(name);
      gridbag.setConstraints(button, c);
      this.getContentPane().add(button);
   }
```

GridBagLayout

```
public void init()
{
    GridBagLayout gridbag = new GridBagLayout();
    GridBagConstraints c = new GridBagConstraints();

    this.getContentPane().setLayout(gridbag);
    c.fill = GridBagConstraints.BOTH;
    c.weightx = 1.0;
    this.makebutton("Button1", gridbag, c);
    this.makebutton("Button2", gridbag, c);
    this.makebutton("Button3", gridbag, c);
    //end row
    c.gridwidth = GridBagConstraints.REMAINDER;
    this.makebutton("Button4", gridbag, c);
```

GridBagLayout

```
c.weightx = 0.0;                //reset to the default
this.makebutton("Button5",gridbag,c);
                                //another row
c.gridwidth = GridBagConstraints.RELATIVE;
                                //next-to-last in row
this.makebutton("Button6", gridbag, c);

c.gridwidth=GridBagConstraints.REMAINDER;
                                //end row
this.makebutton("Button7", gridbag, c);

c.gridwidth = 1;                //reset to the default
c.gridheight = 2;
c.weighty = 1.0;
```

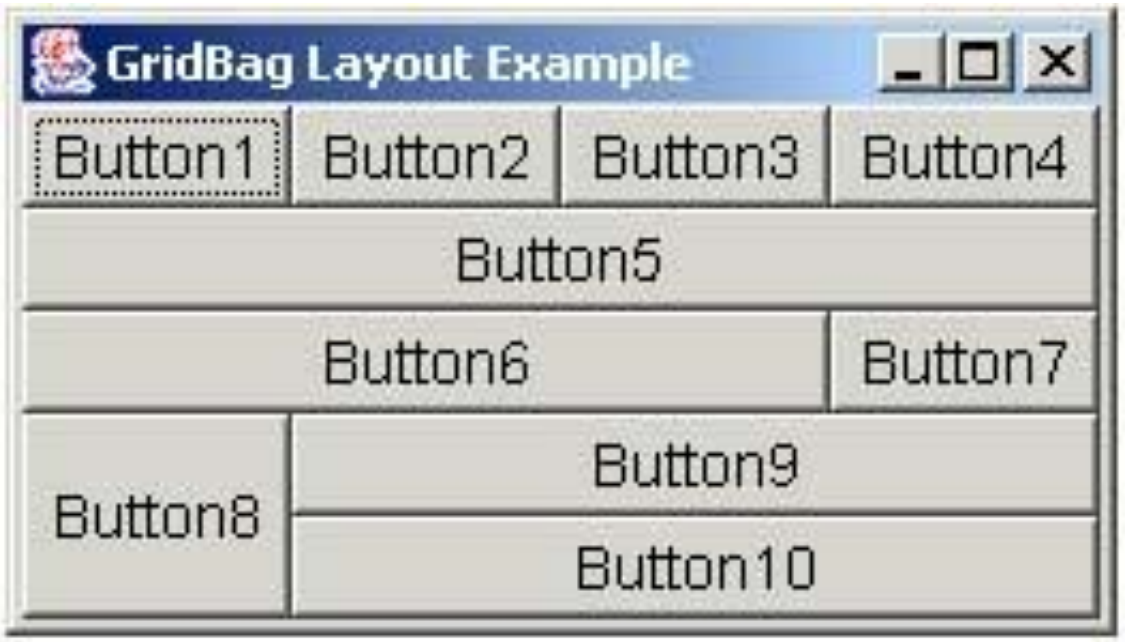
GridBagLayout

```
        this.makebutton("Button8", gridbag, c);

        c.weighty = 0.0;      //reset to the default
        c.gridwidth = GridBagConstraints.REMAINDER;
                                        //end row
        c.gridheight = 1;     //reset to the default
        this.makebutton("Button9", gridbag, c);
        this.makebutton("Button10", gridbag, c);

        this.setSize(300, 100);
    }
```

GridBagLayout

```
    public static void main(String args[])
    {
        JFrame f = new JFrame("GridBag Layout Example");
        GridBagEx1 ex1 = new GridBagEx1();

        ex1.init();
        f.getContentPane().add("Center", ex1);
        f.pack();
        f.setSize(f.getPreferredSize());
        f.setVisible(true);
    }
}
```

GridBagLayout

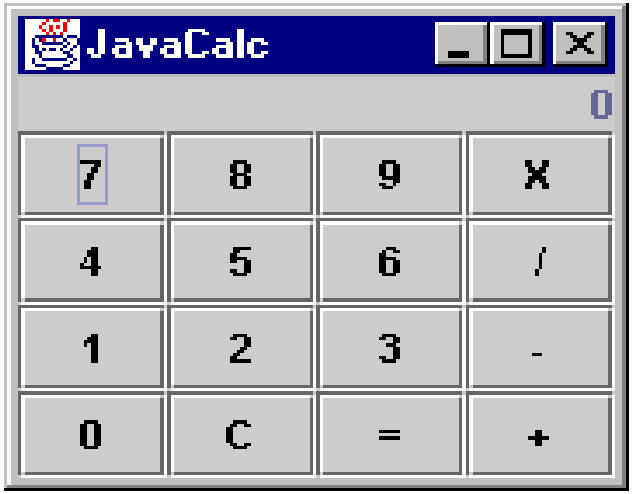


Containers

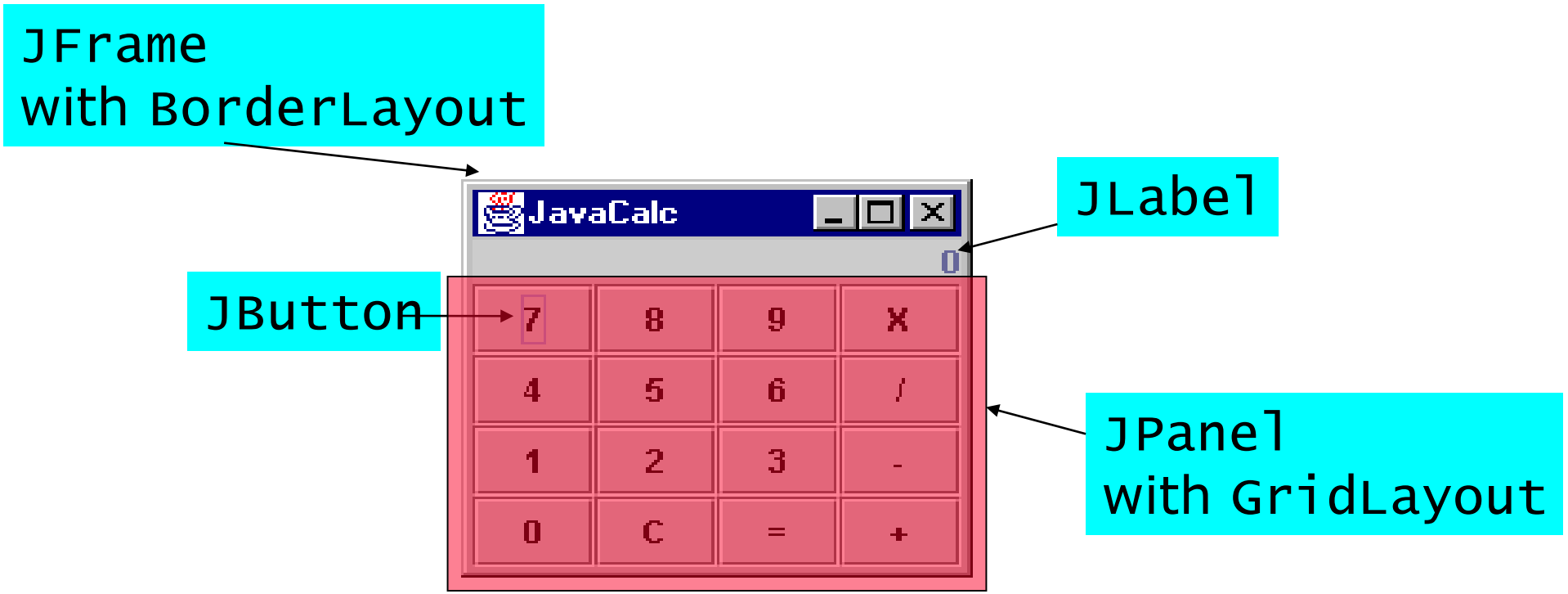
- A *JFrame* is not the only type of container that you can use in Swing

- Some subclasses of *Container* are:
  - *Panel (→ JPanel)*
    - *Applet (→ JApplet)*
  - *Window (→ JWindow)*
    - *Dialog (→ JDialog)*
    - *Frame (→ JFrame)*

A Simple 4 Function Calculator



Swing Components



CalcGui.java

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

public class CalcGui implements
{
   // Labels for the buttons
   private static final String labels =
                              "789X456/123-0C=+";

   private static final int NUMROWS = 4;
   private static final int NUMCOLS = 4;

   private JLabel display;       // The display
```

CalcGui.java

```
    public CalcGui(String name)
    {  // A Frame for the calculator
        JFrame win = new JFrame(name);

// Create the button panel
        JPanel buttons = new JPanel();
        buttons.setLayout(new GridLayout(NUMROWS,
                                         NUMCOLS));

        JButton b;
        for (int i = 0; i < labels.length(); i++)
        {
            b = new JButton(labels.substring(i, i + 1));
            buttons.add( b );
        }
```

CalcGui.java

```
// Create the display
      display = new JLabel("0", JLabel.RIGHT)

// "Assemble" the calculator
      Container content = win.getContentPane();
      content.setLayout(new BorderLayout());

      content.add("North", display);
      content.add("Center", buttons);

// Display it and let the user run with it :-)
      win.pack();
      win.setVisible(true);
   }
}
```

Java Gui Builders

- Java Frame Builder
- JVider (with plugin for eclipse)
- Matisse (for NetBeans Developers)
- Jigloo (with plugin for eclipse)

Java Gui Code Samples & Tutorials

- http://examples.oreilly.com/jswing2/code/
- http://www.codeguru.com/java/Swing/index.shtml
- http://javafaq.nu/java/free-swing-book/index.shtml
- http://www.tutorialized.com/tutorials/Java/Swing/1