



Layout Managers

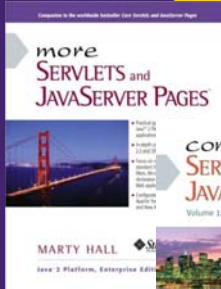
Arranging Elements in Windows

Originals of slides and source code for examples: <http://courses.coreservlets.com/Course-Materials/java.html>
Also see the Java 8 tutorial – <http://www.coreservlets.com/java-8-tutorial/>
and customized Java training courses (onsite or at public venues) – <http://courses.coreservlets.com/java-training.html>



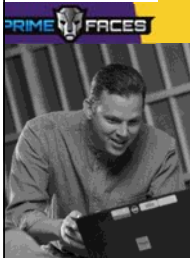
3

Customized Java EE Training: <http://courses.coreservlets.com/>
Java 7, Java 8, JSF 2.2, PrimeFaces, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.



For live Java-related training, email hall@coreservlets.com

Marty is also available for consulting and development support



Taught by lead author of *Core Servlets & JSP*, co-author of *Core JSF* (4th Ed), & this tutorial. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
 - JSF 2.2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 7 or 8 programming, custom mix of topics
 - Courses available in any state or country. Maryland/DC area companies can also choose afternoon/evening courses.
- Courses developed and taught by coreservlets.com experts (edited by Marty)
 - Spring, Hibernate/JPA, GWT, Hadoop, HTML5, RESTful Web Services

Contact hall@coreservlets.com for details



Topics in This Section

- **How layout managers simplify interface design**
- **Standard layout managers**
 - FlowLayout, BorderLayout, CardLayout, GridLayout, GridBagLayout
- **Positioning components manually**
- **Strategies for using layout managers effectively**

5

Layout Managers

- **Assigned to each Container**
 - Give *sizes* and *positions* to components in the window
 - Helpful for windows whose size changes or that display on multiple operating systems
- **Relatively easy for simple layouts**
 - But, it is surprisingly hard to get complex layouts with a single layout manager
- **Controlling complex layouts**
 - Use nested containers (each with its own layout manager)
 - Use invisible components and layout manager options
 - Write your own layout manager
 - Turn some layout managers off and arrange some things manually

6



Simple Layout Managers



Customized Java EE Training: <http://courses.coreservlets.com/>

Java 7, Java 8, JSF 2.2, PrimeFaces, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

7

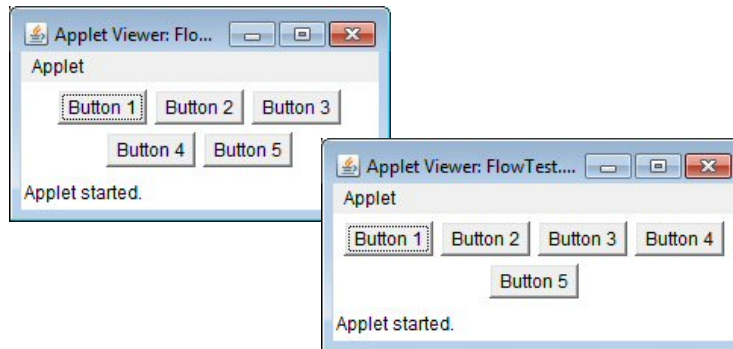
FlowLayout

- **Default layout for Panel, JPanel, and Applet**
- **Behavior**
 - Resizes components to their **preferred** size
 - Places components in rows **left to right, top to bottom**
 - Rows are **centered** by default
- **Constructors**
 - **FlowLayout()**
 - Centers each row and keeps 5 pixels between entries in a row and between rows
 - **FlowLayout(int alignment)**
 - Same 5 pixels spacing, but changes the alignment of the rows
 - `FlowLayout.LEFT`, `FlowLayout.RIGHT`, `FlowLayout.CENTER`
 - **FlowLayout(int alignment, int hGap, int vGap)**
 - Specify the alignment as well as the horizontal and vertical spacing between components (in pixels)

8

FlowLayout: Example

```
public class FlowTest extends Applet {
    public void init() {
        // setLayout(new FlowLayout()); [Default]
        for(int i=1; i<6; i++) {
            add(new Button("Button " + i));
        }
    }
}
```



9

BorderLayout

- **Default for Frame, JFrame, Dialog, JApplet**
- **Behavior**
 - Divides the Container into **five regions**
 - Each region is identified by a corresponding BorderLayout constant
 - NORTH, SOUTH, EAST, WEST, and CENTER
 - NORTH and SOUTH **respect the preferred height** of the component
 - EAST and WEST **respect the preferred width** of the component
 - CENTER is given the remaining space
- **Is allowing a maximum of five components too restrictive? Why not?**

10

BorderLayout (Continued)

- **Constructors**

- BorderLayout()
 - Border layout with no gaps between components
- BorderLayout(int hGap, int vGap)
 - Border layout with the specified empty pixels between regions

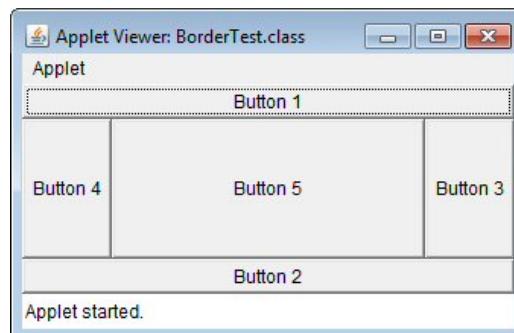
- **Adding Components**

- add(component, BorderLayout.*REGION*)
- Always specify the region in which to add the component
 - CENTER is the default, but specify it explicitly to avoid confusion with other layout managers

11

BorderLayout: Example

```
public class BorderLayoutTest extends Applet {  
    public void init() {  
        setLayout(new BorderLayout());  
        add(new Button("Button 1"), BorderLayout.NORTH);  
        add(new Button("Button 2"), BorderLayout.SOUTH);  
        add(new Button("Button 3"), BorderLayout.EAST);  
        add(new Button("Button 4"), BorderLayout.WEST);  
        add(new Button("Button 5"), BorderLayout.CENTER);  
    }  
}
```



12

GridLayout

- **Behavior**

- Divides window into **equal-sized rectangles** based upon the number of rows and columns specified
 - Items placed into cells left-to-right, top-to-bottom, based on the order added to the container
- Ignores the preferred size of the component; each component is **resized to fit into its grid cell**
- Too few components results in blank cells
- Too many components results in extra columns

13

GridLayout (Continued)

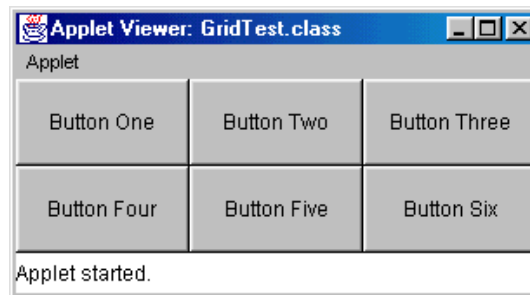
- **Constructors**

- `GridLayout()`
 - Creates a single row with one column allocated per component
- `GridLayout(int rows, int cols)`
 - Divides the window into the specified number of rows and columns
 - Either rows or cols (but not both) can be zero
- `GridLayout(int rows, int cols, int hGap, int vGap)`
 - Uses the specified gaps between cells

14

GridLayout, Example

```
public class GridTest extends Applet {  
    public void init() {  
        setLayout(new GridLayout(2,3)); // 2 rows, 3 cols  
        add(new Button("Button One"));  
        add(new Button("Button Two"));  
        add(new Button("Button Three"));  
        add(new Button("Button Four"));  
        add(new Button("Button Five"));  
        add(new Button("Button Six"));  
    }  
}
```



15

CardLayout

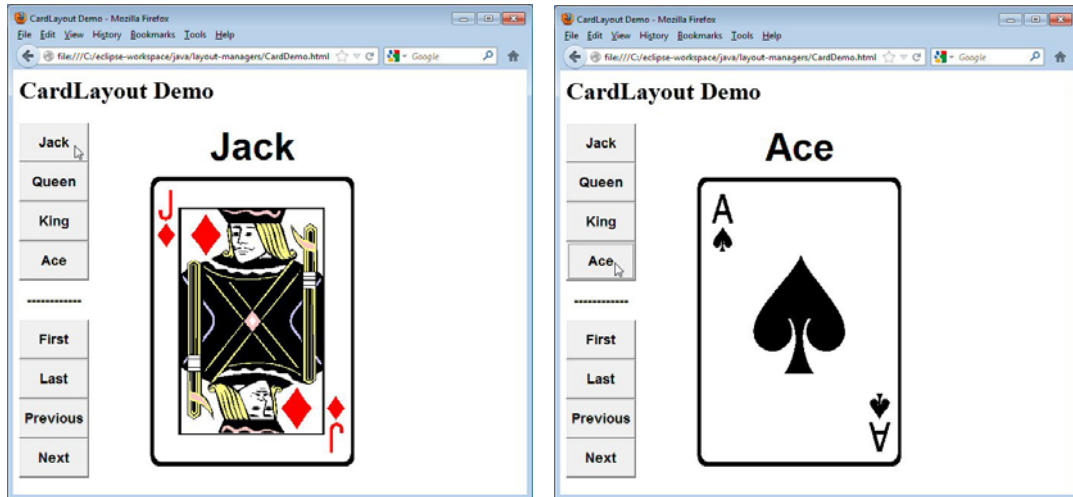
- **Behavior**

- Stacks components on top of each other, displaying the top one
- Associates a name with each component in window

```
Panel cardPanel;  
CardLayout layout new CardLayout();  
cardPanel.setLayout(layout);  
...  
cardPanel.add("Card 1", component1);  
cardPanel.add("Card 2", component2);  
...  
layout.show(cardPanel, "Card 1");  
layout.first(cardPanel);  
layout.next(cardPanel);
```

16

CardLayout, Example



17

© 2014 Marty Hall



GridBagLayout



18

Customized Java EE Training: <http://courses.coreservlets.com/>
Java 7, Java 8, JSF 2.2, PrimeFaces, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

GridBagLayout

- **Behavior**

- Divides the window into grids, without requiring the components to be the same size
 - About three times more flexible than the other standard layout managers, but *nine* times harder to use
- Each component managed by a grid bag layout is associated with an instance of `GridBagConstraints`
 - The `GridBagConstraints` specifies:
 - How the component is laid out in the display area
 - In which cell the component starts and ends
 - How the component stretches when extra room is available
 - Alignment in cells
- Java 5 introduced `SpringLayout`, with similar power but much less complexity

19

GridBagLayout: Basic Steps

- **Set the layout, saving a reference to it**

```
GridBagLayout layout = new GridBagLayout();
setLayout(layout);
```
- **Allocate a `GridBagConstraints` object**

```
GridBagConstraints constraints =
    new GridBagConstraints();
```
- **Set up the `GridBagConstraints` for component 1**

```
constraints.gridx = x1;
constraints.gridy = y1;
constraints.gridwidth = width1;
constraints.gridheight = height1;
```
- **Add component 1 to the window, including constraints**

```
add(component1, constraints);
```
- **Repeat the last two steps for each remaining component**

20

GridBagConstraints

- Copied when component added to window

- Thus, can reuse the GridBagConstraints

```
GridBagConstraints constraints =  
    new GridBagConstraints();  
constraints.gridx = x1;  
constraints.gridy = y1;  
constraints.gridwidth = width1;  
constraints.gridheight = height1;  
add(component1, constraints);  
constraints.gridx = x1;  
constraints.gridy = y1;  
add(component2, constraints);
```

21

GridBagConstraints Fields

- **gridx, gridy**

- Specifies the top-left corner of the component
- Upper left of grid is located at (gridx, gridy)=(0,0)
- Set to **GridBagConstraints.RELATIVE** to auto-increment row/column

```
GridBagConstraints constraints =  
    new GridBagConstraints();  
constraints.gridx =  
    GridBagConstraints.RELATIVE;  
container.add(new Button("one"),  
               constraints);  
container.add(new Button("two"),  
               constraints);
```

22

GridBagConstraints Fields (Continued)

- **gridwidth, gridheight**
 - Specifies the number of columns and rows the Component occupies
 - `constraints.gridwidth = 3;`
 - **GridBagConstraints.REMAINDER** lets the component take up the remainder of the row/column
- **weightx, weighty**
 - Specifies how much the cell will **stretch** in the x or y direction if space is left over
 - `constraints.weightx = 3.0;`
 - Constraint affects the cell, not the component (use `fill`)
 - Use a value of 0.0 for no expansion in a direction
 - Values are relative, not absolute

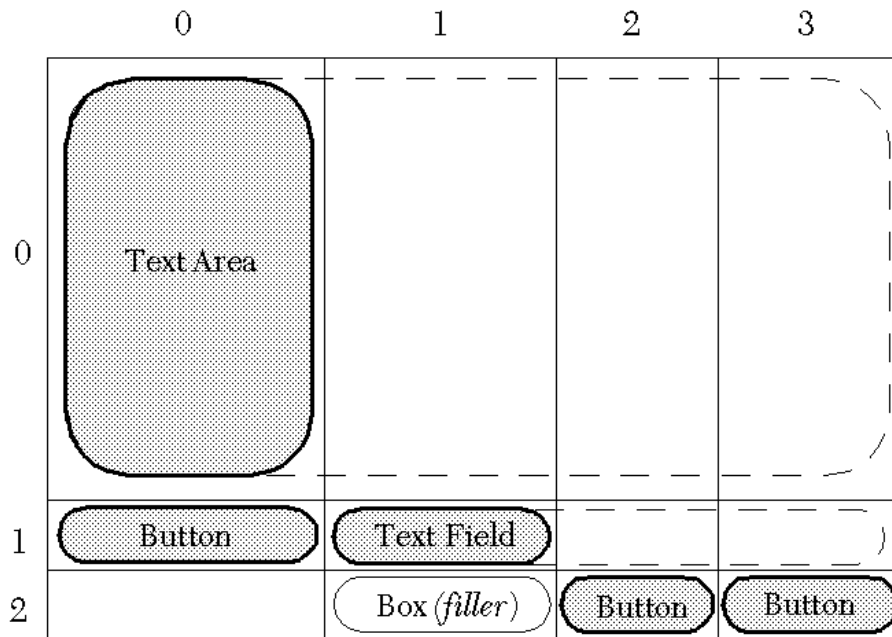
23

GridBagConstraints Fields (Continued)

- **fill**
 - Specifies what to do to an element that is smaller than the cell size
 - `constraints.fill = GridBagConstraints.VERTICAL;`
 - The size of row/column is determined by the widest/tallest element in it
 - Can be NONE, HORIZONTAL, VERTICAL, or BOTH
- **anchor**
 - If the fill is set to `GridBagConstraints.NONE`, then the anchor field determines where the component is placed
 - `constraints.anchor = GridBagConstraints.NORTHEAST;`
 - Can be NORTH, EAST, SOUTH, WEST, NORTHEAST, NORTHWEST, SOUTHEAST, or SOUTHWEST

24

GridBagLayout: Example



25

GridBagLayout: Example

```
public GridBagTest() {
    setLayout(new GridBagLayout());
    textArea = new JTextArea(12, 40); // 12 rows, 40 cols
    bSaveAs = new JButton("Save As");
    fileField = new JTextField("C:\\\\Document.txt");
    bOk = new JButton("OK");
    bExit = new JButton("Exit");
    GridBagConstraints c = new GridBagConstraints();
    // Text Area.
    c.gridx = 0;
    c.gridy = 0;
    c.gridwidth = GridBagConstraints.REMAINDER;
    c.gridheight = 1;
    c.weightx = 1.0;
    c.weighty = 1.0;
    c.fill = GridBagConstraints.BOTH;
    c.insets = new Insets(2,2,2,2); //t,l,b,r
    add(textArea, c);
    ...
}
```

26

GridBagLayout: Example (Continued)

```
// Save As Button.
c.gridx      = 0;
c.gridy     = 1;
c.gridwidth = 1;
c.gridheight = 1;
c.weightx   = 0.0;
c.weighty   = 0.0;
c.fill      = GridBagConstraints.VERTICAL;
add(bSaveAs,c);

// Filename Input (Textfield).
c.gridx      = 1;
c.gridwidth  = GridBagConstraints.REMAINDER;
c.gridheight = 1;
c.weightx    = 1.0;
c.weighty    = 0.0;
c.fill      = GridBagConstraints.BOTH;
add(fileField,c);
...

```

27

GridBagLayout: Example (Continued)

```
// Exit Button.
c.gridx      = 3;
c.gridwidth  = 1;
c.gridheight = 1;
c.weightx    = 0.0;
c.weighty    = 0.0;
c.fill      = GridBagConstraints.NONE;
add(bExit,c);

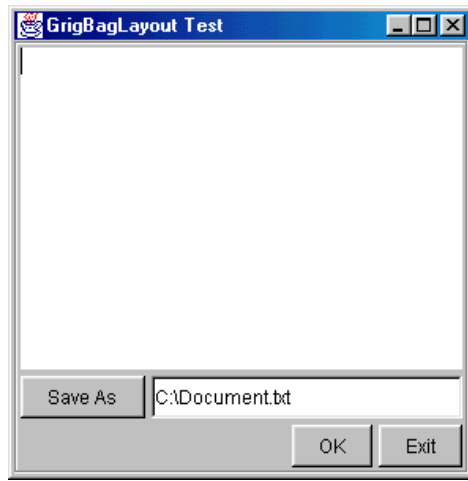
// Filler so Column 1 has nonzero width.
Component filler =
    Box.createRigidArea(new Dimension(1,1));
c.gridx      = 1;
c.weightx    = 1.0;
add(filler,c);

...
}

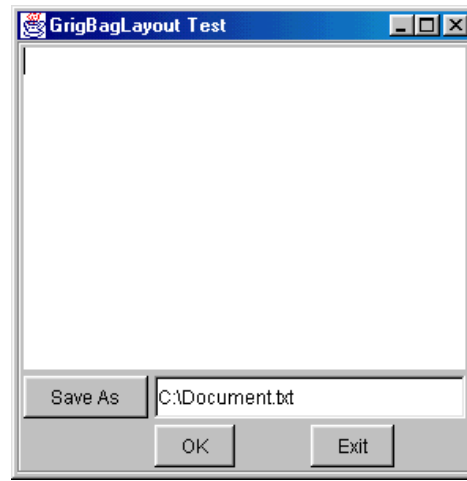
```

28

GridBagLayout: Result



With Box filler at (2,1)



Without Box filler at (2,1)

29

© 2014 Marty Hall



Strategies for Using Layout Managers



30

Customized Java EE Training: <http://courses.coreservlets.com/>

Java 7, Java 8, JSF 2.2, PrimeFaces, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Disabling the Layout Manager

- **Behavior**

- If the layout is set to `null`, then components must be *sized* and *positioned* by hand

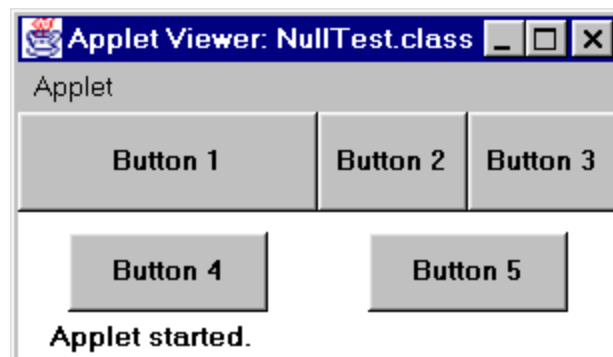
- **Positioning components**

- `component.setSize(width, height)`
- `component.setLocation(left, top)`
- or
- `component.setBounds(left, top, width, height)`

31

No Layout Manager: Example

```
setLayout(null);  
Button b1 = new Button("Button 1");  
Button b2 = new Button("Button 2");  
...  
b1.setBounds(0, 0, 150, 50);  
b2.setBounds(150, 0, 75, 50);  
...  
add(b1);  
add(b2);  
...
```



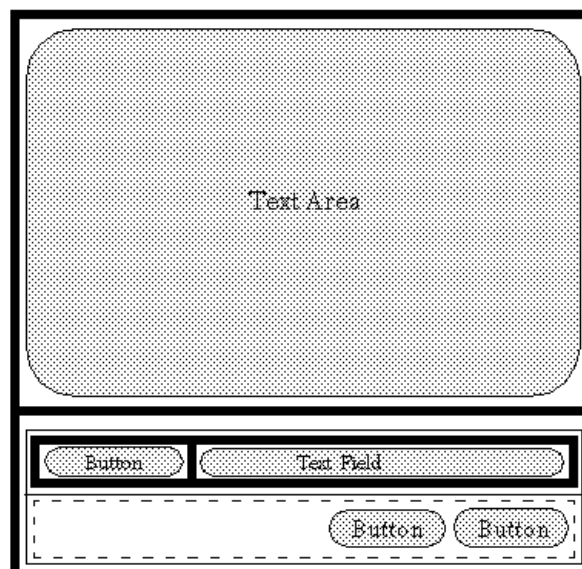
32

Using Layout Managers Effectively

- **Use nested containers**
 - Rather than struggling to fit your design in a single layout, try dividing the design into sections
 - Let each section be a panel with its own layout manager
- **Turn off the layout manager for some containers**
- **Adjust the empty space around components**
 - Change the space allocated by the layout manager
 - Override insets in the Container
 - Use a Canvas or a Box as an invisible spacer

33

Nested Containers: Example



34

Nested Containers: Example

```
public NestedLayout() {  
  
    setLayout(new BorderLayout(2,2));  
  
    textArea = new JTextArea(12,40); // 12 rows, 40 cols  
    bSaveAs = new JButton("Save As");  
    fileField = new JTextField("C:\\\\Document.txt");  
    bOk = new JButton("OK");  
    bExit = new JButton("Exit");  
  
    add(textArea, BorderLayout.CENTER);  
  
    // Set up buttons and textfield in bottom panel.  
    JPanel bottomPanel = new JPanel();  
    bottomPanel.setLayout(new GridLayout(2,1));  
  
}
```

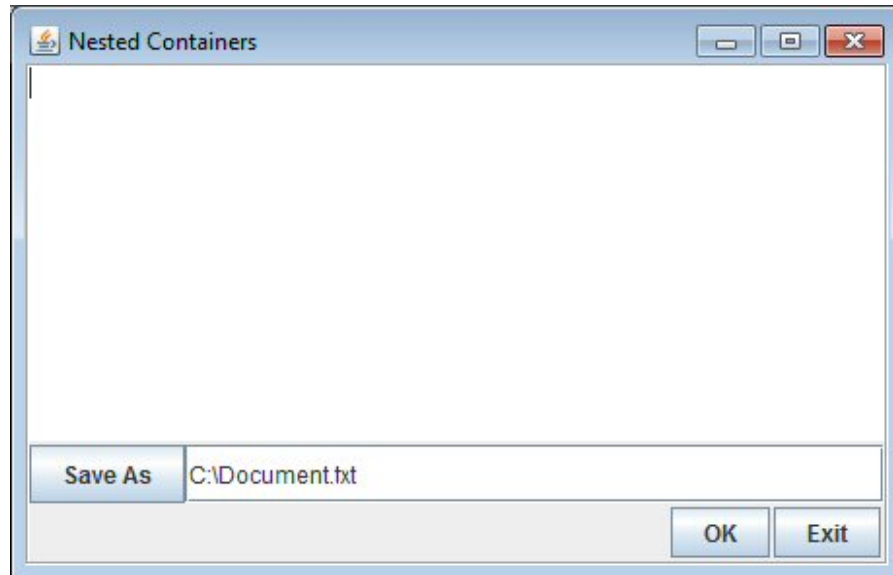
35

Nested Containers, Example

```
JPanel subPanel1 = new JPanel();  
JPanel subPanel2 = new JPanel();  
subPanel1.setLayout(new BorderLayout());  
subPanel2.setLayout  
    (new FlowLayout(FlowLayout.RIGHT, 2, 2));  
  
subPanel1.add(bSaveAs, BorderLayout.WEST);  
subPanel1.add(fileField, BorderLayout.CENTER);  
subPanel2.add(bOk);  
subPanel2.add(bExit);  
  
bottomPanel.add(subPanel1);  
bottomPanel.add(subPanel2);  
  
add(bottomPanel, BorderLayout.SOUTH);  
}
```

36

Nested Containers: Result



37

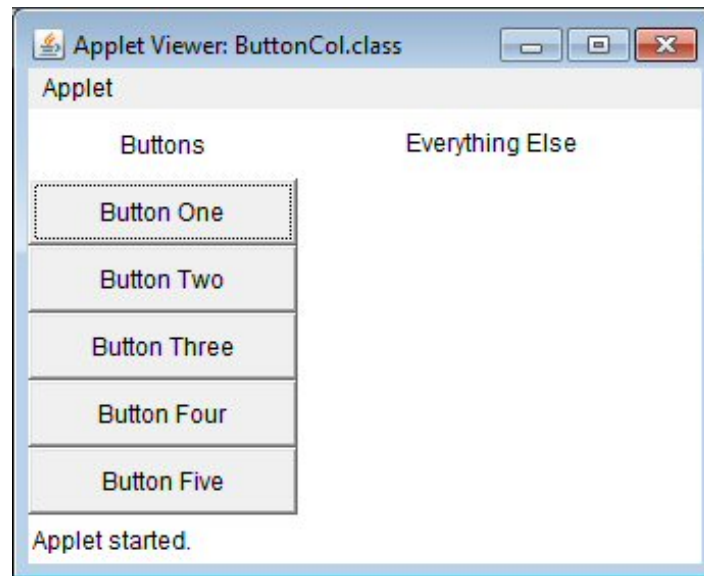
Turning Off Layout Manager for Some Containers: Example

- Suppose that you wanted to arrange a column of buttons (on the left) that take **exactly 40%** of the width of the container

```
setLayout(null);
int width1 = getSize().width*4/10;,
int height = getSize().height;
Panel buttonPanel = new Panel();
buttonPanel.setBounds(0, 0, width1, height);
buttonPanel.setLayout(new GridLayout(6, 1));
buttonPanel.add(new Label("Buttons", Label.CENTER));
buttonPanel.add(new Button("Button One"));
...
buttonPanel.add(new Button("Button Five"));
add(buttonPanel);
Panel everythingElse = new Panel();
int width2 = getSize().width - width1,
everythingElse.setBounds(width1+1, 0, width2, height);
```

38

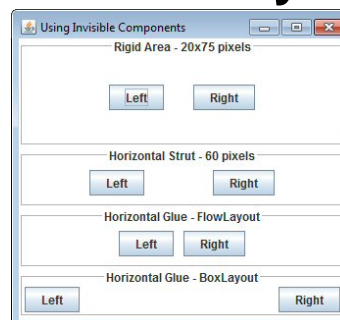
Turning Off Layout Manager for Some Containers: Result



39

Adjusting Space Around Components

- **Change the space allocated by the layout manager**
 - Most LayoutManagers accept a horizontal spacing (hGap) and vertical spacing (vGap) argument
 - For GridBagLayout, change the insets
- **Use invisible components to add space. BorderLayout supports this directly.**
 - See InvisibleComponentTest



40



Wrap-Up

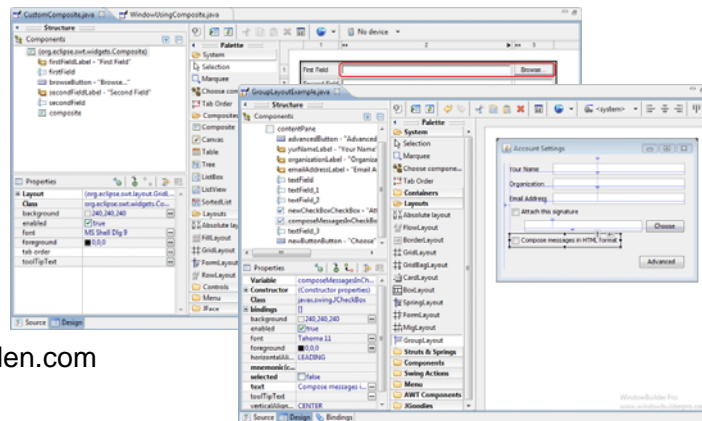


Customized Java EE Training: <http://courses.coreservlets.com/>

Java 7, Java 8, JSF 2.2, PrimeFaces, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Drag-and-Drop Swing GUI Builders

- **Free**
 - Matisse (“NetBeans GUI Builder”) built into NetBeans
 - Also available in MyEclipse. *Not in regular Eclipse.*
 - WindowBuilder Pro
 - Originally a commercial product, then bought and released for free by Google. *For Eclipse.*
 - <https://developers.google.com/java-dev-tools/download-wbpro>
- **Commercial**
 - JFormDesigner
 - jformdesigner.com
 - Jvider
 - jvider.com
 - SpeedJG
 - wsoftware.de
 - Jigloo
 - <http://www.cloudgarden.com/jigloo/>



Other Layout Managers

- **BoxLayout**
 - Lets you put components in horizontal or vertical rows and control the sizes and gaps. Simple, but useful.
- **GroupLayout**
 - Groups components into hierarchies, then positions each group. Mostly designed for use by GUI builders.
- **SpringLayout**
 - Alternative to GridBagLayout that lets you give complex constraints for each component. Almost exclusively designed for use by GUI builders.
- **Details and visual summaries**
 - <http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>

43

Summary

- **Default layout managers**
 - Applet and Panel: FlowLayout
 - Frame and Dialog: BorderLayout
- **Preferred sizes**
 - FlowLayout: honors all
 - BorderLayout:
 - North/South honors preferred height
 - East/West honors preferred width
 - GridLayout: ignores preferred sizes
- **GridBagLayout**
 - The most complicated but most flexible manager
- **Design strategy**
 - Use nested containers, each with relatively simple layout

44



Questions?

More info:

<http://courses.coreservlets.com/Course-Materials/java.html> - General Java programming tutorial

<http://www.coreservlets.com/java-8-tutorial/> - Java 8 tutorial

<http://courses.coreservlets.com/java-training.html> - Customized Java training courses, at public venues or onsite at *your* organization

<http://coreservlets.com/> - JSF 2, PrimeFaces, Java 7 or 8, Ajax, jQuery, Hadoop, RESTful Web Services, Android, HTML5, Spring, Hibernate, Servlets, JSP, GWT, and other Java EE training



Customized Java EE Training: <http://courses.coreservlets.com/>

Java 7, Java 8, JSF 2.2, PrimeFaces, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.