

Chapter 8 Getting Started with Graphics Programming

- ☞ Graphics Class Hierarchy
- ☞ Frames
 - Creating frames, centering frames, adding components to frames
- ☞ Layout Managers
 - FlowLayout, GridLayout, BorderLayout
- ☞ Drawing on Panels
 - The `paintComponent` method
- ☞ Using Colors, Fonts, and Font Metrics
- ☞ Drawing Geometric Figures
 - Lines, Rectangles, Ovals, Arcs, and Polygons
- ☞ Event-Driven Programming
 - Event Source, Listener, Listener Interface



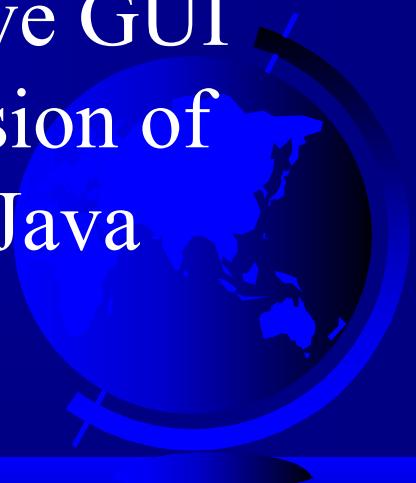
Chapter 8 Getting Started with Graphics Programming

- ☞ Until now you have used the character-mode programming, including input and output. It is called console application.
- ☞ No graphical interface, no window, no images, no button, no ...
- ☞ Today, either application software or web applications use "graphical user interface" (GUI, goo-ee), a communication interface between users and programmers



Chapter 8 Getting Started with Graphics Programming

- ☞ Sun introduce Abstract Window Toolkit (AWT) together with JDK. AWT is especially designed for programming GUI in Java.
- ☞ AWT has many useful Java classes which can be used in Java GUI programming.
- ☞ IN order to develop more comprehensive GUI projects, recently Sun introduces extension of AWT called swing, which is under the Java package of javax, together with JDK



Chapter 8 Getting Started with Graphics Programming

- ☞ Swing components are less dependent on the target platform and use less of the native GUI resources. Swing components are often called lightweight components, while AWT components are called heavyweight components
- ☞ AWT components are still supported by Java 2.
- ☞ Java has a very rich set of classes which help building graphic user interfaces, including frame, panel, label, button, text field, text area, combo box, check box, radio button, menu, scroll bar, scroll pane, and tabbed pane

Chapter 8 Getting Started with Graphics Programming

- ☞ Java API system heavily use class inheritance and interface. It contains the essential classes

Component superclass of all user interface classes

Color color of graphics

Container group components (frame, applet, and panel
are containers)

Jcomponent superclass of all the lightweight Swing
components, which are drawn directly on canvases

Japplet container holding Swing interface
components

Jdialog popup window as a temporary window to
receive additional information



Chapter 8 Getting Started with Graphics Programming

JFrame container holding Swing user interface components

JPanel invisible container holding user interface components. Panels can be nested. JPanel can be used as a canvas to draw graphics

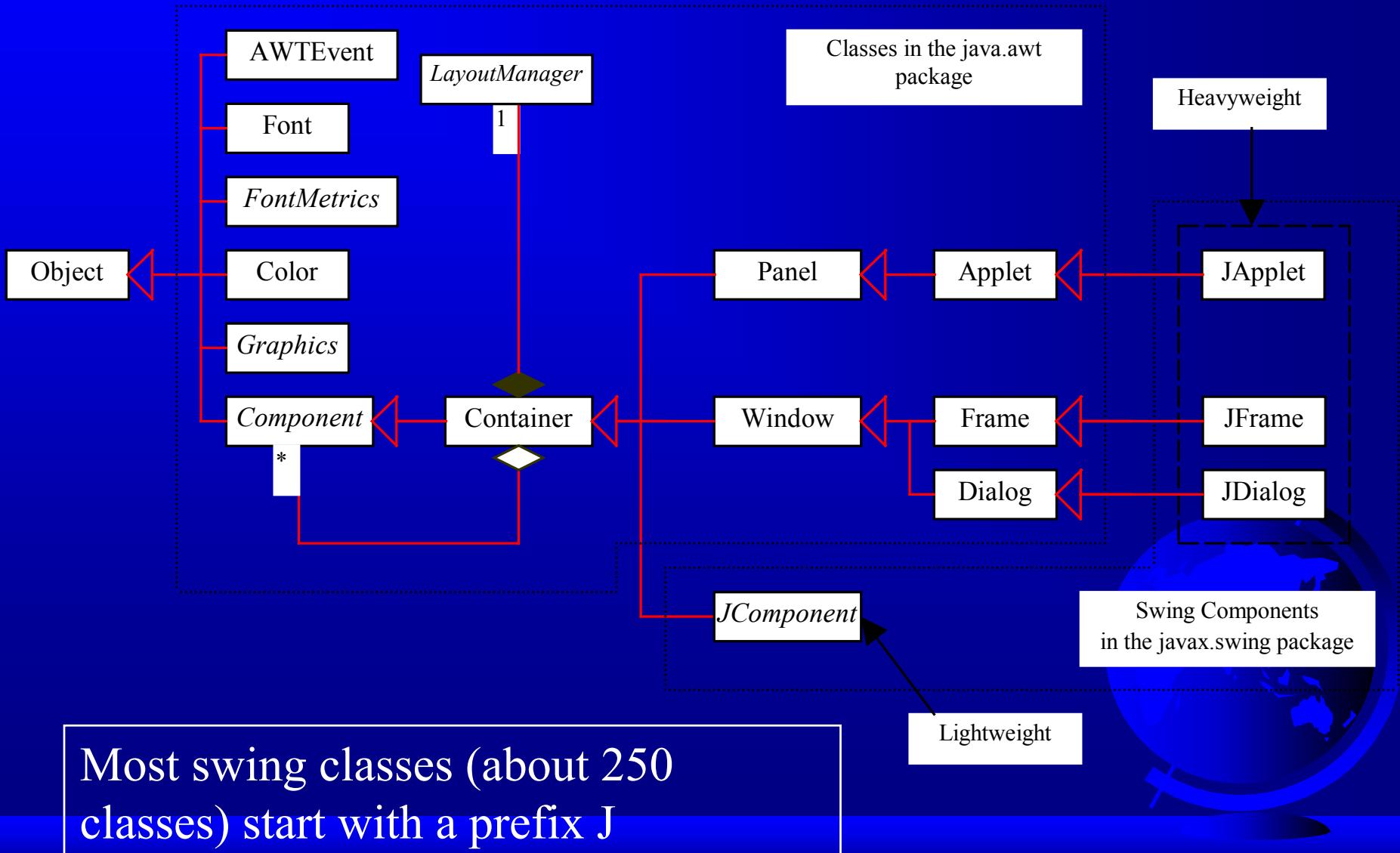
Graphics abstract class that provides a graphical content for drawing strings, lines, and simple shapes

Font used to specify different font.

FontFormat used to measure the properties of fonts used in drawings



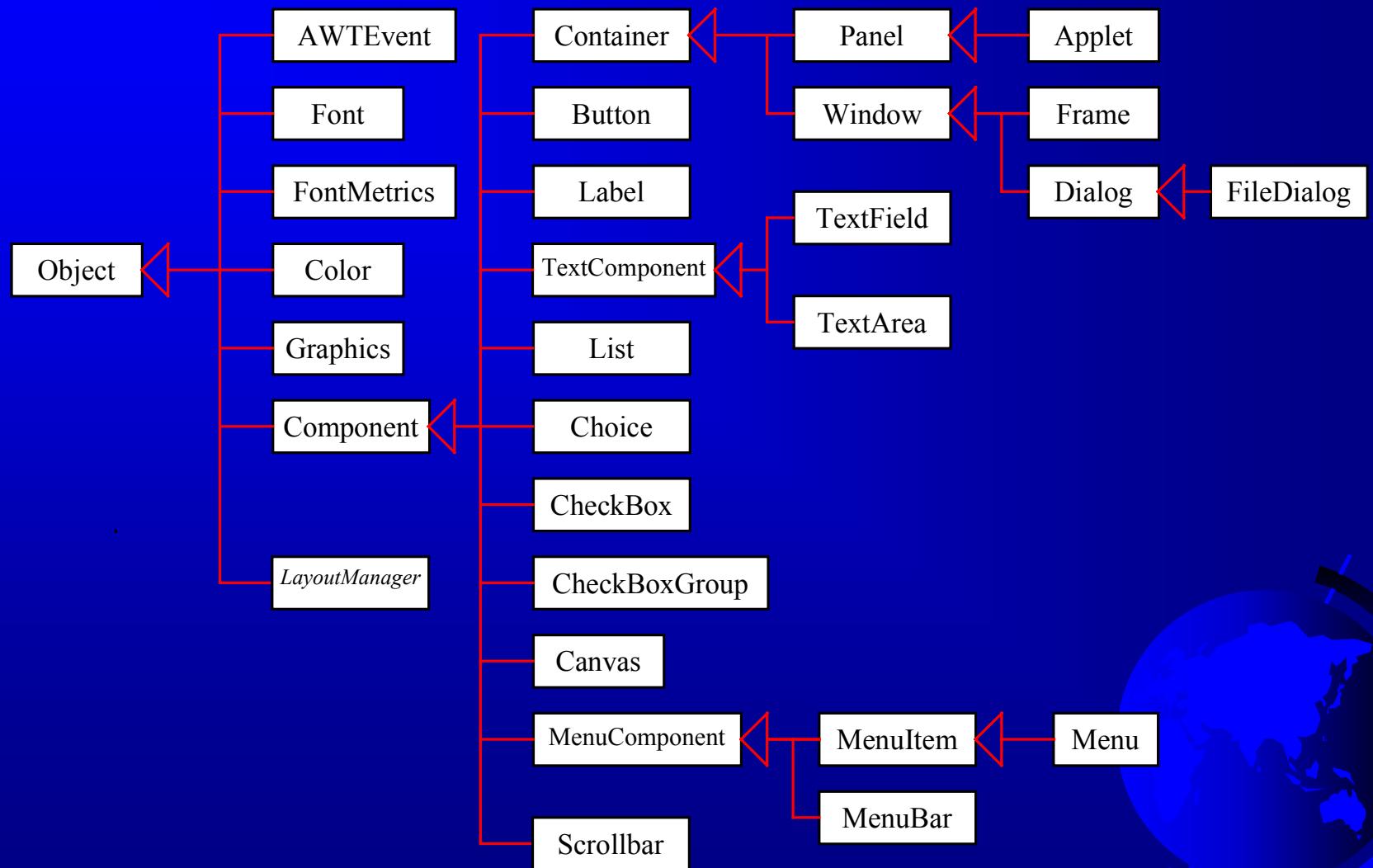
Graphics Class Hierarchy (Swing)



JComponent



AWT (Optional)



Frames

- ☞ Either a frame or an applet is needed to create a user interface which contains user interface components. Here we only discuss about frame. Applet will be covered in next chapter.
- ☞ Frame is a window that is not contained inside another window. Frame is the basis to contain other user interface components in Java graphical applications.
- ☞ The Frame class can be used to create windows.



UI Components

Frame

Frame

Pull-down Menus

Panel

User Interface
Components (UI)

Panel

Panel
UI

Panel
UI

Panel
UI

Applet

Applet

Pull-down Menus

Panel

User Interface
Components

Panel

User Interface
Components

Panel

User Interface
Components

Panel

User Interface
Components

panel

Creating Frames

```
import javax.swing.*;
public class MyFrame
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("Test Frame");
        frame.setSize(400, 300);
        frame.setVisible(true);
        frame.setDefaultCloseOperation
            (JFrame.EXIT_ON_CLOSE);
    }
}
```

NOTE: To enable it to run in JDK 1.2, the EXIT_ON_CLOSE option is commented.





Test Frame



Centering Frames

JFrame constructors:

```
JFrame frame = new JFrame()
```

```
JFrame frame = new JFrame(String title)
```

JFrame is not displayed until the
frame.setVisible(true)

is applied.

```
frame.setSize(400,300)
```

Specifies the size of frame (400 by 300 pixels)



Centering Frames

By default, a frame is displayed in the upper-left corner of the screen. To display a frame at a specified location, you can use the `setLocation(int x, int y)` method in the JFrame class. This method places the upper-left corner of a frame at location (x, y).



Obtain the width and height of Screen

Using `java.awt.Toolkit` class, you can obtain information about the sizes of a screen and a frame.

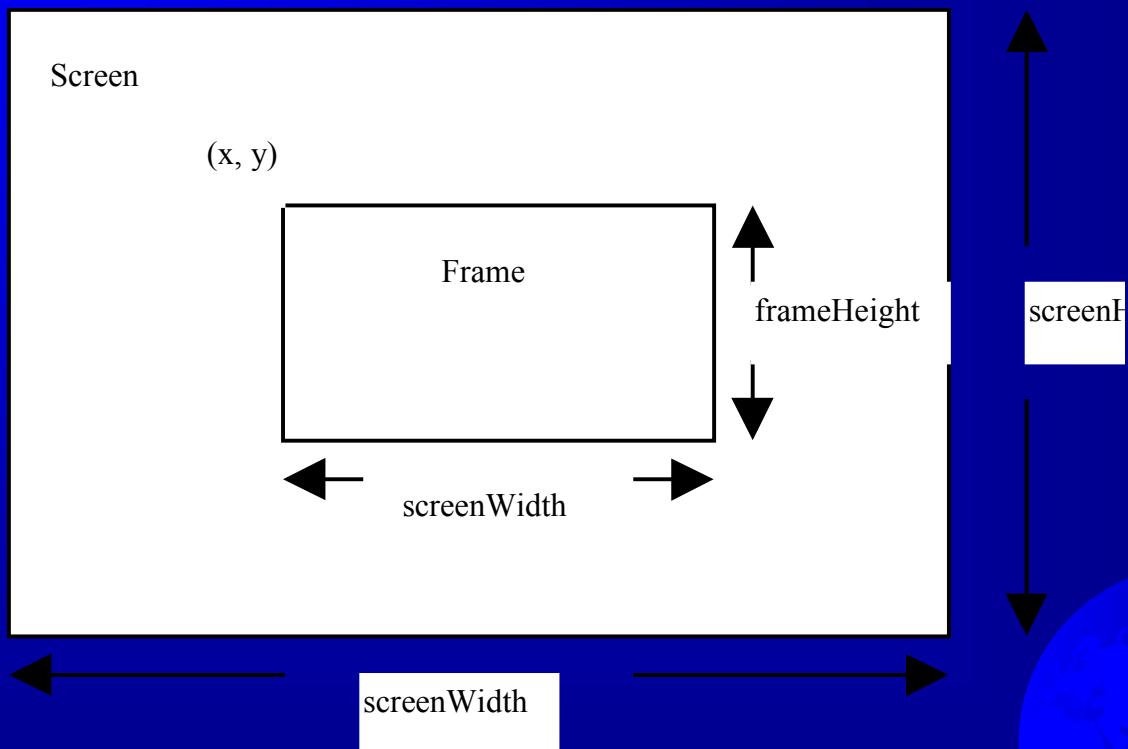
```
Dimension screenSize=Toolkit.getDefaultToolkit().getScreenSize();
int screenWidth=screenSize.width;
int screenHeight=screenSize.height;
```

```
Dimension frameSize=frame.getSize();
int x=(screenWidth-frameSize.width)/2;
int y=(screenHeight-frameSize.height)/2;
```

Location of frame



Centering Frames, cont.



```
// CenterFrame.java: Display a frame
import javax.swing.*;
import java.awt.*;

public class CenterFrame
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("Centered Frame");
        frame.setSize(400, 300);

        // New in JDK 1.3 to exit the program upon closing
        frame.setDefaultCloseOperation
            (JFrame.EXIT_ON_CLOSE);
    }
}
```

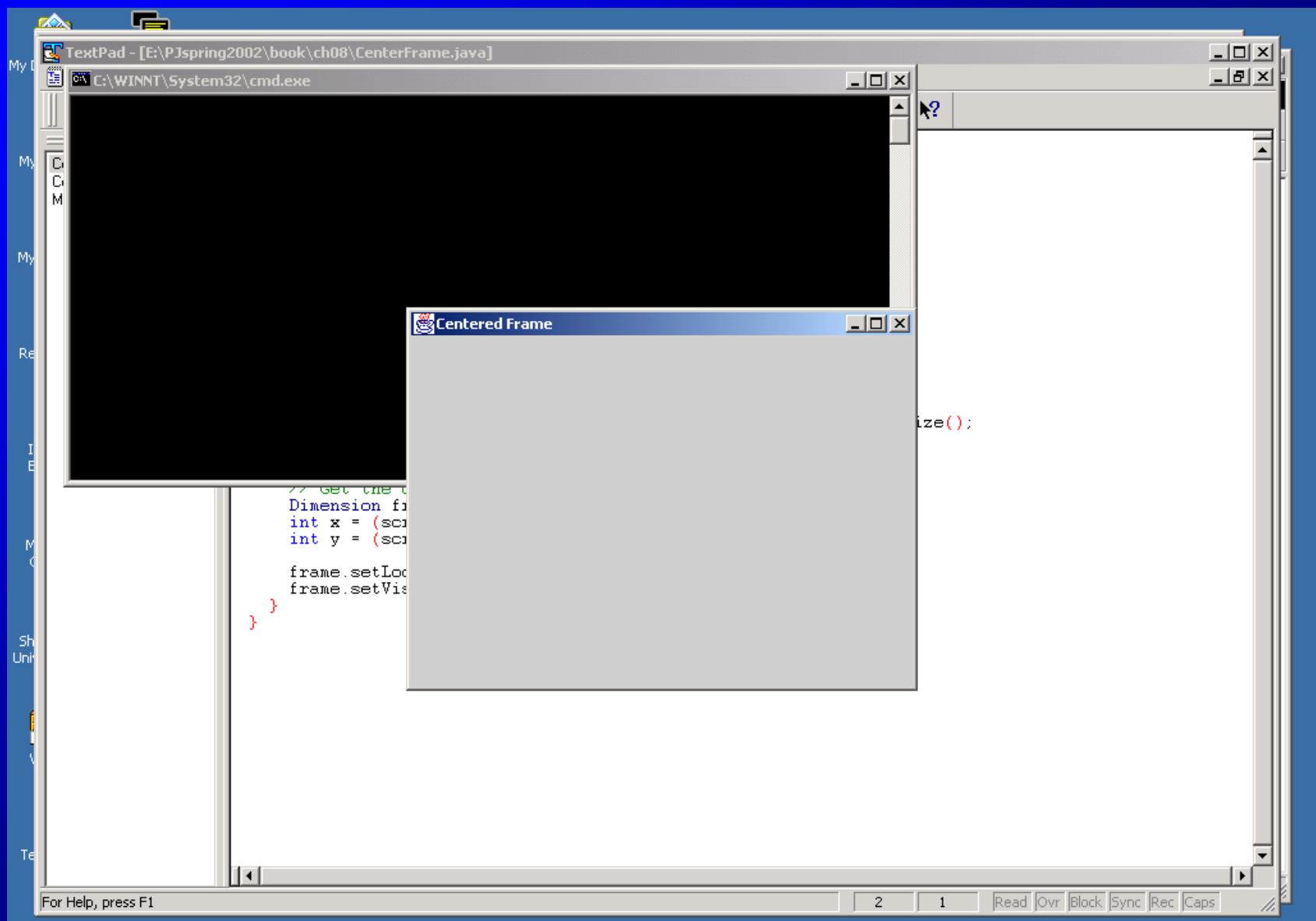


```
// Get the dimension of the screen
Dimension screenSize =
    Toolkit.getDefaultToolkit().getScreenSize();
int screenWidth = screenSize.width;
int screenHeight = screenSize.height;

// Get the dimension of the frame
Dimension frameSize = frame.getSize();
int x = (screenWidth - frameSize.width)/2;
int y = (screenHeight - frameSize.height)/2;

frame.setLocation(x, y);
frame.setVisible(true);
}
```





Adding Components into a Frame

```
// MyFrameWithComponents.java: Display a frame
import javax.swing.*;

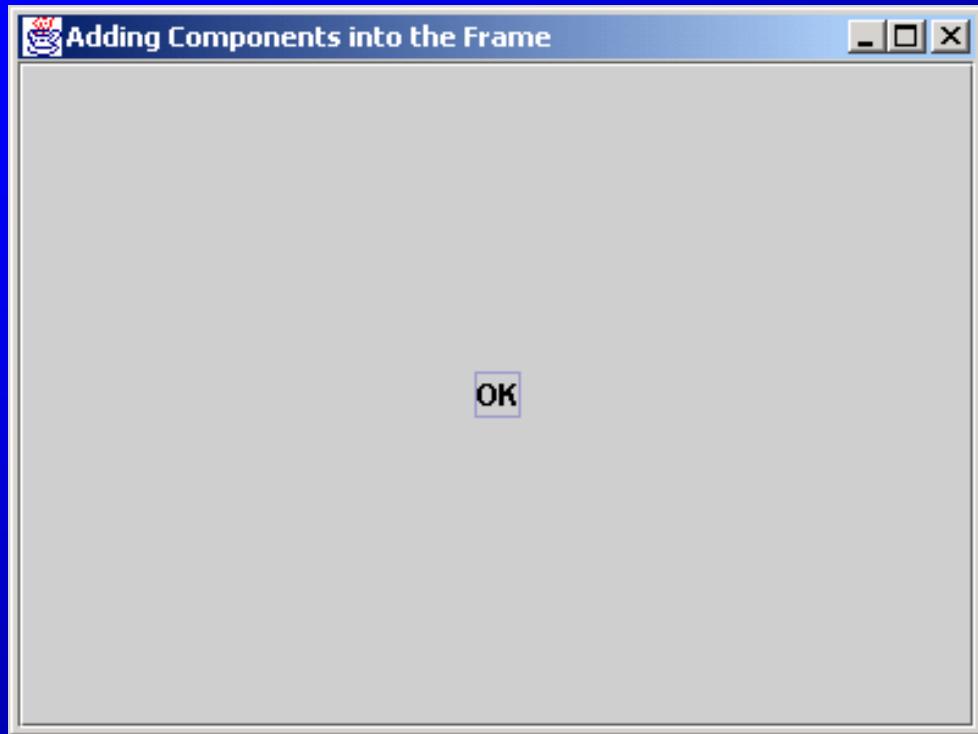
public class MyFrameWithComponents
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("Adding
Components into the Frame");
```



Adding Components into a Frame

```
// Add a button into the frame  
frame.getContentPane().add(new JButton("OK"));  
  
frame.setSize(400, 300);  
frame.setVisible(true);  
// frame.setDefaultCloseOperation  
// (JFrame.EXIT_ON_CLOSE);  
}  
}
```





contentPane() method in the Jframe class returns the content pane of the frame.



Layout Managers

- ☞ Default layout of components in content pane is located in the center of the frame.
- ☞ Java's layout managers provide a level of abstraction to automatically map your user interface on all windowing systems.
- ☞ The UI components are placed in containers. Each container has a layout manager to arrange the UI components within the container.



Layout Managers

- ☞ The syntax to set a layout manager is as follows:

```
Container.setLayout(new SpecificLayout());
```

JFrame, JPanel, JApplet

FlowLayout, GridLayout,
BorderLayout, CardLayout,
GridBagLayout



Kinds of Layout Managers

- ☞ FlowLayout
- ☞ GridLayout
- ☞ BorderLayout
- ☞ CardLayout
- ☞ GridBagLayout



FlowLayout

FlowLayout is the simplest layout manager.

The components are arranged in the container from left to right.

The layout is changed with the resize of the frame.

There are three FlowLayout constants used to align the components

FlowLayout.RIGHT, FlowLayout.LEFT,
or FlowLayout.CENTER



FlowLayout Constructors

- `public FlowLayout(int align, int hGap, int vGap)`
Constructs a new `FlowLayout` with a specified alignment, horizontal gap, and vertical gap. The *gaps* are the distances in pixel between components.
- `public FlowLayout(int alignment)`
Constructs a new `FlowLayout` with a specified alignment and a default gap of five pixels for both horizontal and vertical.
- `public FlowLayout()`
Constructs a new `FlowLayout` with a default center alignment and a default gap of five pixels for both horizontal and vertical.



Example 8.1

Testing the FlowLayout Manager

The components are arranged in the container from left to right in the order in which they were added. When one row becomes filled, a new row is started.



```
// ShowFlowLayout.java: Demonstrate using FlowLayout
import javax.swing.JButton;
import javax.swing.JFrame;
import java.awt.Container;
import java.awt.FlowLayout;

public class ShowFlowLayout extends JFrame
{
    // Default constructor
    public ShowFlowLayout()
    {
        // Get the content pane of the frame
        Container container = getContentPane();
```



```
// Set FlowLayout, aligned left with horizontal gap 10  
// and vertical gap 20 between components  
container.setLayout  
    (new FlowLayout(FlowLayout.RIGHT, 10, 20));  
  
// Add buttons to the frame  
for (int i=1; i<=10; i++)  
    container.add(new JButton("Component " + i));  
}
```

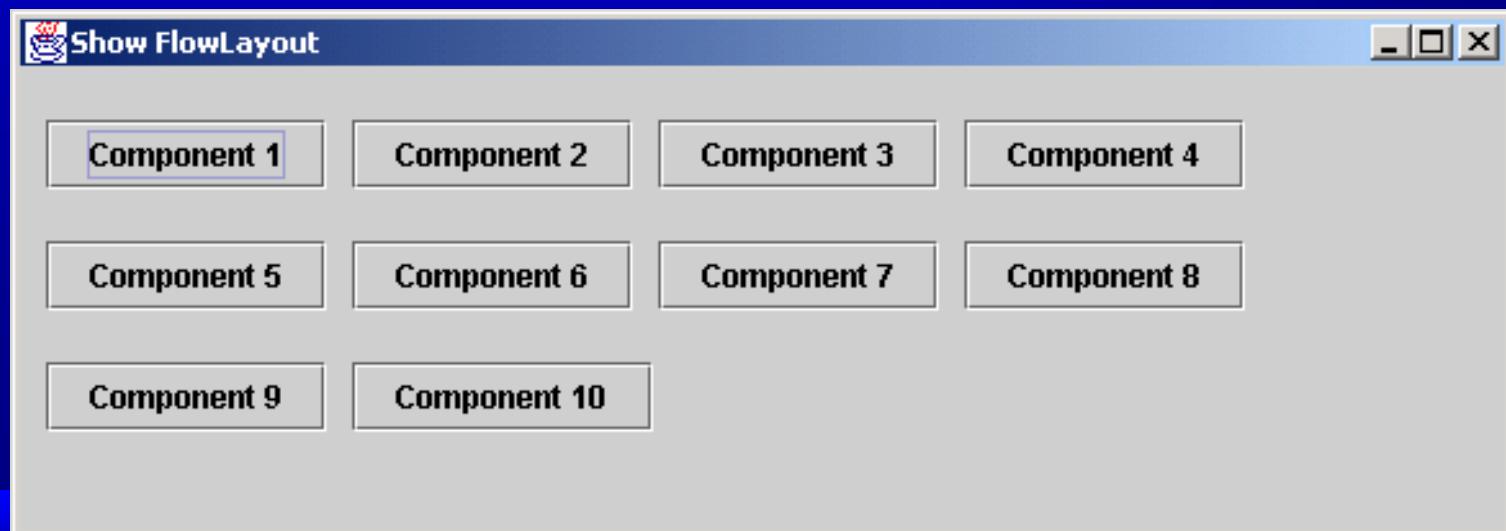


```
// Main method
public static void main(String[] args)
{
    ShowFlowLayout frame = new ShowFlowLayout();
    frame.setTitle("Show FlowLayout");
    frame.setDefaultCloseOperation
        (JFrame.EXIT_ON_CLOSE);
    frame.setSize(200, 200);
    frame.setVisible(true);
}
```





```
container.setLayout  
    (new FlowLayout(FlowLayout.LEFT, 10, 20));
```



Example 8.2

Testing the GridLayout Manager

The GridLayout manager arranges components in a grid (matrix) formation with the number of rows and columns defined by the constructor. The components are placed in the grid from left to right starting with the first row, then the second, and so on.



GridLayout Constructors

- ☞ `public GridLayout(int rows,
int columns)`

Constructs a new GridLayout with the specified number of rows and columns.

- ☞ `public GridLayout(int rows, int
columns, int hGap, int vGap)`

Constructs a new GridLayout with the specified number of rows and columns, along with specified horizontal and vertical gaps between components.



GridLayout Constructors

- ☞ If row and column numbers are both non-zero, row number has high privilege or is determined, while the column number is calculated.
- ☞ If the row number is zero, the column number is determined and the row number is calculated.
- ☞ If the column number is zero, the row number is determined, and the column number is calculated.



```
// ShowGridLayout.java: Demonstrate using GridLayout
import javax.swing.JButton;
import javax.swing.JFrame;
import java.awt.GridLayout;
import java.awt.Container;

public class ShowGridLayout extends JFrame
{
    // Default constructor
    public ShowGridLayout()
    {
        // Get the content pane of the frame
        Container container = getContentPane();
```

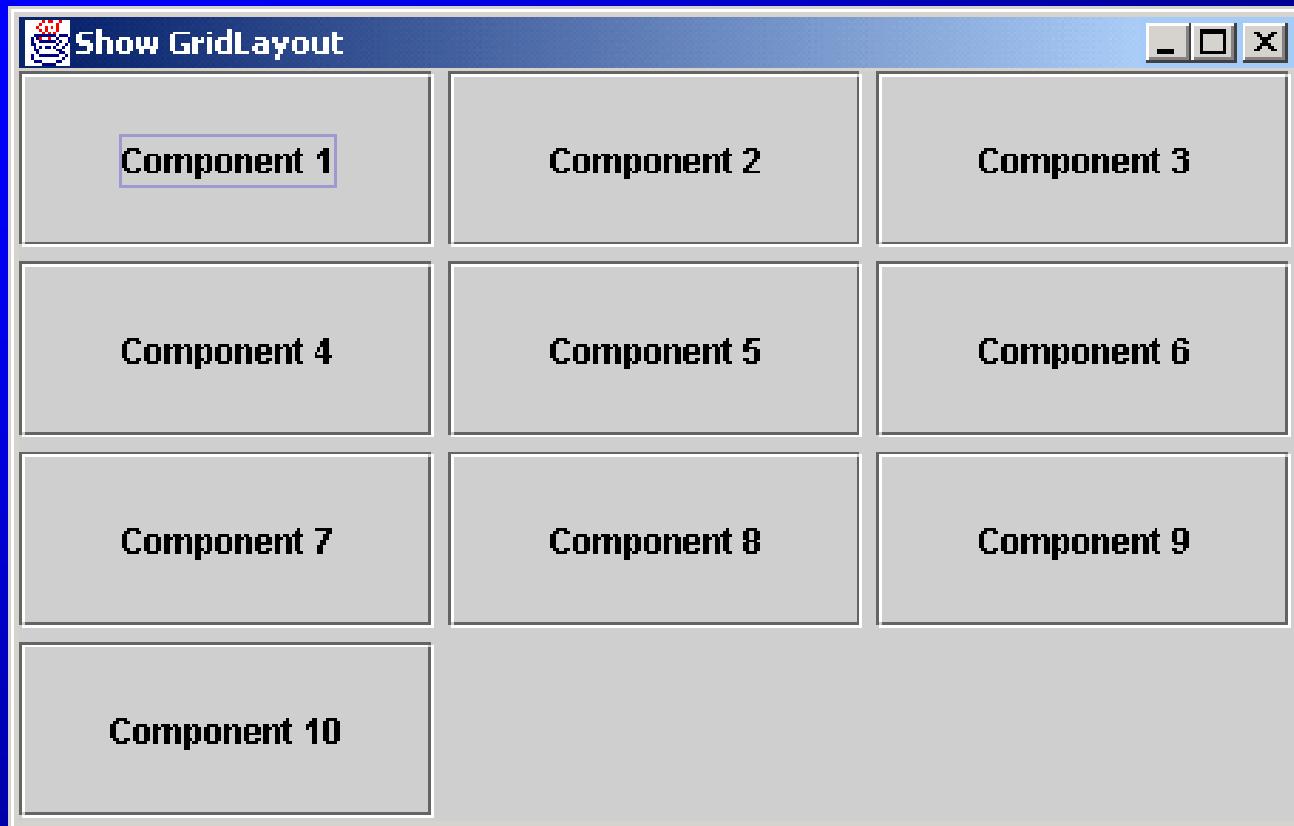


```
// Set GridLayout, 4 rows, 3 columns, and gaps 5 between  
// components horizontally and vertically  
container.setLayout(new GridLayout(4, 3, 5, 5));  
  
// Add buttons to the frame  
for (int i=1; i<=10; i++)  
    container.add(new JButton("Component " + i));  
}
```



```
// Main method
public static void main(String[] args)
{
    ShowGridLayout frame = new ShowGridLayout();
    frame.setTitle("Show GridLayout");
    // frame.setDefaultCloseOperation
    // (JFrame.EXIT_ON_CLOSE);
    frame.setSize(200, 200);
    frame.setVisible(true);
}
```





```
container.setLayout(new GridLayout(4, 6, 5, 5));
```



```
container.setLayout(new GridLayout(0, 3, 5, 5));
```



```
container.setLayout(new GridLayout(3, 0, 5, 5));
```



Example 8.3

Testing the BorderLayout Manager

The BorderLayout manager divides the window into five areas: East, South, West, North, and Center. Components are added to a BorderLayout by using

`add(Component, constraint)`, where constraint is `BorderLayout.East`, `BorderLayout.South`, `BorderLayout.West"`, `BorderLayout.North"`, or `BorderLayout.Center`.



Example 8.3

Testing the BorderLayout Manager

Constructors:

public BorderLayout(int hGap, int vGap)

-- hGap and vGap are horizontal and vertical gaps between components

public BorderLayout()



```
// ShowBorderLayout.java: Demonstrate using  
// BorderLayout  
import javax.swing.JButton;  
import javax.swing.JFrame;  
import java.awt.Container;  
import java.awt.BorderLayout;  
  
public class ShowBorderLayout extends JFrame  
{  
    // Default constructor  
    public ShowBorderLayout()  
    {  
        // Get the content pane of the frame  
        Container container = getContentPane();
```

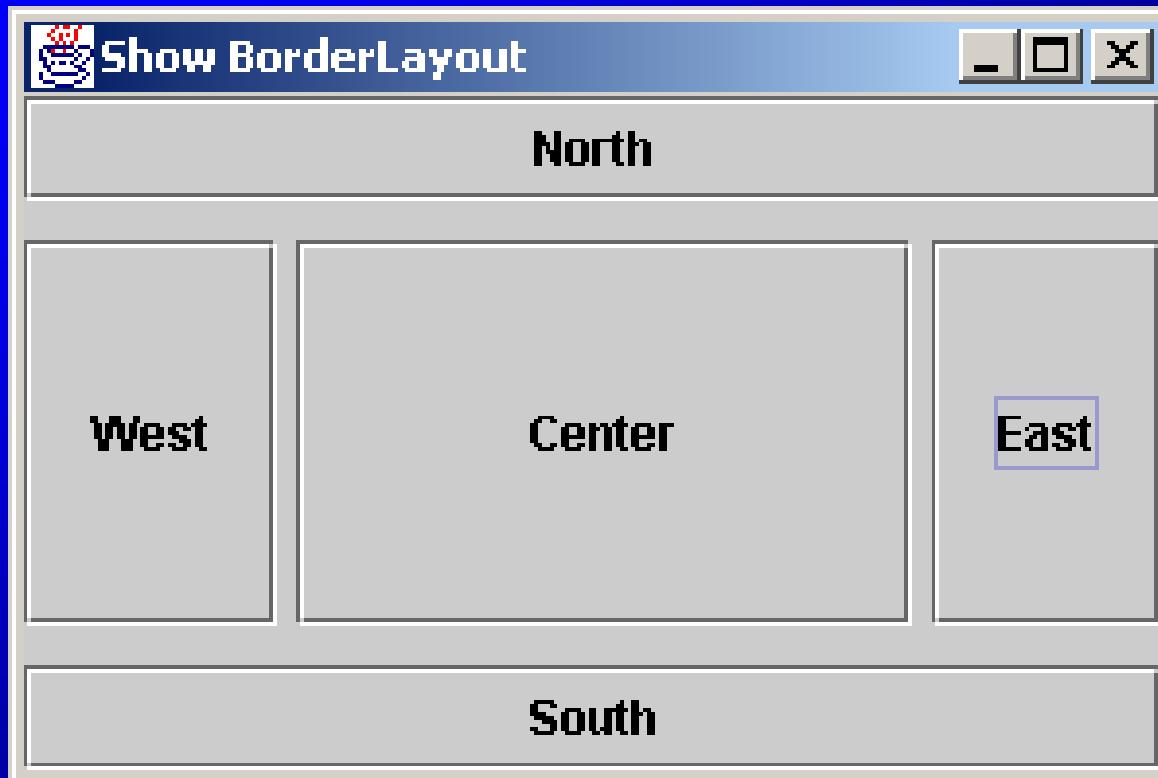


```
// Set BorderLayout with horizontal gap 5 and  
// vertical gap 10  
container.setLayout(new BorderLayout(5, 10));  
// Add buttons to the frame  
container.add(new JButton ("East"),  
             BorderLayout.EAST);  
container.add(new JButton("South"),  
             BorderLayout.SOUTH);  
container.add(new JButton("West"),  
             BorderLayout.WEST);  
container.add(new JButton("North"),  
             BorderLayout.NORTH);  
container.add(new JButton("Center"),  
             BorderLayout.CENTER);  
}
```

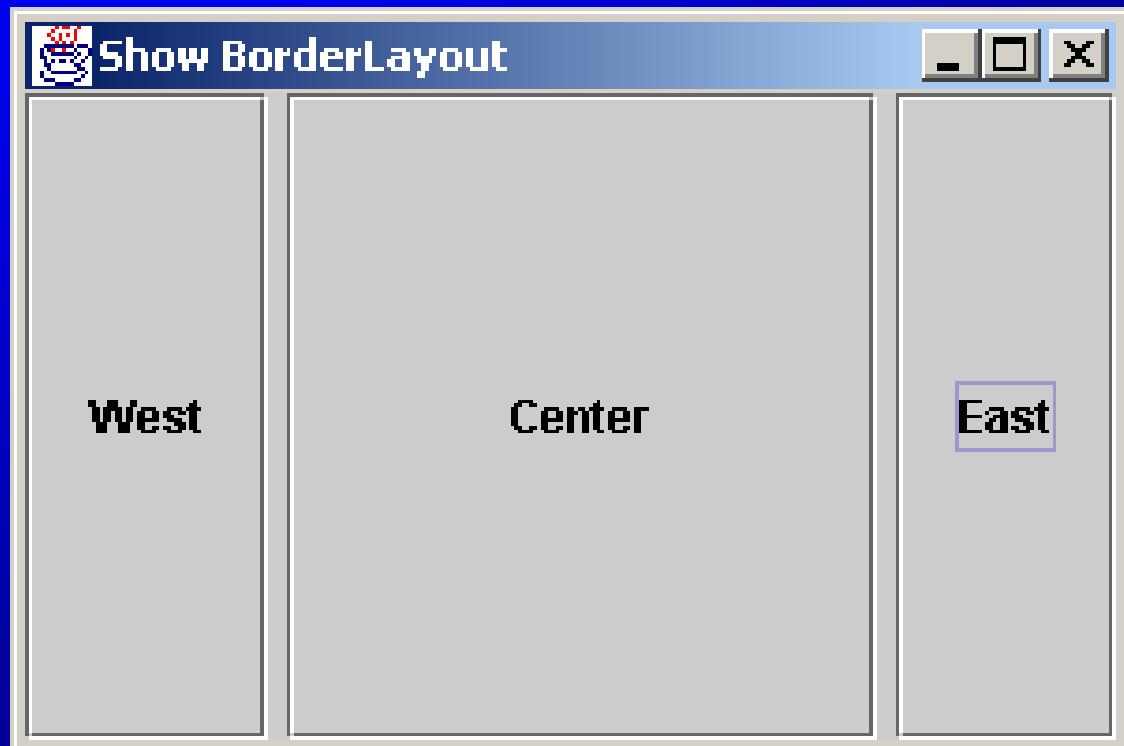


```
// Main method
public static void main(String[] args)
{
    ShowBorderLayout frame = new ShowBorderLayout();
    frame.setTitle("Show BorderLayout");
    // EXIT_ON_CLOSE = 3
    // frame.setDefaultCloseOperation(3);
    frame.setSize(300, 200);
    frame.setVisible(true);
}
```

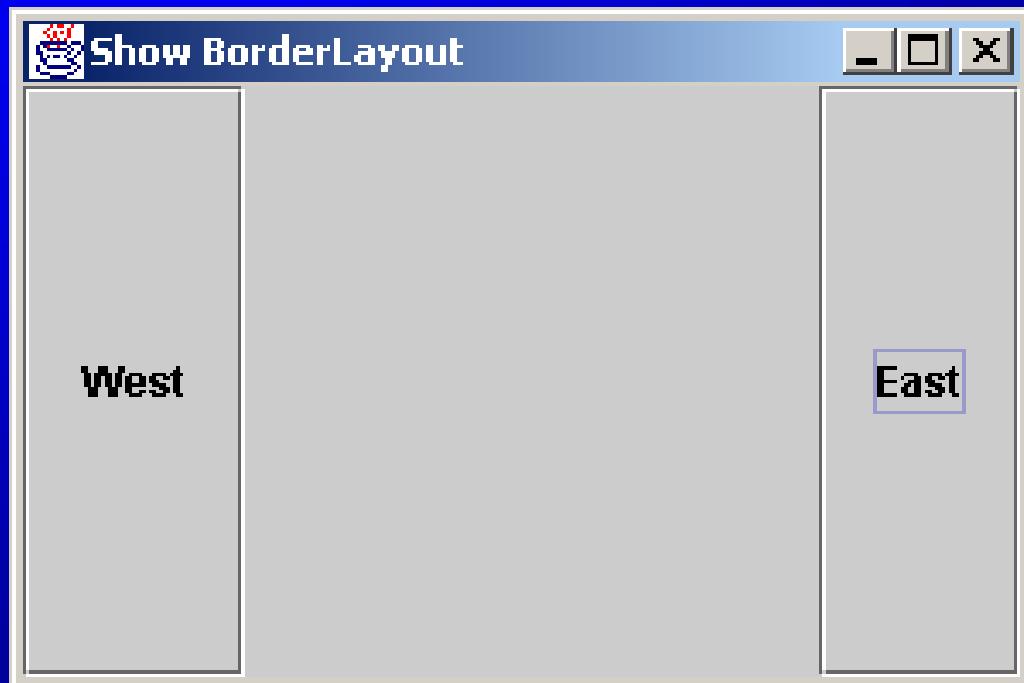




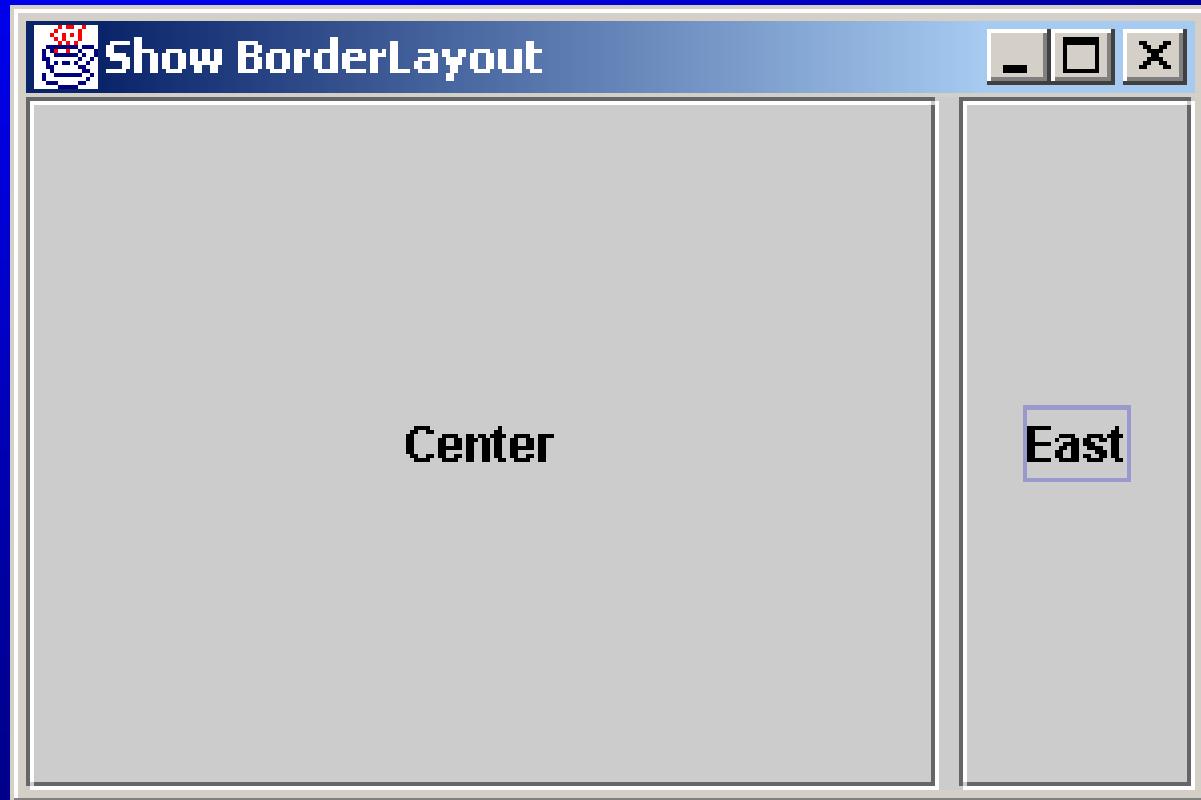
East, West, and Center only



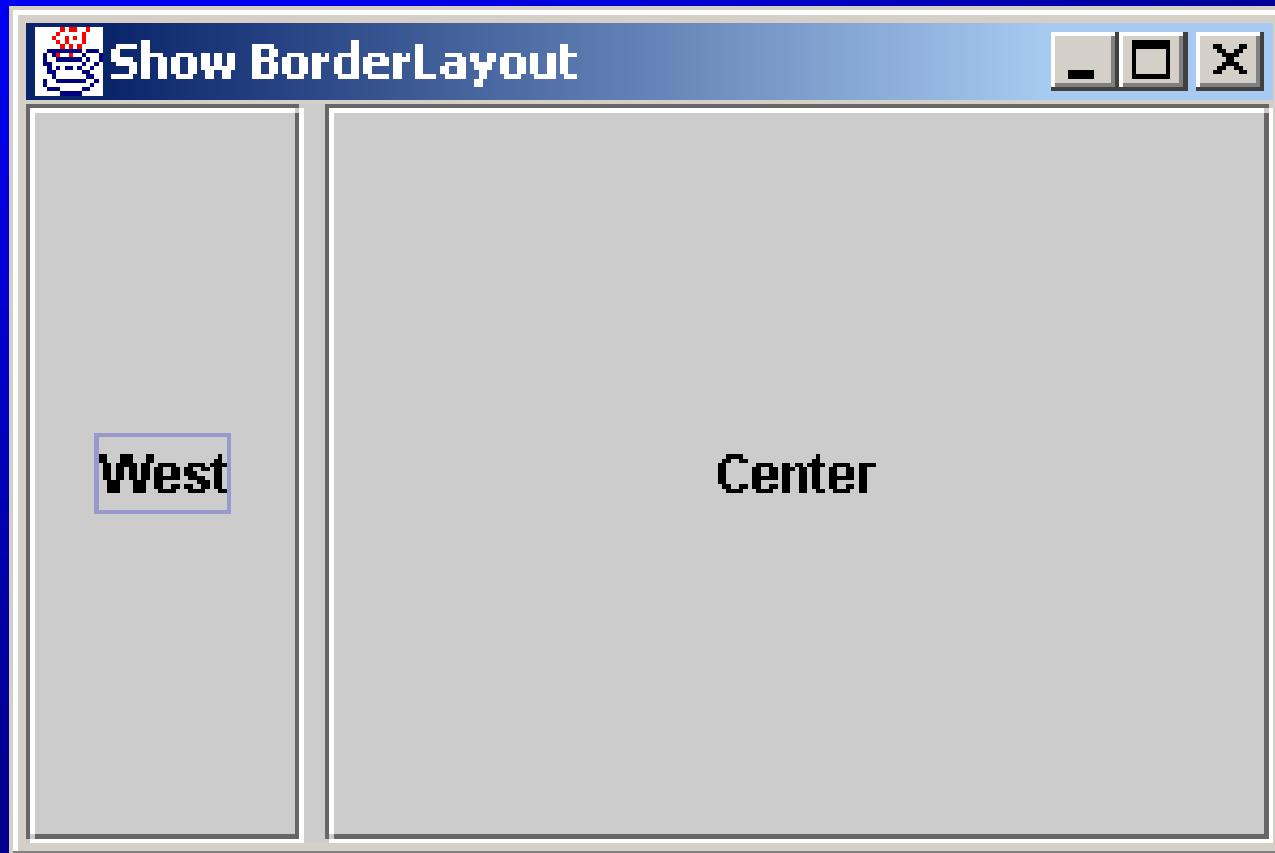
East and West only



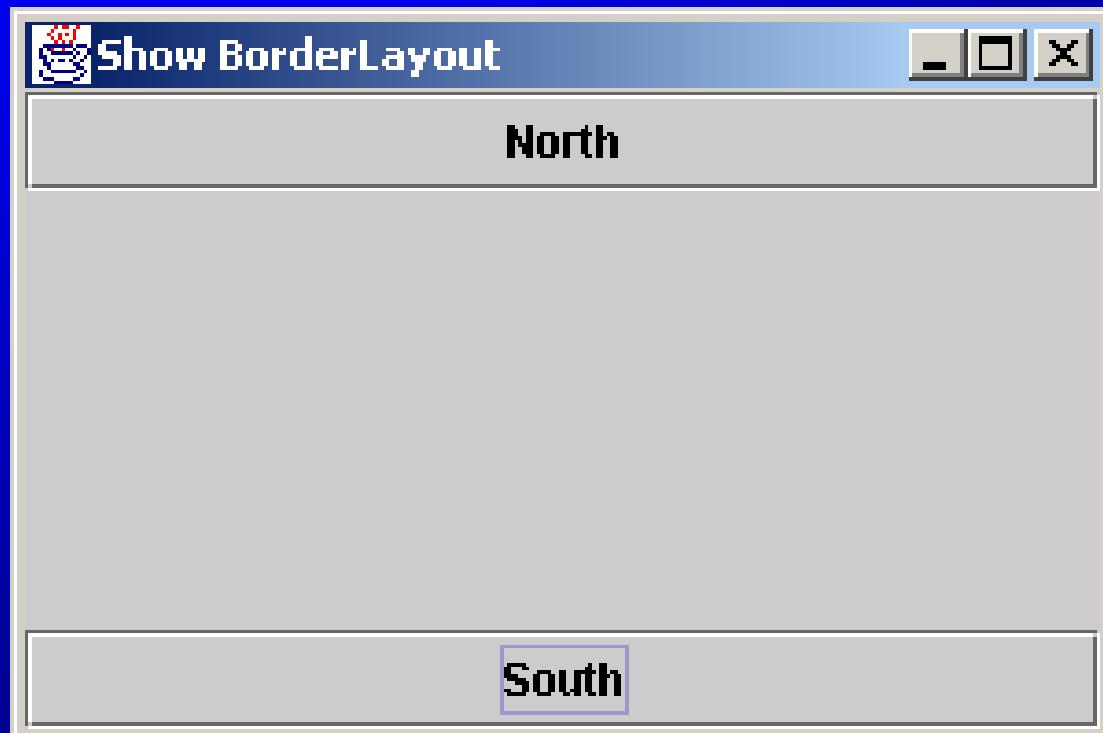
East and Center only



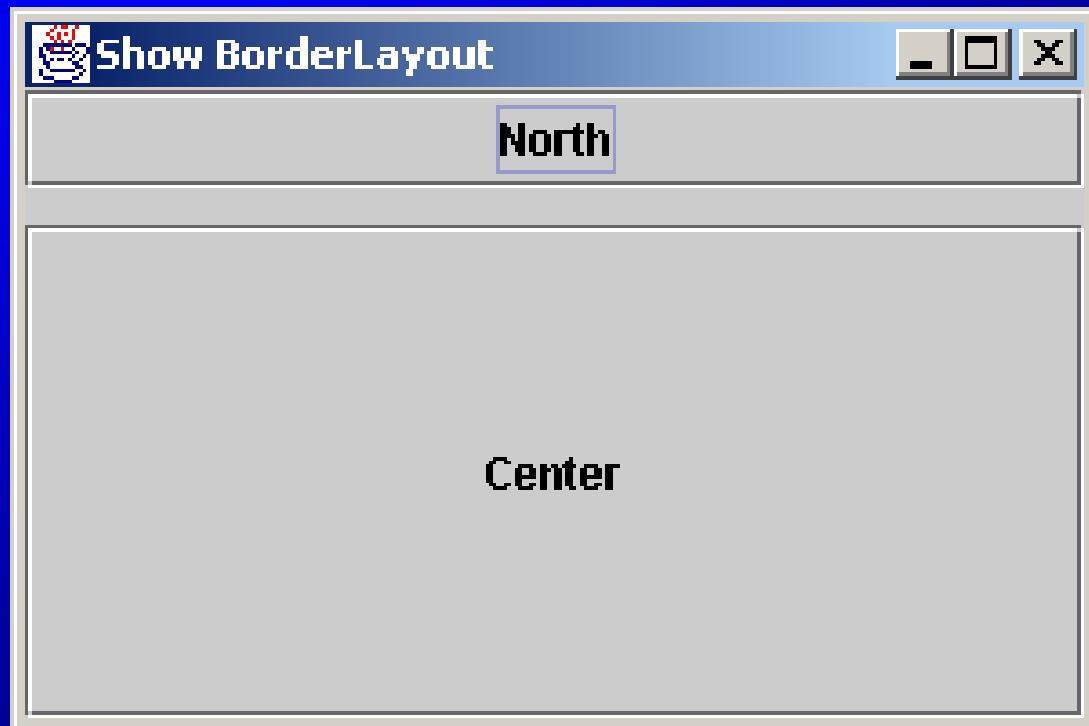
West and Center only



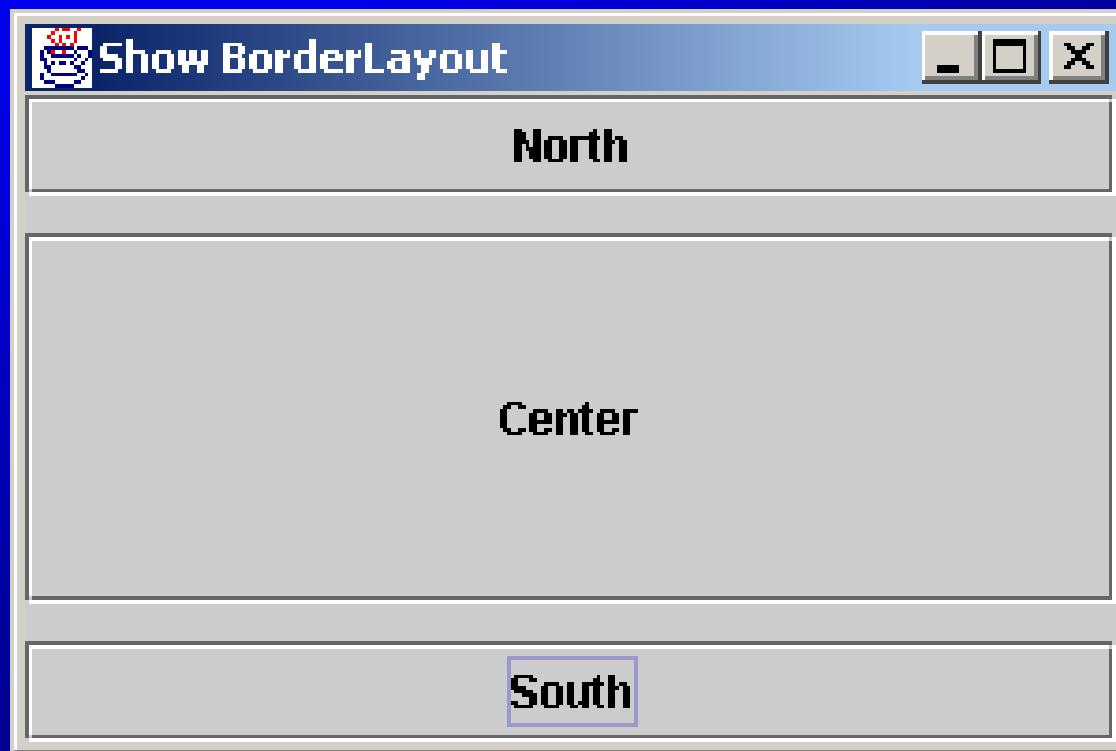
North and South only



North and Center only



North, South, and Center only



Note:

In using BorderLayout, you need to specify the location of the components

Without side area specification, center of stretch the un-occupied area

Absence of an index is interpreted as BorderLayout.CENTER



Using Panels as Containers

- ☞ Panels act as smaller containers for grouping user interface components.
- ☞ It is recommended that you place the user interface components in panels and place the panels in a frame. You can also place panels in a panel.

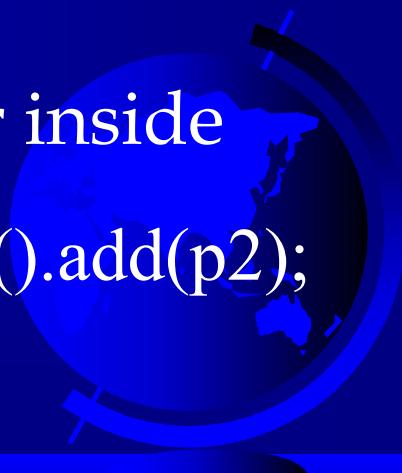


Using Panels as Containers

- ☞ Constructors:
 - JPanel p =new JPanel();
- ☞ Use add() method to add component to the Panel object
- ☞ Default JPanel uses FlowLayout
- ☞ Panels can be added inside a frame or inside another panel

 p1.getContentPane().add(p2);

f.getContentPane().add(p);



Example 8.4 Testing Panel

This example uses panels to organize components. The program creates a user interface for a Microwave oven.



```
// TestPanels.java: Use panels to group components
import java.awt.*;
import javax.swing.*;

public class TestPanels extends JFrame
{
    // Default constructor
    public TestPanels()
    {
        // Get the content pane of the frame
        Container container = getContentPane();

        // Set BorderLayout for the frame
        container.setLayout(new BorderLayout());
```



```
// Create panel p1 for the buttons and set GridLayout  
JPanel p1 = new JPanel();  
p1.setLayout(new GridLayout(4, 3));  
  
// Add buttons to the panel  
for (int i=1; i<=9; i++)  
{  
    p1.add(new JButton(" " + i));  
}  
  
p1.add(new JButton(" " + 0));  
p1.add(new JButton("Start"));  
p1.add(new JButton("Stop"));
```



```
// Create panel p2 to hold a text field and p1
JPanel p2 = new JPanel();
p2.setLayout(new BorderLayout());
p2.add(new TextField("Time to be displayed here"),
       BorderLayout.NORTH);
p2.add(p1, BorderLayout.CENTER);

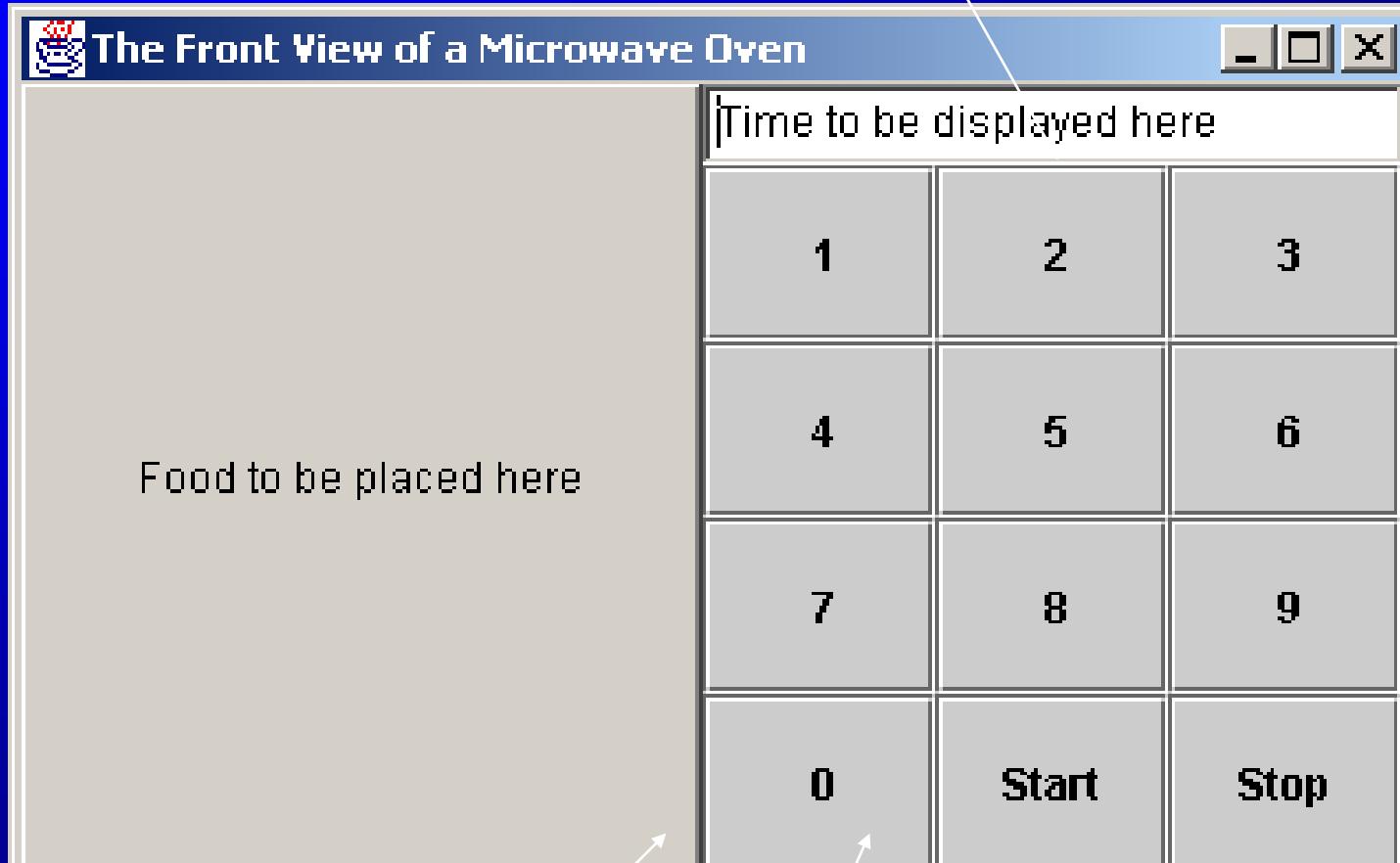
// Add p2 and a button to the frame
container.add(p2, BorderLayout.EAST);
container.add(new Button("Food to be placed here"),
             BorderLayout.CENTER);
}
```



```
// Main method
public static void main(String[] args)
{
    TestPanels frame = new TestPanels();
    frame.setTitle("The Front View of a Microwave Oven");
    // frame.setDefaultCloseOperation
    (JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 250);
    frame.setVisible(true);
}
```



p2 panel has textfield and p1 with BorderLayout



A button and p2 fill the frame

p1 panel with GridLayout

Another example of mixed panel layout

```
import java.awt.*;
import javax.swing.*;

public class DifferentLayout extends JFrame
{
    public DifferentLayout()
    {
        // Get the content pane of the frame
        Container container = getContentPane();

        container.setLayout(new GridLayout(2, 1, 5,5));
```



```
JPanel jptop =new JPanel();
jptop.setLayout(new FlowLayout());
```

```
JTextArea jta=new JTextArea("Hello");
```

```
jptop.add(jta);
```

```
JPanel jpbottom =new JPanel();
jpbottom.setLayout(new BorderLayout());
JPanel jp1=new JPanel();
JPanel jp2=new JPanel();
JPanel jp3=new JPanel();
```



```
jp1.setLayout(new FlowLayout());  
jp2.setLayout(new FlowLayout());  
jp3.setLayout(new FlowLayout());
```

```
jp1.add(new JButton("Button1"));  
jp2.add(new JButton("Button2"));  
jp3.add(new JButton("Button3"));
```

```
jpbottom.add(jp1,BorderLayout.WEST);  
jpbottom.add(jp2,BorderLayout.EAST);  
jpbottom.add(jp3,BorderLayout.CENTER);
```

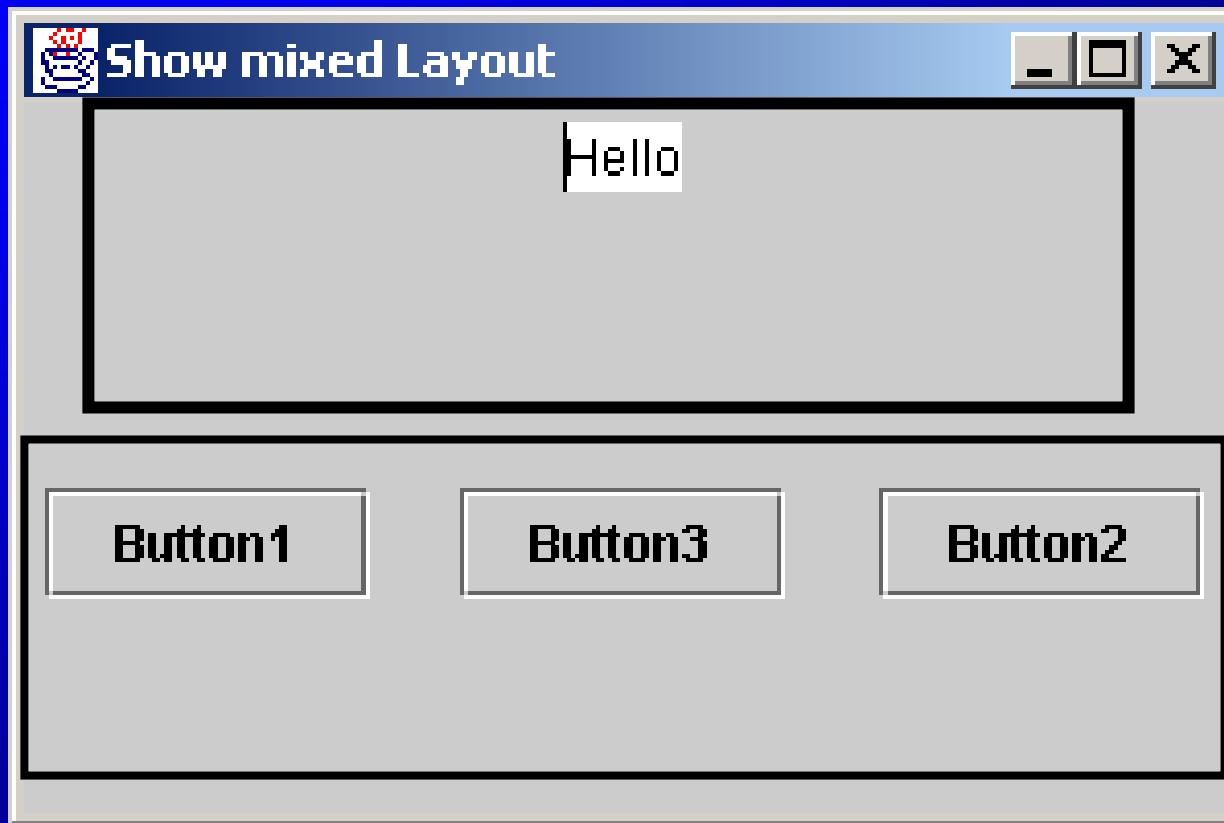


```
container.add(jptop);
container.add(jpbottom);
}

// Main method
public static void main(String[] args)
{
    DifferentLayout frame = new DifferentLayout();
    frame.setTitle("Show mixed Layout");
    frame.setSize(300, 200);
    frame.setVisible(true);
}
```



top panel has one textarea



bottom panel has three panels. Each of panel has one button

Drawing on Panels

JPanel is invisible. It is used to group components. In addition, it can be used to draw graphics (including text) and enable user interaction.

To draw in a panel, you create a new class that extends JPanel and override the paintComponent method to tell the panel how to draw things. That means you need to design your own JPanel subclass.

You can then display strings, draw geometric shapes, and view images on the panel.

The Color Class

```
Color c = new Color(r, g, b);
```

r, g, and b specify a color by its red, green, and blue components.

Example:

```
Color c = new Color(128, 100, 100);
```



Setting Colors

You can use the following methods to set the component's background and foreground colors:

`setBackground(Color c)`

`setForeground(Color c)`

Example:

```
setBackground(Color.yellow);  
setForeground(Color.red);
```



The Font Class

```
Font myFont = Font(name, style, size);
```

Example:

```
Font myFont = new Font("SansSerif ", Font.BOLD, 16);
```

```
Font myFont = new Font("Serif", Font.BOLD+Font.ITALIC, 12);
```



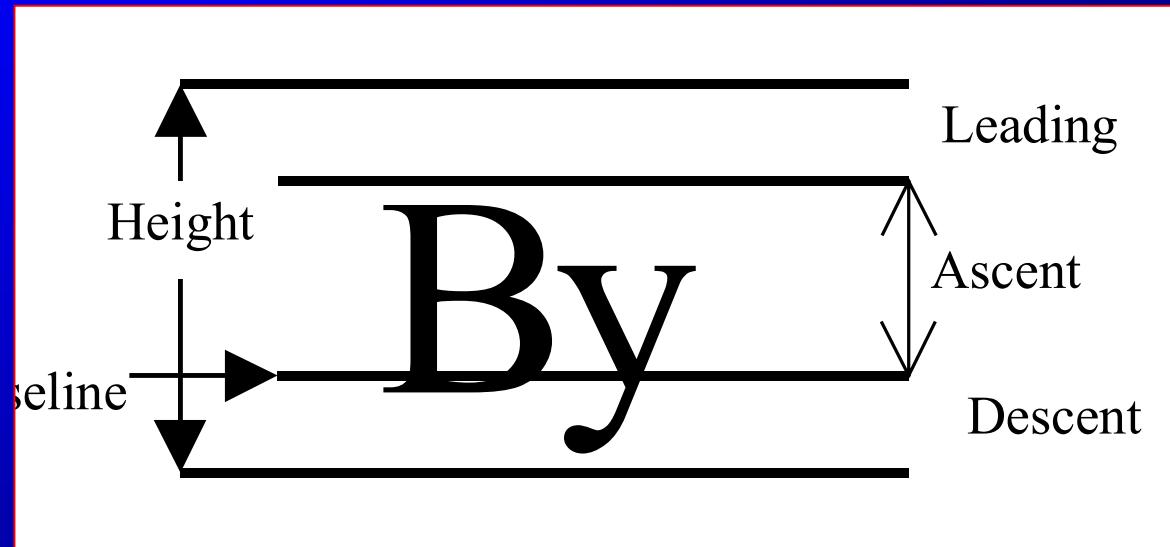
Setting Fonts

```
public void paint(Graphics g)
{
    Font myFont = new Font("Times", Font.BOLD, 16);
    g.setFont(myFont);
    g.drawString("Welcome to Java", 20, 40);

    //set a new font
    g.setFont(new Font("Courier", Font.BOLD+Font.ITALIC, 12));
    g.drawString("Welcome to Java", 20, 70);
}
```



The FontMetrics Class



Get FontMetrics

- ☞ `g.getFontMetrics(Font f);`
- or
- ☞ `g.getFontMetrics();`

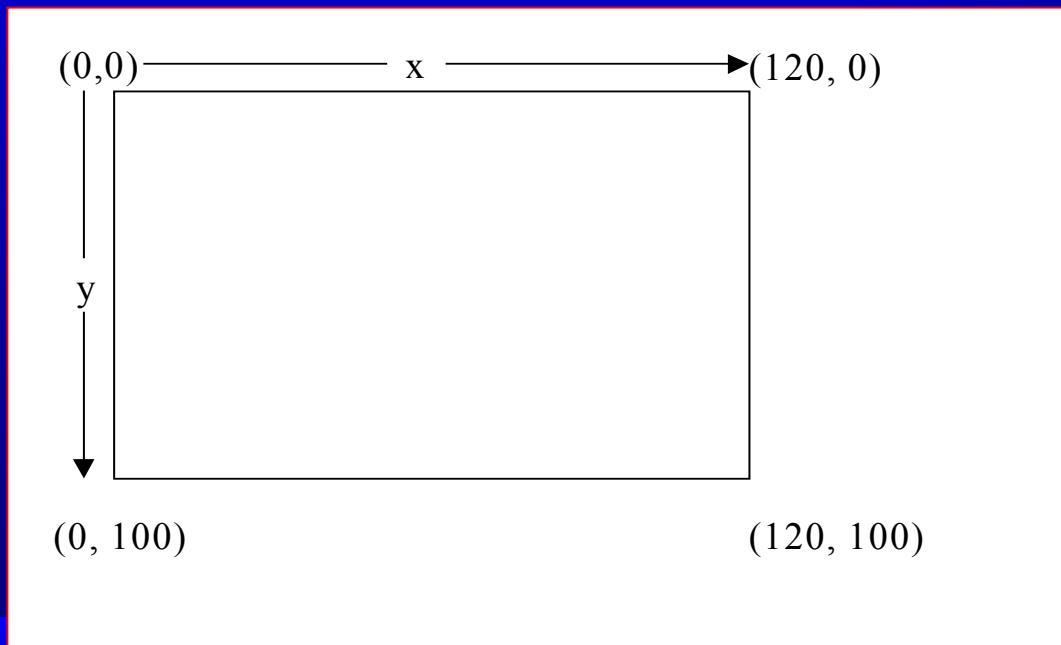
- ☞ `public int getAscent()`
- ☞ `public int getDescent()`
- ☞ `public int getLeading()`
- ☞ `public int getHeight()`
- ☞ `public int stringWidth(String str)`



Example 8.5

Using FontMetrics

- ☞ Objective: Display “Welcome to Java” in SansSerif 20-point bold, centered in the frame.



```
// MessagePanel.java: Display a message on a JPanel
import java.awt.Font;
import java.awt.FontMetrics;
import java.awt.Dimension;
import java.awt.Graphics;
import javax.swing.JPanel;

public class MessagePanel extends JPanel
{// Message to display
    private String message = "Welcome to Java";
    // (x, y) coordinates where the message is displayed
    private int xCoordinate = 20;
    private int yCoordinate = 20;
```



```
// Indicating whether the message is displayed in the center  
private boolean centered;
```

```
// Default constructor  
public MessagePanel()  
{  
}
```

```
// Constructor with a message parameter  
public MessagePanel(String message)  
{  
    this.message = message;  
}
```



```
public String getMessage()  
{  
    return message;  
}
```

```
public void setMessage(String message)  
{  
    this.message = message;  
}
```

```
public int getXCoordinate()  
{  
    return xCoordinate;  
}
```



```
public void setXCoordinate(int x)
{
    this.xCoordinate = x;
}
```

```
public int getYCoordinate()
{
    return yCoordinate;
}
```

```
public void setYCoordinate(int y)
{
    this.yCoordinate = y;
}
```



```
public boolean isCentered()
{
    return centered;
}
```

```
public void setCentered(boolean centered)
{
    this.centered = centered;
}
```



```
public void paintComponent(Graphics g)
{
    super.paintComponent(g);

    if (centered)
    {
        // Get font metrics for the current font
        FontMetrics fm = g.getFontMetrics();

        // Find the center location to display
        int w = fm.stringWidth(message); // Get the string width
        int h = fm.getAscent(); // Get the string height
        xCoordinate = (getWidth()-w)/2;
        yCoordinate = (getHeight()+h)/2;
    }
}
```



```
g.drawString(message, xCoordinate, yCoordinate);  
}
```

```
public Dimension getPreferredSize()  
{  
    return new Dimension(200, 100);  
}
```

```
public Dimension getMinimumSize()  
{  
    return new Dimension(200, 100);  
}  
}
```



```
// TestFontMetrics.java:  
//Draw a message at the center of a panel  
import java.awt.Font;  
import java.awt.FontMetrics;  
import java.awt.Graphics;  
import javax.swing.*;  
  
public class TestFontMetrics extends JFrame  
{  
    // Default constructor  
    public TestFontMetrics()  
    {  
        MessagePanel messagePanel =  
            new MessagePanel("Welcome to Java");
```

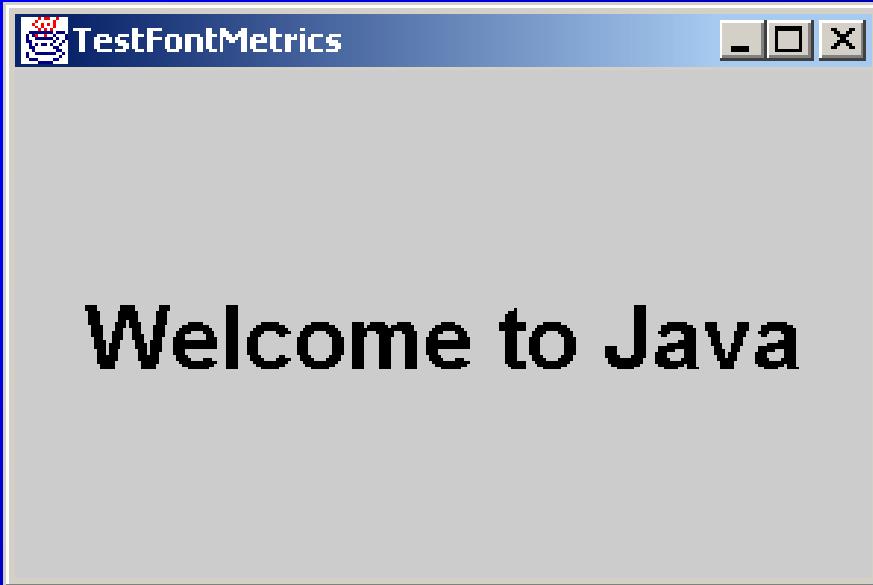


```
// Set font SansSerif 20-point bold  
messagePanel.setFont  
(new Font("SansSerif", Font.BOLD, 30));  
  
// Center the message  
messagePanel.setCentered(true);  
  
getContentPane().add(messagePanel);  
}
```



```
// Main method
public static void main(String[] args)
{
    TestFontMetrics frame = new TestFontMetrics();
    // frame.setDefaultCloseOperation
    (JFrame.EXIT_ON_CLOSE);
    frame.setSize(300, 200);
    frame.setTitle("TestFontMetrics");
    frame.setVisible(true);
}
```





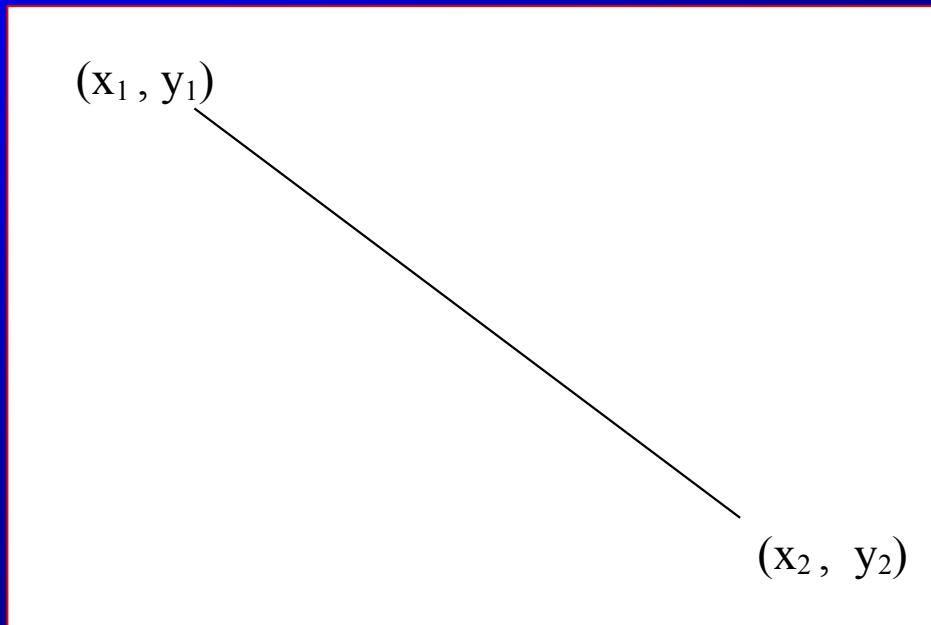
Drawing Geometric Figures

- ☞ Drawing Lines
- ☞ Drawing Rectangles
- ☞ Drawing Ovals
- ☞ Drawing Arcs
- ☞ Drawing Polygons



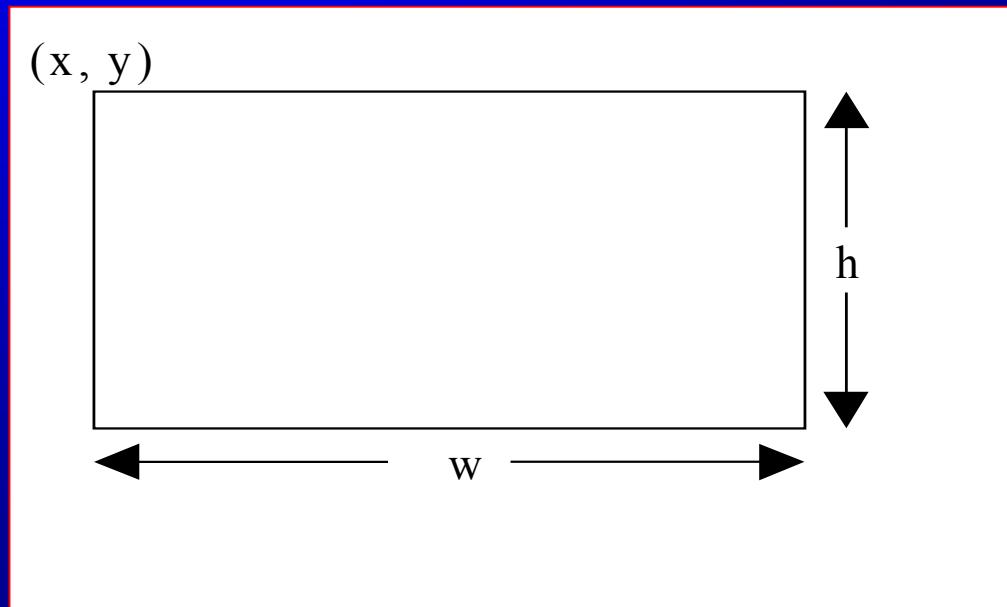
Drawing Lines

```
drawLine(x1, y1, x2, y2);
```



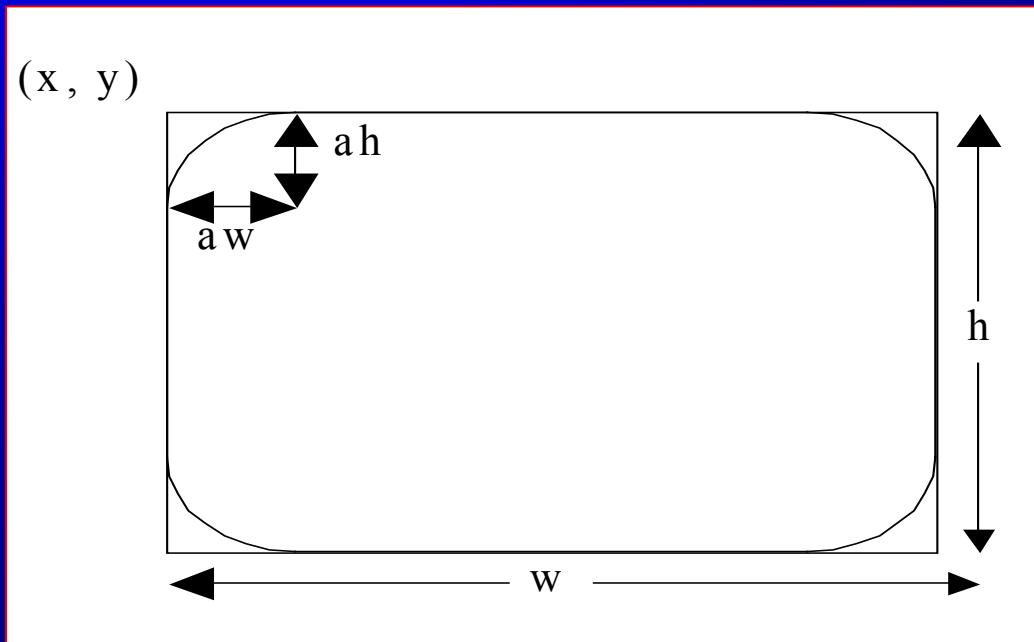
Drawing Rectangles

- ☞ `drawRect(x, y, w, h);`
- ☞ `fillRect(x, y, w, h);`



Drawing Rounded Rectangles

- ☞ `drawRoundRect (x, y, w, h, aw, ah);`
- ☞ `fillRoundRect (x, y, w, h, aw, ah);`



```
// TestRect.java: Demonstrate drawing rectangles
import java.awt.Graphics;
import java.awt.Color;
import javax.swing.JPanel;
import javax.swing.JFrame;

public class TestRect extends JFrame
{
    // Default constructor
    public TestRect()
    {
        setTitle("Show Rectangles");
        getContentPane().add(new RectPanel());
    }
}
```



```
// Main method
public static void main(String[] args)
{
    TestRect frame = new TestRect();
    frame.setDefaultCloseOperation
        (JFrame.EXIT_ON_CLOSE);
    frame.setSize(300,250);
    frame.setVisible(true);
}
```



```
class RectPanel extends JPanel
{
    public void paintComponent(Graphics g)
    {
        super.paintComponent(g);

        // Set new color
        g.setColor(Color.red);

        // Draw a rectangle
        g.drawRect(30, 30, 100, 100);

        // Draw a rounded rectangle
        g.drawRoundRect(140, 30, 100, 100, 60, 30);
    }
}
```

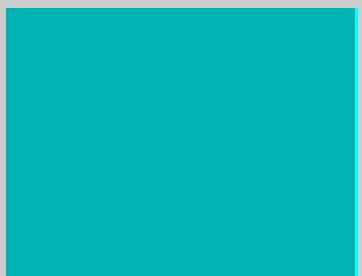
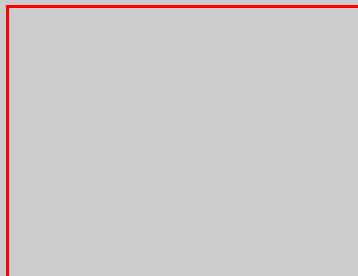


```
// Change the color to cyan  
g.setColor(Color.cyan);  
  
// Draw a 3D rectangle  
g.fill3DRect(30, 140, 100, 100, true);  
  
// Draw a raised 3D rectangle  
g.fill3DRect(140, 140, 100, 100, false);  
}  
}
```



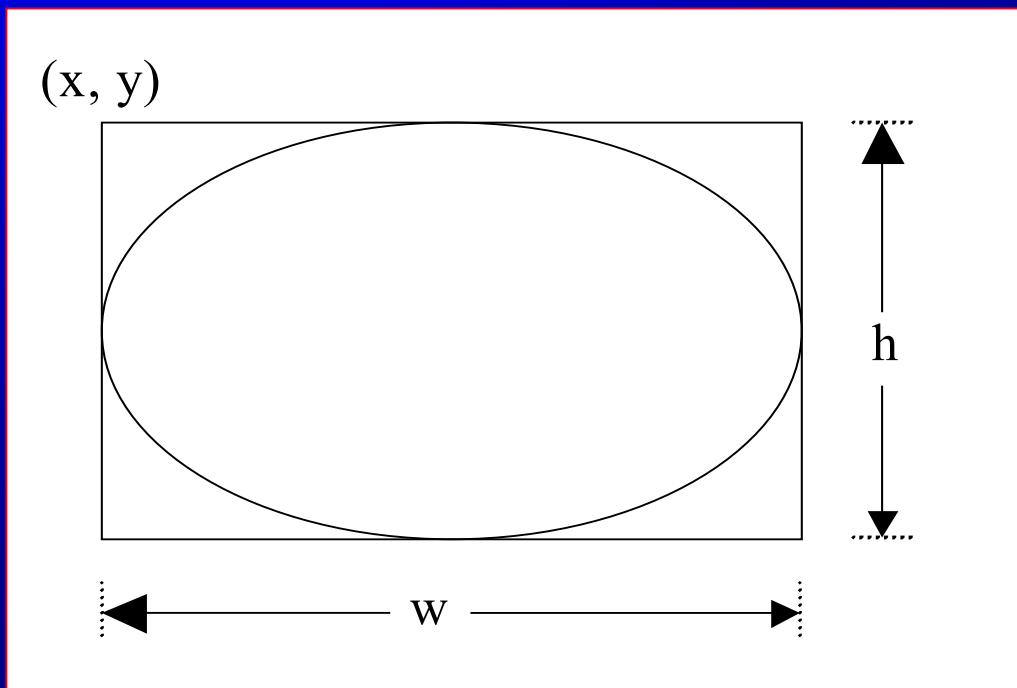


Show Rectangles



Drawing Ovals

- ☞ `drawOval(x, y, w, h);`
- ☞ `fillOval(x, y, w, h);`



```
// TestOvals.java: Demonstrate drawing ovals
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.Color;
import java.awt.Graphics;

public class TestOvals extends JFrame
{
    // Default constructor
    public TestOvals()
    {
        setTitle("Show Ovals");
        getContentPane().add(new OvalsPanel());
    }
}
```



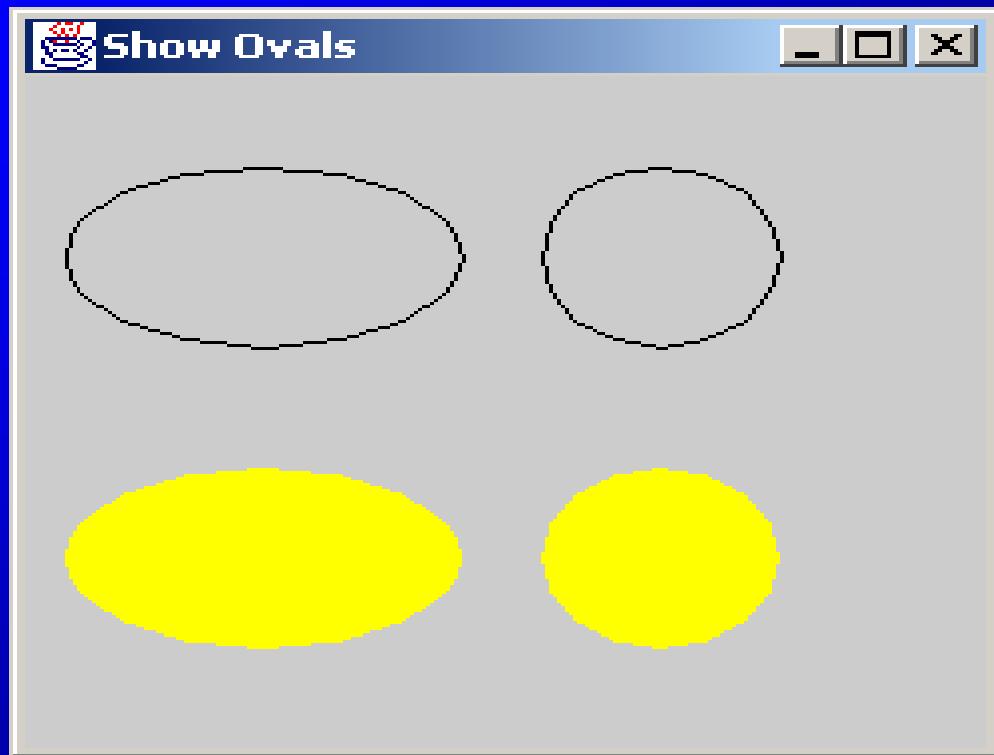
```
// Main method
public static void main(String[] args)
{
    TestOvals frame = new TestOvals();
    frame.setDefaultCloseOperation
        (JFrame.EXIT_ON_CLOSE);
    frame.setSize(250, 250);
    frame.setVisible(true);
}
```



```
// The class for drawing the ovals on a panel
class OvalsPanel extends JPanel
{
    public void paintComponent(Graphics g)
    {
        super.paintComponents(g);

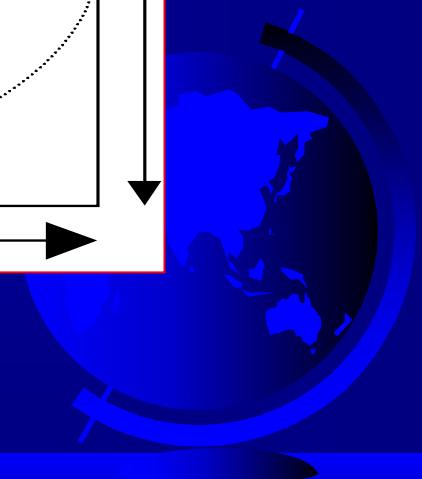
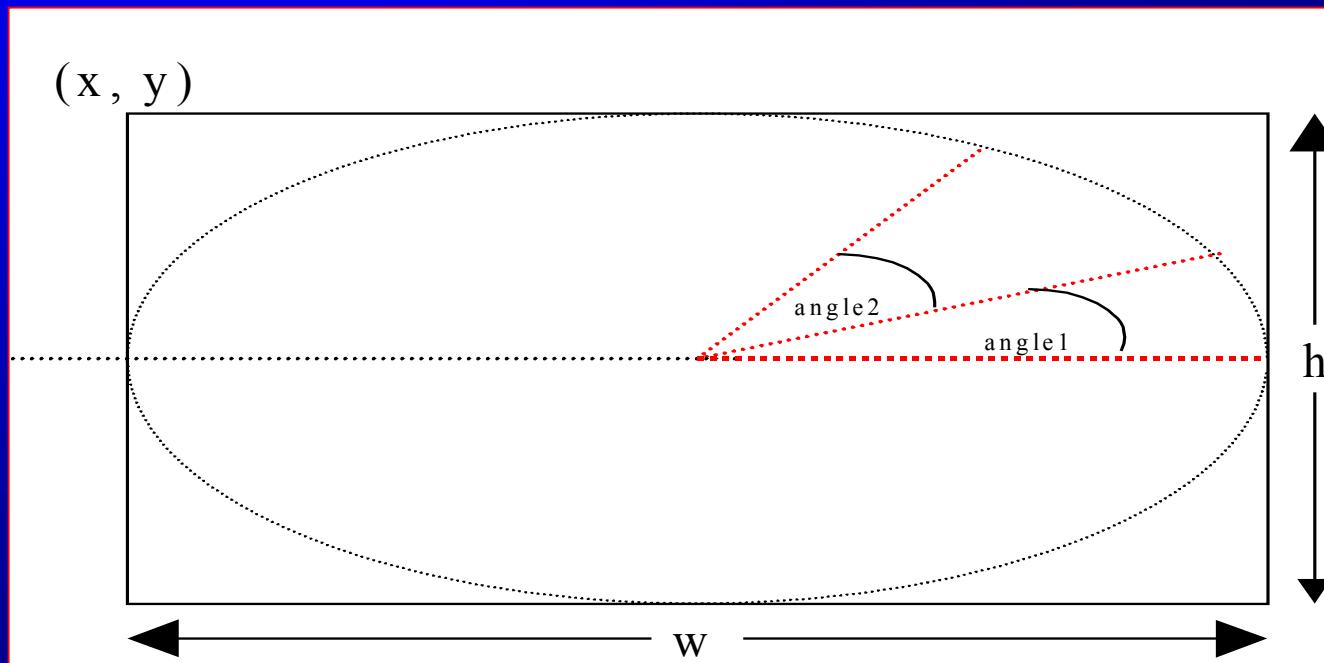
        g.drawOval(10, 30, 100, 60);
        g.drawOval(130, 30, 60, 60);
        g.setColor(Color.yellow);
        g.fillOval(10, 130, 100, 60);
        g.fillOval(130, 130, 60, 60);
    }
}
```





Drawing Arcs

- ☞ `drawArc(x, y, w, h, angle1, angle2);`
- ☞ `fillArc(x, y, w, h, angle1, angle2);`



```
// TestArcs.java: Demonstrate drawing arcs
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.Color;
import java.awt.Graphics;

public class TestArcs extends JFrame
{
    // Default constructor
    public TestArcs()
    {
        setTitle("Show Arcs");
        getContentPane().add(new ArcsPanel());
    }
}
```

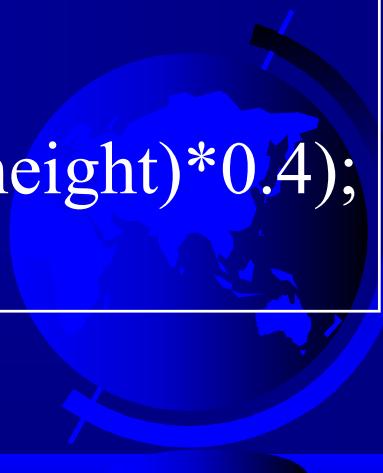


```
// Main method
public static void main(String[] args)
{
    TestArcs frame = new TestArcs();
    frame.setDefaultCloseOperation
        (JFrame.EXIT_ON_CLOSE);
    frame.setSize(250, 300);
    frame.setVisible(true);
}
```



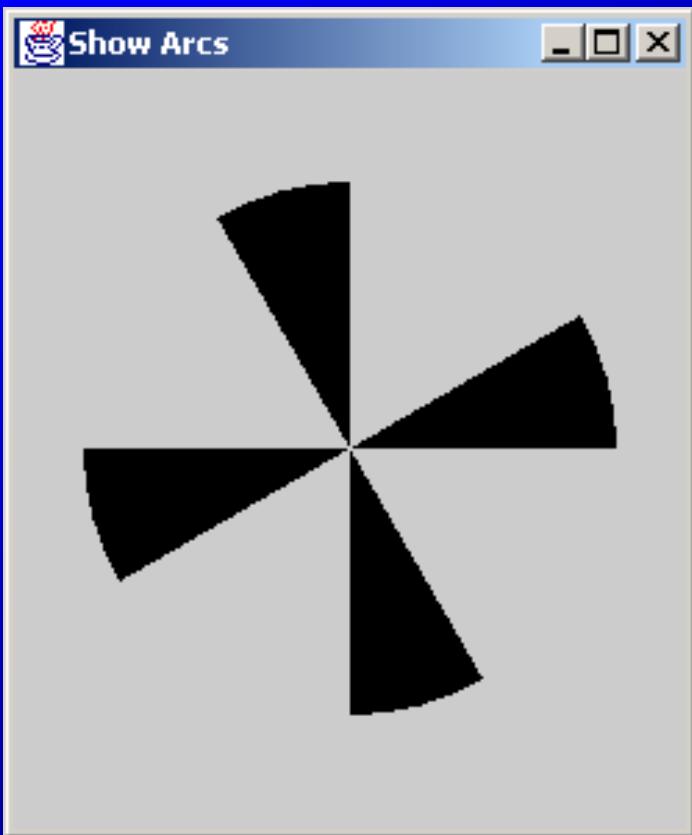
```
// The class for drawing arcs on a panel
class ArcsPanel extends JPanel
{
    // Draw four blazes of a fan
    public void paintComponent(Graphics g)
    {
        super.paintComponent(g);

        int xCenter = getWidth()/2;
        int yCenter = getHeight()/2;
        int radius =
            (int)(Math.min(getSize().width, getSize().height)*0.4);
```



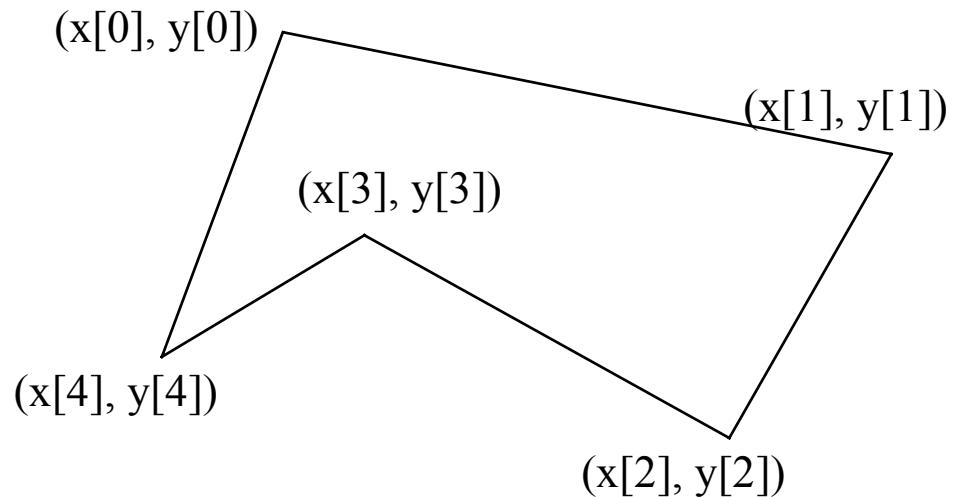
```
int x = xCenter - radius;  
int y = yCenter - radius;  
  
g.fillArc(x, y, 2*radius, 2*radius, 0, 30);  
g.fillArc(x, y, 2*radius, 2*radius, 90, 30);  
g.fillArc(x, y, 2*radius, 2*radius, 180, 30);  
g.fillArc(x, y, 2*radius, 2*radius, 270, 30);  
}  
}
```





Drawing Polygons

```
int[] x = {40, 70, 60, 45, 20};  
int[] y = {20, 40, 80, 45, 60};  
g.drawPolygon(x, y, x.length);  
g.fillPolygon(x, y, x.length);
```



```
// TestPolygon.java: Demonstrate drawing polygons
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.Graphics;
import java.awt.Polygon;

public class TestPolygon extends JFrame
{
    // Default constructor
    public TestPolygon()
    {
        setTitle("Show Polygon");
        getContentPane().add(new PolygonsPanel());
    }
}
```



```
// Main method
public static void main(String[] args)
{
    TestPolygon frame = new TestPolygon();
    frame.setDefaultCloseOperation
        (JFrame.EXIT_ON_CLOSE);
    frame.setSize(200, 250);
    frame.setVisible(true);
}
```



```
// Draw a polygon in the panel
class PolygonsPanel extends JPanel
{
    public void paintComponent(Graphics g)
    {
        super.paintComponent(g);

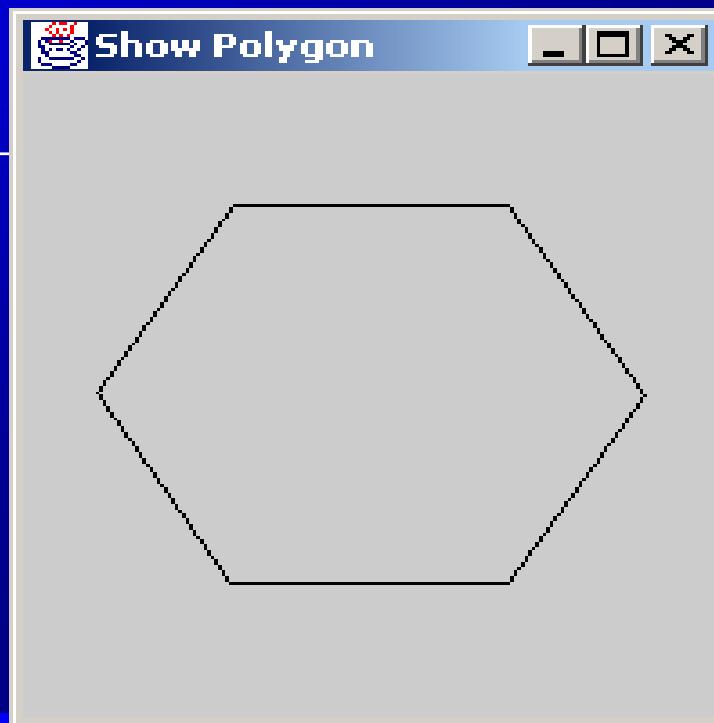
        int xCenter = getWidth()/2;
        int yCenter = getHeight()/2;
        int radius =
            (int)(Math.min(getSize().width, getSize().height)*0.4);

        // Create a Polygon object
        Polygon polygon = new Polygon();
```



```
// Add points to the polygon
polygon.addPoint(xCenter + radius, yCenter);
polygon.addPoint
    (((int)(xCenter + radius*Math.cos(2*Math.PI/6)),
    (int)(yCenter - radius*Math.sin(2*Math.PI/6)));
polygon.addPoint
    (((int)(xCenter + radius*Math.cos(2*2*Math.PI/6)),
    (int)(yCenter - radius*Math.sin(2*2*Math.PI/6)));
polygon.addPoint
    (((int)(xCenter + radius*Math.cos(3*2*Math.PI/6)),
    (int)(yCenter - radius*Math.sin(3*2*Math.PI/6)));
polygon.addPoint((int)
    (xCenter + radius*Math.cos(4*2*Math.PI/6)),
    (int)(yCenter - radius*Math.sin(4*2*Math.PI/6)));
```

```
    polygon.addPoint((int)
        (xCenter + radius*Math.cos(5*2*Math.PI/6)),
        (int)(yCenter - radius*Math.sin(5*2*Math.PI/6)));
    // Draw the polygon
    g.drawPolygon(polygon);
}
}
```



Example 8.6

Drawing a Clock

- ☞ Objective: Use drawing and trigonometric methods to draw a clock showing the specified hour, minute, and second in a frame.



```
// DisplayClock.java: Display a clock in a panel
import java.awt.*;
import javax.swing.*;

public class DisplayClock extends JFrame
{
    // Main method with three arguments:
    // args[0]: hour
    // args[1]: minute
    // args[2]: second
    public static void main(String[] args)
    {
        // Declare hour, minute, and second values
        int hour = 0;
```



```
int minute = 0;  
int second = 0;  
  
// Check usage and get hour, minute, second  
if (args.length > 3)  
{  
    System.out.println(  
        "Usage: java DisplayClock hour minute second");  
    System.exit(0);  
}  
else if (args.length == 3)  
{  
    hour = new Integer(args[0]).intValue();  
    minute = new Integer(args[1]).intValue();
```



```
second = new Integer(args[2]).intValue();
}
else if (args.length == 2)
{
    hour = new Integer(args[0]).intValue();
    minute = new Integer(args[1]).intValue();
}
else if (args.length == 1)
{
    hour = new Integer(args[0]).intValue();
}
```



```
// Create a frame to hold the clock  
DisplayClock frame = new DisplayClock();  
frame.setTitle("Display Clock");  
frame.getContentPane().add  
    (new DrawClock(hour, minute, second));  
// frame.setDefaultCloseOperation  
    (JFrame.EXIT_ON_CLOSE);  
frame.setSize(300, 350);  
frame.setVisible(true);  
}  
}
```



```
// DrawClock.java: Display a clock in JPanel
import java.awt.*;
import javax.swing.*;

public class DrawClock extends JPanel
{
    private int hour;
    private int minute;
    private int second;
    protected int xCenter, yCenter;
    protected int clockRadius;
```



```
// Construct a clock panel  
public DrawClock(int hour, int minute, int second)  
{  
    this.hour = hour;  
    this.minute = minute;  
    this.second = second;  
}
```

```
// Draw the clock  
public void paintComponent(Graphics g)  
{  
    super.paintComponent(g);
```



```
// Initialize clock parameters
clockRadius =
    (int)(Math.min(getSize().width, getSize().height)*0.7*0.5);
xCenter = getWidth()/2;
yCenter = getHeight()/2;

// Draw circle
g.setColor(Color.black);
g.drawOval(xCenter - clockRadius, yCenter - clockRadius
    2*clockRadius, 2*clockRadius);
g.drawString("12", xCenter-5, yCenter-clockRadius);
g.drawString("9", xCenter-clockRadius-10, yCenter+3);
g.drawString("3", xCenter+clockRadius, yCenter+3);
g.drawString("6", xCenter-3, yCenter+clockRadius+10);
```

```
// Draw second hand
int sLength = (int)(clockRadius*0.9);
int xSecond =
    (int)(xCenter +
        sLength*Math.sin(second*(2*Math.PI/60)));
int ySecond =
    (int)(yCenter -
        sLength*Math.cos(second*(2*Math.PI/60)));
g.setColor(Color.red);
g.drawLine(xCenter, yCenter, xSecond, ySecond);
// Draw minute hand
int mLength = (int)(clockRadius*0.75);
int xMinute =
    (int)(xCenter +
        mLength*Math.sin(minute*(2*Math.PI/60))));
```



```
int yMinute =  
    (int)(yCenter -  
        mLength*Math.cos(minute*(2*Math.PI/60)));  
g.setColor(Color.blue);  
g.drawLine(xCenter, yCenter, xMinute, yMinute);  
  
// Draw hour hand  
int hLength = (int)(clockRadius*0.6);  
int xHour = (int)(xCenter +  
    hLength*  
        Math.sin((hour+minute/60.0)*(2*Math.PI/12)));  
int yHour = (int)(yCenter -  
    hLength*  
        Math.cos((hour+minute/60.0)*(2*Math.PI/12)));
```



```
g.setColor(Color.green);
g.drawLine(xCenter, yCenter, xHour, yHour);

// Display current time in string
g.setColor(Color.red);
String time = "Hour: " + hour + " Minute: " + minute +
    " Second: " + second;
FontMetrics fm = g.getFontMetrics();
g.drawString(time, (getWidth() -
    fm.stringWidth(time))/2, yCenter+clockRadius+30);
}

}
```



C:\WINNT\System32\cmd.exe - java DisplayClock 2 30 4

The system cannot find the path specified.

D:\>dir

Volume in drive D is PKBACK# 001
Volume Serial Number is 1BF5-2E22

Directory of D:\

07/13/2001	01:16p	<DIR>	PJspring2002
11/29/2001	09:16a	1,228,393	JDBC.zip
11/29/2001	09:20a	2,546,254	txpeng450.zip
02/20/2002	03:43p	1,322	BOOTEX.LOG
		3 File(s)	3,775,969 bytes
		1 Dir(s)	50,364,416 bytes free

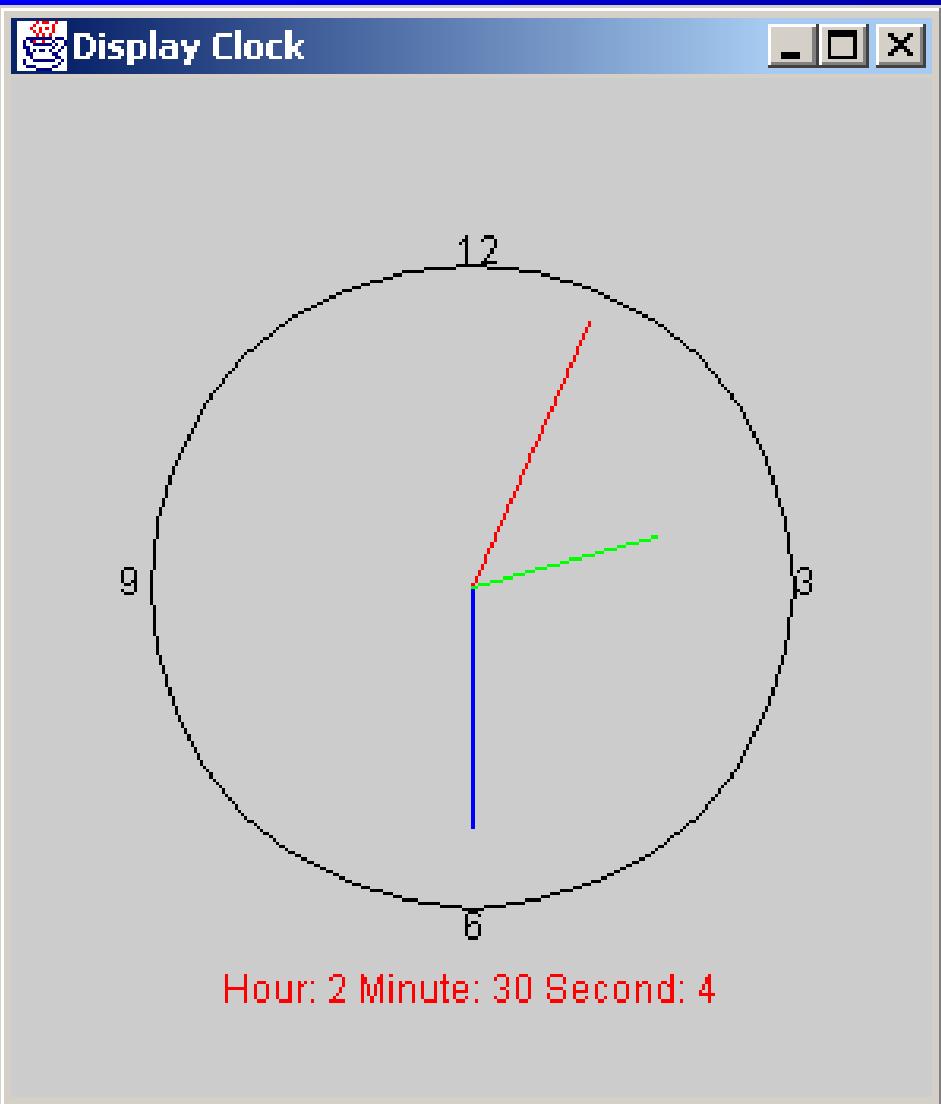
D:\>cd pjspring2002

D:\PJspring2002>cd book

D:\PJspring2002\book>cd ch08

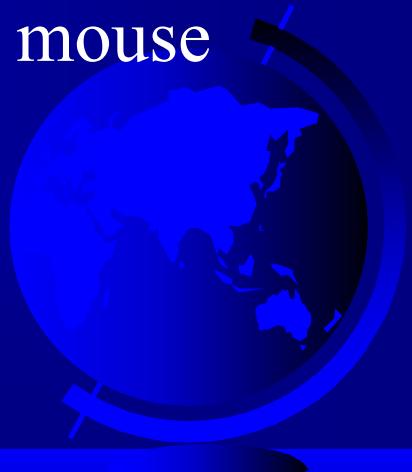
D:\PJspring2002\book\ch08>javac DisplayClock.java

D:\PJspring2002\book\ch08>java DisplayClock 2 30 4



Event-Driven Programming

- ☞ *Procedural programming* is executed in procedural order. Event it is a OOP.
- ☞ Java graphic programming is event-driven programming
- ☞ In *event-driven programming*, code is executed upon activation of events (a button click, mouse movement.



Events

- ☞ An *event* can be defined as a type of **signal** to the program that something has happened.
- ☞ The event is generated by external user actions such as mouse movements, mouse button clicks, and keystrokes, or by the operating system, such as a timer.
- ☞ The program should respond to or ignore the event generated by user



Events

- ☞ GUI component on which the event is generated is called "Source object"
- ☞ Button is a source object for clicking-button of the event class.
- ☞ Identify source object using getSource() method in EventObject class

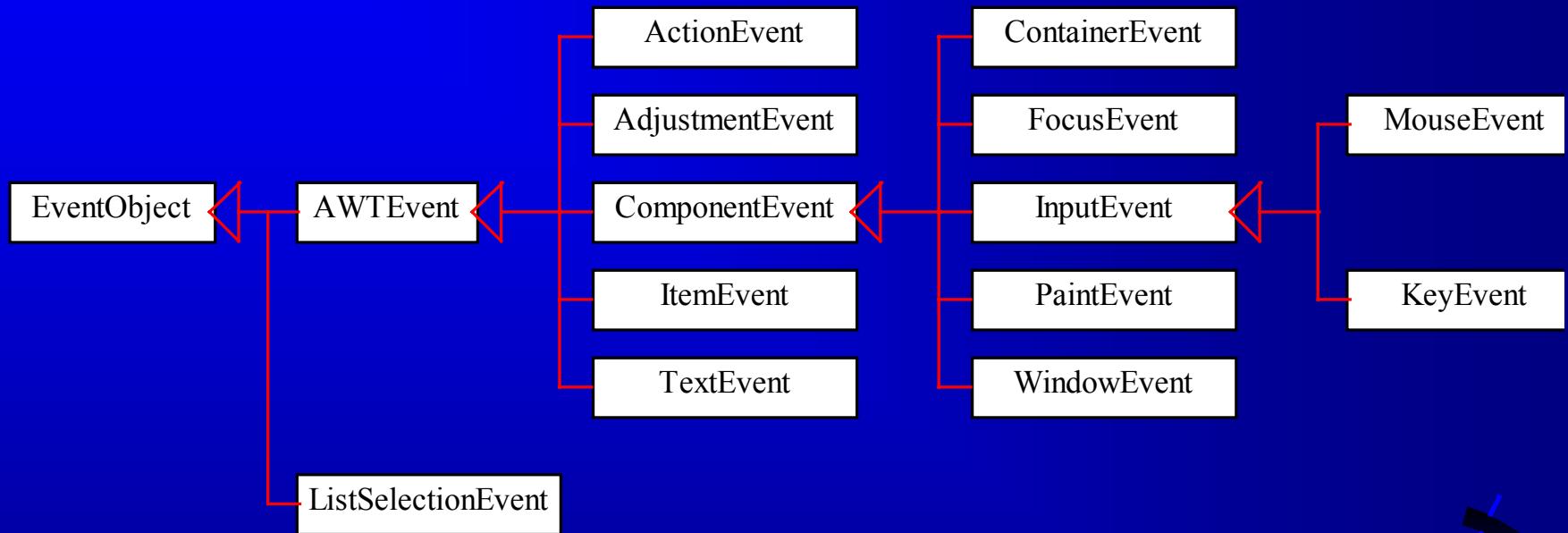


Event Information

- ☞ **id:** A number that identifies the event.
- ☞ **target:** The source component upon which the event occurred.
- ☞ **arg:** Additional information about the source components.
- ☞ **x, y coordinates:** The mouse pointer location when a mouse movement event occurred.
- ☞ **clickCount:** The number of consecutive clicks for the mouse events. For other events, it is zero.
- ☞ **when:** The time stamp of the event.
- ☞ **key:** The key that was pressed or released.



Event Classes



Selected User Actions

User Action	Source Object	Event Type Generated
Clicked on a button	JButton	ActionEvent
Changed text	JTextComponent	TextEvent
Double-clicked on a list item	JList	ActionEvent
Selected or deselected an item with a single click	JList	ItemEvent
Selected or deselected an item	JComboBox	ItemEvent

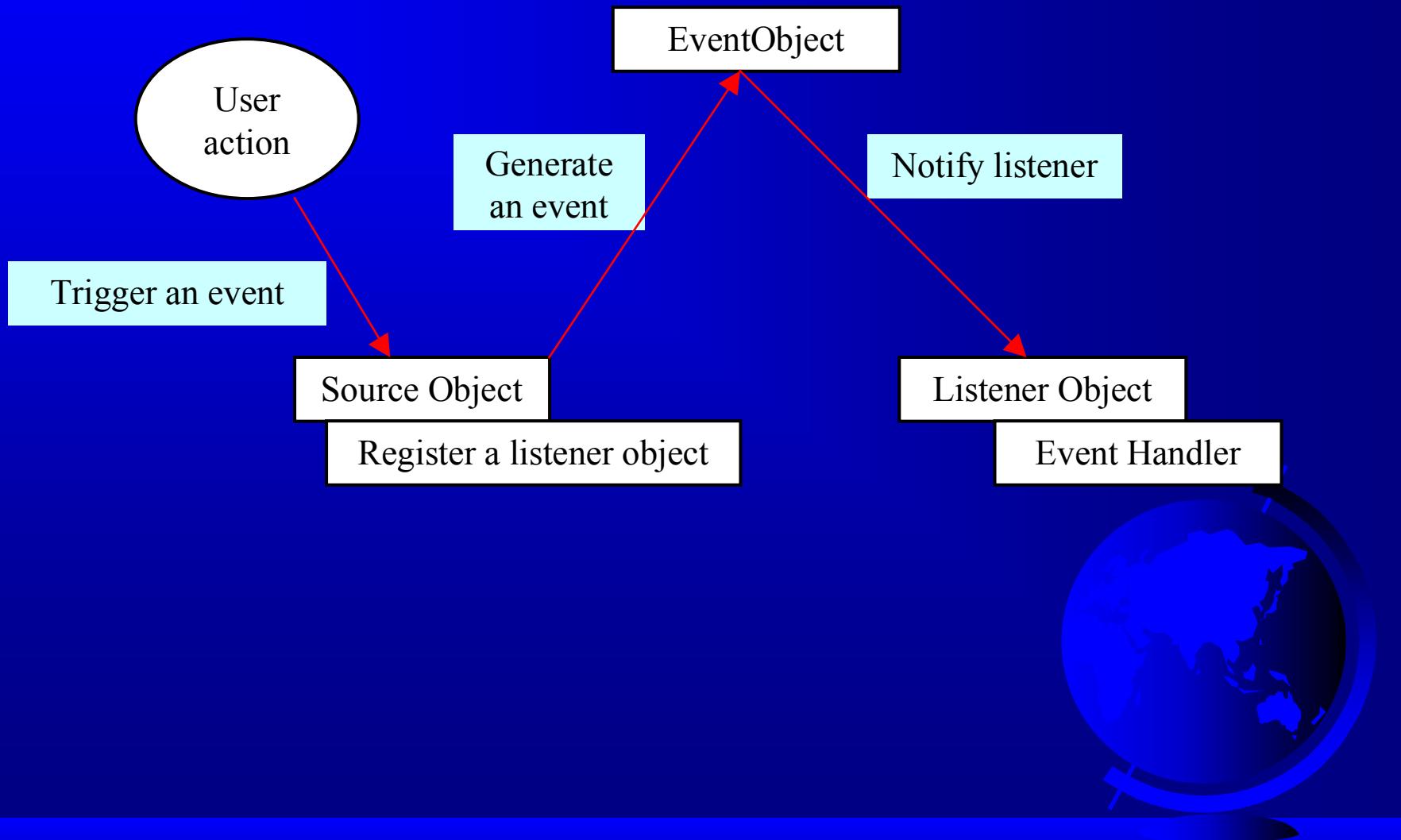
Details can be found in Table 8.1



Event Registration, Listening, and Handling



The Delegation Model



The Delegation Model

A JFrame object must register with a JButton object

```
jb.addActionListener(this);
```

1. When user click the button, the JButton object (jb) generates an ActionEvent and notifies the listener (frame) by invoking this method.
2. Registration methods are dependent on event type.
See the next Table



The Delegation Model

3. To become a listener, the listener must implement the standard handler, which is defined in the corresponding event-listener-interface.



Selected Event Handlers

Event Class

ActionEvent
ItemEvent
WindowEvent

Listener Interface

ActionListener
ItemListener
WindowListener

ContainerEvent

ContainerListener

Listener Methods (Handlers)

actionPerformed (ActionEvent)
itemStateChanged (ItemEvent)
windowClosing (WindowEvent)
windowOpened (WindowEvent)
windowIconified (WindowEvent)
windowDeiconified (WindowEvent)
windowClosed (WindowEvent)
windowActivated (WindowEvent)
windowDeactivated (WindowEvent)
componentAdded (ContainerEvent)
componentRemoved (ContainerEvent)

See Table 8.2



Example 8.7

Handling Simple Action Events

- ☞ Objective: Display two buttons OK and Cancel in the window. A message is displayed on the console to indicate which button is clicked, when a button is clicked.



```
// TestActionEvent.java: Create a Close button in the frame
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class TestActionEvent extends JFrame
    implements ActionListener
{
    // Create an object for "Close" button
    private JButton jbtOk = new JButton("OK");
    private JButton jbtCancel = new JButton("Cancel");
```

Object of TestActionEvent
can be a listener



```
// Default constructor
public TestActionEvent()
{
    // Set the window title
    setTitle("TestActionEvent");

    // Set FlowLayout manager to arrange the components
    // inside the frame
    getContentPane().setLayout(new FlowLayout());

    // Add buttons to the frame
    getContentPane().add(jbtOk);
    getContentPane().add(jbtCancel);
```



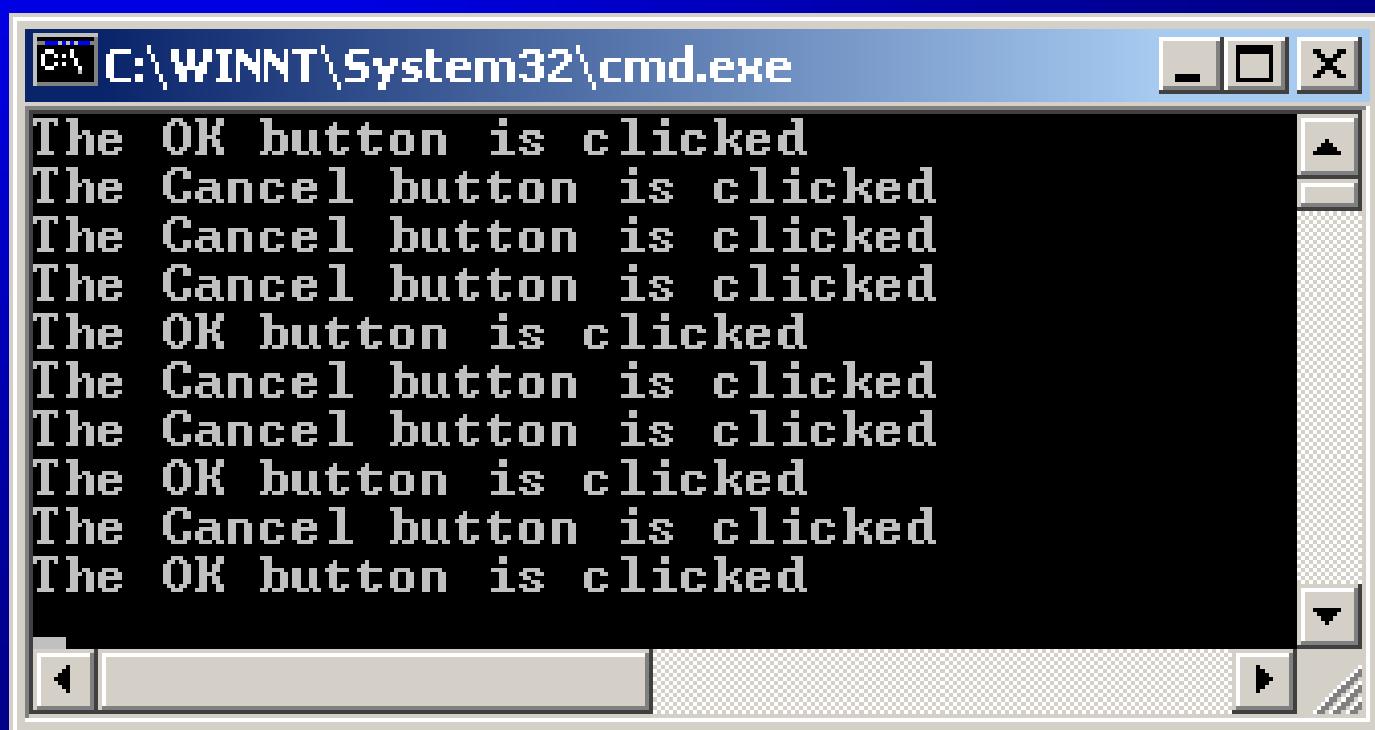
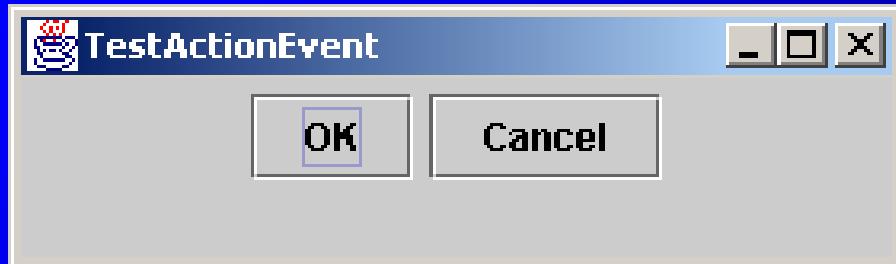
```
// Register listeners
jbtOk.addActionListener(this);
jbtCancel.addActionListener(this);
}

// Main method
public static void main(String[] args)
{
    TestActionEvent frame = new TestActionEvent();
    // frame.setDefaultCloseOperation
                (JFrame.EXIT_ON_CLOSE);
    frame.setSize(100, 80);
    frame.setVisible(true);
}
```



```
// This method will be invoked when a button is clicked.  
public void actionPerformed(ActionEvent e)  
{  
    if (e.getSource() == jbtOk)  
    {  
        System.out.println("The OK button is clicked");  
    }  
    else if (e.getSource() == jbtCancel)  
    {  
        System.out.println("The Cancel button is clicked");  
    }  
}
```





Example 8.8

Handling Window Events

- ☞ Objective: Demonstrate handling the window events. Any subclass of the Window class can generate the following window events: window opened, closing, closed, activated, deactivated, iconified, and deiconified. This program creates a frame, listens to the window events, and displays a message to indicate the occurring event.



```
// TestWindowEvent.java:  
//Create a frame to test window events  
import java.awt.*;  
import java.awt.event.*;  
import javax.swing.JFrame;  
  
public class TestWindowEvent extends JFrame  
    implements WindowListener  
{  
    // Main method  
    public static void main(String[] args)  
    {  
        TestWindowEvent frame = new TestWindowEvent();  
        // frame.setDefaultCloseOperation  
        // (JFrame.EXIT_ON_CLOSE);
```

```
frame.setTitle("Test Window Event");
frame.setSize(100, 80);
frame.setVisible(true);
}

// Default constructor
public TestWindowEvent()
{
    super();
    addWindowListener(this); // Register listener
}
```



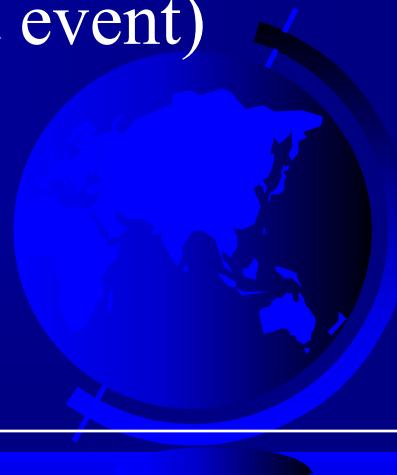
```
/**  
 * Handler for window deiconified event  
 * Invoked when a window is changed from a minimized  
 * to a normal state.  
 */  
public void windowDeiconified(WindowEvent event)  
{  
    System.out.println("Window deiconified");  
}
```



```
/**
```

```
* Handler for window iconified event  
* Invoked when a window is changed from a normal to a  
* minimized state. For many platforms, a minimized  
* window  
* is displayed as the icon specified in the window's  
* iconImage property.  
* @see Frame#setIconImage  
*/
```

```
public void windowIconified(WindowEvent event)  
{  
    System.out.println("Window iconified");  
}
```



```
/**  
 * Handler for window activated event  
 * Invoked when the window is set to be the user's  
 * active window, which means the window (or one of its  
 * subcomponents) will receive keyboard events.  
 */  
public void windowActivated(WindowEvent event)  
{  
    System.out.println("Window activated");  
}
```



```
/**  
 * Handler for window deactivated event  
 * Invoked when a window is no longer the user's active  
 * window, which means that keyboard events will no longer  
 * be delivered to the window or its subcomponents.  
 */
```

```
public void windowDeactivated(WindowEvent event)  
{  
    System.out.println("Window deactivated");  
}
```

```
/**  
 * Handler for window opened event  
 * Invoked the first time a window is made visible.  
 */
```



```
public void windowOpened(WindowEvent event)
{
    System.out.println("Window opened");
}

/**
 * Handler for window closing event
 * Invoked when the user attempts to close the window
 * from the window's system menu. If the program does not
 * explicitly hide or dispose the window while processing
 * this event, the window close operation will be cancelled.
 */

public void windowClosing(WindowEvent event)
{
    System.out.println("Window closing");
}
```



```
/**  
 * Handler for window closed event  
 * Invoked when a window has been closed as the result  
 * of calling dispose on the window.  
 */  
public void windowClosed(WindowEvent event)  
{  
    System.out.println("Window closed");  
}  
}
```





Test Window Event



C:\WINNT\System32\cmd.exe



```
Window activated  
Window opened  
Window deactivated  
Window activated  
Window iconified  
Window deactivated
```

-

