# Frames, Panels, and Layout Managers

Terminology, content panes, panels, layout managers: BorderLayout, FlowLayout, CardLayout, BoxLayout, GridLayout, organizing with panels

## Containers and windows

- **Containers** are designed to hold other components.
  - Container is a concrete class that extends Component.
- **Windows** are top-level containers and provide the GUI interface to the window manager of the OS.
- **JFrames** are top-level containers. They typically have the three window buttons (close, minify, maximize), borders, and title bars.
  - JFrame is the Swing class designed for creating ordinary framed windows in GUI applications.
- **JDialogs** are secondary windows.

## Creating a simple GUI

This code is the start of a GUI framework.

```
import javax.swing.JFrame;

public class SimpleFrame extends JFrame {

   public SimpleFrame() {
      this.setTitle("Simple GUI");
      this.setSize(400, 200);
      this.setLocation(50, 50);
      this.setDefaultCloseOperation(EXIT_ON_CLOSE);
      this.setVisible(true);
   }
   public static void main(String[] args) {
   new SimpleFrame();
   }
}
```

# Simple GUI discussion

- `public class SimpleFrame extends Jframe`

  SimpleFrame inherits all of the members of JFrame.
- `this.setTitle("Simple GUI");`

  puts a title on the window
- `this.setSize(400, 200);`

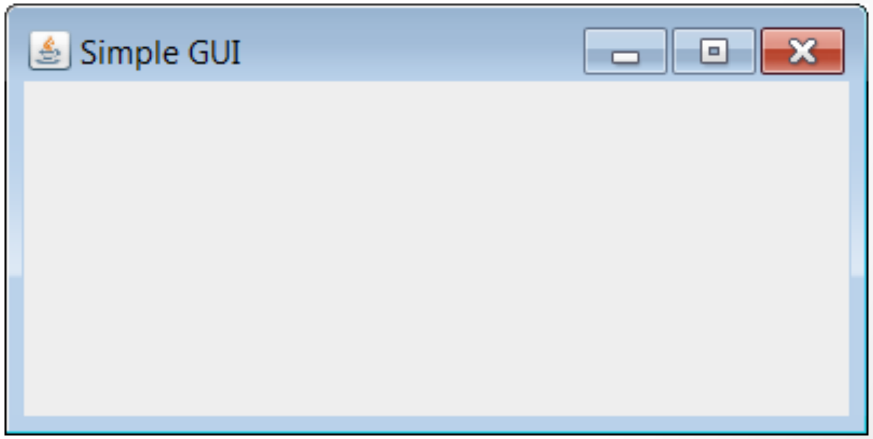
  sets the window size to 400X200 pixels
- `this.setLocation(50, 50);`

  places the window at location (50,50) from top left corner
- `this.setDefaultCloseOperation(EXIT_ON_CLOSE);`

  when the window closes,

  the app exits
- `this.setVisible(true);`

  makes the window appear



## Adding to content panes

- **Content panes** are lower level containers. Most are embedded in primary windows, and are used to organize the layout/structure of the primary window.
- Examples of content panes are **JPanel**, **JOptionPane**, **JTabbedPane**, **JSplitPane** etc
- Each JFrame has a content pane, which you can access via the JFrame method getContentPane().
- To put a new panel or other widget such as a button to a JFrame, *add* it to its content pane.

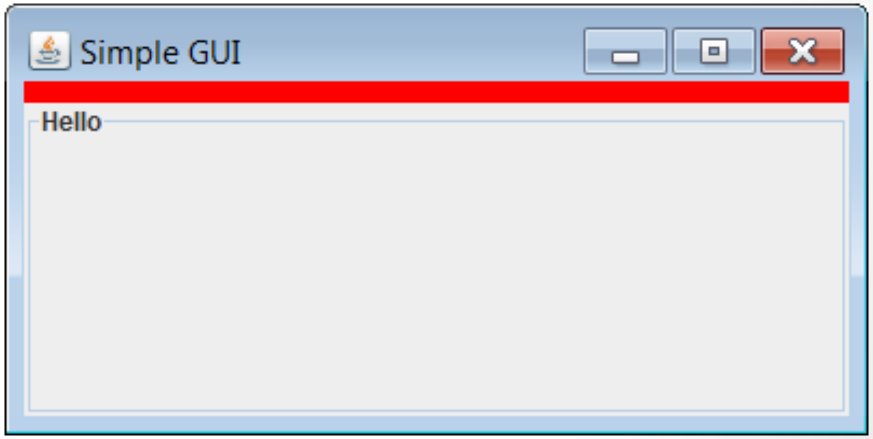
# Enhancing with panels

Add a red panel at the top, titled panel below.

```
public class SimpleFrameWithPanes extends JFrame {

   public SimpleFrameWithPanes() {
      // Set title. Size, location
      this.setVisible(true);

      Container c = this.getContentPane();
      JPanel pnlOne = new JPanel();
      JPanel pnlTwo = new JPanel();
      pnlOne.setBackground(Color.RED);
      pnlTwo.setBorder(new TitledBorder("Hello"));
      c.add(pnlOne, BorderLayout.NORTH);
      c.add(pnlTwo, BorderLayout.CENTER);
      }
   public static void main(String[] args) { // …
```

# Panel enhancement discussion

- `Container c = this.getContentPane();`
  Gets the content pane for this JFrame
- `JPanel pnlOne = new JPanel(); JPanel pnlTwo = new JPanel();`
  Creates two JPanel objects
- `pnlOne.setBackground(Color.RED);`
  Background of pnlOne will be red.
- `pnlTwo.setBorder(new TitledBorder("Hello"));`
  Puts a border with embedded title around pnlTwo
- `c.add(pnlOne, BorderLayout.NORTH);`
  Puts pnlOne on top
- `c.add(pnlTwo, BorderLayout.CENTER);`
  Puts pnlTwo on bottom

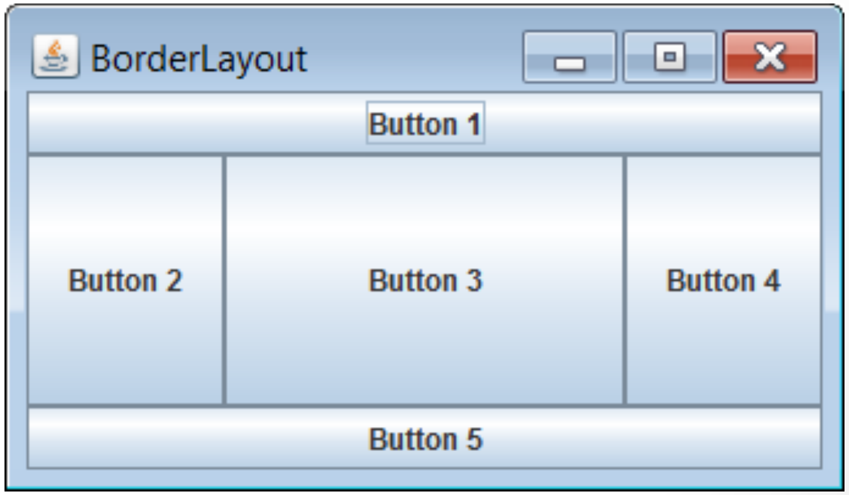


# Layout managers

- **Layout managers** determine how components are arranged on windows:
  - **BorderLayout**. Components are north, south, east, west, or center.
  - **FlowLayout**. Components are laid out according to when they are added: left to right, starting at the top and flowing to the next level as needed.
  - **BoxLayout**. Components are laid on a horizontal or vertical line.
  - **CardLayout**. Components are laid on different "cards." Exactly one card is visible at a time.
  - **GridLayout**. Components are laid in a grid of equal sized cells.
  - **GridBagLayout**. Components are laid in a grid of variable sized cells.

## BorderLayout

- Default layout manager for JFrame.
- Components are at BorderLayout.NORTH, BorderLayout.SOUTH, BorderLayout.EAST, BorderLayout.WEST, BorderLayout.CENTER;
- Components fill the available space.
- All components except at CENTER take the minimum required space (except for fill restriction).

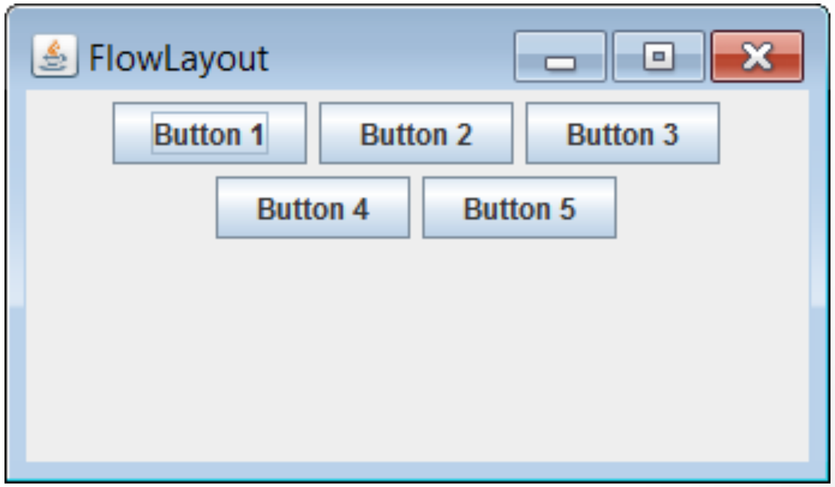


## BorderLayout sample code

```
public class BorderLayoutDemo extends JFrame{

  private JButton btn1 = new JButton("Button 1");
  private JButton btn2 = new JButton("Button 2");
  private JButton btn3 = new JButton("Button 3");
  private JButton btn4 = new JButton("Button 4");
  private JButton btn5 = new JButton("Button 5");

  public BorderLayoutDemo() {
     this.setSize(344,200);
     Container c = getContentPane();
     c.add(btn1, BorderLayout.NORTH);
     c.add(btn2, BorderLayout.WEST);
     c.add(btn3, BorderLayout.CENTER);
     c.add(btn4, BorderLayout.EAST);
     c.add(btn5, BorderLayout.SOUTH);
     setTitle("BorderLayout");
     setVisible(true);
  }
```

# FlowLayout

- FlowLayout is the simplest layout manager, with components appearing in order of when they were added.
- Components in a FlowLayout take the minimum amount of space required.
- You need to set the layout manager for a JFrame (or its content pane) in order for it to have any layout except BorderLayout.

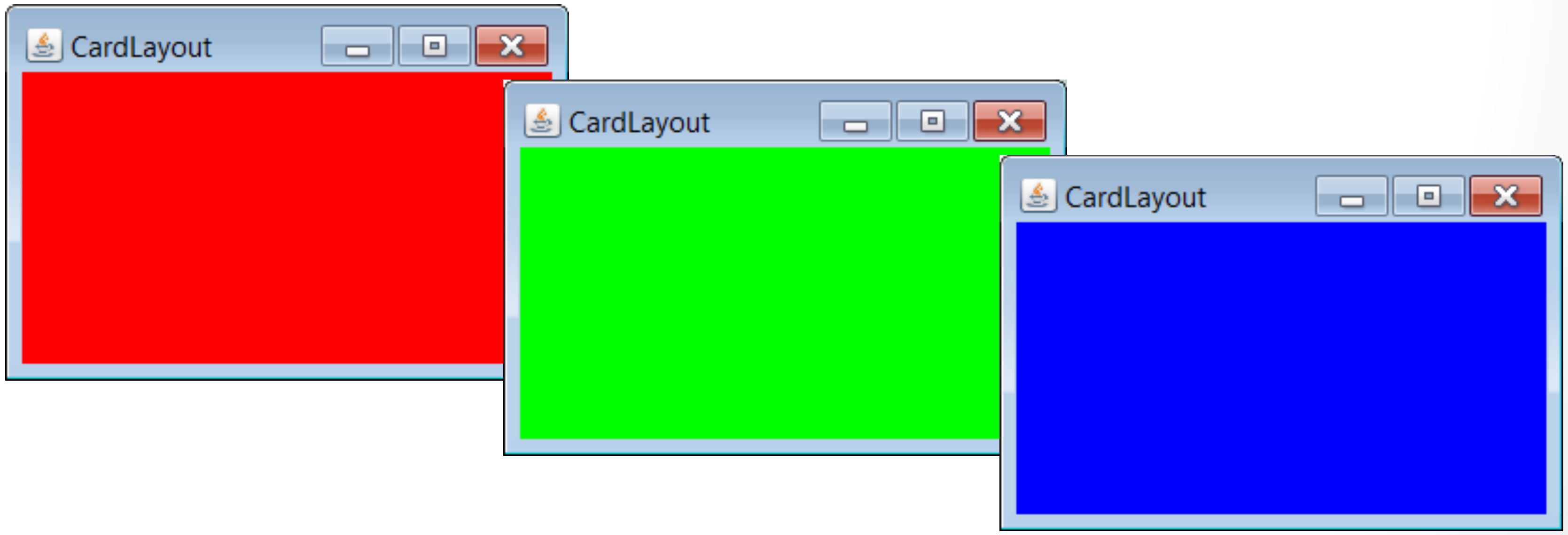


## FlowLayout sample code

```
public class FlowLayoutDemo extends JFrame{
   private JButton btn1 = new JButton("Button 1");
   private JButton btn2 = new JButton("Button 2");
   private JButton btn3 = new JButton("Button 3");
   private JButton btn4 = new JButton("Button 4");
   private JButton btn5 = new JButton("Button 5");

   public FlowLayoutDemo() {
      this.setLayout(new FlowLayout());
      this.setSize(344,200);
      Container c = getContentPane();
      c.add(btn1);
      c.add(btn2);
      c.add(btn3);
      c.add(btn4);
      c.add(btn5);
      setTitle("FlowLayout");
      setVisible(true);
   }
```

# CardLayout

- CardLayout is appropriate when you want to display only one high-level component at a time.
- Switching cards requires listeners, tabbed panes, panels, or other mechanisms.
- The images below are for a card layout in which each card has a panel, and each panel has a different background color.



# CardLayout sample code

```
Color[] clrArray = {Color.RED, Color.GREEN, Color.BLUE};
JPanel[] pnlArray = new JPanel[3];

public CardLayoutDemo() {
    this.setSize(300,200);
    Container c = getContentPane();
    CardLayout m = new CardLayout();
    c.setLayout(m);
    for (int i = 0; i < 3; i++) {
        pnlArray[i] = new JPanel();
        pnlArray[i].setBackground(clrArray[i]);
        c.add(pnlArray[i],"");
    }
    // Continued next page
```

## CardLayout sample code (cont)

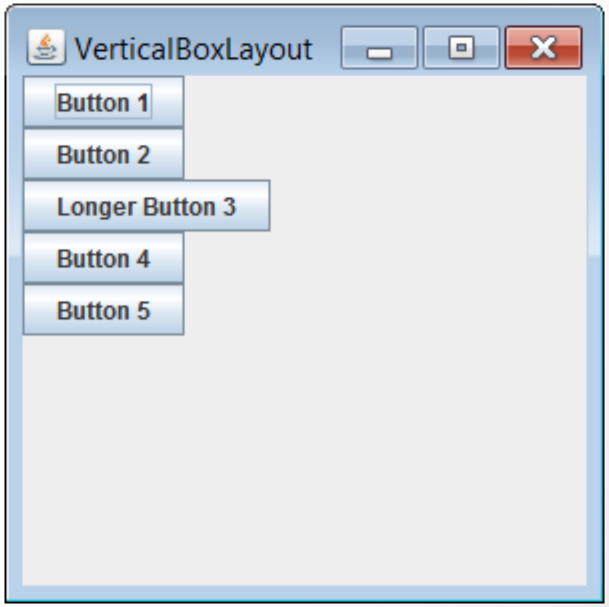
```
public CardLayoutDemo() {
   // Set size, layout, and add panels to c
   setTitle("CardLayout");
   setVisible(true);
   while (true) {
      try {
         Thread.sleep(1000);
         m.next(c);
      } catch(InterruptedException ex) {
         Thread.currentThread().interrupt();
      }
   //
```

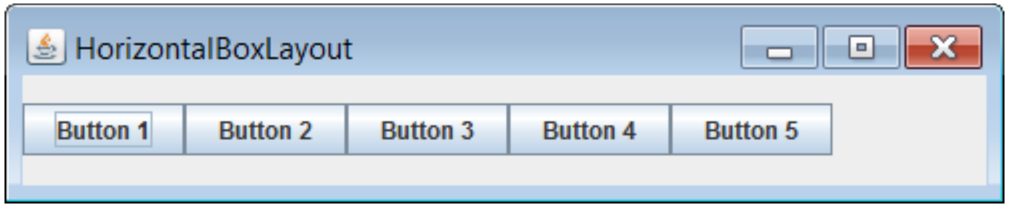
# BoxLayout

- BoxLayout stacks components on top of each other vertically or beside each other horizontally.
- Components take the minimum space required.
- Code:

```
Container c = getContentPane();
this.setLayout(new BoxLayout(c,BoxLayout.PAGE_AXIS));
```

- Components can be aligned left, right, center – but alignment is tricky.

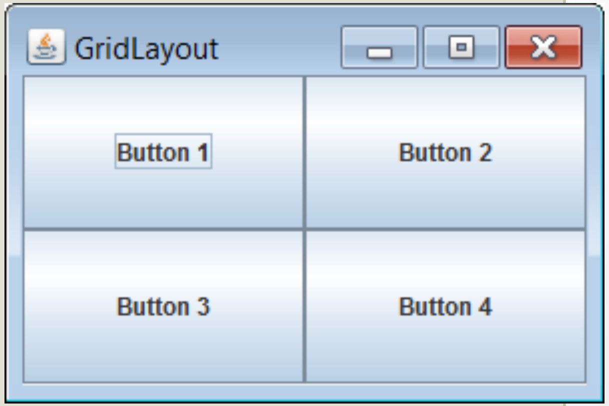




# GridLayout

- Components are on a rectangular grid.
- Components fill the grid cells.

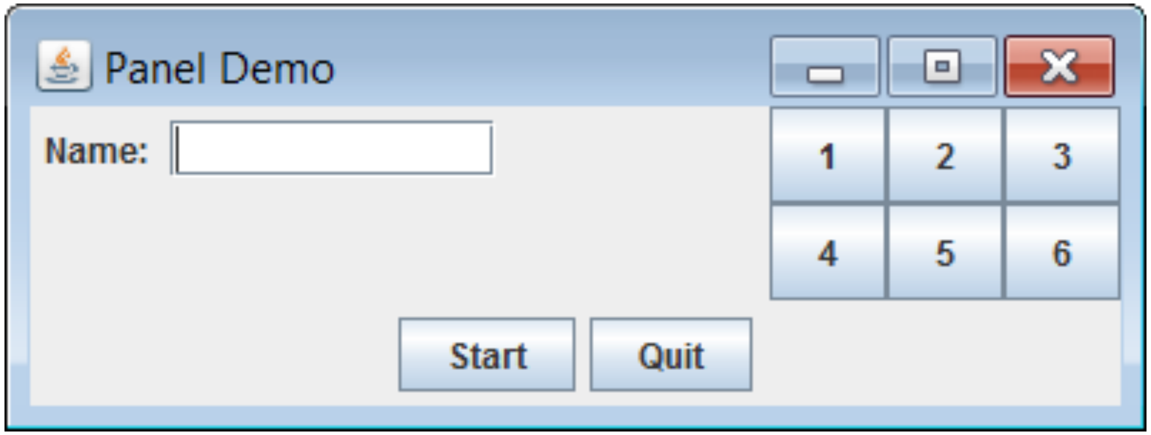
```
public class GridLayoutDemo extends JFrame {

  private JButton btn1 = new JButton("Button 1");
  private JButton btn2 = new JButton("Button 2");
  private JButton btn3 = new JButton("Button 3");
  private JButton btn4 = new JButton("Button 4");

  public GridLayoutDemo() {
    Container c = getContentPane();
    setSize(300,200);
    c.setLayout(new GridLayout(2,2));
    c.add(btn1);
    c.add(btn2);
    c.add(btn3);
    c.add(btn4);
    setTitle("GridLayout");
    setVisible(true);
  }
```



# Using JPanels for organization

- You can apply layout managers to JPanels.
- Different layout managers help customize displays.
- Default JPanel layout manager is FlowLayout.

The demo program below has 3 panels. Two use FlowLayout (the default for JPanels); one uses GridLayout.



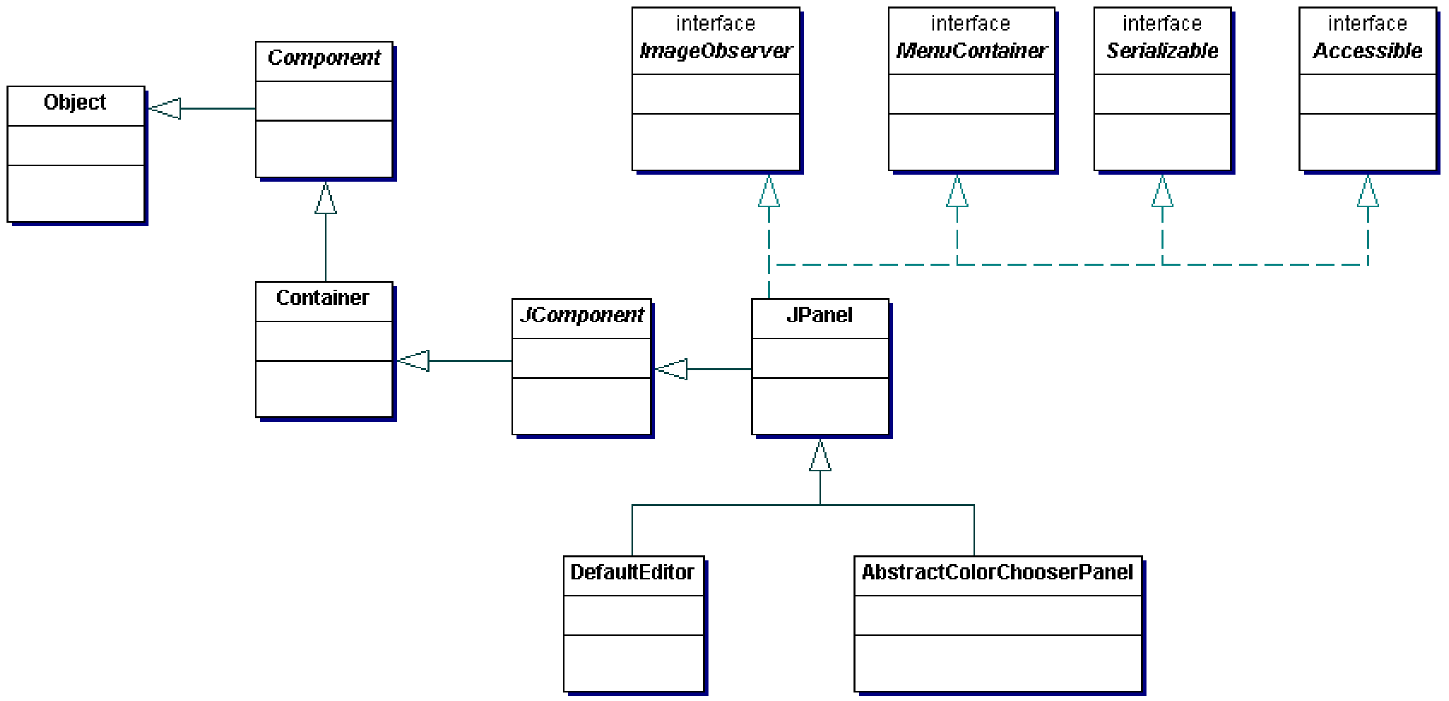
# JPanel sample code – part 1

```
private JPanel pnlText = new JPanel();
private JPanel pnlGrid = new JPanel();
private JPanel pnlButtons = new JPanel();

public PanelDemo() {
   Container c = getContentPane();
   setSize(600,400);

   pnlText.add(new JLabel("Name: "));
   pnlText.add(new JTextField(10));
   c.add(pnlText, BorderLayout.WEST);

   pnlGrid.setLayout(new GridLayout(2,3));
   for (int i = 0; i < 6; i++)
      pnlGrid.add(new JButton("" + (i + 1)));
   c.add(pnlGrid, BorderLayout.EAST);
```

# JPanel sample code – part 2

```
   // Container c = getContentPane();
   // Add widgets to pnlText
   // c.add(pnlText);

   // Set layout manager for pnlGrid
   // Add button widgets to pnlGrid
   // c.add(pnlGrid, BorderLayout.EAST);

   pnlButtons.add(new JButton("Start"));
   pnlButtons.add(new JButton("Quit"));
   c.add(pnlButtons, BorderLayout.SOUTH);

   setTitle("Panel Demo");
   setSize(400,150);
   setVisible(true);
 }
```

# JPanel class hierarchy



A JPanel *is-a* JComponent *is-a* Container *is-a* Component *is-an* Object.