Introduction to Graphical Interface Programming in Java

James Brucker

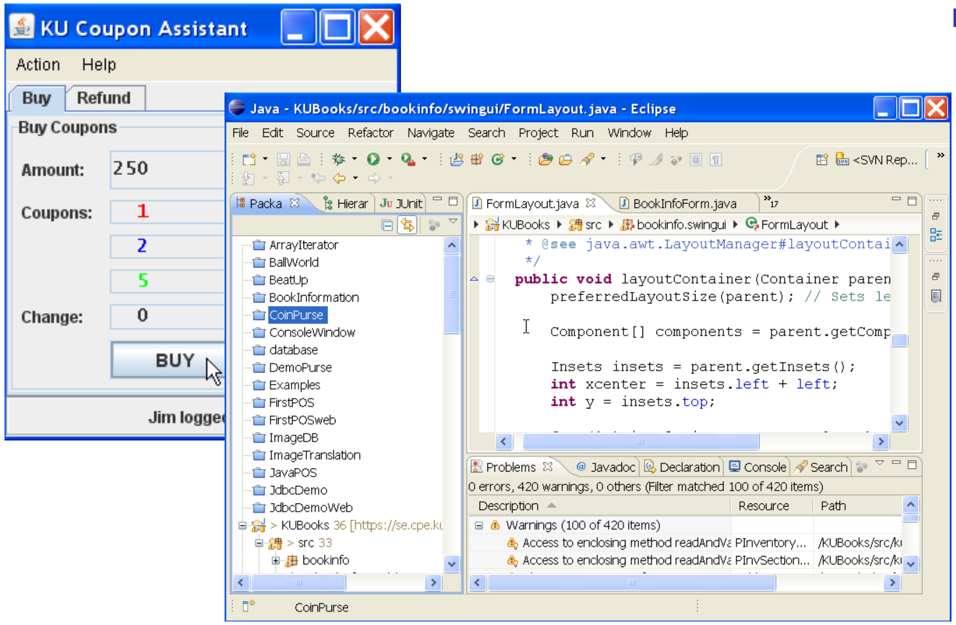
GUI versus Graphics Programming

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



Amount:	250
Coupons:	1
	2
	5
Change:	0

Jim logge

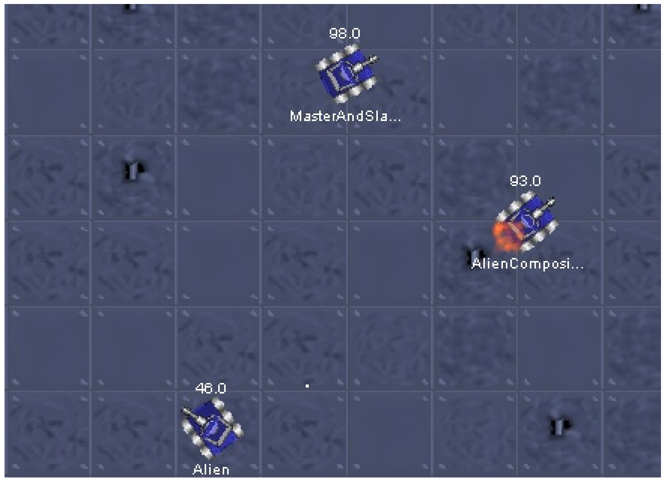
```
 * @see java.awt.LayoutManager#layoutContai
 */
public void layoutContainer(Container paren
    preferredLayoutSize(parent); // Sets le

    Component[] components = parent.getComp

    Insets insets = parent.getInsets();
    int xcenter = insets.left + left;
    int y = insets.top;
```

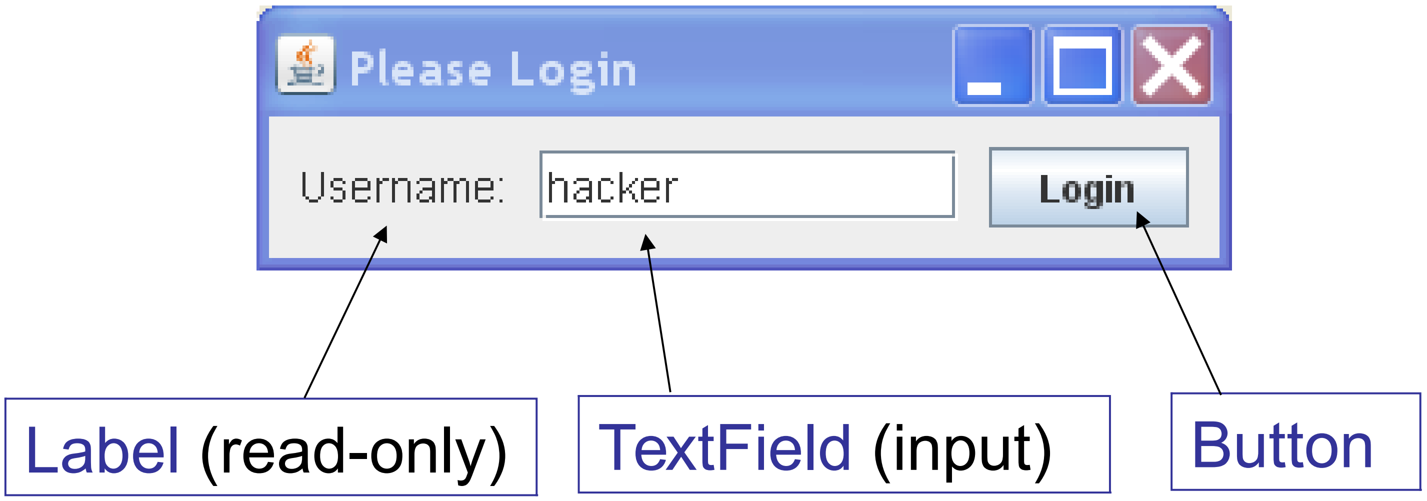
0 errors, 420 warnings, 0 others (Filter matched 100 of 420 items)

Description	Resource	Path
Warnings (100 of 420 items)		
Access to enclosing method readAndVa	PInventory...	/KUBooks/src/ku
Access to enclosing method readAndVa	PInvSection...	/KUBooks/src/ku



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 apps.

AWT - Abstract Windowing Toolkit

- the first Java graphics framework

Swing - newer, OS-independent framework

SWT - Standard Widget Toolkit (Eclipse, Firefox)

- Eclipse project, designed for efficiency

Java 3D, JMonkey, JOGL, ...

- frameworks for 3D interfaces

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`

- uses GUI of operating system, e.g. Windows, MacOS,
- efficient, low overhead
- applications look different on each platform (Linux, Mac, Windows)
- difficult to write and test good quality applications
- different bugs on each OS (due to difference in OS GUI) . . .

"*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 use **`setVisible( true )`**

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

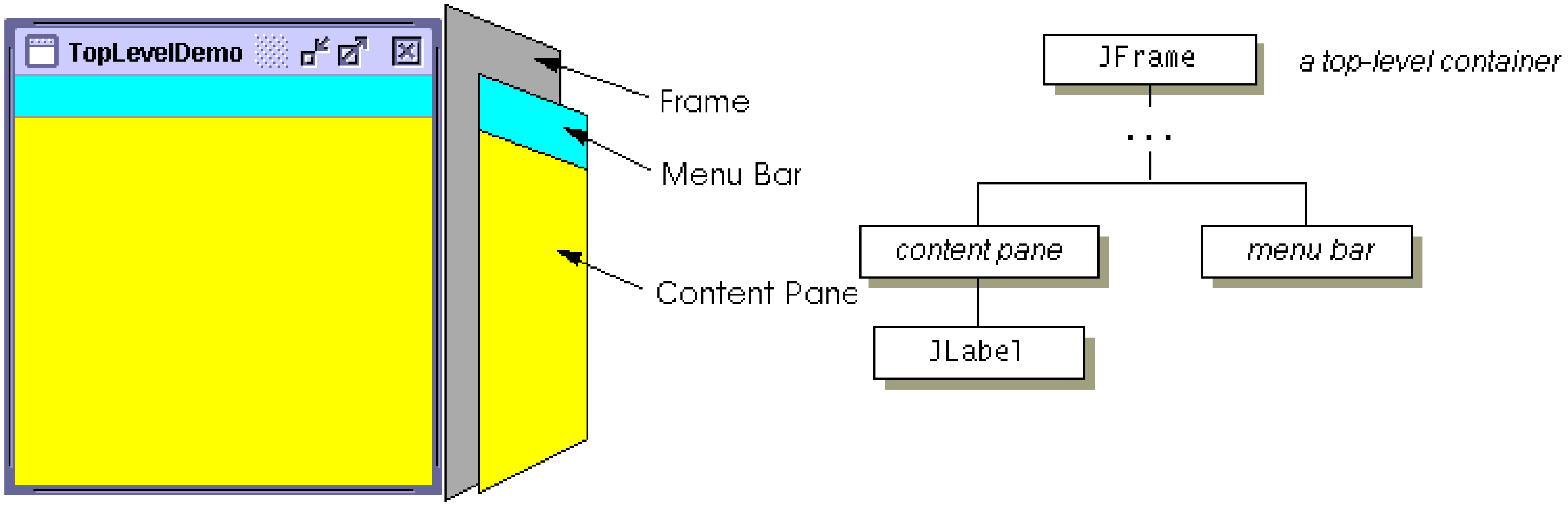
Swing Graphics Toolkit

- `javax.swing` packages
- Swing doesn't depend on OS graphics, uses only basic functionality such as painting a region.
- provides its own behavior, own "look and feel"
- applications look the same on all platforms
- applications *may not* look like native GUI apps
- slower than AWT, because more work done in software
- 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 )`**

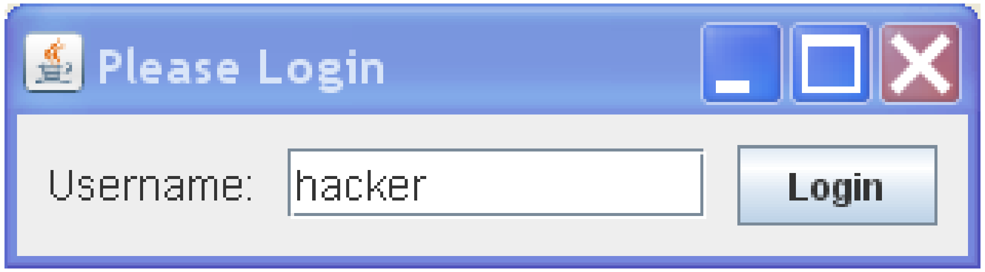
```
  public static void main(String [] args) {
    SwingDemo demo = new SwingDemo( );
    demo.frame.setVisible( true );
  }
}
```

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.



Top Level Containers

A window that be displayed on screen:

JFrame - title bar, menu, and content pane

JWindow - no title bar or menu.

JDialog - for creating dialog windows

JApplet - for Applets (run in web browser). No title bar or border.

What's a Component?

- Also called "widgets".

Label

Textbox

Button

Slider

Checkbox

ComboBox

RadioButton

ProgressBar

xxx

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

and (for Windows):

.../tutorial/ui/features/compWin.html

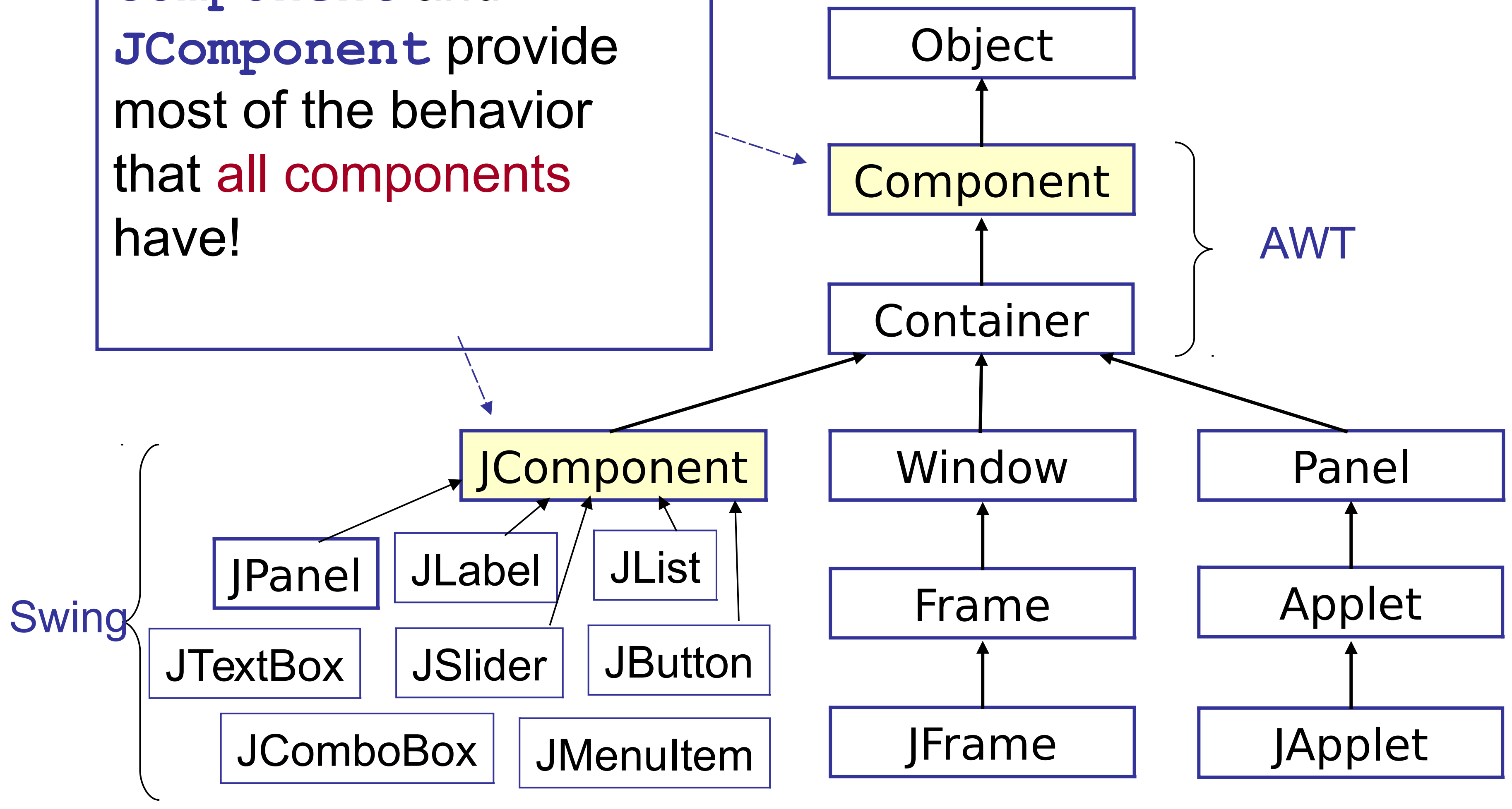
What Can a Component *Do*?

Common Behavior

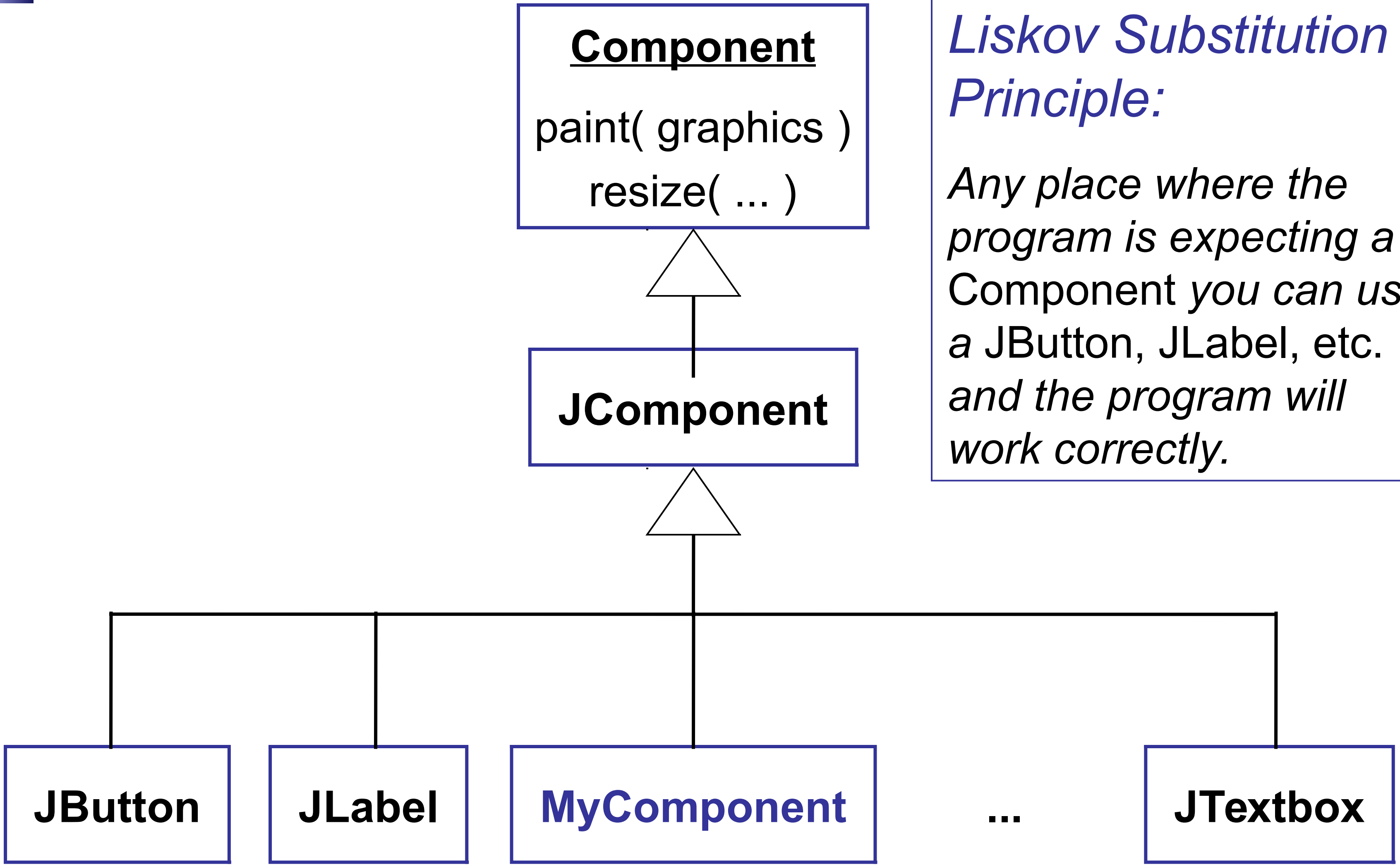
```
setLocation( x, y )
setIcon( imageIcon )
setBackground( color )
setForeground( color )
setEnabled( boolean )
setVisible( boolean )
setToolTipText( string )
```

Important Components to Know

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



Hierarchy of Graphics Components



Liskov Substitution Principle:

Any place where the program is expecting a Component *you can use a* JButton, JLabel, etc. *and the program will work correctly.*

Playing with a JButton

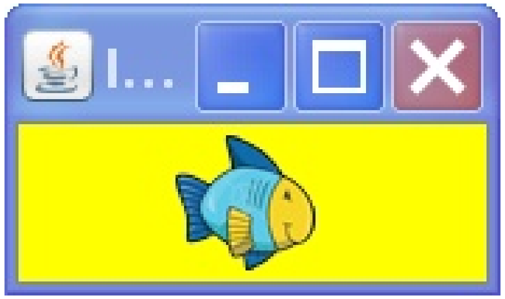
```
import java.awt.*;
import javax.swing.*;
JButton button = new JButton( "Press Me" );
//TODO: add button to a frame and pack
button.setBackground( Color.YELLOW );
button.setForeground( Color.BLUE );
button.setToolTipText( "Make my day." );
button.setFont( new Font( "Arial", Font.BOLD, 24 ) );
button.setEnabled(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

Components don't have to be boring

filename, URL, or InputStream.

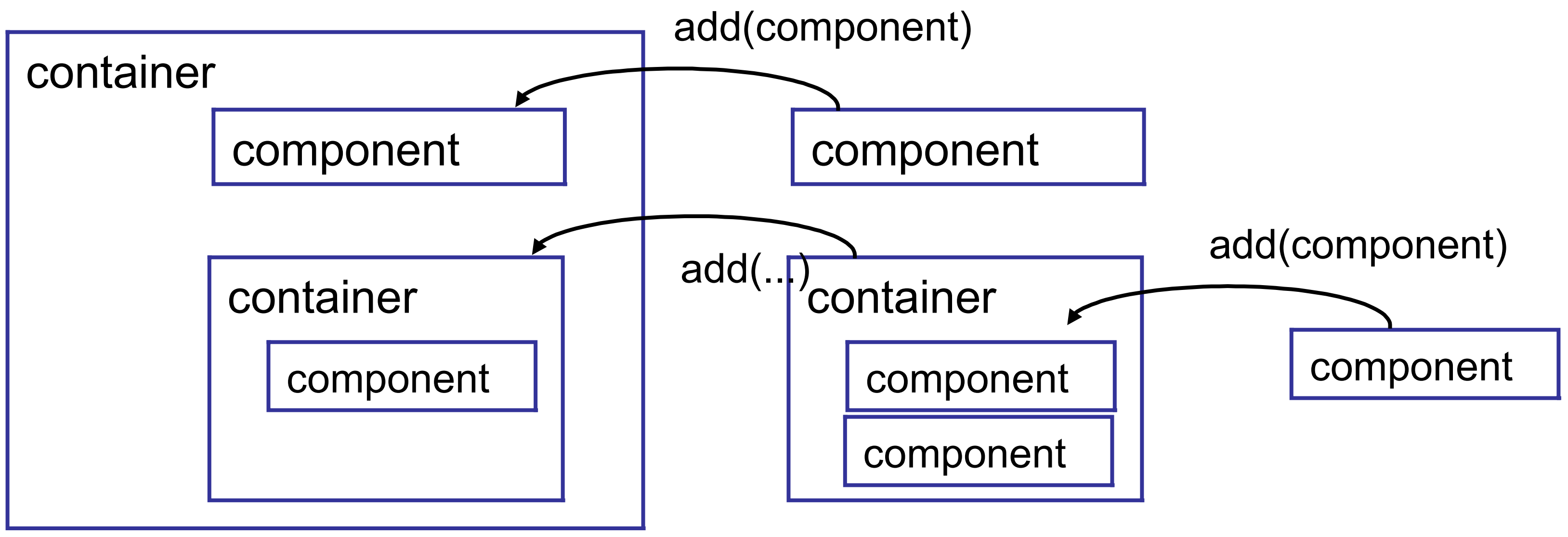
Can by GIF, JPEG, PNG

```
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

Steps to Creating a GUI Interface

The Secrets of GUI Interface revealed.

Steps to creating a GUI Interface

1. Design it on paper

2. Choose components and containers

3. Create a window or dialog.

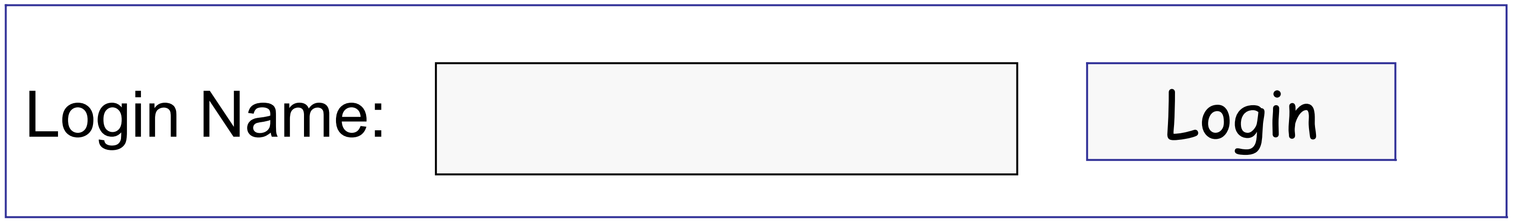
4. Add components to the window.

5. Preview the UI.

6. Add behavior - respond to user actions.

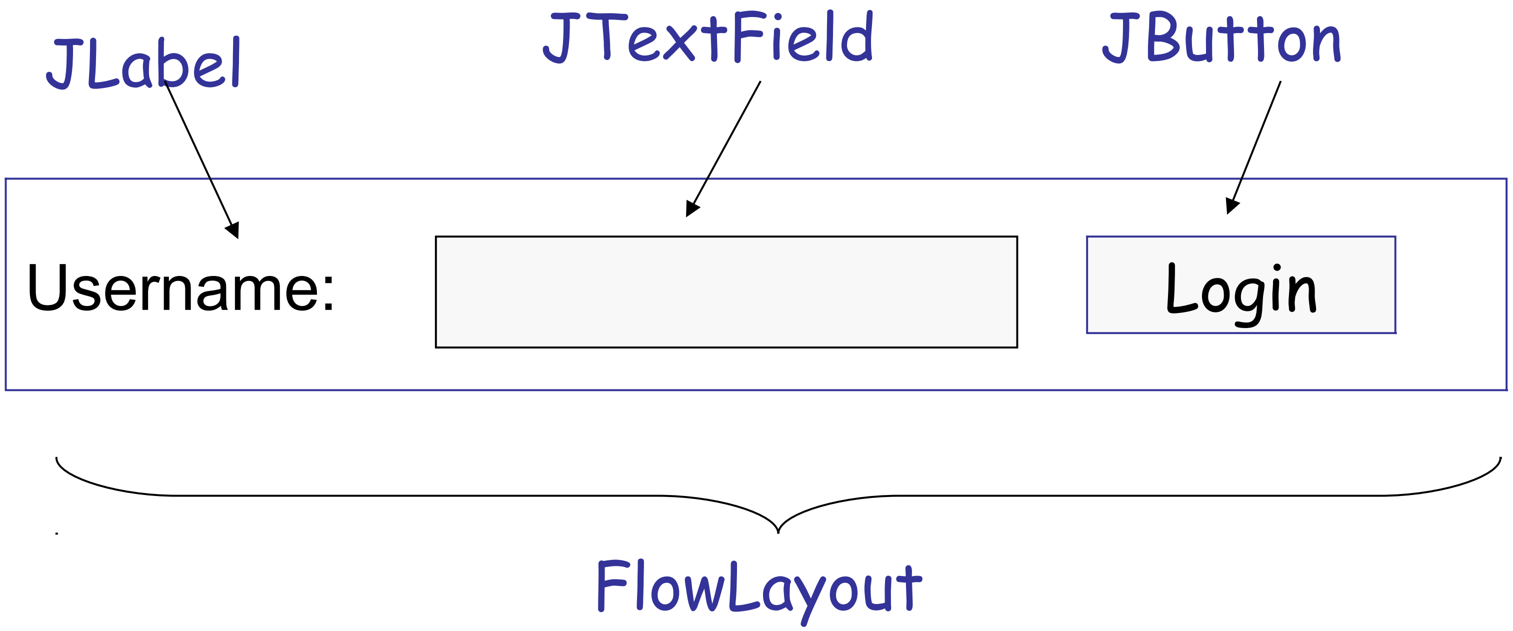
Step 1: Design it on paper

- Know what the interface is supposed to do
- Decide what information to present to user and what input he should supply.
- Decide the components and layout on paper



Step 2: Choose Components & Layout

- Choose the components and layout on paper



Step 3: Create a Window (JFrame)

```
import javax.swing.*;
public class SwingExample implements Runnable {
    JFrame frame;
    public SwingExample( ) {
        frame = new JFrame();
        frame.setTitle("Please Login");
        initComponents( );
    }

    private void initComponents( ) {
        // initialize components here
        frame.pack();
    }

    public void run() {
        frame.setVisible( true );
    }
```

Step 3: (alt) Be a JFrame

```
import javax.swing.*;
public class SwingExample extends JFrame {

   public SwingExample( ) {
      super.setTitle("Please Login");
      initComponents( );
   }

   private void initComponents( ) {
      // initialize components here
      this.pack();
   }

   public void run() {
      this.setVisible( true );
   }
```

Step 3.1: Decorate the Frame

- We can add decoration to the JFrame or components.
- Add a title:

```
   frame.setTitle( "Please Login" );
```

Step 3.2: Close Application on Exit?

- Even if you close the window, the GUI thread still running!
  - GUI applications can run forever!
  - Your program must tell the GUI to exit.

How to make it quit when you close the window?

1. handle a WindowClosingEvent, or
2. specify Exit-on-Close behavior

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

Step 4: Add Components to window

- Add components to the *content pane* of a JFrame.
- As a *convenience*, you can **add** directly to JFrame.

```
private void initComponents( ) {
   JLabel label1 = new JLabel("Username:");
   input = new JTextField( 12 );
   button = new JButton("Login");

   frame.add( label1 );
   frame.add( input );
   frame.add( button );
   // pack components. This sets the window size.
   frame.pack( );
}
```

Step 4.1: Choose a Layout Manager

Each container uses a **Layout Manager** to manage the position and size of components.

Classic Layout Managers are:

- BorderLayout (default for JFrame)
- FlowLayout (default for JPanel)
- BoxLayout
- GridLayout
- GridBagLayout (the most powerful)
- CardLayout

4.2: Set the Layout Manager

Set the container's layout manager

```
private void initComponents( ) {
   JLabel label1 = new JLabel("Username:");
   input = new JTextField( 12 );
   button = new JButton("Login");
   frame.setLayout( new FlowLayout() );
   frame.add( label1 );
   frame.add( input );
   frame.add( button );
   frame.pack( );
}
```

Adding Components to a Panel

Most graphical UI use many "panels" or "panes" to group components.

This makes layout and management easier.

```
void initComponents( ) {
   JLabel label1 = new JLabel("Username:")
   input = new JTextField( 12 );
   button = new JButton("Login");

   JPanel panel = new JPanel();
   panel.add( label1 );
   panel.add( input );
   panel.add( button );

   frame.getContentPane( ).add( panel );
```

Put components in a **panel**

Add **panel** to the frame

Step 5: Preview the Interface

To show the window, call **`setVisible(true)`**

```
public class SwingExample {
   ....
   // create a run() method to display the window
   public void run() {
      frame.setVisible( true );
   }
```

```
public class Main {
   public static void main( String [] args ) {
      SwingExample gui = new SwingExample( );
      gui.run( );
   }
```

Problem: Window is too small

- If your application shows only a title bar, it means you forgot to set the window size.

You must either:

  - **`pack()`** **`or`**
  - **`setSize(width, height)`**

Usually you should use **`pack( )`**

```
public class SwingExample {
   JFrame frame;
   ...
   public void run() {
      frame.pack( );    // set size = best size
      frame.setVisible(true);
   }
```

Step 6: Add Behavior

Your application must ***do something*** when user presses a button, moves the mouse, etc.

Graphics programs are ***event driven.***

Events:

- button press
- got focus
- mouse movement
- text changed
- slider moved

Why a layout manager?

Demo:

compare a Java application and Visual C# application when resizing a window.

In Java, the layout manager will rearrange or resize components.

In Visual C#, the components disappear.

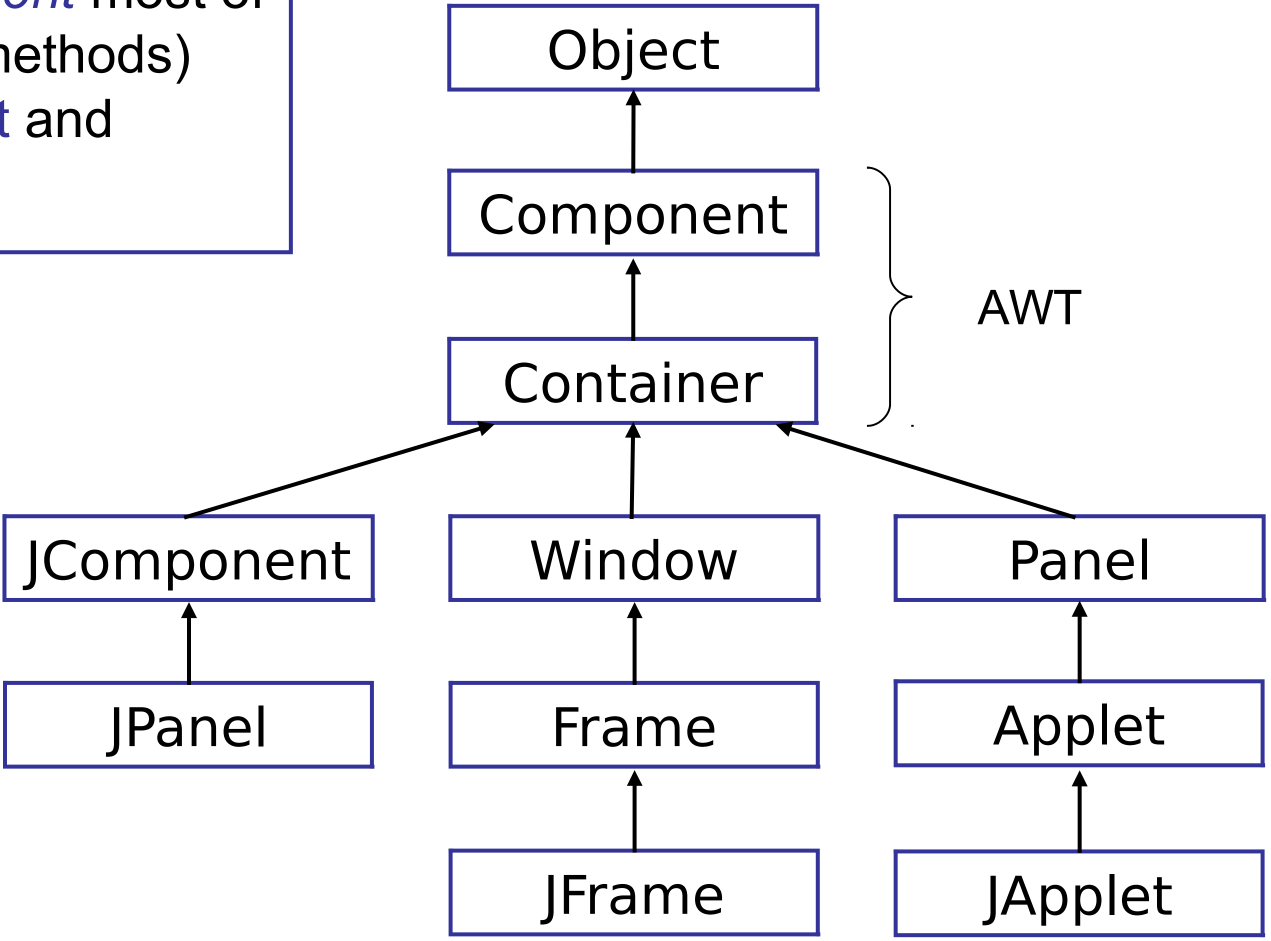
Layout Managers

Classic Layout Managers are:

- BorderLayout (default for JFrame)
- FlowLayout (default for JPanel)
- BoxLayout
- GridLayout
- GridBagLayout
- CardLayout
- SpringLayout

Graphics Class Hierarchy (again)

Components *inherit* most of their behavior (methods) from Component and JComponent.



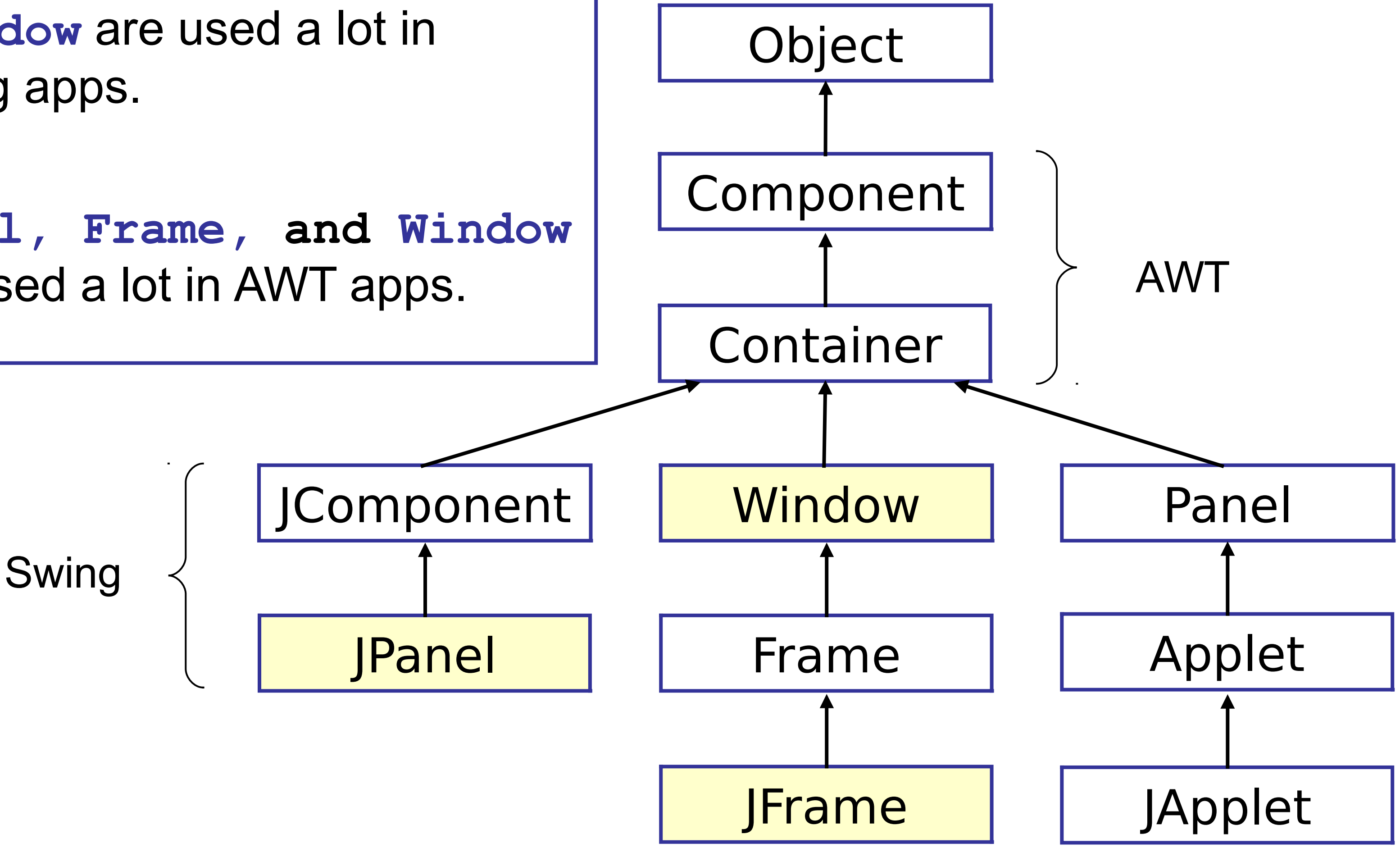
Exercise: JComponent

- Look at JavaDoc for **`JComponent`**.
- What properties can you "set" for any component?

Important Containers to Know

`JPanel`, `JFrame`, and `JWindow` are used a lot in Swing apps.

`Panel`, `Frame`, and `Window` are used a lot in AWT apps.

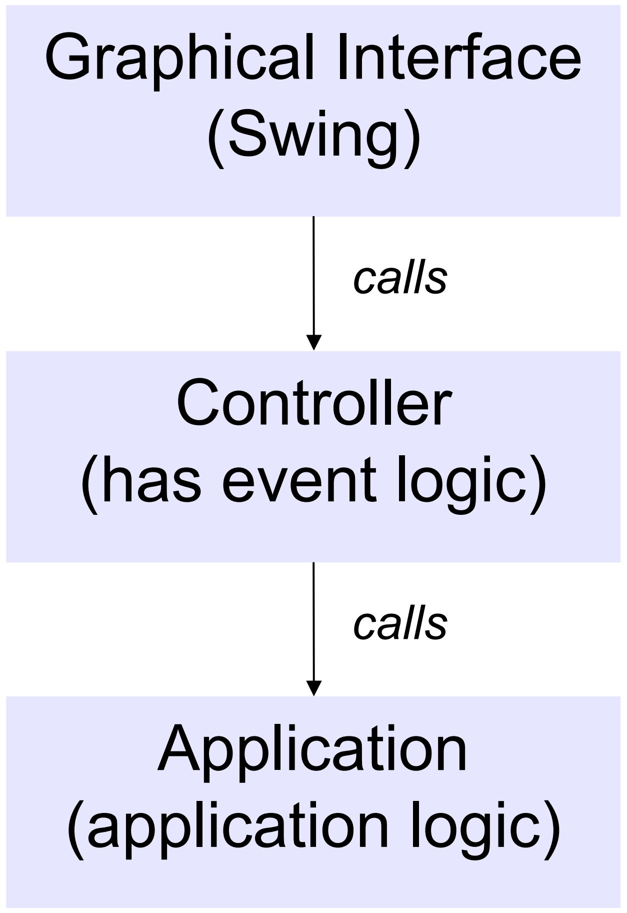


How to Design a GUI Application

Separate the GUI classes from the program logic.

- Program logic is part of the "model" or domain layer.
- GUI *calls* model for information.
  - Try to limit GUI -> Model communication to *just one class*
  - This reduces coupling between GUI and logic classes.
- Model ***does not call*** methods of GUI objects.
  - Use the **Observer Pattern.** Model (observable) notifies GUI (observer) when its state changes.

Layers in a GUI Application



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.