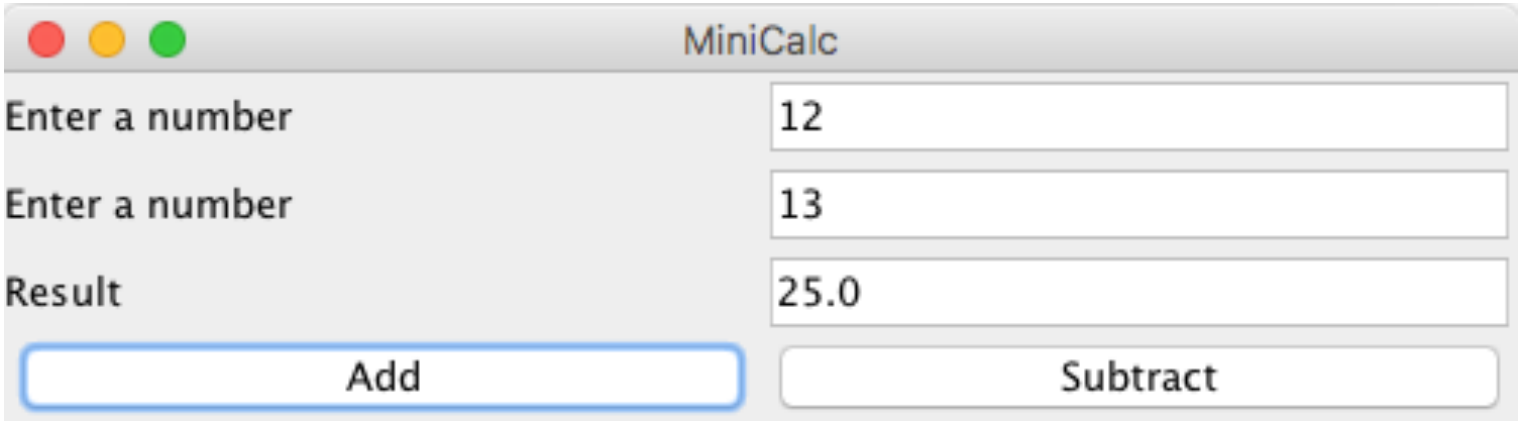
Ch. 16 pt 2

GUIs

Using Several Components

- How do we arrange the GUI components?
  - Using layout managers.
- How do we respond to event from several sources?
  - Create separate listeners, or
  - determine the source of the event with a single listener.

A mini calculator



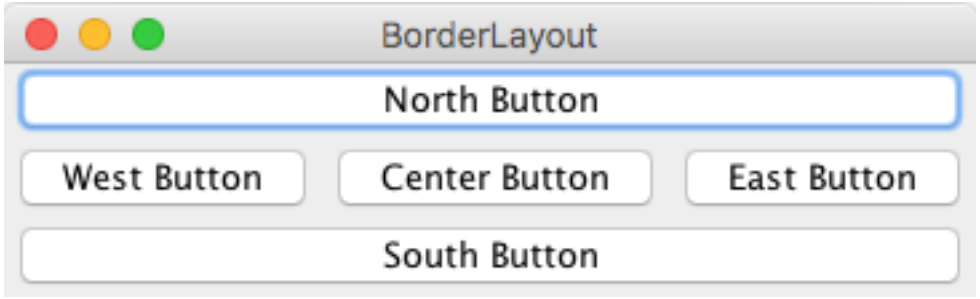
```
//MiniCalc.java - demo GridLayout
import java.awt.*;
import javax.swing.*;

class MiniCalc {
   public static void main(String[] args) {
      JFrame frame = new JFrame("MiniCalc");
      Container pane = frame.getContentPane();
      // create the major components
      JTextField firstNumber = new JTextField(20);
      JTextField secondNumber = new JTextField(20);
      JTextField result = new JTextField(20);
      JButton addButton = new JButton("Add");
      JButton subButton = new JButton("Subtract");
```

```
        // there will be 4 rows of 2 components each
        pane.setLayout(new GridLayout(4, 2));
        // add all of the components to the content pane
        pane.add(new JLabel("Enter a number"));
        pane.add(firstNumber);
        pane.add(new JLabel("Enter a number"));
        pane.add(secondNumber);
        pane.add(new JLabel("Result"));
        pane.add(result);
        pane.add(addButton);
        pane.add(subButton);
        // setup the listener, listening to the buttons
        DoMath listener =
                        new DoMath(firstNumber, secondNumber, result);
        subButton.addActionListener(listener);
        addButton.addActionListener(listener);
        frame.pack();
        frame.show();
    }
}
```

```
//DoMath.java - respond to two different buttons
import javax.swing.*;
import java.awt.event.*;
class DoMath implements ActionListener {
   private JTextField inputOne, inputTwo, output;
   DoMath(JTextField first, JTextField second, JTextField result)
   {
      inputOne = first;
      inputTwo = second;
      output = result;
   }
   public void actionPerformed(ActionEvent e) {
      double first, second;
      first =
          Double.parseDouble(inputOne.getText().trim());
      second =
          Double.parseDouble(inputTwo.getText().trim());
      if (e.getActionCommand().equals("Add"))
         output.setText(String.valueOf(first + second));
      else
         output.setText(String.valueOf(first - second));
   }
}
```

The `DoMath` object gets access to the input/ output text fields through the `ActionEvent` parameter passed through the `actionPerformed()` method.

Border Layout Manager



```
class BorderLayoutTest { // import statements omitted
   public static void main(String[] args) {
      JFrame frame = new JFrame("BorderLayout");
      Container pane = frame.getContentPane();
      pane.add(
          new JButton("North Button"),BorderLayout.NORTH);
      pane.add(
          new JButton("South Button"),BorderLayout.SOUTH);
      pane.add(
          new JButton("East Button"), BorderLayout.EAST);
      pane.add(
          new JButton("West Button"), BorderLayout.WEST);
      pane.add(
        new JButton("Center Button"),BorderLayout.CENTER);
      frame.pack();
      frame.setVisible(true);
   }
}
```

Layout Managers

- FlowLayout
  - Adjusts layout as window size changes (default)
- GridLayout
  - Displays components of specified rows and columns
- GridBagLayout
  - Lays components in a grid; some components span multiple cells. This is a sophisticated layout manager.
  - zyBooks has a good example
- BorderLayout
  - Arranges components in N, S, E, W, and Center

Layout Managers

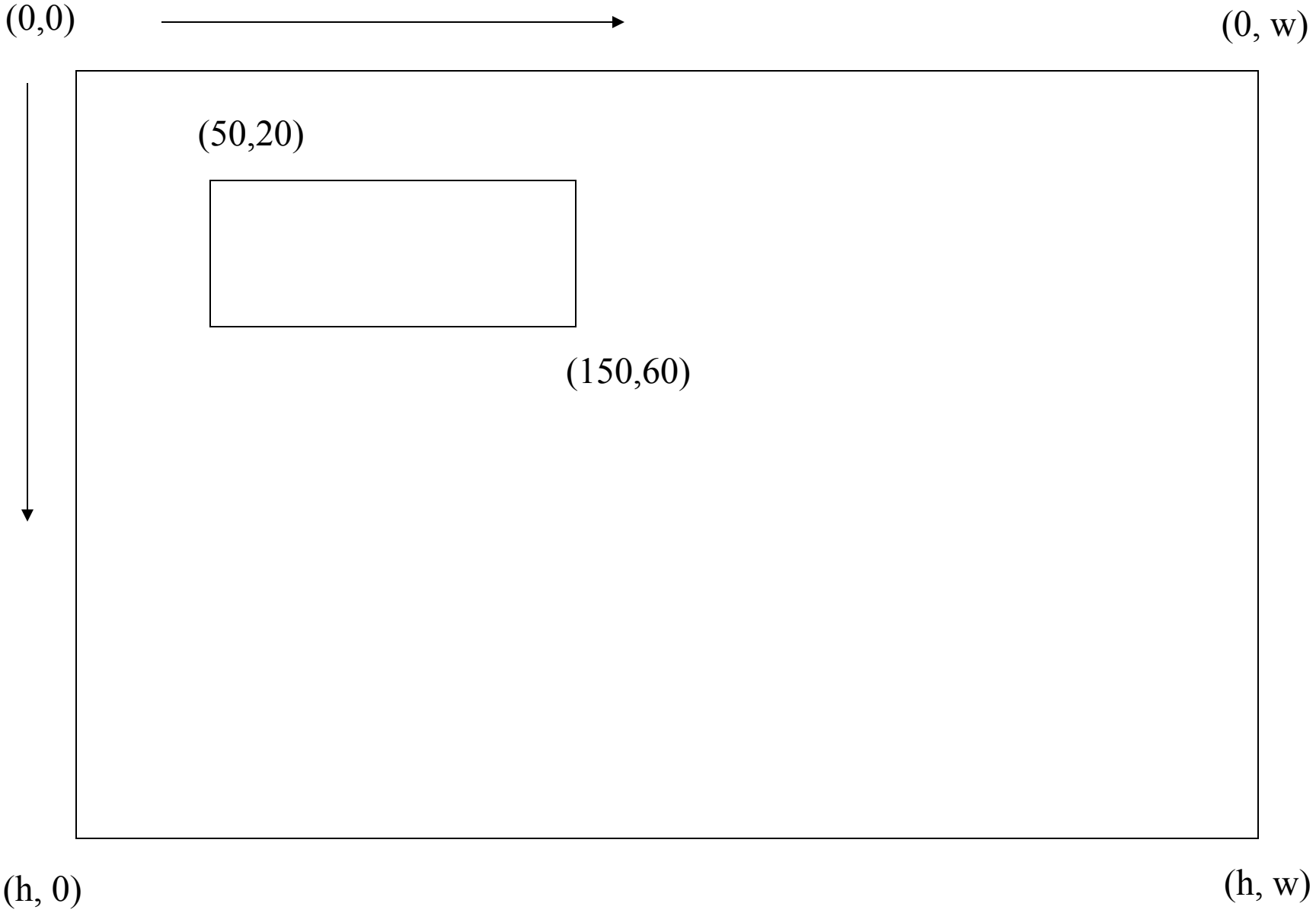
- BoxLayout
  - Puts components in a single row or column.
- CardLayout
  - Lets you implement an area that contains different components at different times. A CardLayout is often controlled by a combo box.
- GroupLayout
  - See java documentation for details
- SpringLayout
  - Lets you specify precise relationships between the edges of components under its control.

Drawing with Swing

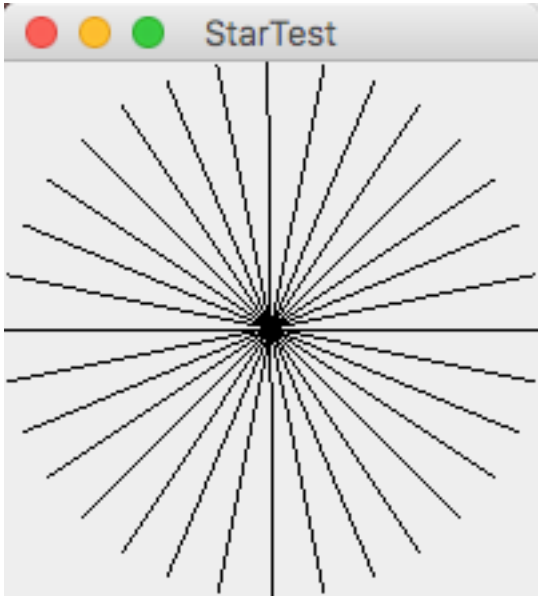
- Drawing is done on an instance of class java.awt.Graphics (or java.awt.Graphics2D).
- Using Graphics, units are always in terms of pixels.
- The origin is in the upper left corner with positive directions being down and to the right.

Drawing with Swing

- Many useful methods for drawing provided by java.awt.Graphics class. See documentation
- https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html


(0,0)
(0, w)
(50,20)
(150,60)
(h, 0)
(h, w)

```
//StarTest.java - display a starburst
import java.awt.*;
import javax.swing.*;
class StarTest {
   public static void main(String[] args) {
      JFrame frame = new JFrame("StarTest");
      Container pane = frame.getContentPane();
      Star star = new Star();
      pane.add(star);
      frame.pack();
      frame.setVisible(true);
   }
}
```

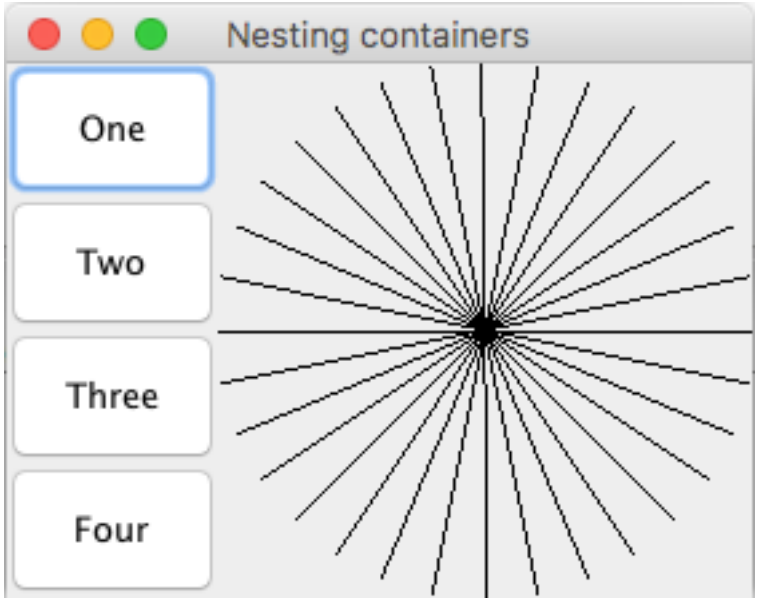


```
class Star extends JComponent {//import statements omitted
   public void paintComponent(Graphics g) {
      for (double angle = 0; angle < Math.PI;
                             angle = angle + Math.PI / 16)
      {
         double x1, x2, y1, y2;
         // compute coordinates of endpoints of a line
         // cosine and sine range from -1 to 1
         // multiplying by RADIUS changes the
         // range to be from -RADIUS to RADIUS
         // adding RADIUS gives the final range of
         // 0 to 2 * RADIUS
         x1 = Math.cos(angle) * RADIUS + RADIUS;
         y1 = Math.sin(angle) * RADIUS + RADIUS;
         x2 = Math.cos(angle + Math.PI) * RADIUS +RADIUS;
         y2 = Math.sin(angle + Math.PI) * RADIUS +RADIUS;
         g.drawLine((int)x1, (int)y1, (int)x2, (int)y2);
      }
   }
```

```
   // make the JComponent big enough to show the image
   public Dimension getMinimumSize() {
      return new Dimension(2 * RADIUS, 2 * RADIUS);
   }
   public Dimension getPreferredSize() {
      return new Dimension(2 * RADIUS, 2 * RADIUS);
   }
   private static final int RADIUS = 100;
}
```

`getMinimumSize()` and `getPreferredSize()` determine the size of the JComponent.

Nesting Containers



```
class Nesting { //import statements omitted
   public static void main(String[] args) {
      JFrame frame = new JFrame("Nesting containers");
      Container pane = frame.getContentPane();
      JPanel panel = new JPanel();
      panel.setLayout(new GridLayout(4, 1));
      panel.add(new JButton("One"));
      panel.add(new JButton("Two"));
      panel.add(new JButton("Three"));
      panel.add(new JButton("Four"));
      pane.add(panel, BorderLayout.WEST);
      pane.add(new Star(), BorderLayout.CENTER);
      frame.pack();
      frame.setVisible(true);
   }
}
```

Mouse Input

- This next program will allow us to draw on the screen with a mouse.

```
//SimplePaint.java - a program to draw with the mouse

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

class SimplePaint {
   public static void main(String[] args) {
      JFrame frame = new JFrame("SimplePaint");
      Container pane = frame.getContentPane();
      DrawingCanvas canvas = new DrawingCanvas();
      PaintListener listener = new PaintListener();
      canvas.addMouseMotionListener(listener);
      pane.add(canvas);
      frame.pack();
      frame.setVisible(true);
   }
}
```

```
// DrawingCanvas.java - a blank Canvas

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

class DrawingCanvas extends JComponent {
   //these methods are needed to give the component a size
   public Dimension getMinimumSize() {
      return new Dimension(SIZE, SIZE);
   }
   public Dimension getPreferredSize() {
      return new Dimension(SIZE, SIZE);
   }
   private static final int SIZE = 500;
}
```

```
//PaintListener.java - do the actual drawing

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

class PaintListener implements MouseMotionListener
{
   public void mouseDragged(MouseEvent e) {
      DrawingCanvas canvas =
                     (DrawingCanvas)e.getSource();
      Graphics g = canvas.getGraphics();
      g.fillOval(e.getX() - radius, e.getY() – radius,
                 diameter, diameter);
   }
   public void mouseMoved(MouseEvent e){}
   private int radius = 3;
   private int diameter = radius * 2;
}
```