8. GUI

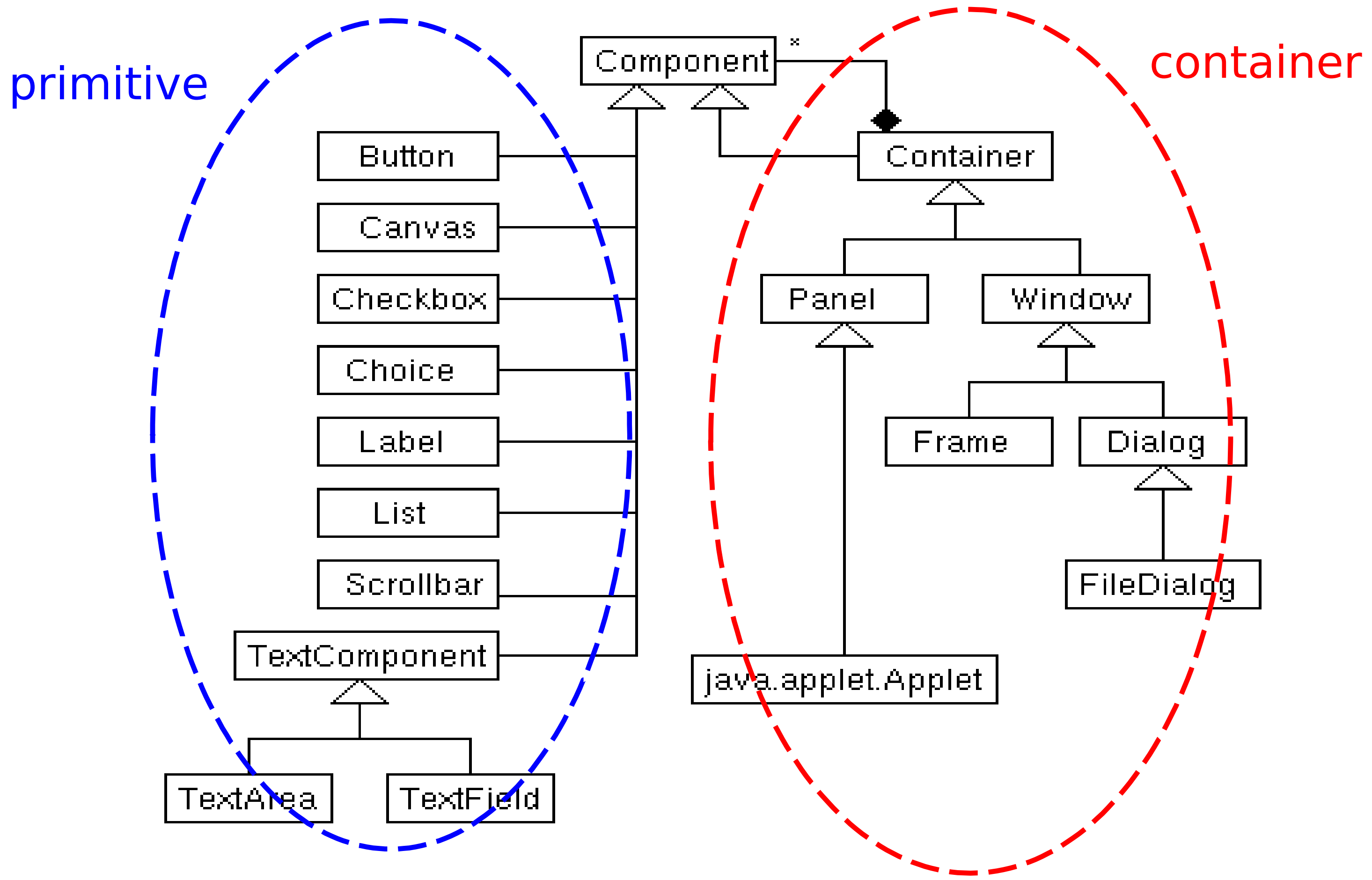
Introduction

- Graphical User Interface, abbreviated GUI, is a type of interface that takes advantage of a computer’s graphic ability to allow users to interact with electronic devices through graphical icons and visual indicators.
- There are two packages that generate GUI components in Java.
  - *java.awt*
  - *javax.swing*

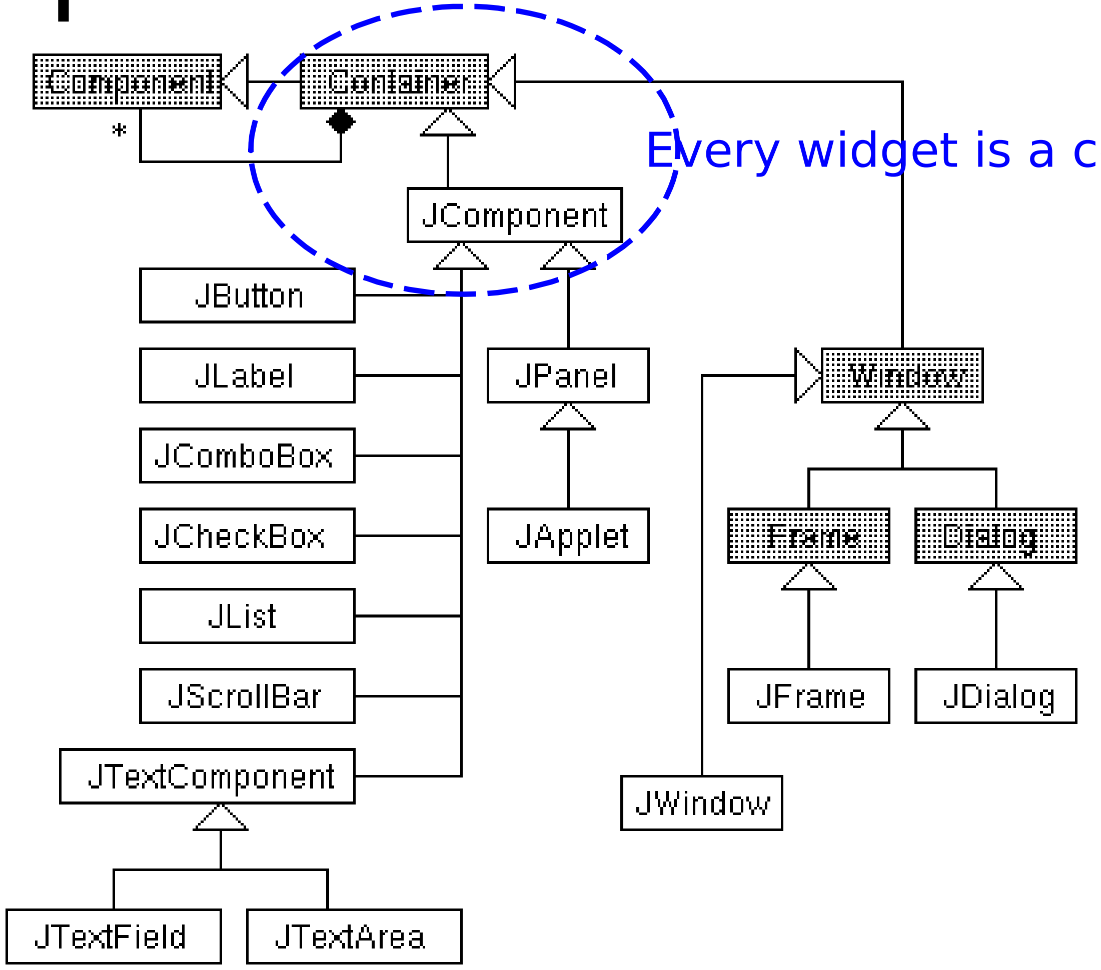
AWT vs. Swing

- AWT
  - *Heavyweight* components
  - Associated with native components called *peers*
  - Same behaviour, but platform-dependent look
  - Package java.awt
- Swing
  - *Lightweight* components, i.e., no peer components
  - *Same look and feel* across platforms
  - Support *pluggable* look and feel
  - Package javax.swing

AWT Components



Swing Components



Every widget is a container!

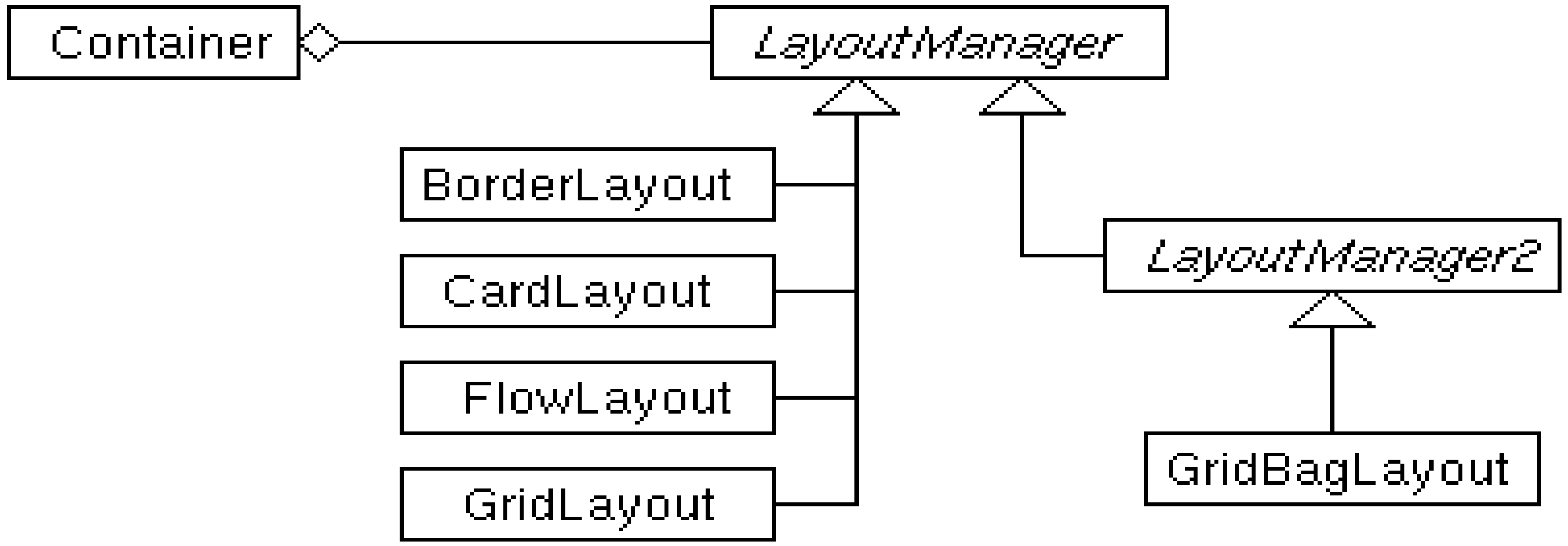
Containers

- In Java, all GUI objects go into a Container.
- A top level container can stand alone in a web browser or in an operating system. The two most common top-level containers are:
  - JFrame
  - JApplet
- Some containers may only be added to other containers.
  - JPanel

Layout Managers

- Associated with containers
- Automate the layout of elements
  - When elements are added to the container
  - When the window is resized
    - automatically adjust the positions and sizes of the elements.

Hierarchy of Layout Managers



Q: Can you identify the design pattern used here?

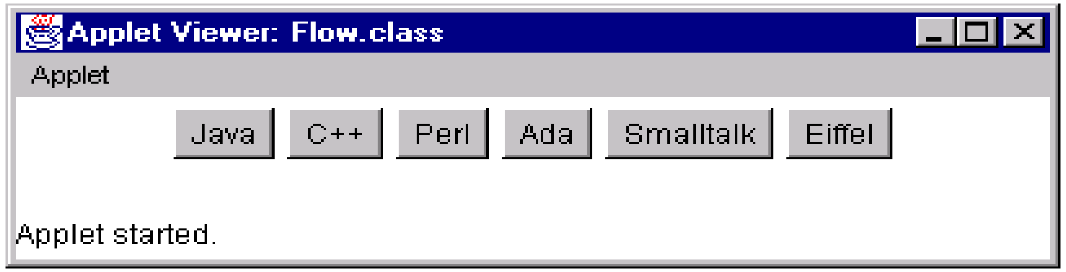
Using Layout Managers

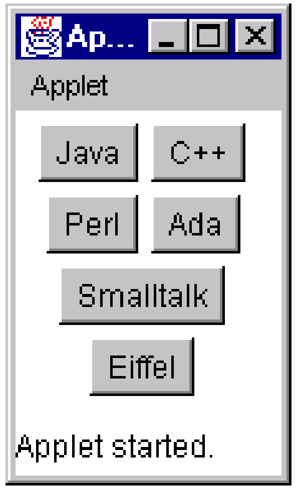
Method	Description
setLayout(lm)	Set lm as the layout manager
add(comp)	Add a component
add(comp, cst)	Add a component with constraint

```
public class CounterApplet extends Applet {
  public CounterApplet () {
    setLayout(new FlowLayout());
    add(new JButton(“Increment"));
    add(new JButton(“Decrement”));
  }
}
```

Flow Layout

width=400 height=50



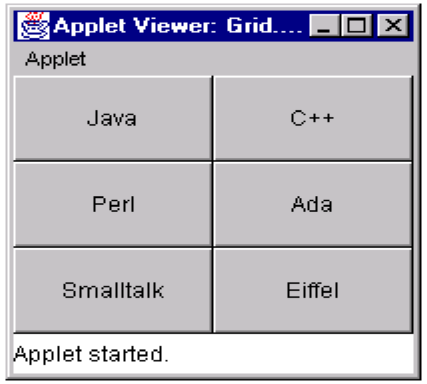


width=100 height=120

Flow Layout (Cont.)

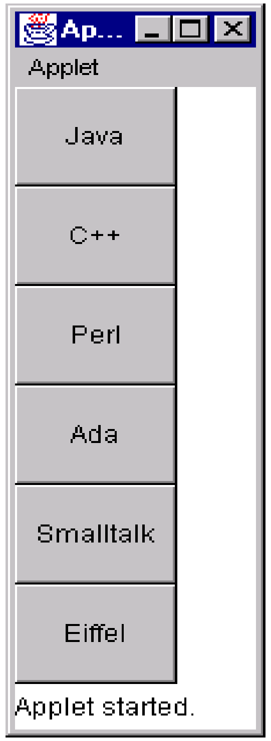
```
public class Flow extends Applet {
  public Flow () {
    setLayout(new FlowLayout());
    add(new JButton("Java"));
    add(new JButton("C++"));
    add(new JButton("Perl"));
    add(new JButton("Ada"));
    add(new JButton("Smalltalk"));
    add(new JButton("Eiffel"));
  }
}
```

Grid Layout

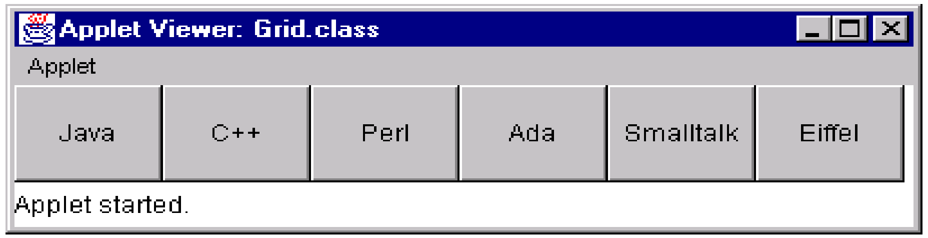


3x2 grid

0x1 grid



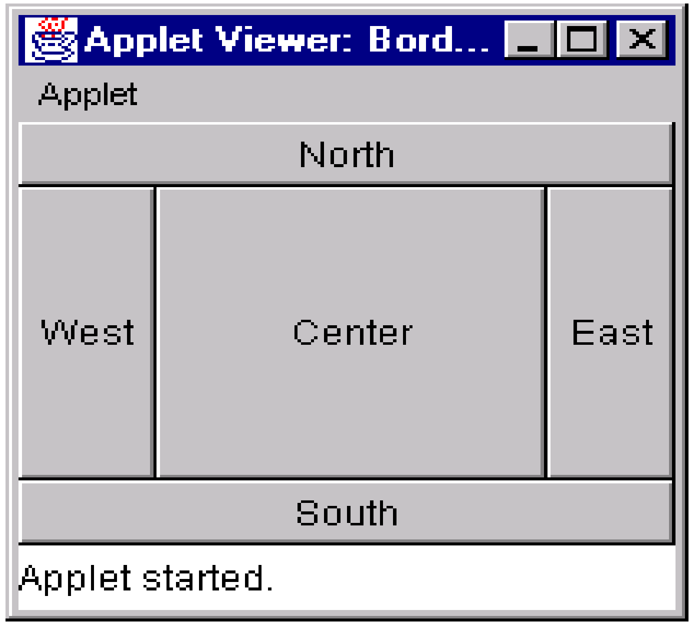
1x0 grid



Grid Layout (Cont.)

```
public class Grid extends Applet {
  public void init () {
    int row = 0;
    int col = 0;
    String att = getParameter("row");
    if (att != null) {
       row = Integer.parseInt(att);
    }
    att = getParameter("col");
    if (att != null) {
       col = Integer.parseInt(att);
    }
    if (row == 0 && col == 0) {
        row = 3; col = 2;
    }
    setLayout(new GridLayout(row, col));
    add(new JButton("Java"));
    add(new JButton("C++"));
    add(new JButton("Perl"));
    add(new JButton("Ada"));
    add(new JButton("Smalltalk"));
    add(new JButton("Eiffel"));
  }
}
```

Border Layout



Applet started.

Border Layout (Cont.)

```
public class Border extends Applet {
  public Border () {
    setLayout(new BorderLayout());
    add(new JButton("North"), BorderLayout.NORTH);
    add(new JButton("South"), BorderLayout.SOUTH);
    add(new JButton("East"), BorderLayout.EAST);
    add(new JButton("West"), BorderLayout.WEST);
    add(new JButton("Center"), BorderLayout.CENTER);
  }
}
```

Calculator Example

- What do we need for a Calculator GUI?
  - 16 JButtons
    - Numbers 0-9
    - Operators + - x / = .
  - 3 JTextFields
    - 2 operands
    - 1output

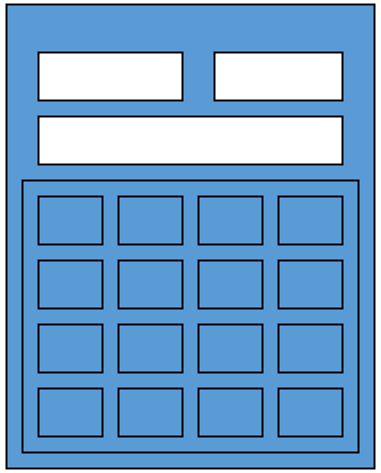
- Which need to respond to events?

Declare Object Data

```
import java.awt.*;
import javax.swing.*;
public class Calculator extends JFrame
{
     JButton [] numbers = new JButton[10];
     JButton plus;
     JButton minus;
     JButton mult;
     JButton div;
     JButton equals;
     JButton dot;
     JTextField output;
     JTextField operand1;
     JTextField operand2;
}
```

Constructor

- Constructor is where everything will be created.
- Before beginning decide
  - how to break up your frame into panels,
  - which LayoutManager goes where,
  - what components will go where.



Instantiate Object Data

```
public Calculator()
{
super("My Calculator");
numbers = new JButton[10];
for(int i = 0; i < 10; i++)
numbers[i] = new JButton("" + i);
plus = new JButton("+");
minus = new JButton("+");
mult = new JButton("x");
div = new JButton("/");
equals = new JButton("=");
dot = new JButton(".");
operand1 = new JTextField(10);
operand2 = new JTextField(10);
output = new JTextField(21);

setSize(300,400);
setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
}
```

Setting properties for the frame, too

Top Panel

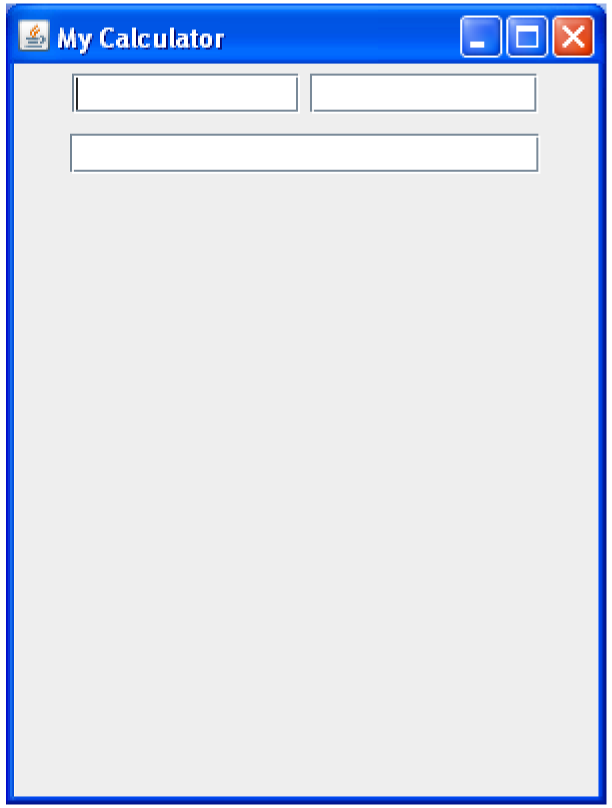
- Need to split the top panel into a grid with two panels. Why?

```
JPanel top = new JPanel();
top.setLayout(new GridLayout(2,1));
add(top, BorderLayout.NORTH);

JPanel input = new JPanel();
input.add(operand1);
input.add(operand2);
top.add(input);

JPanel results = new JPanel();
results.add(output);
top.add(results);
```

Rendering of Previous Code



The Center Panel

- The center will also consist of a grid with four rows and four columns.
- What happens if we add buttons directly to grid?
- What can we do to get our desired effect?
- What do we want the calculator to do when we resize?

Panels of Panels

- **Often GUI programmers create methods to create Panels.**

```
private JPanel
getRow(JButton b1, JButton b2, JButton b3, JButton b4)
{
  JPanel row = new JPanel();
  row.setLayout(new   BoxLayout(row,BoxLayout.X_AXIS));
  row.add(b1);
  row.add(b2);
  row.add(b3);
  row.add(b4);
  return row;
}
```

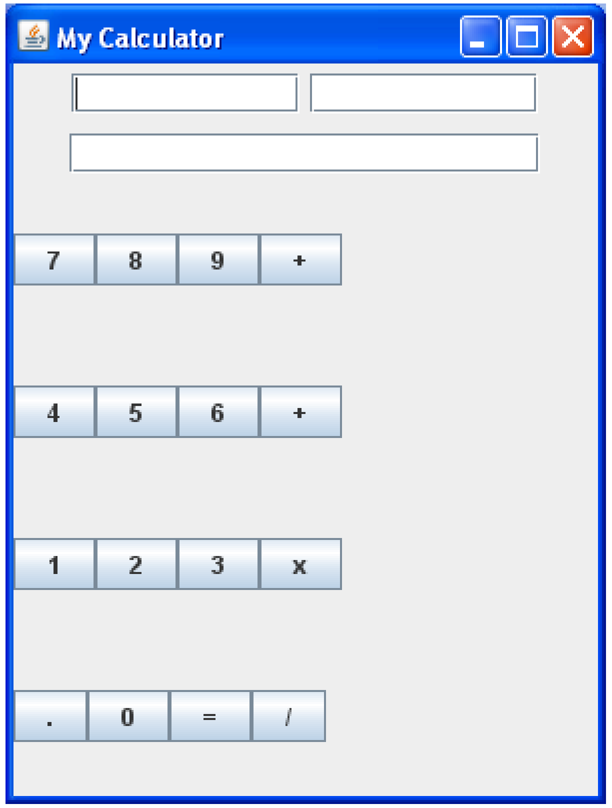
Panels of Panels (cont.)

- Several calls to the method are made from the constructor.

```
JPanel center = new JPanel();
center.setLayout(new GridLayout(4,1));
center.add(getRow(numbers[7], numbers[8], numbers[9], plus));
center.add(getRow(numbers[4], numbers[5], numbers[6], minus));
center.add(getRow(numbers[1], numbers[2], numbers[3], mult));
center.add(getRow(dot, numbers[0], equals, div));
add(center);
```

Calculator

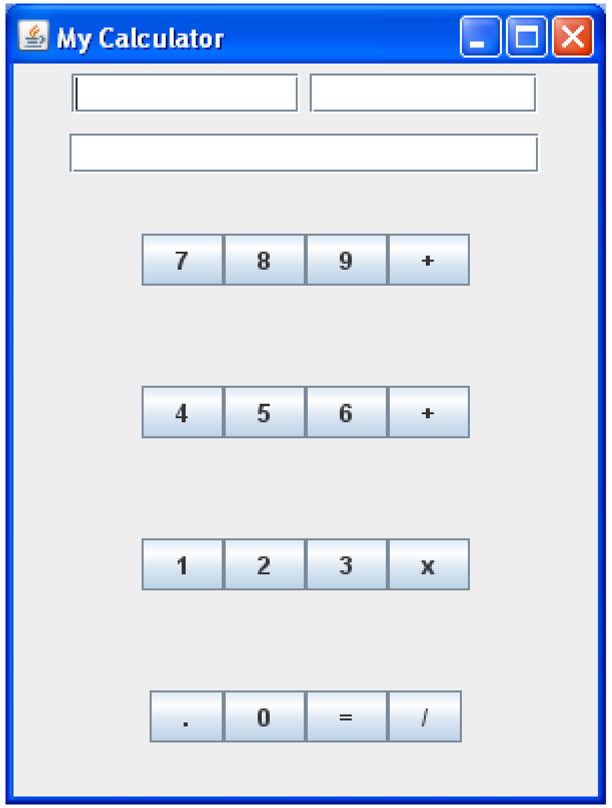
- Adding the previous code, the calculator now renders like so.



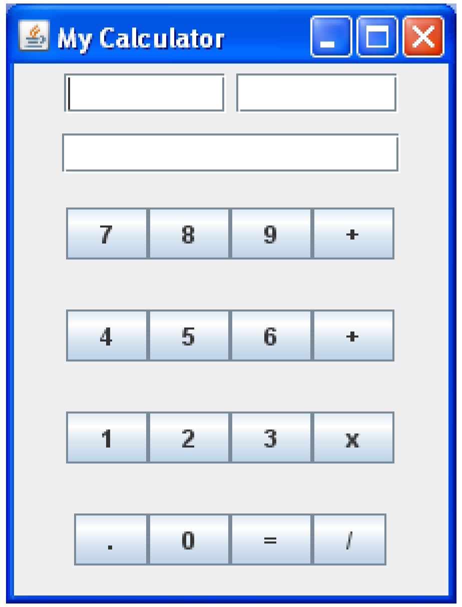
- Adjust the Panel method to incorporate some glue.

```
private JPanel
getRow(JButton b1, JButton b2, JButton b3, JButton b4)
{
  JPanel row = new JPanel();
  row.setLayout(new
  BoxLayout(row,BoxLayout.X_AXIS));
  row.add(Box.createHorizontalGlue());
  row.add(b1);row.add(b2);row.add(b3);row.add(b4);
  row.add(Box.createHorizontalGlue());
  return row;
}
```

- Now it looks like so.



```
setSize(225,300)
operand1 = new JTextField(7);
operand2 = new JTextField(7);
output = new JTextField(15);
```



GUI Event handling

Introduction

- Mechanism to write control code
- Composed of:
  - Event
  - Event source
  - Event listener (or handler)

- Event
  - A way for GUI components to communicate with the rest of application
  - Implemented as event classes (e.g., ActionEvent)
- Event source
  - Components generating events
  - Examples: buttons, check boxes, combo boxes, etc.

- Event listener (or handler)
  - Objects that receives and processes events
  - Must implement an appropriate *listener* interface
  - Must inform the source its interest in handling a certain type of events (by registering)
  - May listen to several sources and different types of events

Example

```
// create a button
JButton button = new JButton(“Increment”);

// register an action listener
button.addActionListener(new ButtonActionListener());

// Action listener class
class ButtonActionListener implements ActionListener {
   public void actionPerformed(ActionEvent e) {
        // handle the event e …
        System.out.println(“Increment button pressed!”);
   }
}
```

Events and Listeners

Event	Listener	Adapter
ActionEvent	*ActionListener*	
ComponentEvent	*ComponentListener*	ComponentAdapter
FocusEvent	*FocusListener*	FocusAdapter
KeyEvent	*KeyListener*	KeyAdapter
MouseEvent	*MouseListener*	MouseAdapter
	MouseMotionListener	MouseMotionAdapter
WindowEvent	*WindowListener*	WindowAdapter
ItemEvent	*ItemListener*	
TextEvent	*TextListener*	

...

Simple GUI Example

```
import javax.swing.JFrame;
class SimpleGUI extends JFrame{
    SimpleGUI(){
            setSize(400,400); //set frames size in pixels
            setDefaultCloseOperation(EXIT_ON_CLOSE);
        show();
         }

         public static void main(String[] args){
            SimpleGUI gui = new SimpleGUI();
            System.out.println(“main thread coninues”);
         }
}
```

Another Simple GUI

```
import javax.swing.*;
class SimpleGUI extends JFrame{
    SimpleGUI(){
         setSize(400,400); //set frames size in pixels
         setDefaultCloseOperation(EXIT_ON_CLOSE);
         JButton but1 = new JButton(“Click me”);
      Container cp = getContentPane();//must do this
         cp.add(but1);
      show();
      }

      public static void main(String[] args){
         SimpleGUI gui = new SimpleGUI();
         System.out.println(“main thread coninues”);
      }}
```

Add Layout Manager

```
import javax.swing.*; import java.awt.*;
class SimpleGUI extends JFrame{
     SimpleGUI(){
            setSize(400,400); //set frames size in pixels
            setDefaultCloseOperation(EXIT_ON_CLOSE);
            JButton but1 = new JButton(“Click me”);
        Container cp = getContentPane();//must do this
            cp.setLayout(new FlowLayout(FlowLayout.CENTER);
         cp.add(but1);
         show();
         }

         public static void main(String[] args){
            SimpleGUI gui = new SimpleGUI();
            System.out.println(“main thread coninues”);
         }}
```

Add call to event handler

```
import javax.swing.*; import java.awt.*;
class SimpleGUI extends JFrame{
        SimpleGUI(){
                setSize(400,400); //set frames size in pixels
                setDefaultCloseOperation(EXIT_ON_CLOSE);
                JButton but1 = new JButton(“Click me”);
            Container cp = getContentPane();//must do this
                cp.setLayout(new FlowLayout(FlowLayout.CENTER);
             but1.addActionListener(new MyActionListener());
             cp.add(but1);
             show();
             }
        public static void main(String[] args){
                SimpleGUI gui = new SimpleGUI();
                System.out.println(“main thread coninues”);
             }}
```

Event Handler Code

```
class MyActionListener implements ActionListener{
   public void actionPerformed(ActionEvent ae){
        JOptionPane.showMessageDialog(“I got clicked”, null);
   }



}
```

Add second button/event

```
class SimpleGUI extends JFrame{
      SimpleGUI(){
              /* .... */
          JButton but1 = new JButton(“Click me”);
              JButton but2 = new JButton(“exit”);
              MyActionListener al = new MyActionListener();
          but1.addActionListener(al);
          but2.addActionListener(al);
          cp.add(but1);
               cp.add(but2);
          show();
          }
}
```

How to distinguish events

```
class MyActionListener implents ActionListener{
    public void actionPerformed(ActionEvent ae){
        if (ae.getSource()  ==  but2){
       System.exit(1);
         }
         else if (ae.getSource() == but1){
              JOptionPane.showMessageDialog(null, “I’m clicked”);
         }
}
```

Question: How are but1, but2 brought into scope to do this?
Question: Why is this better?