PHẦN 2 - SWING

THIẾT KẾ GIAO DIỆN SỬ DỤNG SWING

Outline

- Swing Components and the Containment Hierarchy
- Layout Management
- Event Handling
- Painting
- More Swing Features and Concepts

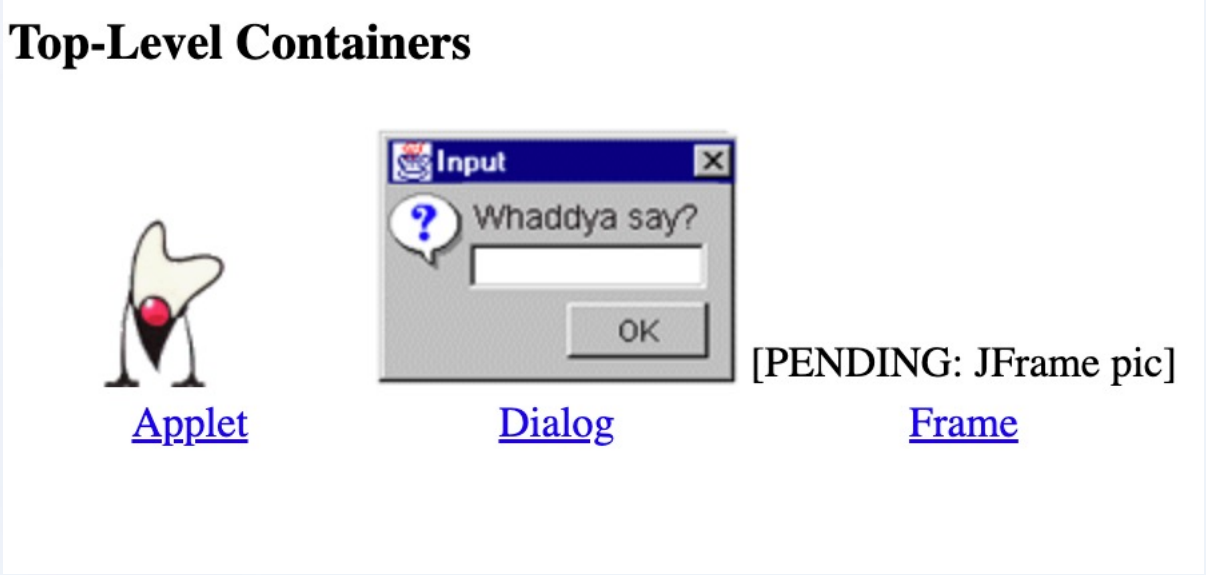
Introduction to Java Swing Components

- Swing containers can be classified into three main categories:
  - Top-level containers: JFrame, JWindow, and JDialog
  - General-purpose containers: JPanel, JScrollPane, JToolBar, JSplitPane, and JTabbedPane
  - Special-purpose containers: JInternalFrame and JLayeredPane

Introduction to Java Swing Components

- Top-Level Containers

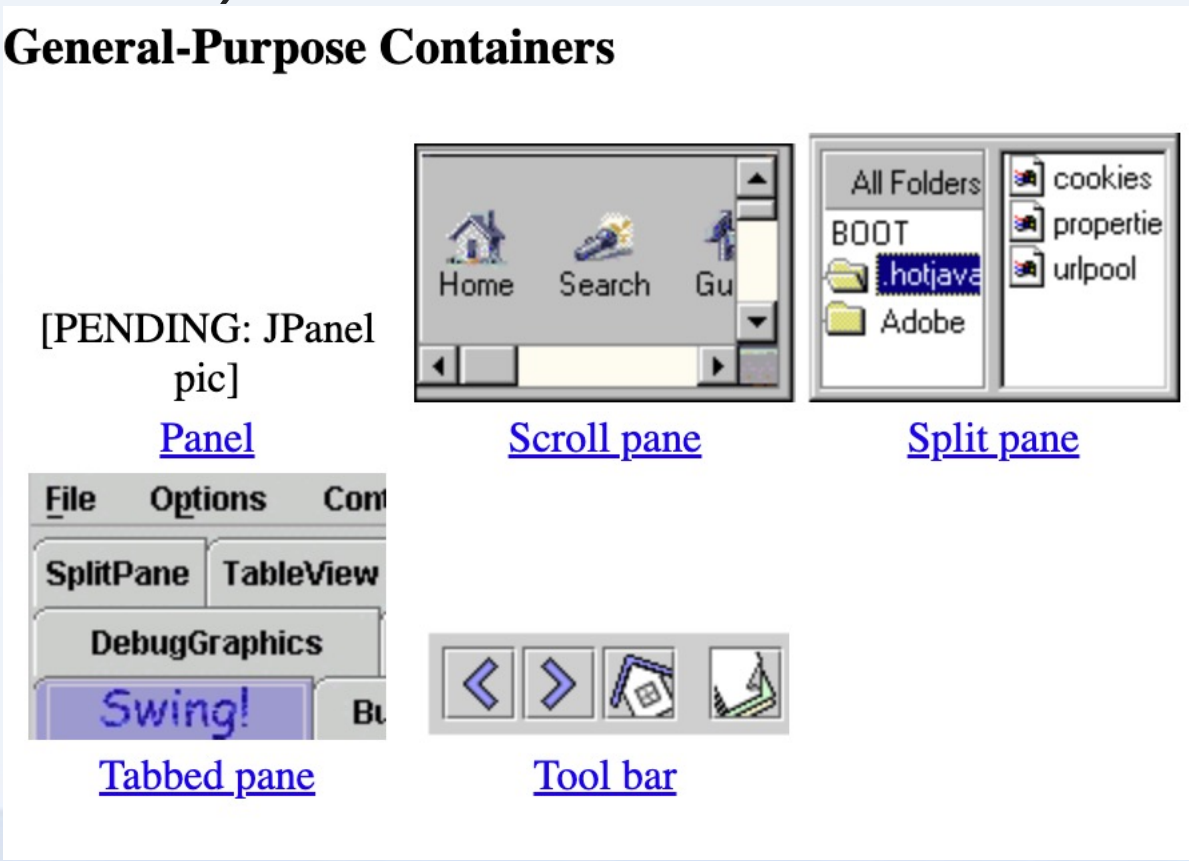
  The components at the top of any Swing containment hierarchy



Introduction to Java Swing Components

- General-Purpose Containers

  Intermediate containers that can be used under many different circumstances (JPanel, JScrollPane, JTabbedPane, JToolBar)

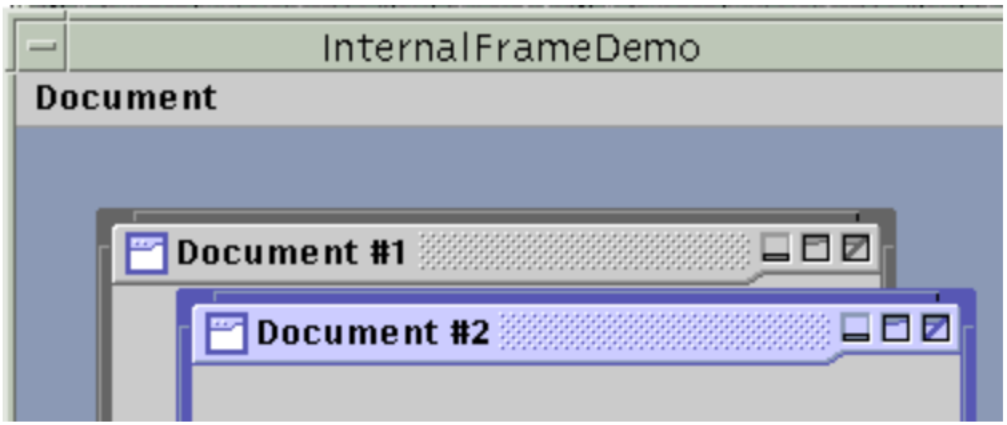


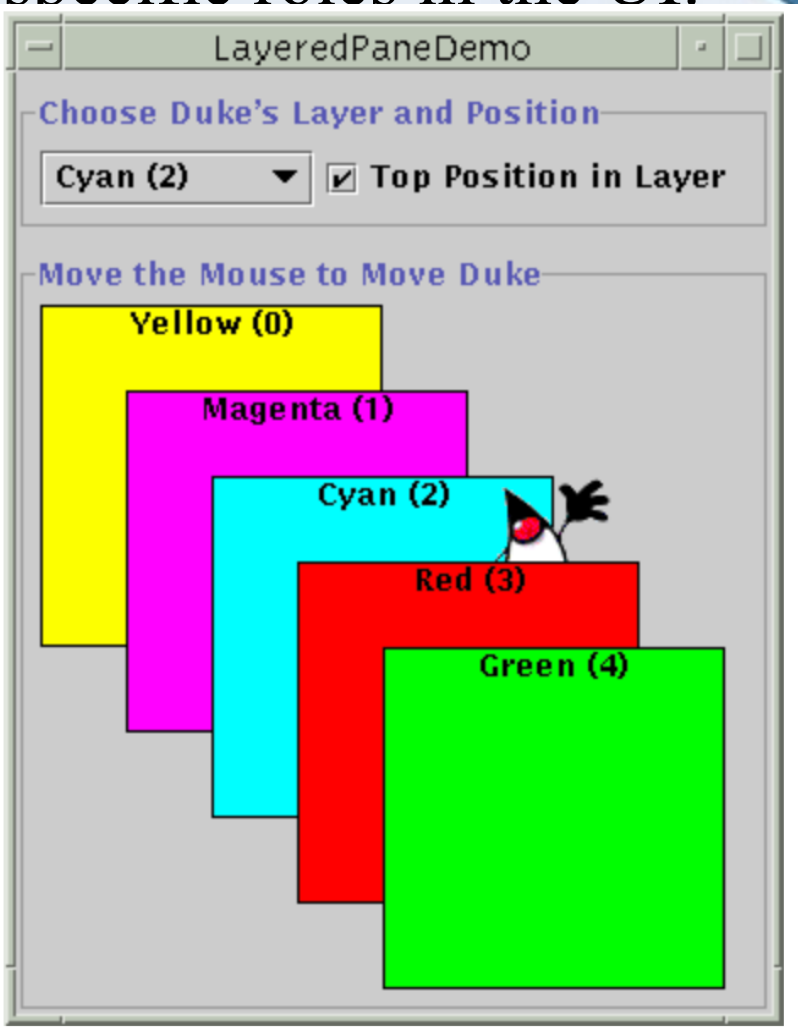
Introduction to Java Swing Components

- **Special-Purpose Containers**

  Intermediate containers that play specific roles in the UI.

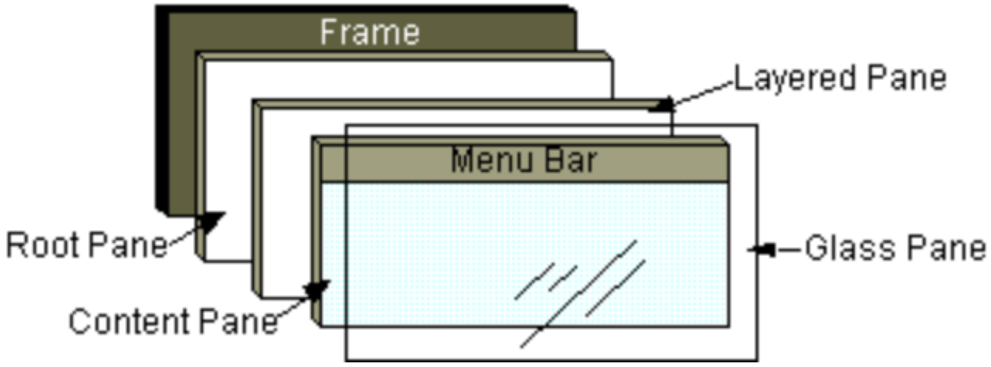
  - Internal frame (JInternalFrame)
  - Layered pane (JLayeredPane)
  - Root pane (JRootPane)

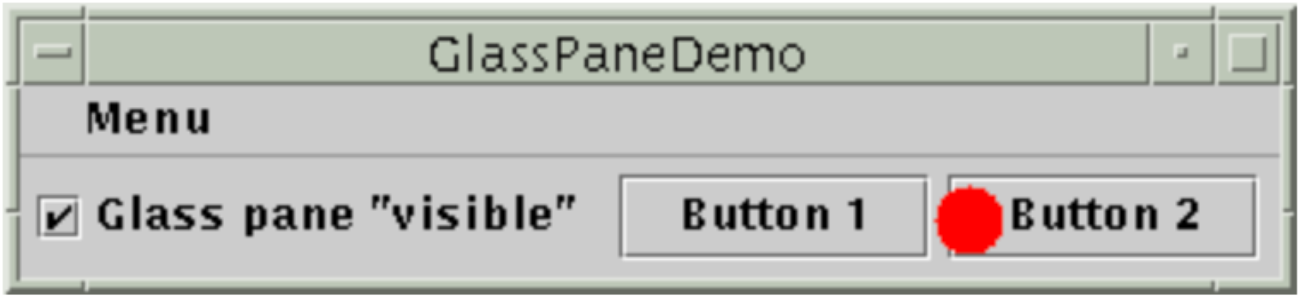




Introduction to Java Swing Components

- **Special-Purpose Containers**
  - Root pane (JRootPane)

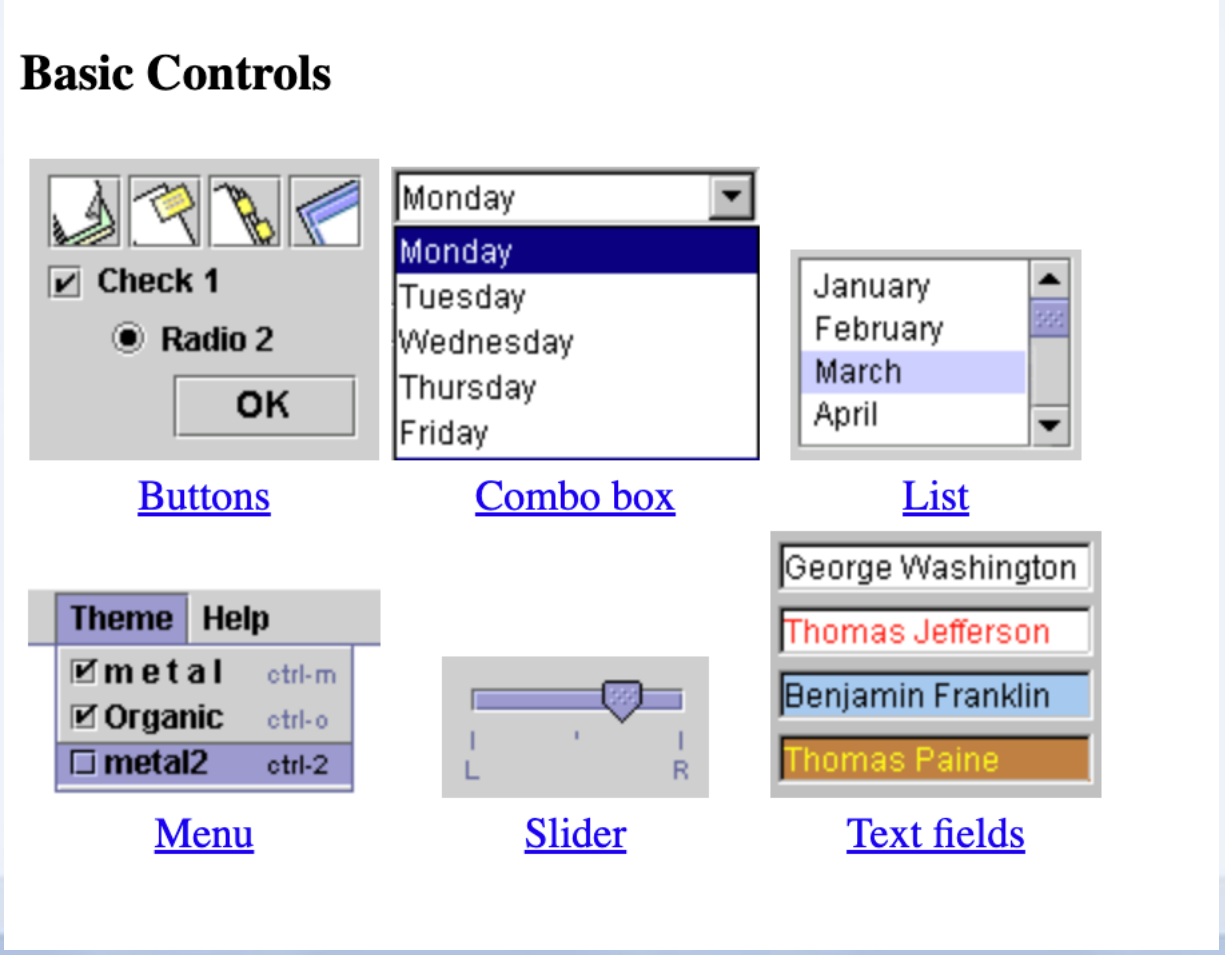




Introduction to Java Swing Components

- **Basic Controls**

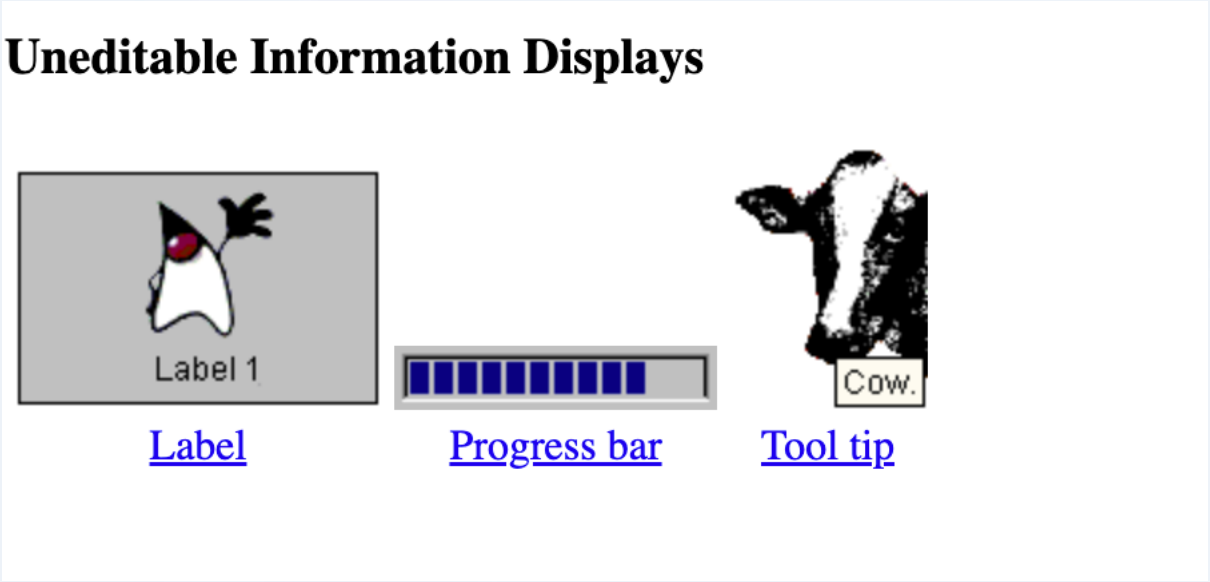
  Atomic components that exist primarily to get input from the user; they generally also show simple state.



Introduction to Java Swing Components

- **Uneditable Information Displays**

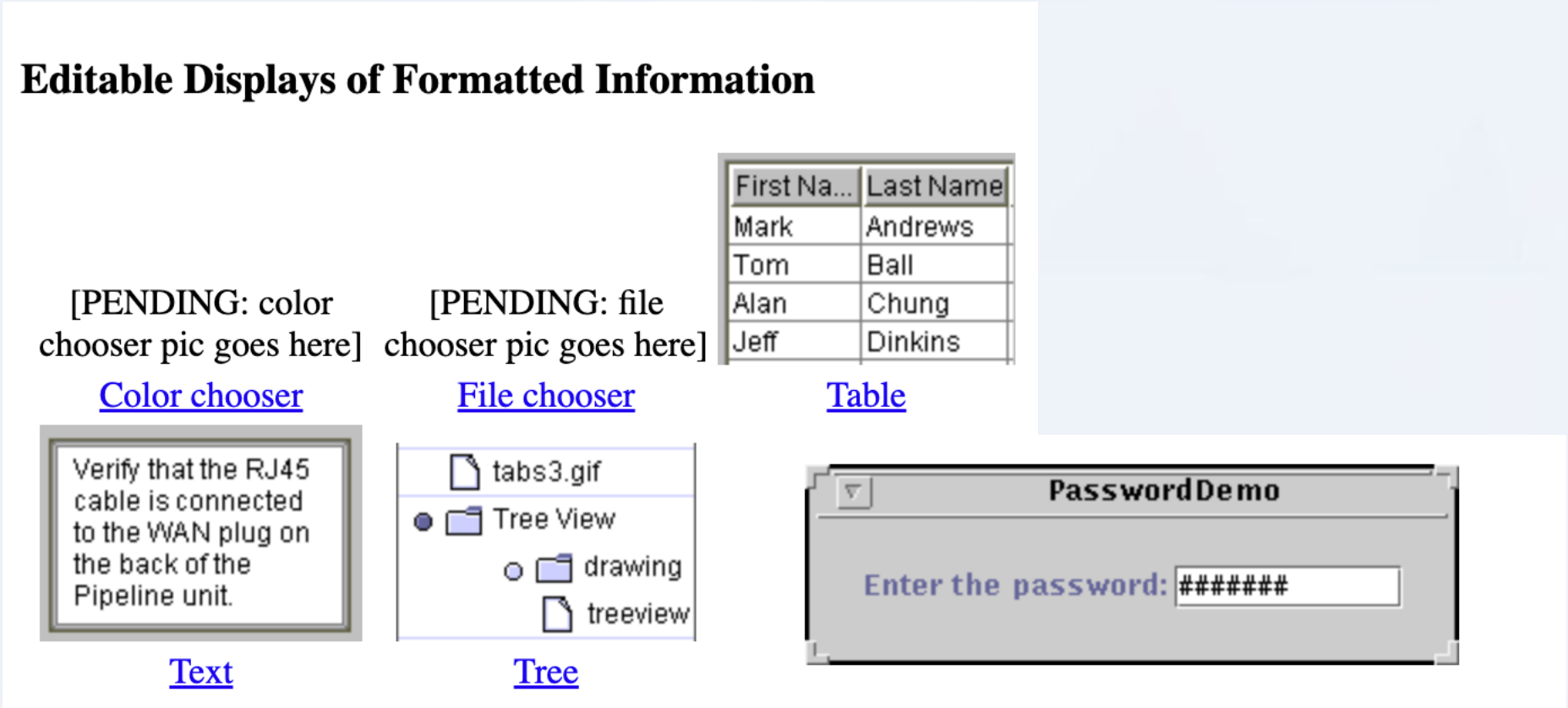
  Atomic components that exist solely to give the user information.



Introduction to Java Swing Components

- **Editable Displays of Formatted Information**

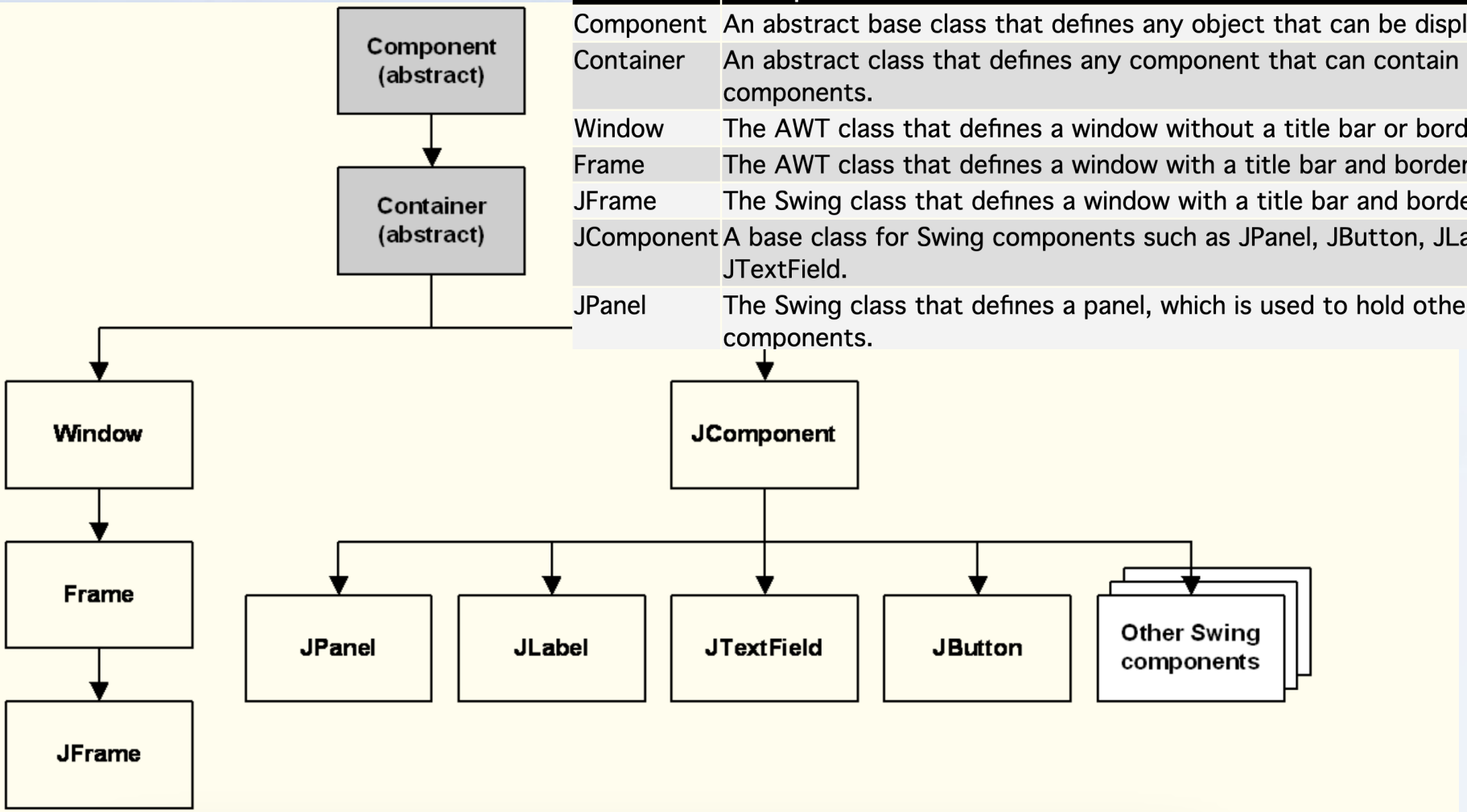
  Atomic components that display highly formatted information that (if you choose) can be edited by the user.



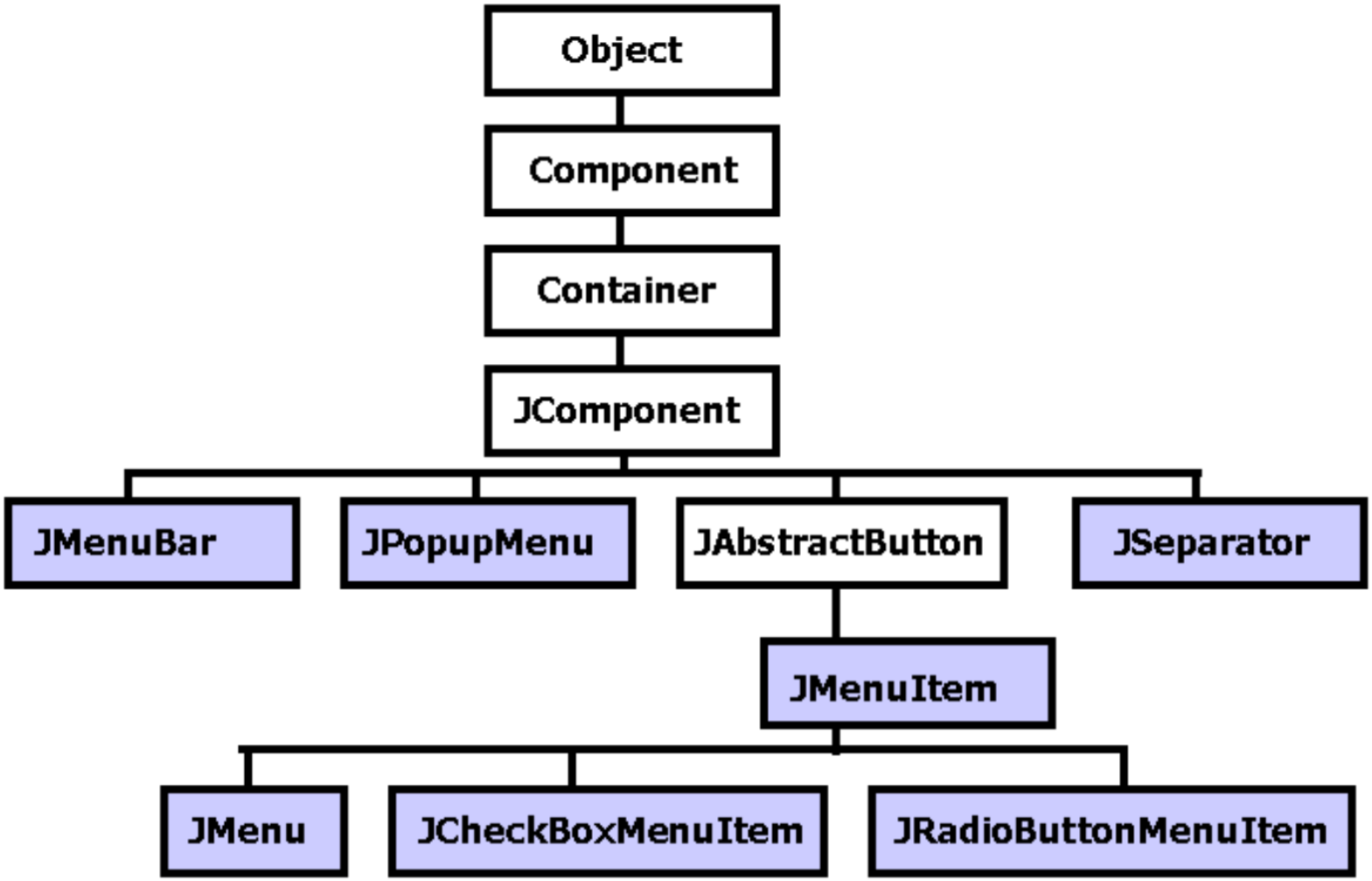
the Containment Hierarchy

A summary of the classes in the Component hierarchy

Class	Description
Component	An abstract base class that defines any object that can be displayed.
Container	An abstract class that defines any component that can contain other components.
Window	The AWT class that defines a window without a title bar or border.
Frame	The AWT class that defines a window with a title bar and border.
JFrame	The Swing class that defines a window with a title bar and border.
JComponent	A base class for Swing components such as JPanel, JButton, JLabel, a JTextField.
JPanel	The Swing class that defines a panel, which is used to hold other components.

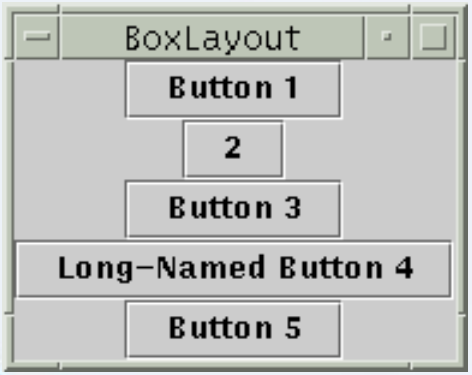


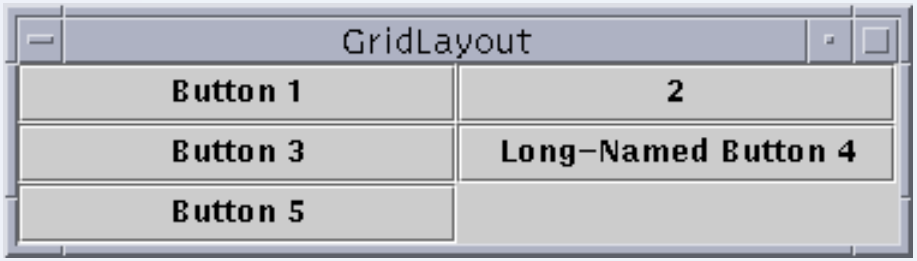
the Containment Hierarchy

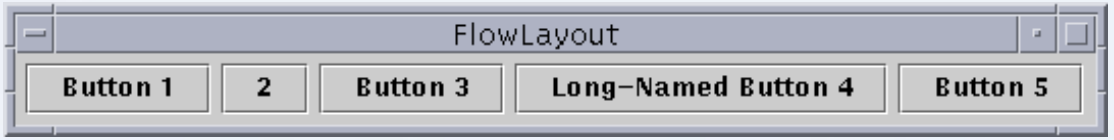


Layout Management

- Types of managers
  1. BorderLayout
  2. GridLayout
  3. BoxLayout
  4. GridBagLayou
  5. FlowLayout
  6. CardLayout







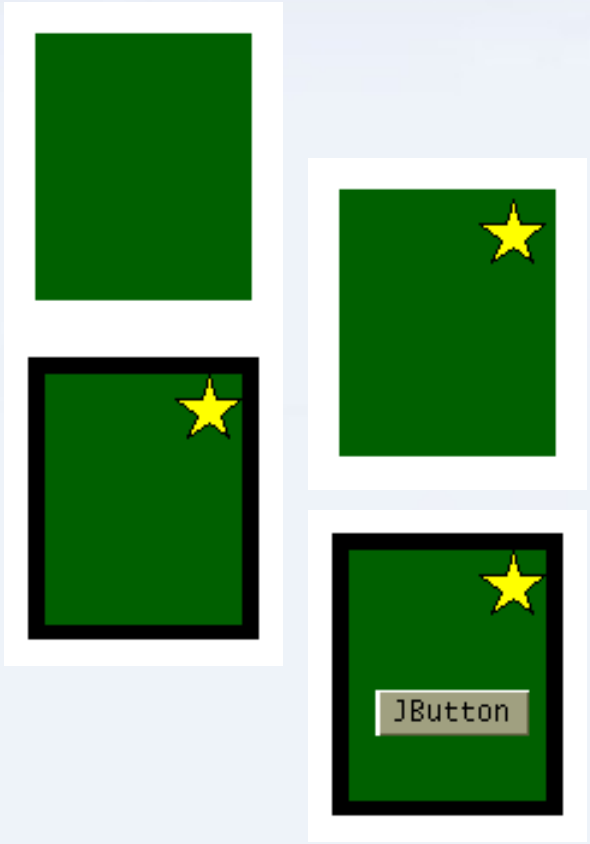
Event Handling

- Some typical component events and listeners

Act that results in event	Listener
User clicks a button, presses return while typing in a text field, or chooses a menu item	ActionListener
User closes a window	WindowListener
User presses a mouse button while the cursor is over a component	MouseListener
User moves the mouse over a component	MouseMotionListener
Component becomes visible	ComponentListener
Component gets the keyboard focus	FocusListener
Table or list selection changes	ListSelectionListener

Painting

- Visibility based on containment hierarchy
  - Background (if opaque)
  - Custom painting
  - Border
  - Child components



Your First Program


```
import javax.swing.*;
public class FirstProgram {
    private static void createAndShowGUI() {
        //Make sure we have nice window decorations.
        JFrame.setDefaultLookAndFeelDecorated(true);
        //Create and set up the window.
        JFrame frame = new JFrame("HelloWorld");

frame.setDefaultCloseOperation(Frame.EXIT_ON_CLOSE);
```

Your First Program

```
//Add the ubiquitous "Hello World" label.
    JLabel label = new JLabel("<html>  Hello World " +
    "<span style='font-size:18.0pt;color:red'>SWING </html>");
    frame.getContentPane().add(label);
    //Display the window.
    frame.pack();
    frame.setVisible(true);
}
public static void main(String[] args) {
    //Schedule a job for the event-dispatching thread:
    //creating and showing this application's GUI.
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            createAndShowGUI();
        }
    });
}}
```

The basic step in Swing Program

1. Import the pertinent packages.
2. Set up a top-level container.
3. Event handling

Step 1: Import the pertinent packages

The first line imports the main Swing package:

```
import javax.swing.*;
```

This is the only package that HelloWorldSwing needs. However, most Swing programs also need to import two AWT packages:

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

These packages are required because Swing components use the AWT infrastructure, including the AWT event model.

Step 2: Set up a top-level container

```
//Create a Top-Level Copntainer
JFrame frame = new JFrame("HelloWorld");
 ... Here is code construct and add the components


frame.pack(); //Calculate a frame size
frame.setVisible(true); //show a frame
```

HelloWorldSwing also has one component, a label that reads "Hello World." These two lines of code construct and then add the component to the frame:

```
//Create a Label component
final JLabel label = new JLabel("Hello World");
//Add the Label to a Content pane
frame.getContentPane().add(label);
```

Step 3: Event handling

To close the window when the close button is clicked, we include this code in our HelloWorldSwing program:

```
frame.setDefaultCloseOperation(
                                JFrame.EXIT_ON_CLOSE);
```

JFrame provides the setDefaultCloseOperation method to configure the default action for when the user clicks the close button. For single-window applications, most likely you want the application to exit.

Swing Application

The basic task :

- Importing Swing packages
- Setting up the top-level container
- Setting up buttons and labels
- Adding components to containers
- Adding borders around components
- Handling events

Swing Application

```
public class SwingApplication extends JFrame {
    private static String labelPrefix = "Number of button clicks: ";
    private int numClicks = 0;

  public SwingApplication(String title){
        super(title);
        //Setting up a button and label
        final JLabel label = new JLabel(labelPrefix + "0    ");
        JButton button = new JButton("I'm a Swing button!");
        button.setMnemonic(KeyEvent.VK_I);
        //Handling event
        button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                numClicks++;
                label.setText(labelPrefix + numClicks);
        }
  });
```

Swing Application

```
JPanel pane = new JPanel();
//Adding borders around components
pane.setBorder(BorderFactory.createEmptyBorder(30,30,10,30));
//Setting up the layout
pane.setLayout(new GridLayout(0, 1));
//Adding components to container
pane.add(button);
pane.add(label);
//Setting up the top-level container
getContentPane().add(pane);
//Handling event
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
//Calculating a frame size
pack();
//Show a frame
setVisible(true);
};
```

JButton Class

Constructor Summary

public JButton(Icon icon)
Creates a button with an icon.
Parameters:icon - the Icon image to display on the button `JButton`

public JButton(String text)
Creates a button with text.
Parameters:text - the text of the button

public JButton(String text, Icon icon)
Creates a button with initial text and an icon.
Parameters:text - the text of the button
icon - the Icon image to display on the button

JButton Class

Method Detail

public void setText(String text)
Sets the button's text.

public void setVerticalTextPosition(int textPosition)
Sets the vertical position of the text relative to the icon.

public void **setActionCommand**(String actionCommand) Sets the action command for this button.

JButton Class

Method Detail

public void **setMnemonic**(int mnemonic)

Sets the keyboard mnemonic on the current model. The mnemonic is the key which when combined with the look and feel's mouseless modifier (usually **Alt**) will activate this button if focus is contained somewhere within this button's ancestor window. A mnemonic must correspond to a single key on the keyboard and should be specified using one of the VK_XXX keycodes defined in java.awt.event.KeyEvent. Mnemonics are case-insensitive.

public void setMnemonic(char mnemonic)

This method is now obsolete, please use setMnemonic(int) to set the mnemonic for a button. This method is only designed to handle character values which fall between 'a' and 'z' or 'A' and 'Z'.

JButton Class

Method Detail

public void addActionListener(ActionListener l) Adds an `ActionListener` to the button.

public void **setEnabled**(boolean b)

Enables (or disables) the button. Overrides: setEnabled in class `JComponent`

Parameters:b - true to enable the button, otherwise false

Class ActionEvent

- **`public String getActionCommand()`**
  Returns the command string associated with this action.
- **`public int getModifiers()`**
  Returns the modifier keys held down during this action event.
- **`public Object getSource()`**
  The object on which the Event initially occurred.
- **`public static final int SHIFT_MASK`**
  The shift modifier. An indicator that the shift key was held down during the event.
- **`public static final int CTRL_MASK`**
  The control modifier. An indicator that the control key was held down during the event.
- **`public static final int ALT_MASK`**
  The alt modifier. An indicator that the alt key was held down during the event.

JLabel Class

Constructor Summary

public JLabel(String text)

Creates a `JLabel` instance with the specified text. The label is aligned against the leading edge of its display area, and centered vertically.

public JLabel(Icon image)

Creates a `JLabel` instance with the specified image. The label is centered vertically and horizontally in its display area.

public JLabel(String text,Icon icon,int horizontalAlignment)

Creates a `JLabel` instance with the specified text, image, and horizontal alignment. The label is centered vertically in its display area. The text is on the trailing edge of the image.

JLabel Class

Method Detail

public String getText()

Returns the text string that the label displays.

public void setText(String text)

Defines the single line of text this component will display. If the value of text is null or empty string, nothing is displayed.

public void setDisplayedMnemonic(int key)

Specify a keycode that indicates a mnemonic key. This property is used when the label is part of a larger component. If the labelFor property of the label is not null, the label will call the requestFocus method of the component specified by the labelFor property when the mnemonic is activated.

public void setDisplayedMnemonic(char aChar)

Specifies the displayedMnemonic as a char value.

JLabel Class

Method Detail

public void setVerticalTextPosition(int textPosition)

Sets the vertical position of the label's text, relative to its image. The default value of this property is CENTER. This is a JavaBeans bound property. Parameters:textPosition - One of the following constants defined in SwingConstants: TOP, CENTER (the default), or BOTTOM.

public void setHorizontalTextPosition(int textPosition)

Sets the horizontal position of the label's text, relative to its image.

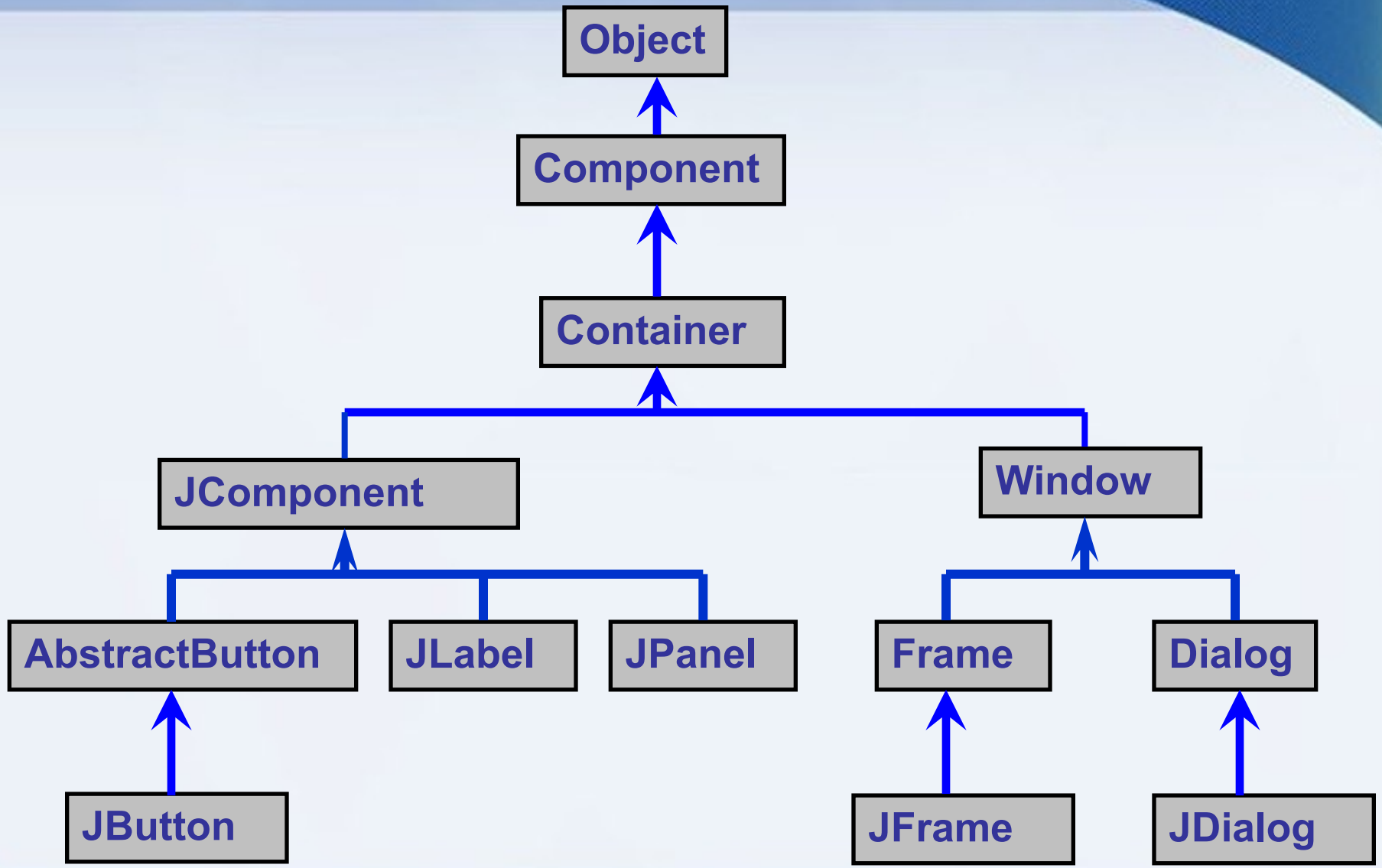
public void setLabelFor(Component c)

Set the component this is labelling. Can be null if this does not label a Component. If the displayedMnemonic property is set and the labelFor property is also set, the label will call the requestFocus method of the component specified by the labelFor property when the mnemonic is activated.

Parameters:c - the Component this label is for, or null if the label is not the label for a component

JFRAME

Inheritance hierarchy

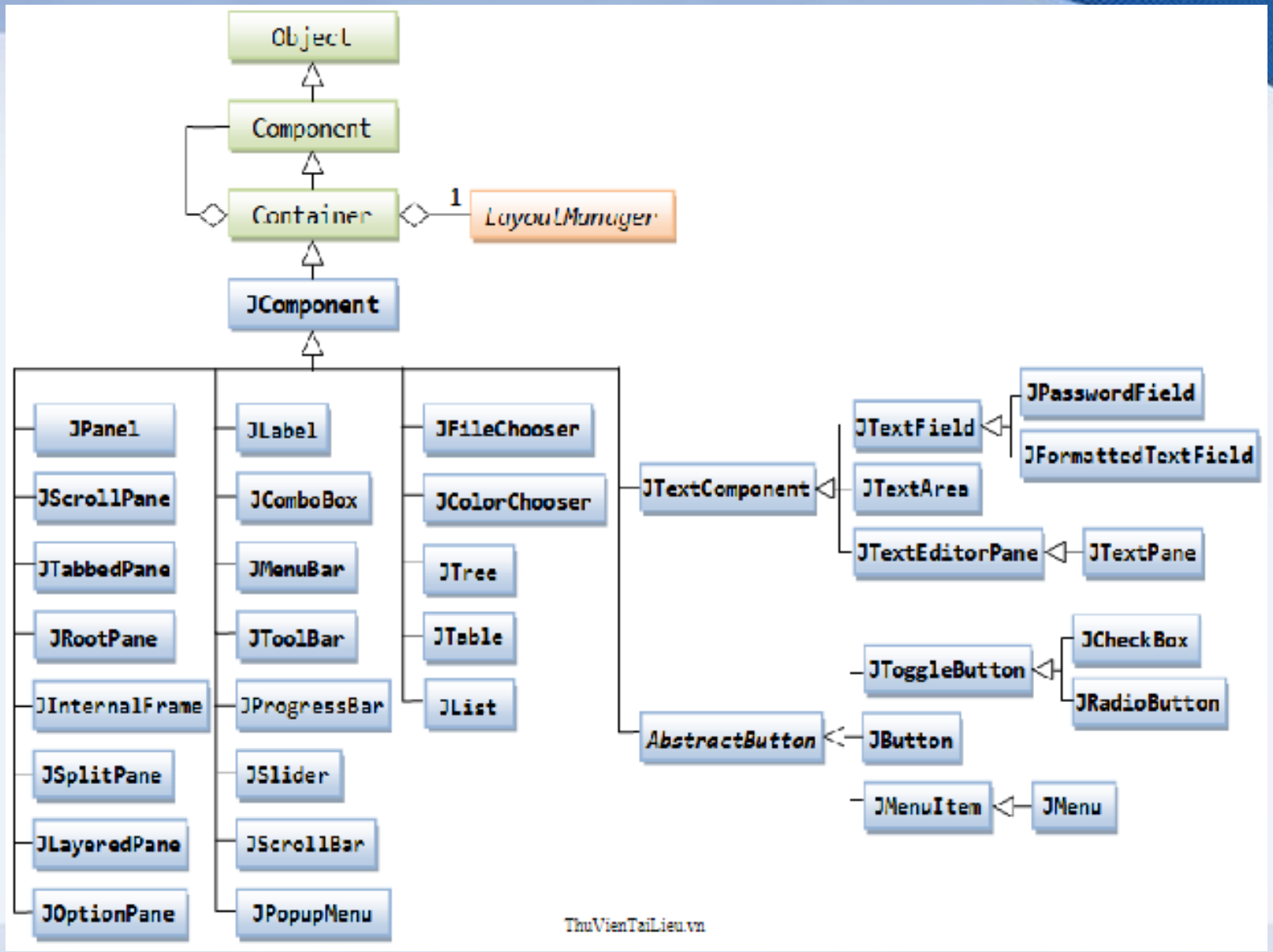


Inheritance hierarchy

A summary of the classes in the Component hierarchy

Class	Description
Component	An abstract base class that defines any object that can be displayed.
Container	An abstract class that defines any component that can contain other components.
Window	The AWT class that defines a window without a title bar or border.
Frame	The AWT class that defines a window with a title bar and border.
JFrame	The Swing class that defines a window with a title bar and border.
JComponent	A base class for Swing components such as JPanel, JButton, JLabel, and JTextField.
JPanel	The Swing class that defines a panel, which is used to hold other components.
JLabel	The Swing class that defines a label.
JTextField	The Swing class that defines a text field.
JButton	The Swing class that defines a button.

Inheritance hierarchy



Constructor Summary

- JFrame()
  Constructs a new frame that is initially invisible.
- JFrame(String title)
  Creates a new, initially invisible Frame with the specified title.

Method Summary

- protected void frameInit()
  Called by the constructors to init the JFrame properly.
- public void setDefaultCloseOperation(int operation)
  Sets the operation that will happen by default when the user initiates a "close" on this frame. You must specify one of the following choices:
  - DO_NOTHING_ON_CLOSE (defined in WindowConstants): The do-nothing default window close operation.
  - HIDE_ON_CLOSE (defined in WindowConstants): The hide-window default window close operation
  - DISPOSE_ON_CLOSE (defined in WindowConstants): The dispose-window default window close operation.
  - EXIT_ON_CLOSE (defined in Jframe): The exit application default window close operation.

Method Summary

- public int getDefaultCloseOperation()
  Returns the operation that occurs when the user initiates a "close" on this frame.
- public void setJMenuBar(JMenuBar menubar)
  Sets the menubar for this frame.
- public JMenuBar getJMenuBar()
  Returns the menubar set on this frame.
- public Container getContentPane()
  Returns the contentPane object for this frame.

Method Summary

- public JLayeredPane getLayeredPane()
  Returns the layeredPane object for this frame.
- public void setLayeredPane(JLayeredPane layeredPane)
  Sets the layeredPane property. This method is called by the constructor.
- public Component getGlassPane()
  Returns the glassPane object for this frame.
- public void setGlassPane(Component glassPane)
  Sets the glassPane property. This method is called by the constructor.

Method Summary

- public String getTitle()
  Gets the title of the frame. The title is displayed in the frame's border.
- public void setTitle(String title)
  Sets the title for this frame to the specified string.
- public boolean isResizable()
  Indicates whether this frame is resizable by the user. By default, all frames are initially resizable.
- public void setResizable(boolean resizable)
  Sets whether this frame is resizable by the user.

Method Summary

- public void setCursor(Cursor cursor)
  !!Component method
  Sets the cursor image to the specified cursor. This cursor image is displayed when the contains method for this component returns true for the current cursor location, and this Component is visible, displayable, and enabled. Setting the cursor of a Container causes that cursor to be displayed within all of the container's subcomponents, except for those that have a non-null cursor.

  Parameters:cursor - One of the constants defined by the Cursor class; if this parameter is null then this component will inherit the cursor of its parent
  CROSSHAIR_CURSOR, TEXT_CURSOR, WAIT_CURSOR, HAND_CURSOR, MOVE_CURSOR

Method Summary - Frame Positioning

- public void setLocation(int x, int y)
  !!Component method
  Moves this component to a new location. The top-left corner of the new location is specified by the x and y parameters in the coordinate space of this component's parent.
  Parameters:x - the x-coordinate of the new location's top-left corner in the parent's coordinate spacey - the y-coordinate of the new location's top-left corner in the parent's coordinate space.
- public void setLocationRelativeTo(Component c)
  !!Window Method
  Sets the location of the window relative to the specified component. If the component is not currently showing, or c is null, the window is centered on the screen. If the bottom of the component is offscreen, the window is displayed to the right of the component.

Method Summary - Frame Positioning

- public void setBounds(int x, int y, int width, int height) !!Component Method

  Moves and resizes this component. The new location of the top-left corner is specified by x and y, and the new size is specified by width and height.

Toolkit Class

- public static Toolkit getDefaultToolkit()
  Gets the default toolkit.
- public abstract Dimension getScreenSize()
  Gets the size of the screen.
  Returns:the size of this toolkit's screen, in pixels.
- public abstract Image getImage(String filename)
  Returns an image which gets pixel data from the specified file, whose format can be either GIF, JPEG or PNG.

DemoFrameTest.java

```
package frame;
import javax.swing.*;
public class DemoFrameTest {
  public static void main(String[] args) {
     JFrame frame = new DemoFrame();
     frame.setDefaultCloseOperation(
                                    JFrame.EXIT_ON_CLOSE);
     frame.setVisible(true);
  }
}
```

DemoFrame.java

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

public class DemoFrame extends JFrame{
  Toolkit kit;
  JButton locationButton, cursorButton, iconButton;
  public DemoFrame() {
     kit = Toolkit.getDefaultToolkit();
     Dimension screenSize = kit.getScreenSize();
     int screenHeight = screenSize.height;
     int screenWidth = screenSize.width;
     setSize(screenWidth / 2, screenHeight / 2);
     setTitle("untitled Frame");
     setResizable(false);
```

DemoFrame.java

```
Container pane = getContentPane();
pane.setLayout(new FlowLayout());
locationButton = new JButton(" Center a Frame");
locationButton.addActionListener(new ActionListener(){
  public void actionPerformed(ActionEvent e) {
    setLocationRelativeTo(null);
    setTitle("a Centered Frame");
  }});

cursorButton = new JButton(” Set Cursor");
cursorButton.addActionListener(new ActionListener(){
  public void actionPerformed(ActionEvent e) {
    setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
    setTitle("a Cross-Hair cursor");
  }});
```

DemoFrame.java

```
 iconButton = new JButton(" set Frame Icon");
iconButton.addActionListener(new ActionListener(){
   public void actionPerformed(ActionEvent e) {
      Image img = kit.getImage("image/bird.gif");
      setIconImage(img);
      setTitle("a Bird icon");
  }});
pane.add(locationButton);
pane.add(cursorButton);
pane.add(iconButton);
setVisible(true);
  }
}
```

API - java.awt.Component

- boolean isVisible()
  checks if this component is set to be visible. Components are initially visible, with the exception of top-level components such as JFrame.
- void setVisible(boolean b)
  shows or hides the component depending on whether b is true or false.
- boolean isShowing()
  checks if this component is showing on the screen. For this, it must be visible and be inside a container that is showing.
- boolean isEnabled()
  checks if this component is enabled. An enabled component can receive keyboard input. Components are initially enabled.
- void setEnabled(boolean b)
  enables or disables a component.

API - java.awt.Component

- Point getLocation()
  returns the location of the top-left corner of this component, relative to the top-left corner of the surrounding container. (A `Point` object p encapsulates an `x`- and a `y`-coordinate which are accessible by `p.x` and `p.y`.)
- Point getLocationOnScreen()
  returns the location of the top-left corner of this component, using the screen's coordinates.
- void setBounds(int x, int y, int width, int height)
  moves and resizes this component. The location of the top-left corner is given by x and y, and the new size is given by the width and height parameters.

API - java.awt.Component

- void setLocation(int x, int y)
- void setLocation(Point p)
  move the component to a new location. The x- and y-coordinates (or `p.x` and `p.y`) use the coordinates of the container if the component is not a top-level component, or the coordinates of the screen if the component is top level (for example, a `JFrame`).
- Dimension getSize()
  gets the current size of this component.
- void setSize(int width, int height)
- void setSize(Dimension d)
  resize the component to the specified width and height.

API - java.awt.Frame

- void setResizable(boolean b)
  determines whether the user can resize the frame.
- void setTitle(String s)
  sets the text in the title bar for the frame to the string s.
- void setIconImage(Image image)
  Parameters: image, The image you want to appear as the icon for the frame
- .....

API - java.awt.Toolkit

- static Toolkit getDefaultToolkit()
  returns the default toolkit.
- Dimension getScreenSize()
  gets the size of the user's screen.
- Image getImage(String filename)
  loads an image from the file with name filename.