

CS20004: Object Oriented Programming using Java

Lec- 25

In this Discussion . . .

- Why Study GUIs ?
 - Java GUI Libraries
 - GUI Terminologies
- What is Swing ?
- Using Swing
- Component & Container Classes
- Container Types
- Swing controls
- Anatomy of the program: Hello, World !

Why Study GUIs ?

- Learn about event-driven programming techniques
- Practice learning and using a large, complex API
- A chance to see how it is designed and learn from it:
 - design patterns: model-view separation, callbacks, listeners, inheritance vs. delegation
 - refactoring vs. reimplementing an ailing API
- Because GUIs are neat!

Why are we still teaching GUI and desktop programming in 2024?

Because there are still many desktop applications
around (even if they are less and less used)

Because the same concepts and the same knowledge
can be applied in the development of apps for
smartphones and tablet

Graphical user interfaces (GUI) and OOPS


Object oriented programming is very
well suited for **GUI programming**

GUI components or controls are natural objects: **windows** ,
buttons , **labels** , **text fields** , etc., GUI programming is
naturally **asynchronous** and **event oriented**

In an application with a GUI, the **main** method is
responsible to **initialize** and **assemble** the **GUI** and the
application logic , and then to make the GUI visible

Java GUI Libraries



- Swing 
 - Abstract Window Toolkit (AWT)
 - Part of Java SE
- JavaFX
- Standard Widget Toolkit (SWT)
- All available for Windows, Linux, and MacOS

Java GUI Libraries



- **Swing:** the main Java GUI library
 - *Benefits:* Features; cross-platform compatibility; OO design
 - Paints GUI controls itself pixel-by-pixel
 - Does not delegate to OS's window system
- **Abstract Window Toolkit (AWT):** Sun's initial GUI library
 - Maps Java code to each operating system's real GUI system
 - *Problems:* Limited to lowest common denominator (limited set of UI widgets); clunky to use.

Advice: Use Swing. You occasionally have to use AWT (Swing is built on top of AWT). *Beware:* it's easy to get them mixed up.

Swing Library



- All Swing components are under **javax.swing.***
- **As Swing uses the AWT event model**, we need to add the following in order to use events:
 - **java.awt.***
 - **Java.awt.event.***

What is Swing ?

- Swing is a set of program components for Java programmers that provide the ability to create graphical user interface (GUI) components.

Replaces the Abstract Window Toolkit or AWT as of Java 1.1

Features:

- **Lightweight.** Not built on native window-system windows.
- **Much bigger set of built-in controls.** Trees, image buttons, tabbed panes, sliders, toolbars, color choosers, tables, text areas to display HTML or RTF, etc.
- **Much more customizable;** Can change border, text alignment, or add image to almost any control.
- **Can change look and feel at runtime, or design own look and feel.**
- **Model-View-Controller architecture** lets you change the internal data representation (lists, trees, tables).

Using Swing : How do we use Swing ?

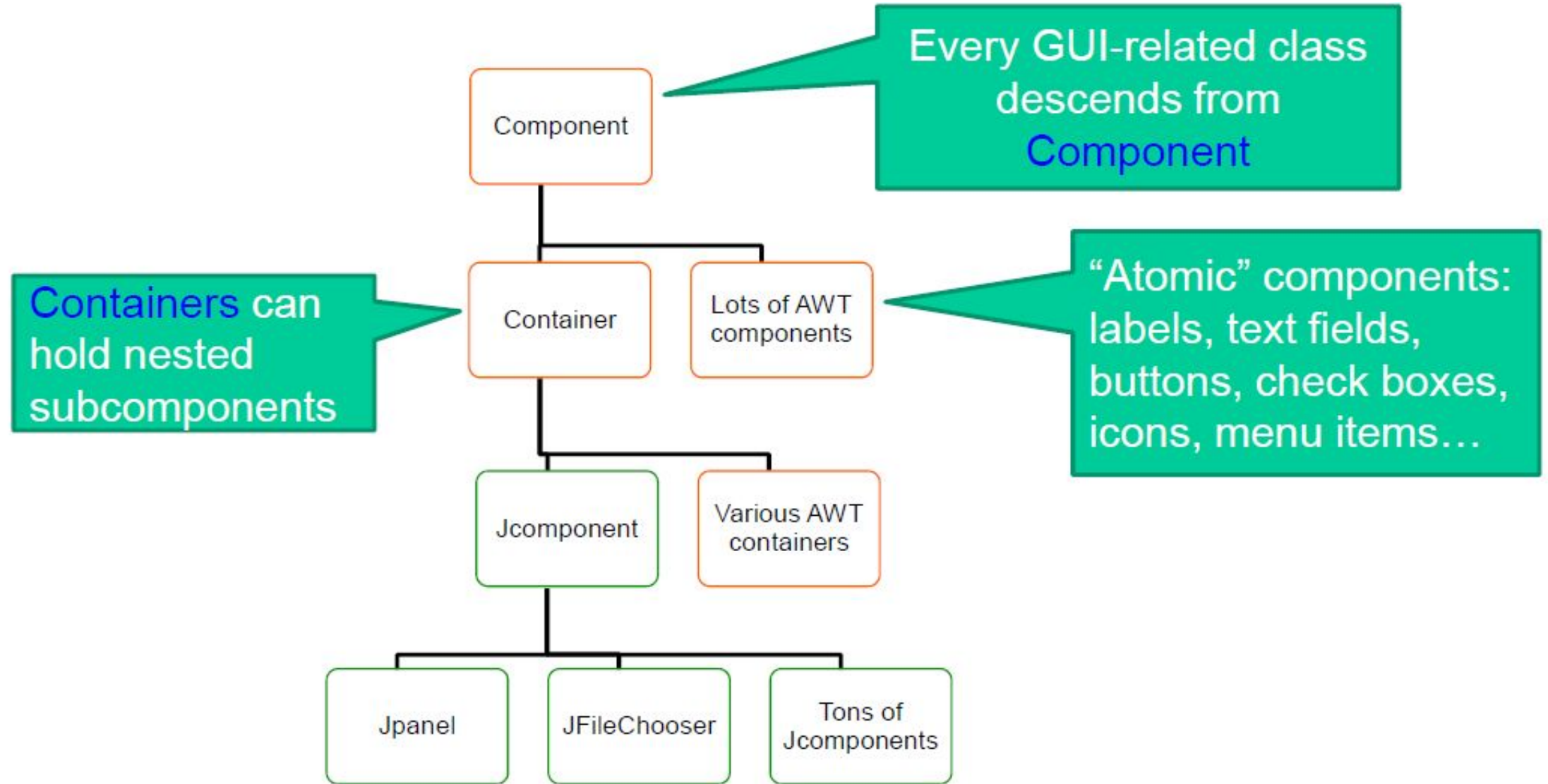
- Swing provides many standard GUI components such as buttons, lists, menus, and text areas, which you combine to create your program's GUI.

Swing components start with the letter **J**; JFrame, JButton, etc.

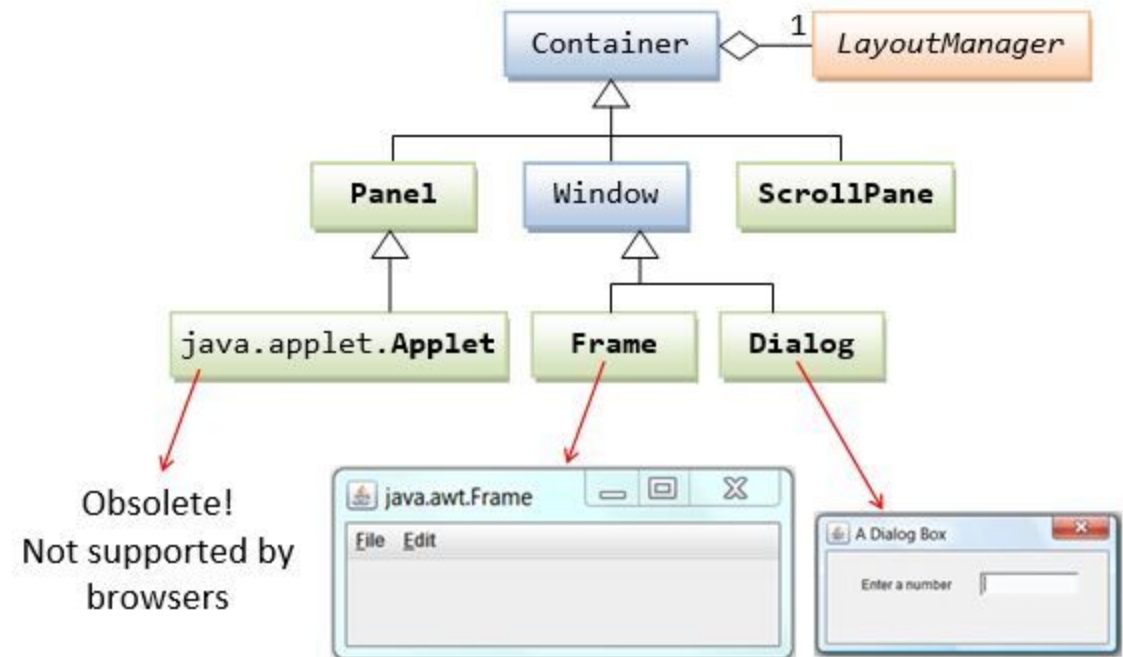
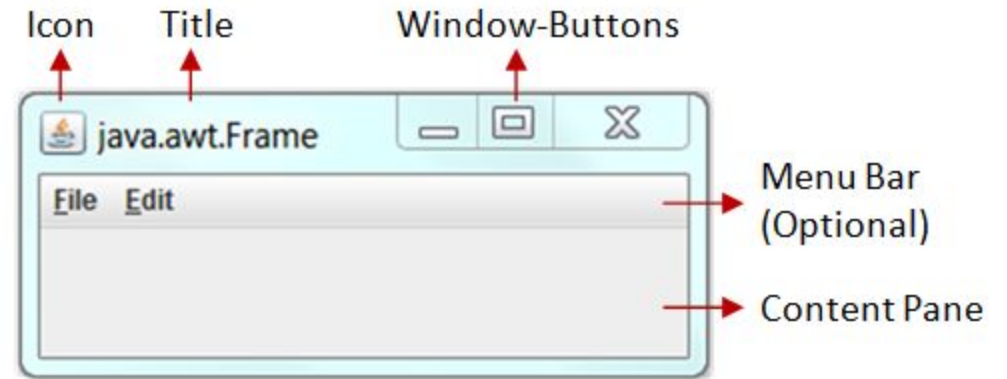
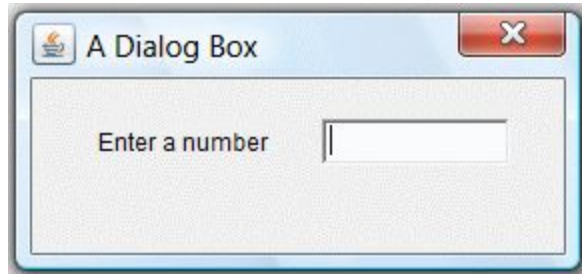
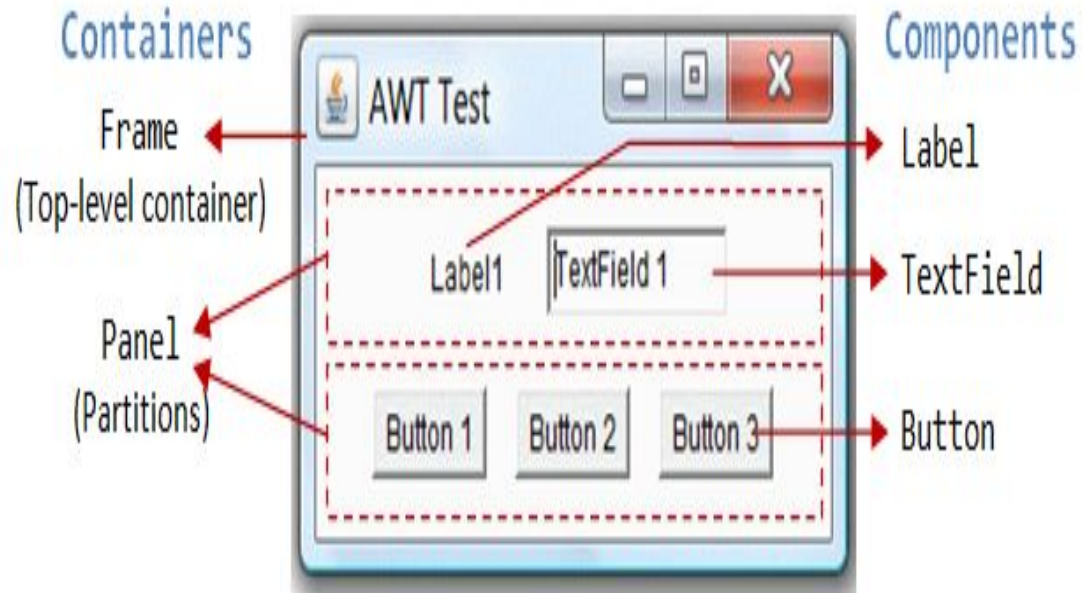
Ways:

- Use **containers** and **layout managers** to create windows.
- Use **components** and **event handlers** for user interaction
- `import javax.swing.*;`

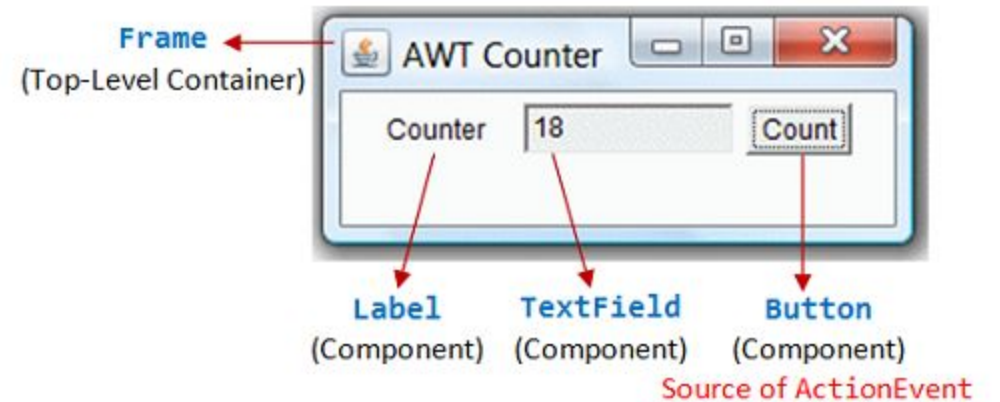
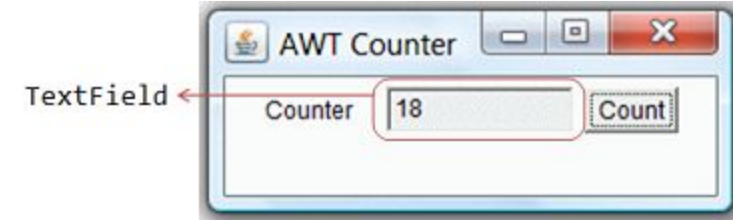
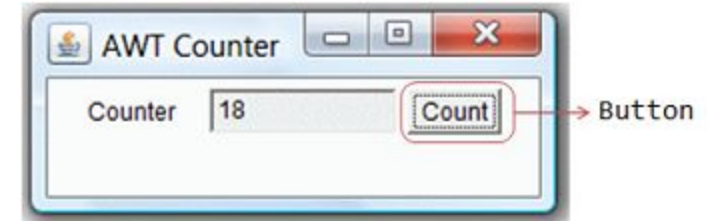
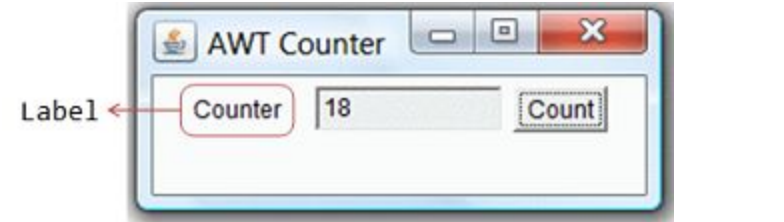
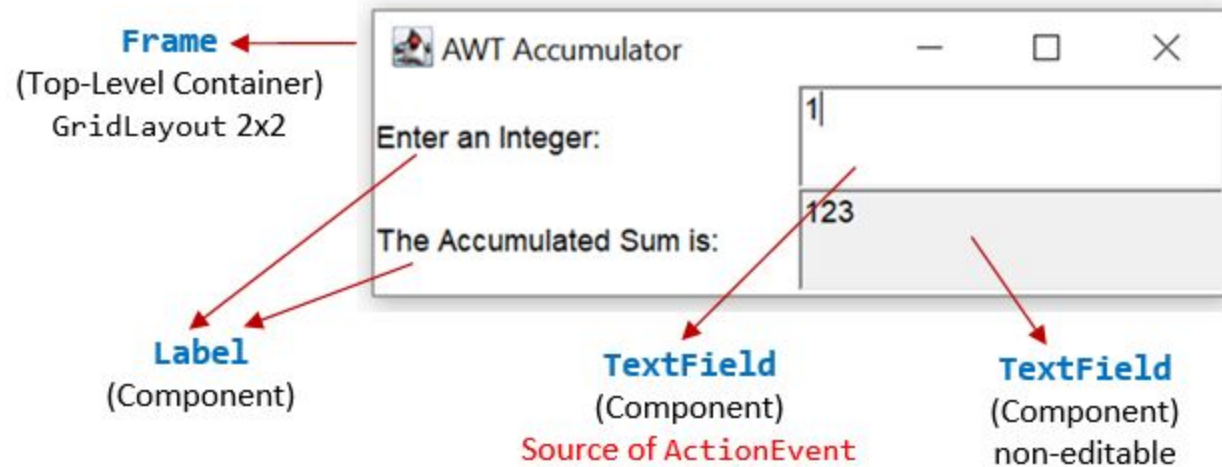
Component & Container Classes



AWT Containers and Components



AWT Containers and Components

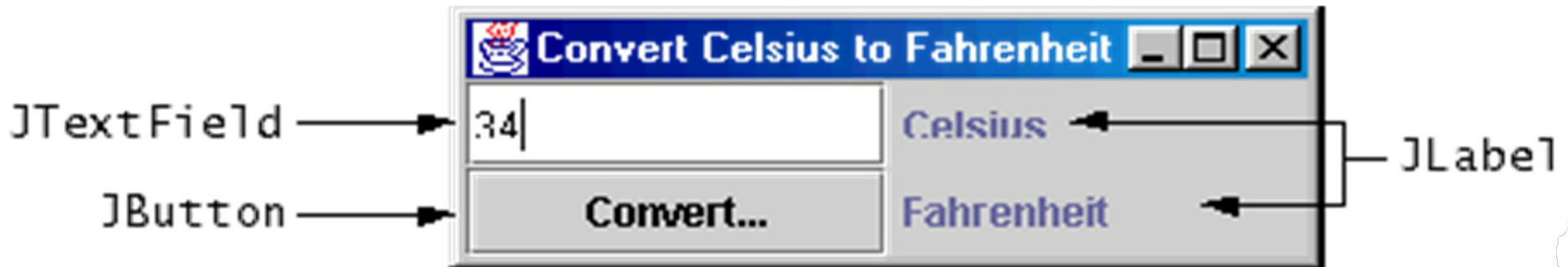


Swing Controls

- In Java Swing, there are various components or controls that can be used to create graphical user interfaces (GUIs). Some of the commonly used Swing controls include:
 - **JButton:** A button that can be clicked to trigger an action.
 - **JLabel:** A non-editable text component used for displaying text or images.
 - **TextField:** A single-line text field that allows the user to input text.
 - **TextArea:** A multi-line text area that allows the user to input or display multiple lines of text.
 - **JCheckBox:** A checkbox component that allows the user to select one or more options.
 - **JRadioButton:** A radio button component that allows the user to select one option from a group of options.
 - **JComboBox:** A drop-down list component that allows the user to select one item from a list of options.
 - **JList, JSlider, JSpinner, JProgressBar, JScrollPane, JSplitPane, etc.**

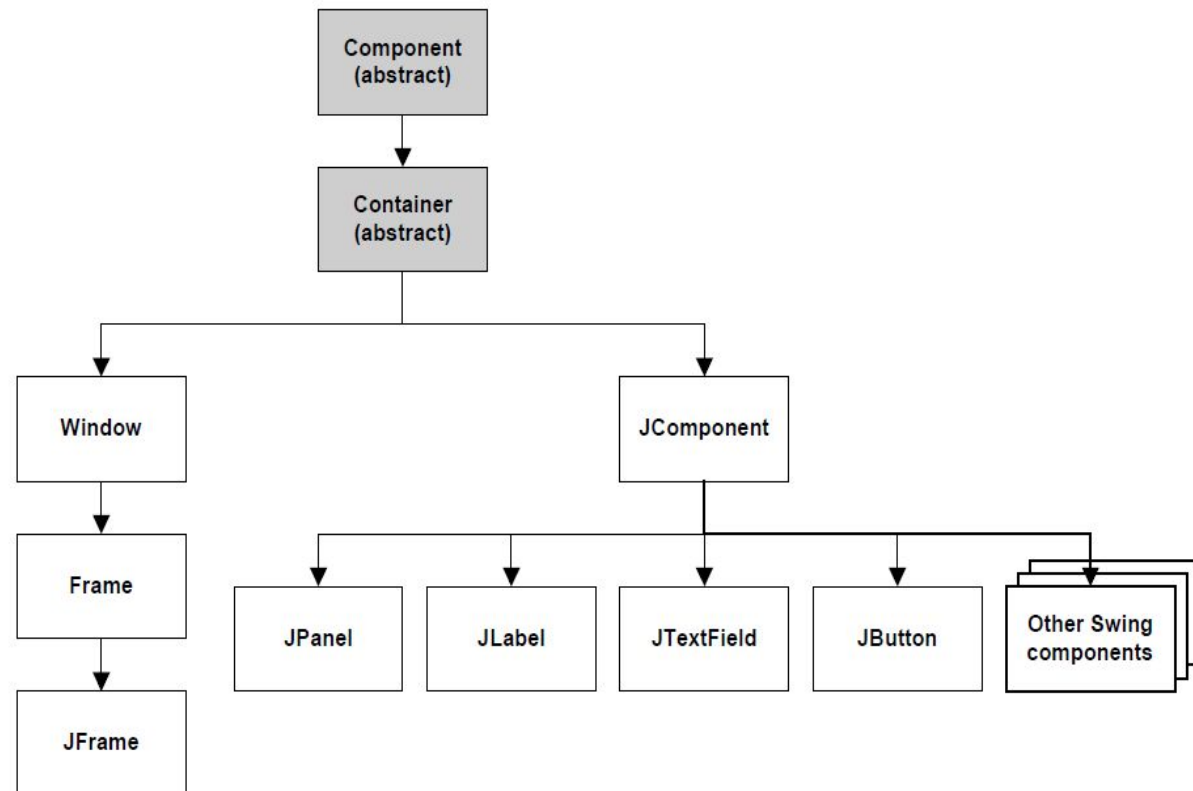
GUI Terminology

window	<ul style="list-style-type: none">• A first-class citizen of the graphical desktop• Also called a top-level container• Examples: frame, dialog box, applet
component	<ul style="list-style-type: none">• A GUI widget that resides in a window• Also called controls in many other languages• Examples: button, text box, label
container	<ul style="list-style-type: none">• A component that hosts (holds) components• Examples: panel, box





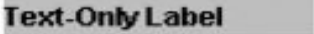



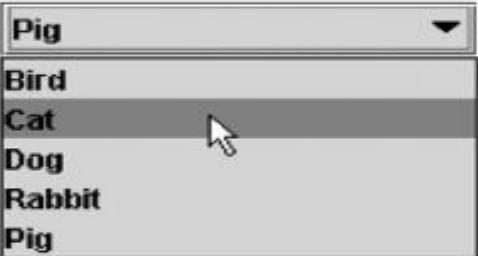

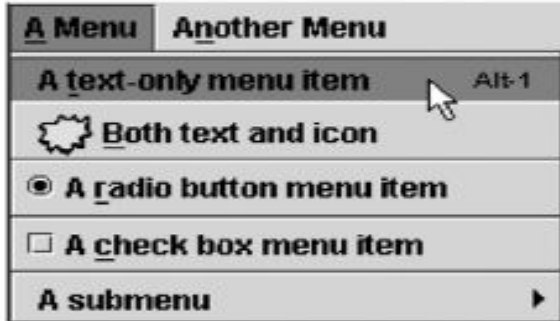
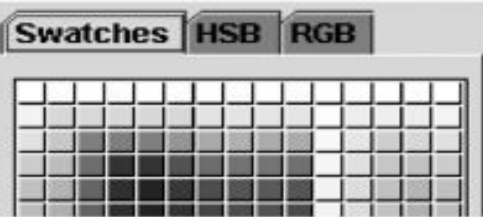







GUI Terminology

- The window that contains the GUI is called a frame.
- The frame contains a panel that contains the controls that are displayed by the application.
- The Abstract Window Toolkit (AWT) is an older technology for creating GUIs that look and act a little different on different platforms.
- Swing is a newer technology that creates GUIs that are consistent from platform to platform.



Swing Controls Depictions

JButton 	JCheckBox 	JRadioButton 	 														
JTextField 	JSlider 	JToolBar 															
JComboBox 	JList 	JMenuBar, JMenu, JMenuItem 															
JColorChooser 	JFileChooser 	JTable <table><tr><th>First Name</th><th>Last Name</th><th>Favorite F</th></tr><tr><td>Jeff</td><td>Dinkins</td><td rowspan="5"></td></tr><tr><td>Ewan</td><td>Dinkins</td></tr><tr><td>Amy</td><td>Fowler</td></tr><tr><td>Hania</td><td>Gajewska</td></tr><tr><td>David</td><td>GEARV</td></tr></table>	First Name	Last Name	Favorite F	Jeff	Dinkins		Ewan	Dinkins	Amy	Fowler	Hania	Gajewska	David	GEARV	JTree 
First Name	Last Name	Favorite F															
Jeff	Dinkins																
Ewan	Dinkins																
Amy	Fowler																
Hania	Gajewska																
David	GEARV																

Swing/AWT Inheritance Hierarchy

```
Component (AWT)
  Window
    Frame
      JFrame (Swing)
      JDialog
  Container
    JComponent (Swing)
      JButton      JColorChooser  JFileChooser
      JComboBox    JLabel         JList
      JMenuBar     JOptionPane    JPanel
      JPopupMenu   JProgressbar    JScrollbar
      JScrollPane  JSlider         JSpinner
      JSplitPane   JTabbedPane     JTable
      JToolBar     JTree           JTextArea
      JTextField   ...
```

Component:

- Swing components are derived from the **JComponent** class.
- **JComponent** provides the functionality that is common to all components.
- For example, JComponent supports the pluggable look and feel.
- **JComponent** inherits the AWT classes Container and Component.
- Thus, a Swing component is built on and compatible with an AWT component.
- All of Swing components are represented by classes defined within the package javax.swing.
- The image on the left shows the class names for Swing components (including those used as containers).

Component Field (actual Properties)

- Each has a get (or is) accessor and a set modifier.
- Examples: getColor, setFont, isVisible,

name	description
background	background color behind component
border	border line around component
enabled	whether it can be interacted with
focusable	whether key text can be typed on it
font	font used for text in component
foreground	foreground color of component
height, width	component's current size in pixels
visible	whether component can be seen
tooltip text	text shown when hovering mouse
size, minimum / maximum / preferred size	various sizes, size limits, or desired sizes that the component may take

Swing/AWT Inheritance Hierarchy

```
Component (AWT)
  Window
    Frame
      JFrame (Swing)
      JDialog
  Container
    JComponent (Swing)
      JButton      JColorChooser  JFileChooser
      JComboBox    JLabel         JList
      JMenuBar     JOptionPane    JPanel
      JPopupMenu   JProgressBar   JScrollbar
      JScrollPane  JSlider         JSpinner
      JSplitPane   JTabbedPane    JTable
      JToolBar     JTree          JTextArea
      JTextField   ...
```

Container: Top-level

- Swing defines two types of containers. The first are **top-level containers**: **JFrame**, **JApplet**, **JWindow**, and **JDialog**.
- These containers do not inherit JComponent.
- They do, however, inherit the AWT classes Component and Container.
- Unlike Swing's other components, which are lightweight, the top-level containers are heavyweight.
- This makes the top-level containers a special case in the Swing component library.

Swing/AWT Inheritance Hierarchy

Component (AWT)

Window

Frame

JFrame (Swing)

JDialog

Container

JComponent (Swing)

JButton

JColorChooser

JFileChooser

JComboBox

JLabel

JList

JMenuBar

JOptionPane

JPanel

JPopupMenu

JProgressBar

JScrollbar

JScrollPane

JSlider

JSpinner

JSplitPane

JTabbedPane

JTable

JToolBar

JTree

JTextArea

JTextField

...

Container: Top-level

- As the name implies, a top-level container must be at the top of a **containment hierarchy**.
- A top-level container is not contained within any other container.
- Furthermore, every containment hierarchy must begin with a top-level container.
- The one most commonly used for applications is JFrame. The one used for applets is JApplet
- **Technically, all JComponent's are containers**

Swing/AWT Inheritance Hierarchy

Component (AWT)

Window

Frame

JFrame (Swing)

JDialog

Container

JComponent (Swing)

JButton

JColorChooser

JFileChooser

JComboBox

JLabel

JList

JMenuBar

JOptionPane

JPanel

JPopupMenu

JProgressBar

JScrollbar

JScrollPane

JSlider

JSpinner

JSplitPane

JTabbedPane

JTable

JToolBar

JTree

JTextArea

JTextField

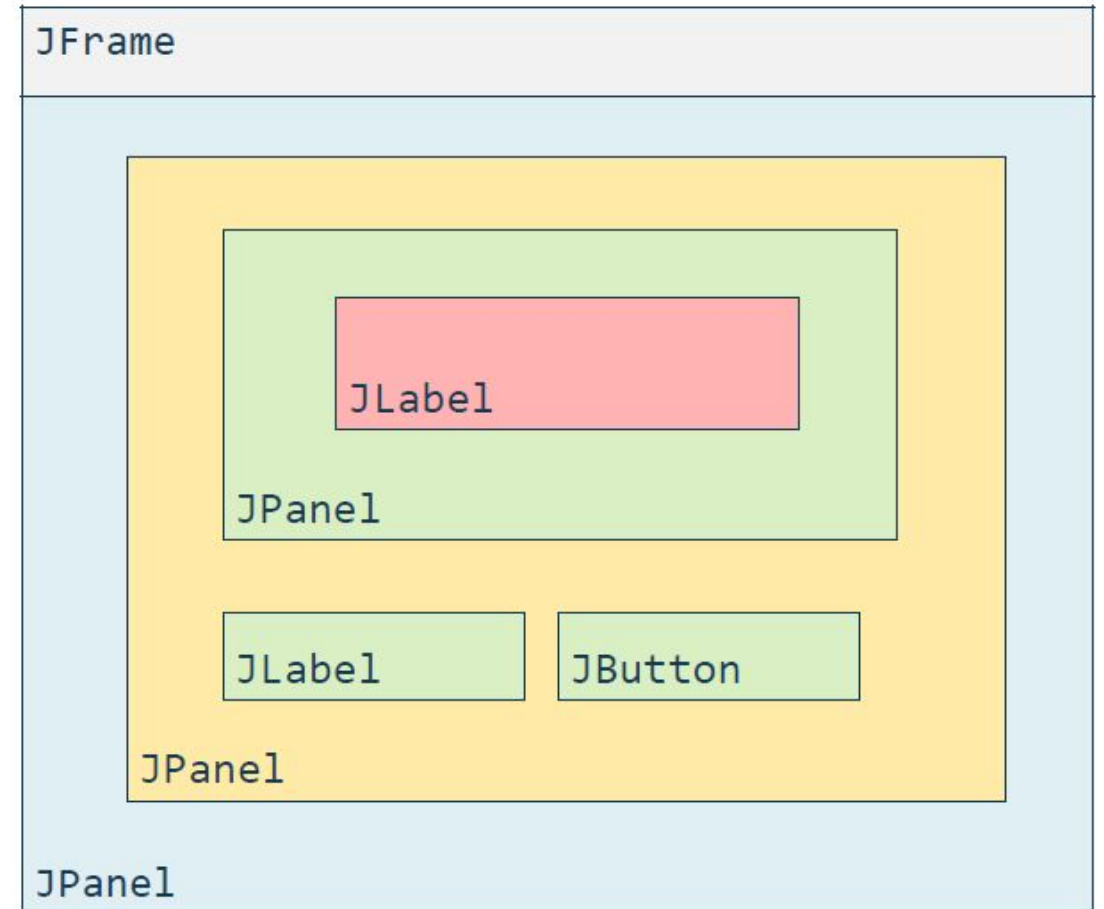
...

Container: Lightweight or intermediate or Mid-level

- The second type of containers supported by Swing are **lightweight or intermediate or mid-level containers**.
- Lightweight containers do inherit JComponent.
- An example of a lightweight container is **JPanel**, which is a general-purpose container.
- Lightweight containers are often used to organize and manage groups of related components because a lightweight container can be contained within another container.
- Thus, you can use lightweight containers such as JPanel to create subgroups of related controls that are contained within an outer container

The rule of the game: Containment Hierarchy

- To make a component visible, its containment hierarchy must be included into a **JFrame** or another window object
- **JPanels** are containers to which usually we add components
- Each component can belong to just one container
- Other containers to which we add components are JToolBar, JMenu, and JPopupMenu



Containers

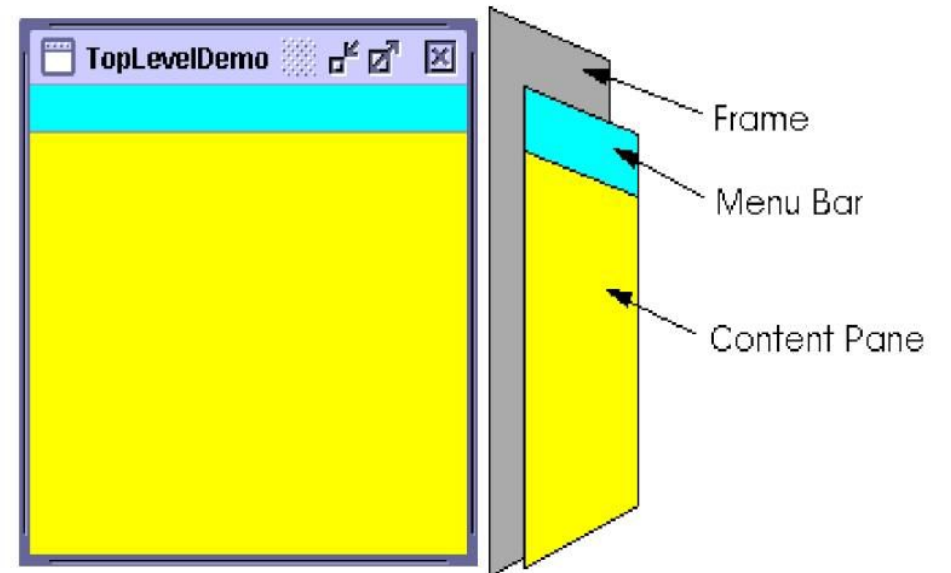
a main window

- Every Java program that has a GUI has at least one top-level container
- Swing provides containers such as windows and toolbars.
 - JFrame, JDialog
 - JPanel, JTabbedPane, JScrollPane, JInternalFrame

a secondary window, dependent on another window

- **Top-level Containers:**

- Every GUI component must be part of a containment hierarchy.
- Each GUI component can be contained only once.
- Each top-level container has a content pane.
- You can optionally add a menu bar to a top-level container.

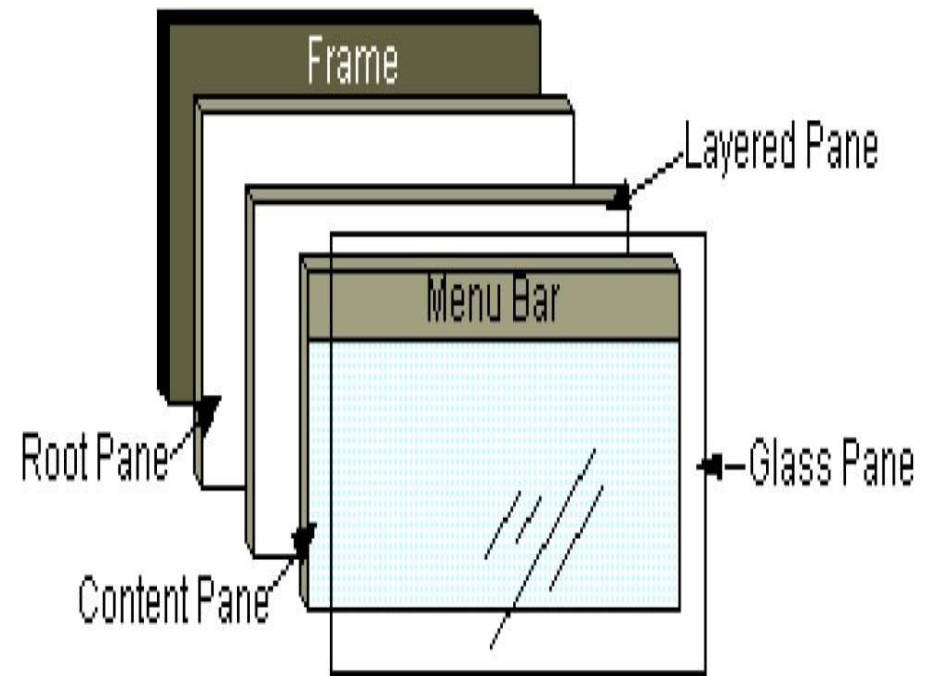


Containers

- Each top-level container defines a set of panes. At the top of the hierarchy is an instance of `JRootPane`.
- ***JRootPane*** is a lightweight container whose purpose is to manage the other panes. It also helps manage the optional menu bar.

Lightweight Containers:

- The *root pane* manages the content pane and the menu bar, along with a couple of other containers.
- The panes that comprise the root pane are called the glass pane, the content pane, and the layered pane.

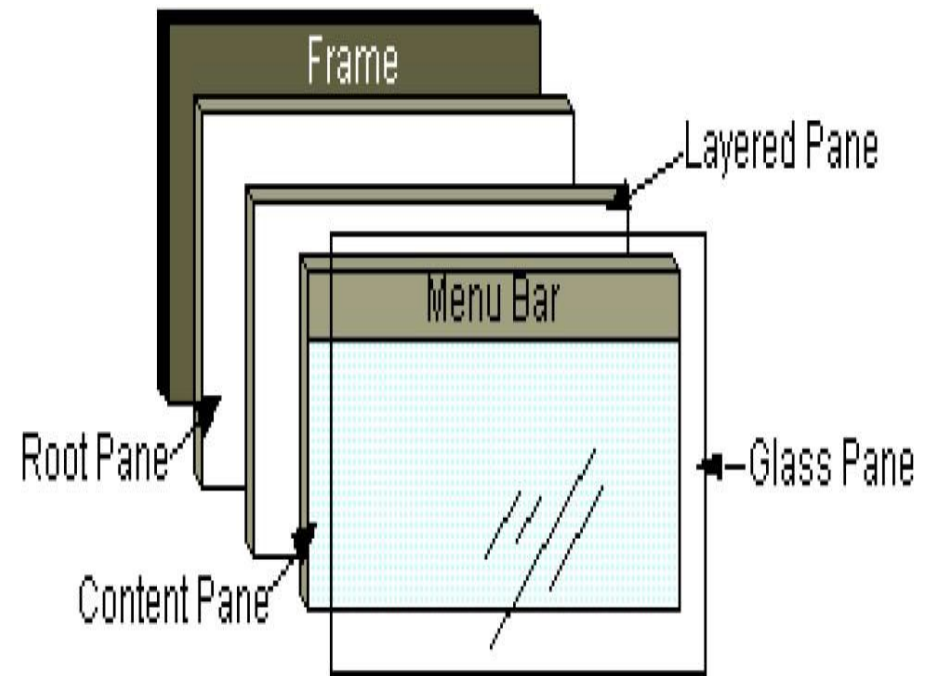


Containers

- Each top-level container defines a set of panes. At the top of the hierarchy is an instance of JRootPane.
- JRootPane is a lightweight container whose purpose is to manage the other panes. It also helps manage the optional menu bar.

Lightweight Containers:

- The *layered pane* directly contains the menu bar and content pane
- The *glass pane* is often used to intercept input events occurring over the top-level container, and can also be used to paint over multiple components.



JFrames

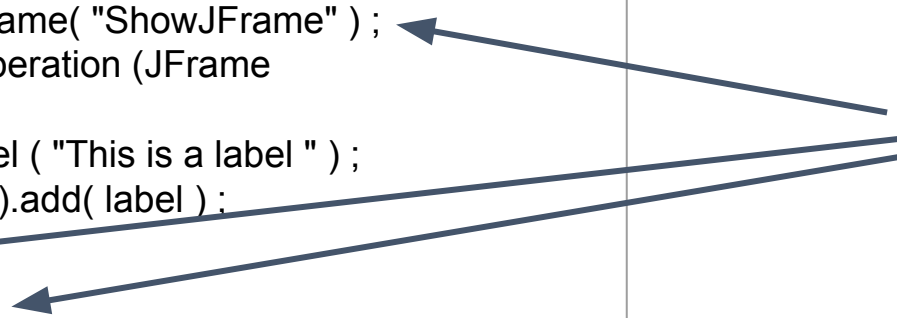
- A JFrame is a window that has decorations such as a border, a title and buttons for closing and iconifying the window.
- The decorations on a frame are platform dependent.
- Use **JFrame.getContentPane.add()** to add components.
- pack() sets the frame size based on the preferred sizes of sub-components

JFrames (Contd.)

```
import javax . swing . * ;

class ShowFrame
{
    public static void main( String args [ ] )
    {
        JFrame frame = new JFrame( "ShowJFrame" ) ;
        frame.setDefaultCloseOperation (JFrame
.EXIT_ON_CLOSE) ;
        JLabel label = new JLabel ( "This is a label " ) ;
        frame.getContentPane ( ).add( label ) ;
        frame.pack ( ) ;
        frame.setVisible ( true ) ;
    }
}
```

Setting up a frame




JFrames (Contd.)

```
import javax . swing . * ;

class ShowFrame
{
    public static void main( String args [ ] )
    {
        JFrame frame = new JFrame( "ShowJFrame" ) ;
        frame.setDefaultCloseOperation ( JFrame
.EXIT_ON_CLOSE ) ;
        JLabel label = new JLabel ( "This is a label " ) ;
        frame.getContentPane( ).add( label ) ;
        frame.pack ( ) ;
        frame.setVisible ( true ) ;
    }
}
```

Adding a component to a
frame



JFrames (Contd.)

```
import javax . swing . * ;

class ShowFrame
{
    public static void main( String args [ ] )
    {
        JFrame frame = new JFrame( "ShowJFrame" ) ;
        frame.setDefaultCloseOperation ( JFrame .EXIT_ON_CLOSE) ;
        JLabel label = new JLabel ( "This is a label " ) ;
        frame.getContentPane( ).add( label ) ;
        frame.pack ( ) ;
        frame.setVisible ( true ) ;
    }
}
```



Closing a frame

JFrames (Contd.)

```
import javax . swing . * ;

class ShowFrame
{
    public static void main( String args [ ] )
    {
        JFrame frame = new JFrame( "ShowJFrame" ) ;
        frame.setDefaultCloseOperation (JFrame
.EXIT_ON_CLOSE) ;
        JLabel label = new JLabel ( "This is a label " ) ;
        frame.getContentPane ( ).add( label ) ;
        frame.pack ( ) ;
        frame.setVisible ( true ) ;
    }
}
```

```
C:\Users\KIIT\Desktop\6th Sem Jan-July-2024\OOPS-Java\Labs>javac ShowFrame.java
```

```
C:\Users\KIIT\Desktop\6th Sem Jan-July-2024\OOPS-Java\Labs>java ShowFrame
```



Lightweight or Intermediate or Mid-level containers

- Also known as panels or panes.
- Simplify the positioning of other components:
 - JPanel
- Play a visible and interactive role:
 - JScrollPane
 - JTabbedPane

JPanel: a general-purpose container

- Commonly used as a place for graphics, or to hold a collection of button, labels, etc.
- Needs to be added to a window or other container

```
frame.add(new JPanel(...))
```

- JPanels can be nested to any depth
- Many methods/fields in common with JFrame (since both inherit from Component) [Advice: can't find a method/field? Check the superclass(es)]
- Some new methods. Particularly useful:

setPreferredSize(Dimension d)

JPanel

```
import javax.swing.*;

class ShowPanel
{
    public static void main( String args [ ] )
    {
        JFrame frame = new JFrame( "ShowJPanel" );
        JPanel panel = new JPanel( );
        JLabel label = new JLabel ( "This is a label with
some text in i t . " );
        JButton button = new JButton ( "Click Me" );
        panel.add( label );
        panel.add(button );
        frame.getContentPane( ).add(panel );
        frame.pack ( );
        frame.setVisible(true);
    }
}
```

```
C:\Users\KIIT\Desktop\6th Sem Jan-July-2024\OOPS-Java\Labs>javac ShowPanel.java
```

```
C:\Users\KIIT\Desktop\6th Sem Jan-July-2024\OOPS-Java\Labs>java ShowPanel
```



JScrollPane

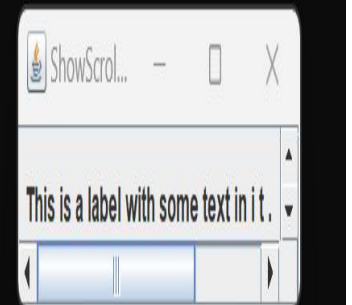
- A JScrollPane provides a scrollable view of a component

```
import javax.swing.*;

class ShowScrollPane
{
    public static void main(String args[ ])
    {
        JFrame frame = new JFrame( "ShowScrollPane"
    );
        JPanel panel = new JPanel ( );
        JLabel label = new JLabel( "This is a label with
some text in i t . " ) ;
        JButton button = new JButton ( "Click Me" ) ;
        panel . add(label) ;
        panel . add(button) ;
        JScrollPane sp = new JScrollPane(panel );
        frame.getContentPane( ).add(sp );
        frame.pack( );
        frame.setVisible(true);
    }
}
```

```
C:\Users\KIIT\Desktop\6th Sem Jan-July-2024\OOPS-Java\Labs>javac ShowScrollPane.java
```

```
C:\Users\KIIT\Desktop\6th Sem Jan-July-2024\OOPS-Java\Labs>java ShowScrollPane
```



Containers and Layout

- What if we add several components to a container? How are they positioned relative to each other?
 - **Answer:** each container has a layout manager.



Layout Management & Managers

- The process of determining the size and position of components.
- Kinds:
 - FlowLayout (left to right, top to bottom) – default for JPanel
 - BorderLayout (“center”, “north”, “south”, “east”, “west”) – default for JFrame (Ex- `pane . add(button , BorderLayout .CENTER) ;`)
 - GridLayout (regular 2-D grid)
 - others... (some are incredibly complex)
- Every content pane is initialized to use a BorderLayout.

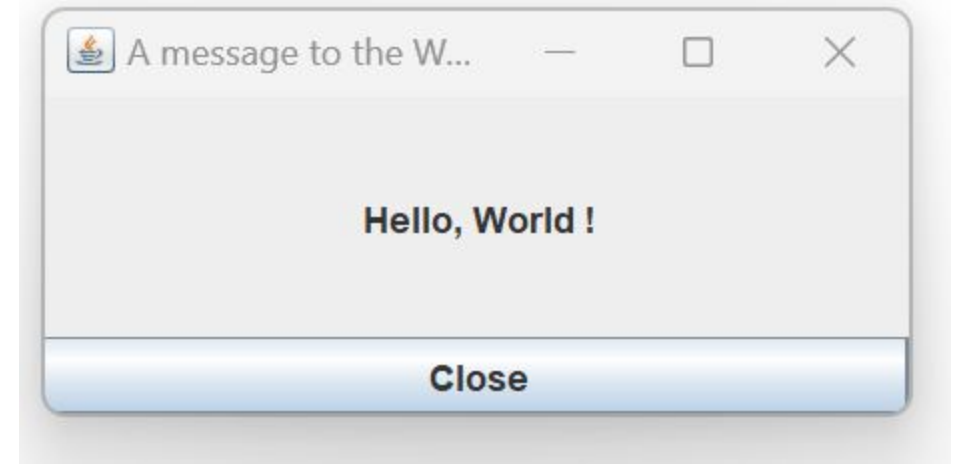
Abstractly !! : Swing Way of Working

- Place components in a container; add the container to a frame.
 - container: An object that stores components and governs their positions, sizes, and resizing behavior.

Hello, World !

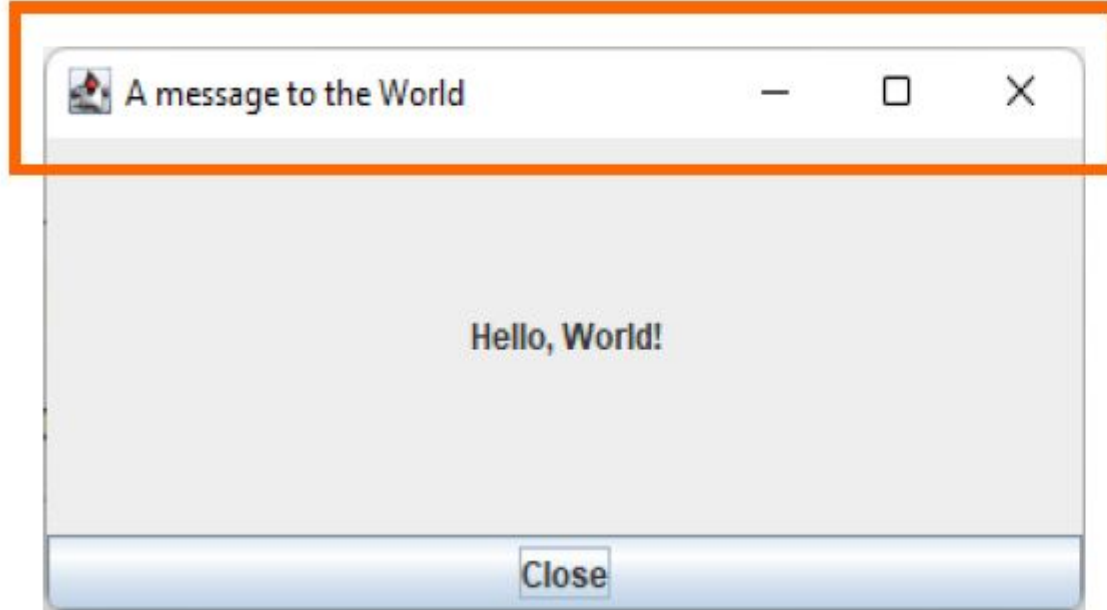
```
import javax.swing.*;
import java.awt.*;
public class HelloWorld
{
    public static void main(String [] args)
    {
        SwingUtilities.invokeLater(HelloWorld::helloWorld);
    }
    private static void helloWorld ()
    {
        JFrame frame = new JFrame("A message to the
World");
frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON
_CLOSE);
        JLabel label = new JLabel("Hello, World !");
```

```
label.setHorizontalAlignment(SwingConstants.CENTER);
        frame.getContentPane().add(label,
BorderLayout.CENTER);
        JButton closeButton = new JButton("Close");
        closeButton.addActionListener(x-> frame.dispose());
frame.getContentPane().add(closeButton , BorderLayout.SOUTH);
        frame.setSize(400,200);
        frame.setVisible(true);
    }
}
```



HelloWorld.java

```
JFrame frame = new JFrame("A message to the World");  
frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
```



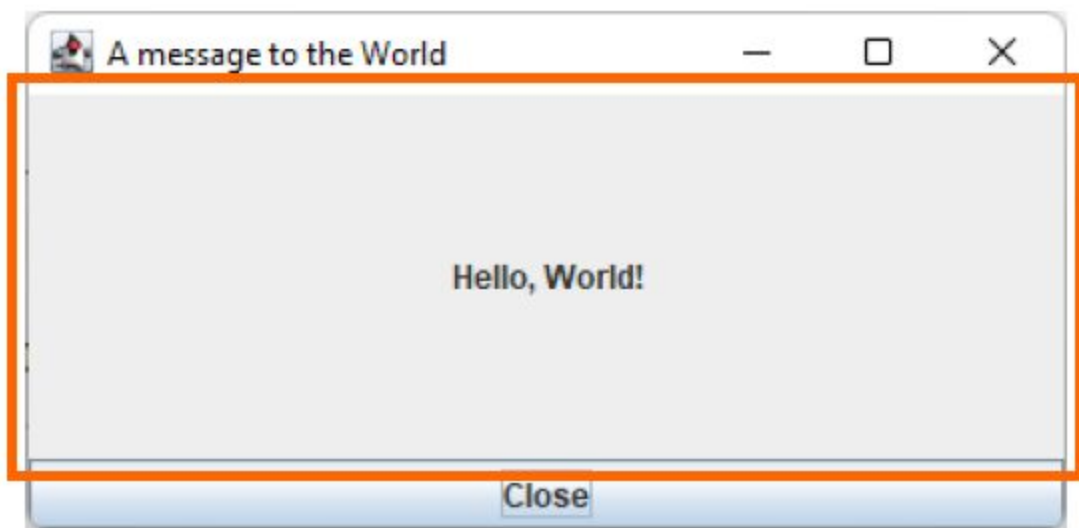
A **JFrame** represents a **window** with all the decorations: **icon**, **title**, and buttons to **minimize**, **maximize**, and **close**

The behavior of the **close** button can be customized, for example to **dispose** the **JFrame**

By **disposing** a **JFrame**, we **close** the **JFrame** if open, and we **release** all the resources associated to this **JFrame**

HelloWorld.java

```
JLabel label = new JLabel("Hello, World!");  
label.setHorizontalAlignment(SwingConstants.CENTER);  
frame.getContentPane().add(label, BorderLayout.CENTER);
```



The content pane of a JFrame uses the **BorderLayout** manager by **default**

A **JLabel** is a Swing component used to represents a piece of text with an icon

To make a Swing component visible, we must add it to a **container**, if there are no intermediate containers, we can add it to the **content pane** of the JFrame directly

A container uses a **layout manager** to layout the components it contains. When adding a component to a container we can specify a **constraint**

HelloWorld.java

```
JButton closeButton = new JButton("Close");  
closeButton.addActionListener(x -> frame.dispose());  
frame.getContentPane().add(closeButton, BorderLayout.SOUTH);
```



A **JButton** is a Swing component able to respond to **user actions**. For example, when the user clicks on the button, it triggers an action listener

HelloWorld.java

```
frame.setSize(400, 200);  
frame.setVisible(true);
```



A JFrame and its **content pane** are shown in the screen when we make the frame **visible**

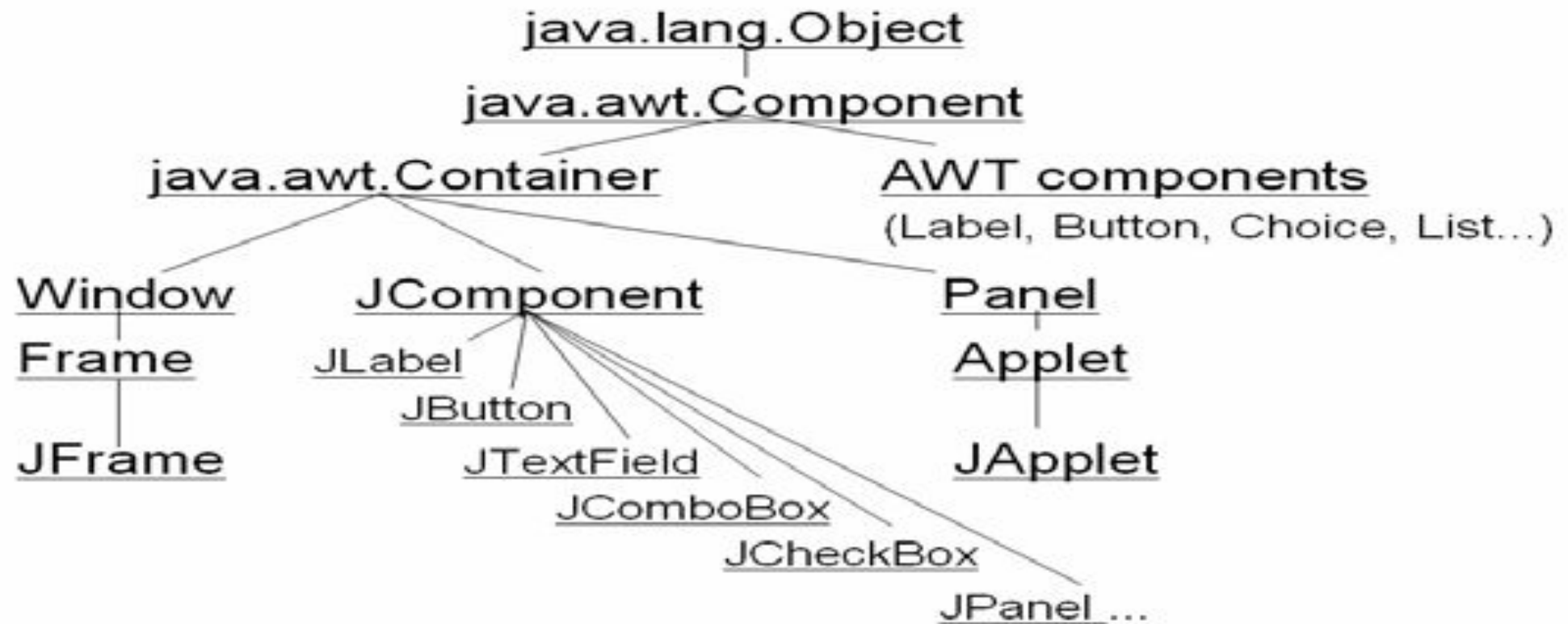
HelloWorld.java

```
public static void main(String[] args) {  
    SwingUtilities.invokeLater(HelloWorld::helloWorld);  
}
```

Almost all GUI code **MUST** run on the **Event Dispatch Thread** by using either **invokeLater** or **invokeAndWait**

static void <u>invokeAndWait</u> (<u>Runnable</u> doRun)	Causes <i>doRun.run()</i> to be executed synchronously on the AWT event dispatching thread.
static void <u>invokeLater</u> (<u>Runnable</u> doRun)	Causes <i>doRun.run()</i> to be executed asynchronously on the AWT event dispatching thread.

AWT and Swing [From Some other reference material]



Homework

-

Find out the difference between AWT and Swing in Java