Unit 5.3

GUIs

UD

Sep 2023

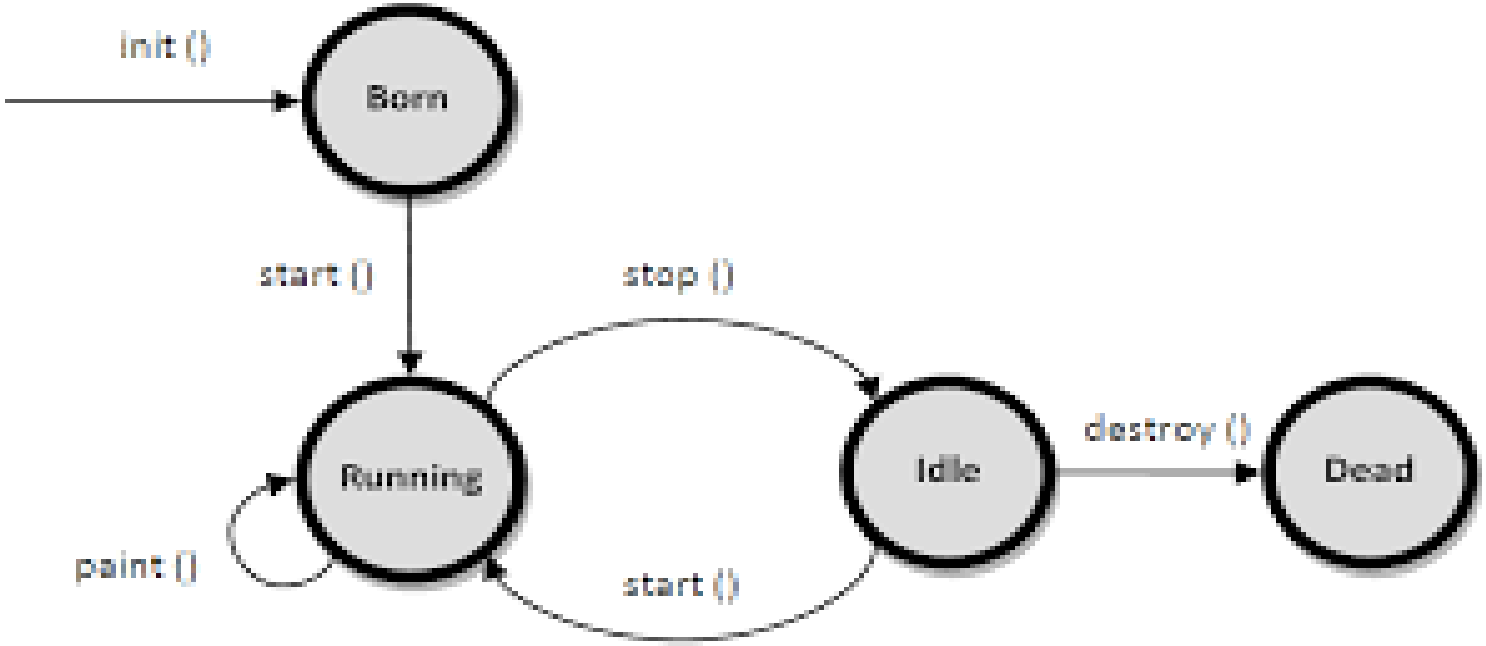
Introduction

- Application Programs
  - ✓Created: typically on local computer
  - ✓Compiled: using javac
  - ✓Executed: using java interpreter
  - ✓Each application programs contains one main() method

- Applets (***no longer supported in Java***)
  - ✓Small programs primary used in Internet Programming; browser supplies the main method
  - ✓Created: Either developed in local systems (Local Applet) or remote systems (Remote Applet) and executed thru Java compatible Web Browser or appletviewer (command-line)
  - ✓**Local Applet** stored in local system; can be located w/o Internet connection
  - ✓Find **Remote Applet** using Uniform Resource Locator (**URL**) and download via internet for execution (download-on-demand)
  - ✓Display data provided by the server, handle user input or provide simple functions, e.g premium calculator that **execute locally** rather than on server allowing some functionality moved from server to client

***Note**: Many browsers removed standards-based plugin support as Applet is deprecated since JDK 9. **Java** developers should migrate from **Java applets** to plugin-free **Java** Web-start technologies.*

Applet Life Cycle



- Born state:
  - ✓ Enters this state when first loaded
  - ✓ Achieved by calling init() of Applet class, e.g. public void init()
  - ✓ Occurs only once in applet's life cycle
  - ✓ Best place to define the GUI Components (buttons, text fields, scrollbars, etc.), lay them out, and add listeners to them

Applet Life Cycle

- Running state:
  - ✓ Enters this state when start() method of Applet class is called , e.g. public void start()
  - ✓ start() method is automatically called after the browser calls init()
  - ✓ Called to restart an applet after it has been stopped
  - ✓ Unlike init(), start() can be called any number of items
  - ✓ start() called each time user leaves web page and comes back (page to be loaded) -> applet resumes execution (restarted)
  - ✓ Used mostly in conjunction with stop( )
  - ✓ start() and stop( ) are used when the Applet is doing time-consuming calculations that you don't want to continue when the page is not in front

- Idle or Stopped state:
  - ✓ Applet becomes idle when it is stopped, e.g. public void stop()
  - ✓ stop() method automatically called when the user leaves page with current applet
  - ✓ Can be called repeatedly in the same applet
  - ✓ stop() method suspends applet execution until the user clicks on it

Applet Life Cycle

- Dead state:
  - ✓ Enters this state when the applet is from memory and achieved by calling destroy() method of Applet class, e.g. public void destroy()
  - ✓ destroy() method performs shutdown activities, e.g. release of threads etc.
  - ✓ destroy() method called only when the browser shuts down

- Idle or Stopped state:
  - ✓ Applet becomes idle when it is stopped, e.g. public void stop()
  - ✓ stop() method automatically called when the user leaves page with current applet
  - ✓ Can be called repeatedly in the same applet
  - ✓ stop() method suspends applet execution until the user clicks on it
  - ✓ stop() called just before destroy( )
  - ✓ Use stop( ) if the applet is doing heavy computation that you don't want to continue when the browser is on some other page
  - ✓ Used mostly in conjunction with start()

- Display state:
  - ✓ Enters this state whenever it has to perform some output operation on the screen and achieved by calling paint() method of Applet class, e.g. public void paint()
  - ✓ Happens immediately after applet enters into running state
  - ✓ Override paint() method to display required graphics

Simple Applet

```
import java.awt.*;
import java.applet.Applet;
/*<Applet
   code="HelloWorld.class"
   width=150 height=100>
</Applet>
*/
public class HelloWorld extends Applet {
   public void paint( Graphics g ) {
      g.drawString( "Hello World!", 30, 30 );
                  System.out.println("Running Java program
directly!");
   }
}
```

<u>Commands:</u>

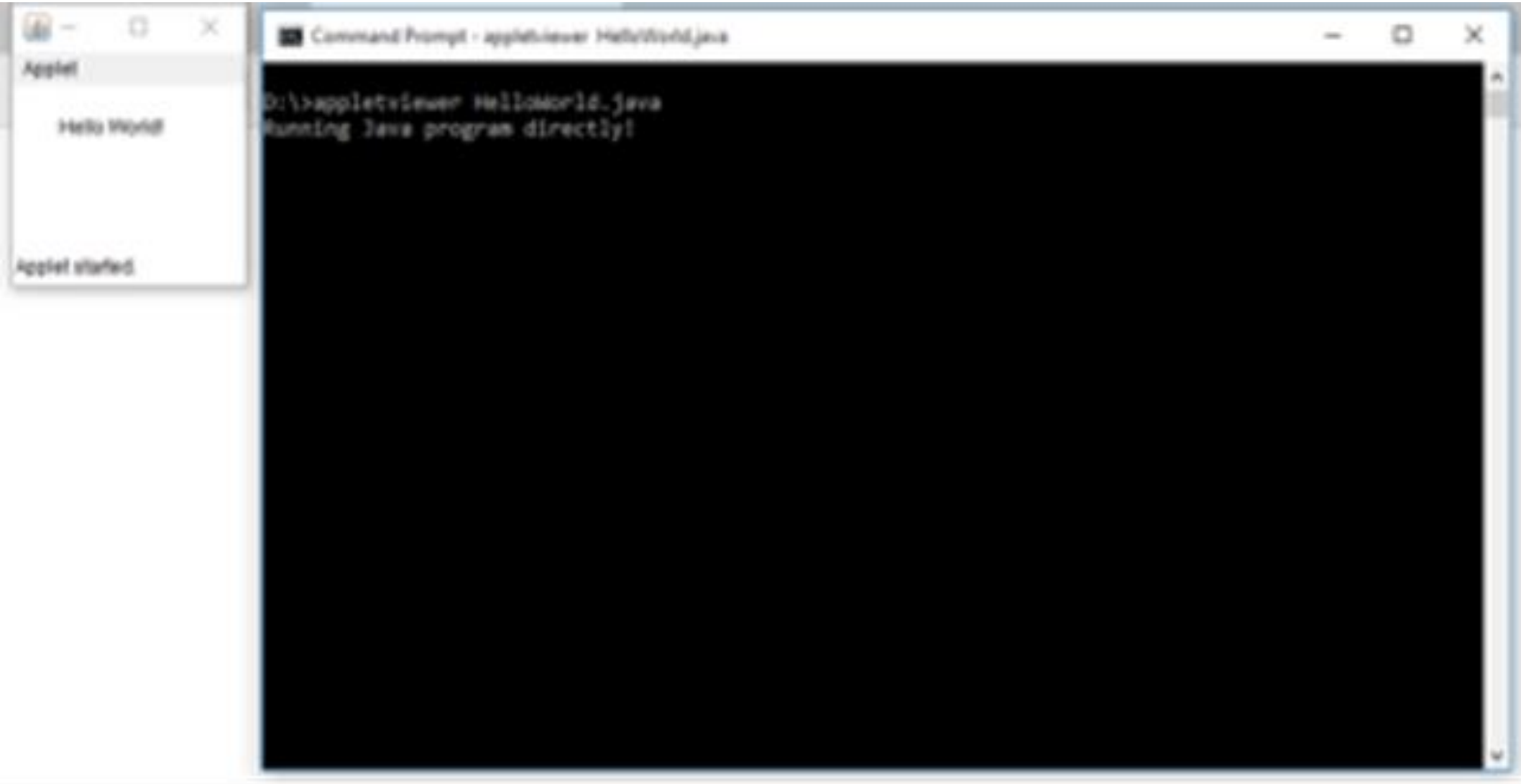
$ javac HelloWorld.java

$ appletviewer HelloWorld.**java**

Note: Above uses commented HTML. Alternatively, create Helloworld.html file and run

$ appletviewer Helloworld.html

Running Simple Applet



Simple Graphic methods

- A Graphics is something you can paint on

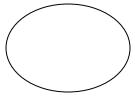
```
g.drawString("Hello", 20, 20);
```

Hello

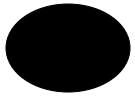
```
g.drawRect(x, y, width, height);
```

```
g.fillRect(x, y, width, height);
```

```
g.drawOval(x, y, width, height);
```



```
g.fillOval(x, y, width, height);
```



```
g.setColor(Color.red);
```

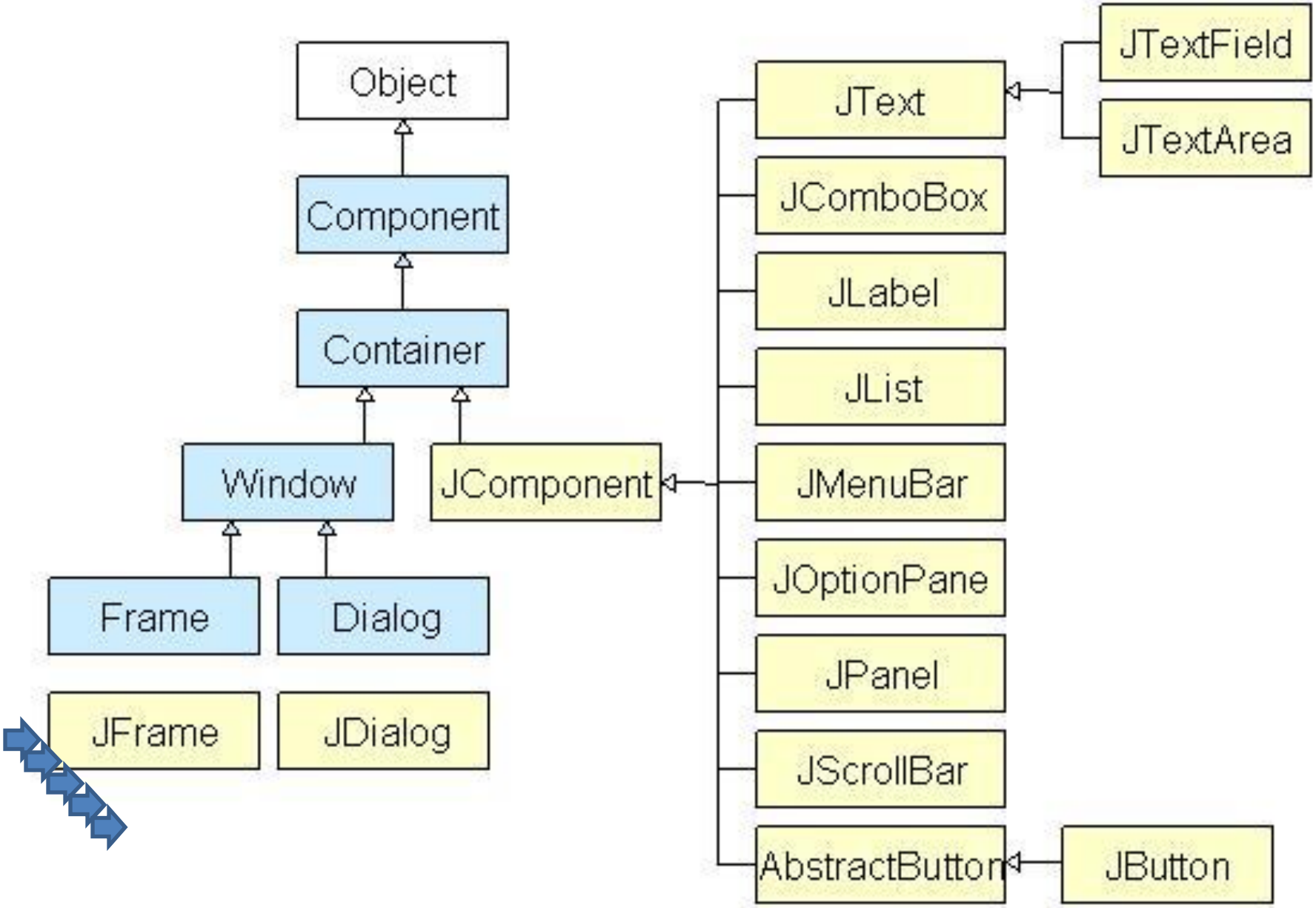
SWING

- Graphic User Interface (GUI) Toolkits
  - ✓ **AWT**: Abstract Window Toolkit: used in earlier days; heavy-weight or platform-specific functions and look-and-feel; limited features
  - ✓ **SWING**: Modern GUI; builds on the foundation of AWT; addresses AWT limitations; lightweight components; pluggable look-and-feel features

- <u>SWING Key Items</u>
  - ✓ **Components**:
    - Independent visual control, e.g push button, slider etc.
    - Derived from **JComponent** class except the 4 top-level containers
    - **JComponent** class in turn inherited from **AWT classes Container** and **Component**
    - Classes defined under **javax.swing**

  - ✓ **Containers**: group of components:
    - Top-level or Heavy-weight containers: **JFrame, Japplet, JWindow** and **Jdialog**. Inherited from **AWT classes Container** and **Component**
    - Lightweight containers, e.g. **Jpanel** and **JRootPane**: Inherits **Jcomponent**

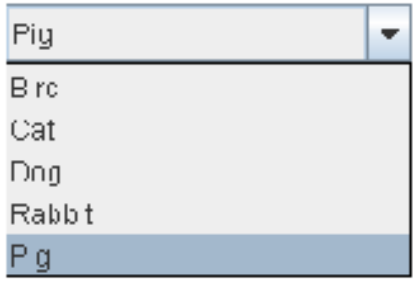
SWING Class Hierarchy Diagram



SWING Components

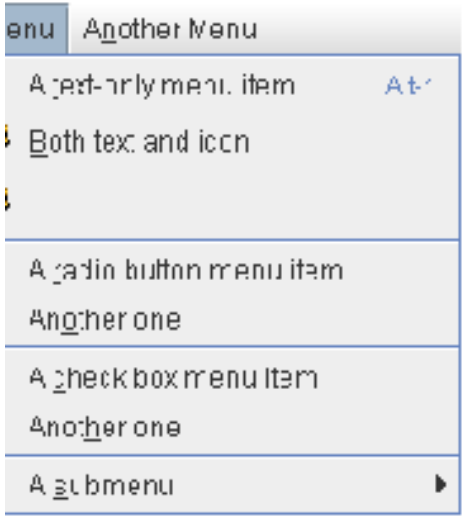
JButton

JCheckBox



JComboBox

JList

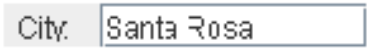


JMenu

JRadioButton

JSlider

JSpinner



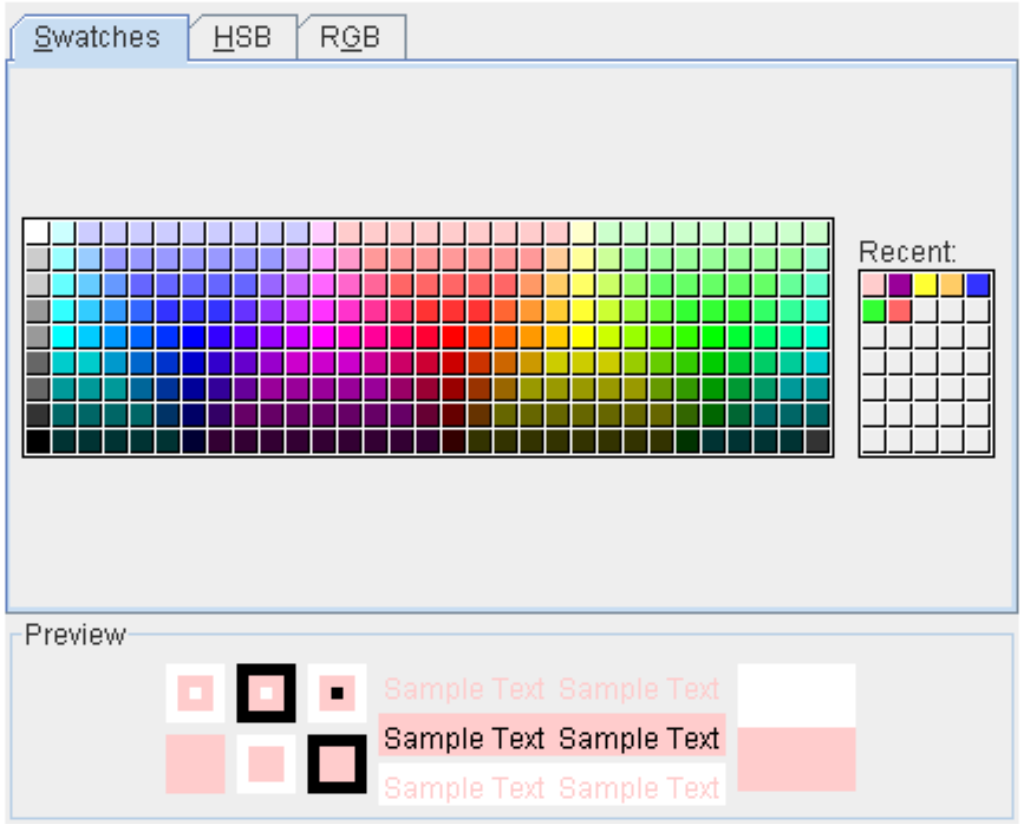
JTextField

JPasswordField

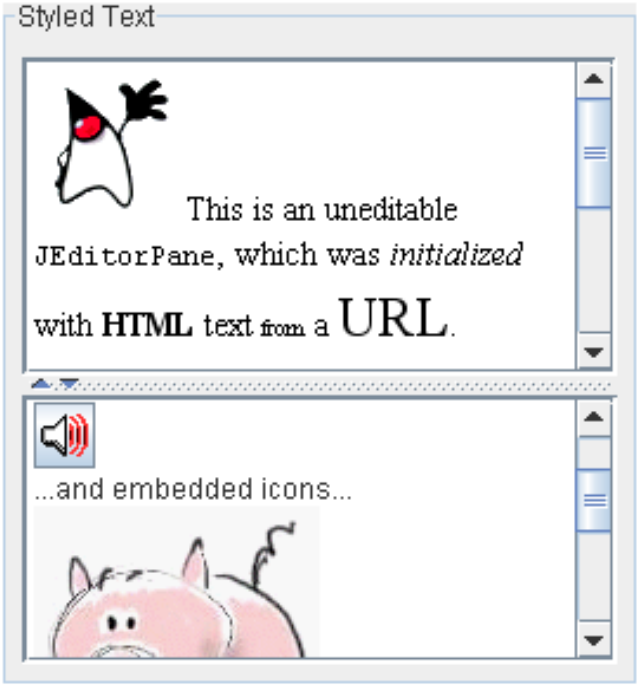
SWING Components

Interactive Displays of Highly Formatted Information

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

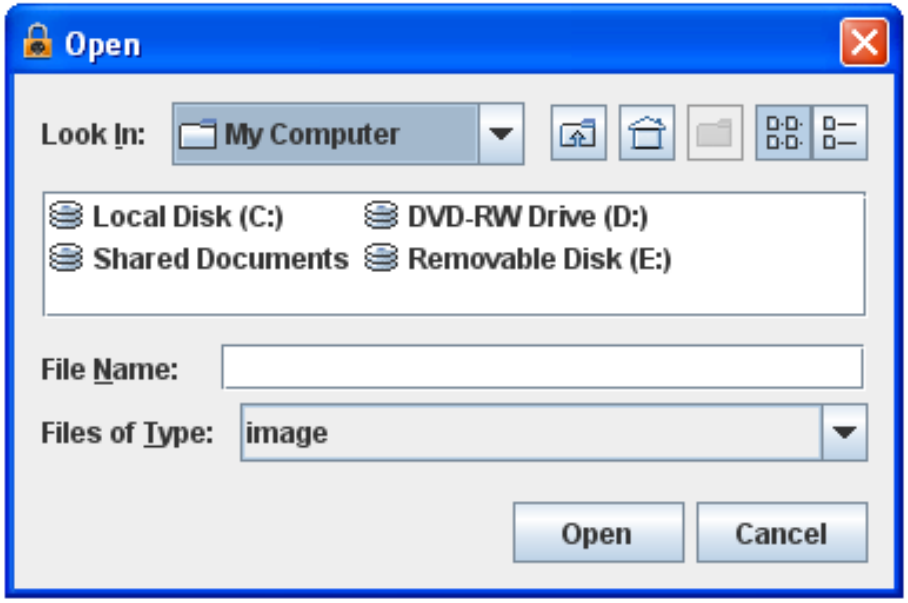


JColorChooser



JEditorPane and JTextPane

SWING Components



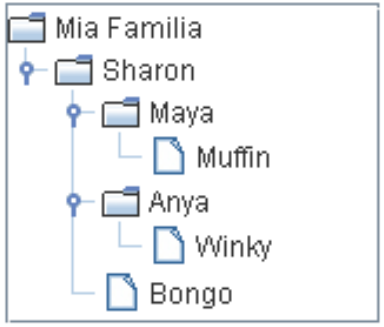
JFileChooser

Host	User	Password	Last Modified
Biocca Games	Freddy	!#asf6Awwzb	Mar 16, 2006
zabble	ichabod	Tazb!34$fZ	Mar 6, 2006
Sun Developer	fraz@hotmail.co...	AasW541!fbZ	Feb 22, 2006
Heirloom Seeds	shams@gmail....	bkz[ADF78!	Jul 29, 2005
Pacific Zoo Shop	seal@hotmail.c...	vbAf124%z	Feb 22, 2006

JTable

This is an editable JTextArea. A text area is a "plain" text component, which means that although it can display text in any font, all of the text is in the same font.

JTextArea

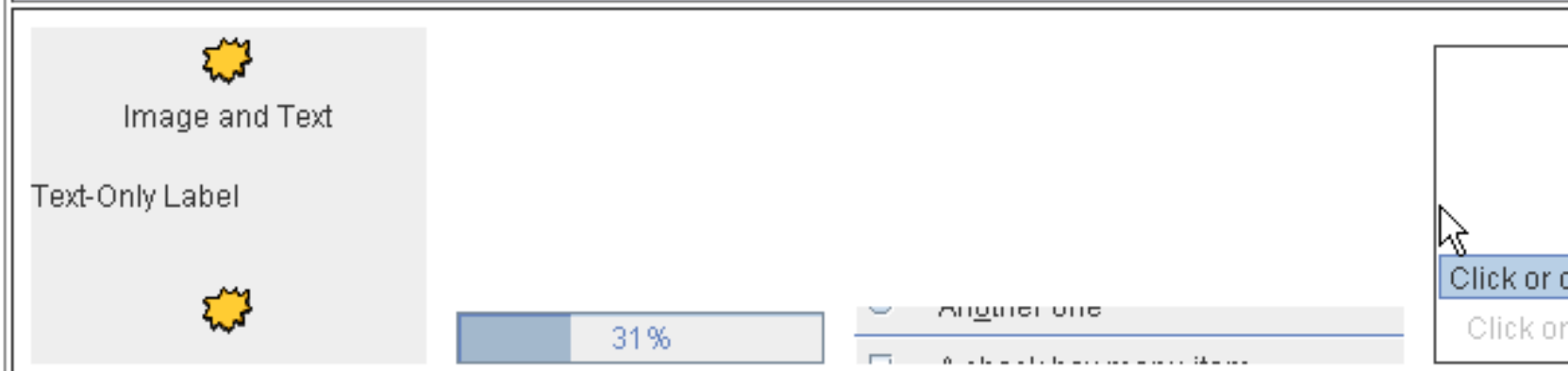


JTree

SWING Components

Uneditable Information Displays

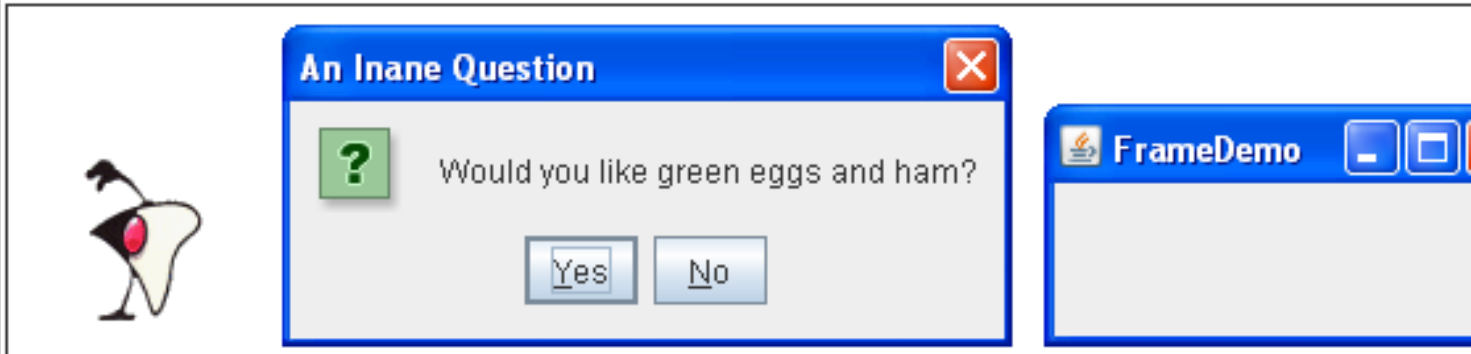
These components exist solely to give the user information.



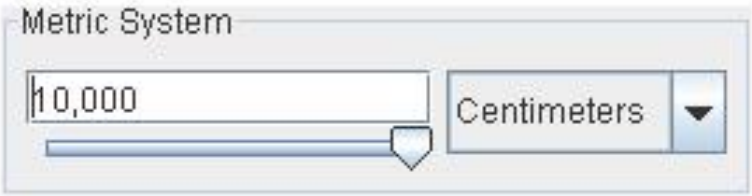
JLabel JProgressBar JSeparator

Top-Level Containers

At least one of these components must be present in any Swing application.



JApplet JDialog JFrame



JPanel

JScrollPane

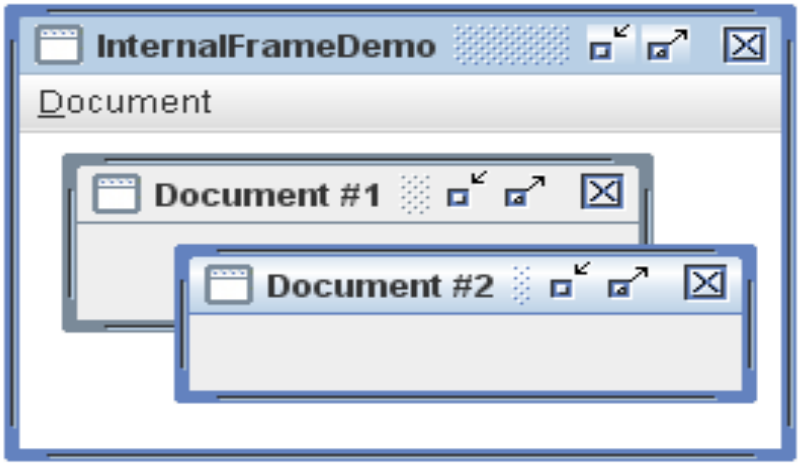
JSplitPane

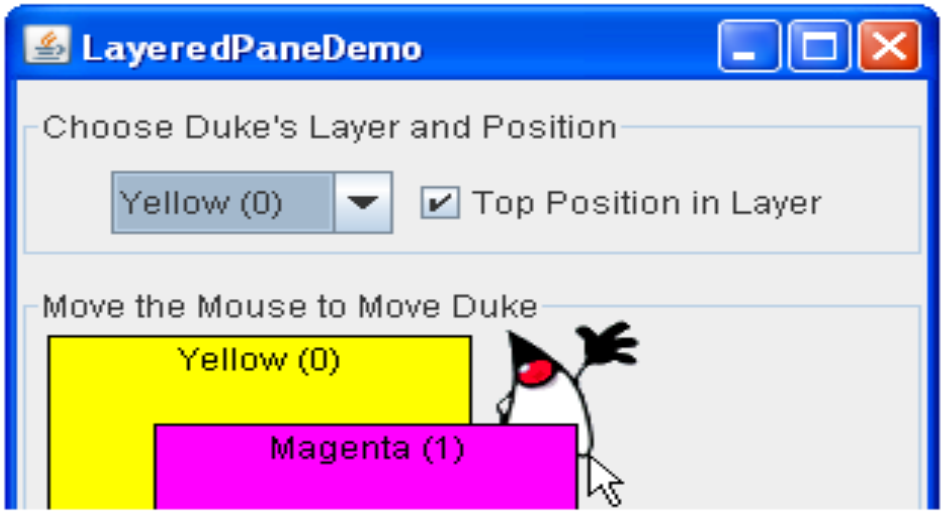
JTabbedPane

SWING Components

Special-Purpose Containers

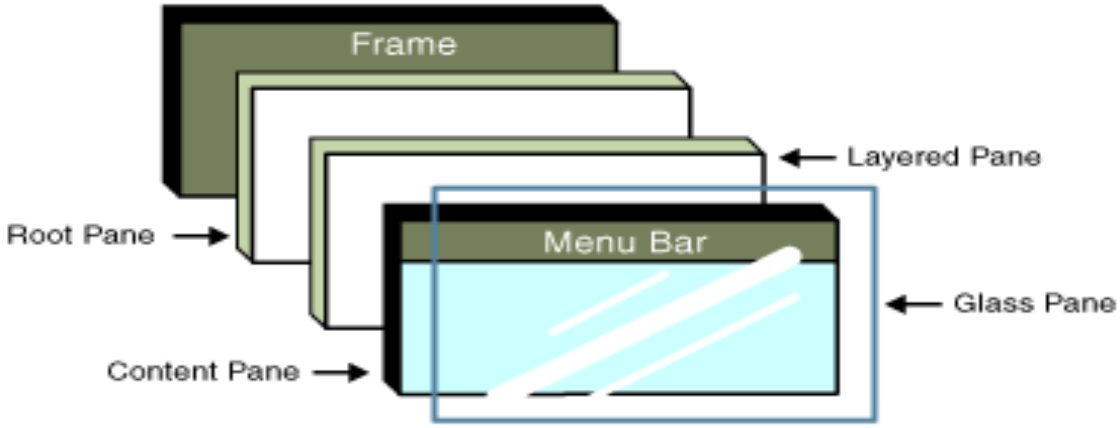
These special-purpose containers play specific roles in the UI.





JInternalFrame

JLayeredPane



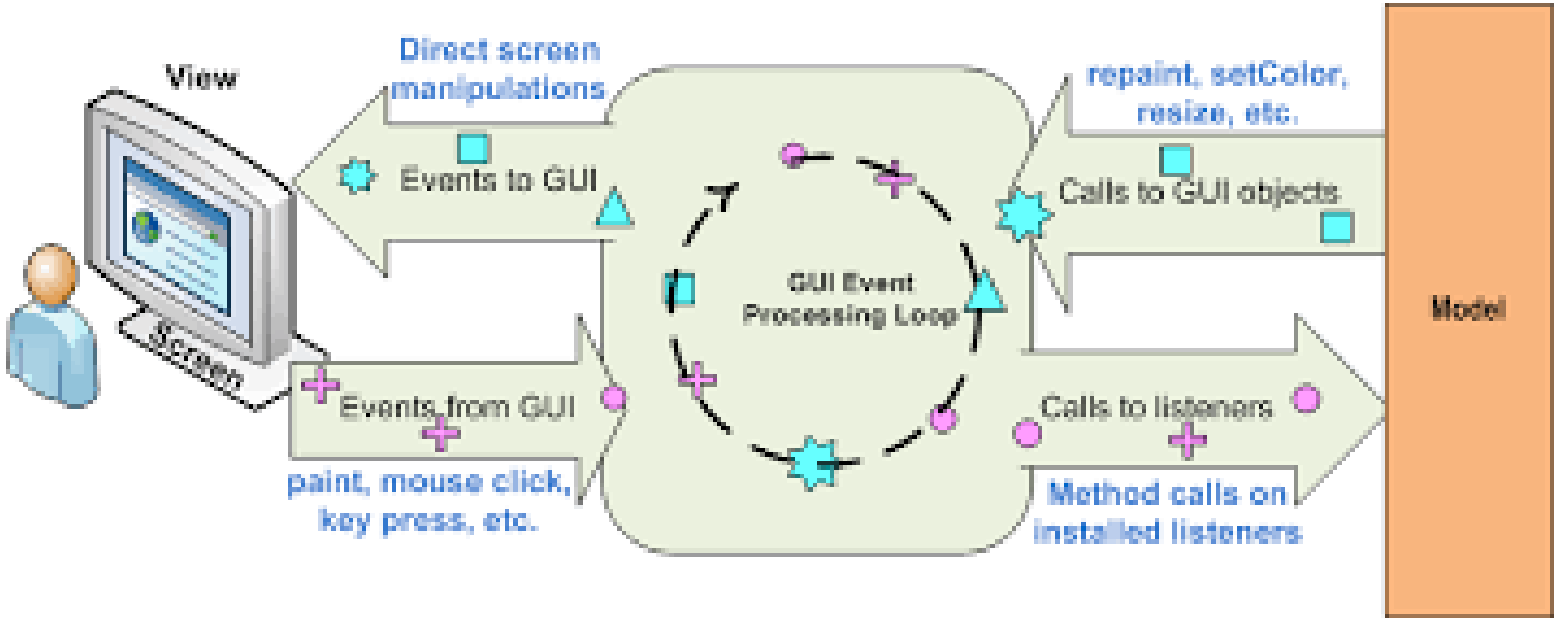
Root pane

Simple GUI using SWING

```
import javax.swing.*;
class SimpleGui {
    public static void main(String args[]){
        JFrame frame = new JFrame("Simple GUI");

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300,300);


        JButton button = new JButton("Press");


        // Adds Button to content pane of frame

        frame.getContentPane().add(button);



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

Event Handling

- Event is generated with clicking of button, typing of key, item selected from list, timer goes off etc
- Delegation Event Model - Swing's event handling approach
  - Source generates an event (derived from **java.util.EventObject** and describing state change in source) and sends to one or more listeners
  - Listener waits till the event is received. When received, it processes the event and then returns. Action events defined under **ActionListener** interface
  - A listener must register (using **addKeyListener()**, **addMouseMotionListener()** etc.) with a source for receiving event notifications
  - Advantage: Application logic (processing events) completely separated from UI logic (generating events), i.e. UI element "delegate" event processing to separate code



Simple implementation of Listener

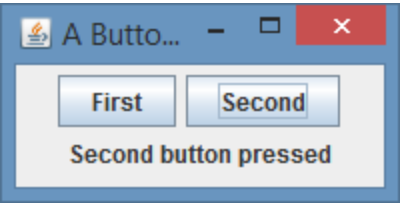
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class ButtonDemo implements ActionListener {
        JLabel jlab;
        ButtonDemo() {
                JFrame jfrm=new JFrame("A Button Example"); //Create new JFrame container
                jfrm.setSize(200, 100); //Set frame's initial size
                jfrm.setLayout( new FlowLayout() ); //Specify FlowLayout for Layout Manager
                jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //Terminate when user closes
application
                JButton jbtnFirst = new JButton("First"); //Create push buttons
                JButton jbtnSecond = new JButton("Second");
                jbtnFirst.addActionListener(this); //Set action listners for buttons
                jbtnSecond.addActionListener(this);
                jfrm.add(jbtnFirst); //Add buttons to content pane
                jfrm.add(jbtnSecond);
                jlab = new JLabel("Press a button.."); // Create text-based label
                jfrm.add(jlab); // Add label to frame
                jfrm.setVisible(true); //Display frame
        }
```

Simple implementation of Listener (contd)

```
        public void actionPerformed(ActionEvent ae) {     //  Button processing
                if (ae.getActionCommand().equals("First"))
                        jlab.setText("First button pressed");
                else
                        jlab.setText("Second button pressed");
        }

        public static void main(String[] args) {
                new ButtonDemo();
        }
}
```

Run: java ButtonDemo *(neither appletviewer nor html file required)*

Output:



Note:
- You may use null layout for using absolute positions, e.g. setBound(x, y, width, height)
- You may create create JTextField object and use getText / setText for accessing / populating field values.

***JTextField**: Enables user to enter a line of text. Inherits the abstract class **JTextComponent**. Usage: JTextField(int cols), cols specifies width of the text field in columns. Text can be set by using setText().*