# Java GUI

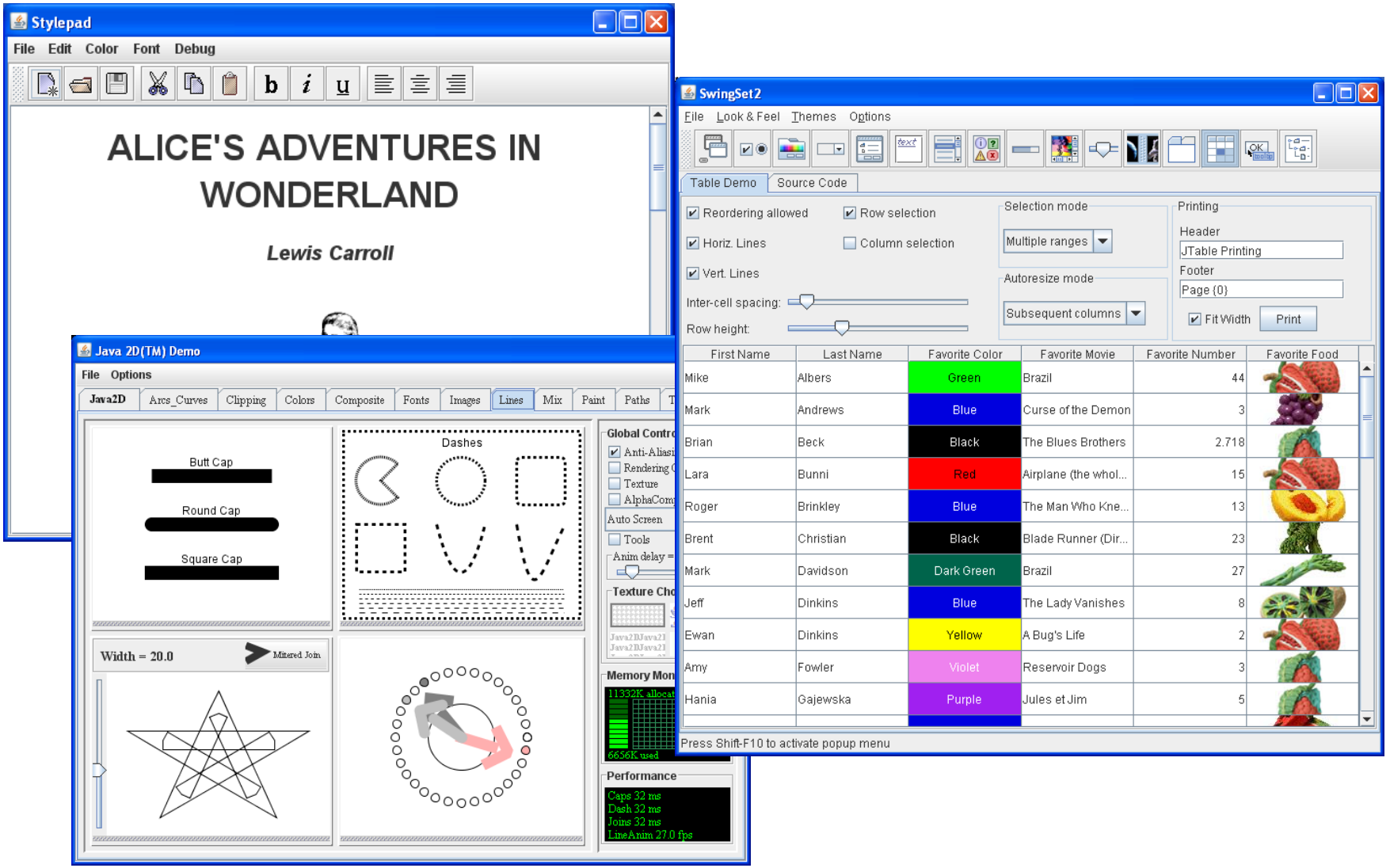
# Contents

- **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**

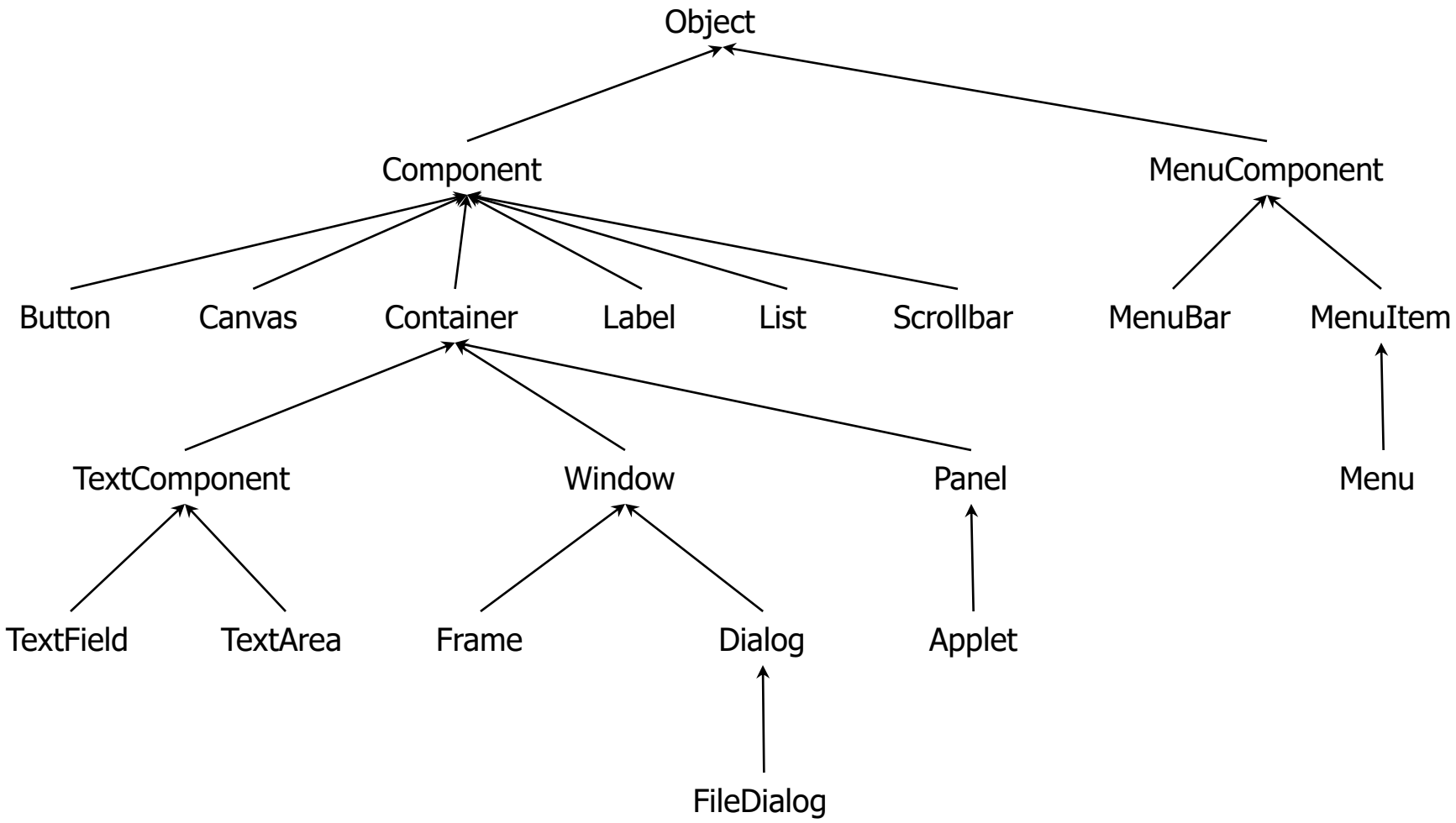
# Abstract Window Toolkit(AWT)

- The graphical user interface (GUI) enables interaction between the user and the program by using the mouse,  keyboard, or another input device
- The *Abstract Window Toolkit* (AWT) and *Swing* provide standard components to build a GUI
- 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


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
Butt Cap
Round Cap
Square Cap
Dashes
Width = 20.0
Mitered Join
Global Contro
Anti-Aliasi
Rendering
Texture
AlphaComp
Auto Screen
Tools
Anim delay =
Texture Cho
Memory Mon
11332K allocat
6656K used
Performance
Caps 32 ms
Dash 32 ms
Joins 32 ms
LineAnim 27.0 fps
SwingSet2
File Look & Feel Themes Options
Table Demo Source Code
Reordering allowed
Row selection
Horiz. Lines
Column selection
Vert. Lines
Inter-cell spacing:
Row height:
Selection mode
Multiple ranges
Autoresize 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
Mark Andrews Blue Curse of the Demon 3
Brian Beck Black The Blues Brothers 2.718
Lara Bunni Red Airplane (the whol... 15
Roger Brinkley Blue The Man Who Kne... 13
Brent Christian Black Blade Runner (Dir... 23
Mark Davidson Dark Green Brazil 27
Jeff Dinkins Blue The Lady Vanishes 8
Ewan Dinkins Yellow A Bug's Life 2
Amy Fowler Violet Reservoir Dogs 3
Hania Gajewska Purple Jules et Jim 5
Press Shift-F10 to activate popup menu

# AWT Class Hierarchy

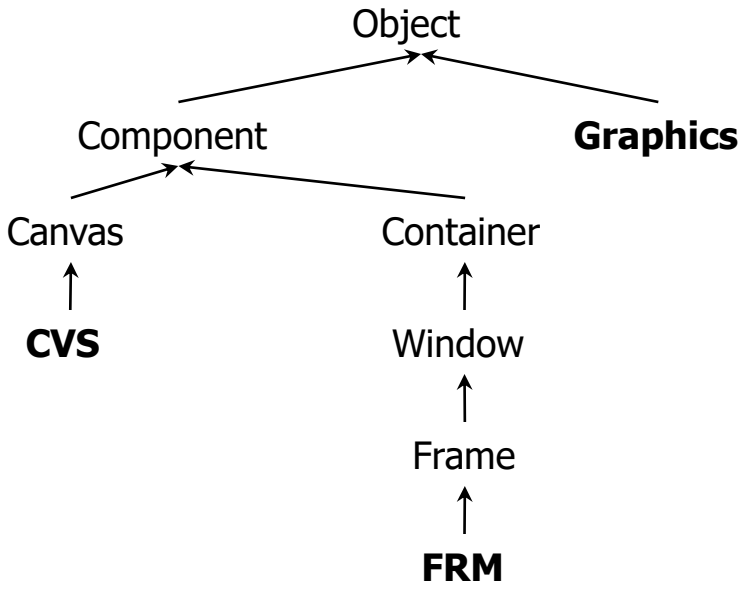


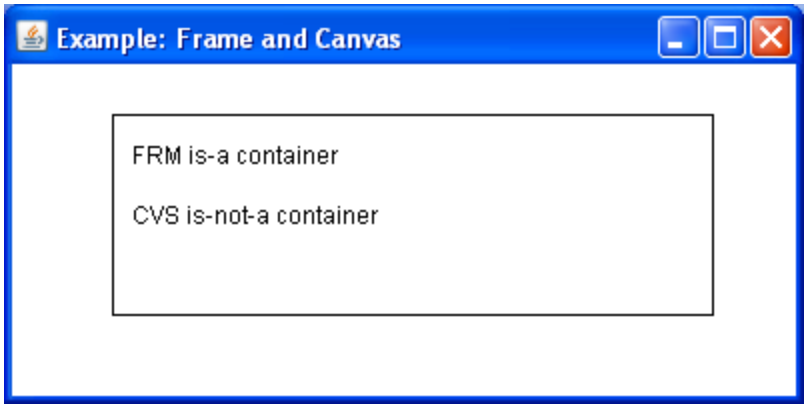
# Component-oriented

- 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

# 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);
                }

        }

}
```



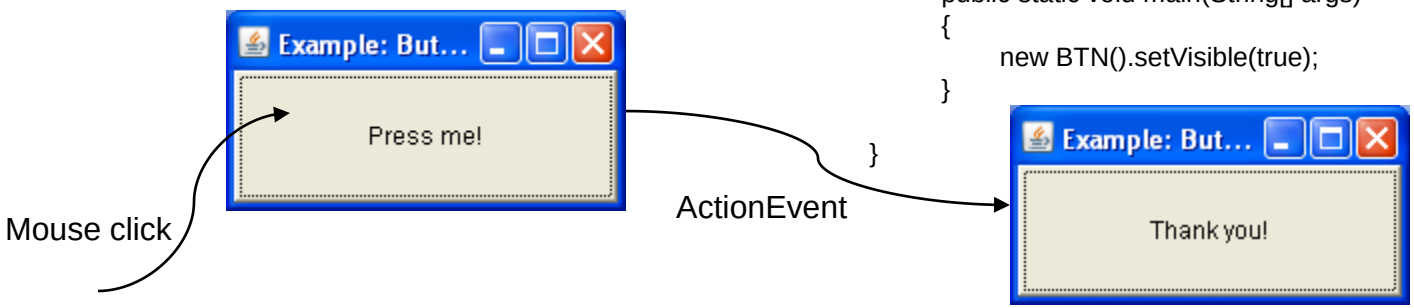

- *A Canvas is used to draw some shapes on it using the Graphics. It has the paint method.*
- *CVS* is an inner class
- A *Graphics* object is used to draw shapes on the canvas
- *FRM* is a container – it contains a CVS object

# 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!");

        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);
    }
}
```

Anonymous Class



# Label and TextField

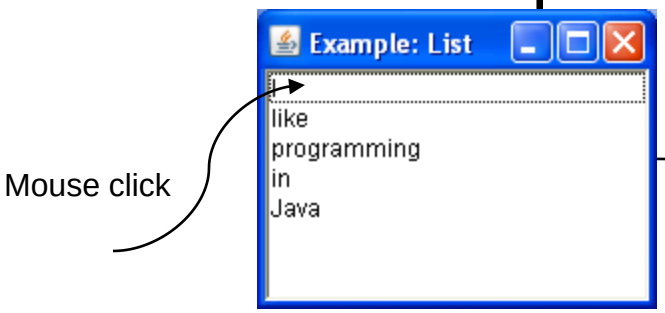
- A *Label* displays a string that cannot be changed by a user
- A *TextField* allows a user to enter or edit one line of text
- A *FlowLayout* arranges components :
  - in a directional flow (left-to-right, or right-to-left)
  - horizontally until no more components fit on the same line

```
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

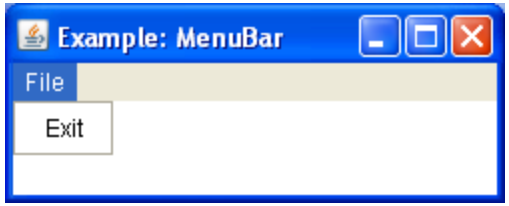
```
import java.awt.*;
import java.awt.event.*;

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

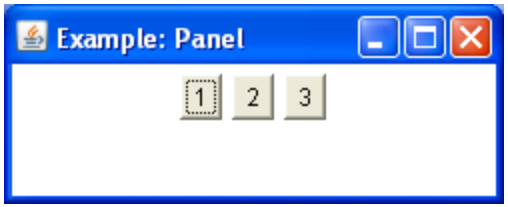



```
import java.awt.*;
import java.awt.event.*;

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);
        }

}
```



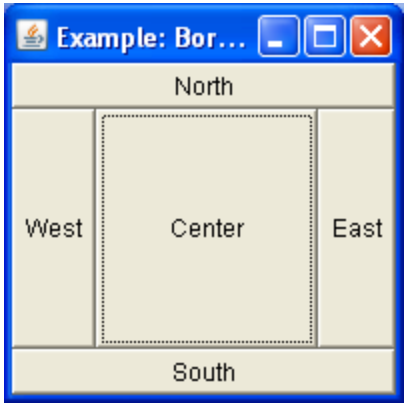
# Layout Managers

- 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!");
      }
}
```

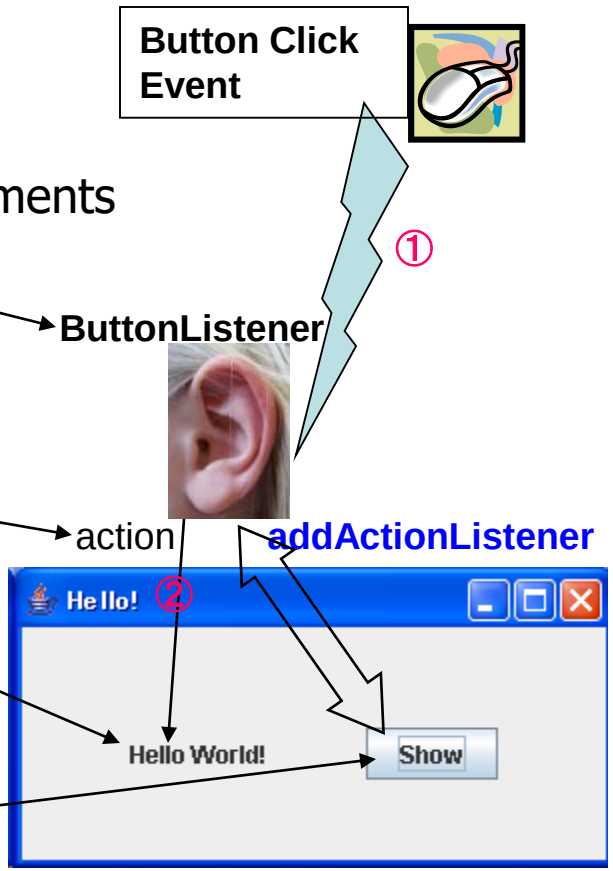
  - Create an event listener object

    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");
 }
} );
```

# Using the FocusListener

- FocusListener :
  - Focus events are fired whenever a component gains or loses the **keyboard focus**.
  - Define an overriding class of event type

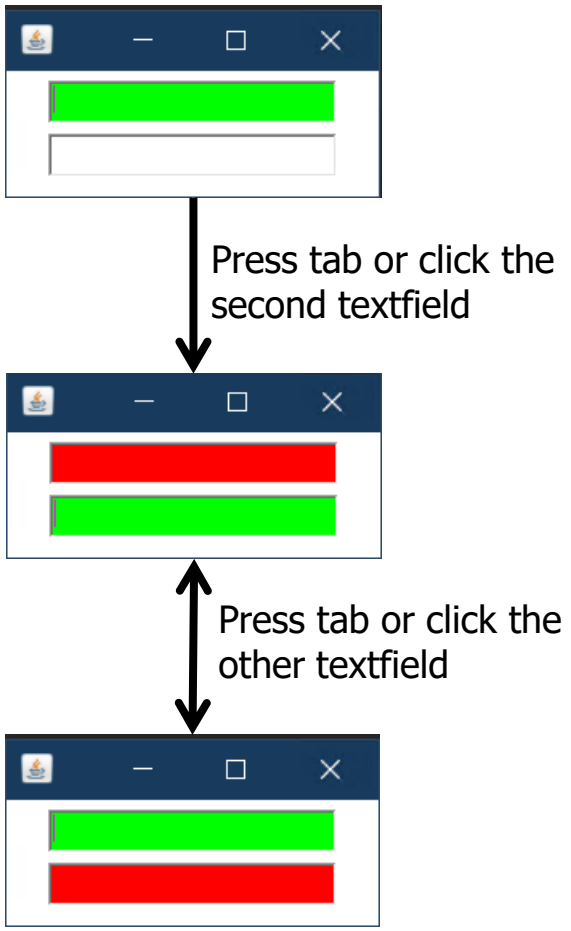
```
class myFocusListener implements FocusListener {
    @Override
    public void focusGained(FocusEvent e) {
    }
    @Override
    public void focusLost(FocusEvent e) {
    }
}
```

  - Register the event listener object

```
TextField textField = new TextField();
textField.addFocusListener(new myFocusListener());
```

# An Example Using FocusListener

```
import java.awt.*;
import java.awt.event.*;
public class FocusListenerDemo {
   static class myFocusListener implements FocusListener {
      @Override
      public void focusGained(FocusEvent e) { // set green background
         TextField textField = (TextField)e.getSource();
         textField.setBackground(Color.GREEN);
      }
      @Override
      public void focusLost(FocusEvent e) { // set red background
         TextField textField = (TextField)e.getSource();
         textField.setBackground(Color.RED);
      }
   }
   static class Demo extends Frame {
      public Demo() {
         this.setLayout(new FlowLayout());
         TextField textField1 = new TextField();  // create textfield
         TextField textField2 = new TextField();
         textField1.setColumns(15);
         textField2.setColumns(15);
         this.add(textField1);
         this.add(textField2);
         textField1.addFocusListener(new myFocusListener()); // register listener
         textField2.addFocusListener(new myFocusListener());
         this.setSize(200,100);
      }
   }
   public static void main(String[] args) {
      Demo demo = new Demo();
      demo.setVisible(true);
   }
}
```



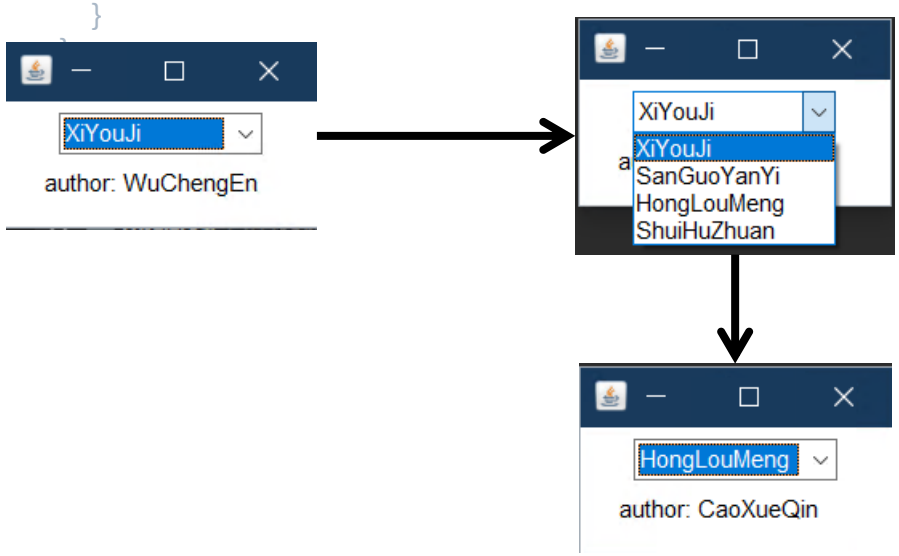
# Using the ItemListener

- ItemListener
  - Item events are fired by components that implement the ***ItemSelectable*** interface.
  - Generally, ***ItemSelectable*** components maintain on/off state for one or more items.
  - The AWT components that fire item events include ***Checkbox***, ***Choice***, ***List*** etc...
  - Define an overriding class of event type

```
class MyItemListener implements ItemListener {
    @Override
    public void itemStateChanged(ItemEvent e) {
}
```

# An Example Using ItemListener

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

public class ItemListenerDemo {
   static class MyItemListener implements ItemListener {
      private String[] books, authors;
      private Label author;
      public MyItemListener(String[] books, String[] authors, Label
author) {
         this.books = books;
         this.authors = authors;
         this.author = author;
      }
      @Override
      public void itemStateChanged(ItemEvent e) {
         String item = (String)e.getItem();
         for (int i = 0; i < books.length; ++i)
            if(item.equals(books[i]))
               author.setText("author: " + authors[i]);
      }
   }
   static class Demo extends Frame {
      private Label author;
      public Demo() {
         setLayout(new FlowLayout());
         Choice choice = new Choice();
         String[] books =
{"XiYouJi","SanGuoYanYi","HongLouMeng","ShuiHuZhuan"};
         String[] authors =
{"WuChengEn","LuoGuanZhong","CaoXueQin","ShiNaiAn"};
         for(int i = 0;i < 4; ++i)
            choice.add(books[i]);
         author = new Label("author: " + authors[0]);
         this.add(choice);
         this.add(author);
         choice.addItemListener(new
MyItemListener(books,authors,author));
         this.setSize(100,100);
      }
   }
   public static void main(String[] args) {
      Demo demo = new Demo();
      demo.setVisible(true);
   }
}
```



# Using the KeyListener

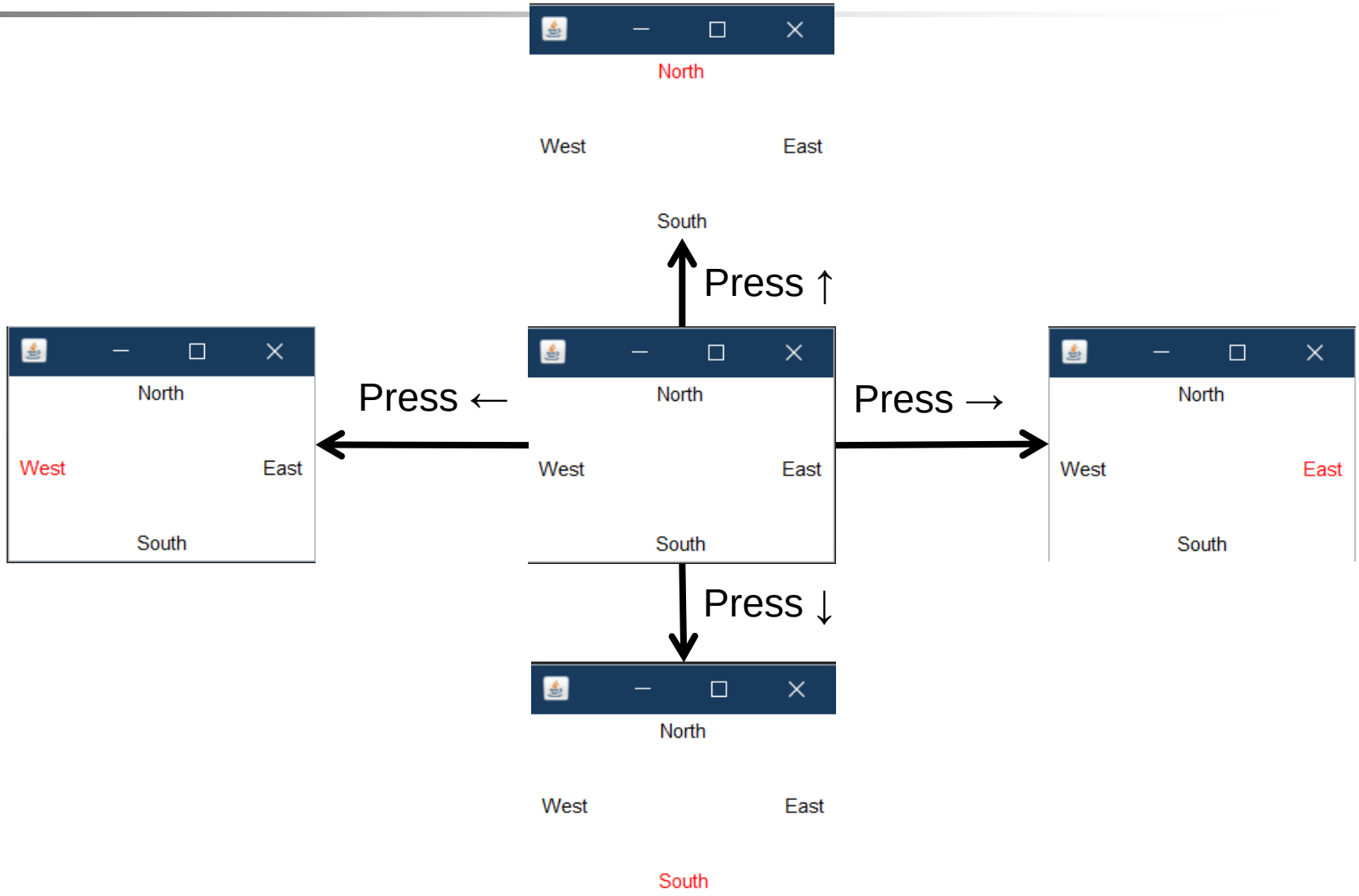
- KeyListener
  - Usage is the same with ActionListener, FocusListener and ItemListener

| Method | Purpose |
|---|---|
| keyTyped(KeyEvent) | Called just after the user types a Unicode character into the listened-to component. |
| keyPressed(KeyEvent) | Called just after the user presses a key while the listened-to component has the focus. |
| keyReleased(KeyEvent) | Called just after the user releases a key while the listened-to component has the focus. |

# An Example Using KeyListener

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

public class KeyListenerDemo {
   static class myKeyListener implements KeyListener {
      private Label[] labels;
      private int currentIndex;
      public myKeyListener(Label[] labels) {
         this.labels = labels;
         this.currentIndex = -1;
      }
      @Override
      public void keyTyped(KeyEvent e) {}
      @Override
      public void keyReleased(KeyEvent e) {}
      @Override
      public void keyPressed(KeyEvent e) {
         int index = -1;
         int[] keyCode = {KeyEvent.VK_LEFT, KeyEvent.VK_RIGHT,
KeyEvent.VK_DOWN, KeyEvent.VK_UP};
         for (int i = 0; i < 4; ++i)
            if (e.getKeyCode() == keyCode[i])
               index = i;
         if (index != -1) {
            if (currentIndex != -1)
               labels[currentIndex].setForeground(Color.BLACK);
            currentIndex = index;
            labels[currentIndex].setForeground(Color.red);
         }
      }
   }
   static class Demo extends Frame {
      private Label[] labels;
      public Demo() {
         this.setLayout(new BorderLayout());
         String[] directions = {BorderLayout.WEST, BorderLayout.EAST,
BorderLayout.SOUTH, BorderLayout.NORTH};
         labels = new Label[4];
         for(int i = 0;i < 4; ++i) {
            labels[i] = new Label(directions[i],Label.CENTER);
            this.add(labels[i], directions[i]);
         }
         this.addKeyListener(new myKeyListener(labels));
         this.setSize(200,150);
      }
   }
   public static void main(String[] args) {
      Demo demo = new Demo();
      demo.setVisible(true);
   }
}
```

# An Example Using KeyListener



# Using the MouseListener

- MouseListener
  - Mouse events notify when the user uses the mouse (or similar input device) to interact with a component.
  - Mouse events occur when the cursor enters or exits a component's onscreen area and when the user presses or releases one of the mouse buttons.
  - Usage is the same with ActionListener, FocusListener, etc.

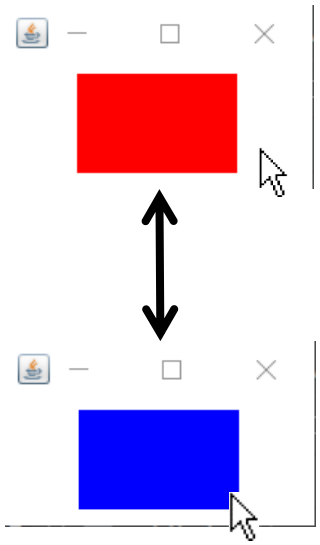
# Using the MouseListener

- MouseListener

| Method | Purpose |
| --- | --- |
| mouseClicked(MouseEvent) | Called just after the user clicks the listened-to component. |
| mouseEntered(MouseEvent) | Called just after the cursor enters the bounds of the listened-to component. |
| mouseExited(MouseEvent) | Called just after the cursor exits the bounds of the listened-to component. |
| mousePressed(MouseEvent) | Called just after the user presses a mouse button while the cursor is over the listened-to component. |
| mouseReleased(MouseEvent) | Called just after the user releases a mouse button after a mouse press over the listened-to component. |

# Using the MouseListener

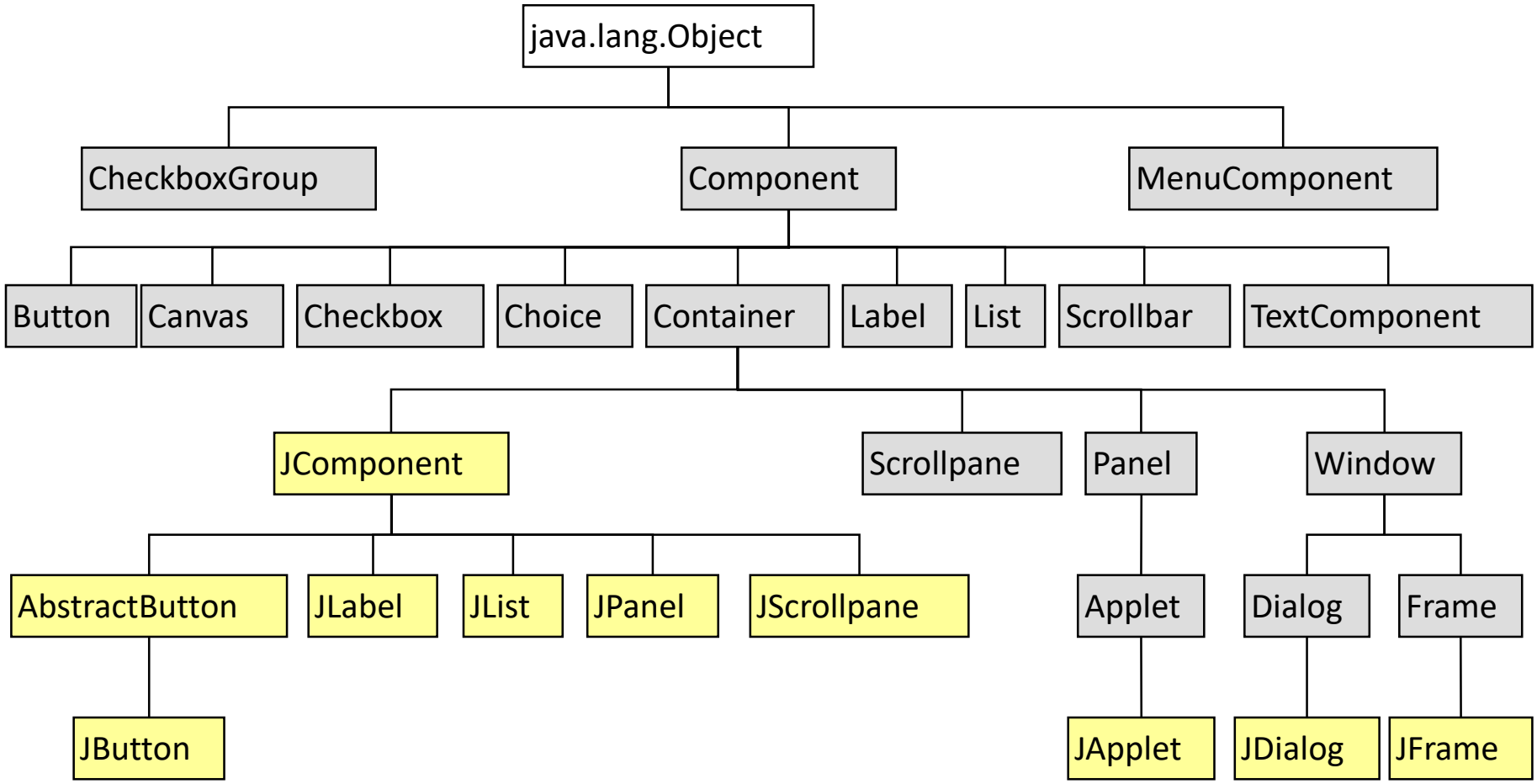
```
import java.awt.*;
import java.awt.event.*;

public class MouseListenerDemo {
   static class MyMouseListener implements MouseListener {
      private Canvas canvas;
      public MyMouseListener(Canvas canvas) {
         this.canvas = canvas;
      }
      @Override
      public void mouseClicked(MouseEvent e) { }
      @Override
      public void mousePressed(MouseEvent e) { }
      @Override
      public void mouseReleased(MouseEvent e) { }
      @Override
      public void mouseEntered(MouseEvent e) {
         this.canvas.setBackground(Color.blue);
      }
      @Override
      public void mouseExited(MouseEvent e) {
         this.canvas.setBackground(Color.red);
      }
   }
   static class Demo extends Frame {
      public Demo() {
         this.setLayout(new FlowLayout());
         Canvas canvas = new Canvas();
         canvas.setBackground(Color.red);
         canvas.addMouseListener(new MyMouseListener(canvas));
         canvas.setSize(80,50);
         this.add(canvas);
         this.setSize(100,100);
      }
   }
   public static void main(String[] args) {
      Demo demo = new Demo();
      demo.setVisible(true);
   }
}
```



# Swing

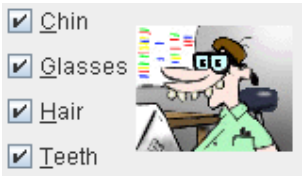
- Differences between AWT and Swing:
  - Swing components use no native code and they can be present on every platform
  - Typically, Swing components start their names with ‘J’
  - Have capabilities beyond what equivalent AWT components can offer
  - Swing components need not be rectangular
  - Swing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

# Partial AWT and Swing Class Hierarchy

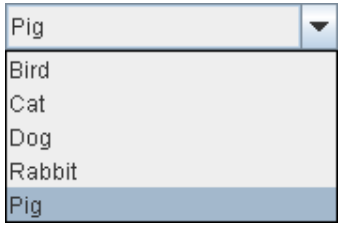


# Swing Components (Java Look and Feel)


JButton

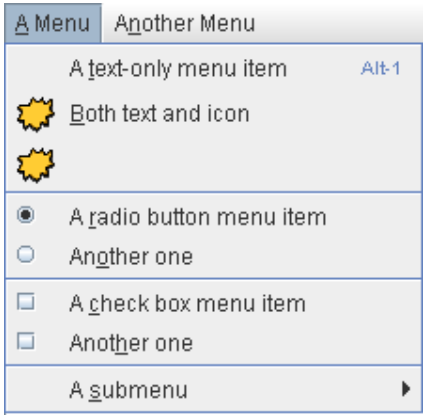



JCheckBox

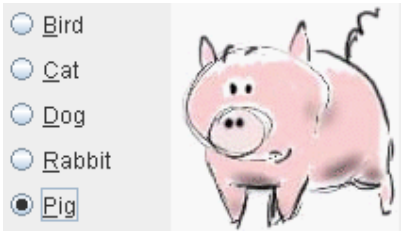



JComboBox


JList




JMenu




JRadioButton


JSlider

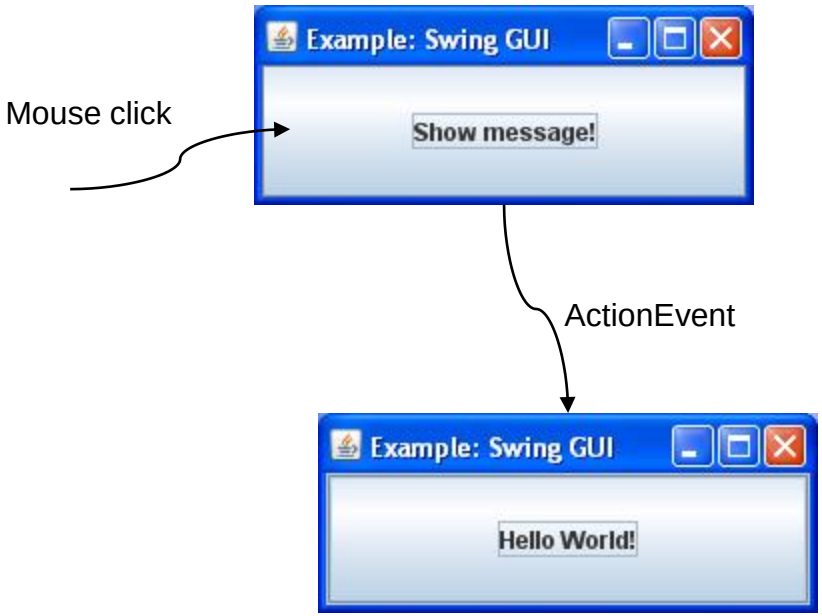

JSpinner


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!");
        }

}
```



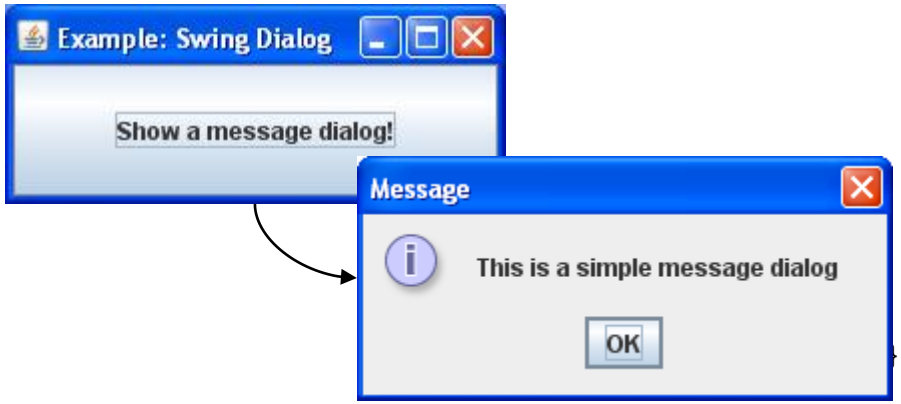
# Creating New Window Frame

```
// 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

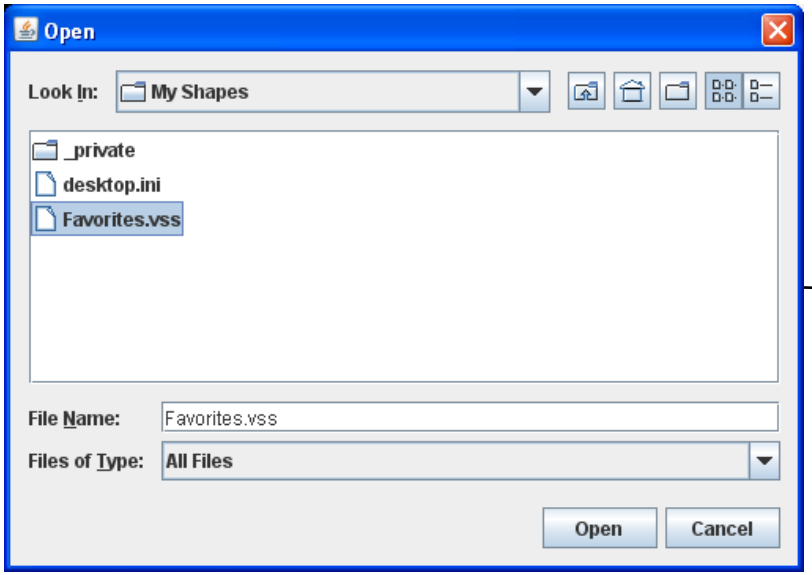
- A *dialog* is a special window to convey a message or provides a special function
- Every dialog is dependent on a frame – when that frame is destroyed, so are its dependent dialogs
- A *modal* dialog blocks user input to all other windows in the program




```
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());
        }

}
```

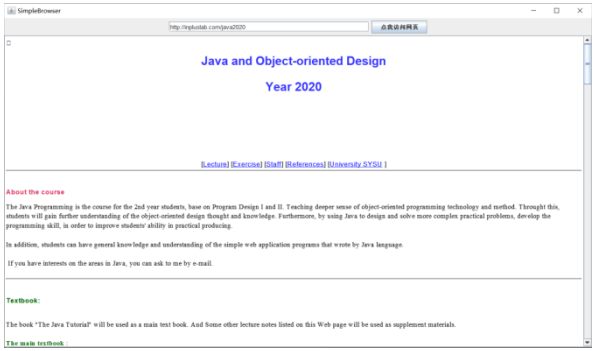
# AWT and Swing Example: Simple Browser

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

import javax.swing.*;
import javax.swing.event.*;
import javax.swing.text.html.*;

public class SimpleBrowser{
    public static void main(String[] args) throws
Exception {

        // html显示组件
        final JEditorPane jep = new JEditorPane();
        jep.setEditable(false);
        // 设置主页
        jep.setContentType("text/html;charset=utf-8");
        try {
            jep.setPage("http://inpluslab.com/java2020");
        } catch (IOException e) {
            jep.setText("<html>Error! Could not load
page</html>");
        }
        // 带滑动条的组件 用于存放显示html的jep组件
        JScrollPane scrollpane = new JScrollPane(jep);

        // 输入框 输入URL
        final JTextField jtf = new JTextField(40);
        jtf.setText("http://inpluslab.com/java2020");

        // 按钮
        final JButton goBtn = new JButton("点我访问网
页");

        // 上方菜单盒子
        JPanel menuBox = new JPanel();
        menuBox.add(jtf);
        menuBox.add(goBtn);
```

```
        // 添加超链接点击事件回调函数 并将JEditorPane
的页面改为超链接的页面
        jep.addHyperlinkListener(new HyperlinkListener()
{
            public void hyperlinkUpdate(HyperlinkEvent
event) {

if(event.getEventType()==HyperlinkEvent.EventType.A
CTIVATED) {
                try {
                    jep.setPage(event.getURL());
                } catch (IOException e) {
                    jep.setText("<html>Error! Could not
load page</html>");
                }
            }
        }
        });

        // 绑定访问按钮点击事件 从JTextField输入框获取
URL并且访问
        goBtn.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    jep.setPage(jtf.getText());
                } catch (IOException e1) {
                    jep.setText("<html>Error! Could not load
page</html>");
                }
            }
        });
        // 绑定输入框回车按键事件
        jtf.addKeyListener(new KeyAdapter() {
            public void keyPressed(KeyEvent event) {

if(event.getKeyChar()==KeyEvent.VK_ENTER) {
                goBtn.doClick();        // 按下回车等于点
击按钮
            }
        }
        });
```

```
        JFrame jf = new JFrame("SimpleBrowser");

jf.setDefaultCloseOperation(WindowConstants.DISPOSE
_ON_CLOSE);
        jf.setSize(1200, 700);
        jf.add(menuBox, BorderLayout.NORTH);
                    // 必须设置方位为North才能在上方显
示
        jf.add(scrollpane, BorderLayout.CENTER);
                    // 设置访问为center
        jf.setVisible(true);
    }
}
```

Running :



# AWT and Swing Example: Calculator

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

@SuppressWarnings("serial")
public class Calculator extends JFrame {
 private JTextField textField; // 显示文本框
 private void init() {
   textField = new JTextField();
   textField.setEditable(false);
   textField.setHorizontalAlignment (JTextField.RIGHT);
   textField.setFont(new Font(null, Font.PLAIN, 20));
   // 按键容器
   JPanel panel = new JPanel();
   panel.setLayout(new GridLayout(4, 4));
   Container = getContentPane();
   container.add(textField, BorderLayout.NORTH);
   container.add(panel, BorderLayout.CENTER);

   panel.add(createButton('7'));
   panel.add(createButton('8'));
   panel.add(createButton('9'));
   panel.add(createButton('/'));
   panel.add(createButton('4'));
   panel.add(createButton('5'));
   panel.add(createButton('6'));
   panel.add(createButton('*'));
   panel.add(createButton('1'));
   panel.add(createButton('2'));
   panel.add(createButton('3'));
   panel.add(createButton('-'));
   panel.add(createButton('0'));
   panel.add(createButton('.'));
   panel.add(createButton('='));
   panel.add(createButton('+'));
 }
```

```
public JButton createButton (char key) {
  // 创建按钮
  JButton button = new JButton(String.valueOf(key));
  button.setFont(new Font("粗体", Font.PLAIN, 15));

  // 单击时触发
  button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) {
      // 获取目标按钮
      JButton sourceBtn = (JButton) event.getSource();
      // 获取按钮上的字符
      char bottonKey = sourceBtn.getText().charAt(0);
      // 执行字符对应的操作
      calculatorAction(bottonKey);
    }
  });
  return button;
}
private void calculatorAction(char key) {
  switch(key) {
    case '.': case '0': case '1':
    case '2': case '3': case '4':
    case '5': case '6': case '7':
    case '8': case '9':
      String text = textField.getText() + key;
      textField.setText(text);
      break;
    default:
      break;
  }
}
```

```
 public static void main(String[] args) {
   Calculator = new Calculator();
   calculator.setTitle("Calculator");
   calculator.setSize(300, 300);
   calculator.setLocationRelativeTo(null);
   calculator.setResizable(false);

calculator.setDefaultCloseOperation(EXIT_ON_CLOSE);
   calculator.init();
   // 显示窗口
   calculator.setVisible(true);
 }
}
```

Running :