Ch.15: Advanced GUI Topics

Content Pane & Layout Managers

Lec.(5)

Understanding the Content Pane

- Content pane contains all the visible components in the container's user interface.
- Ignore the content pane if you only add components to, remove components from, or set the layout manager of a JFrame.

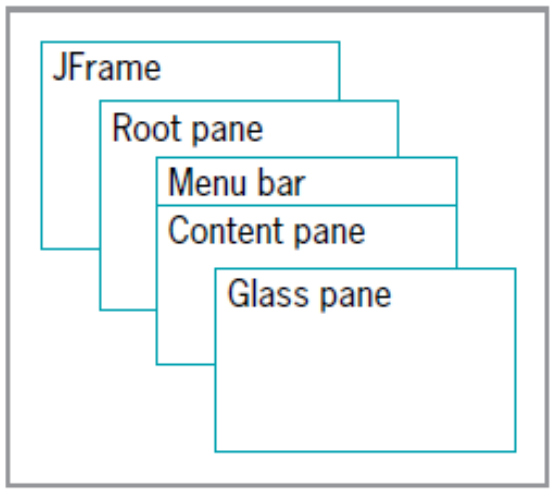


Figure 15-1 Parts of a JFrame

- refer to the content pane for all other actions, such as setting the background color.

Referring to the content Pane

- this.getContentPane().add(aButton);
- getContentPane().add(aButton);
- add(aButton);
- ----------
- Container con = getContentPane();
- con.add(button1);
- con.add(button2);
- con.add(button3);

```
import java.awt.*;
import javax.swing.*;
public class JFrameWithExplicitContentPane extends JFrame
{
   private final int SIZE = 180;
   private Container con = getContentPane();
   private JButton button = new JButton("Press Me");
   public JFrameWithExplicitContentPane()
   {
      super("Frame");
      setSize(SIZE, SIZE);
      con.setLayout(new FlowLayout());
      con.add(button);
   }
   public static void main(String[] args)
   {
      JFrameWithExplicitContentPane frame =
         new JFrameWithExplicitContentPane();
      frame.setVisible(true);
   }
}
```

Figure 15-2 The JFrameWithExplicitContentPane class

Color Class

- defines colors for you to use in your applications

BLACK	GREEN	RED
BLUE	LIGHT_GRAY	WHITE
CYAN	MAGENTA	YELLOW
DARK_GRAY	ORANGE	
GRAY	PINK	

Table 15-1 `Color` class constants

- You can also create your own Color object with the following statement:

```
Color someColor = new Color(r, g, b); //numbers can range from 0 to 255 ,
```

E.g.: 0, 0, 0 for black and 255, 255, 255 for white color

```
import java.awt.*;
import javax.swing.*;
import java.awt.Color;
public class JFrameWithColor extends JFrame
{
   private final int SIZE = 180;
   private Container con = getContentPane();
   private JButton button =
      new JButton("Press Me");
   public JFrameWithColor()
   {
      super("Frame");
      setSize(SIZE, SIZE);
      con.setLayout(new FlowLayout());
      con.add(button);
      con.setBackground(Color.YELLOW);
      button.setBackground(Color.RED);
      button.setForeground(Color.WHITE);
   }
   public static void main(String[] args)
   {
      JFrameWithColor frame =
         new JFrameWithColor();
      frame.setVisible(true);
   }
}
```

Learning More About Layout Managers

- A layout manager is an object that controls the size and position (that is, the layout) of components inside a Container object

Layout Manager	When to Use
`BorderLayout`	Use when you add components to a maximum of five sections arranged in north, south, east, west, and center positions
`FlowLayout`	Use when you need to add components from left to right; `FlowLayout` automatically moves to the next row when needed, and each component takes its preferred size
`GridLayout`	Use when you need to add components into a grid of rows and columns; each component is the same size
`CardLayout`	Use when you need to add components that are displayed one at a time
`BoxLayout`	Use when you need to add components into a single row or a single column
`GridBagLayout`	Use when you need to set size, placement, and alignment constraints for every component that you add

Table 15-2 Java layout managers

BorderLayout

- The default manager class for all content panes.
- Use with any container that has five or fewer components.
- it fill the screen in five regions: **north, south, east, west, and center.**
- constants for the regions

  BorderLayout.NORTH, BorderLayout.SOUTH, BorderLayout.EAST, BorderLayout.WEST, and BorderLayout.CENTER
- e.g:- *con.add(button, BorderLayout.CENTER);*

```
import javax.swing.*;
import java.awt.*;
public class JDemoBorderLayout extends JFrame
{
   private JButton nb = new JButton("North Button");
   private JButton sb = new JButton("South Button");
   private JButton eb = new JButton("East Button");
   private JButton wb = new JButton("West Button");
   private JButton cb = new JButton("Center Button");
   private Container con = getContentPane();
   public JDemoBorderLayout()
   {
      con.setLayout(new BorderLayout());
      con.add(nb, BorderLayout.NORTH);
      con.add(sb, BorderLayout.SOUTH);
      con.add(eb, BorderLayout.EAST);
      con.add(wb, BorderLayout.WEST);
      con.add(cb, BorderLayout.CENTER);
      setSize(400, 150);
   }
   public static void main(String[] args)
   {
      JDemoBorderLayout frame = new JDemoBorderLayout();
      frame.setVisible(true);
   }
}
```

Figure 15-7 The JDemoBorderLayout class

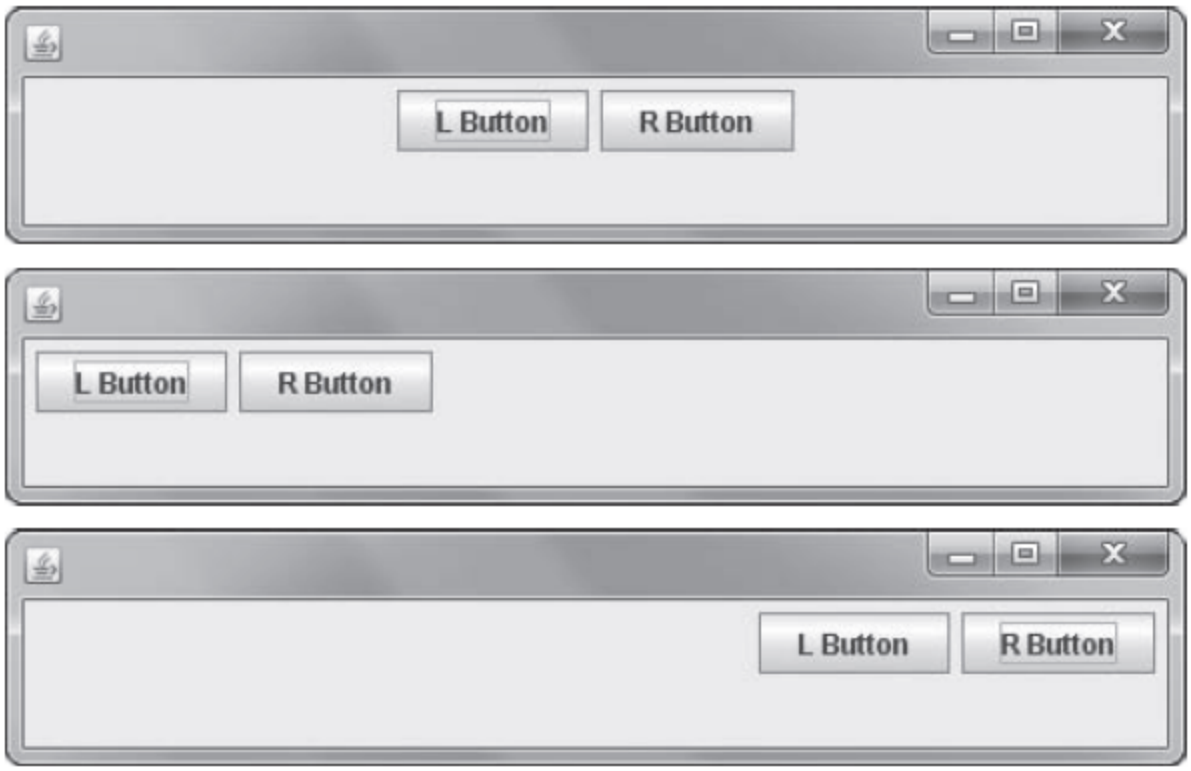
- Note each component expands based on its region's size

FlowLayout

- arrange components in rows across the width of a Container.
- each component retains its default size, or preferred size. For example, a JButton's preferred size is the size that is large enough to hold the JButton's text.
- **Alignment constants** :
  - FlowLayout.LEFT
  - FlowLayout.CENTER
  - FlowLayout.RIGHT

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class JDemoFlowLayout extends JFrame implements ActionListener
{
   private JButton lb = new JButton("L Button");
   private JButton rb = new JButton("R Button");
   private Container con = getContentPane();
   private FlowLayout layout = new FlowLayout();
   public JDemoFlowLayout()
   {
      con.setLayout(layout);
      con.add(lb);
      con.add(rb);
      lb.addActionListener(this);
      rb.addActionListener(this);
      setSize(500, 100);
   }
   public void actionPerformed(ActionEvent event)
   {
      Object source = event.getSource();
      if(source == lb)
         layout.setAlignment(FlowLayout.LEFT);
      else
         layout.setAlignment(FlowLayout.RIGHT);
      con.invalidate();
      con.validate();
   }
   public static void main(String[] args)
   {
      JDemoFlowLayout frame = new JDemoFlowLayout();
      frame.setVisible(true);
   }
}
```

Figure 15-8 The JDemoFlowLayout application



- The invalidate() call marks the container as needing to be laid out.
- The validate() call causes the components to be rearranged based on the newly assigned layout.

GridLayout

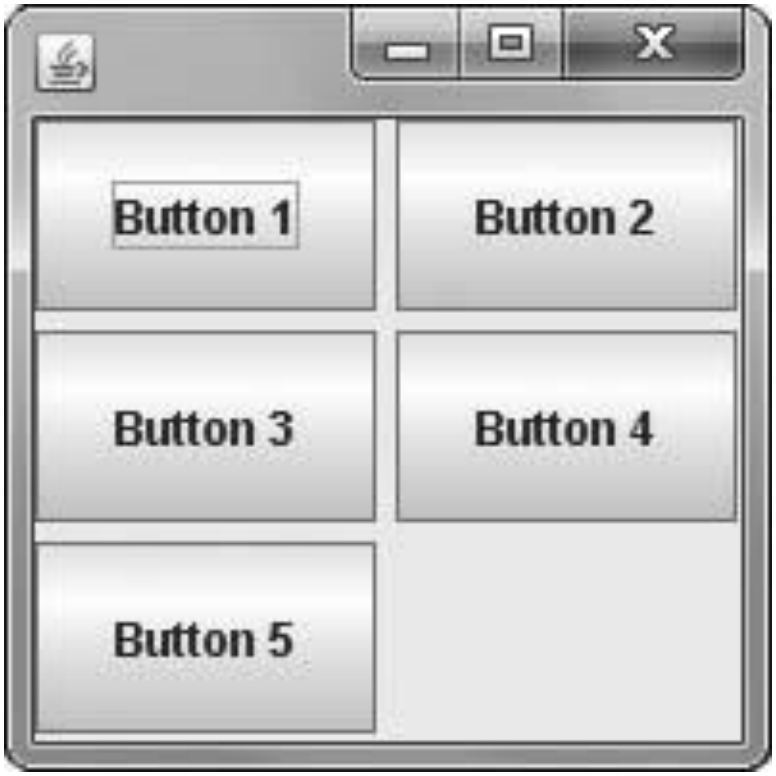
- Arrange components into equal rows and columns
- you indicate the numbers of rows and columns you want.
- E.g: GridLayout with four horizontal rows and five vertical columns

```
setLayout(new GridLayout(4, 5))
```

- Components are placed in sequence from left to right across each row.
- you can’t skip a position or specify an exact position for a component.

GridLayout

- a GridLayout with 3 horizontal rows and 2 vertical columns, and horizontal and vertical gaps of 5 pixels.



```
import javax.swing.*;
import java.awt.*;
public class JDemoGridLayout extends JFrame
{
   private JButton b1 = new JButton("Button 1");
   private JButton b2 = new JButton("Button 2");
   private JButton b3 = new JButton("Button 3");
   private JButton b4 = new JButton("Button 4");
   private JButton b5 = new JButton("Button 5");
   private GridLayout layout = new GridLayout(3, 2, 5, 5);
   private Container con = getContentPane();
   public JDemoGridLayout()
   {
      con.setLayout(layout);
      con.add(b1);
      con.add(b2);
      con.add(b3);
      con.add(b4);
      con.add(b5);
      setSize(200, 200);
   }
   public static void main(String[] args)
   {
      JDemoGridLayout frame = new JDemoGridLayout();
      frame.setVisible(true);
   }
}
```

Figure 15-10 The JDemoGridLayout class

CardLayout

- The CardLayout manager generates a stack of containers or components, one on top of another
- Each component in the group is referred to as a card
- each card can be any component type—for example, a JButton, JLabel, or Jpanel

Constructors

- CardLayout( ) creates a card layout without a horizontal or vertical gap.
- CardLayout(int hgap, int vgap) creates a card layout with the specified horizontal and vertical gaps. The horizontal gaps are placed at the left and right edges. The vertical gaps are placed at the top and bottom edges.

Methods

- **add(aString, aComponent);** add a component to a content pane whose layout manager is CardLayout
  - aString represents a name you want to use to identify the Component card that is added.
- **next(getContentPane());** flips to the next card of the container
- **previous(getContentPane()),**

  first(getContentPane()), and last(getContentPane()); to flip to the previous, first, and last card, respectively.

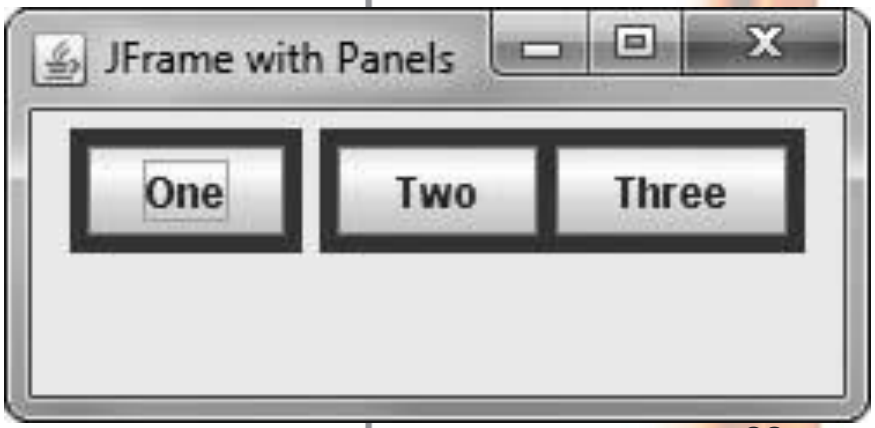
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class JDemoCardLayout extends JFrame
   implements ActionListener
{
   private CardLayout cards = new CardLayout();
   private JButton b1 = new JButton("Ace of Hearts");
   private JButton b2 = new JButton("Three of Spades");
   private JButton b3 = new JButton("Queen of Clubs");
   private Container con = getContentPane();
   public JDemoCardLayout()
   {
      con.setLayout(cards);
      con.add("ace", b1);
      b1.addActionListener(this);
      con.add("three", b2);
      b2.addActionListener(this);
      con.add("queen", b3);
      b3.addActionListener(this);
      setSize(200, 100);

   }
   public void actionPerformed(ActionEvent e)
   {
      cards.next(getContentPane());
   }
   public static void main(String[] args)
   {
      JDemoCardLayout frame = new JDemoCardLayout();
      frame.setVisible(true);
   }
}
```

Figure 15-13 Output of `JDemoCardLayout` when it first appears on the screen, after the user clicks once, and after the user clicks twice

JPanel Class

- A JPanel is a plain, borderless surface that can hold lightweight UI components.
- JPanel is a Container hold other UI components, such as JButtons, JCheckBoxes, or even other Jpanels
- By using JPanels within JPanels, you can create an infinite variety of screen layouts.
- FlowLayout is the default layout manager for JPanel

```
import javax.swing.*;
import java.awt.*;
import java.awt.Color;
public class JFrameWithPanels extends JFrame
{
    private final int WIDTH = 250;
    private final int HEIGHT = 120;
    private JButton button1 = new JButton("One");
    private JButton button2 = new JButton("Two");
    private JButton button3 = new JButton("Three");
    public JFrameWithPanels()
    {
        super("JFrame with Panels");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel panel1 = new JPanel();
        JPanel pane12 = new JPanel();
        Container con = getContentPane();
        con.setLayout(new FlowLayout());
        con.add(panel1);
        con.add(panel2);
        panel1.add(button1);
        panel1.setBackground(Color.BLUE);
        panel2.add(button2);
        panel2.add(button3);
        panel2.setBackground(Color.BLUE);
        setSize(WIDTH, HEIGHT);
    }
```



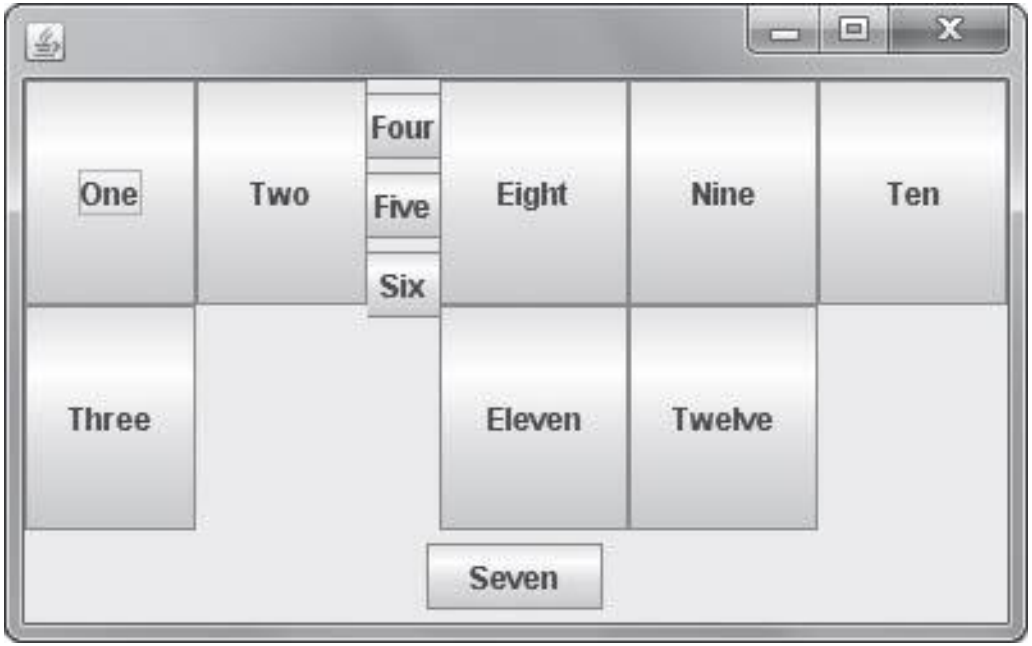
JPanel() Constructors

- JPanel() creates a JPanel with double buffering and a flow layout.
- JPanel(LayoutManager layout) creates a JPanel with the specified layout manager and double buffering.
- JPanel(boolean isDoubleBuffered) creates a JPanel with a flow layout and the specified double-buffering strategy
- Double buffering provides the viewer with updated screens that do not flicker while being redrawn.

example

```
import javax.swing.*;
import java.awt.*;
public class JDemoManyPanels extends JFrame
{
 // Twelve buttons
    private JButton button01 = new JButton("One");
    private JButton button02 = new JButton("Two");
    private JButton button03 = new JButton("Three");
    private JButton button04 = new JButton("Four");
    private JButton button05 = new JButton("Five");
    private JButton button06 = new JButton("Six");
    private JButton button07 = new JButton("Seven");
    private JButton button08 = new JButton("Eight");
    private JButton button09 = new JButton("Nine");
    private JButton button10 = new JButton("Ten");
    private JButton button11 = new JButton("Eleven");
    private JButton button12 = new JButton("Twelve");

 // Four panels
    private JPanel panel01 = new JPanel(new GridLayout(2, 0));
    private JPanel panel02 = new JPanel(new FlowLayout());
    private JPanel panel03 = new JPanel(new FlowLayout());
    private JPanel panel04 = new JPanel(new GridLayout(2, 0));
```

```
    public JDemoManyPanels()
    {
        setLayout(new BorderLayout());
        add(panel01, BorderLayout.WEST);
        add(panel02, BorderLayout.CENTER);
        add(panel03, BorderLayout.SOUTH);
        add(panel04, BorderLayout.EAST);

        panel01.add(button01);
        panel01.add(button02);
        panel01.add(button03);

        panel02.add(button04);
        panel02.add(button05);
        panel02.add(button06);

        panel03.add(button07);

        panel04.add(button08);
        panel04.add(button09);
        panel04.add(button10);
        panel04.add(button11);
        panel04.add(button12);

        setSize(400, 250);
    }
    public static void main(String[] args)
    {
        JDemoManyPanels frame = new JDemoManyPanels();
        frame.setVisible(true);
    }
}
```


One
Two
Four
Five
Six
Eight
Nine
Ten
Three
Eleven
Twelve
Seven