

*core*  
**WEB**  
*programming*

# Basic Swing

GUI Controls in Java 2

# Agenda

- **New features**
- **Basic approach**
- **Summary of Swing components**
  - Starting points
    - JApplet, JFrame
  - Swing equivalent of AWT components
    - JLabel, JButton, JPanel, JSlider
  - New Swing components
    - JColorChooser, JInternalFrame, JOptionPane, JToolBar, JEditorPane
  - Other simple components
    - JCheckBox, JRadioButton, JTextField, JTextArea, JFileChooser

# New Features

- **Many more built-in controls**
  - Image buttons, tabbed panes, sliders, toolbars, color choosers, HTML text areas, lists, trees, and tables.
- **Increased customization of components**
  - Border styles, text alignments, and basic drawing features. Images can be added to almost any control.
- **A pluggable “look and feel”**
  - Not limited to “native” look.
- **Many miscellaneous small features**
  - Built-in double buffering, tool-tips, dockable toolbars, keyboard accelerators, custom cursors, etc.
- **Model-view-controller architecture**
  - Can change internal representation of trees, lists, tables.

# Swing vs. AWT Programming

- **Naming convention**
  - All Swing component names begin with a capital J and follow the format *JXxx*. E.g., JFrame, JPanel, JApplet, JDialog, JButton. Many are just AWT names with a J.
- **Lightweight components**
  - Most Swing components are *lightweight*: formed by drawing in the underlying window.
- **Use of paintComponent for drawing**
  - Custom drawing code is in paintComponent, not paint. Double buffering turned on by default.
- **New Look and Feel as default**
  - With Swing, you have to explicitly set the native look.
- **Don't mix Swing and AWT in same window**

# Windows Look and Feel

File Look & Feel Themes

Table Demo Source Code

Reordering allowed  
 Horiz. Lines  
 Vert. Lines  
Inter-cell spacing:

Column selection  
 Row selection  
Row height:

Selection mode: Multiple ranges

Autosize mode: Subsequent columns

First Name	Last Name	Favorite Color	Favorite Movie	Favorite Number	Favorite Food
Mike	Albers	Green	Brazil	44	
Mark	Andrews	Blue	Curse of the Demon	3	
Brian	Beck	Black	The Blues Brothers	2,718	
Lara	Bunni	Red	Airplane (the whole series)	15	
Roger	Brinkley	Blue	The Man Who Knew Too Much	13	
Brent	Christian	Black	Blade Runner (Director's Cut)	23	
Mark	Davidson	Dark Green	Brazil	27	
Jeff	Dinkins	Blue	The Lady Vanishes	8	
Ewan	Dinkins	Yellow	A Bug's Life	2	
Amy	Fowler	Violet	Reservoir Dogs	3	

# Motif Look and Feel

File Look & Feel Themes



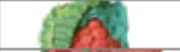




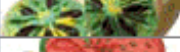

Table Demo Source Code

Reordering allowed  
 Horiz. Lines  
 Vert. Lines  
Inter-cell spacing:

Column selection  
 Row selection  
Row height:

Selection mode: Multiple ranges

Autosize mode: Subsequent columns

First Name	Last Name	Favorite Color	Favorite Movie	Favorite Number	Favorite Food
Mike	Albers	Green	Brazil	44	
Mark	Andrews	Blue	Curse of the Demon	3	
Brian	Beck	Black	The Blues Brothers	2.718	
Lara	Bunni	Red	Airplane (the whole series)	15	
Roger	Brinkley	Blue	The Man Who Knew Too Much	13	
Brent	Christian	Black	Blade Runner (Director's Cut)	23	
Mark	Davidson	Dark Green	Brazil	27	
Jeff	Dinkins	Blue	The Lady Vanishes	8	
Ewan	Dinkins	Yellow	A Bug's Life	2	

# Java Look and Feel

File Look & Feel Themes

Table Demo Source Code

Reordering allowed  
 Horiz. Lines  
 Vert. Lines  
Inter-cell spacing:

Column selection  
 Row selection  
Row height:

Selection mode: Multiple ranges  
Autosize mode: Subsequent columns

First Name	Last Name	Favorite Color	Favorite Movie	Favorite Number	Favorite Food
Mike	Albers	Green	Brazil	44	
Mark	Andrews	Blue	Curse of the Demon	3	
Brian	Beck	Black	The Blues Brothers	2,718	
Lara	Bunni	Red	Airplane (the whole series)	15	
Roger	Brinkley	Blue	The Man Who Knew Too Much	13	
Brent	Christian	Black	Blade Runner (Director's Cut)	23	
Mark	Davidson	Dark Green	Brazil	27	
Jeff	Dinkins	Blue	The Lady Vanishes	8	
Ewan	Dinkins	Yellow	A Bug's Life	2	
Amy	Fowler	Violet	Reservoir Dogs	3	

# Setting Native Look and Feel

- Most applications should use native look, not default “Java” look
- Changing is tedious, so use static method

```
public class WindowUtilities {
    public static void setNativeLookAndFeel() {
        try {
            UIManager.setLookAndFeel (
                UIManager.getSystemLookAndFeelClassName ());
        } catch (Exception e) {
            System.out.println("Error setting native LAF: "
                + e);
        }
    }
    ...
}
```

# Whirlwind Tour of Basic Components

- **Starting points**
  - JApplet, JFrame
- **Swing equivalent of AWT components**
  - JLabel, JButton, JPanel, JSlider
- **New Swing components**
  - JColorChooser, JInternalFrame, JOptionPane, JToolBar, JEditorPane
- **Other simple components**
  - JCheckBox, JRadioButton, JTextField, JTextArea, JFileChooser

# Starting Point 1: JApplet

- **Content pane**

- A JApplet contains a content pane in which to add components. Changing other properties like the layout manager, background color, etc., also applies to the content pane. Access the content pane through `getContentPane`.

- **Layout manager**

- The default layout manager is `BorderLayout` (as with `Frame` and `JFrame`), not `FlowLayout` (as with `Applet`). `BorderLayout` is really layout manager of content pane.

- **Look and feel**

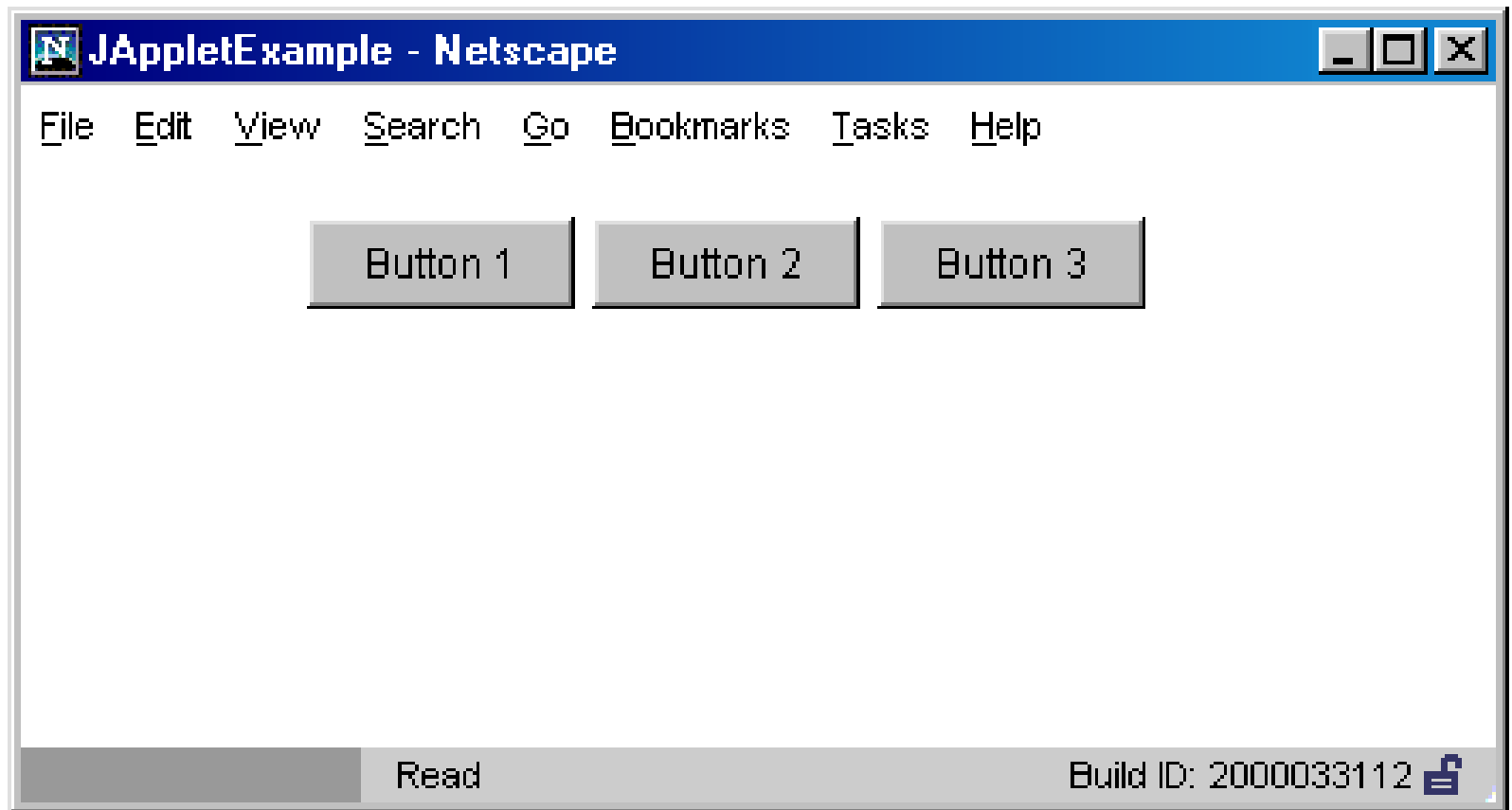
- The default look and feel is Java (Metal), so you have to explicitly switch the look and feel if you want the native look.

# JApplet: Example Code

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

public class JAppletExample extends JApplet {
    public void init() {
        WindowUtilities.setNativeLookAndFeel();
        Container content = getContentPane();
        content.setBackground(Color.white);
        content.setLayout(new FlowLayout());
        content.add(new JButton("Button 1"));
        content.add(new JButton("Button 2"));
        content.add(new JButton("Button 3"));
    }
}
```

# JApplet: Example Output



# Starting Point 2: JFrame

- **Content pane**
  - JFrame uses content pane in same way as does JApplet.
- **Auto-close behavior**
  - JFrames close automatically when you click on the Close button (unlike AWT Frames). However, closing the last JFrame does not result in your program exiting the Java application. So, your “main” JFrame still needs a WindowListener to call System.exit. Or, alternatively, if using JDK 1.3 or later, you can call setDefaultCloseOperation(EXIT\_ON\_CLOSE). This permits the JFrame to close; however, you won’t be able to complete any house cleaning as you might in the WindowListener.
- **Look and feel**
  - The default look and feel is Java (Metal)

# JFrame: Example Code

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

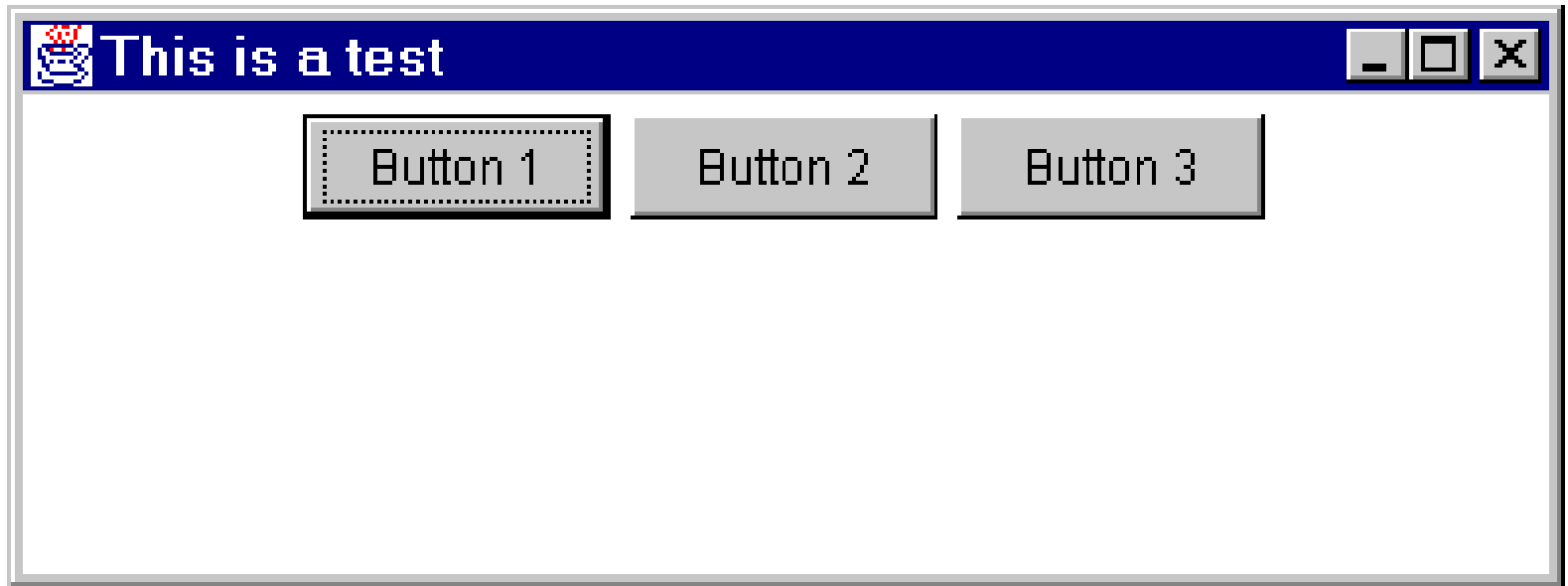
public class JFrameExample {
    public static void main(String[] args) {
        WindowUtilities.setNativeLookAndFeel();
        JFrame f = new JFrame("This is a test");
        f.setSize(400, 150);
        Container content = f.getContentPane();
        content.setBackground(Color.white);
        content.setLayout(new FlowLayout());
        content.add(new JButton("Button 1"));
        content.add(new JButton("Button 2"));
        content.add(new JButton("Button 3"));
        f.addWindowListener(new ExitListener());
        f.setVisible(true);
    }
}
```

# JFrame Helper: ExitListener

```
import java.awt.*;
import java.awt.event.*;

public class ExitListener extends WindowAdapter {
    public void windowClosing(WindowEvent event) {
        System.exit(0);
    }
}
```

# JFrame: Example Output



# Swing Equivalents of AWT Components

- **JLabel**
  - New features: HTML content images, borders
- **JButton**
  - New features: icons, alignment, mnemonics
- **JPanel**
  - New feature: borders
- **JSlider**
  - New features: tick marks and labels

# JLabel

- **Main new feature: HTML content**

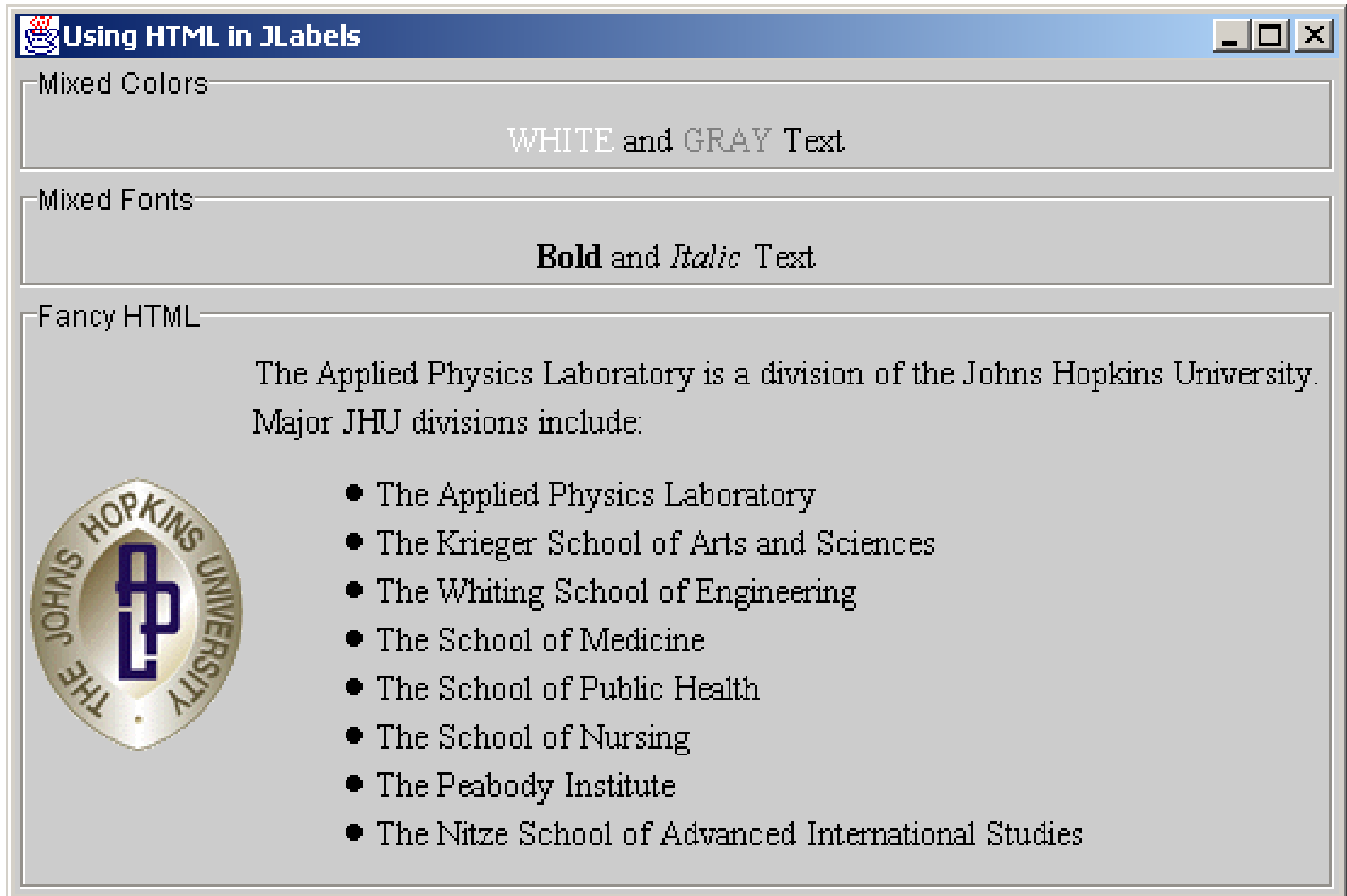
- If text is "`<html>...</html>`", it gets rendered as HTML
- HTML labels only work in JDK 1.2.2 or later, or in Swing 1.1.1 or later.
- In JDK 1.2 the label string must begin with `<html>`, not `<HTML>`. It is case-insensitive in JDK 1.3 and 1.4.
- JLabel fonts are ignored if HTML is used. If you use HTML, all font control must be performed by HTML.
- You must use `<P>`, not `<BR>`, to force a line break.
- Other HTML support is spotty.
  - Be sure to test each HTML construct you use. Permitting the user to enter HTML text at runtime is asking for trouble.

- **Other new features: images, borders**

# JLabel: Example Code

```
String labelText =
    "<html><FONT COLOR=WHITE>WHITE</FONT> and " +
    "<FONT COLOR=GRAY>GRAY</FONT> Text</html>";
JLabel coloredLabel =
    new JLabel(labelText, JLabel.CENTER);
...
labelText =
    "<html><B>Bold</B> and <I>Italic</I> Text</html>";
JLabel boldLabel =
    new JLabel(labelText, JLabel.CENTER);
labelText =
    "<html>The Applied Physics Laboratory is..." +
    "of the Johns Hopkins University." +
    "<P>" + ... "...</html>";
```

# JLabel: Example Output



The screenshot shows a Java Swing window titled "Using HTML in JLabels" with three sections demonstrating HTML formatting in JLabels:

- Mixed Colors:** Displays the text "WHITE and GRAY Text" where "WHITE" is white and "GRAY" is gray.
- Mixed Fonts:** Displays the text "**Bold** and *Italic* Text" where "Bold" is in a bold font and "Italic" is in an italic font.
- Fancy HTML:** Displays a paragraph of text and a bulleted list. The text reads: "The Applied Physics Laboratory is a division of the Johns Hopkins University. Major JHU divisions include:" followed by a list of seven divisions, each preceded by a black dot. To the left of the list is the Johns Hopkins University logo, which is a circular emblem with "THE JOHN HOPKINS UNIVERSITY" around the perimeter and a stylized "JH" in the center.

# JButton

- **Main new feature: icons**
  1. Create an ImageIcon by passing the ImageIcon constructor a String representing a GIF or JPG file (animated GIFs are supported!).
    - From an applet, call `getImage(getCodeBase()...)` normally, then pass resultant Image to ImageIcon.
  2. Pass the ImageIcon to the JButton constructor.
    - Alternatively, call `setIcon`. In fact, there are 7 possible images (rollover images, images for when button is depressed, etc.)
- **Other features**
  - HTML content as with JLabel
  - Alignment: location of image with respect to text
  - Mnemonics: keyboard accelerators that let you use `Alt-someChar` to trigger the button.

# JButton: Example Code

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

public class JButtons extends JFrame {
    public static void main(String[] args) {
        new JButtons();
    }

    public JButtons() {
        super("Using JButton");
        WindowUtilities.setNativeLookAndFeel();
        addWindowListener(new ExitListener());
        Container content = getContentPane();
        content.setBackground(Color.white);
        content.setLayout(new FlowLayout());
    }
}
```

# JButton: Example Code (Continued)

```
JButton button1 = new JButton("Java");
content.add(button1);
ImageIcon cup = new ImageIcon("images/cup.gif");
JButton button2 = new JButton(cup);
content.add(button2);
JButton button3 = new JButton("Java", cup);
content.add(button3);
JButton button4 = new JButton("Java", cup);
button4.setHorizontalTextPosition
                    (SwingConstants.LEFT);
content.add(button4);
pack();
setVisible(true);
}
}
```

# JButton: Example Output



# JPanel

- **Main new feature: borders**

- Create a Border object by calling `BorderFactory.createXxxBorder`.
- Supply the Border object to the JPanel by means of `setBorder`.

```
JPanel p = new JPanel();  
p.setBorder(BorderFactory.createTitledBorder("Java"));
```

- **Other features:**

- Layout manager settings
  - Can pass the layout manager to the JPanel constructor
- Setting preferred size
  - There is no `JCanvas`. If you want JPanel to act like Canvas, call `setPreferredSize`.

# Standard Borders

- **Static methods in BorderFactory**

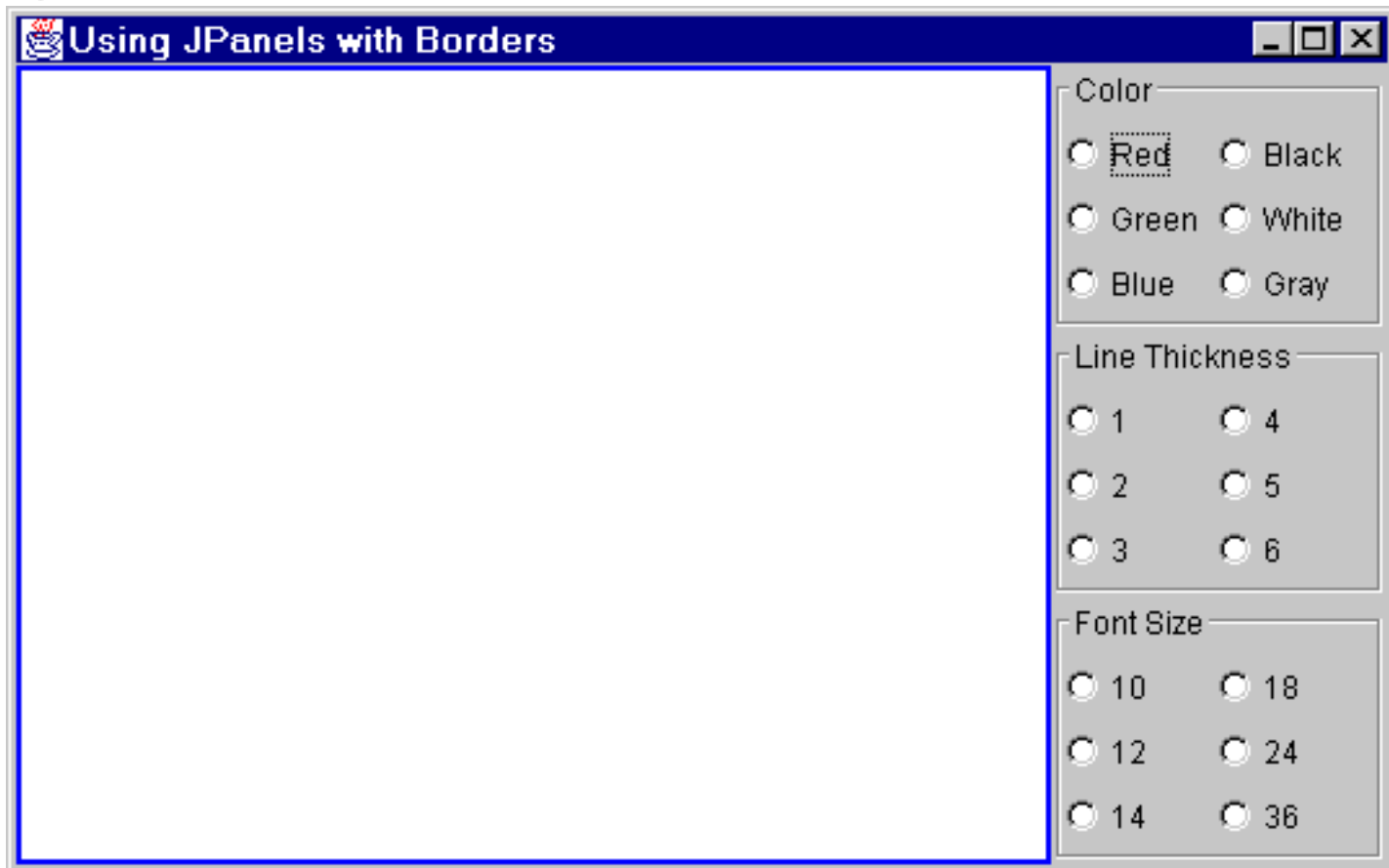
- createEmptyBorder(int top, int left, int bottom, int right)
  - Creates an EmptyBorder object that simply adds space (margins) around the component.
- createLineBorder(Color color)
- createLineBorder(Color color, int thickness)
  - Creates a solid-color border
- createTitledBorder(String title)
- createTitledBorder(Border border, String title)
  - The border is an etched line unless you explicitly provide a border style in second constructor.
- createEtchedBorder()
- createEtchedBorder(Color highlight, Color shadow)
  - Creates a etched line without the label

# JPanel: Example Code

```
public class SixChoicePanel extends JPanel {
    public SixChoicePanel(String title, String[] buttonLabels)
    {
        super(new GridLayout(3, 2));
        setBackground(Color.lightGray);
        setBorder(BorderFactory.createTitledBorder(title));
        ButtonGroup group = new ButtonGroup();
        JRadioButton option;
        int halfLength = buttonLabels.length/2;
        for(int i=0; i<halfLength; i++) {
            option = new JRadioButton(buttonLabels[i]);
            group.add(option);
            add(option);
            option = new JRadioButton(buttonLabels[i+halfLength]);
            group.add(option);
            add(option);
        }
    }
}
```

# JPanel: Example Output

- Left window uses createLineBorder
- Right window has three SixChoicePanels



# JSlider

- **Basic use**

- public JSlider()
- public JSlider(int orientation)
- public JSlider(int min, int max)
- public JSlider(int min, int max, int initialValue)
- public JSlider(int orientation, int min, int max, int initialValue)

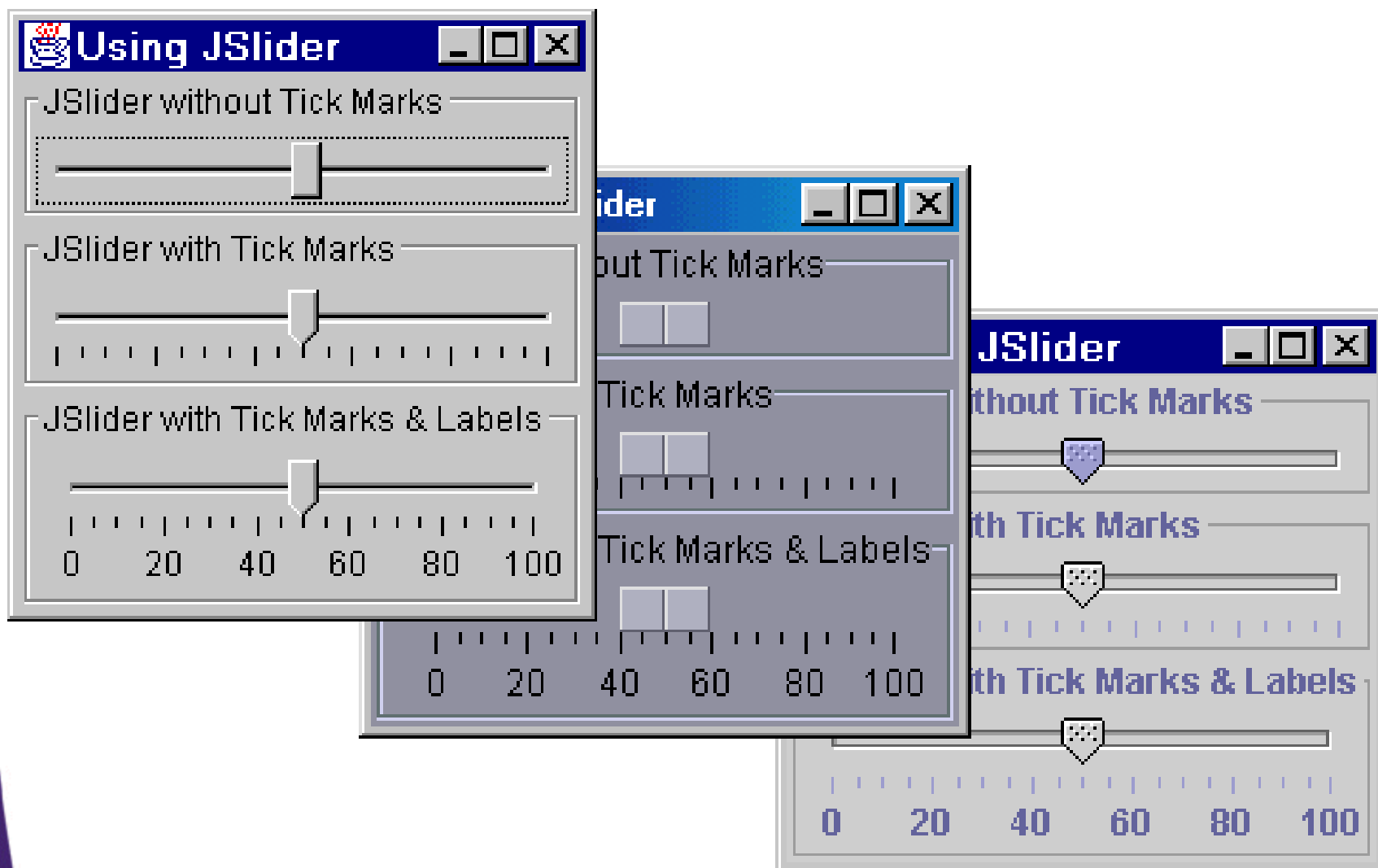
- **New features: tick marks and labels**

- setMajorTickSpacing
- setMinorTickSpacing
- setPaintTicks
- setPaintLabels (icons allowed as labels)

# JSlider: Example Code

```
JSlider slider1 = new JSlider();
slider1.setBorder(...);
content.add(slider1, BorderLayout.NORTH);
JSlider slider2 = new JSlider();
slider2.setBorder(...);
slider2.setMajorTickSpacing(20);
slider2.setMinorTickSpacing(5);
slider2.setPaintTicks(true);
content.add(slider2, BorderLayout.CENTER);
JSlider slider3 = new JSlider();
slider3.setBorder(...);
slider3.setMajorTickSpacing(20);
slider3.setMinorTickSpacing(5);
slider3.setPaintTicks(true);
slider3.setPaintLabels(true);
content.add(slider3, BorderLayout.SOUTH);
```

# JSlider: Example Output (Windows, Motif, Java LAF)



# JColorChooser

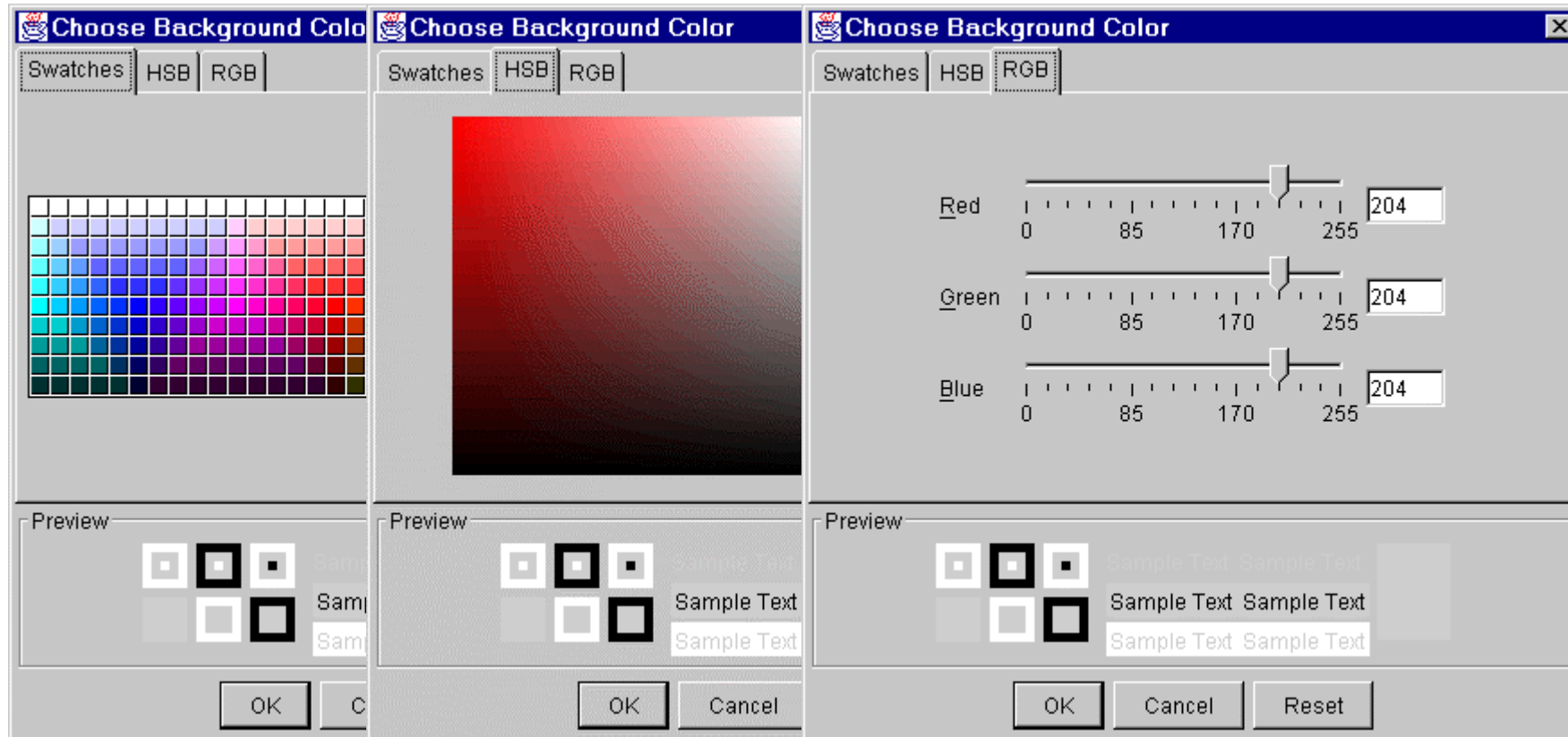
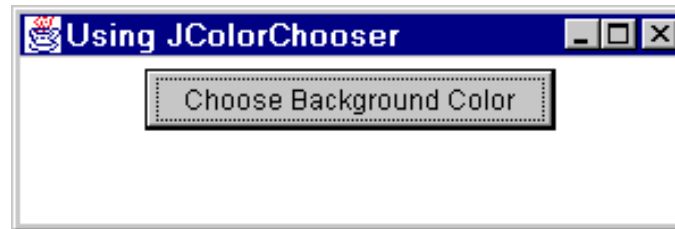
- **Open**
  - Call `JColorChooser.showDialog`
    - First argument: parent component
    - Second argument: title string
    - Third argument: initially-selected Color
- **Return value**
  - Selected Color if "OK" chosen
  - null if "Cancel" chosen

# JColorChooser: Example Code

- Button that lets you change color of window

```
public void actionPerformed(ActionEvent e) {
    Color bgColor
        = JColorChooser.showDialog
            (this,
             "Choose Background Color",
             getBackground());
    if (bgColor != null)
        getContentPane().setBackground(bgColor);
}
```

# JColorChooser: Example Output



# Internal Frames

- **MDI: Multiple Document Interface**

- Program has one large “desktop” pane that holds all other windows. The other windows can be iconified (minimized) and moved around within this desktop pane, but not moved outside the pane. Furthermore, minimizing the desktop pane hides all the contained windows as well.
- Examples: Microsoft PowerPoint, Corel Draw, Borland JBuilder, and Allaire HomeSite

- **Swing Support for MDI**

- JDesktopPane
  - Serves as a holder for the other windows.
- JInternalFrame
  - Acts mostly like a JFrame, except that it is constrained to stay inside the JDesktopPane.

# Using JFrame

- **Main constructor**

- public JFrame(String title,  
boolean resizable,  
boolean closeable,  
boolean maximizable,  
boolean iconifiable)

- **Other useful methods**

- moveToFront
- moveToBack
- setSize (required!)
- setLocation (required!)

# Internal Frames: Example Code

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

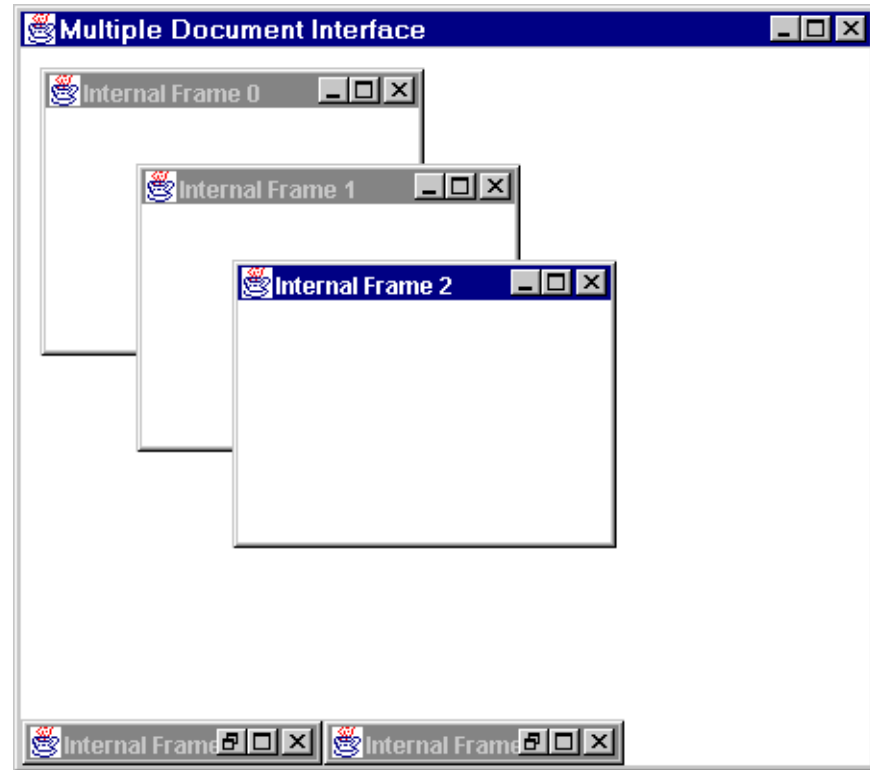
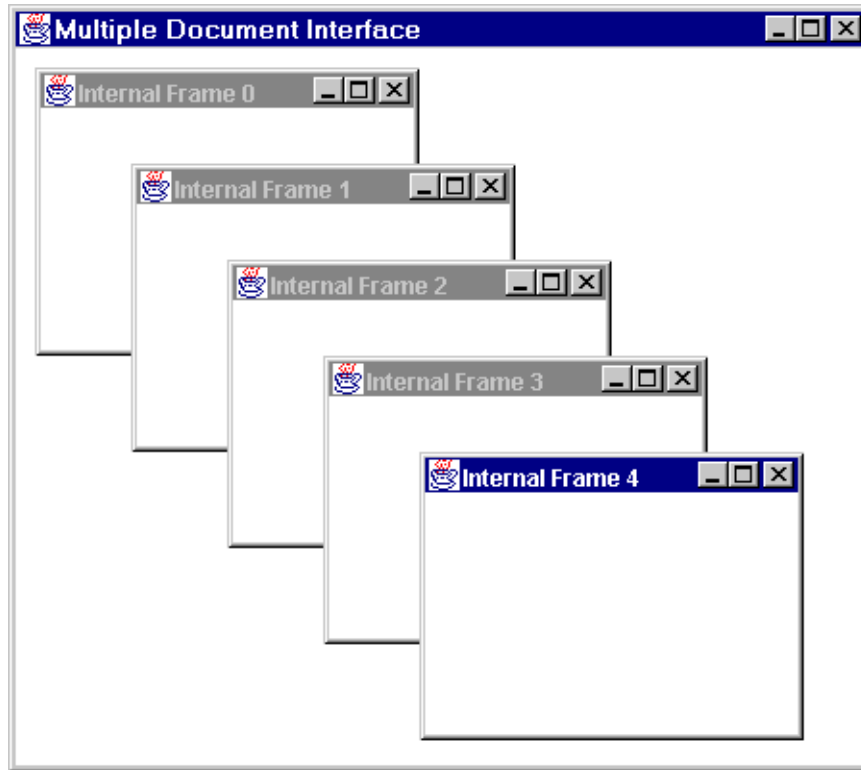
public class JInternalFrames extends JFrame {
    public static void main(String[] args) {
        new JInternalFrames();
    }

    public JInternalFrames() {
        super("Multiple Document Interface");
        WindowUtilities.setNativeLookAndFeel();
        addWindowListener(new ExitListener());
        Container content = getContentPane();
        content.setBackground(Color.white);
    }
}
```

# Internal Frames: Example Code (Continued)

```
JDesktopPane desktop = new JDesktopPane();
desktop.setBackground(Color.white);
content.add(desktop, BorderLayout.CENTER);
setSize(450, 400);
for(int i=0; i<5; i++) {
    JInternalFrame frame
        = new JInternalFrame("Internal Frame " + i,
                               true, true, true, true);
    frame.setLocation(i*50+10, i*50+10);
    frame.setSize(200, 150);
    frame.setBackground(Color.white);
    frame.setVisible(true);
    desktop.add(frame);
    frame.moveToFront();
}
setVisible(true);
```

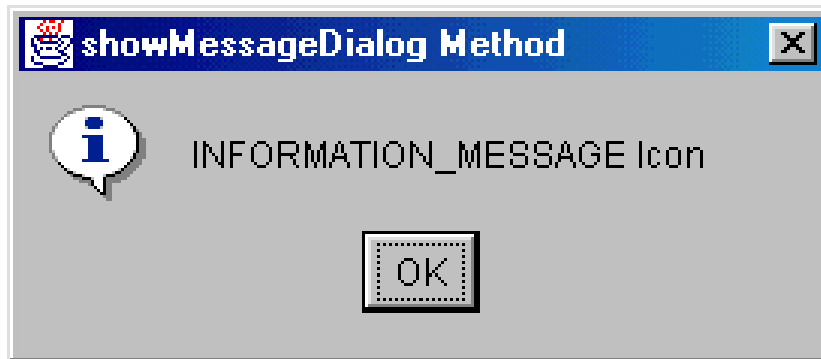
# Internal Frames: Example Output



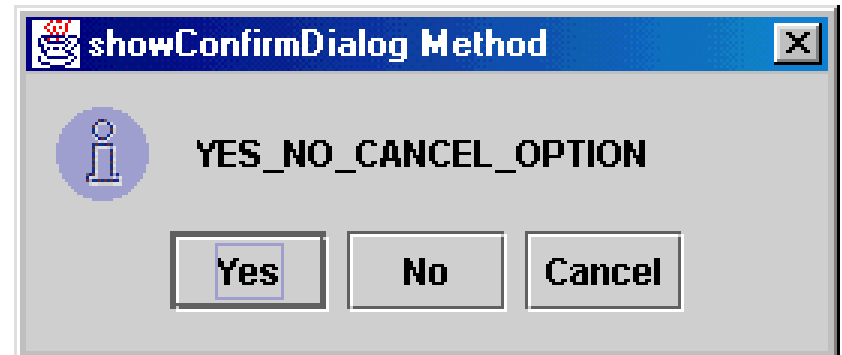
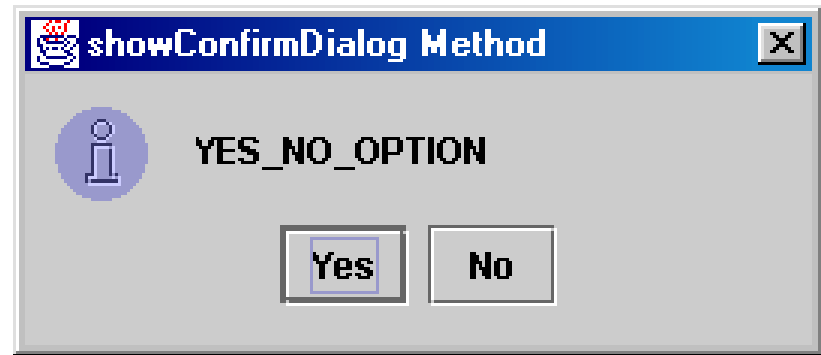
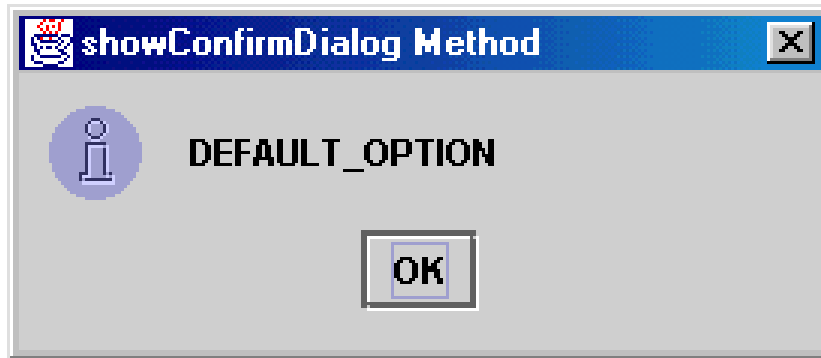
# JOptionPane

- **Very rich class with many options for different types of dialog boxes.**
- **Five main static methods**
  - `JOptionPane.showMessageDialog`
    - Icon, message, OK button
  - `JOptionPane.showConfirmDialog`
    - Icon, message, and buttons:  
OK, OK/Cancel, Yes/No, or Yes/No/Cancel
  - `JOptionPane.showInputDialog` (2 versions)
    - Icon, message, textfield or combo box, buttons
  - `JOptionPane.showOptionDialog`
    - Icon, message, array of buttons or other components

# JOptionPane Message Dialogs (Windows LAF)

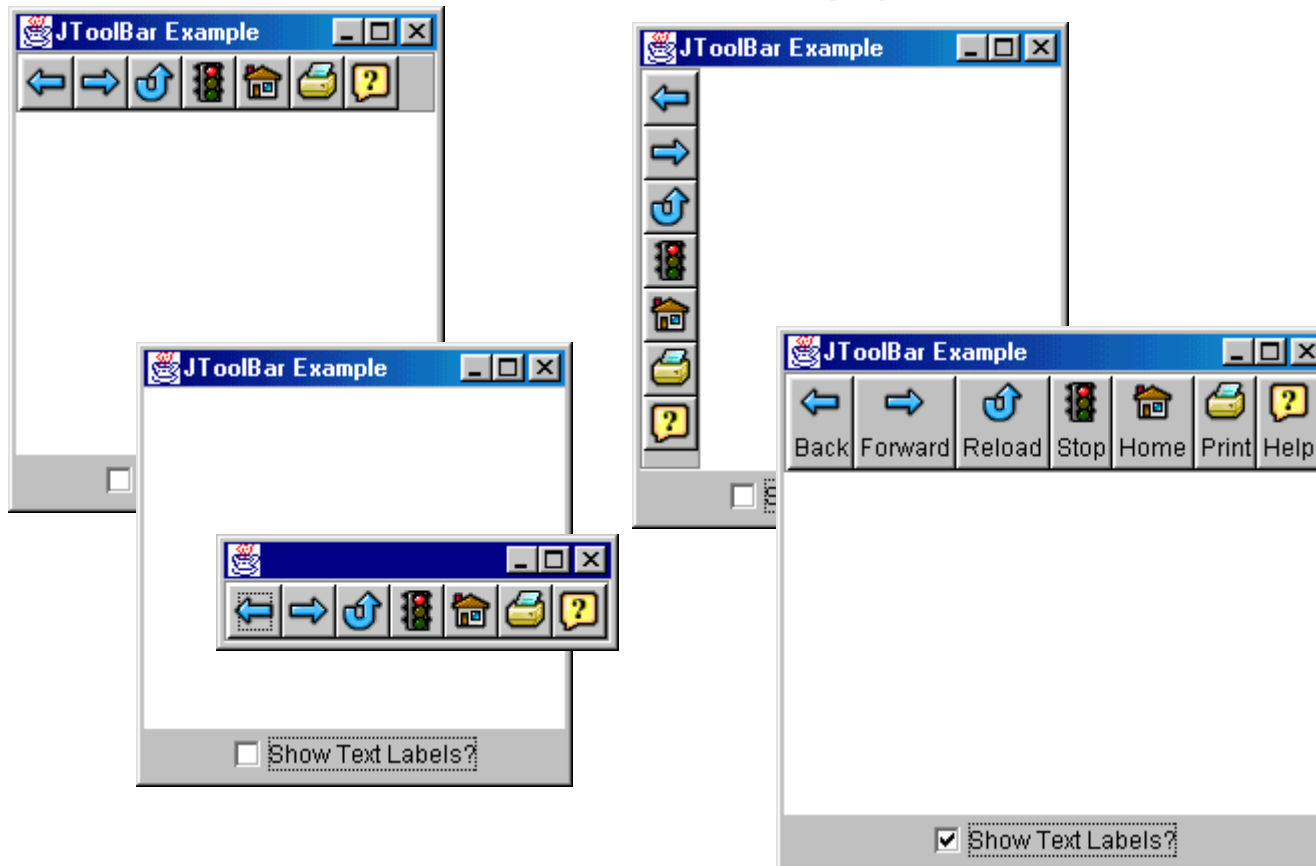


# JOptionPane Confirmation Dialogs (Java LAF)



# JToolBar

- Acts mostly like a JPanel for buttons
- Dockable: can be dragged and dropped



# JEditorPane

- Acts somewhat like a text area
- Can display HTML and, if `HyperLinkListener` attached, can follow links



# Other Simple Swing Components

- **JCheckBox**

- Note uppercase B (vs. Checkbox in AWT)



- **JRadioButton**

- Use a ButtonGroup to link radio buttons



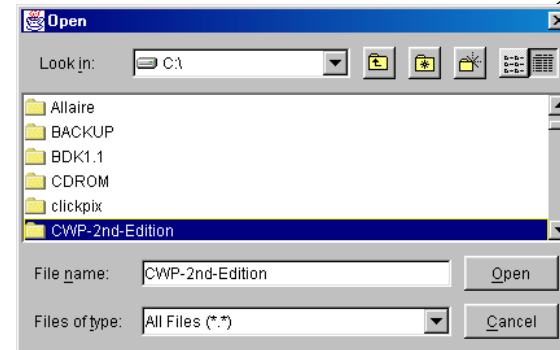
- **JTextField**

- Just like AWT TextField except that it does not act as a password field (use JPasswordField for that)

- **JTextArea**

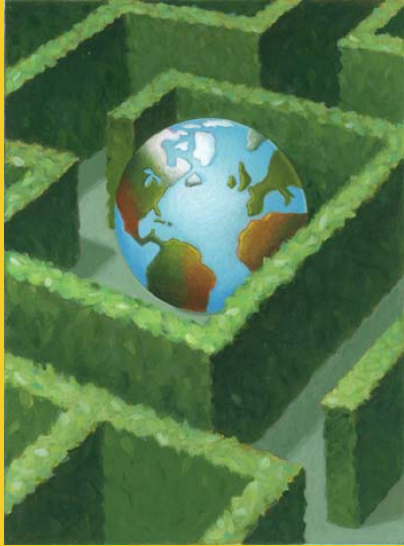
- Place in JScrollPane if you want scrolling

- **JFileChooser**



# Summary

- **Port simple AWT components by adding J to front of class name**
- **Put custom drawing in paintComponent**
  - Call `super.paintComponent` at beginning unless you turn off double buffering
- **Java look and feel is default**
  - But you almost always want native look and feel
- **Frames and applets use content pane**
  - Don't put anything directly in window
- **Most components support borders & icons**
- **Many new components**



*core*  
**WEB**  
*programming*

**Questions?**