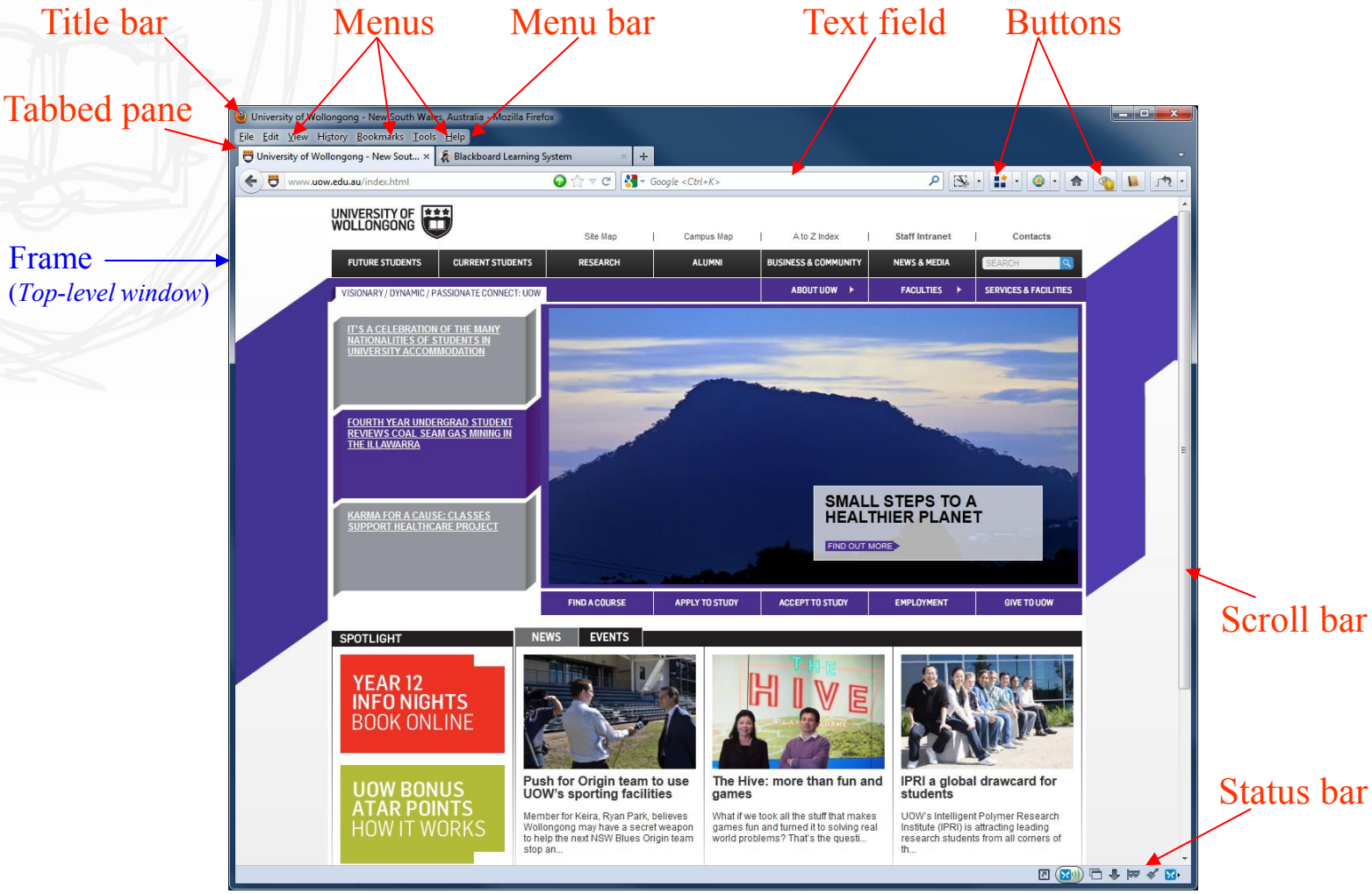
GUI Programming I

GUI Components

An Example of GUI – Components



UNIVERSITY OF WOLLONGONG

Concepts of GUI Programming

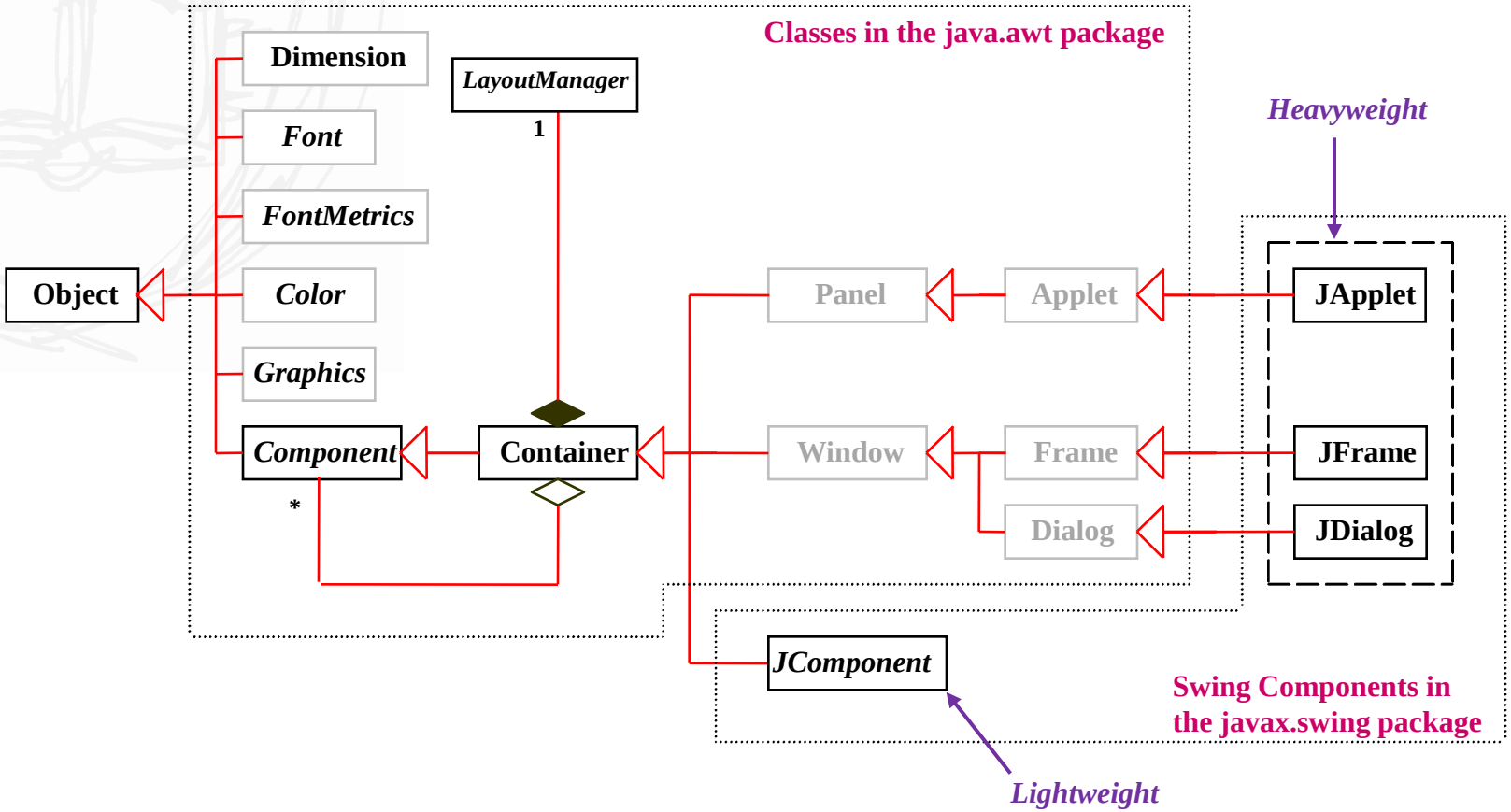
- GUI components
  - How to use them to construct GUI
    - Swing components
- Event handling
  - How to interact with them
    - Actions, listeners and event delegation model
- Layout management
  - How to visually arrange them
    - Layout managers

- MVC design pattern
  - How to efficiently program them
    - Model, view and controller
    - Content, visual appearance and behaviour

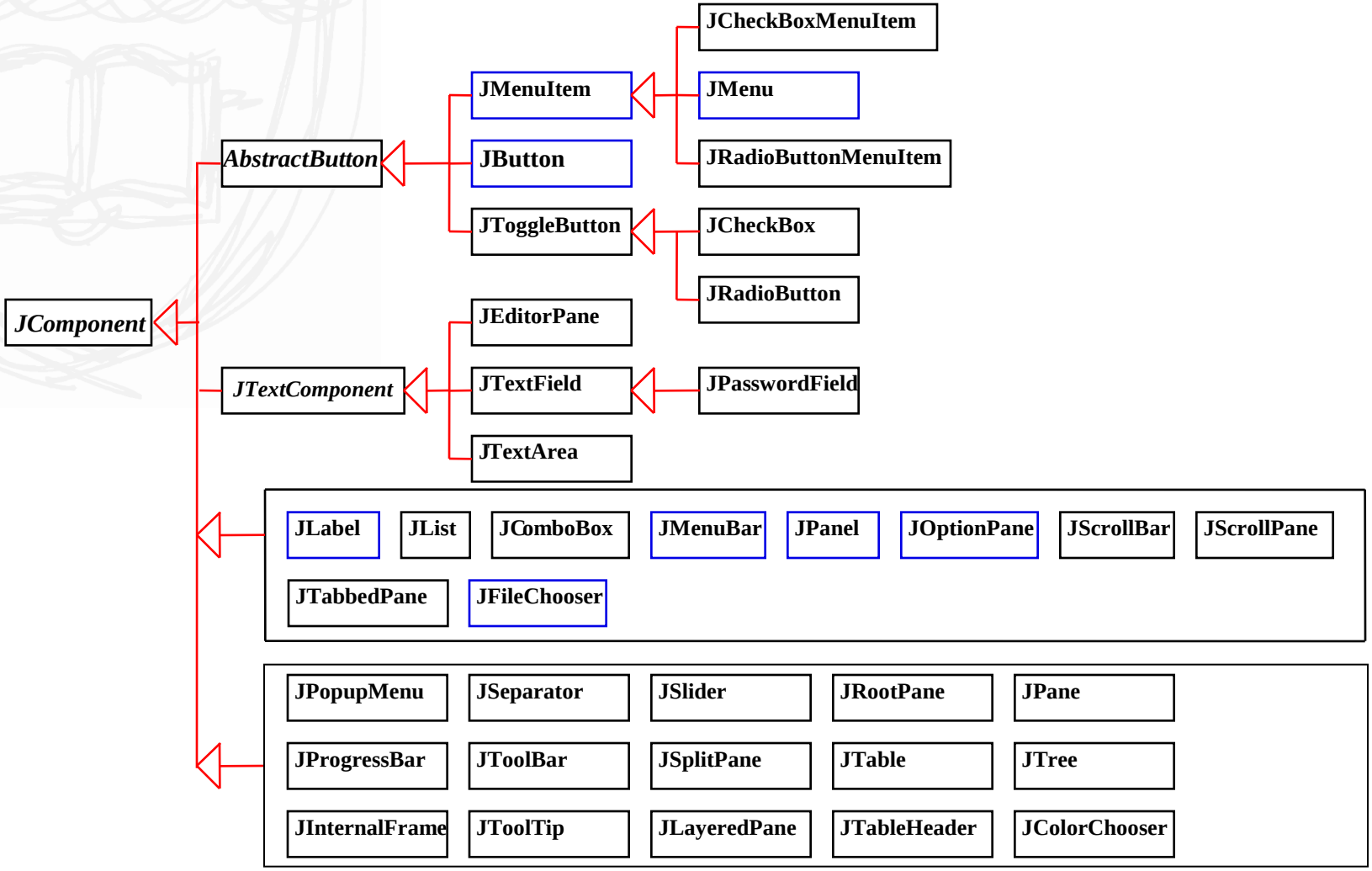
Overview of GUI Components

- Container Classes
  - To contain other components
    - **`JFrame, JPanel, Japplet`**
- Component Classes
  - UI components
    - **`JButton, JMenu, JTextArea`** etc.
- Helper Classes
  - Used by components and containers to draw and place objects
    - **`Graphics, Color, Font, LayoutManager`** etc

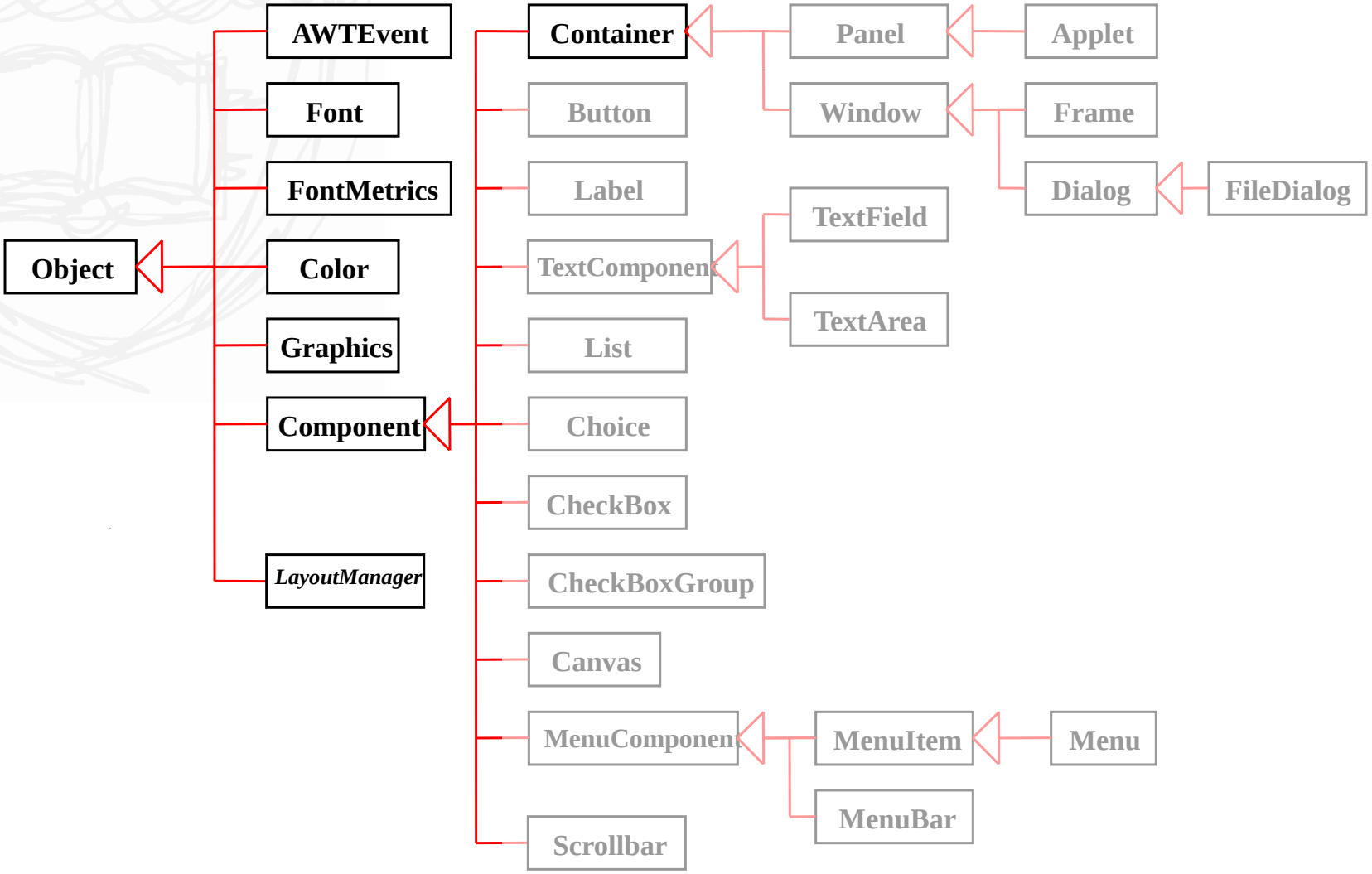
Java GUI Class Hierarchy



JComponent and Basic GUI Classes



AWT GUI Components



AWT and Swing GUI Components

- AWT - Abstract Window Toolkit
  - Delegate GUI component creation and behaviour to *native GUI toolkit* on each platform (Windows, Solaris, Macintosh, Linux/unix)
  - Hard to have consistent and predicable experience
  - Some platforms do not have rich collection of GUI components
  - Fine for simple GUI
  - Prone to platform-specific bugs
- Swing
  - Much richer and more convenient set of GUI components (>250 classes)
  - Class names start with J, replacing AWT components with the same name
  - Depends far less on the underlying platform
  - Lightweight components written in Java
  - Consistent user experience across platforms
  - Pluggable Look and Feel
  - Always use **`paintComponent()`** method, not **`paint()`** method

Recommendation: Use Swing for GUI construction

Example 1: Creating a Frame

```
import javax.swing.*;

public class HelloWorld {
    public static void main(String[] args){

        //Create and set up the frame (window).
①       JFrame frame = new JFrame();
②       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(350, 200);

        //Create and add the label
③       JLabel helloLabel = new JLabel("Hello World !", JLabel.CENTER);
        frame.add(helloLabel);

        //Display the frame
④       frame.setVisible(true);
    }
}
```

Example 2: Creating a Frame

```
import javax.swing.*;

public class HelloFrameWorld extends JFrame {

    public static void main(String args[]) {
         new HelloFrameWorld();  //The frame (window) is created
    }

    HelloFrameWorld() {
        //Set up the frame (window).
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(350, 200);
        //Create and add the label
        JLabel helloLabel = new JLabel("Hello Frame World !",
        JLabel.CENTER);
        add(helloLabel);
        //Display the frame
        setVisible(true);
    }
}
```

Example 3: Creating a Frame

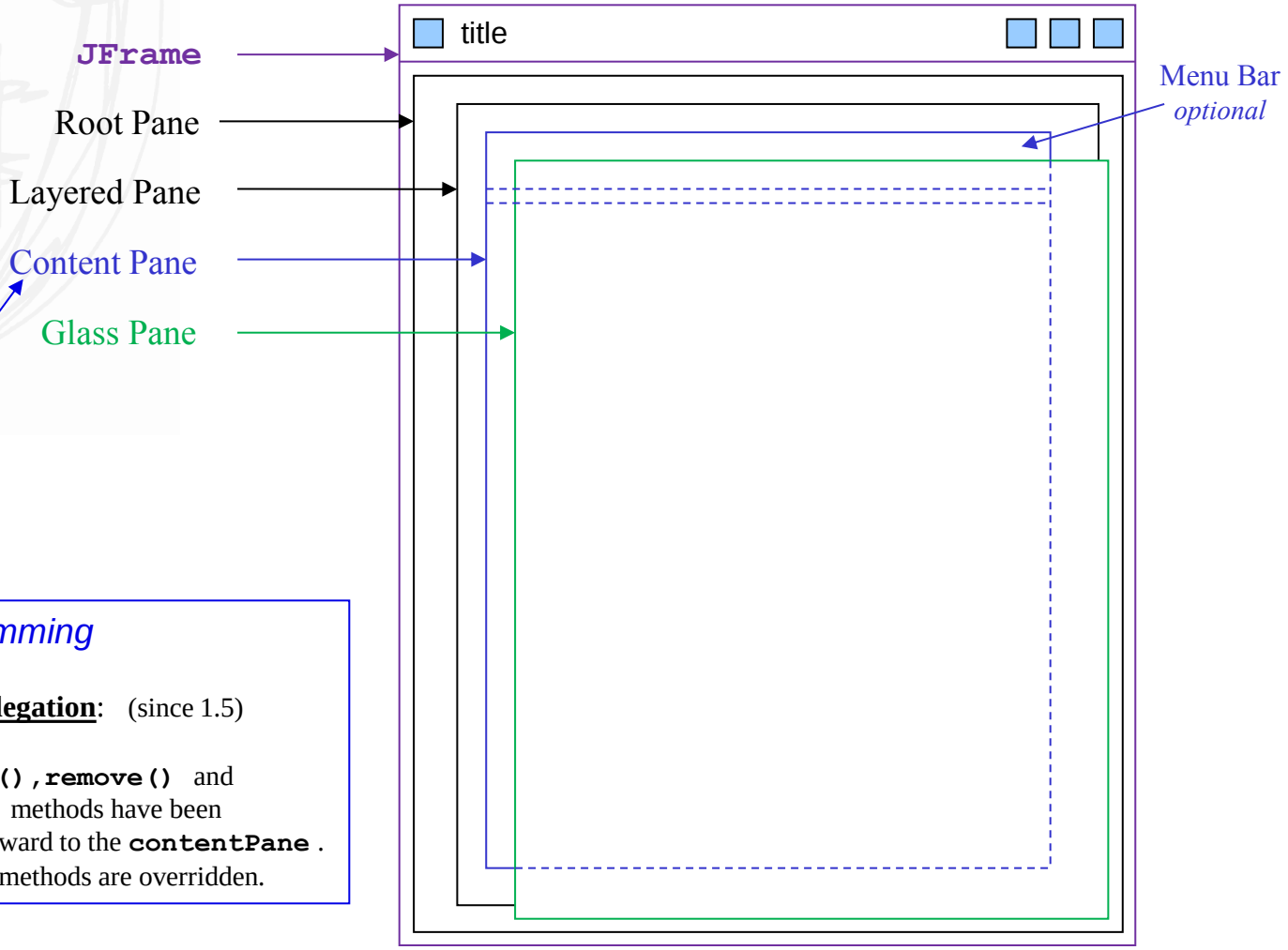
```
import javax.swing.*;

public class HelloClassWorld {

    public static void main(String[] args) {
        //Create and set up your own frame (window).
        HelloFrame frame = new HelloFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //Display the frame
        frame.setVisible(true);
    }
}

class HelloFrame extends Jframe {
    public HelloFrame(){
        setSize(350, 200);
        //Create and add the label
        JLabel helloLabel = new JLabel("Hello Class World !", JLabel.CENTER);
        add(helloLabel);
    }
}
```

Example 4: Creating a Frame

```
import javax.swing.*;

public class HelloSafeWorld {
    // Create the GUI and show it.  For thread safety, this method should be invoked
    // from the event-dispatching thread.
    private static void createAndShowGUI() {
        //Create and set up the frame (window).
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(350, 200);
        //Create and add the label
        JLabel helloLabel = new JLabel("Hello Safe World !", JLabel.CENTER);
        frame.add(helloLabel);
        //Display the frame
        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();
            }
        });
    }
}
```

Do this in your product code

Internal Structure of `JFrame`

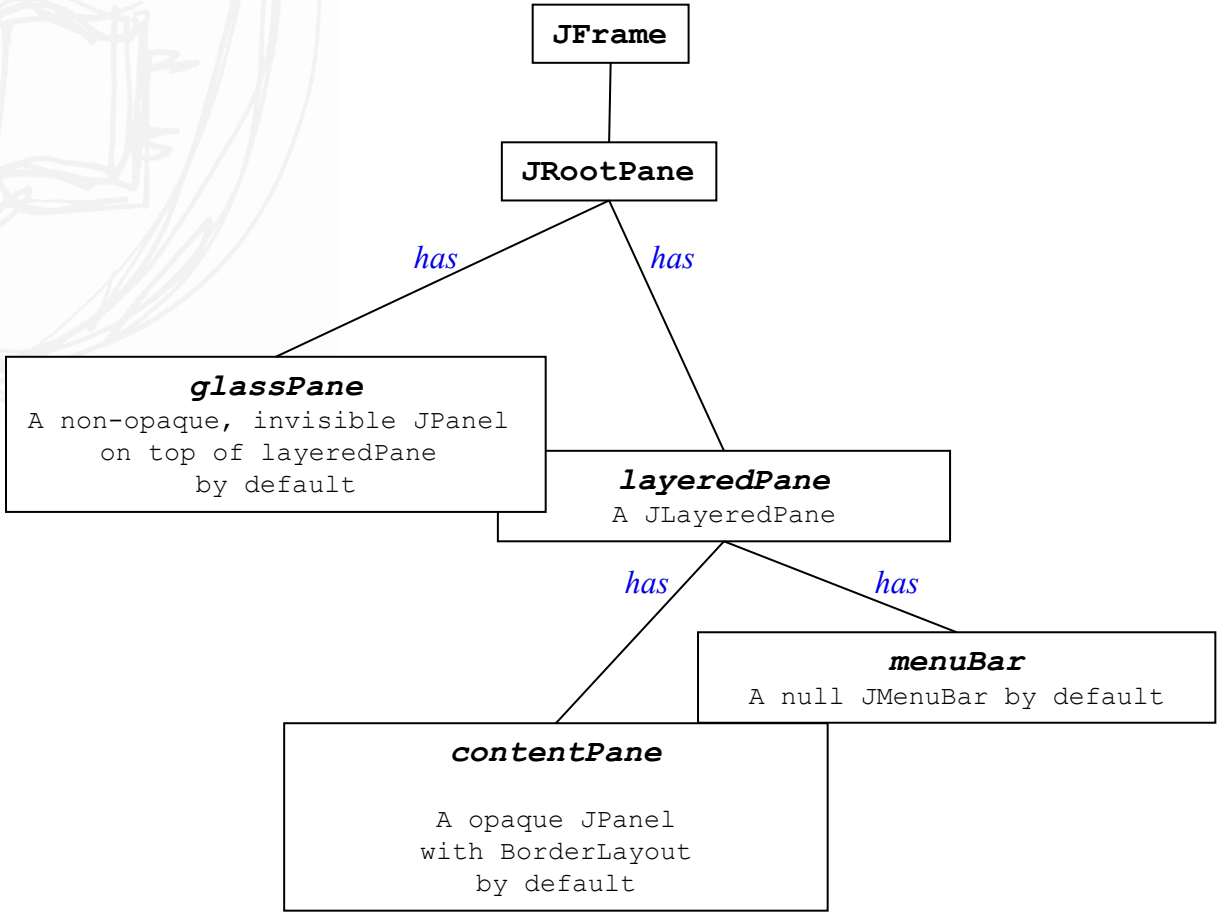


Swing Programming

Content Pane Delegation: (since 1.5)

- **`JFrame`**'s **`add(),remove()`** and **`setLayout()`** methods have been overridden to forward to the **`contentPane`** .
- Only these three methods are overridden.

Hierarchy of a `JFrame`



glassPane , ***layeredPane***, ***contentPane*** and ***menuBar*** are all variable names used by **JRootPane**

JFrame methods

javax.swing.JFrame
+JFrame() +JFrame(title: String) +setSize(width: int, height: int): void +setLocation(x: int, y: int): void +setVisible(visible: boolean): void +setDefaultCloseOperation(mode: int): void +setLocationRelativeTo(c: Component): void +pack(): void +setIconImage(image: Image): void +setTitle(title: String): void +setDefaultLookAndFeelDecorated(decorated: boolean): void +getContentPane(): Container +setJMenuBar(menuBar: JMenuBar): void +getGlassPane(): Component

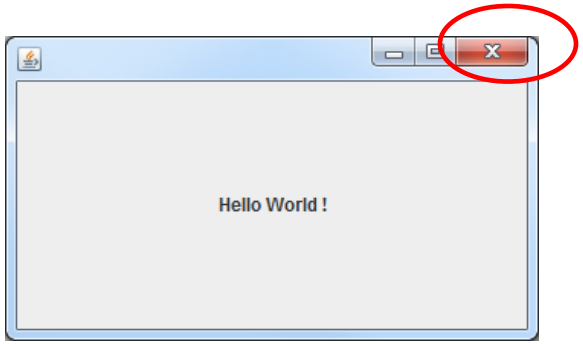
Some methods are omitted

Responding to Window-Closing

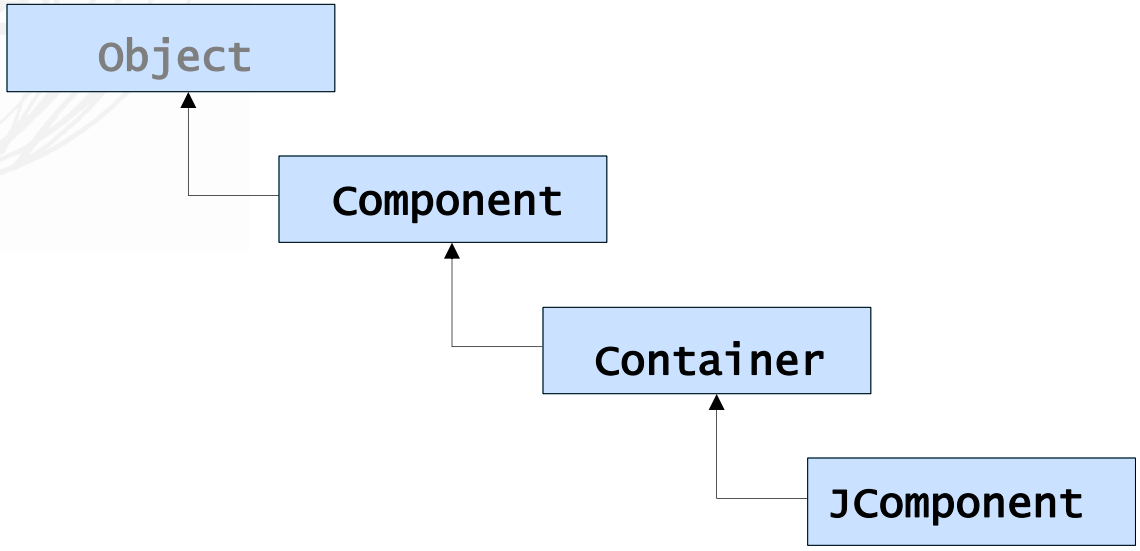
- By default, when the user closes a frame onscreen, the frame is hidden. The program ***does not*** terminate !
  - Although invisible, the frame still exists and the program can make it visible again
- Specify default close behaviour using

```
setDefaultCloseOperation()
```

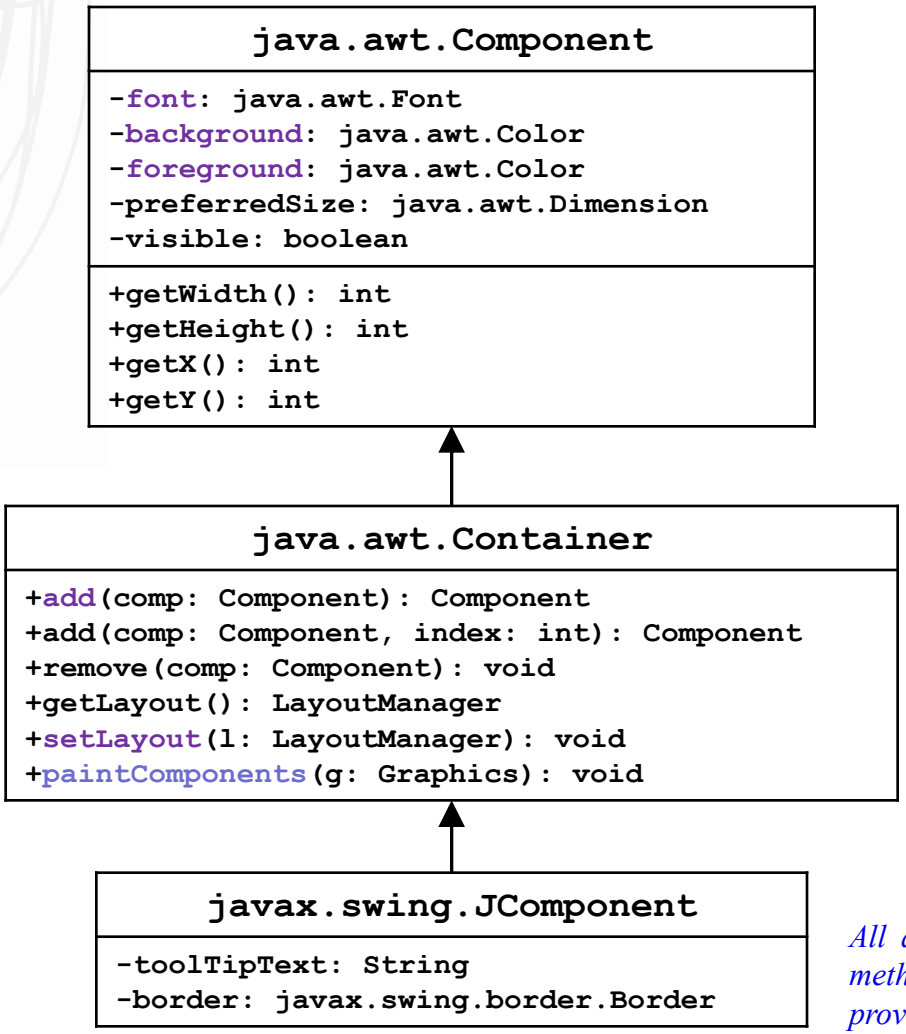
- Window constants:
  - **`DO_NOTHING_ON_CLOSE`**
  - **`HIDE_ON_CLOSE`**
    - the default for **`JDialog`** and **`JFrame`**
  - **`DISPOSE_ON_CLOSE`**
    - the default for **`JInternalFrame`**
  - **`EXIT_ON_CLOSE`**
    - defined in the **`JFrame`** class
    - recommended for applications only



Superclasses of Swing Component



Common Methods of JCompoments



All accessor (get) and mutator (set) methods for private properties are provided in the class but omitted here

Overview of Swing Components

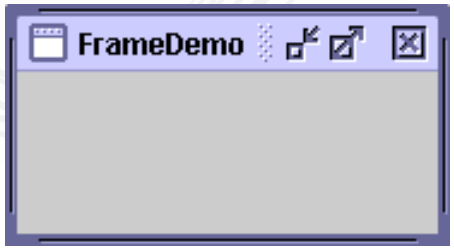
- Class **`Component`**
  - Contains **`paint`** method for drawing **`Component`** onscreen
- Class **`Container`**
  - Collection of related components
  - Contains **`add()`** method for adding components
- Class **`JComponent`**
  - *Pluggable look and feel* for customizing look and feel
  - Shortcut keys (*mnemonics*)
  - Common event-handling capabilities
  - All Swing components whose names begin with "J" descend from the **`JComponent`** class, with the exception of top-level containers,
    - For example, **`JPanel`**, **`JScrollPane`**, **`JButton`**, and **`JTable`**

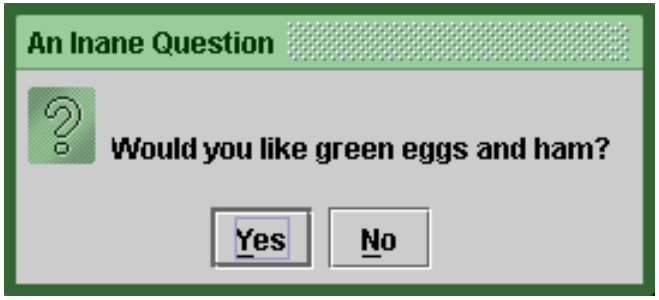
Some Basic GUI Components

Component	Description
`JLabel`	An area where uneditable text or icons can be displayed.
`JTextField`	An area in which the user inputs data from the keyboard. The area can also display information.
`JButton`	An area that triggers an event when clicked with the mouse.
`JCheckBox`	A GUI component that is either selected or not selected.
`JComboBox`	A drop-down list of items from which the user can make a selection by clicking an item in the list or possibly by typing into the box.
`JList`	An area containing a list of items from which the user can make a selection by clicking on any element in the list. Multiple elements can be selected.
`JPanel`	A container in which components can be placed and organized.

Top-Level Containers

Top-Level Containers – At least one of these components must be present





Frame Dialog Applet

They are heavyweight and created by ***native OS windowing system***, while other Swing GUI components, lightweight and created by ***Java***, are added into them

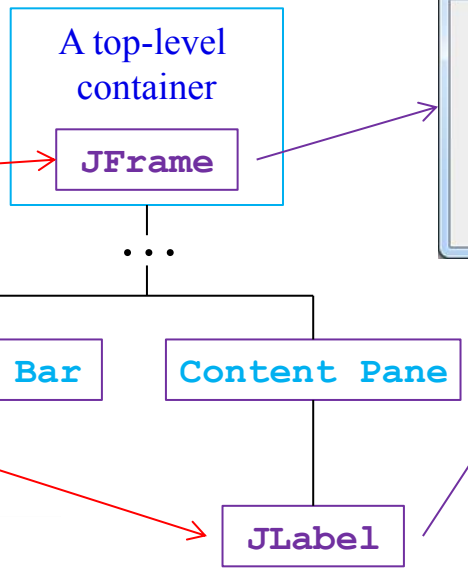
Top-level Container Classes

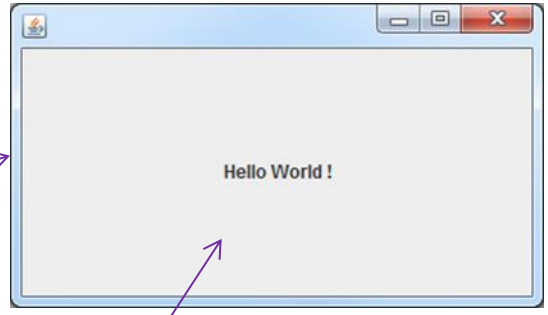
- To appear onscreen, every GUI component must be part of a *containment hierarchy*.
  - A containment hierarchy is a tree of components that has a top-level container as its root.
- Each GUI component can be contained only once.
  - If a component is already in a container and you try to add it to another container, the component will be removed from the first container and then added to the second.
- Each top-level container has a content pane
  - that contains the visible components in that top-level container's GUI.
- You can optionally add a menu bar to a top-level container.
  - The menu bar is by convention positioned within the top-level container, but outside the content pane.

Containment Hierarchy

- Standalone Application has at least one containment hierarchy with the **`JFrame`** as its root

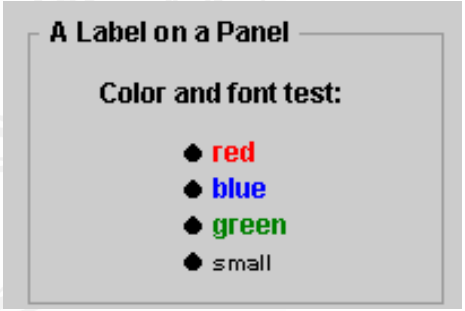
```
import javax.swing.*;

public class HelloWorld {
    public static void main(String[] args){

        //Create and set up the frame (window).
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(350, 200);

        //Create and add the label
        JLabel helloLabel = new JLabel("Hello World !"
        frame.add(helloLabel);

        //Display the frame
        frame.setVisible(true);
    }
}
```



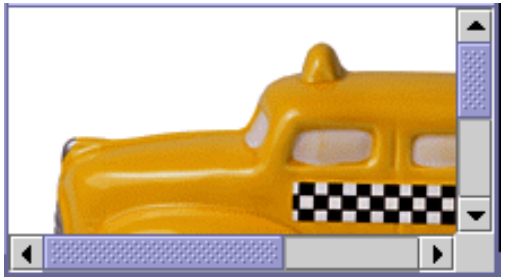


General-Purpose Containers

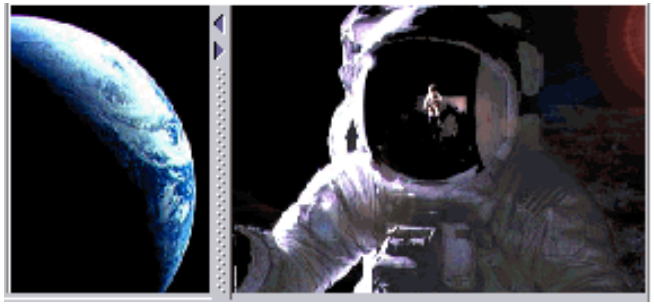
General-Purpose Containers – used to contain other components



Panel



Scroll Pane



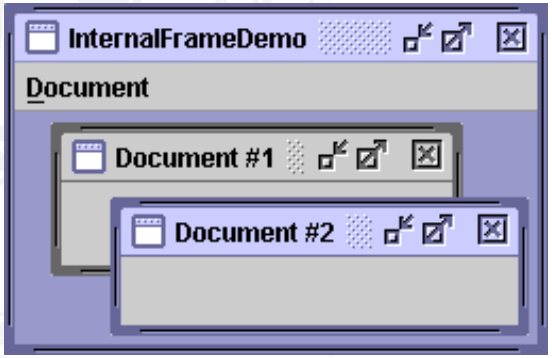
Split Pane

Tabbed Pane

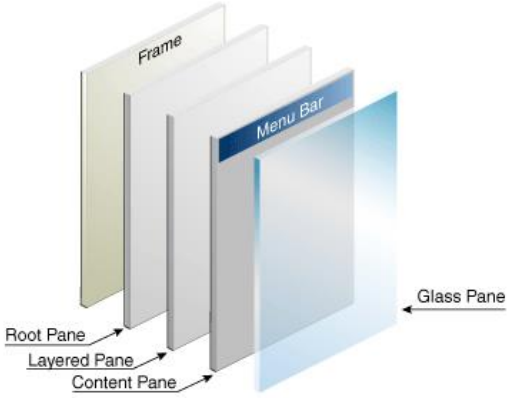
Tool Bar

Special-Purpose Containers

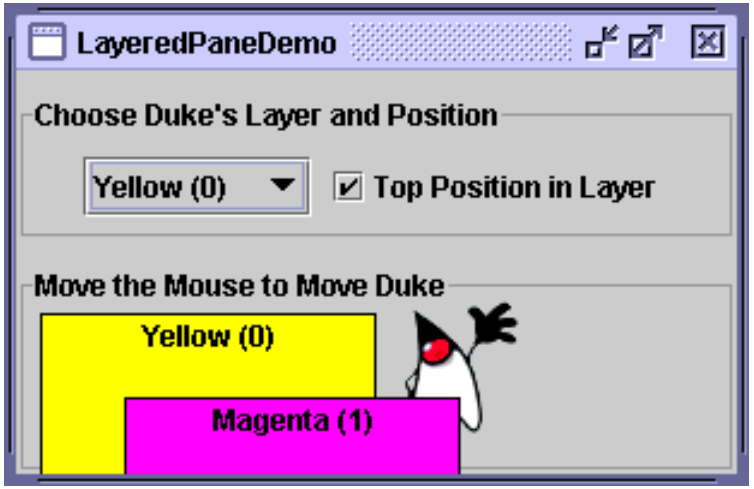
Special-Purpose Containers – play specific roles in the GUI



Internal Frame



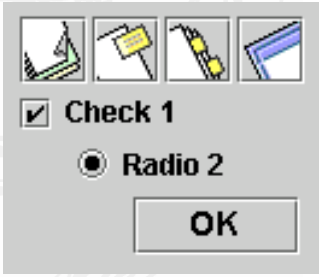
Root Pane



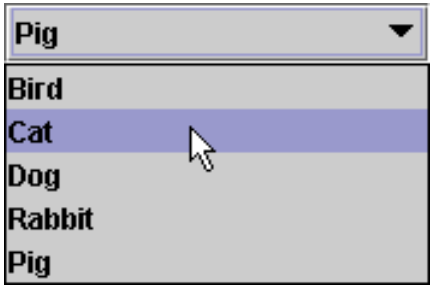
Layered Pane

Basic Controls

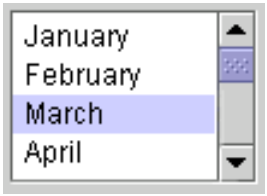
Basic Controls – To get input from the user; also show simple state



Buttons

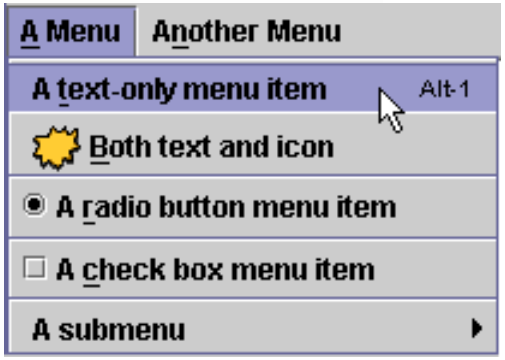


Combo Box

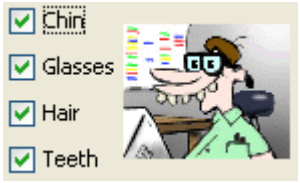


List

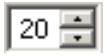
Slider



Menu



Check Box

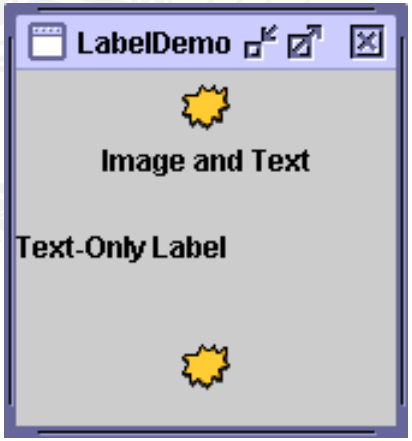


Spinner

Text Field
Formatted Text Field

Uneditable Information Displays

Uneditable Information Displays – To give the user information



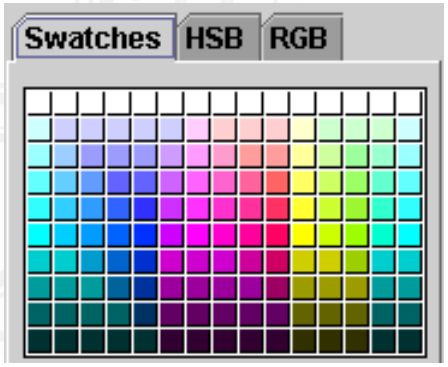
Label

Progress Bar

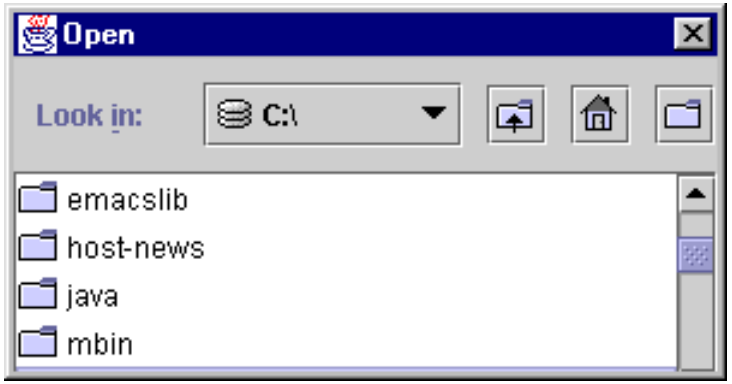
Tool Tip

Interactive Displays

Interactive Displays of Highly Formatted Information – modifiable by the user



Color Chooser

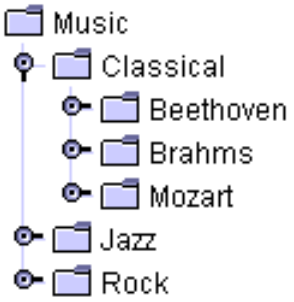


File Chooser

First Name	Last Name	Favorite Food
Jeff	Dinkins	
Ewan	Dinkins	
Amy	Fowler	
Hania	Gajewska	
David	Geary	

Table

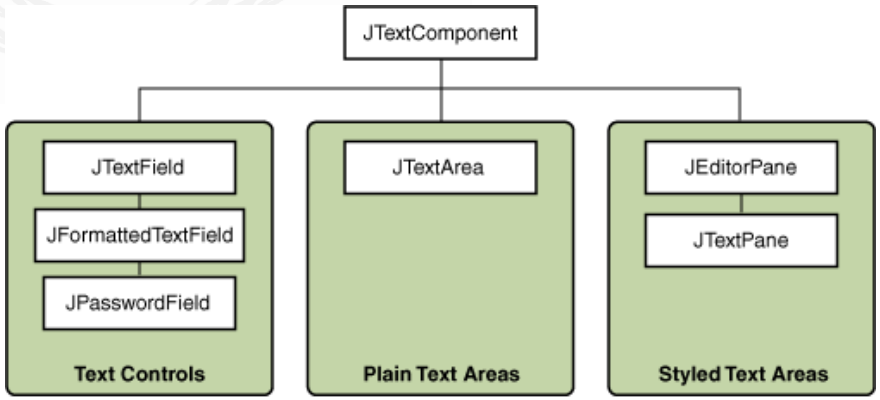
Text



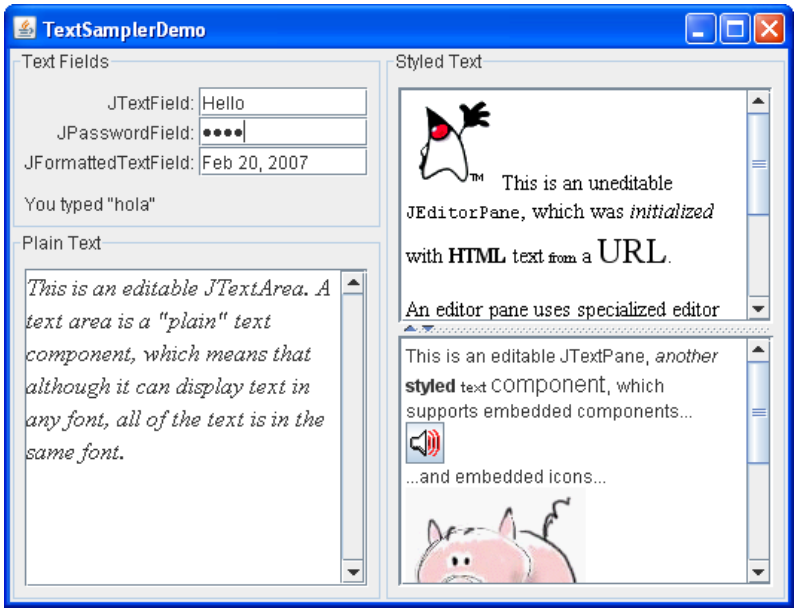
Tree

Text Components

- Display text and optionally allow users to edit the text



`JTextComponent` Hierarchy



Examples of text components

Helper Classes

- Helper classes are not subclasses of **`Component`** class
- Used to describe the properties of GUI components

- In **`java.awt`** package
  - **`Color, Font, Dimension`**
  - **`Border, Cursor`**
  - **`Graphics, LayoutManager`**

Color Class

```
Color myColor = new Color(int r, int g, int b);
```

- `r`, `g`, and `b` specify a color by its red, green, and blue components.
- There are some `Color` class constants (standard colors):

```
Color.BLACK, Color.WHITE, Color.RED, Color.CYAN, …
```

Example:

```
Color myColor = new Color(128, 100, 100);
setColor(myColor);
setBackground(myColor);
```

Color Constants and Their RGB Values

Color constant	Color	RGB value
`public final static Color ORANGE`	orange	255, 200, 0
`public final static Color PINK`	pink	255, 175, 175
`public final static Color CYAN`	cyan	0, 255, 255
`public final static Color MAGENTA`	magenta	255, 0, 255
`public final static Color YELLOW`	yellow	255, 255, 0
`public final static Color BLACK`	black	0, 0, 0
`public final static Color WHITE`	white	255, 255, 255
`public final static Color GRAY`	gray	128, 128, 128
`public final static Color LIGHT_GRAY`	light gray	192, 192, 192
`public final static Color DARK_GRAY`	dark gray	64, 64, 64
`public final static Color RED`	red	255, 0, 0
`public final static Color GREEN`	green	0, 255, 0
`public final static Color BLUE`	blue	0, 0, 255

Difficult to read? Read the API Documentation

Font Class

```
Font myFont = new Font(String name, int style, int size);
```

- Font name
  - **Monospaced**, **SansSerif**, **Serif**, etc.
- Font style
  - **Font.PLAIN**, **Font.ITALIC** and **Font.BOLD**
- Font size
  - Measured in points (1/72 of inch)

Example:

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

setFont(new Font("SansSerif", Font.BOLD, 17));
```

Example: Setting Up GUI

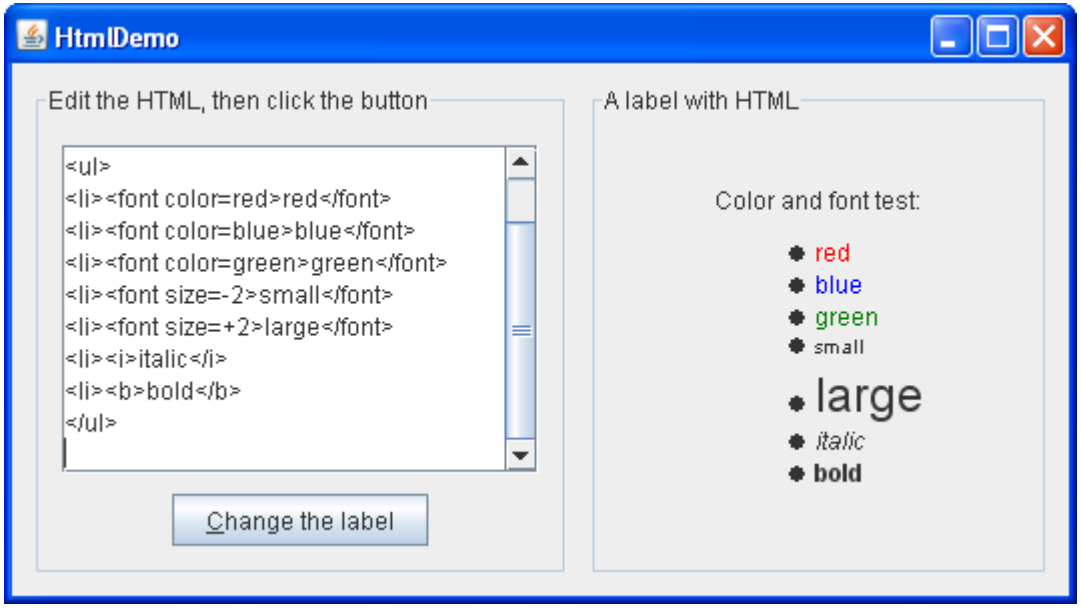
```
public class HelloBeautifulWorld {
    public static void main(String[] args){

        /* Create and set up the frame (window) */
        JFrame frame = new JFrame("Creating A Beautiful Frame"); //with a window title
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(350, 200);                                  //Set the frame size
        frame.setLocationRelativeTo(null);                        //Center the frame
        frame.getContentPane().setBackground(Color.MAGENTA);      //Set background color
                                                                  //  of the ContentPane
        /* Create, setup and add the label */
        JLabel helloLabel = new JLabel("Hello Beautiful World !", JLabel.CENTER);
        helloLabel.setForeground(new Color(255,250,0));
        helloLabel.setFont(new Font("Serif", Font.BOLD+Font.ITALIC, 28));
        helloLabel.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
                                                                  //Set the cursor
        frame.add(helloLabel);

        /* Display the frame */
        frame.setVisible(true);
    }
}
```

HTML in Swing Components

- Many Swing components display a text string as part of their GUI.
  - By default, a component's text is displayed in a single font and color, all on one line. You can determine the font and color of a component's text by invoking the component's setFont and setForeground methods

```
label = new JLabel("A label");
label.setFont(new Font("Serif", Font.PLAIN, 14));
label.setForeground(new Color(0xffffdd));
```

- HTML can be used to mix fonts or colors within text or multiple lines
  - buttons, menu items, labels, tool tips, and tabbed panes



Exceptions in GUI Applications

- An error message appears on the console with unhandled exceptions, but the GUI application may or may not continue running
- A dialog box often is used to display messages for handled exceptions
- An event may be ignored in GUI applications, but an exception can not be ignored

Unhandled exception: **ErrorDemo.java**

Handled exception: **ErrorDemo2.java**

Swing Demo

- JDK Demos includes useful demonstration programs (**`SwingSet2`** and **`SwingSet3`**) that show all the Swing components and enables you to explore with them interactively
- They display the source code for each demo
- **`SwingSet2`** can be found in **`demo\jfc\SwingSet2`** under your JDK Demo directory and **`SwingSet3`** can be run using JavaWebstart at

http://download.java.net/javadesktop/swingset3/SwingSet3.jnlp