Lecture 17

Layout Managers

Layout managers

- Layouts tell Java where to put components in containers (JPanel, content pane, etc).
- Every panel (and other container) has a default layout, but it's better to set the layout explicitly for clarity.
- Create a new layout object and use the container's setLayout method to set the layout.
- Each layout has its own way to resize components to fit the layout, and you must become familiar with these.

The layouts in this lecture

- FlowLayout
- BorderLayout
- GridLayout
- BoxLayout
- Null layout

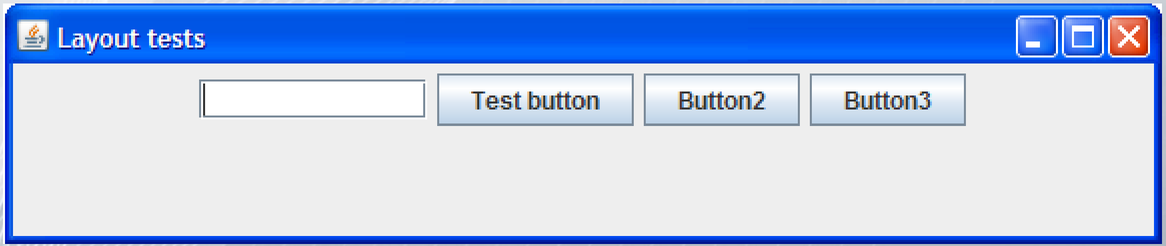
FlowLayout

- Arranges components from left-to-right and top-to-bottom, centering components horizontally with a five pixel gap between them.
- When a container size is changed (a window is resized), FlowLayout recomputes new positions for all components subject to these constraints.
- Use FlowLayout because it's quick and easy. We often start with a FlowLayout and later switch to a better layout, if necessary
- It's default layout for JPanel

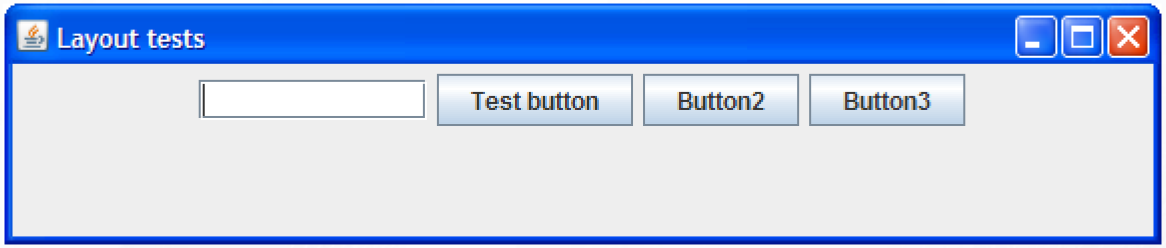
FlowLayout

```
public class FlowPanel extends JPanel{
    public FlowPanel() {
        setLayout(new FlowLayout());

        JTextField text = new JTextField(10);
        add(text);
        JButton b = new JButton("Test button");
        add(b);
        add(new JButton("Button2"));
        add(new JButton("Button3"));
    }
}
```

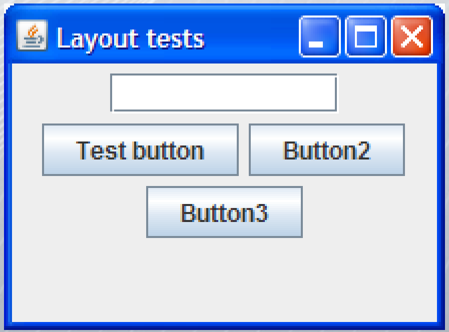


FlowLayout



The window above is the default size after packing the FlowLayout.

The window bellow shows the same window after it has been resized by dragging the lower-right corner.



FlowLayout

- Constructors:

```
new FlowLayout() // default is centered with 5 pixel gaps
new FlowLayout(int align)
new FlowLayout(int align, int hgap, int vgap)
```

- Alignment

```
FlowLayout.LEFT
FlowLayout.CENTER //the default
FlowLayout.RIGHT
```

- Spacing

  The default spacing is good for most purposes and is rarely changed. hgap is the size in pixels of the horizontal gap (distance) between components, and vgap is the vertical gap.

FlowLayout

- Typical uses:
  - Quick implementation. This is the most common use of FlowLayout. You can get something working quickly, and change it, if necessary, in a later iteration.
  - Space around component in a BorderLayout. As a subpanel to keep components from expanding.
  - Multiple components in a BorderLayout region. When you want to add several components to a BorderLayout region, drop them into a FlowLayout panel and put that in the BorderLayout
- Problem:

  FlowLayout makes components as small as possible, and does not use their preferred size.

BorderLayout

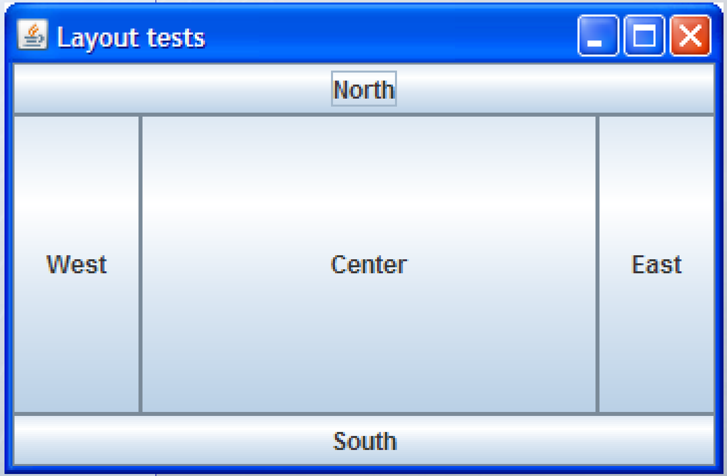
- Divides a containerinto 5 geographical sections: North, South, East, West, and Center.
- This is a very commonly used layout.
- Components start at their preferred size, but are expanded as needed to fill the region they are in.
- You can add at most one component to each region
- If there is nothing in an region, its size will be reduced to zero
- North and South cells are stretched horizontally, and East and West are stretched vertically to fill all the space. The center is stretched vertically and horizontally as needed

BorderLayout

```
public class BorderPanel extends JPanel{
    public BorderPanel() {
        //create components
        JButton north  = new JButton("North");
        JButton east   = new JButton("East");
        JButton south  = new JButton("South");
        JButton west   = new JButton("West");
        JButton center = new JButton("Center");

        //set layout
        BorderLayout l = new BorderLayout();
        setLayout(l);

        //add the components to the panel
        add(north , BorderLayout.NORTH);
        add(east  , BorderLayout.EAST);
        add(south , BorderLayout.SOUTH);
        add(west  , BorderLayout.WEST);
        add(center, BorderLayout.CENTER);
    }
}
```



BorderLayout

- To prevent component resizing, add a component to a JPanel with FlowLayout, and then add that panel to the BorderLayout.

  This is a common way to prevent resizing. The FlowLayout panel will stretch, but the component in it will not.

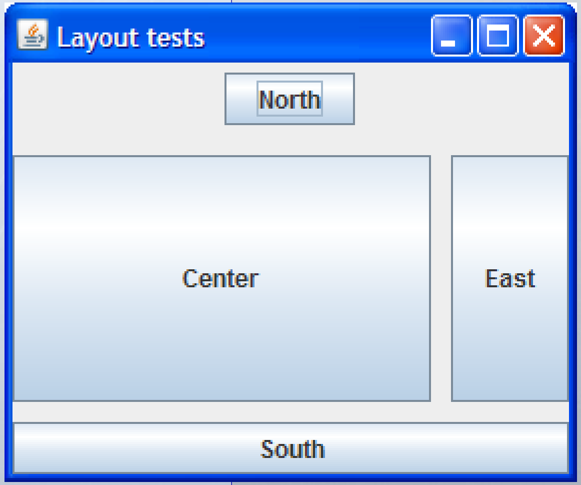
- If nothing has been added to a region, the neighboring regions expand to fill that space.
- BorderLayout is default for JFrame

BorderLayout

```
public class BorderPanel extends JPanel{
    public BorderPanel() {
        //create components
        JPanel north  = new JPanel();
        north.add(new JButton("North"));
        JButton east   = new JButton("East");
        JButton south  = new JButton("South");
        JButton center = new JButton("Center");

        //set layout
        BorderLayout l = new BorderLayout(10, 10);
        setLayout(l);

        //add the components to the panel
        add(north , BorderLayout.NORTH);
        add(east  , BorderLayout.EAST);
        add(south , BorderLayout.SOUTH);
        add(center, BorderLayout.CENTER);
    }
}
```



BorderLayout

- If you don't need any space between regions, use the default constructor. You can also specify the number of pixels between regions.

Set horizontal and vertical gap

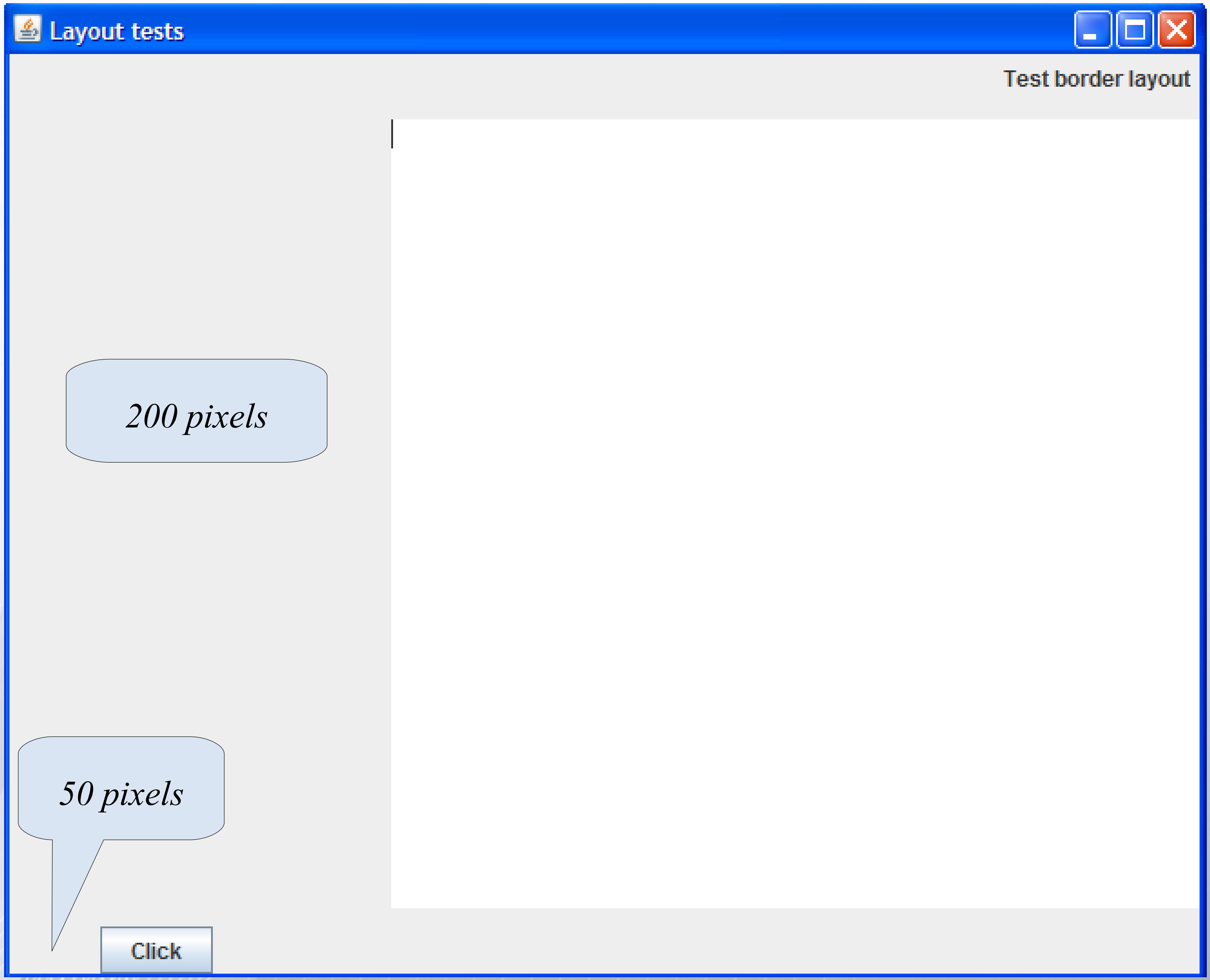
```
BorderLayout l = new BorderLayout();

BorderLayout l2 = new BorderLayout(10, 10);
```

- BorderLayout is often used in combination with other panels and layout (a panel with FlowLayout)

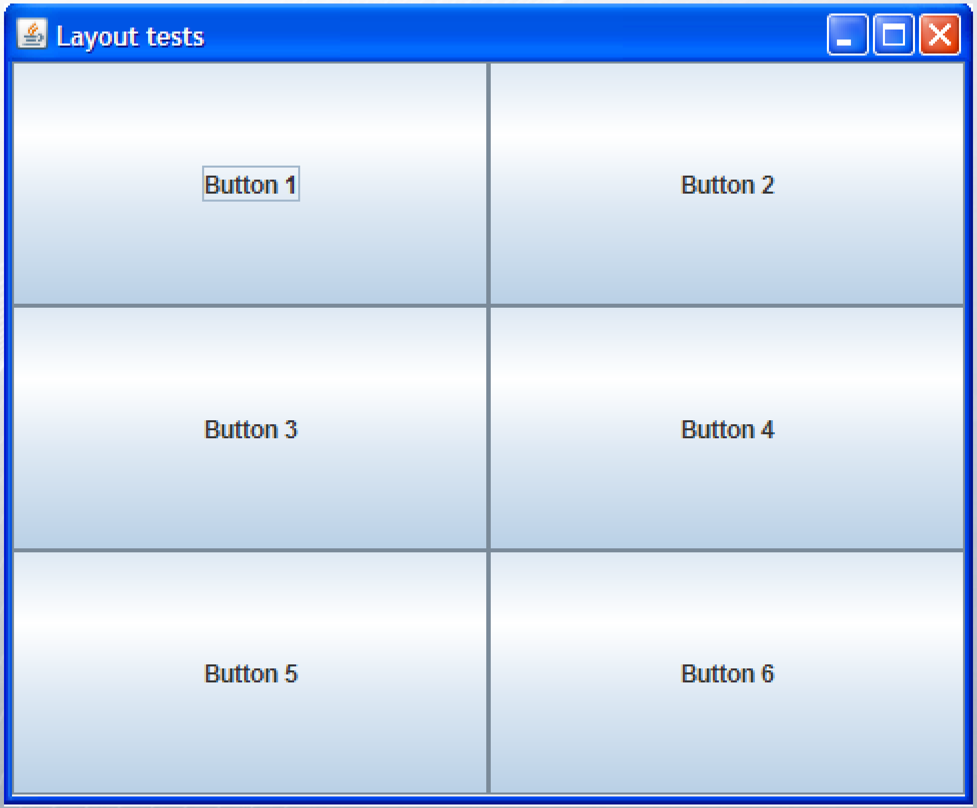
Exercise

Try to do this:



GridLayout

- GridLayout lays out components in a rectangular grid
- All cells are equal size (it forces components to have the same size)



GridLayout

```java
public class GridPanel extends JPanel {
    public GridPanel(){
        setLayout(new GridLayout(3, 2));
        add(new JButton("Button 1"));
        add(new JButton("Button 2"));
        add(new JButton("Button 3"));
        add(new JButton("Button 4"));
        add(new JButton("Button 5"));
        add(new JButton("Button 6"));
    }
}
```

Constructors:

```java
setLayout(new GridLayout());  // One row.  Columns expand.
setLayout(new GridLayout(rows, cols));
setLayout(new GridLayout(rows, cols, hgap, vgap));
```

GridLayout

- To add Components to a GridLayout:
  - Use the add(. . .) method to add components to a container with a GridLayout.
  - You do not (can not) use the row and column to tell where to add the components (they are added starting at the top left and going across the row)
- Empty Cells
  - There is no way to leave a cell empty in a grid layout.
  - Although it often works ok to leave the final cells empty, this is not recommended
  - The easiest way to do this is to put an empty label in any

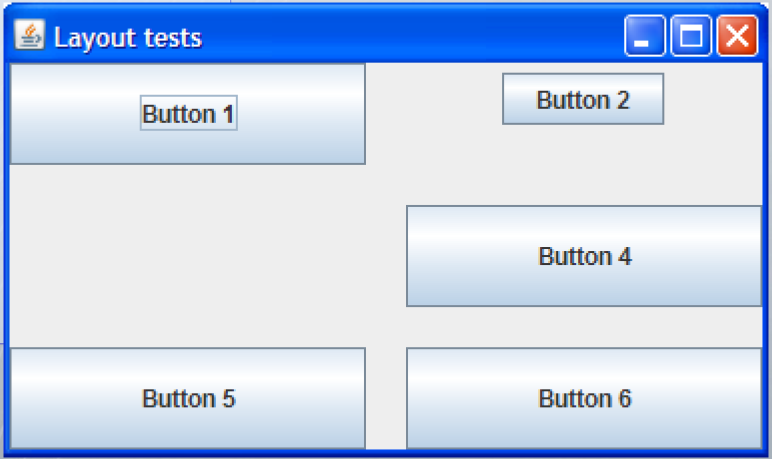
GridLayout

```java
public class GridPanel extends JPanel {
    public GridPanel(){
        setLayout(new GridLayout(3, 2, 20, 20));

        add(new JButton("Button 1"));

        JPanel p = new JPanel();
        JButton b2 = new JButton("Button 2");
        p.add(b2);
        add(p);

        add(new JLabel());
        add(new JButton("Button 4"));
        add(new JButton("Button 5"));
        add(new JButton("Button 6"));
    }
}
```



BoxLayout and Boxes

- BoxLayout arranges components either horizontally or vertically in a panel.
- You can control alignment and spacing of the components.
- Complicated layouts can be made by combining many panels, some with horizontal layout and some with vertical layouts.
- The class javax.swing.Box is similar to Jpanel with BoxLayout

BoxLayout and Boxes

- How to create a JPanel with BoxLayout?

  Choose either a horizontal layout (BoxLayout.X_AXIS) or vertical layout (BoxLayout.Y_AXIS) for a Jpanel.

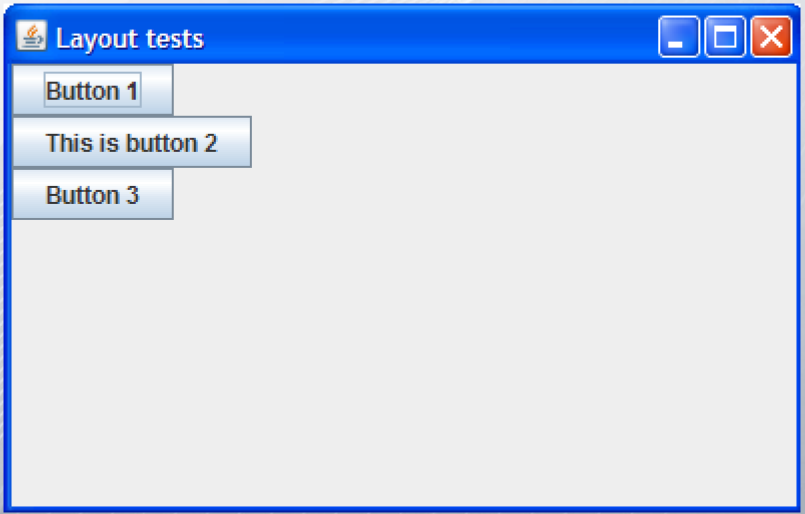
```
BoxLayout l = new BoxLayout(this, BoxLayout.Y_AXIS);
setLayout(l);
```

- Unlike other layouts, the panel/container must be passed to the BoxLayout constructor.

BoxLayout and Boxes

```java
public class BoxPanel extends JPanel {
    public BoxPanel(){
        BoxLayout l = new BoxLayout(this, BoxLayout.Y_AXIS);
        setLayout(l);

        add(new JButton("Button 1"));
        add(new JButton("This is button 2"));
        add(new JButton("Button 3"));
    }
}
```



BoxLayout and Boxes

- The Box class
  - The Box class was designed to be a simple, and slightly more efficient, substitute for a JPanel with a BoxLayout.
  - The Box class has a number of necessary methods for working with BoxLayouts.
  - Because Box is a Container with BoxLayout, all discussions of spacing and alignment apply equally well to both JPanels with BoxLayouts and Boxes.

BoxLayout and Boxes

- Creating Boxes

  You can create the two kinds of boxes with:

```
Box vb = Box.createVerticalBox();
Box hb = Box.createHorizontalBox();
```

- No Borders on Boxes

  Boxes are lighter weight (ie, more efficient) than JPanel, but they don't support Borders. If you need borders, either use a JPanel with BoxLayout, or put the Box into a JPanel with a border.

BoxLayout and Boxes

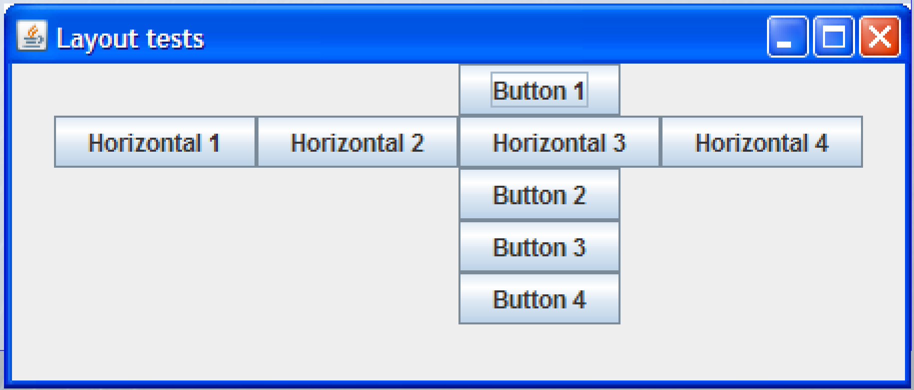
```
public class BoxPanel1 extends JPanel{
    public BoxPanel1(){
        this.setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));

        add(new JButton("Button 1"));

        Box horizontalBox = Box.createHorizontalBox();

        horizontalBox.add(new JButton("Horizontal 1"));
        horizontalBox.add(new JButton("Horizontal 2"));
        horizontalBox.add(new JButton("Horizontal 3"));
        horizontalBox.add(new JButton("Horizontal 4"));
        add(horizontalBox);
        //horizontalBox.setAlignmentX(Box.LEFT_ALIGNMENT);

        add(new JButton("Button 2"));
        add(new JButton("Button 3"));
        add(new JButton("Button 4"));
    }
}
```

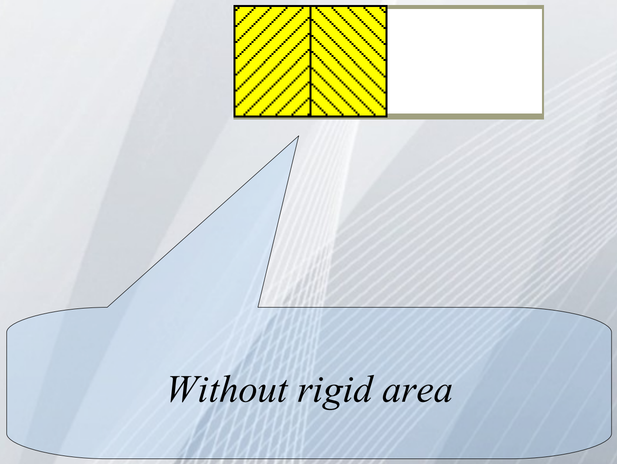


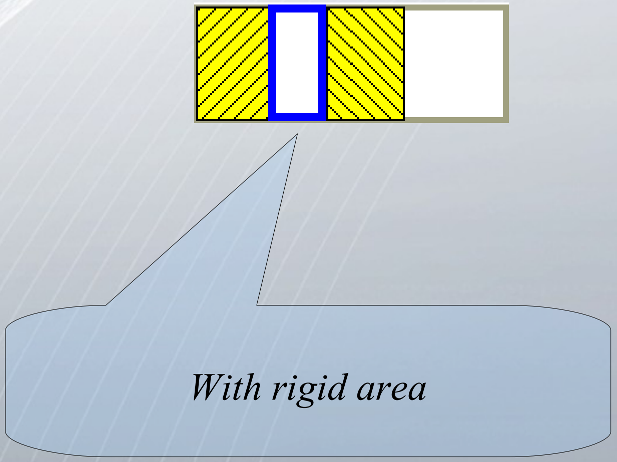
Using rigid area and glue

- Rigid area

Use this when you want a fixed-size space between two components.

```
container.add(firstComponent);
container.add(Box.createRigidArea(new Dimension(20,0)));
container.add(secondComponent);
```

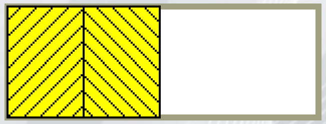




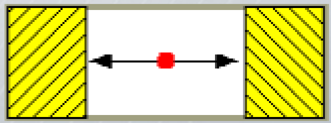
Using rigid area and glue

- Glue
  - Use this to specify where excess space in a layout should go.
  - Think of it as a kind of elastic glue — stretchy and expandable, yet taking up no space unless you pull apart the components that it is sticking to.

```
container.add(firstComponent);
container.add(Box.createHorizontalGlue());
container.add(secondComponent);
```



Without horizontal glue

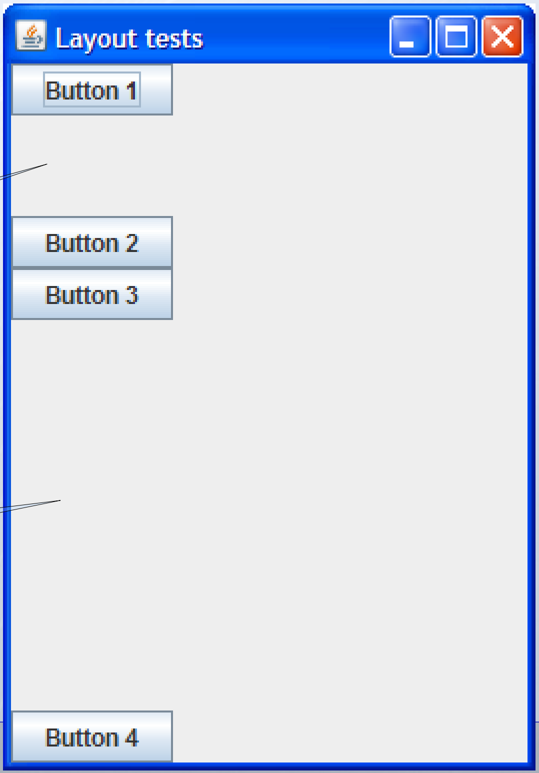


With horizontal glue

BoxLayout and Boxes

```
public class BoxPanel2 extends JPanel{
    public BoxPanel2(){
        this.setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));

        add(new JButton("Button 1"));
        // add rigit area
        add(Box.createRigidArea(new Dimension(0, 50)));
        add(new JButton("Button 2"));
        add(new JButton("Button 3"));
        // add horizontal glue
        add(Box.createVerticalGlue());
        add(new JButton("Button 4"));
    }
}
```



Rigid area

Vertical glue

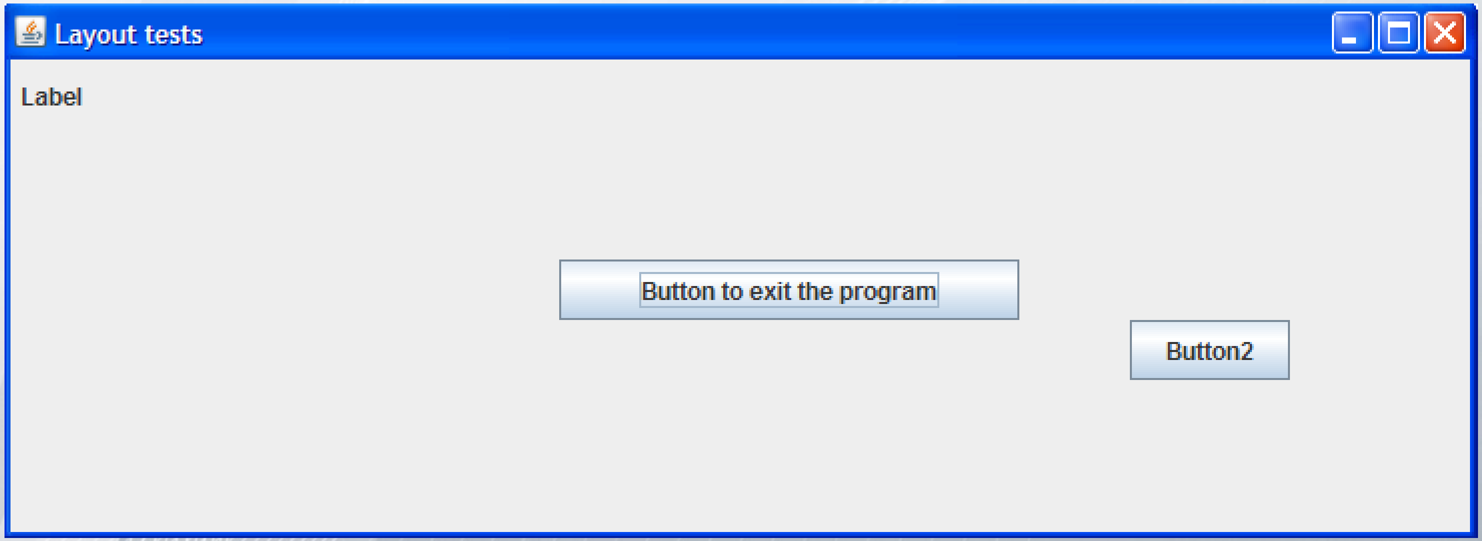
NullLayout (absolute positioning)

- You can set the layout manager to null
- But this is generally bad practice
- Using null layout you must set component's bounds in pixels

```java
public class NullLayoutPanel extends JPanel{
    public NullLayoutPanel() {
        setLayout(null);
        JLabel l = new JLabel("Label");
        JButton b = new JButton("Button to exit the program");
        JButton b1 = new JButton("Button2");
        l.setBounds(5, 10, 62, 16);
        b.setBounds(274, 100, 230, 30);
        b1.setBounds(559, 130, 80, 30);
        add(l);
        add(b);
        add(b1);
    }
}
```

NullLayout (absolute positioning)

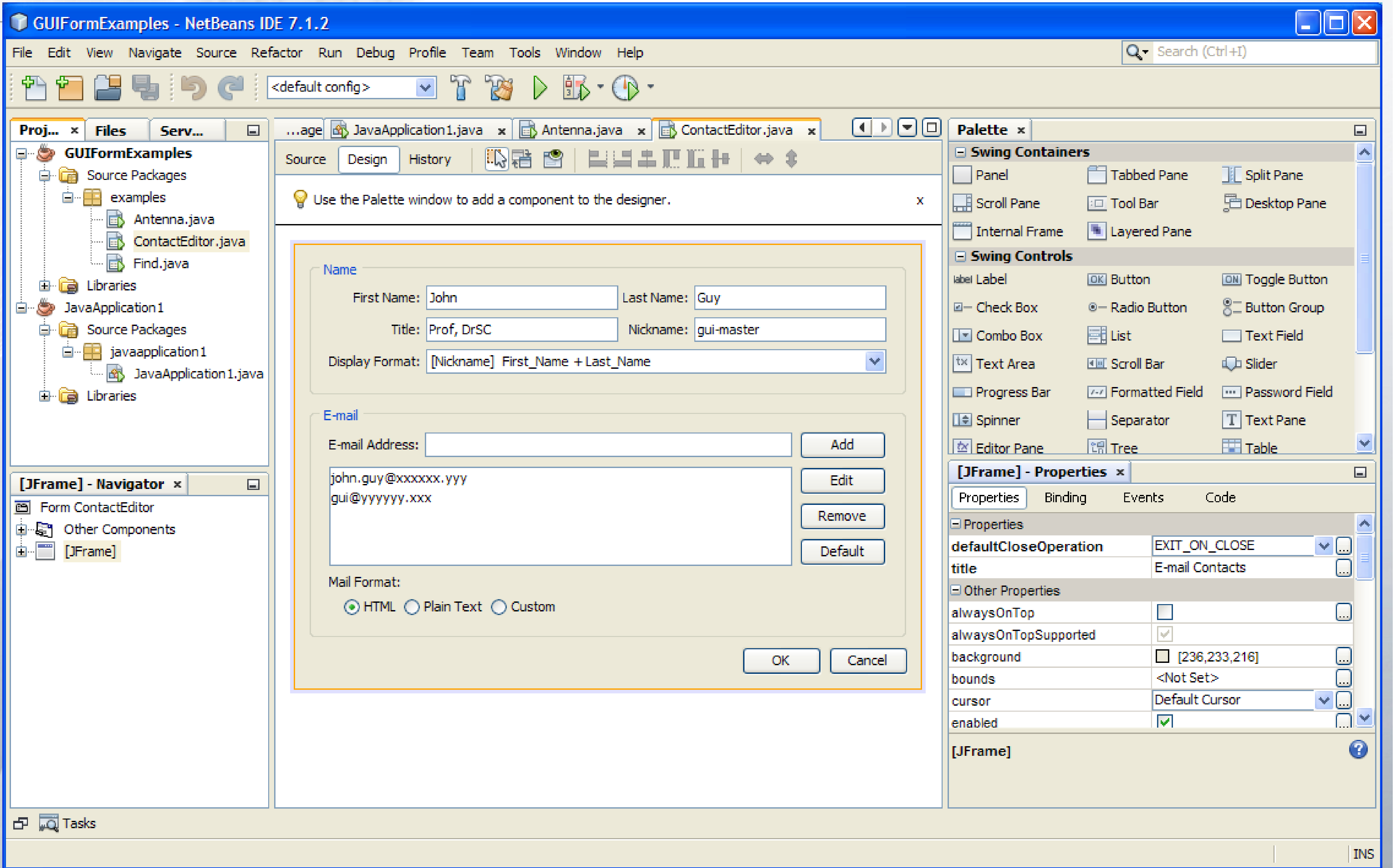
- Problems with null layout that regular layouts don't have:
  - Difficult to change, therefore hard (expensive) to maintain
  - Hard to get right in the first place
  - The components not resizable



More complex layouts and GUI builders

- Creating your layouts by hand is simple for the simple layouts, but for really good layouts you can use GridBagLayout
- It's a bit difficult but very powerfull
- Another way to deal with a layout in large and complex GUI application is to use GUI builder
- NetBeans IDE has powerful GUI builder
- There are many plugins for GUI builders for Eclipse IDE

GUI builders. NetBeans IDE


GUIFormExamples - NetBeans IDE 7.1.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
Proj... Files Serv...
GUIFormExamples
Source Packages
examples
Antenna.java
ContactEditor.java
Find.java
Libraries
JavaApplication1
Source Packages
javaapplication1
JavaApplication1.java
Libraries
[JFrame] - Navigator
Form ContactEditor
Other Components
[JFrame]
Tasks
...age JavaApplication1.java Antenna.java ContactEditor.java
Source Design History
Palette
Swing Containers
Panel Tabbed Pane Split Pane
Scroll Pane Tool Bar Desktop Pane
Internal Frame Layered Pane
Swing Controls
Label Button Toggle Button
Check Box Radio Button Button Group
Combo Box List Text Field
Text Area Scroll Bar Slider
Progress Bar Formatted Field Password Field
Spinner Separator Text Pane
Editor Pane Tree Table
[JFrame] - Properties
Properties Binding Events Code
INS

Use the Palette window to add a component to the designer.

Name

First Name: John   Last Name: Guy

Title: Prof, DrSC   Nickname: gui-master

Display Format: [Nickname] First_Name + Last_Name

E-mail

E-mail Address: Add

john.guy@xxxxxx.yyy
gui@yyyyyy.xxx

Edit Remove Default

Mail Format: [✓ HTML] Plain Text Custom

OK Cancel

Properties

defaultCloseOperation	EXIT_ON_CLOSE
title	E-mail Contacts

Other Properties

alwaysOnTop	☐
alwaysOnTopSupported	✓
background	☐ [236,233,216]
bounds	<Not Set>
cursor	Default Cursor
enabled	✓

[JFrame]