

Java Programming II

Java GUI

Contents

- ◆ **Overview of the AWT**
 - Canvas
 - Button, TextField, List
 - Menu, MenuBar and MenuItem
 - Panel
- ◆ **Layout Managers**
- ◆ **Events and Delegation Model**
- ◆ **Swing**
 - Creating New Window Frame
 - Dialogs and File Chooser

Abstract Window Toolkit(AWT)

- ◆ The *Abstract Window Toolkit* (AWT) and *Swing* provide standard components to build a graphical user interface (GUI)
- ◆ The GUI enables interaction between the user and the program by using the mouse, keyboard, or another input device
- ◆ The AWT provides a mechanism to paint different shapes on the screen (e.g., lines, rectangles, text, etc.), and create different elements on a screen (buttons, lists, and others)

Example: GUI

The image displays three Java GUI applications. The top-left window is 'Stylepad', a text editor showing the title 'ALICE'S ADVENTURES IN WONDERLAND' by Lewis Carroll. The bottom-left window is 'Java 2D(TM) Demo', which features a 'Global Control' panel with checkboxes for Anti-Aliasing, Rendering, Texture, and AlphaCompositing, as well as a 'Texture Chooser' and a 'Performance' monitor. The right window is 'SwingSet2', which displays a 'Table Demo' with a table of user preferences and a 'Printing' section.

Stylepad

File Edit Color Font Debug

ALICE'S ADVENTURES IN WONDERLAND

Lewis Carroll

Java 2D(TM) Demo

File Options

Java2D Arcs_Curves Clipping Colors Composite Fonts Images Lines Mix Paint Paths T

Width = 20.0 Minored Join

SwingSet2

File Look & Feel Themes Options

Table Demo Source Code

☒ Reordering allowed ☒ Row selection

☒ Horiz. Lines ☐ Column selection

☒ Vert. Lines

Inter-cell spacing: [Slider]

Row height: [Slider]

Selection mode: Multiple ranges

Autosize mode: Subsequent columns

Printing

Header: jTable Printing

Footer: Page {0}

☒ Fit Width Print

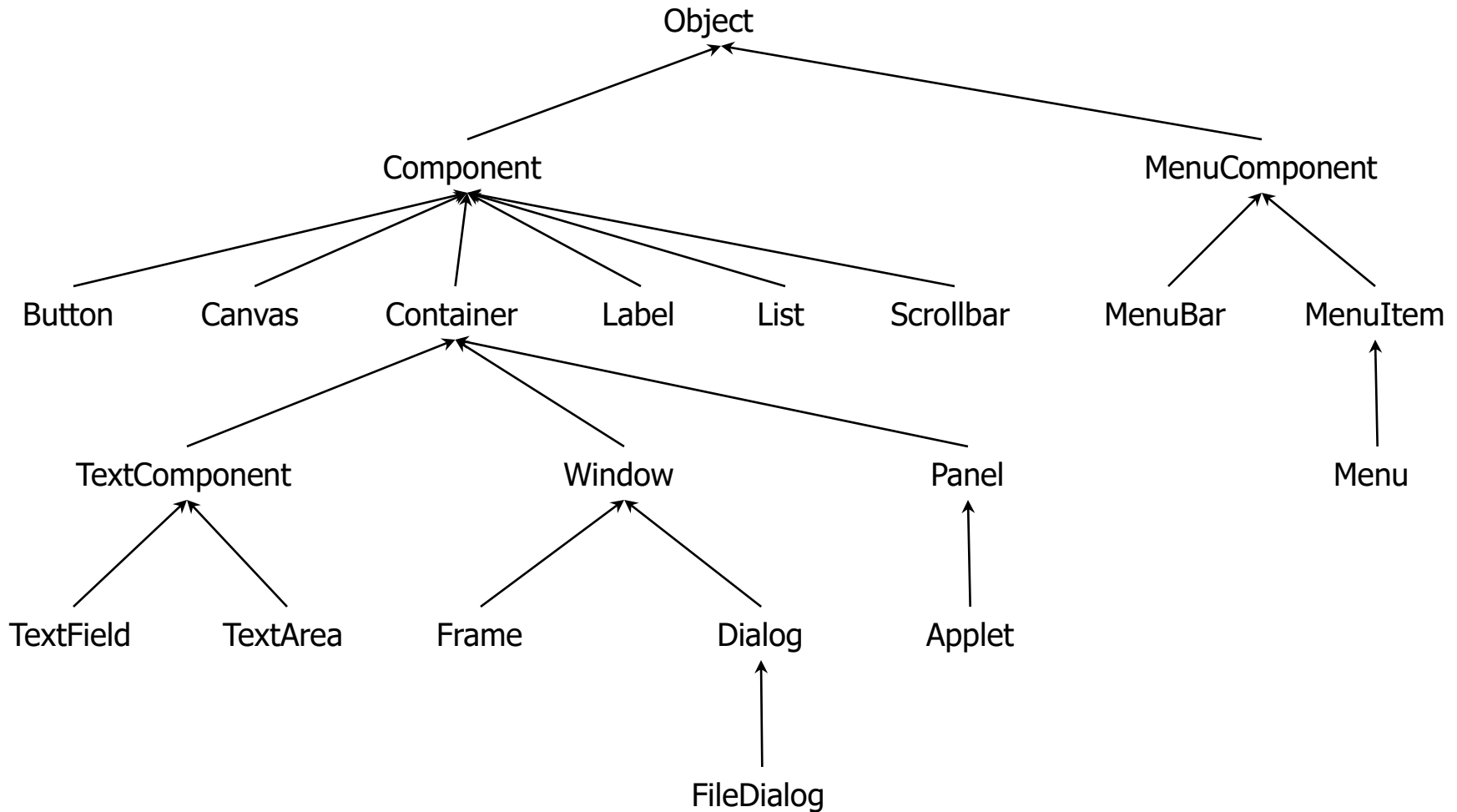
First Name	Last Name	Favorite Color	Favorite Movie	Favorite Number	Favorite Food
Mike	Albers	Green	Brazil	44	[Image]
Mark	Andrews	Blue	Curse of the Demon	3	[Image]
Brian	Beck	Black	The Blues Brothers	2.718	[Image]
Lara	Bunni	Red	Airplane (the whol...	15	[Image]
Roger	Brinkley	Blue	The Man Who Kne...	13	[Image]
Brent	Christian	Black	Blade Runner (Dir...	23	[Image]
Mark	Davidson	Dark Green	Brazil	27	[Image]
Jeff	Dinkins	Blue	The Lady Vanishes	8	[Image]
Ewan	Dinkins	Yellow	A Bug's Life	2	[Image]
Amy	Fowler	Violet	Reservoir Dogs	3	[Image]
Hania	Gajewska	Purple	Jules et Jim	5	[Image]

Press Shift-F10 to activate popup menu

Performance

Caps 32 ms
Dash 32 ms
Join 32 ms
LineAnim 27.0 fps

AWT Class Hierarchy



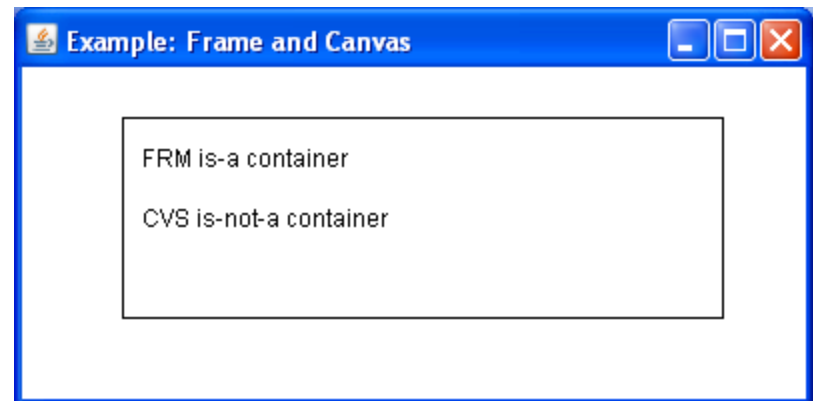
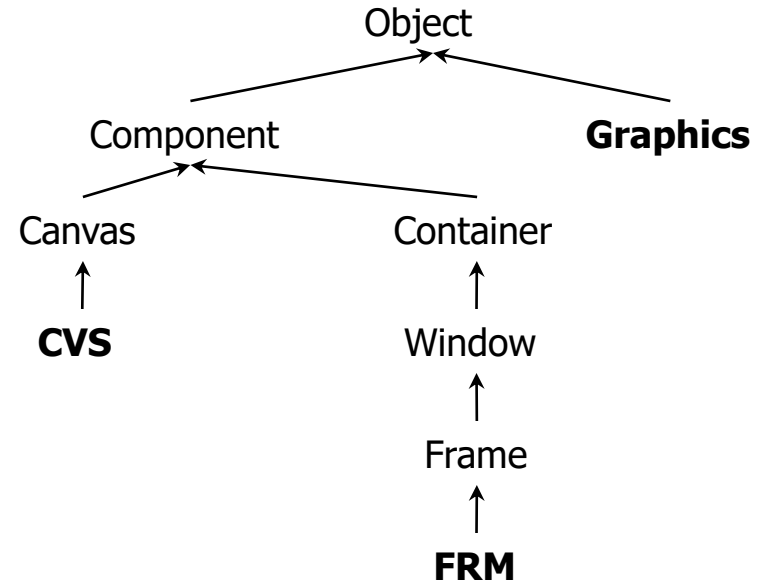
Component

- ◆ Component is an abstract class
- ◆ A component is an object having a graphical representation
- ◆ Components can be displayed on the screen
- ◆ Components allow the user to interact with the program

Example: Frame and Canvas

```
public class FRM extends Frame {  
  
    public FRM()  
    {  
        super("Example: Frame and Canvas");  
  
        add(new CVS()); // add a canvas to paint  
        setSize(400,200);  
    }  
  
    public static void main(String[] args)  
    {  
        new FRM().setVisible(true);  
    }  
  
    class CVS extends Canvas {  
        // paint this canvas  
        public void paint(final Graphics g)  
        {  
            g.drawRect(50,25,300,100);  
            g.drawString("FRM is-a container",60,50);  
            g.drawString("CVS is-not-a container",60,80);  
        }  
    }  
}
```

- ◆ *A Canvas is used to draw some shapes on it using the Graphics. It has the paint method.*
- ◆ CVS is an inner class
- ◆ A Graphics object is used to draw shapes on the canvas
- ◆ FRM is a container – it contains a CVS object



Button

- ◆ A *Button* is a component that simulates the appearance of a push button
- ◆ When the user presses the mouse inside a button an *ActionEvent* is generated

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

```
class BTN extends Frame {
```

```
    BTN()  
    {
```

```
        super("Example: Button");
```

```
        final Button b = new Button("Press me!");
```

Anonymous
Class

```
        b.addActionListener(new ActionListener() {
```

```
            // the event handler
```

```
            public void actionPerformed(ActionEvent ae) {
```

```
                b.setLabel("Thank you!");
```

```
            }
```

```
        });
```

```
        add(b);
```

```
        setSize(200,100);
```

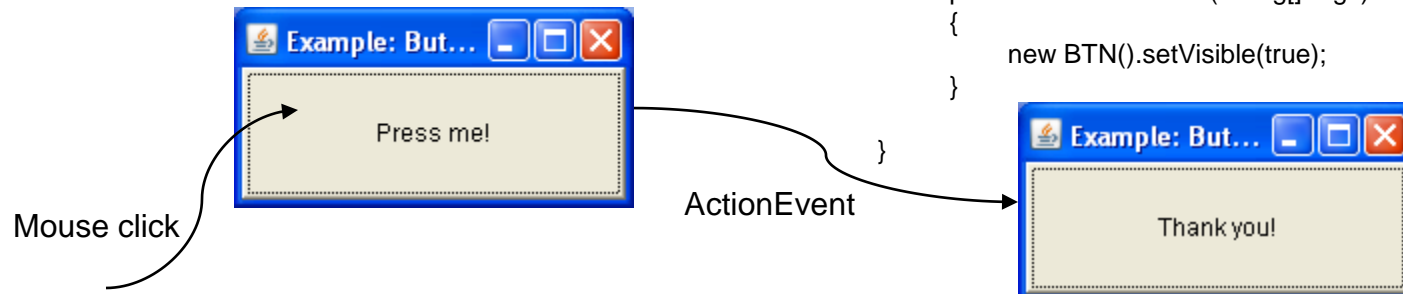
```
    }
```

```
    public static void main(String[] args)
```

```
    {
```

```
        new BTN().setVisible(true);
```

```
    }
```



Label and TextField

- ◆ A *Label* displays a string that cannot be changed by a user
- ◆ A *TextField* allows a user to enter or edit one line of text
- ◆ A *FlowLayout* arranges components :
 - in a directional flow (left-to-right, or right-to-left)
 - horizontally until no more components fit on the same line

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

class LTF extends Frame {

    LTF()
    {
        super("Example: Label & TextField");

        setLayout(new FlowLayout(FlowLayout.LEFT));
        setResizable(false);
        add(new Label("Cannot edit!"));

        final TextField tf = new TextField("Edit me!",37);

        tf.addTextListener(new TextListener() {

            public void textValueChanged(TextEvent te)
            {
                System.out.println(te paramString());
            }

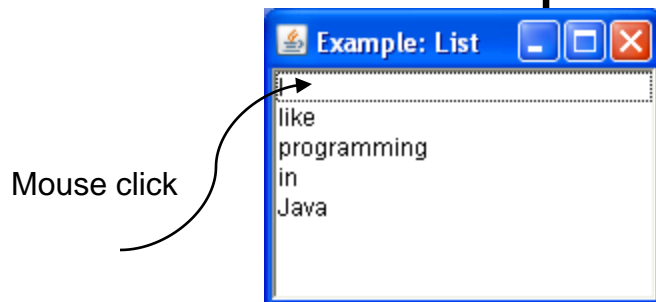
        });
        add(tf);
        setSize(400,100);
    }

    public static void main(String[] args)
    {
        new LTF().setVisible(true);
    }
}
```



List

- ◆ The *List* component presents the user with a scrolling list of text items
- ◆ It can be set up so that the user can choose either one item or multiple items



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

class LST extends Frame {

    LST()
    {
        super("Example: List");

        final List l = new List();

        l.add("I");
        l.add("like");
        l.add("programming");
        l.add("in");
        l.add("Java");
        l.addItemListener(new ItemListener() {

            public void itemStateChanged(ItemEvent ie)
            {
                System.out.println(ie paramString());
            }

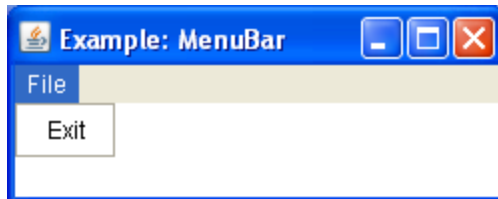
        });
        add(l);
        setSize(200,150);
    }

    public static void main(String[] args)
    {
        new LST().setVisible(true);
    }
}
```

```
C:\WINDOWS\system32\cmd.exe /c java LST
ITEM_STATE_CHANGED,item=1,stateChange=SELECTED
ITEM_STATE_CHANGED,item=0,stateChange=SELECTED
ITEM_STATE_CHANGED,item=2,stateChange=SELECTED
ITEM_STATE_CHANGED,item=3,stateChange=SELECTED
```

Menu, MenuBar and MenuItem

- ◆ A frame may contain a menu bar with options (i.e. items)
- ◆ When the mouse is clicked on an option a drop down menu appears
- ◆ Each menu consists of one or more menu items



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

class MNB extends Frame {

    MNB()
    {
        super("Example: MenuBar");

        final MenuBar mb = new MenuBar();

        setMenuBar(mb);

        final Menu m = new Menu("File");
        MenuItem mi;

        mi = new MenuItem("Exit");
        mi.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent ae)
            {
                System.exit(0);
            }

        });
        m.add(mi);
        mb.add(m);
        setSize(250,100);
    }

    public static void main(String[] args)
    {
        new MNB().setVisible(true);
    }
}
```

Panel

- ◆ Panel is the simplest container class
- ◆ A panel provides space in which an application can attach any other component, including other panels
- ◆ The default layout manager for a panel is the FlowLayout manager

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

class PNL extends Frame {

    PNL()
    {
        super("Example: Panel");

        final Panel p = new Panel();

        p.add(new Button("1"));
        p.add(new Button("2"));
        p.add(new Button("3"));
        add(p);
        setSize(250,100);
    }

    public static void main(String[] args)
    {
        new PNL().setVisible(true);
    }
}
```



Layout Managers

- ◆ A *layout manager* helps in arranging the components in a container
- ◆ Each layout manager:
 - Encapsulates an algorithm for positioning and sizing of components
 - Automatically calculates the coordinates of each component it manages
 - If a container is resized, the layout manager readjusts the placement of the components

BorderLayout

- ◆ Allows placing of components by using the geographic terms:
 - CENTER
 - EAST
 - NORTH
 - SOUTH
 - WEST
- ◆ The components are placed around the edges
- ◆ The component in the center uses the remaining space

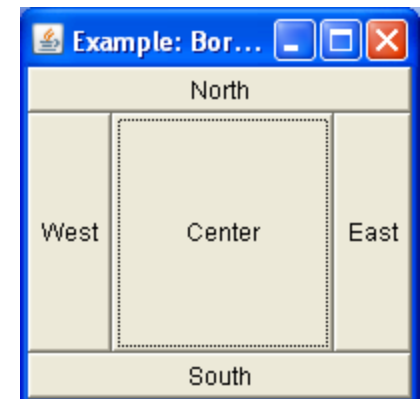
```
import java.awt.*;
import java.awt.event.*;

class BLM extends Frame {

    BLM()
    {
        super("Example: BorderLayout");

        setLayout(new BorderLayout());
        add(new Button("Center"),BorderLayout.CENTER);
        add(new Button("East"),BorderLayout.EAST);
        add(new Button("North"),BorderLayout.NORTH);
        add(new Button("South"),BorderLayout.SOUTH);
        add(new Button("West"),BorderLayout.WEST);
        setSize(200,200);
    }

    public static void main(String[] args)
    {
        new BLM().setVisible(true);
    }
}
```



GridLayout

- ◆ Automatically arranges components in a grid of rows and columns
- ◆ The container is divided into equal-sized cells, and one component is placed in each cell



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

class GLM extends Frame {

    GLM()
    {
        super("Example: GridLayout");

        setLayout(new GridLayout(2,2));
        add(new Button("1,1"));
        add(new Button("1,2"));
        add(new Button("2,1"));
        add(new Button("2,2"));
        setSize(250,100);
    }

    public static void main(String[] args)
    {
        new GLM().setVisible(true);
    }
}
```

Using the ActionListener

◆ Stages for Event Handling by ActionListener

- First, import event class

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

- Define an overriding class of event type (implements ActionListener)

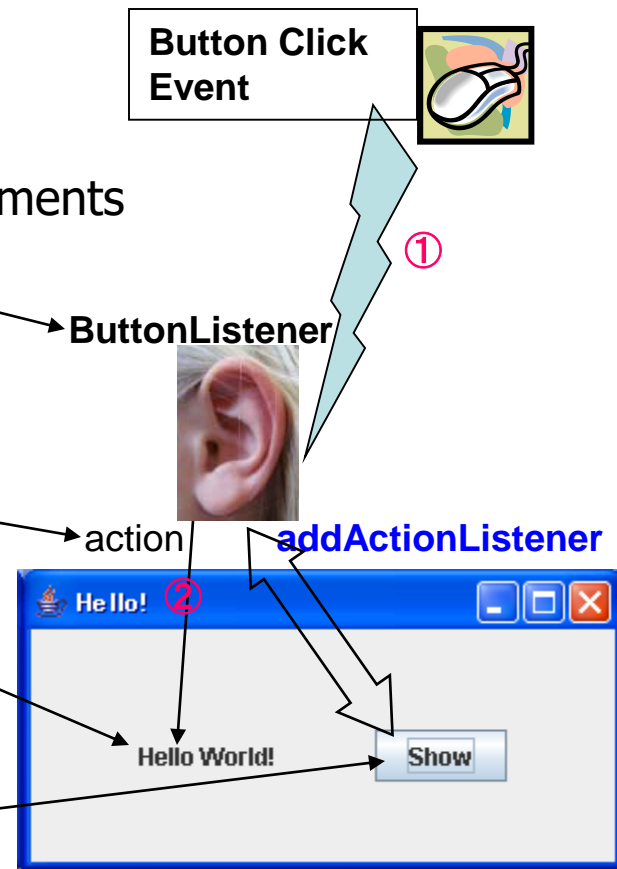
```
Class ButtonListener implements ActionListener {  
    public void actionPerformed(ActionEvent e) {  
        // Write what to be done. . .  
        label.setText("Hello World!");  
    }  
}
```

- Create an event listener object

```
ButtonListener bt = new ButtonListener();
```

- Register the event listener object

```
b1 = new Button("Show");  
b1.addActionListener(bt);
```



A Hello Example Using Button Listener

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

public class HelloAWT extends Frame { // Using Frame
    Label contents;
    Button dispbutton;

    public HelloAWT() { // Constructor
        setLayout(new FlowLayout(FlowLayout.CENTER, 50, 50));

        contents = new Label("        "); // Create Label object
        add(contents); // Add the label to this Frame

        dispbutton = new Button("Show"); // Create Button object
        dispbutton.addActionListener(new DispButtonListener()); // Add Event Listener
        add(dispbutton); // Add the button object to this Frame
    }

    class DispButtonListener implements ActionListener { // Event Listener
        public void actionPerformed(ActionEvent e) { // What to do when the button is
            clicked
            contents.setText("Hello World!");
        }
    }

    public static void main (String[] args) {
        HelloAWT f = new HelloAWT(); // Create Hello GUI
        f.setTitle("Hello!");
        f.setSize(400,150);
        f.setVisible(true);
    }
} // end of "HelloAWT.java"
```



Run: Java HelloAWT

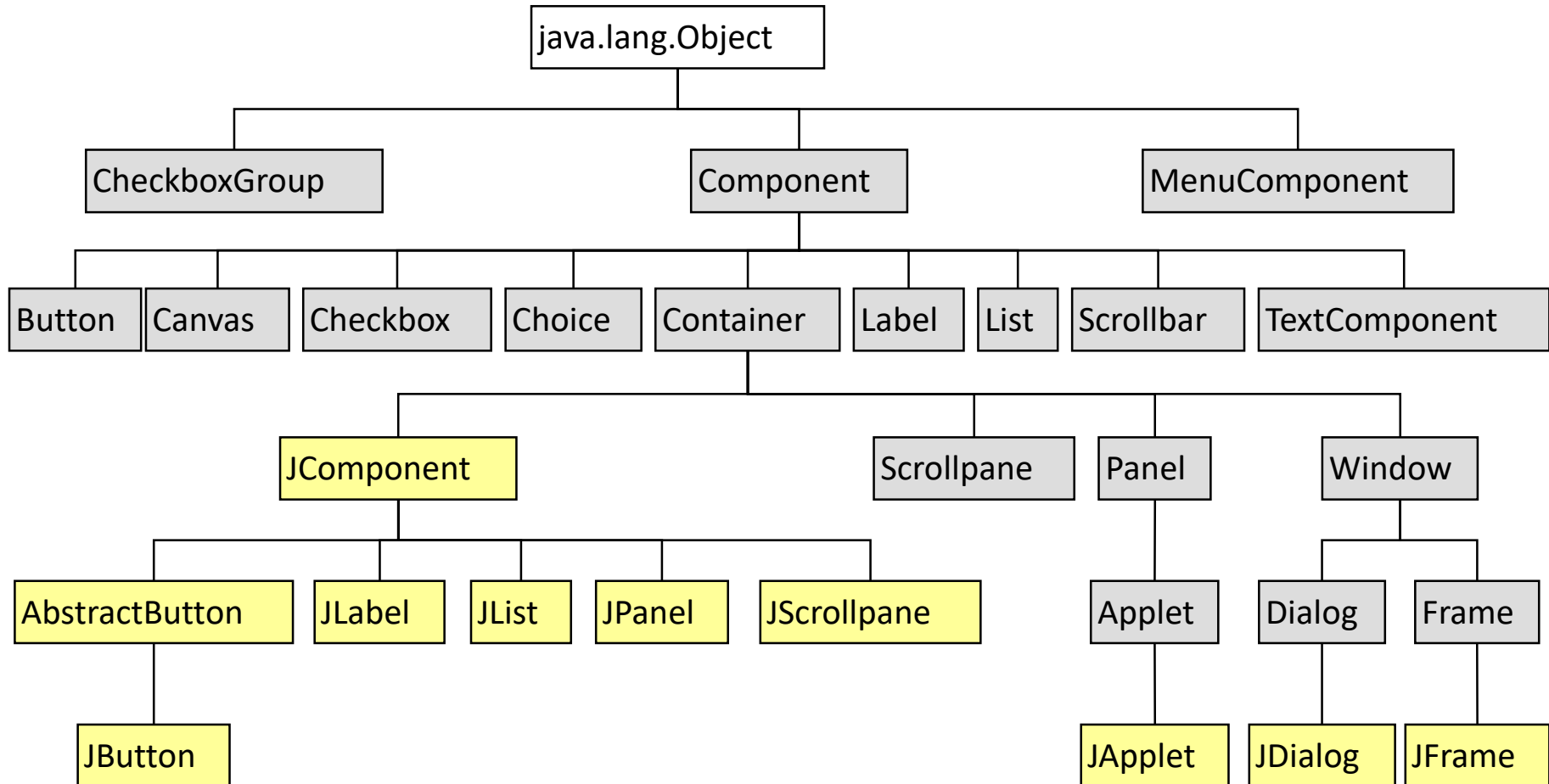
```
// Can be replaced by anonymous class

dispbutton.addActionListener(new
    ActionListener () {
        public void actionPerformed(ActionEvent e) {
            contents.setText("Hello Annoymus");
        }
    });
```

Swing

- ◆ Differences between AWT and Swing:
 - Swing components use no native code and they can be present on every platform
 - Typically, Swing components start their names with 'J'
 - Have capabilities beyond what equivalent AWT components can offer
 - Swing components need not be rectangular
 - Swing components can dynamically change their appearance (i.e. pluggable look-and-feel)

Partial AWT and Swing Class Hierarchy



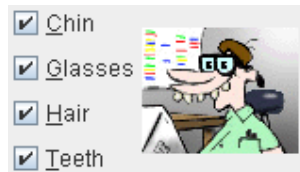
 java.awt.*

 javax.swing.*

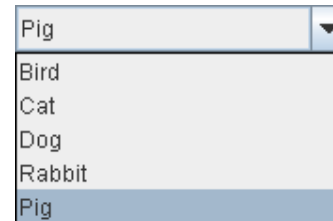
Swing Components (Java Look and Feel)



[JButton](#)



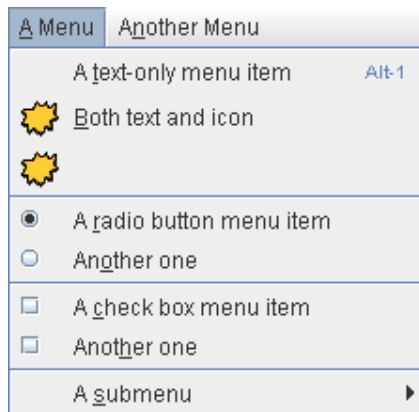
[JCheckBox](#)



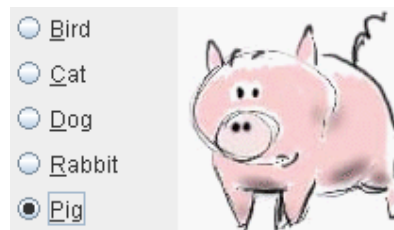
[JComboBox](#)



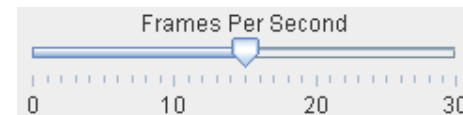
[JList](#)



[JMenu](#)



[JRadioButton](#)



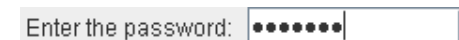
[JSlider](#)



[JSpinner](#)



[JTextField](#)



[JPasswordField](#)

Example: Hello World

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

class HLW extends JFrame {

    HLW()
    {
        super("Example: Swing GUI");

        final JButton b = new JButton("Show message!");

        b.addActionListener(new HLWButtonListener(b));
        add(b);
        setSize(250,100);
    }

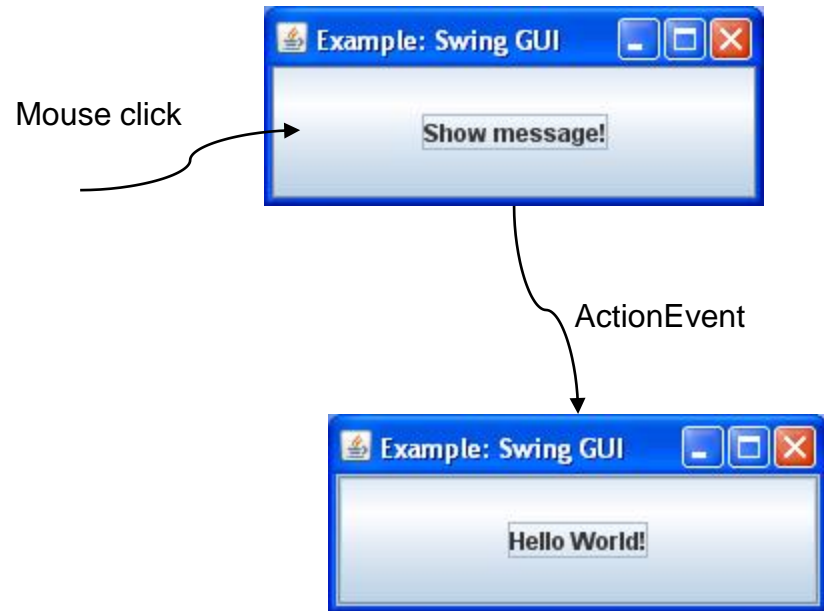
    public static void main(String[] args)
    {
        new HLW().setVisible(true);
    }
}

class HLWButtonListener implements ActionListener {

    private JButton jb;

    HLWButtonListener(JButton b)
    {
        jb = b;
    }

    public void actionPerformed(ActionEvent e)
    {
        jb.setText("Hello World!");
    }
}
```



Creating New Window Frame

```
// Dialog Box
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class CreatNewFrame extends JFrame
{
    JLabel client_title;
    JButton create_button;

    public CreatNewFrame() {

        getContentPane().setLayout(new GridLayout(1,0));

        create_button = new JButton("Create");
        create_button.addActionListener(new ButtonListener());
        getContentPane().add(create_button);
    }

    class ButtonListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            NewFrame nf = new NewFrame();

            nf.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e) {System.exit(0);}
            });

            nf.setTitle("New Window Frame");
            nf.setSize(200,150);
            nf.setVisible(true);
        }
    }
}
```

```
public static void main (String args[]) {
    CreatNewFrame f = new CreatNewFrame();
    f.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e)
        {System.exit(0);}
    });

    f.setTitle("Create New Frame");
    f.setSize(200,150);
    f.setVisible(true);
} // end of CreatNewFrame

class NewFrame extends JFrame {
    JLabel label;

    public NewFrame() {
        getContentPane().setLayout(new FlowLayout());

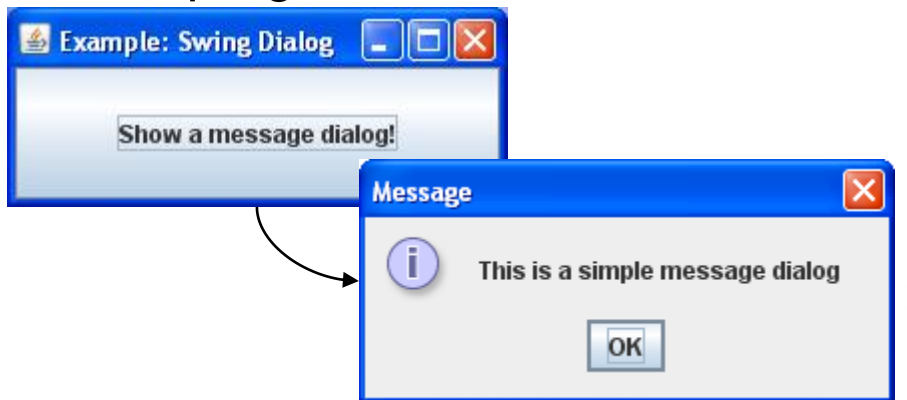
        label = new JLabel("Another New Frame");
        getContentPane().add(label);
    } // NewFrame constructor

} // end of NewFrame class
```



Dialogs

- ◆ A *dialog* is a special window to convey a message or provides a special function
- ◆ Every dialog is dependent on a frame – when that frame is destroyed, so are its dependent dialogs
- ◆ A *modal* dialog blocks user input to all other windows in the program



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

class DLG extends JFrame {

    DLG()
    {
        super("Example: Swing Dialog");

        final JFrame jf = this;
        final JButton jb = new JButton("Show a message
dialog!");

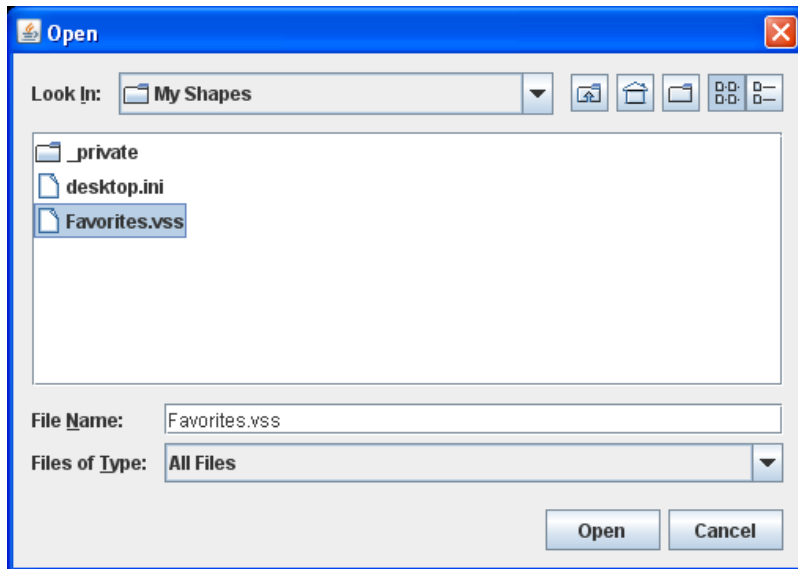
        jb.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent ae) {
                JOptionPane.showMessageDialog(jf,"This is a
simple message dialog");
            }
        });
        add(jb);
        setSize(250,100);
    }

    public static void main(String[] args)
    {
        new DLG().setVisible(true);
    }
}
```

FileChooser

- ◆ File choosers provide a GUI for navigating the file system or open a file
- ◆ To display a file chooser, use the *JFileChooser* API to show a modal dialog containing the file chooser



```
import javax.swing.*;

class FCH extends JFrame {

    final JLabel jl = new JLabel();

    FCH()
    {
        super("Example: Swing FileChooser");

        add(jl);
        setSize(300,50);
    }

    public static void main(String[] args)
    {
        final FCH fch = new FCH();
        final JFileChooser jfc = new JFileChooser();

        fch.setVisible(true);

        final int val = jfc.showOpenDialog(fch);

        if(val == JFileChooser.APPROVE_OPTION)
            fch.jl.setText("You chose to open this file: " +
                jfc.getSelectedFile().getName());
    }
}
```

