Lecture 02
Creating Graphical User Interface
Part 5

Book: Chapter 12: Layout Managers

Contents

- Layout Manager Theory
- Some Pre-defined Layout in Java API
- A demonstration
- Other layout options

Layout Manager Theory

- Layout manager: an object that implements the java.awt.LayoutManager interface and determines the size and position of the components within a container.
- Although components can provide size and alignment hints, a container's layout manager has the final say on the size and position of the components within the container
- Every container has a default layout manager: Frame (BorderLayout), Panel (FlowLayout).

Layout Manager Theory

- The AWT Toolkit (java.awt package) includes 5 main layout manager classes:
  - Flow Layout
  - Grid Layout
  - Border Layout
  - Card Layout
  - Gridbag Layout

The FlowLayout Class

- Arranges components in horizontal rows (left to right in the order they were added to their container), starting new rows if necessary
- Fits as many components as possible into the top row and spills the remainder into a second row
- Leaves a gap of five pixels between components in both horizontal and vertical direction by default
- To change the default gap, the constructor should be called
- Is the default layout manager for any JPanel

The FlowLayout Class

Constructors

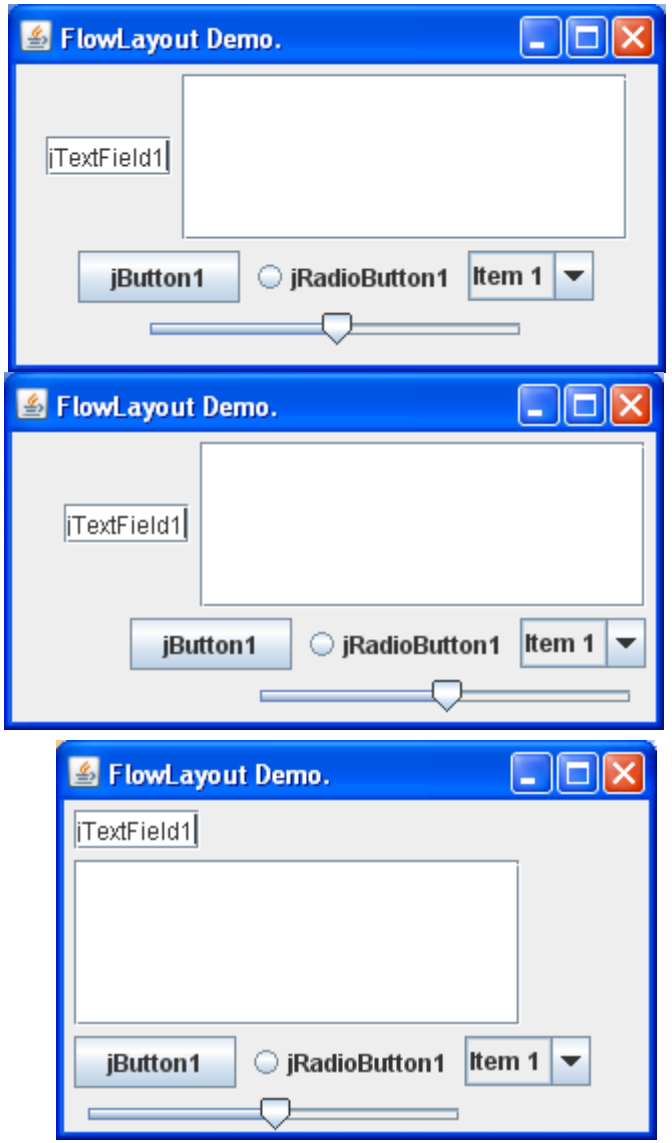
FlowLayout() // align= center, vgap=hgap=5 units.

FlowLayout(int align) // vgap=hgap=5 units

FlowLayout(int align, int hgap, int vgap)

Constants for alignment:

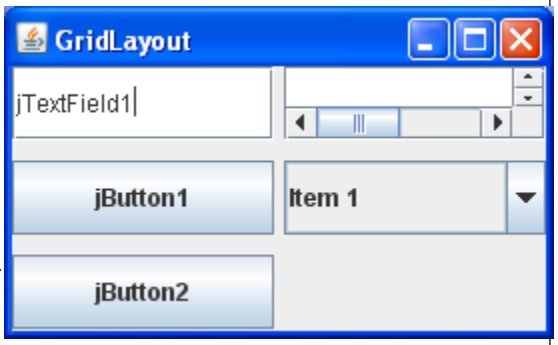
LEFT CENTER RIGHT LEADING TRAILING



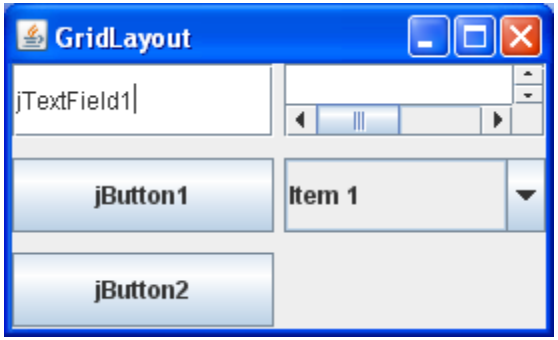
Instructor guide student exploring this layout (Use step-by-step manner)

The GridLayout Class

- Subdivides its territory into a matrix of rows and columns.
  - The number of rows and number of columns are specified as parameters to the manager's constructor.
  - When it performs a layout in a given space, it ignores a component's preferred size
  - Number of rows can not be changed but the number of columns is automatically updated (Java 1.6).



The GridLayout Class



Constructor

GridLayout() // 1x1

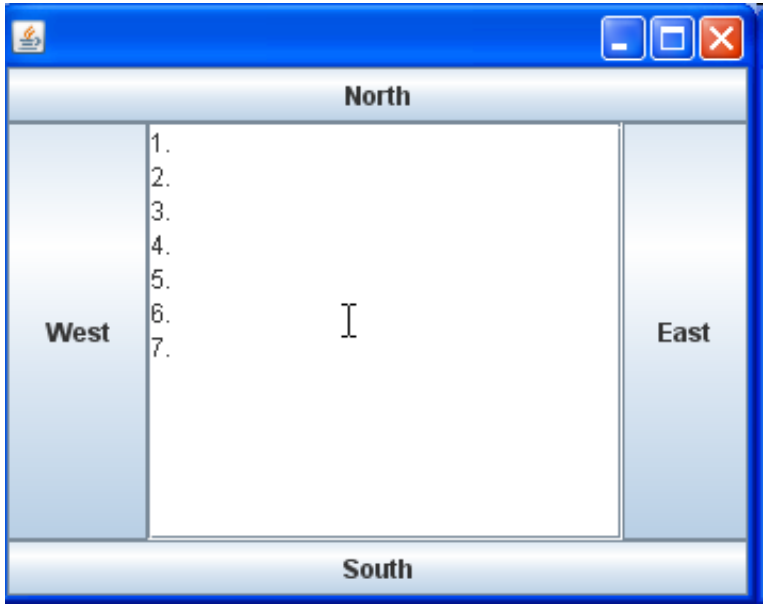
GridLayout(int rows, int cols)

GridLayout(int rows, int cols,int hgap,int vgap)

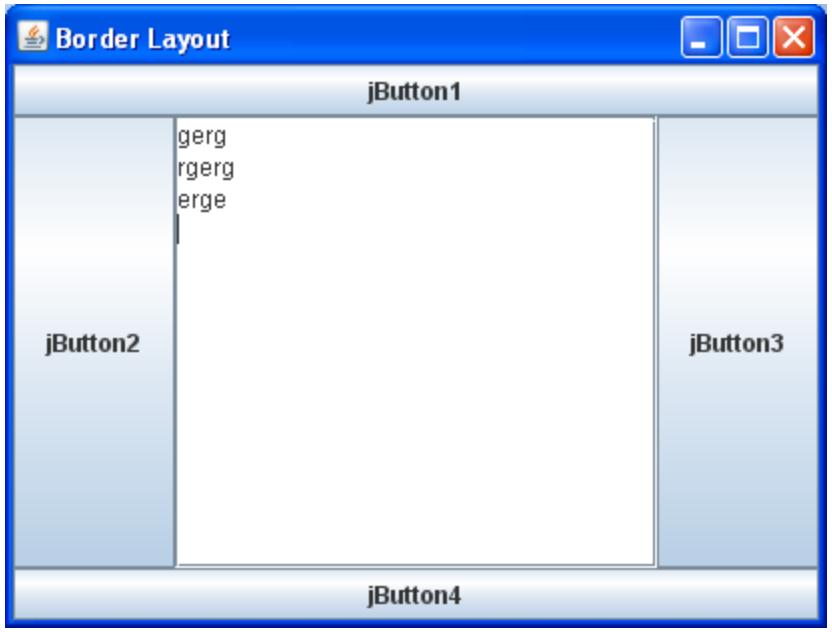
Instructor guide student exploring this layout
(Use step-by-step manner)

The BorderLayout Class

- Default layout of a frame.
- Divides its territory into five regions:
  - North (useful for toolbar)
  - South (useful for status bar)
  - East, West, Center
- Each region may be empty or may contain one component.



The BorderLayout Class



- After a component is added to the North and South regions, The height which the component occupied, can not be vertically resized.
- Similarly for the East and West regions.

Instructor guide student exploring this layout (Use step-by-step manner)

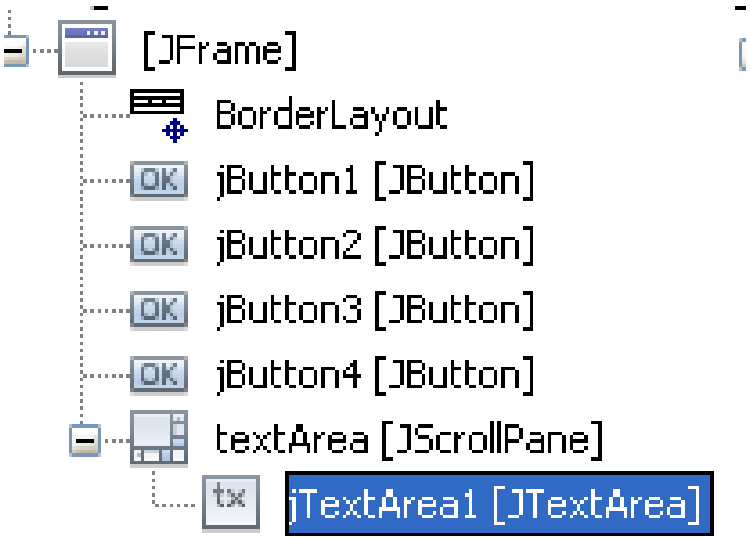
The BorderLayout Class

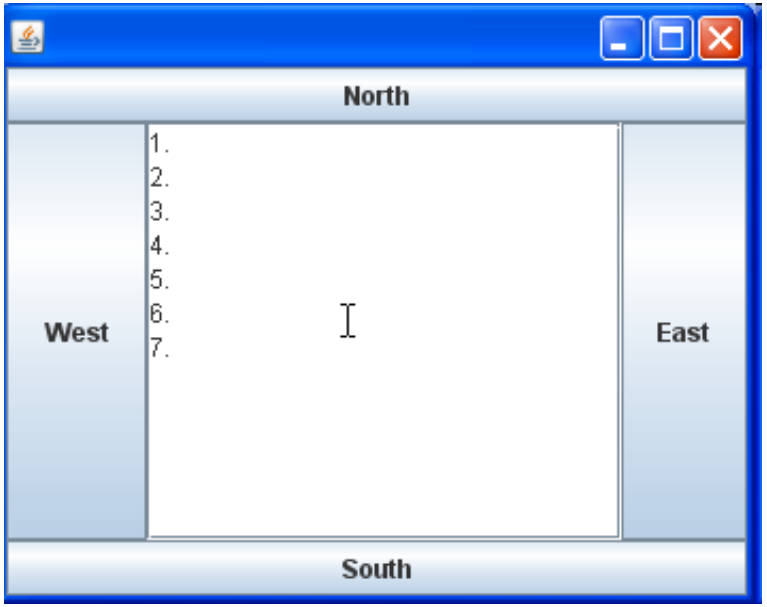
- The default region is CENTER region.
- If there is only one component at the center, this component will cover all the container.

- **Constructors:**
  BorderLayout()
  BorderLayout (int hgap, int vgap)
- **Add a component to the container:**

```
Container.add("East", componentName); // or
Container.add(componentName ,"East"); // or
Container.add(BorderLayout.EAST, componentName); // or
Container.add(componentName , BorderLayout.EAST);
Container.add(componentName); // add to center
```

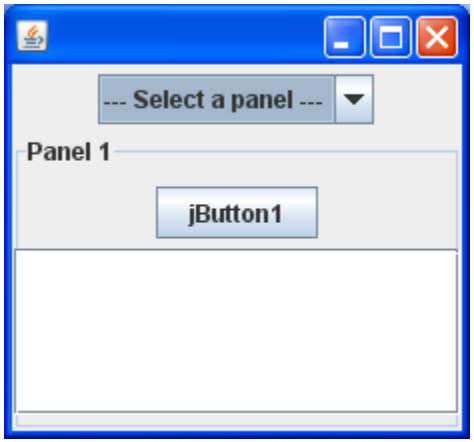
BorderLayout...

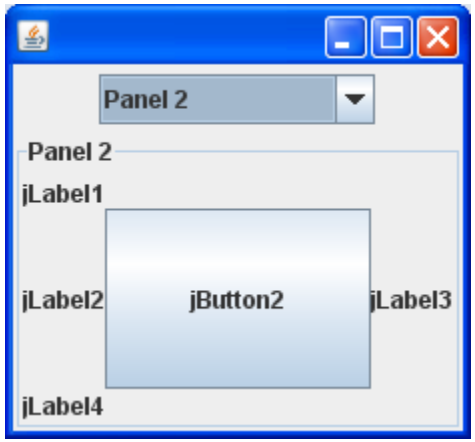


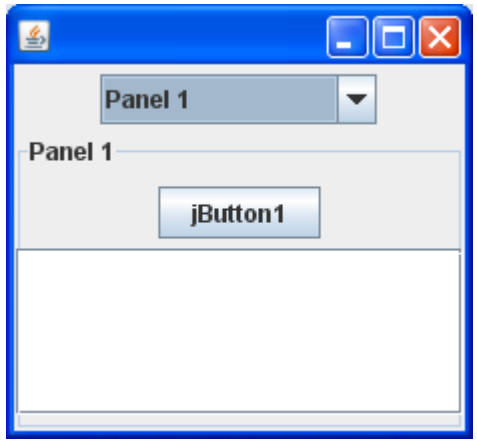



```
jButton1.setText("North");
getContentPane().add(jButton1, java.awt.BorderLayout.PAGE_START);

jButton2.setText("South");
getContentPane().add(jButton2, java.awt.BorderLayout.PAGE_END);

jButton3.setText("West");
getContentPane().add(jButton3, java.awt.BorderLayout.LINE_START);

jButton4.setText("East");
getContentPane().add(jButton4, java.awt.BorderLayout.LINE_END);
```

The CardLayout Class





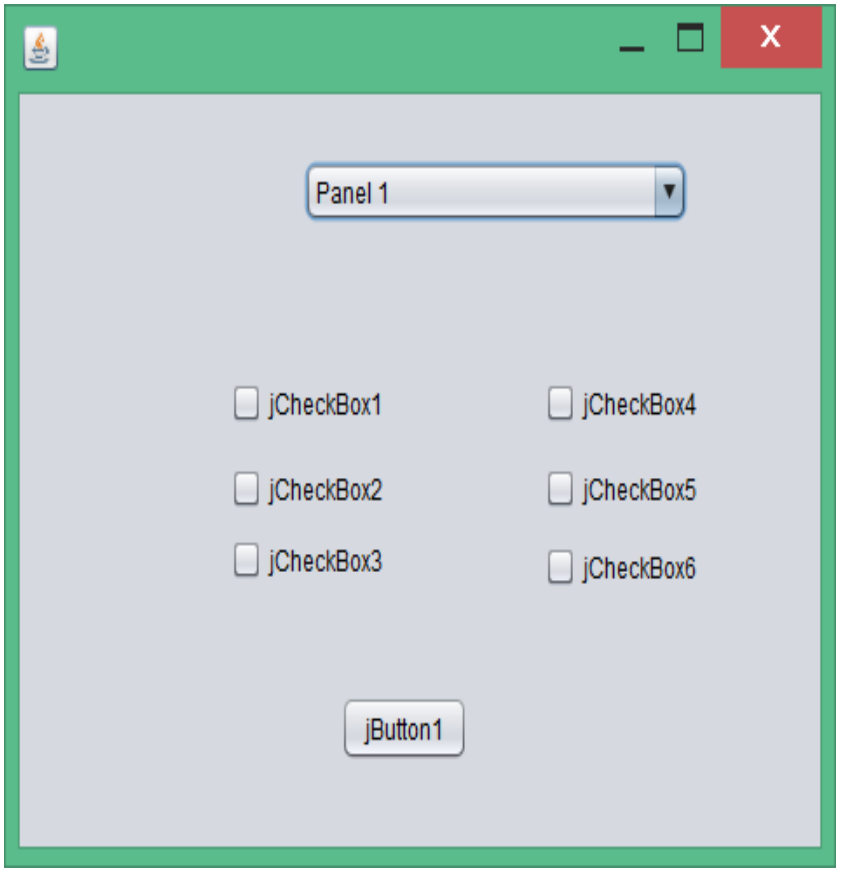


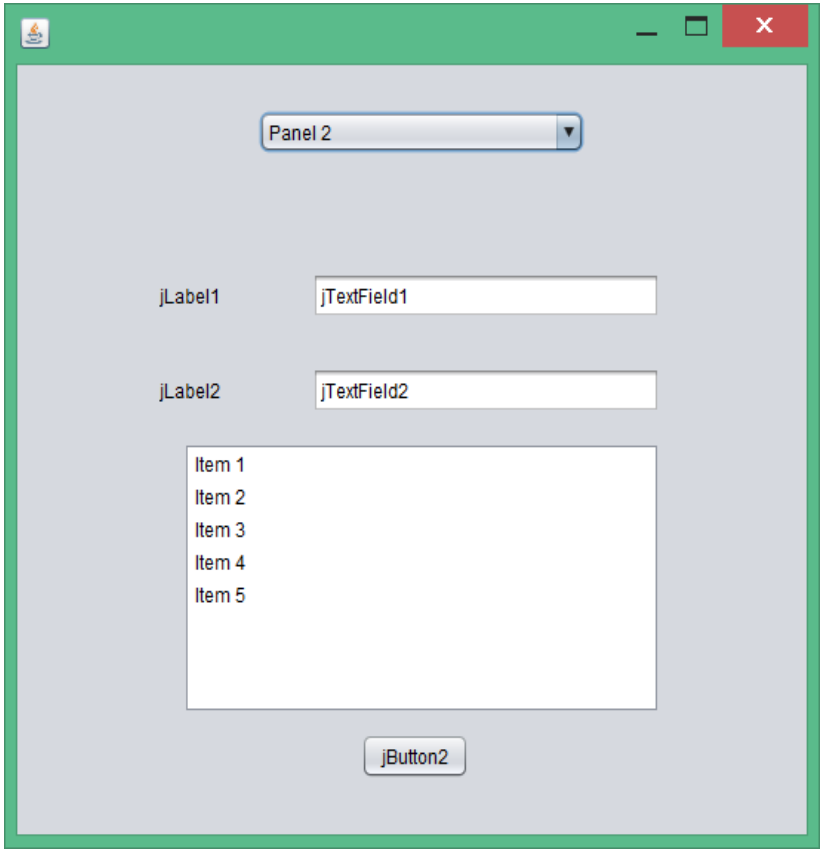
- The CardLayout class helps you manage two or more components (usually JPanel instances) that share the same display space.
- When using CardLayout, you need to provide a way to let the user choose between the components. The above demonstration uses a combo box for this purpose.
- An easier but less flexible way to accomplish the same task is to use a tabbed pane.

CardLayout…

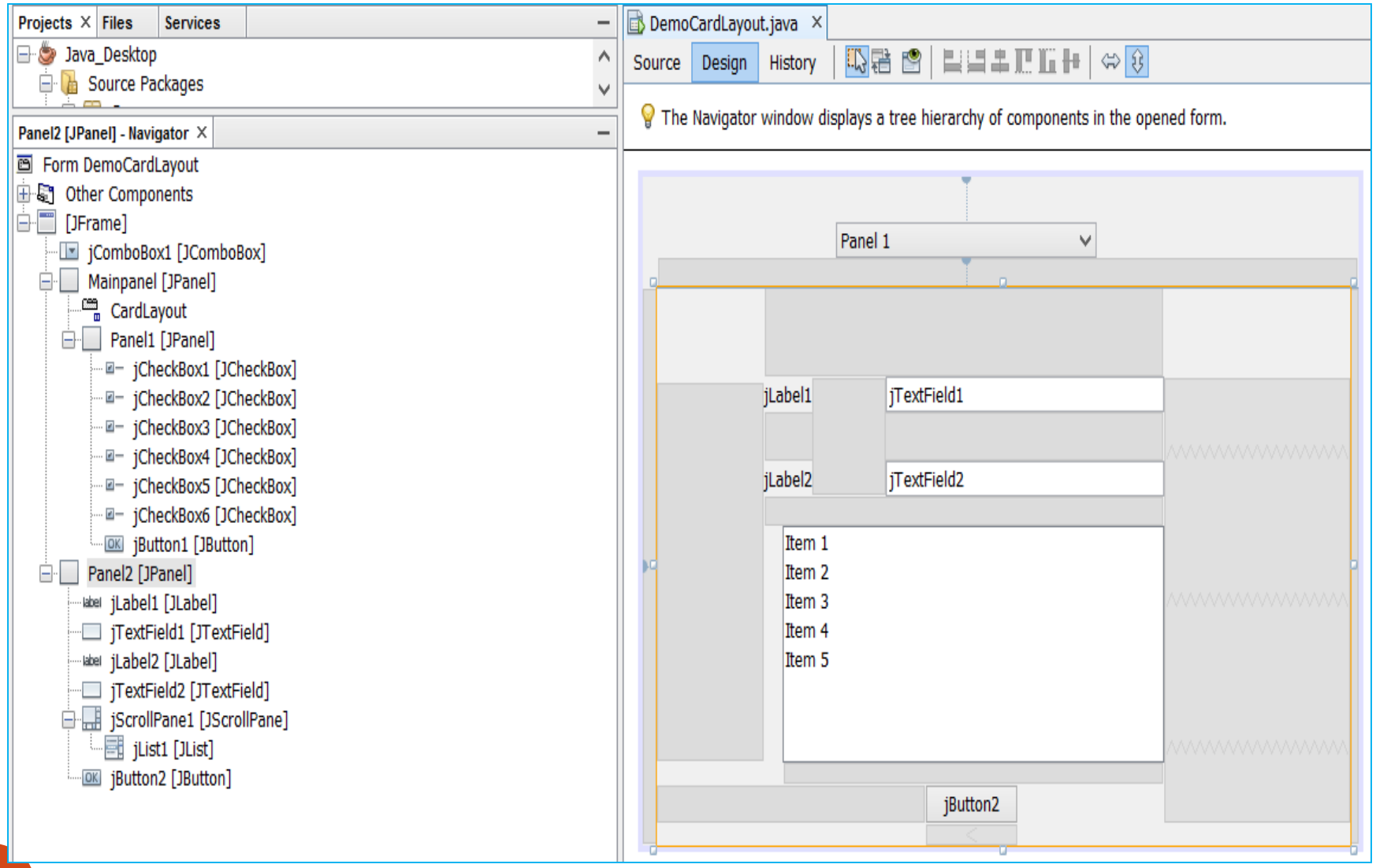
- Main panel contains a card layout.
- Each sub-panel has a name that is identified by the card layout.
- At a time, only one sub-panel is seen by user (current panel).
- Show a sub-panel:
  - **Card.first(mainPanel);**
  - **Card.next(mainPanel);**
  - **Card.previous(mainPanel);**
  - **Card.last(mainPanel);**
  - **Card.show(mainPanel, subPanelName);**

Demo 10: CardLayout





Demo 10: CardLayout



Demo 10: CardLayout

```
private void jComboBox1ActionPerformed(java.awt.event.ActionEvent evt) {
 int pos =jComboBox1.getSelectedIndex();

    if(pos==0){
        Mainpanel.removeAll();
        Mainpanel.add(this.Panel1);
        Mainpanel.repaint();
        Mainpanel.revalidate();
    }

    else{

        Mainpanel.removeAll();
        Mainpanel.add(this.Panel2);
        Mainpanel.repaint();
        Mainpanel.revalidate();
    }


}
```

The GridBagLayout Class

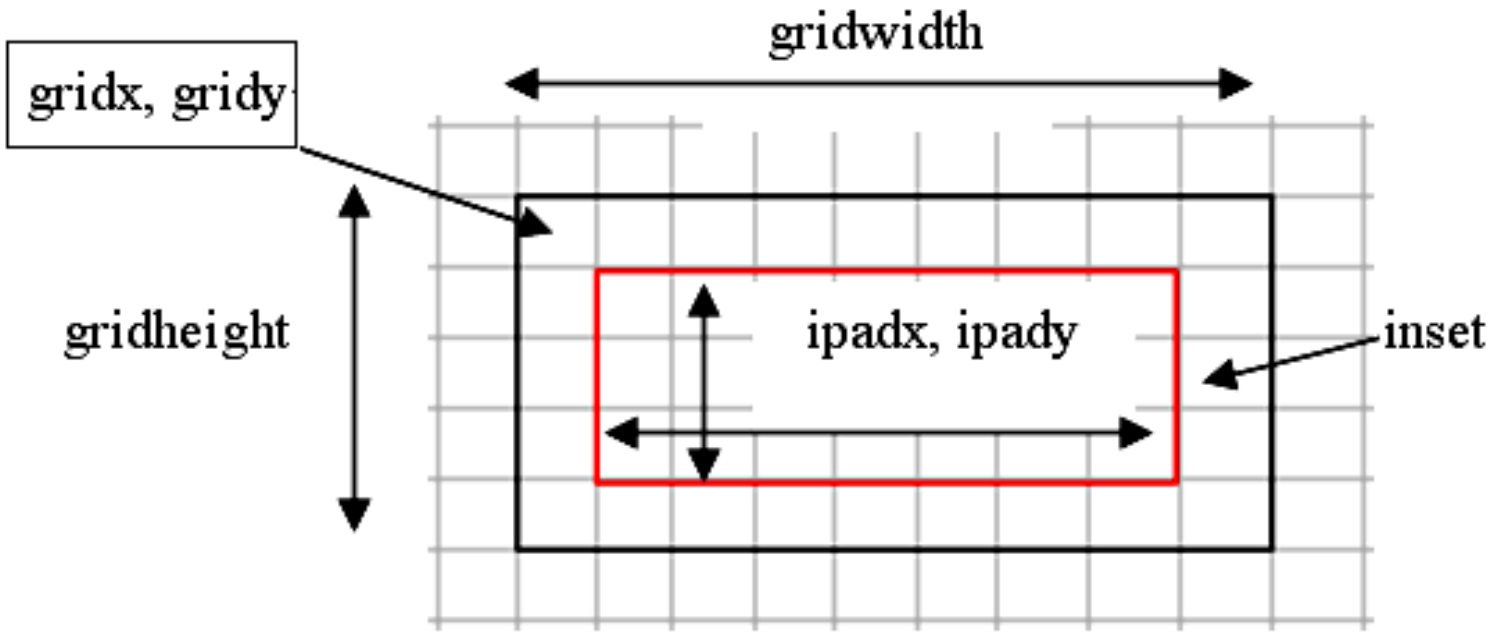
- Is the most powerful layout manager.
- It can perform the work of the Flow, Grid, and Border layout managers if appropriately programmed and is capable of much more, often without the need for nesting multiple panels as is so often required with the other layout managers
- Divides its container into an array of cells:
  - Different cell rows can have different heights,
  - And different cell columns can have different widths.
- Requires a lot of information to know where to put a component. A helper class called **GridBagConstraints** is used to hold all the layout position information.
- To add the component, the add(Component, Object) is used

GridBagLayout …

- Designing a Layout with GridBag
  - Three levels of control are applied to a GridBag layout to make up the final layout in the container.
    - The sizes of the various rows and columns, along with the way they stretch when the container is resized.
    - The cell (or cells) that provides the target space for each component is determined.
    - The final control determines how each component is stretched to fit or, if it isn't, how the component is positioned within the target space.

GridBagLayout …

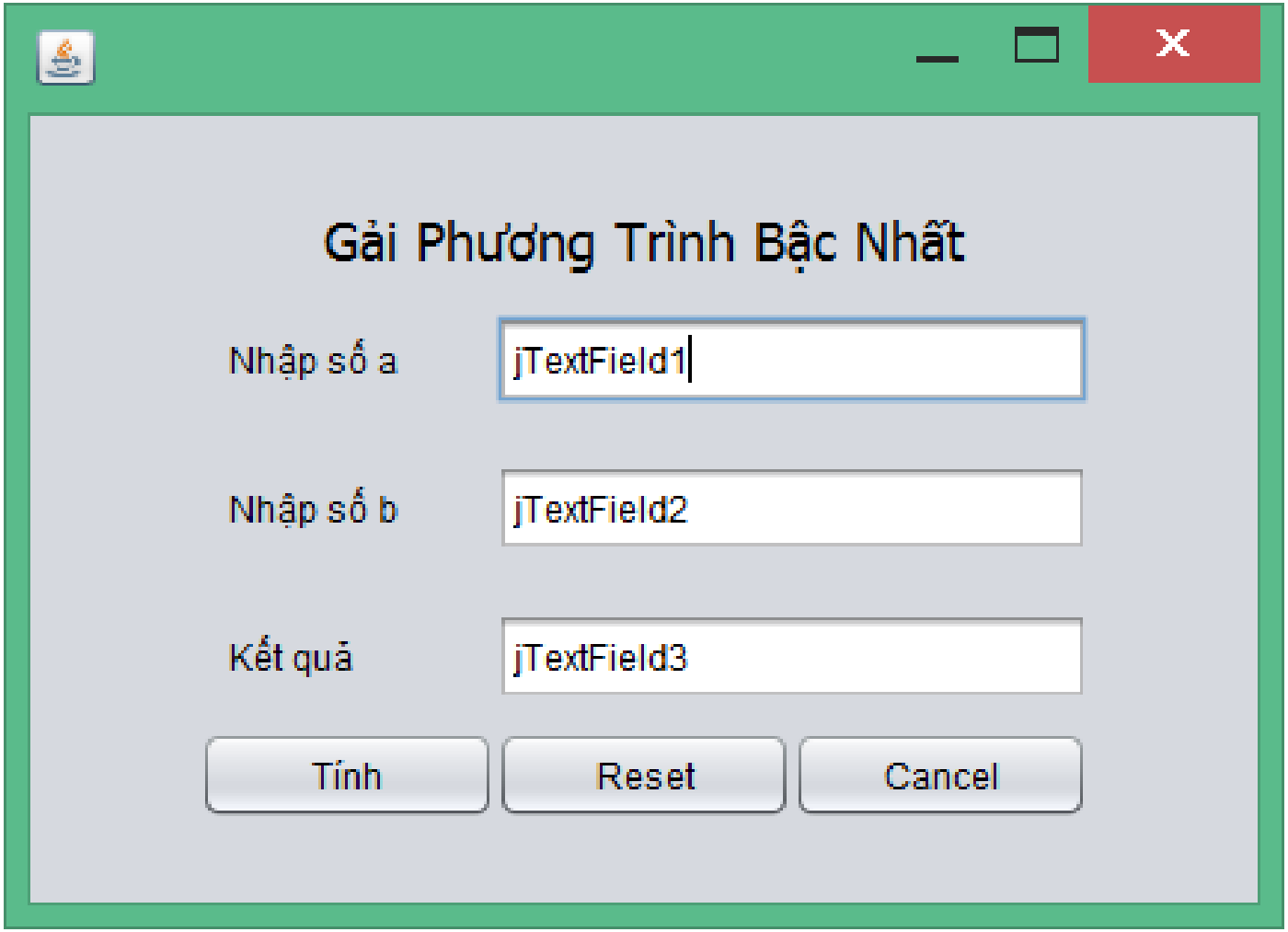
- Parameters are used to draw a component in a GridBag Layout:



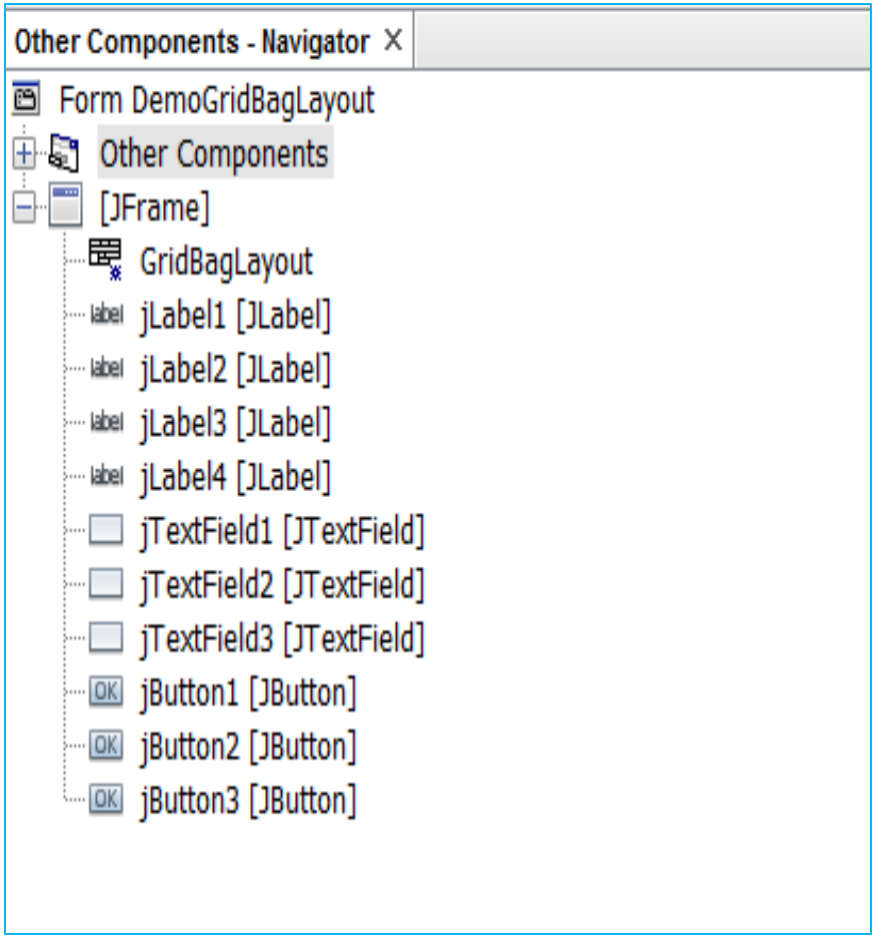
GridBagLayout …

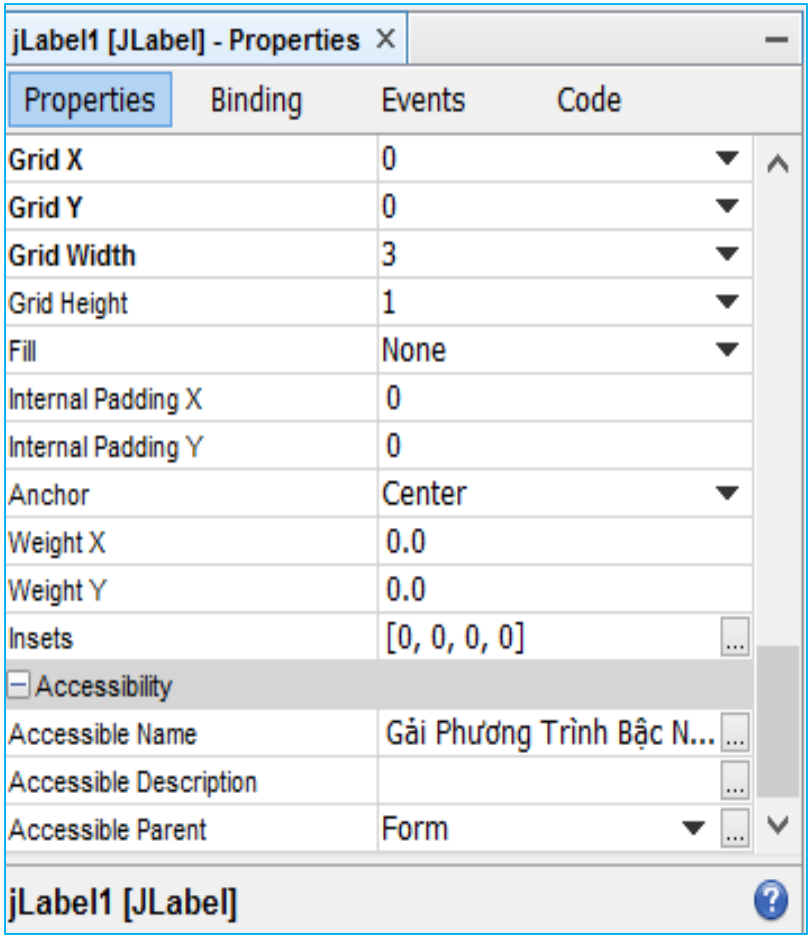
- **Controlling the Rows and Columns**
  - **gridx**, **gridy**: Specify the row and column at the upper left of the component .
  - **gridwidth**, **gridheight** : Specify the number of columns (for gridwidth) or rows (for gridheight) in the component's display area.
  - **Fill** :Used when the component's display area is larger than the component's requested size to determine whether and how to resize the component.
  - **ipadx**, **ipady**: How much to add to the minimum size of the component. The default value is zero. The width of the component will be at least its minimum width plus ipadx*2 pixels.
  - **insets** :The minimum amount of space between the component & the edges of its display area.
  - **Anchor:**Used when the component is smaller than its display area to determine where (within the area) to place the component.
  - **weightx**, **weighty** : Weights are used to determine how to distribute space among columns (weightx) and among rows (weighty); this is important for specifying resizing behavior

GridBagLayout …



GridBagLayout …





Other Layout Options

- If you are in a situation where Flow, Grid, Border, Card, and GridBag will not create the layout you need, your choices are:
  - To find a layout manager from another source
  - To create your own layout manager
  - To use no layout manager
    - Use setLayout(null) method of the container to set no layout manager.
    - No layout honors each component's x, y, width, and height values. Thus, you can call setBounds() on a component, add() it to a container that has no layout manager, and have the component end up where you expect it to be.
    - Disadvantages: Have to write code to detect when the container resizes, and more code to do the right thing when resizing occurs.

Summary

- Layout Manager Theory
- Some Pre-defined Layout in Java API
- A demonstration
- Other layout options

Thank You