



# Introduction to Graphical Interface Programming in Java

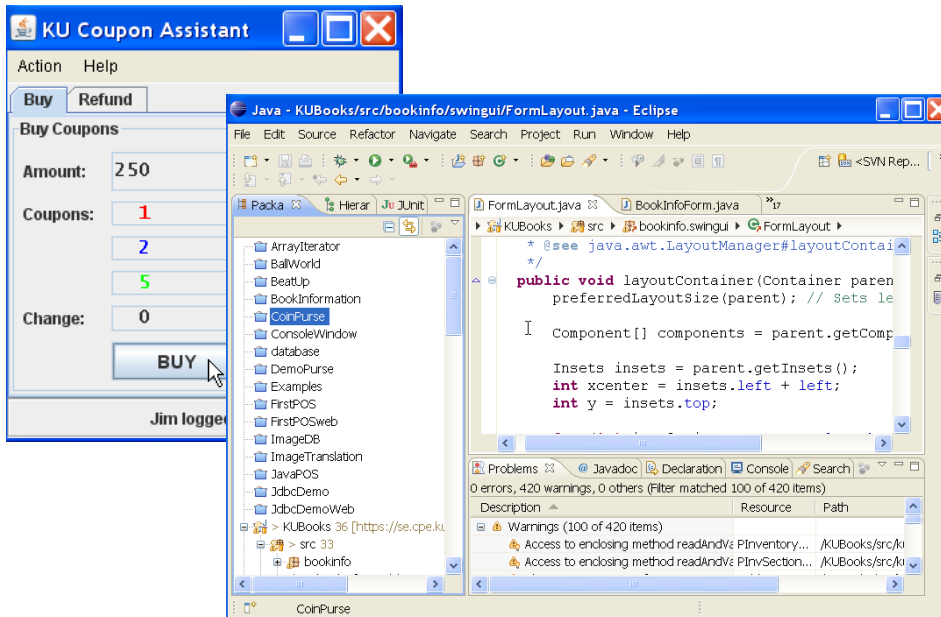
---

Introduction to AWT and Swing

# GUI versus Graphics Programming

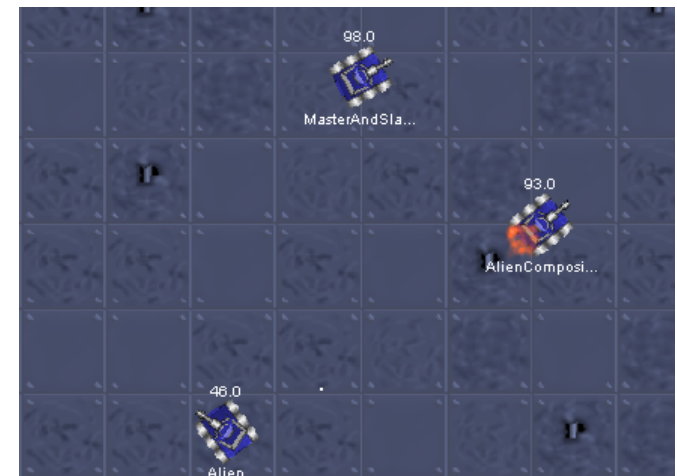
## Graphical User Interface (GUI)

- Purpose is to display info and accept user input
- Built using components
- "Forms" applications



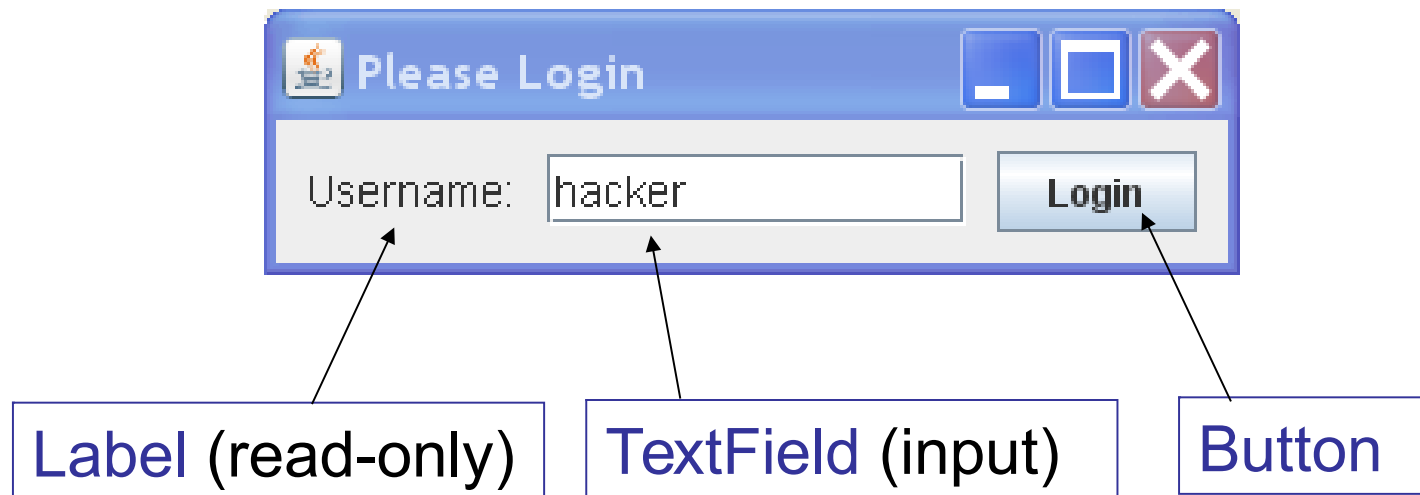
## Graphics Programming

- Manipulate graphical elements on a display
- points, lines, curves, shapes
- texture and lighting
- animation, 3D effect



# Simple GUI Application

A graphical interface is built using **components**.



# Designing a Simple GUI

1. create a window (a container)

2. create components

```
label = new Label("Username:");  
input = new TextField( 12 ); // the width  
button = new Button("Login");
```

3. layout components in a container

```
add( label );  
add( input );  
add( button );
```

4. display the container (**frame** is the container here):

```
frame.setVisible( true );
```

5. wait for user to do something

# Software for Graphics

Java provides **frameworks** for building graphical UIs.

**AWT** - Abstract Windowing Toolkit

- the first Java graphics framework

**Swing** - OS-independent framework.

**JavaFX** - layout is specified in XML (FXML), like **Android**

- use SceneBuilder tool
- can also create layout in code (like Swing)

**SWT** - Standard Widget Toolkit (used in Eclipse, Firefox)

Designed for efficiency. An Eclipse project.

# Graphics Toolkits

Java provides complete "**frameworks**" for building graphical applications.

A *framework* contains all the components and logic needed to manage a graphical interface.

You provide:

- **select components** and set their properties
- **write application logic** that controls how application responds to user actions
- you may **extend** (subclass) existing components to create custom components

# AWT Graphics Toolkit - `java.awt`

- Java's first GUI toolkit
- uses graphical UI components of operating system, e.g. Windows, MacOS, X-windows.
- **efficient**, low overhead
- applications look **different** on each platform
- **difficult to** write and **test** good quality applications
- different bugs on each OS, hence the slogan . . .

*"Write once, debug everywhere"*

# AWT Example

```
import java.awt.*;
public class AwtDemo {
    Frame frame;

    public AwtDemo( ) {
        frame = new Frame("Please Login");
        // create components
        Label label = new Label("Username:");
        TextField input = new TextField(12);
        Button button = new Button("Login");
        // layout components in the container
        frame.setLayout(new FlowLayout());
        frame.add(label);
        frame.add(input);
        frame.add(button);
        frame.pack();
    }
}
```



# AWT Example (continued)

To display the window, call `setVisible( true )`

```
public static void main(String [] args) {  
    AwtDemo demo = new AwtDemo( );  
    demo.frame.setVisible( true );  
}  
}
```

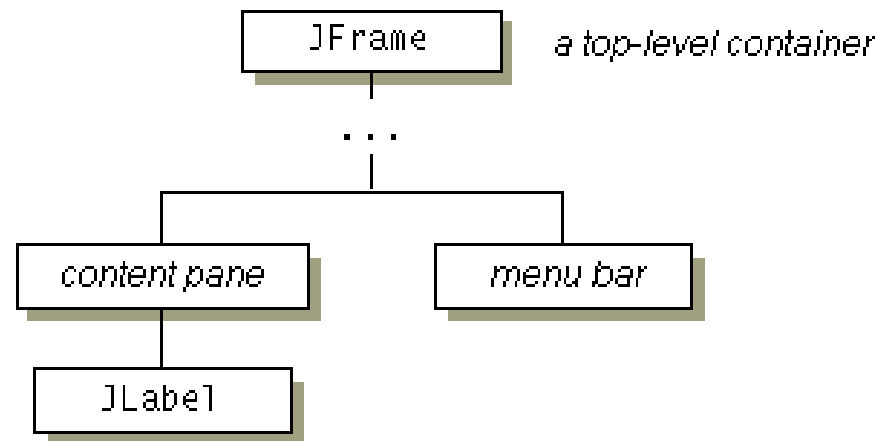
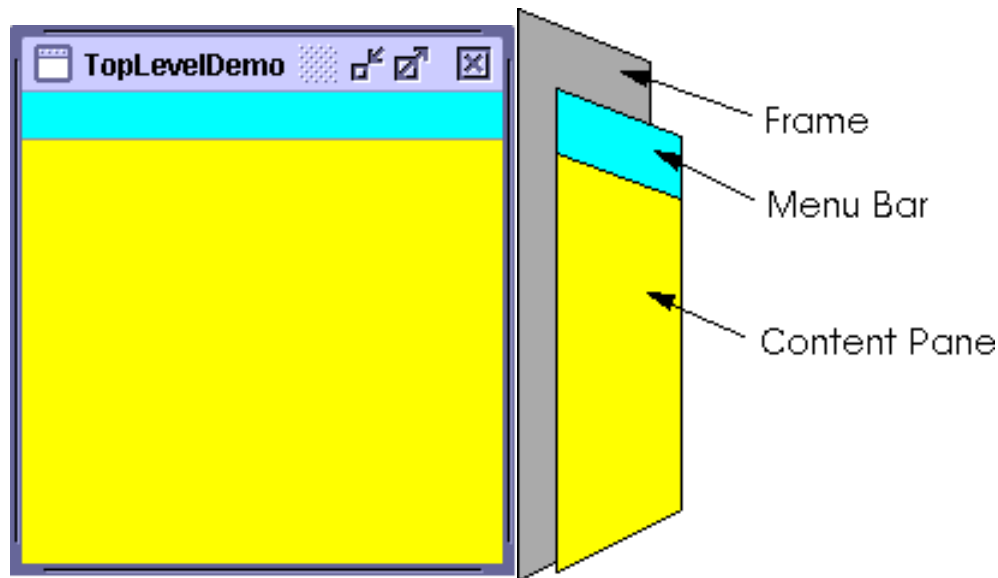
# Swing Graphics Framework

- Improved, OS-independent GUI framework
- classes in `javax.swing` and sub-packages
- Swing implements all components in software
- applications look & behave the same on all platforms
- applications *may not look* like native GUI apps (but you can change that by applying a LookAndFeel)
- part of the Java Foundation Classes (JFC).  
JFC also includes a 2D API and drag-and-drop API

# Swing Containers

JFrame is the top level container for most applications

- has a **title bar**, **menu bar**, and **content pane**
- JFrame is a **heavy weight** component.



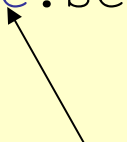
# Swing Example

```
import javax.swing.*;
public class SwingDemo {
    JFrame frame;
    public SwingDemo( ) {
        frame = new JFrame("Please Login");
        // create components
        JTextField label = new JLabel("Username:");
        JTextField input = new JTextField(12);
        JButton button = new JButton("Login");
        // layout components in the container
        ContentPane pane = frame.getContentPane();
        pane.setLayout(new FlowLayout());
        pane.add(label);
        pane.add(input);
        pane.add(button);
        pane.pack();
    }
}
```

# Swing Example (continued)

To display the window use `setVisible( true )`

```
public static void main(String [] args) {  
    SwingDemo demo = new SwingDemo( );  
    demo.frame.setVisible( true );  
}  
}
```



## **This is Bad Programming!**

We should not directly access the private attributes of an object. We should invoke public behavior instead.

# Demo

- Create the simple "Login" interface.



# What is Running My Program?

In the main method, setVisible(true) returns **immediately**.  
main() exits.

But, the application is still running!

**What is in control?** Where is the control of flow?

```
public static void main(String [] args) {  
    SwingDemo demo = new SwingDemo( );  
    demo.frame.setVisible( true );  
    System.out.println("UI started. Bye.");  
}  
}
```

# Swing creates its own Thread

---

Swing runs in a separate thread.

Or more than one thread.

One thread is special: the *Event Dispatcher Thread*

The Event Dispatcher Thread receives and "dispatches" all events involving the UI.

You should perform all UI updates in this thread.



# Top Level Containers

A window that be displayed on screen:

**JFrame** - title bar, menu, and content pane

**JWindow** - no title bar or menu.

**JDialog** - a dialog window, has title bar

**JApplet** - for Applets (run in web browser).

Deprecated now.

# What's a Component?

- Also called "widgets".

Label

Textbox

Button

Slider

Checkbox

ComboBox

RadioButton

ProgressBar

# Know your components

---

You need to know the available components  
... and what they can do.

## Visual Guide to Components

Sun's *Java Tutorial*

*.../tutorial/ui/features/components.html*

*for Windows:*

*tutorial/ui/features/compWin.html*

# What Can a Component *Do*?

## Common Behavior

`setLocation( x, y )`

`setIcon( ImageIcon )`

`setBackground( color )`

`setForeground( color )`

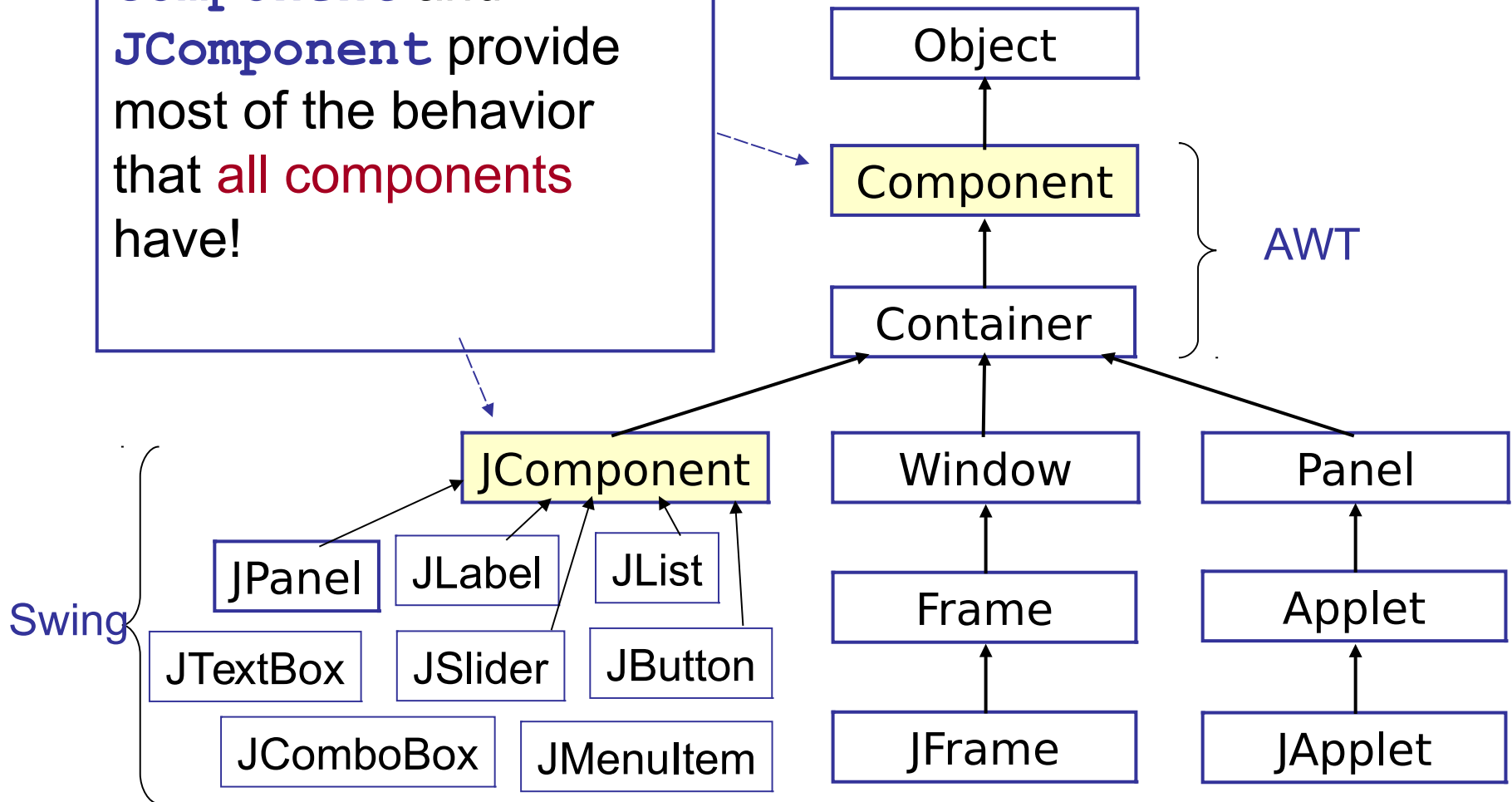
`setEnabled( boolean )`

`setVisible( boolean )`

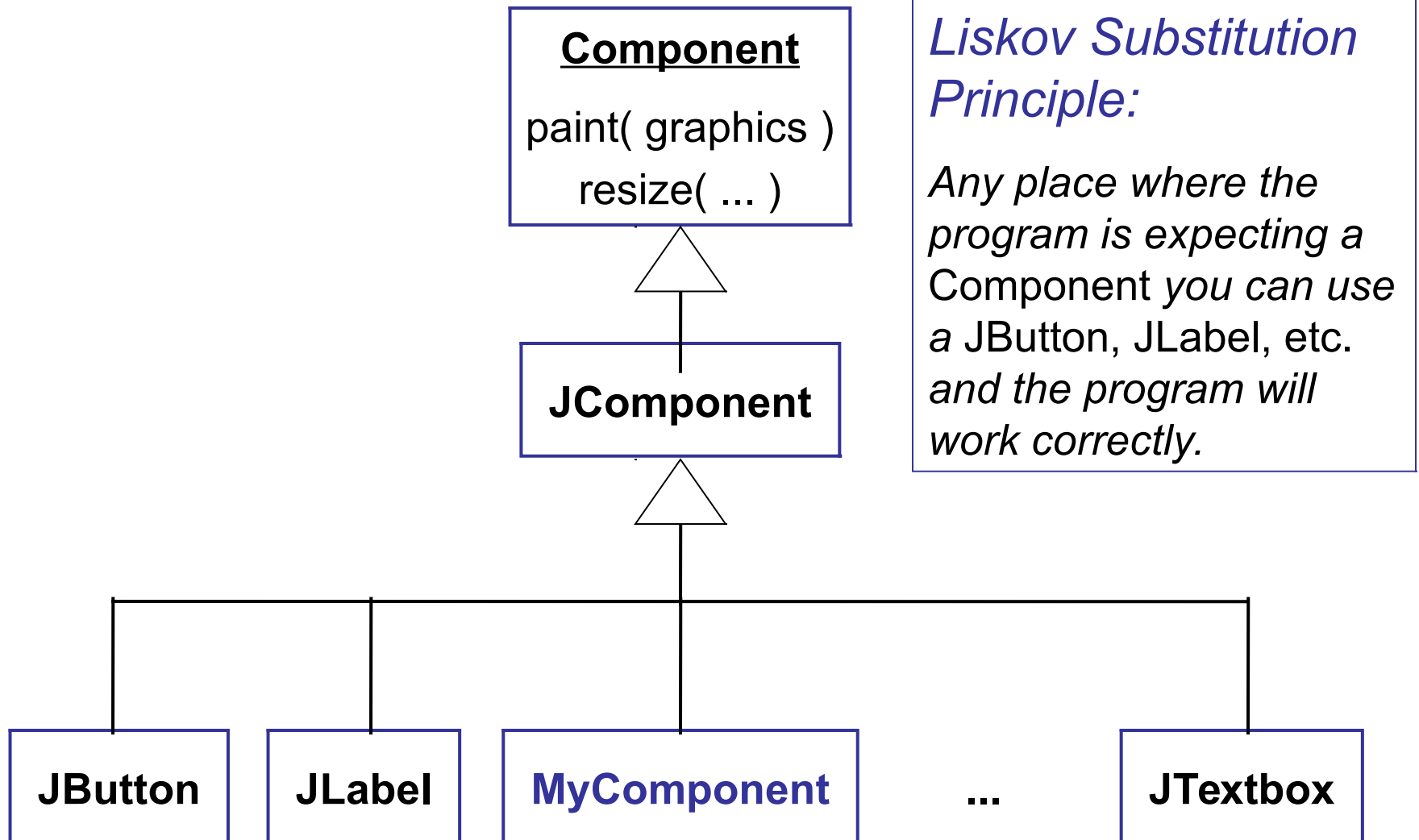
`setToolTipText( string )`

## Important Components to Know

**Component** and **JComponent** provide most of the behavior that **all components** have!



# Hierarchy of Graphics Components



# Playing with a JButton

```
import java.awt.*;  
import javax.swing.*;  
JButton button = new JButton( "Press Me" );  
//TODO: add button to a frame and pack  
button.setBackground( Color.YELLOW );  
button.setForeground( Color.BLUE );  
button.setToolTipText( "Make my day." );  
button.setFont( new Font( "Arial", Font.BOLD, 24 ) );  
button.setEnabled( true );
```

# Components don't have to be boring

filename, URL, or InputStream.

Can be GIF, JPEG, PNG

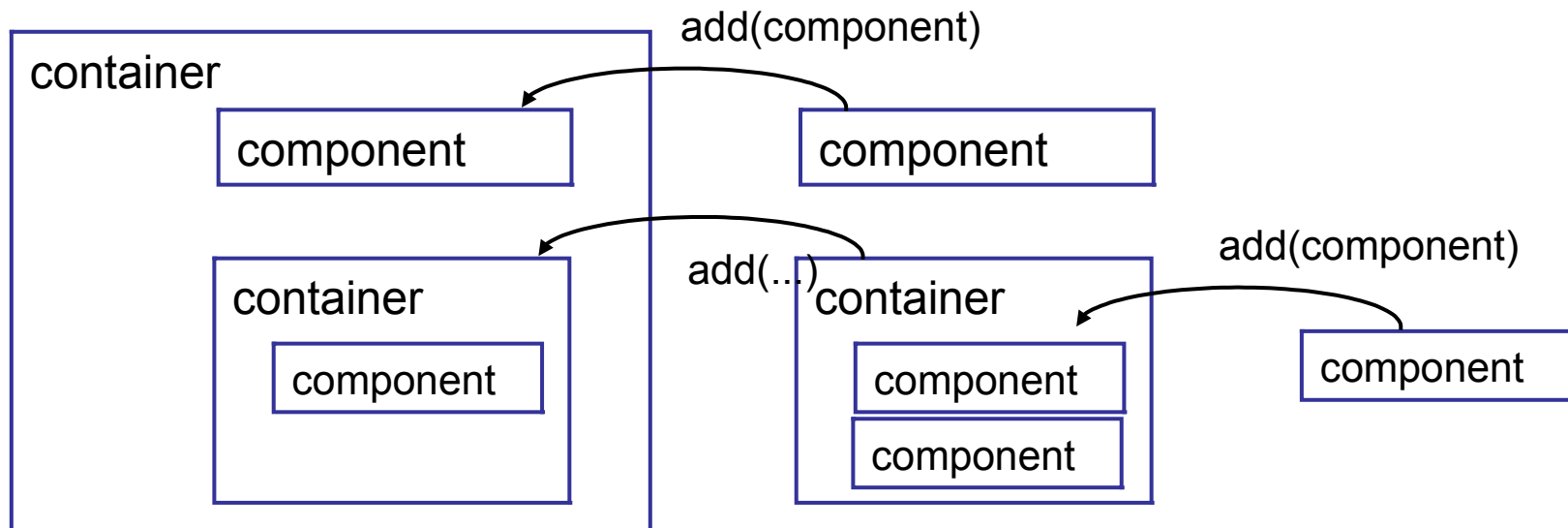
```
ImageIcon icon = new ImageIcon("d:/images/fish.png");  
button.setIcon( icon );
```





# Containers and Components

- A GUI has many **components** in **containers**.
- Use **add** to put component in a container.
- A container is also a component; so a **container** may *contain other containers*.



# Lightweight Containers

A lightweight container is one that **is not** a window.

You must place it inside another container.

Cannot be drawn on screen by itself.

- **JPanel** simple rectangular area - **most common**
- **JTabbedPane** - multiple panels with a tab on top
- **JSplitPane**
- **JInternalFrame** - like a JFrame inside a JFrame

# Learning Java Graphics

- Java Tutorial: *Creating a GUI with JFC/Swing*

The section "*Using Swing Components*" contains "How To..." examples for many components.

Good explanation and examples of Layout Managers.

- "JDK Demos and Samples"

*jdk\_dir/demo/jfc/\**

Great demos with jar files (run) and source code.

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

- *Big Java*, Chapter 19.

# WindowBuilder for Eclipse

---

A graphical editor for creating UI with Swing or SWT.

Windowbuilder is included in **Eclipse**, but you *may* need to download & install extra components.

To find the Window Builder "Update Site" for your version of Eclipse, see:

<http://www.eclipse.org/windowbuilder/download.php>

