

## Chapter 6,13

# *Graphical User Interface (GUI) and Object- Oriented Design (OOD)*



## Objectives

- ☞ To distinguish simple GUI components.
- ☞ To describe the Java GUI API hierarchy.
- ☞ To create user interfaces using frames, panels, and simple UI components.
- ☞ To understand the role of layout managers.
- ☞ To use the FlowLayout, GridLayout, and BorderLayout managers to layout components in a container.
- ☞ To specify colors and fonts using the Color and Font classes.
- ☞ To use JPanel as subcontainers.



## Objectives cont.

- ☞ Discover events and event handlers
- ☞ To write programs to deal with ActionEvent.
- ☞ To write programs to deal with MouseEvent.
- ☞ To write programs to deal with KeyEvent
- ☞ Explore object-oriented design
- ☞ Learn how to identify objects, classes, and members of a class



## Creating GUI Objects

```
// Create a button with text OK
JButton jbtOK = new JButton("OK");
```

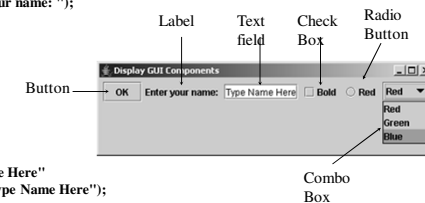
```
// Create a label with text "Enter your name: "
JLabel jlblName = new JLabel("Enter your name: ");
```

```
// Create a text field with text "Type Name Here"
JTextField jtfName = new JTextField("Type Name Here");
```

```
// Create a check box with text bold
JCheckBox jchkBold = new JCheckBox("Bold");
```

```
// Create a radio button with text red
JRadioButton jrbRed = new JRadioButton("Red");
```

```
// Create a combo box with choices red, green, and blue
JComboBox jcbColor = new JComboBox(new String[]{"Red", "Green", "Blue"});
```



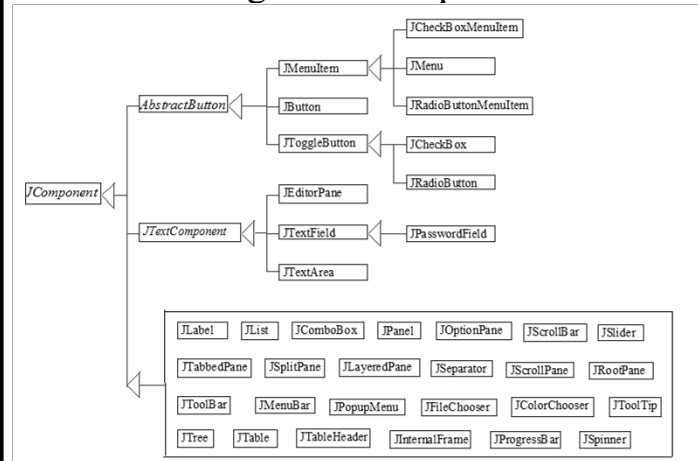
## Swing vs. AWT

So why do the GUI component classes have a prefix *J*? Instead of *JButton*, why not name it simply *Button*? In fact, there is a class already named *Button* in the *java.awt* package.

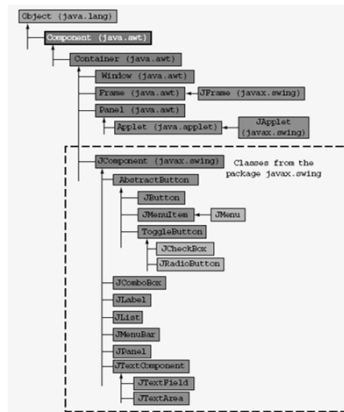
When Java was introduced, the GUI classes were bundled in a library known as the Abstract Windows Toolkit (AWT). For every platform on which Java runs, the AWT components are automatically mapped to the platform-specific components through their respective agents, known as *peers*. AWT is fine for developing simple graphical user interfaces, but not for developing comprehensive GUI projects. Besides, AWT is prone to platform-specific bugs because its peer-based approach relies heavily on the underlying platform. With the release of Java 2, the AWT user-interface components were replaced by a more robust, versatile, and flexible library known as *Swing components*. Swing components are painted directly on canvases using Java code, except for components that are subclasses of *java.awt.Window* or *java.awt.Panel*, which must be drawn using native GUI on a specific platform. Swing components are less dependent on the target platform and use less of the native GUI resource. For this reason, Swing components that don't rely on native GUI are referred to as *lightweight components*, and AWT components are referred to as *heavyweight components*.



## Swing GUI Components



## Inheritance Hierarchy of GUI Classes



## Frames

- ☞ Frame is a window that is not contained inside another window. Frame is the basis to contain other user interface components in Java GUI applications.
- ☞ The *JFrame* class can be used to create windows.
- ☞ For Swing GUI programs, use *JFrame* class to create windows.



## class JFrame

- ☞ GUI window instance created as instance of JFrame
- ☞ Provides various methods to control window attributes



9

## Methods Provided by the class JFrame

TABLE 6-1 Some Methods Provided by the class JFrame

### Method / Description / Example

```
public JFrame ()
//This is used when an object of type JFrame is
//instantiated and the window is created without any title.
//Example: JFrame myWindow = new JFrame();
//          myWindow is a window with no title
```

```
public JFrame(String s)
//This is used when an object of type JFrame is
//instantiated and the title specified by the string s.
//Example: JFrame myWindow = new JFrame("Rectangle");
//          myWindow is a window with the title Rectangle
```

```
public void setSize(int w, int h)
//Method to set the size of the window.
//Example: The statement
//          myWindow.setSize(400, 300);
//          sets the width of the window to 400 pixels and
//          the height to 300 pixels.
```



10

## Methods Provided by the class JFrame (continued)

```
public void setTitle(String s)
//Method to set the title of the window.
//Example: myWindow.setTitle("Rectangle");
//          sets the title of the window to Rectangle.
```

```
public void setVisible(boolean b)
//Method to display the window in the program. If the value of b is
//true, the window will be displayed on the screen.
//Example: myWindow.setVisible(true);
//          After this statement executes, the window will be shown
//          during program execution.
```

```
public void setDefaultCloseOperation(int operation)
//Method to determine the action to be taken when the user clicks
//on the window closing button, X, to close the window.
//Choices for the parameter operation are the named constants -
//EXIT_ON_CLOSE, HIDE_ON_CLOSE, DISPOSE_ON_CLOSE, and
//DO_NOTHING_ON_CLOSE. The named constant EXIT_ON_CLOSE is defined
//in the class JFrame. The last three named constants are defined in
//javax.swing.WindowConstants.
//Example: The statement
//          setDefaultCloseOperation(EXIT_ON_CLOSE);
//          sets the default close option of the window closing to close the
//          window and terminate the program when the user clicks the
//          window closing button, X.
```



11

## Two Ways to Create a Window

- ☞ First way
  - Declare object of type JFrame
  - Instantiate object
  - Use various methods to manipulate window
- ☞ Second way
  - Create class containing application program by extending definition of class JFrame
  - Utilizes mechanism of inheritance



12

## 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);
    }
}
```

MyFrame

Run

13

## Content Pane

- ☞ Inner area of GUI window (below title bar, inside border)
- ☞ To access content pane
  - Declare reference variable of type Container
  - Use method `getContentPane()` of class `JFrame`



14

## Constructors and Methods of the class Container

TABLE 13-2 Methods of the class Container

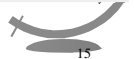
```
public Component add(Component comp)
    //Appends the specified component to the end of this container.

public Component add(Component comp, int index)
    //Adds the specified component to this container at the
    //position specified by index.

public void paint(Graphics g)
    //Paints the container with the graphics component specified by g.

public void update(Graphics g)
    //Invokes the paint method.

public void validate()
    //Validates this container and all of its subcomponents. The
    //method validate is used to cause a container to lay out its
    //subcomponents once more. Typically called after the components
    //it contains have been added to or modified.
```



15

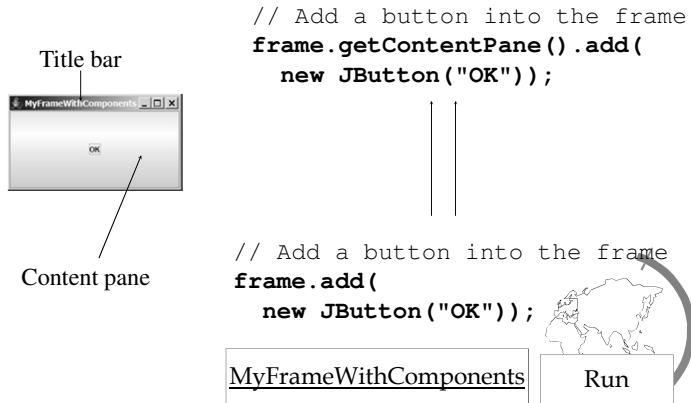
## Adding Components into a Frame

```
// Add a button into the frame
frame.getContentPane().add(
    new JButton("OK"));
```



16

## Content Pane Delegation in JDK 1.5



```
// Add a button into the frame
frame.getContentPane().add(
    new JButton("OK"));

// Add a button into the frame
frame.add(
    new JButton("OK"));
```

MyFrameWithComponents      Run

17

## class JLabel

- ☞ Labels: objects of particular class type
- ☞ class JLabel: used to create labels
- ☞ Label attributes
  - Title
  - Width
  - Height
- ☞ To create a label
  - Instantiate object of type JLabel
  - Modify attributes to control display of labels



18

## class JLabel (continued)

TABLE 6-3 Some Methods Provided by the class JLabel

Method / Description/ Example
<pre>public JLabel(String str) //Constructor to create a label with left-aligned text specified //by str. //Example: JLabel lengthL; //          lengthL = new JLabel("Enter the length:"); //          Creates the label lengthL with the title Enter the length:</pre>
<pre>public JLabel(String str, int align) //Constructor to create a label with the text specified by str. //The value of align can be any one of the following: // SwingConstants.LEFT, SwingConstants.RIGHT, // SwingConstants.CENTER //Example: // JLabel lengthL; // lengthL = new JLabel("Enter the length:"); //          SwingConstants.RIGHT); // The label lengthL is right aligned.</pre>
<pre>public JLabel(String t, Icon icon, int align) //Constructs a JLabel with both text and an icon. //The icon is to the left of the text.</pre>
<pre>public JLabel(Icon icon) //Constructs a JLabel with an icon.</pre>



19

## class JTextField

- ☞ Text fields: objects belonging to class JTextField
- ☞ To create text field
  - Declare reference variable of type JTextField
  - Instantiate object



20

## class JTextField (continued)

TABLE 6-4 Some Methods of the class JTextField

Method / Description
<code>public JTextField(int columns)</code> //Constructor to set the size of the text field.
<code>public JTextField(String str)</code> //Constructor to initialize the object with the text specified //by str.



21

## class JTextField (continued)

TABLE 6-4 Some Methods of the class JTextField (continued)

Method / Description
<code>public JTextField(String str, int columns)</code> //Constructor to initialize the object with the text specified //by str and to set the size of the text field.
<code>public void setText(String str)</code> //Method to set the text of the text field to the string specified //by str.
<code>public String getText()</code> //Method to return the text contained in the text field.
<code>public void setEditable(boolean b)</code> //If the value of the boolean variable b is false, the user cannot //type in the text field. //In this case, the text field is used as a tool to display //the result.
<code>public void addActionListener(ActionListener obj)</code> //Method to register a listener object to a JTextField.



22

## class JButton

- ☞ Provided to create buttons in Java
- ☞ To create button
  - Same technique as creating JLabel and JTextField



23

## class JButton (continued)

TABLE 6-5 Commonly Used Methods of the class JButton

Method / Description
<code>public JButton(Icon ic)</code> //Constructor to initialize the button object with the icon //specified by ic.
<code>public JButton(String str)</code> //Constructor to initialize the button object to the text specified //by str.
<code>public JButton(String str, Icon ic)</code> //Constructor to initialize the button object to the text specified //by str and the icon specified by ic.
<code>public void setText(String str)</code> //Method to set the text of the button to the string specified by str.
<code>public String getText()</code> //Method to return the text contained in the button.
<code>public void addActionListener(ActionListener obj)</code> //Method to register a listener object to the button object.



24

## Class ButtonGroup

This class is used to create a multiple-exclusion scope for a set of buttons. Creating a set of buttons with the same ButtonGroup object means that turning "on" one of those buttons turns off all other buttons in the group.

```
private ButtonGroup Group = new ButtonGroup();  
Group.add(JButton);  
Group.add(JButton);
```



25

## Method Summary

Void	<b>add</b> (AbstractButton b) Adds the button to the group.
Void	<b>clearSelection</b> () Clears the selection such that none of the buttons in the ButtonGroup are selected.
int	<b>getButtonCount</b> () Returns the number of buttons in the group.
ButtonModel	<b>getSelection</b> () Returns the model of the selected button.
boolean	<b>isSelected</b> (ButtonModel m) Returns whether a ButtonModel is selected.
Void	<b>remove</b> (AbstractButton b) Removes the button from the group.
Void	<b>setSelected</b> (ButtonModel m, boolean b) Sets the selected value for the ButtonModel.



26

## Layout Managers

- ☞ Java's layout managers provide a level of abstraction to automatically map your user interface on all window systems.
- ☞ The UI components are placed in containers. Each container has a layout manager to arrange the UI components within the container.
- ☞ Layout managers are set in containers using the `setLayout(LayoutManager)` method in a container.



27

## Kinds of Layout Managers

- ☞ FlowLayout
- ☞ GridLayout
- ☞ BorderLayout
- ☞ There are Several other layout managers



28

## The FlowLayout Class

java.awt.FlowLayout	
-alignment: int	
-hgap: int	
-vgap: int	
<hr/>	
+FlowLayout()	
+FlowLayout(alignment: int)	
+FlowLayout(alignment: int, hgap: int, vgap: int)	

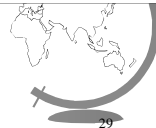
The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

The alignment of this layout manager (default: CENTER).  
The horizontal gap of this layout manager (default: 5 pixels).  
The vertical gap of this layout manager (default: 5 pixels).

Creates a default FlowLayout manager.

Creates a FlowLayout manager with a specified alignment.

Creates a FlowLayout manager with a specified alignment, horizontal gap, and vertical gap.



29

## FlowLayout Example

Write a program that adds three labels and text fields into the content pane of a frame with a FlowLayout manager.

ShowFlowLayout

Run



30

## The GridLayout Class

java.awt.GridLayout	
-rows: int	
-columns: int	
-hgap: int	
-vgap: int	
<hr/>	
+GridLayout()	
+GridLayout(rows: int, columns: int)	
+GridLayout(rows: int, columns: int, hgap: int, vgap: int)	

The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

The number of rows in this layout manager (default: 1).

The number of columns in this layout manager (default: 1).

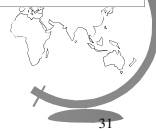
The horizontal gap of this layout manager (default: 0).

The vertical gap of this layout manager (default: 0).

Creates a default GridLayout manager.

Creates a GridLayout with a specified number of rows and columns.

Creates a GridLayout manager with a specified number of rows and columns, horizontal gap, and vertical gap.



31

## GridLayout Example

Write a program using GridLayout manager to display the labels and text fields.

ShowGridLayout

Run



32



## The BorderLayout Manager

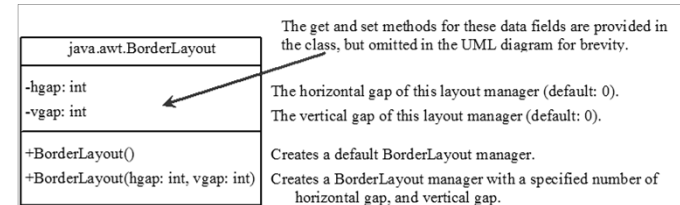
The BorderLayout manager divides the container into five areas: East, South, West, North, and Center. Components are added to a BorderLayout by using the add method.

`add(Component, constraint)`, where constraint is `BorderLayout.EAST`, `BorderLayout.SOUTH`, `BorderLayout.WEST`, `BorderLayout.NORTH`, or `BorderLayout.CENTER`.



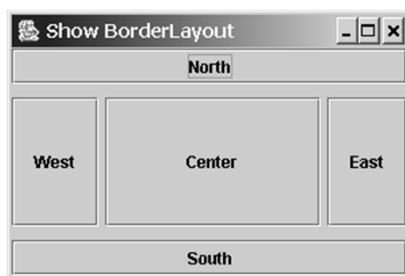
33

## The BorderLayout Class



34

## BorderLayout Example



Show BorderLayout

Run



35

## Using Panels as Sub-Containers

- ☞ Panels act as sub-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.
- ☞ To add a component to JFrame, you actually add it to the content pane of JFrame. To add a component to a panel, you add it directly to the panel using the add method.



36

## Creating a JPanel

You can use `new JPanel()` to create a panel with a default `FlowLayout` manager or `new JPanel(LayoutManager)` to create a panel with the specified layout manager. Use the `add(Component)` method to add a component to the panel. For example,

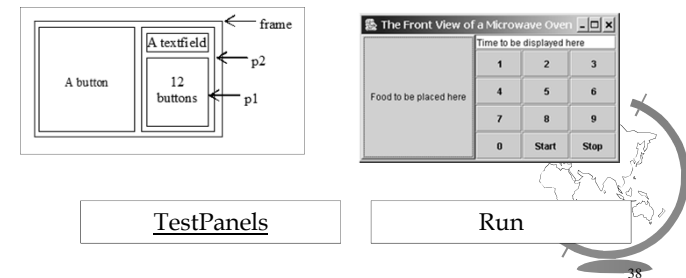
```
JPanel p = new JPanel();  
p.add(new JButton("OK"));
```



37

## Testing Panels Example

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



38

## The Color Class

You can set colors for GUI components by using the `java.awt.Color` class. Colors are made of red, green, and blue components, each of which is represented by a byte value that describes its intensity, ranging from 0 (darkest shade) to 255 (lightest shade). This is known as the *RGB model*.

```
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(228, 100, 255);
```



39

## Constructors of the class Color

TABLE 13-5 Some Constructors and Methods of the class `Color`

```
Color(int r, int g, int b)  
//Constructor  
//Creates a Color object with the red value r, green value g,  
//and blue value b. In this case, r, g, and b can be  
//between 0 and 255.  
//Example: new Color(0, 255, 0)  
// creates a color with no red or blue component  
  
Color(int rgb)  
//Constructor  
//Creates a Color object with the red value r, green value g,  
//and blue value b; RGB value consisting of the red component  
//in bits 16-23, the green component in bits 8-15, and the  
//blue component in bits 0-7.  
//Example: new Color(255)  
// creates a color with no red or green component.
```

40

## Constructors of the class Color (continued)

```
Color(float r, float g, float b)
//Constructor
//Creates a Color object with the red value r, green value g,
//and blue value b. In this case, r, g, and b can be between 0
//and 1.0.
//Example: new Color(1.0, 0, 0)
// creates a color with no green or blue component.

public Color brighter()
//Returns a Color that is brighter.

public Color darker()
//Returns a Color that is darker.

public boolean equals(Object o)
//Returns true if the color of this object is the same as the
//color of the object o; false otherwise.

public int getBlue()
//Returns the value of the blue component.

public int getGreen()
//Returns the value of the green component.

public int getRed()
//Returns the value of the red component.

public int getRGB()
//Returns the RGB value.

public String toString()
//Returns a string with the information about the color.
```

41

## Constants Defined in the class Color

TABLE 13-6 Constants Defined in the class Color

Color.black: (0, 0, 0)	Color.magenta: (255, 0, 255)
Color.blue: (0, 0, 255)	Color.orange: (255, 200, 0)
Color.cyan: (0, 255, 255)	Color.pink: (255, 175, 175)
Color.darkGray: (64, 64, 64)	Color.red: (255, 0, 0)
Color.gray: (128, 128, 128)	Color.white: (255, 255, 255)
Color.green: (0, 255, 0)	Color.yellow: (255, 255, 0)
Color.lightGray: (192, 192, 192)	

42

## Standard Colors

Thirteen standard colors (black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow) are defined as constants in java.awt.Color.

The standard color names are constants, but they are named as variables with lowercase for the first word and uppercase for the first letters of subsequent words. Thus the color names violate the Java naming convention. Since JDK 1.4, you can also use the new constants: BLACK, BLUE, CYAN, DARK\_GRAY, GRAY, GREEN, LIGHT\_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE, and YELLOW.

43

## Setting Colors

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

```
setBackground(Color c)
setForeground(Color c)
```

Example:

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

44

## class Font

- ☞ Shows text in different fonts
- ☞ Contained in package java.awt
- ☞ Available fonts
  - Serif/SanSerif
  - Monospaced
  - Dialog/DialogInput
- ☞ Arguments for constructor
  - String specifying the Font face name
  - int value specifying Font style
  - int value specifying Font size
    - ◆ Expressed in points (72 points = 1 inch)



## The Font Class

### Font Names

Standard font names that are supported in all platforms are: SansSerif, Serif, Monospaced, Dialog, or DialogInput.

### Font Style

Font.PLAIN (0), Font.BOLD (1), Font.ITALIC (2), and Font.BOLD + Font.ITALIC (3)

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

### Example:

```
Font myFont = new Font("SansSerif ", Font.BOLD, 16);
Font myFont = new Font("Serif", Font.BOLD+Font.ITALIC, 12);

JButton jbtOK = new JButton("OK");
jbtOK.setFont(myFont);
```



## Constructors and Methods of the class Font

TABLE 13-4 Some Constructors and Methods of the class Font

```
public Font(String name, int style, int size)
//Constructor
//Creates a new Font from the specified name, style, and point
//size.

public String getFamily()
//Returns the family name of this Font.

public String getFontName()
//Returns the font face name of this Font.
```



## Finding All Available Font Names

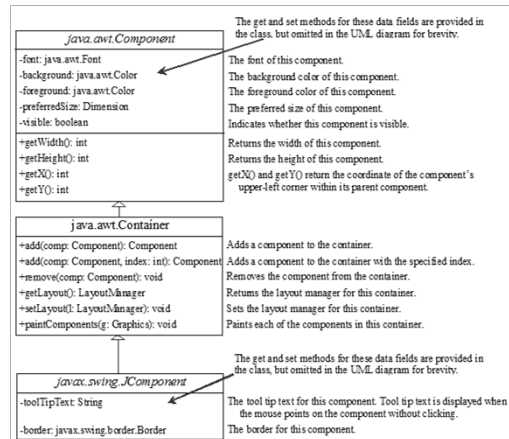
```
GraphicsEnvironment e =
    GraphicsEnvironment.getLocalGraphicsEnvironment();
String[] fontnames =
    e.getAvailableFontFamilyNames();
for (int i = 0; i < fontnames.length; i++)
    System.out.println(fontnames[i]);
```

Finffond

Run



## Common Features of Swing Components



49

## Constructors and Methods of the class Component

TABLE 13-1 Constructors and Methods of the class Component

```

protected Component()
    //Constructor
    //Creates a new instance of a component.

public void addComponentListener(ComponentListener lis)
    //Adds the component listener specified by lis.

public void addFocusListener(FocusListener lis)
    //Adds the focus listener specified by lis.

public void addKeyListener(KeyListener lis)
    //Adds the key listener specified by lis.

public void addMouseListener(MouseListener lis)
    //Adds the mouse listener specified by lis.

public void addMouseMotionListener(MouseMotionListener lis)
    //Adds the mouse motion listener specified by lis.

public void removeComponentListener(ComponentListener lis)
    //Removes the component listener specified by lis.
    
```

50

## Constructors and Methods of the class Component (continued)

```

public void removeKeyListener(KeyListener lis)
    //Removes the key listener specified by lis.

public void removeMouseListener(MouseListener lis)
    //Removes the mouse listener specified by lis.

public void removeMouseMotionListener(MouseMotionListener lis)
    //Removes the mouse motion listener specified by lis.

public Color getBackground()
    //Returns the background color of this component.

public Color getForeground()
    //Returns the foreground color of this component.

public void setBackground(Color c)
    //Sets the background color of this component to color c.

public void setForeground(Color c)
    //Sets the foreground color of this component to color c.

public Font getFont()
    //Returns the font of this component.
    
```

51

## Constructors and Methods of the class Component (continued)

```

public void setFont(Font ft)
    //Sets the font of this component to ft.

public void setSize(int w, int h)
    //Sets the size of this component to width w and height h

public boolean isVisible()
    //Returns true if the component is visible; false otherwise.

public void setVisible(boolean tog)
    //If tog is true, sets the component to visible;
    //if tog is false, the component is not shown.

public void paint(Graphics g)
    //Paints the component with the graphic component specified by g.

public void repaint()
    //Repaints the component.

public void repaint(int x, int y, int wid, int ht)
    //Repaints the rectangular portion of the component from (x, y)
    //to (x + wid, y + ht)
    
```

52

## Constructors and Methods of the class Component (continued)

```
public void setLocation(int x, int y)
    //Sets the component at the location (x, y).

public String toString()
    //Returns a string representation of this component.

public void update(Graphics g)
    //Invokes the paint method.

public void validate()
    //Validates this container and all of its subcomponents; the
    //method validate is used to cause a container to lay out its
    //subcomponents once more. Typically called after the components
    //it contains have been added to or modified.
```



33

## Borders

You can set a border on any object of the JComponent class. Swing has several types of borders. To create a titled border, use

new TitledBorder(String title).

To create a line border, use

new LineBorder(Color color, int width),

where width specifies the thickness of the line.

For example, the following code displays a titled border on a panel:

```
JPanel panel = new JPanel();
panel.setBorder(new TitledBorder("My Panel"));
```



34

## Test Swing Common Features

### Component Properties

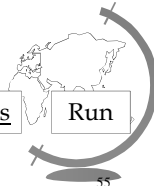
- ☞ font
- ☞ background
- ☞ foreground
- ☞ preferredSize
- ☞ minimumSize
- ☞ maximumSize

### JComponent Properties

- ☞ tooltipText
- ☞ border

TestSwingCommonFeatures

Run



35

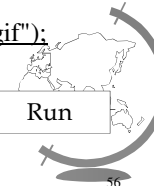
## Image Icons

Java uses the javax.swing.ImageIcon class to represent an icon. An icon is a fixed-size picture; typically it is small and used to decorate components. Images are normally stored in image files. You can use new ImageIcon(filename) to construct an image icon. For example, the following statement creates an icon from an image file us.gif in the image directory under the current class path:

```
ImageIcon icon = new ImageIcon("image/jo.gif");
```

TestImageIcon

Run



36

## class Graphics

- ☞ Provides methods for drawing items such as lines, ovals, and rectangles on the screen
- ☞ Contains methods to set the properties of graphic elements including clipping area, fonts, and colors
- ☞ Contained in the package `java.awt`



57

## Constructors and Methods of the class Graphics

TABLE 13-7 Some Constructors and Methods of the class Graphics

```
protected Graphics()
//Constructs a Graphics object that defines a context in which the
//user can draw. This constructor cannot be called directly.

public void draw3DRect(int x, int y, int w, int h, boolean t)
//Draws a 3D rectangle at (x, y) of the width w and height h. If t is
//true, the rectangle will appear raised.

public abstract void drawArc(int x, int y, int w, int h,
                             int angle, int aangle)
//Draws an arc in the rectangle at the position (x, y) of width w
//and height h. The arc starts at the angle angle with an arc angle
//aangle. Both angles are measured in degrees.

public abstract boolean drawImage(Image img, int xs1, int ys1,
                                  int xs2, int ys2, int xdl, int ydl,
                                  int xd2, int yd2, Color c, ImageObserver ob)
//Draws the image specified by img from the area defined by the
//bounding rectangle, (xs1, ys1) to (xs2, ys2), in the area defined
//by the rectangle (xdl, ydl) to (xd2, yd2). Any transparent color
//pixels are drawn in the color c. The ob monitors the progress of
//the image.
```



58

## Constructors and Methods of the class Graphics (continued)

```
public abstract void drawLine(int xs, int ys, int xd, int yd)
//Draws a line from (xs, ys) to (xd, yd).

public abstract void drawOval(int x, int y, int w, int h)
//Draws an oval at the position (x, y) of width w and height h.

public abstract void drawPolygon(int[] x, int[] y, int num)
//Draws a polygon with the points (x[0], y[0]), ...,
//(x[num - 1], y[num - 1]). Here num is the number of points in
//the polygon.

public abstract void drawPolygon(Polygon poly)
//Draws a polygon as defined by the object poly.

public abstract void drawRect(int x, int y, int w, int h)
//Draws a rectangle at the position (x, y) of width w and
//height h.
```



59

## Constructors and Methods of the class Graphics (continued)

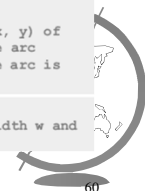
```
public abstract void drawRoundRect(int x, int y, int w, int h,
                                   int arcw, int arch)
//Draws a round-cornered rectangle at the position (x, y) having a
//width w and height h. The shape of the rounded corners is
//determined by the arc with the width arcw and the height arch.

public abstract void drawString(String s, int x, int y)
//Draws the string s at (x, y).

public void fill3DRect(int x, int y, int w, int h, boolean t)
//Draws a 3D filled rectangle at (x, y) of width w height h.
//If t is true, the rectangle will appear raised. The rectangle is
//filled with the current color.

public abstract void fillArc(int x, int y, int w, int h,
                             int angle, int aangle)
//Draws a filled arc in the rectangle at the position (x, y) of
//width w and height h starting at angle angle with the arc
//angle aangle. Both angles are measured in degrees. The arc is
//filled with the current color.

public abstract void fillOval(int x, int y, int w, int h)
//Draws a filled oval at the position (x, y) having a width w and
//height h. The oval is filled with the current color.
```



60

## Constructors and Methods of the class Graphics (continued)

```
public abstract void fillPolygon(int[] x, int[] y, int num)
//Draws a filled polygon with the points (x[0], y[0]), ...,
//(x[num - 1], y[num - 1]). Here num is the number of points in
//the polygon. The polygon is filled with the current color.

public abstract void fillPolygon(Polygon poly)
//Draws a filled polygon as defined by the object poly. The polygon
//is filled with the current color.

public abstract void fillRect(int x, int y, int w, int h)
//Draws a filled rectangle at the position (x, y) of width w
//and height h. The rectangle is filled with the current color.

public abstract void fillRoundRect(int x, int y, int w, int h,
int arcw, int arch)
//Draws a filled, round-cornered rectangle at the position (x, y)
//of width w and height h. The shape of the rounded corners
//is determined by the arc with the width arcw and the height arch.
//The rectangle is filled with the current color.

public abstract Color getColor()
//Returns the current color for this graphics context.

public abstract void setColor(Color c)
//Sets the current color for this graphics context to c.
```



61

## Constructors and Methods of the class Graphics (continued)

```
public abstract Font getFont()
//Returns the current font for this graphics context.

public abstract void setFont(Font f)
//Sets the current font for this graphics context to f.

public void String toString()
//Returns a string representation of this graphics context.
```



62

## Constructors and Methods of the class Graphics (continued)

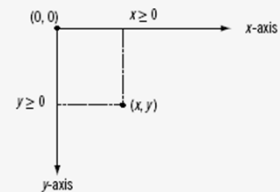


FIGURE 13-6 Java's coordinate system



63

## Constructors and Methods of the class Graphics (continued)

### EXAMPLE 13-4

```
//GrandWelcomeLine Applet
import java.awt.*;
import javax.swing.JApplet;

public class GrandWelcomeLine extends JApplet
{
    public void paint( Graphics g)
    {
        super.paint(g);

        g.setColor(Color.red);

        g.setFont(new Font("Courier", Font.BOLD, 24));
        g.drawString("Welcome to Java Programming", 30, 30);

        g.drawLine(10, 10, 10, 40); //left line
        g.drawLine(10, 40, 430, 40); //bottom line
        g.drawLine(430, 40, 430, 10); //right line
        g.drawLine(430, 10, 10, 10); //top line
    }
}
```



64



## Constructors and Methods of the class Graphics (continued)



FIGURE 13-7 Output of the GrandWelcomeLine applet

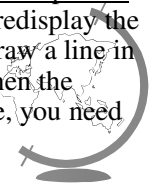


65

## Two Problems With the Preceding Example

- ☞ If you resize the frame, the line is gone.
- ☞ It is awkward to program because you have to make sure that the component to be displayed before obtaining its graphics context using the `getGraphics()` method. For this reason, Lines 20 and 21 are placed after the frame is displayed in Line 17.

To fix the first problem, you need to know its cause. When you resize the frame, the JVM invokes the `paintComponent` method of a Swing component (e.g., a label) to redisplay the graphics on the component. Since you did not draw a line in the `paintComponent` method, the line is gone when the frame is resized. To permanently display the line, you need to draw the line in the `paintComponent` method.



## The paintComponent Method

The `paintComponent` method is defined in `JComponent`, and its header is as follows:

```
protected void paintComponent(Graphics g)
```

This method, defined in the `JComponent` class, is invoked whenever the component is first displayed or redisplayed. The `Graphics` object `g` is created automatically by the JVM for every visible GUI component. The JVM obtains the `Graphics` object and passes it to invoke `paintComponent`.



## Procedural vs. Event-Driven Programming

- ☞ *Procedural programming* is executed in procedural order.
- ☞ In event-driven programming, code is executed upon activation of events.



68

**TABLE 10.1** User Action, Source Object, and Event Type

User Action	Source Object	Event Type Generated
Click a button	JButton	ActionEvent
Change text	JTextComponent	TextEvent
Press return on a text field	JTextField	ActionEvent
Select a new item	JComboBox	ItemEvent, ActionEvent
Select item(s)	JList	ListSelectionEvent
Click a check box	JCheckBox	ItemEvent, ActionEvent
Click a radio button	JRadioButton	ItemEvent, ActionEvent
Select a menu item	JMenuItem	ActionEvent
Move the scroll bar	JScrollBar	AdjustmentEvent
Window opened, closed, iconified, deiconified, or closing	Window	WindowEvent
Component added or removed from the container	Container	ContainerEvent
Component moved, resized, hidden, or shown	Component	ComponentEvent
Component gained or lost focus	Component	FocusEvent
Key released or pressed	Component	KeyEvent
Mouse pressed, released, clicked, entered, or exited	Component	MouseEvent
Mouse moved or dragged	Component	MouseEvent

**Table 10.2** Events, Event Listeners, and Listener Methods

Event Class	Listener Interface	Listener Methods (Handlers)
ActionEvent	ActionListener	actionPerformed(ActionEvent e)
ItemEvent	ItemListener	itemStateChanged(ItemEvent e)
WindowEvent	WindowListener	windowClosing(WindowEvent e) windowOpened(WindowEvent e) windowIconified(WindowEvent e) windowDeiconified(WindowEvent e) windowClosed(WindowEvent e) windowActivated(WindowEvent e) windowDeactivated(WindowEvent e)
ContainerEvent	ContainerListener	componentAdded(ContainerEvent e) componentRemoved(ContainerEvent e)
ComponentEvent	ComponentListener	componentMoved(ComponentEvent e) componentHidden(ComponentEvent e) componentResized(ComponentEvent e) componentShown(ComponentEvent e)
FocusEvent	FocusListener	focusGained(FocusEvent e) focusLost(FocusEvent e)
TextEvent	TextListener	textValueChanged(TextEvent e)
KeyEvent	KeyListener	keyPressed(KeyEvent e) keyReleased(KeyEvent e) keyTyped(KeyEvent e)
MouseEvent	MouseListener	mousePressed(MouseEvent e) mouseReleased(MouseEvent e) mouseEntered(MouseEvent e) mouseExited(MouseEvent e) mouseClicked(MouseEvent e) mouseDragged(MouseEvent e) mouseMoved(MouseEvent e)
AdjustmentEvent	AdjustmentListener	adjustmentValueChanged(AdjustmentEvent e)

## Handling an Event

- ☞ Action event: event created when JButton is clicked
- ☞ Event listener: object that receives message when JButton is clicked
- ☞ In Java, you must register the listener



71

## Handling an Event (continued)

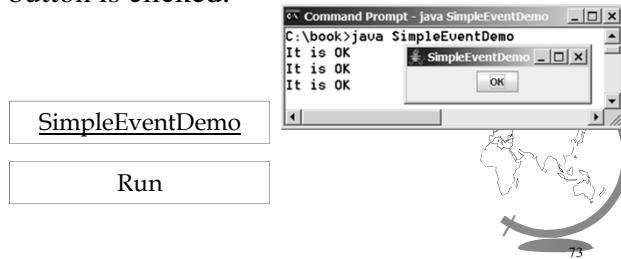
- ☞ class ActionListener
  - Handles action event
  - Part of package java.awt.Event
  - The class ActionListener is a special type of class (interface)
  - Must contain actionPerformed method



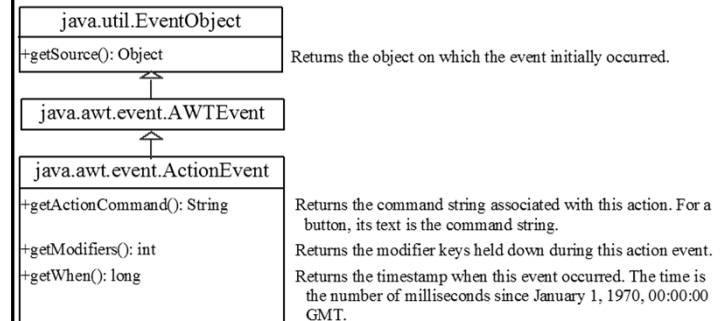
72

## Taste of Event-Driven Programming

- ☞ The example displays a button in the frame. A message is displayed on the console when a button is clicked.



## java.awt.event.ActionEvent



## Inner Class Listeners

A listener class is designed specifically to create a listener object for a GUI component (e.g., a button). It will not be shared by other applications. So, it is appropriate to define the listener class inside the frame class as an inner class.



## Inner Classes

Inner class: A class is a member of another class.

Advantages: In some applications, you can use an inner class to make programs simple.

- ☞ An inner class can reference the data and methods defined in the outer class in which it nests, so you do not need to pass the reference of the outer class to the constructor of the inner class.

ShowInnerClass



## Inner Classes, cont.

```
public class Test {  
    ...  
    public class A {  
        ...  
    }  
}
```

(a)

```
public class Test {  
    ...  
    // Inner class  
    public class A {  
        ...  
    }  
}
```

(b)

```
// OuterClass.java: inner class demo  
public class OuterClass {  
    private int data;  
  
    /** A method in the outer class */  
    public void m() {  
        // Do something  
    }  
  
    // An inner class  
    class InnerClass {  
        /** A method in the inner class */  
        public void mi() {  
            // Directly reference data and method  
            // defined in its outer class  
            data++;  
            m();  
        }  
    }  
}
```

(c)



77

## Inner Classes (cont.)

- ☞ Inner classes can make programs simple and concise.
- ☞ An inner class supports the work of its containing outer class and is compiled into a class named *OuterClassName\$InnerClassName.class*. For example, the inner class InnerClass in OuterClass is compiled into *OuterClass\$InnerClass.class*.



78

## Inner Classes (cont.)

- ☞ An inner class can be declared public, protected, or private subject to the same visibility rules applied to a member of the class.
- ☞ An inner class can be declared static. A static inner class can be accessed using the outer class name. A static inner class cannot access nonstatic members of the outer class



79

## Revising SimpleEventDemo Using Inner Classes

SimpleEventDemoInnerClass

Run



80

## Anonymous Inner Classes

- ☞ An anonymous inner class must always extend a superclass or implement an interface, but it cannot have an explicit extends or implements clause.
- ☞ An anonymous inner class must implement all the abstract methods in the superclass or in the interface.
- ☞ An anonymous inner class always uses the no-arg constructor from its superclass to create an instance. If an anonymous inner class implements an interface, the constructor is Object().
- ☞ An anonymous inner class is compiled into a class named OuterClassName\$n.class. For example, if the outer class Test has two anonymous inner classes, these two classes are compiled into Test\$1.class and Test\$2.class.



81

## Anonymous Inner Classes (cont.)

Inner class listeners can be shortened using anonymous inner classes. An *anonymous inner class* is an inner class without a name. It combines declaring an inner class and creating an instance of the class in one step. An anonymous inner class is declared as follows:

```
new SuperClassName/InterfaceName() {  
    // Implement or override methods in superclass or interface  
    // Other methods if necessary  
}
```



82

## Revising SimpleEventDemo Using Anonymous Inner Classes

SimpleEventDemoAnonymousInnerClass

Run



83

## Example: 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

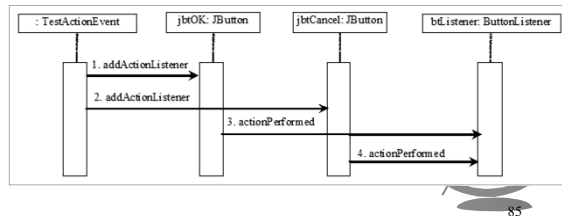
Run



84

## Interaction Between Source and Listener

1. `jbtOK` registers `btListener` by invoking `addActionListener(btListener)`.
2. `jbtCancel` registers `btListener` by invoking `addActionListener(btListener)`.
3. `jbtOK` invokes `btListener`'s `actionPerformed` method to process an `ActionEvent`.
4. `jbtCancel` invokes `btListener`'s `actionPerformed` method to process an `ActionEvent`.



## Example: 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

Run

## Example: Multiple Listeners for a Single Source

- Objective: This example modifies previous Listing to add a new listener for each button. The two buttons OK and Cancel use the frame class as the listener. This example creates a new listener class as an additional listener for the action events on the buttons. When a button is clicked, both listeners respond to the action event.

TestMultipleListener

Run

## MouseEvent

java.awt.event.InputEvent

+getWhen(): long  
+isAltDown(): boolean  
+isControlDown(): boolean  
+isMetaDown(): boolean  
+isShiftDown(): boolean

Returns the timestamp when this event occurred.  
Returns whether or not the Alt modifier is down on this event.  
Returns whether or not the Control modifier is down on this event.  
Returns whether or not the Meta modifier is down on this event.  
Returns whether or not the Shift modifier is down on this event.

java.awt.event.MouseEvent

+getButton(): int  
+getClickCount(): int  
+getPoint(): java.awt.Point  
+getX(): int  
+getY(): int

Indicates which mouse button has been clicked.  
Returns the number of mouse clicks associated with this event.  
Returns a `Point` object containing the x and y coordinates.  
Returns the x-coordinate of the mouse point.  
Returns the y-coordinate of the mouse point.

## Handling Mouse Events

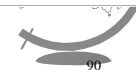
- Java provides two listener interfaces, `MouseListener` and `MouseMotionListener`, to handle mouse events.
- The `MouseListener` listens for actions such as when the mouse is pressed, released, entered, exited, or clicked.
- The `MouseMotionListener` listens for actions such as dragging or moving the mouse.



89

## Handling Mouse Events

<code>java.awt.event.MouseListener</code>	
<code>+mousePressed(e: MouseEvent): void</code>	Invoked when the mouse button has been pressed on the source component.
<code>+mouseReleased(e: MouseEvent): void</code>	Invoked when the mouse button has been released on the source component.
<code>+mouseClicked(e: MouseEvent): void</code>	Invoked when the mouse button has been clicked (pressed and released) on the source component.
<code>+mouseEntered(e: MouseEvent): void</code>	Invoked when the mouse enters the source component.
<code>+mouseExited(e: MouseEvent): void</code>	Invoked when the mouse exits the source component.
<code>java.awt.event.MouseMotionListener</code>	
<code>+mouseDragged(e: MouseEvent): void</code>	Invoked when a mouse button is moved with a button pressed.
<code>+mouseMoved(e: MouseEvent): void</code>	Invoked when a mouse button is moved without a button pressed.



90

## Example: Moving Message Using Mouse

Objective: Create a program to display a message in a panel. You can use the mouse to move the message. The message moves as the mouse drags and is always displayed at the mouse point.



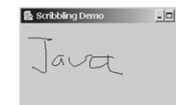
91

MoveMessageDemo

Run

## Example: Handling Complex Mouse Events

Objective: Create a program for drawing using a mouse. Draw by dragging with the left mouse button pressed; erase by dragging with the right button pressed.



92

ScribbleDemo

Run

## Applets

- ☞ Applet: a Java program that is embedded within a Web page and executed by a Web browser
- ☞ Create an applet by extending the class JApplet
- ☞ class JApplet contained in package javax.swing



93

## Members of class JApplet

TABLE 13-3 Some Members of the class JApplet (package javax.swing)

```
public void init()
//Called by the browser or applet viewer to inform this applet
//that it has been loaded into the system.

public void start()
//Called by the browser or applet viewer to inform this applet
//that it should start its execution. It is called after the init
//method and each time the applet is revisited in a Web page.

public void stop()
//Called by the browser or applet viewer to inform this applet
//that it should stop its execution. It is called before the
//method destroy.

public void destroy()
//Called by the browser or applet viewer. Informs this applet that
//it is being reclaimed and that it should destroy any resources
//that it has allocated. The method stop is called before destroy.

public void showStatus(String msg)
//Displays the string msg in the status bar.
```



94

## Members of class JApplet (continued)

```
public Container getContentPane()
//Returns the ContentPane object for this applet.

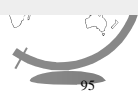
public JMenuBar getJMenuBar()
//Returns the JMenuBar object for this applet.

public URL getDocumentBase()
//Returns the URL of the document that contains this applet.

public URL getCodeBase()
//Returns the URL of this applet.

public void update(Graphics g)
//Calls the paint() method.

protected String paramString()
//Returns a string representation of this JApplet; mainly used
//for debugging.
```



95

## Applets (continued)

- ☞ No main method
- ☞ Methods init, start, and paint guaranteed to be invoked in sequence
- ☞ To develop an applet
  - Override any/all of the methods above



96



## Applet Methods

### ☞ init Method

- Initializes variables
- Gets data from user
- Places various GUI components

### ☞ paint Method

- Performs output



## Skeleton of a Java Applet

```
import java.awt.Graphics;  
import javax.swing.JApplet;  
  
public class WelcomeApplet extends JApplet  
{  
  
}
```

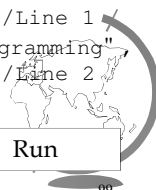


## Applet Displaying Welcome Message

```
//Welcome Applet  
  
import java.awt.Graphics;  
import javax.swing.JApplet;  
  
public class WelcomeApplet extends JApplet  
{  
    public void paint(Graphics g)  
    {  
        super.paint(g);  
        g.drawString("Welcome to Java Programming",  
                    30, 30);  
    }  
}
```

HTML Code

Run



## Differences Between Applets and GUI Applications

### ☞ Applets

- Derived from JApplet
- No main method
- Uses init method
- Displayed by HTML
- Sets title in HTML
- Size set in HTML
- Applet closes when HTML doc closes

### ☞ GUI applications

- class extends JFrame
- Invokes main method
- Uses constructors
- Uses method setVisible
- Uses setTitle method
- Uses method setSize
- Closes with Exit button



## Converting a GUI Application to an Applet

- ☞ Change `JFrame` to `JApplet`
- ☞ Change constructor to method `init`
- ☞ Remove method calls such as `setVisible`, `setTitle`, `setSize`
- ☞ Remove the method `main`
- ☞ If applicable, remove `Exit` button/all code associated with it (e.g. action listener)



## Chapter Summary

- ☞ Creating Applets
- ☞ `class Font`
- ☞ `class Graphics`
- ☞ `class Color`
- ☞ Differences between Applet and GUI application
- ☞ Converting GUI application to Applet



## Chapter Summary (continued)

- ☞ GUI components
  - `JTextArea`
  - `JCheckBox`
  - `JRadioButton`
- ☞ Layout managers
- ☞ Menus
- ☞ Key and mouse events

