# JAVA GUI

# Objective

- Using graphics classes provided in JDK for constructing your own Graphical User Interface (GUI) applications.

## Requirements

- A good grasp of OOP, including composition, inheritance, polymorphism, abstract class and interface.

## Application Required

- JDK and NetBeans (Optional)

# Java GUI

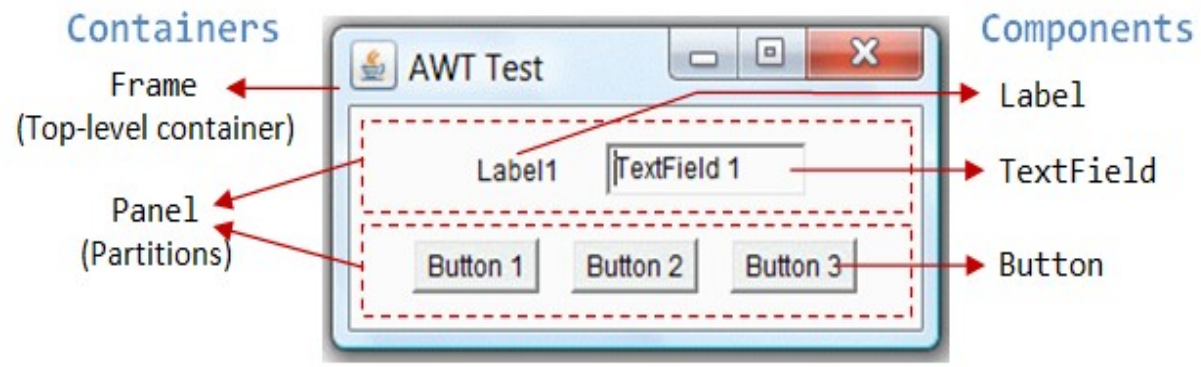
There are current three sets of Java APIs for graphics programming: AWT (Abstract Windowing Toolkit), Swing and JavaFX.

- **AWT**
  - AWT API was introduced in JDK 1.0.
  - Most of the AWT components have become obsolete and should be replaced by newer Swing components.
- **Swing**
  - Introduced as part of Java Foundation Classes (JFC) after the release of JDK 1.1.
  - A much more comprehensive set of graphics libraries that enhances the AWT
  - JFC consists of Swing, Java2D, Accessibility, Internationalization, and Pluggable Look-and-Feel Support APIs.
  - JFC has been integrated into core Java since JDK 1.2.
- **JavaFX**
  - Integrated into JDK 8, is meant to replace Swing.
- Although AWT is rarely used but for knowing complete picture of Java Graphics learning AWT is necessary.

# AWT

- AWT is huge! It consists of 12 packages of 370 classes (Swing is even bigger, with 18 packages of 737 classes as of JDK 8).
- 2 packages are used most commonly- java.awt and java.awt.event
- The java.awt package contains the core AWT graphics classes:
  - GUI Component classes, such as Button, TextField, and Label.
  - GUI Container classes, such as Frame and Panel.
  - Layout managers, such as FlowLayout, BorderLayout and GridLayout.
  - Custom graphics classes, such as Graphics, Color and Font.
- The java.awt.event package supports event handling:
  - Event classes, such as ActionEvent, MouseEvent, KeyEvent and WindowEvent,
  - Event Listener Interfaces, such as ActionListener, MouseListener, MouseMotionListener, KeyListener and WindowListener,
  - Event Listener Adapter classes, such as MouseAdapter, KeyAdapter, and WindowAdapter.

# Containers and Components

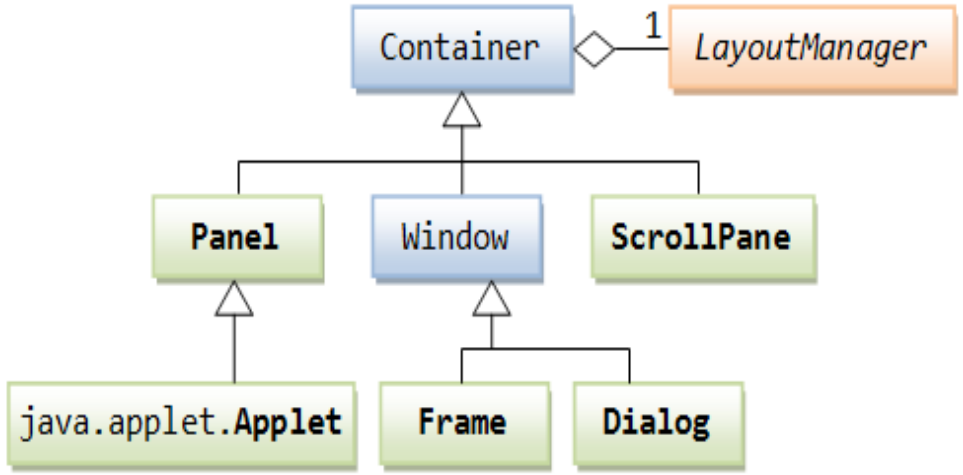



There are two types of GUI elements:

- **Component**: Components are elementary GUI entities, such as Button, Label, and TextField.
- **Container**: Containers, such as Frame and Panel, are used to hold components in a specific layout (such as FlowLayout or GridLayout). A container can also hold sub-containers.
- In a GUI program, a component must be kept in a container.
- Every container has a method called add(Component c). A container (say c) can invoke c.add(aComponent) to add a Component into itself.
- **For example**,

```
Panel pnl = new Panel();          // Panel is a container
Button btn = new Button("Press"); // Button is a component
pnl.add(btn);
```

# Class hierarchy of the AWT Container classes




- A Container has a LayoutManager to layout the components in a certain pattern.
- Complete Java AWT Hierarchy:https://docs.oracle.com/javase/7/docs/api/java/awt/package-tree.html

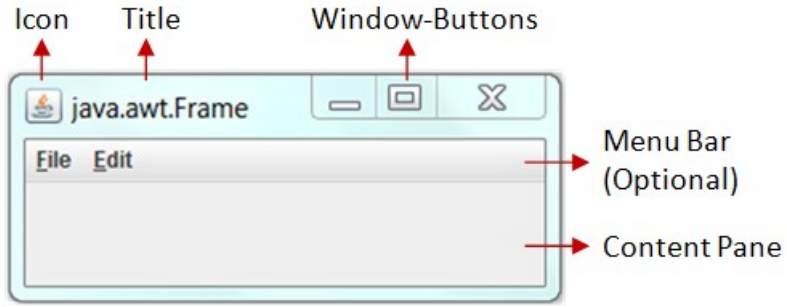
# Types AWT Container Classes

- Top-Level Containers: **Frame**, **Dialog** and **Applet**

- Secondary Containers: **Panel** and **ScrollPane**

# Top-Level Containers: Frame

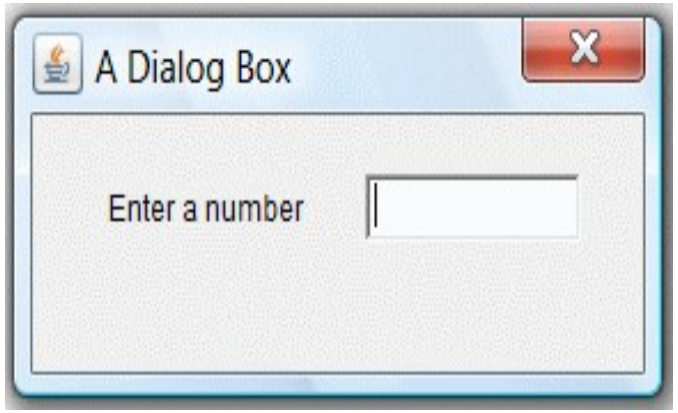
- A Frame provides the "main window" for your GUI application.
- It has a title bar, an optional menu bar, and the content display area.
- To write a GUI program, we typically start with a subclass extending from java.awt.Frame to inherit the main window.



# Frame Demo

```
import java.awt.Frame;  // Using Frame class in package java.awt
public class MyGUIProgram extends Frame {
   // private variables
   ......
   // Constructor to setup the GUI components
   public MyGUIProgram() { ..... }
   // methods
   ......
   // The entry main() method
   public static void main(String[] args) {
      // Invoke the constructor (to setup the GUI) by allocating an instance
      new MyGUIProgram();
   }
}
```

# Top-Level Containers: Dialog and Applet





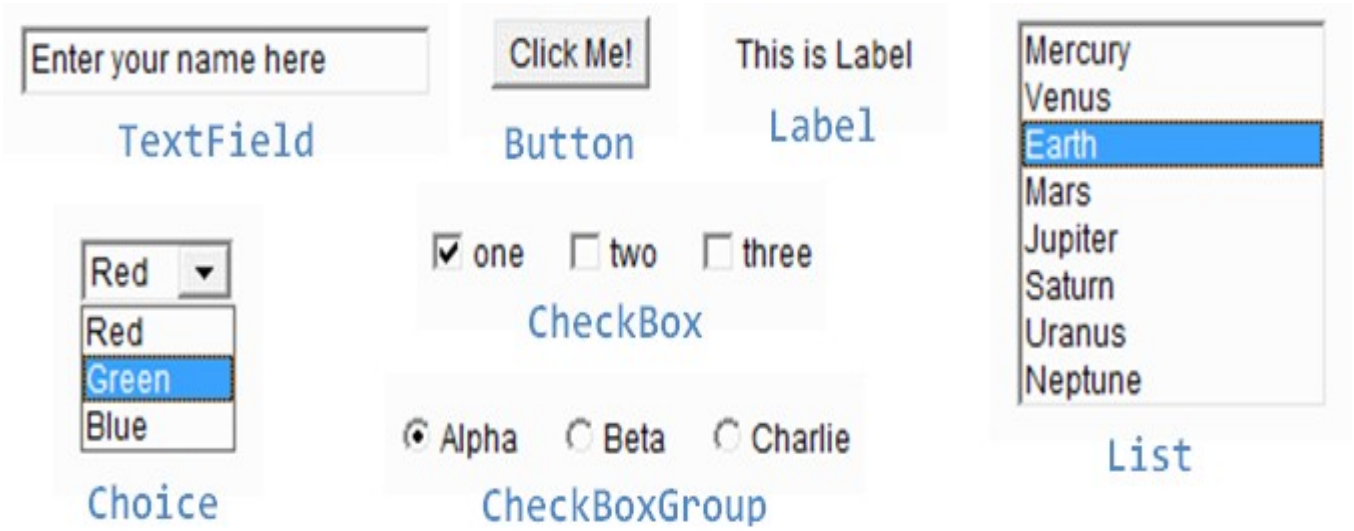
- An AWT Dialog is a "pop-up window" used for interacting with the users. A Dialog has a title-bar (containing an icon, a title and a close button) and a content display area, as illustrated.
- An AWT Applet (in package java.applet) is the top-level container for an applet, which is a Java program running inside a browser.

**NOTE: In this tutorial, we focus on Frame and JFrame as Container**

# Secondary Containers: Panel and ScrollPane

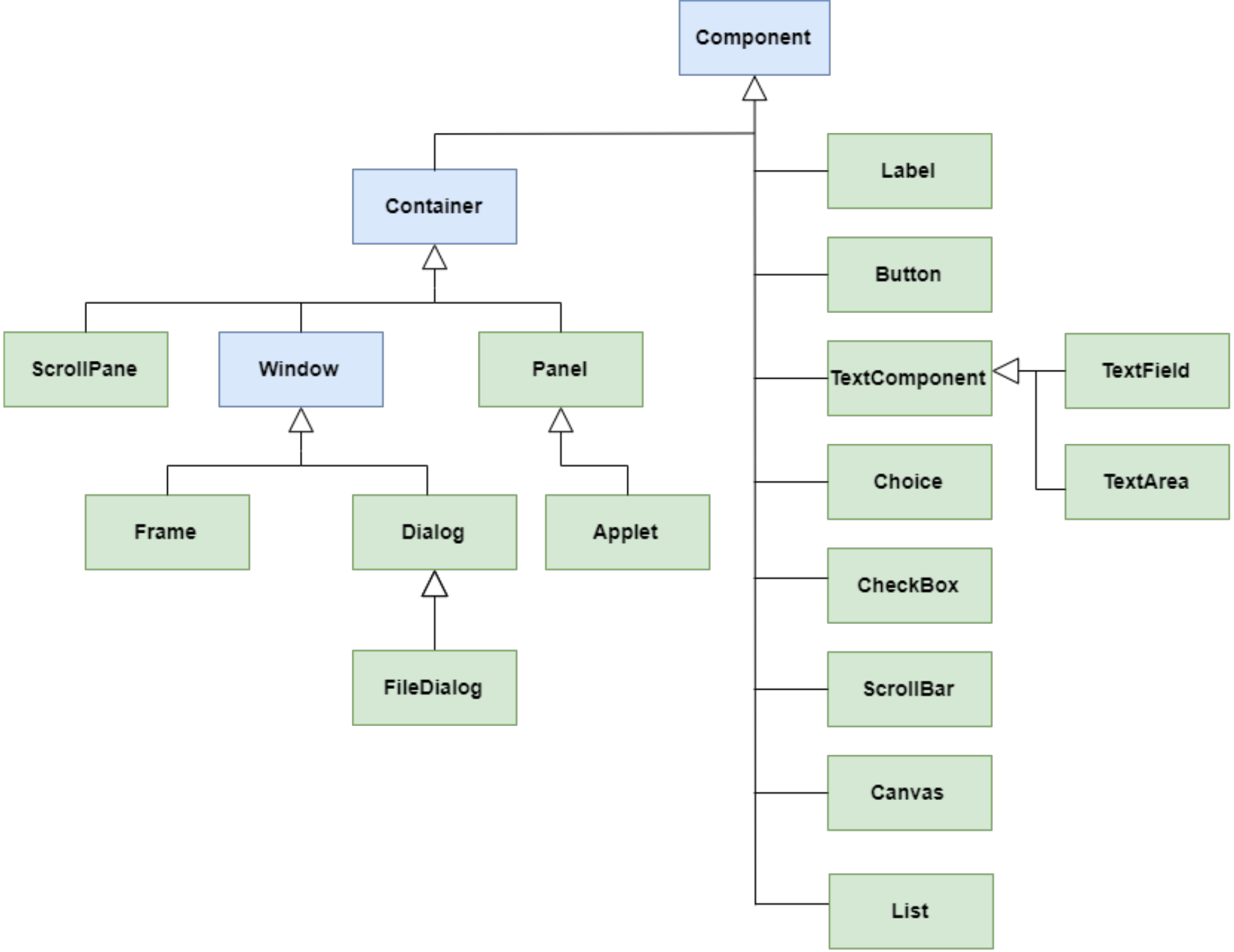
- Secondary containers are placed inside a top-level container or another secondary container. AWT provides these secondary containers:
  - **Panel**: a rectangular box used to layout a set of related GUI components in pattern such as grid or flow.
  - **ScrollPane**: provides automatic horizontal and/or vertical scrolling for a single child component.

# AWT Component Classes



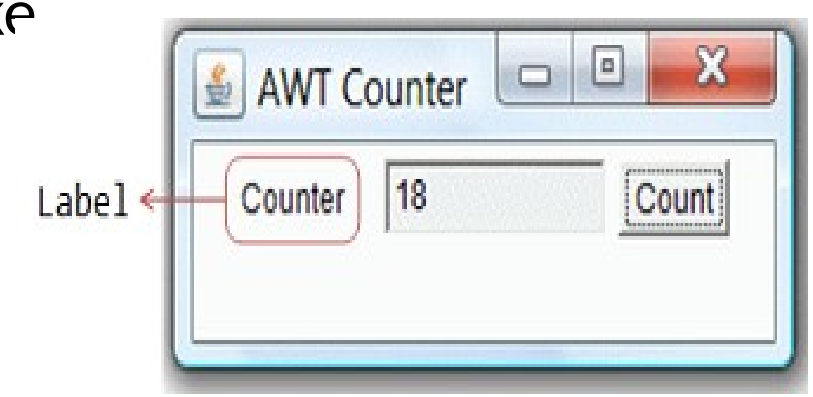

- AWT provides many ready-made and reusable GUI components in package java.awt.
- The frequently-used are: Button, TextField, Label, Checkbox, CheckboxGroup (radio buttons), List, and Choice.

# AWT Component Class Hierarchy



# AWT GUI Component: java.awt.Label

- A java.awt.Label provides a descriptive text string. Take note that System.out.println() prints to the system console, NOT to the graphics screen.
- Use to label other component (such as text field) to provide a text description.



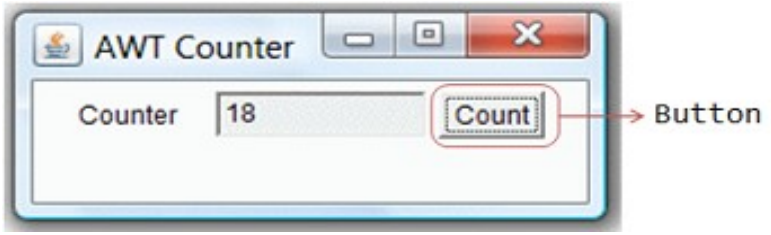

- Constructor

```
public Label(String strLabel, int alignment); // Construct a Label with the given text String, of the text alignment
public Label(String strLabel);                // Construct a Label with the given text String
public Label();                               // Construct an initially empty Label
```

- Public Methods
  - public String getText();
  - public void setText(String strLabel);
  - public int getAlignment();
  - public void setAlignment(int alignment);  // Label.LEFT, Label.RIGHT, Label.CENTER

# AWT GUI Component: java.awt.Button

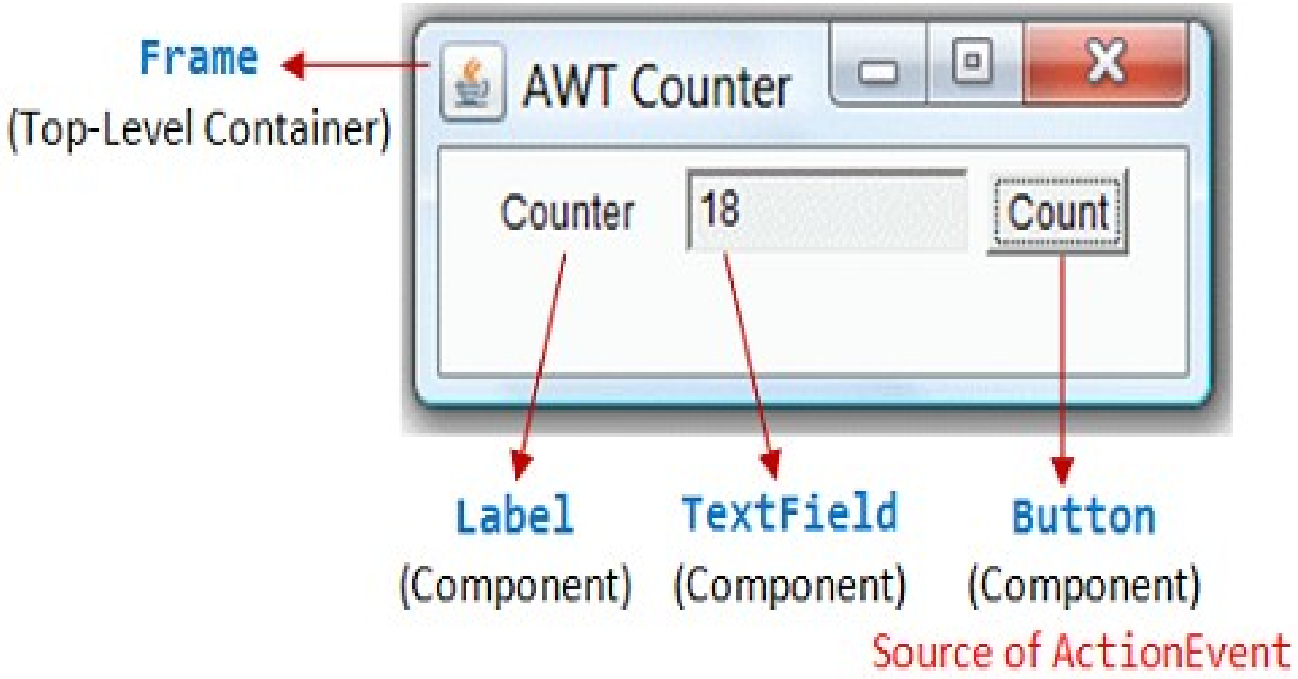
- A java.awt.Button is a GUI component that triggers a certain programmed action upon clicking.
- **Constructors**
  - public Button(String btnLabel);  // Construct a Button with the given label
  - public Button();  // Construct a Button with empty label
- **Public Methods**
  - public String getLabel(); // Get the label of this Button instance
  - public void setLabel(String btnLabel); // Set the label of this Button instance
  - public void setEnable(boolean enable); // Enable or disable this Button. Disabled Button cannot be clicked.
- **Event**
  - Clicking a button fires a so-called ActionEvent and triggers a certain programmed action



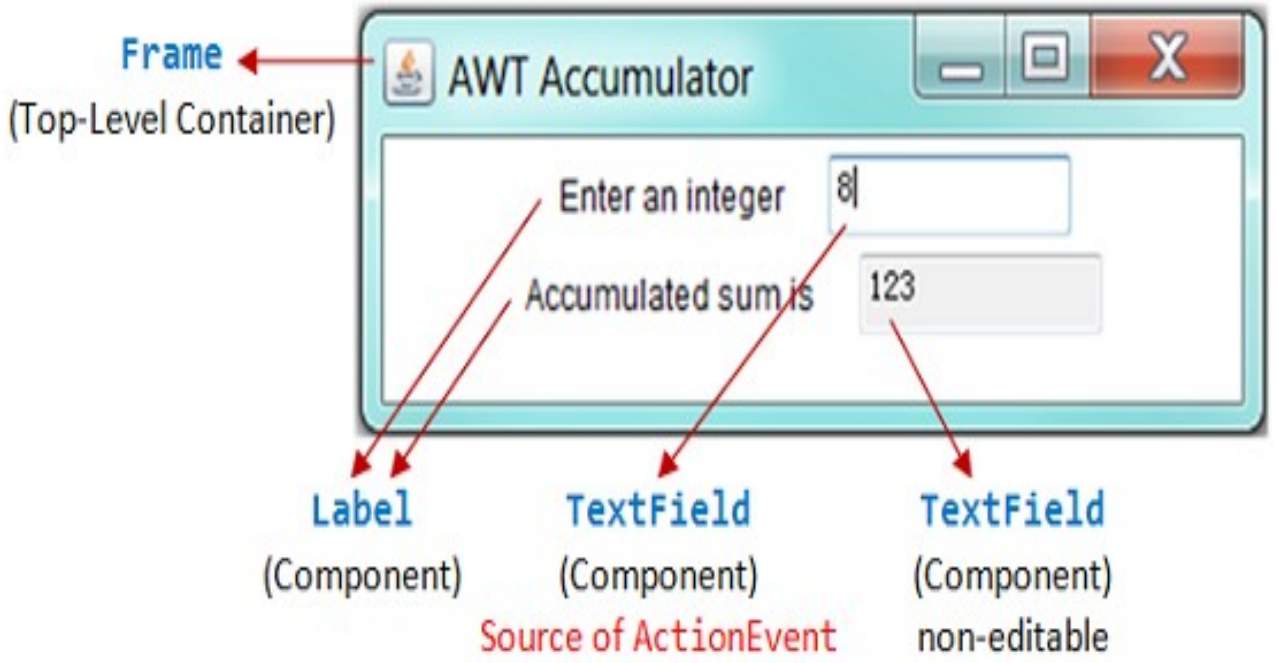
# AWT GUI Component: java.awt.TextField

- A java.awt.TextField is single-line text box for users to enter texts.
- Hitting the "ENTER" key on a TextField object fires an ActionEvent.
- **Constructors**
  - public TextField(String initialText, int columns); // Construct a TextField instance with the given initial text string with the number of columns.
  - public TextField(String initialText); // Construct a TextField instance with the given initial text string.
  - public TextField(int columns); // Construct a TextField instance with the number of columns.
- **Public Methods**
  - public String getText(); // Get the current text on this TextField instance
  - public void setText(String strText); // Set the display text on this TextField instance
  - public void setEditable(boolean editable); // Set this TextField to editable (read/write) or non-editable (read-only)
- **Event**
  - Hitting the "ENTER" key on a TextField fires a ActionEvent, and triggers a certain programmed action.

# Example 1: AWTCounter

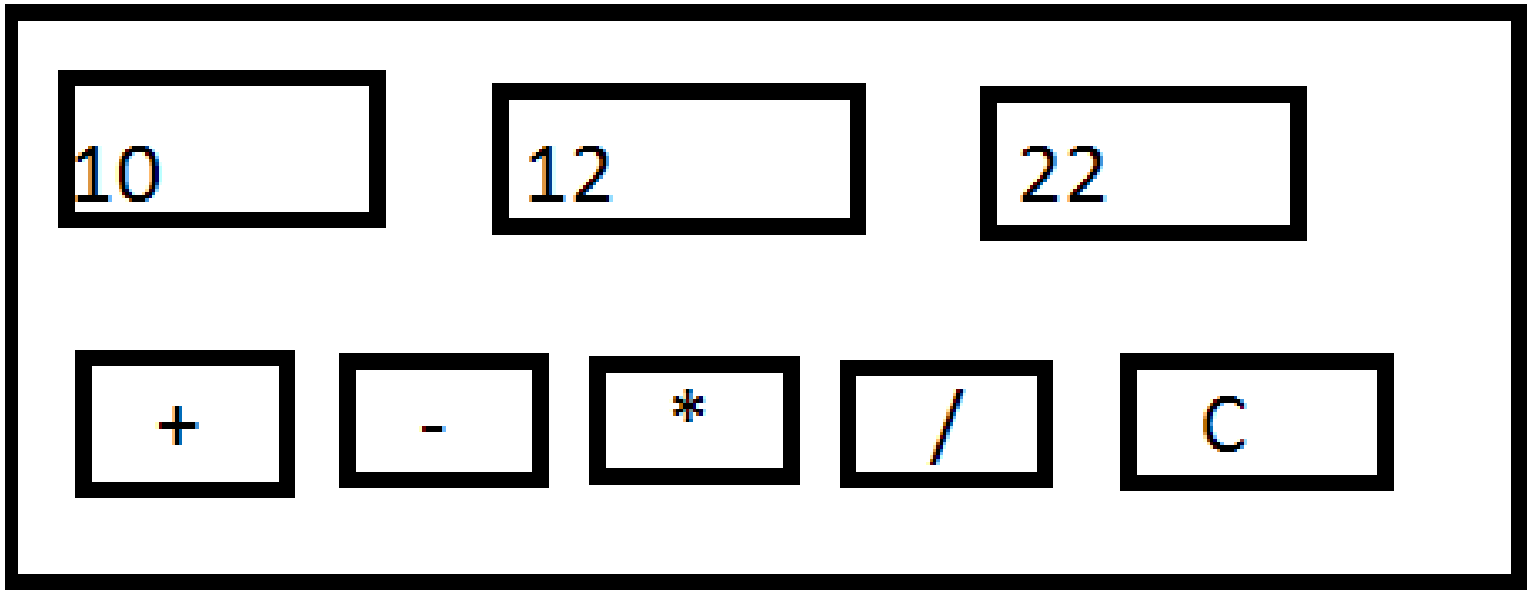


# Example 2: AWTAccumulator



# Home Work

- Small AWT Calculator

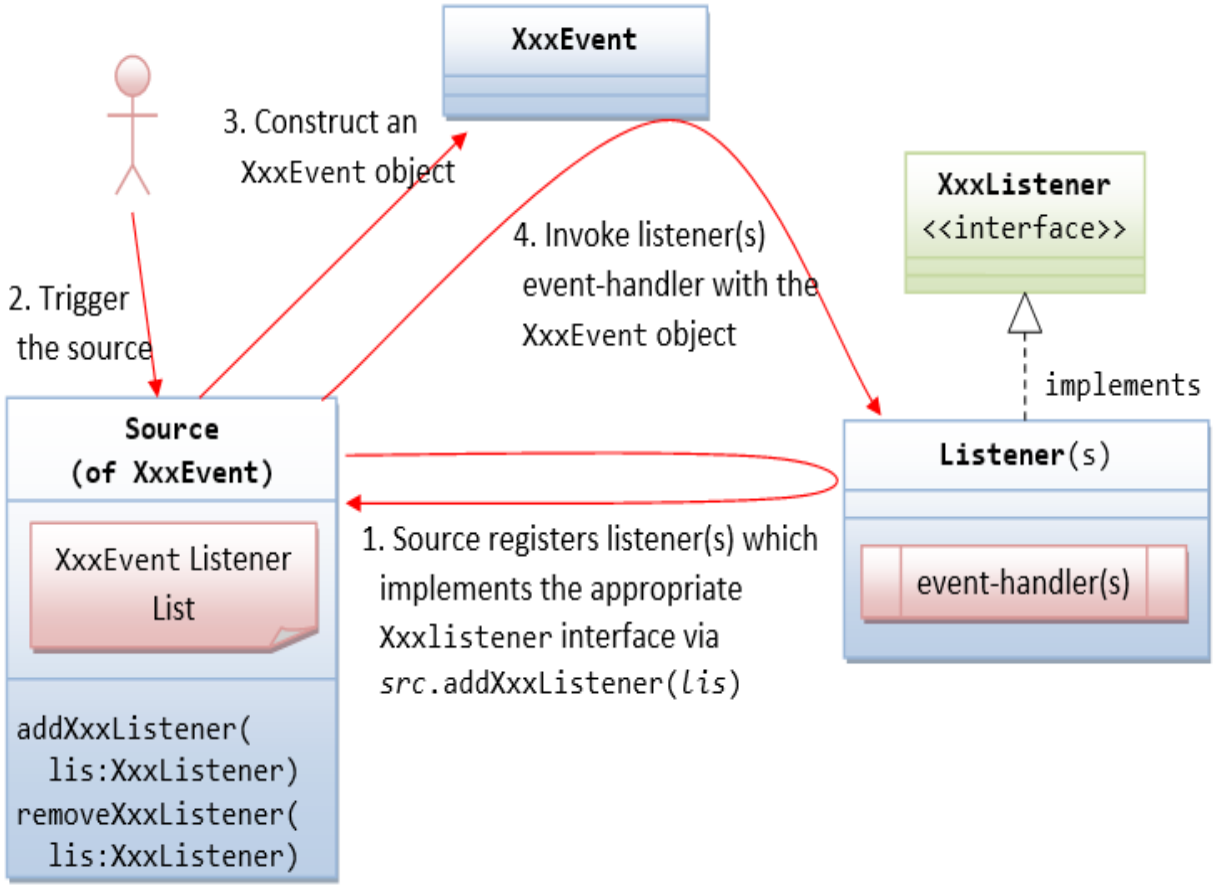


# AWT Event-Handling

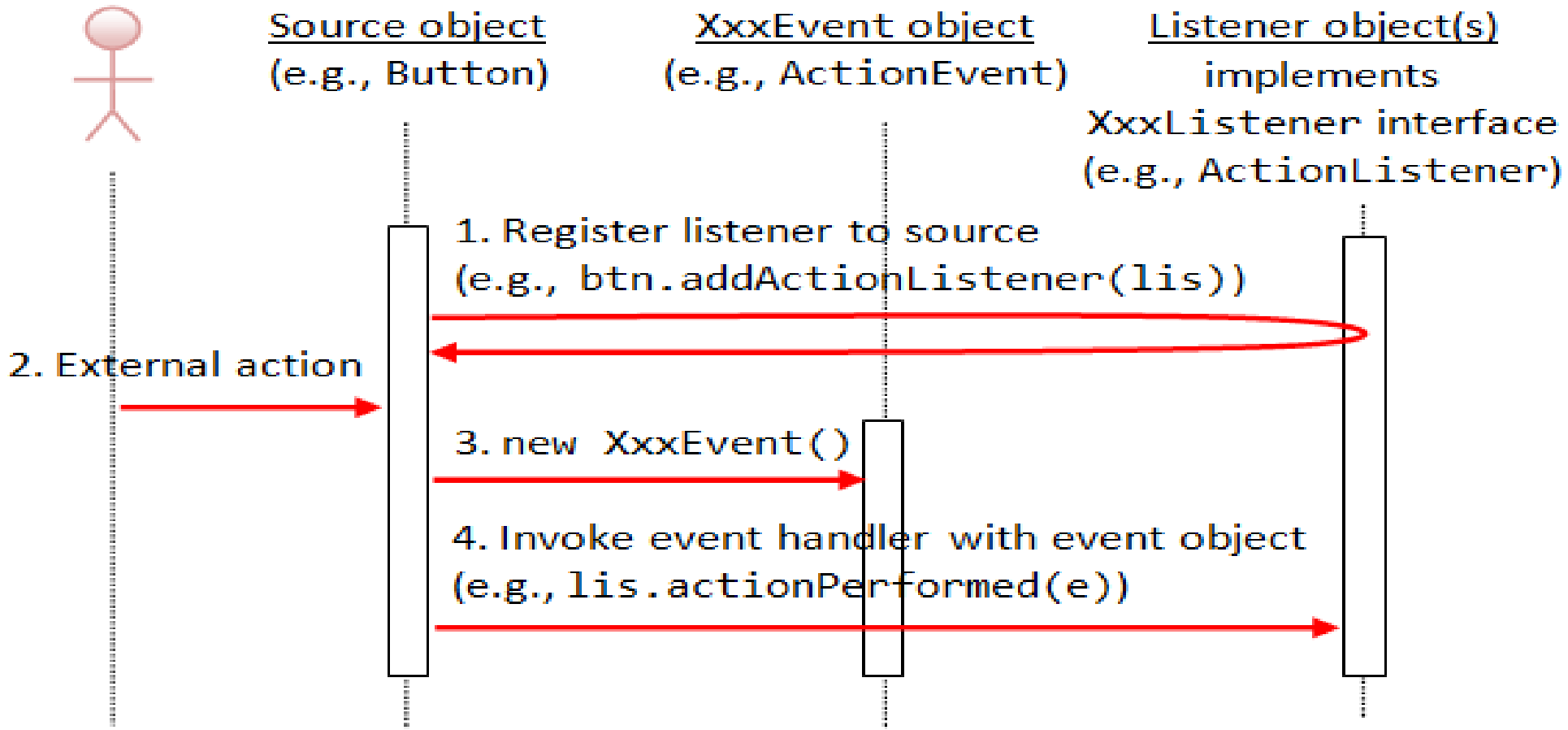
- The AWT's event-handling classes are kept in package java.awt.event.
- Three kinds of objects are involved in the event-handling: a source, listener(s) and an event object.

Steps:-

- The source object (such as Button and Textfield) interacts with the user.
- Upon triggered, the source object creates an event object to capture the action (e.g., mouse-click x and y, texts entered, etc).
- This event object will be messaged to all the registered listener object(s), and an appropriate event-handler method of the listener(s) is called-back to provide the response.
- Follow *subscribe-publish* or *observable-observer* design pattern.



# Sequence Diagram for Event-Handling



# WindowEvent and WindowListener Interface

- A WindowEvent is fired (to all its WindowEvent listeners) when a window (e.g., Frame) has been
  - opened/closed
  - activated/deactivated
  - iconified/deiconified via the 3 buttons at the top-right corner or other means.
- The source of WindowEvent shall be a top-level window-container such as Frame.

# WindowEvent and WindowListener Interface (Cont'd)

- A WindowEvent listener must implement WindowListener interface, which declares 7 abstract event-handling methods, as follows.
  - public void windowClosing(WindowEvent evt)

  // Called-back when the user attempts to close the window by clicking the window close button.

  // This is the most-frequently used handler.
  - public void windowOpened(WindowEvent evt)

  // Called-back the first time a window is made visible.
  - public void windowClosed(WindowEvent evt)

  // Called-back when a window has been closed as the result of calling dispose on the window.
  - public void windowActivated(WindowEvent evt)

  // Called-back when the Window is set to be the active Window.

public void windowDeactivated(WindowEvent evt)

// Called-back when a Window is no longer the active Window.

public void windowIconified(WindowEvent evt)

// Called-back when a window is changed from a normal to a minimized state.

public void windowDeiconified(WindowEvent evt)

// Called-back when a window is changed from a minimized to a normal state.

Among them, the windowClosing(), which is called back upon clicking the window-close button, is the most commonly-used.

# Added support for "close-window button" to Example 1: AWTCounter

```
import java.awt.*;
import java.awt.event.*;
public class WindowEventDemo extends Frame
   implements ActionListener, WindowListener {
   private TextField tfCount;
   private Button btnCount;
   private int count = 0;
   public WindowEventDemo() {
      setLayout(new FlowLayout());
      add(new Label("Counter"));
      tfCount = new TextField("0", 10);
      tfCount.setEditable(false);
      add(tfCount);
      btnCount = new Button("Count");
      add(btnCount);
      btnCount.addActionListener(this);
      addWindowListener(this);
      setTitle("WindowEvent Demo");
      setSize(250, 100);
      setVisible(true);
   }
   public static void main(String[] args) {
      new WindowEventDemo();
   }
   /* ActionEvent handler */
   @Override
   public void actionPerformed(ActionEvent evt) {
      ++count;
      tfCount.setText(count + "");
   }
   // Called back upon clicking close-window button
   @Override
   public void windowClosing(WindowEvent evt) {
      System.exit(0);  // Terminate the program
   }
   // Not Used, BUT need to provide an empty body to compile.
   @Override public void windowOpened(WindowEvent evt) { }
   @Override public void windowClosed(WindowEvent evt) { }
   // For Debugging
   @Override public void windowIconified(WindowEvent evt)
   { System.out.println("Window Iconified"); }
   @Override public void windowDeiconified(WindowEvent evt)
   { System.out.println("Window Deiconified"); }
   @Override public void windowActivated(WindowEvent evt)
   { System.out.println("Window Activated"); }
   @Override public void windowDeactivated(WindowEvent evt)
   { System.out.println("Window Deactivated"); }
}
```

# **MouseEvent:** MouseListener Interface

- A MouseEvent is fired when you
  - press, release, or click (press followed by release) a mouse-button (left or right button) at the source object; position the mouse-pointer at (enter) and away (exit) from the source object.
- A MouseEvent listener must implement the MouseListener interface, which declares the following five abstract methods:

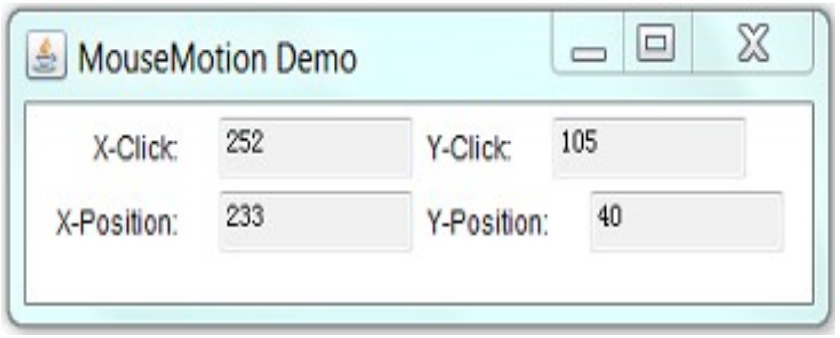
```
public void mouseClicked(MouseEvent evt)
// Called-back when the mouse-button has been clicked (pressed followed by released) on the source.
public void mousePressed(MouseEvent evt)
public void mouseReleased(MouseEvent evt)
// Called-back when a mouse-button has been pressed/released on the source.
// A mouse-click invokes mousePressed(), mouseReleased() and mouseClicked().
public void mouseEntered(MouseEvent evt)
public void mouseExited(MouseEvent evt)
// Called-back when the mouse-pointer has entered/exited the source.
```

# **MouseEvent:** MouseMotionListener

- A MouseEvent is also fired when you move and drag the mouse pointer at the source object.
- But you need to use MouseMotionListener to handle the mouse-move and mouse-drag.
- The MouseMotionListener interface declares the following two abstract methods:

```
public void mouseDragged(MouseEvent e)
// Called-back when a mouse-button is pressed on the source component and then dragged.
public void mouseMoved(MouseEvent e)
// Called-back when the mouse-pointer has been moved onto the source component but no buttons have been pushed.
```

# Example3: MouseMotionDemo

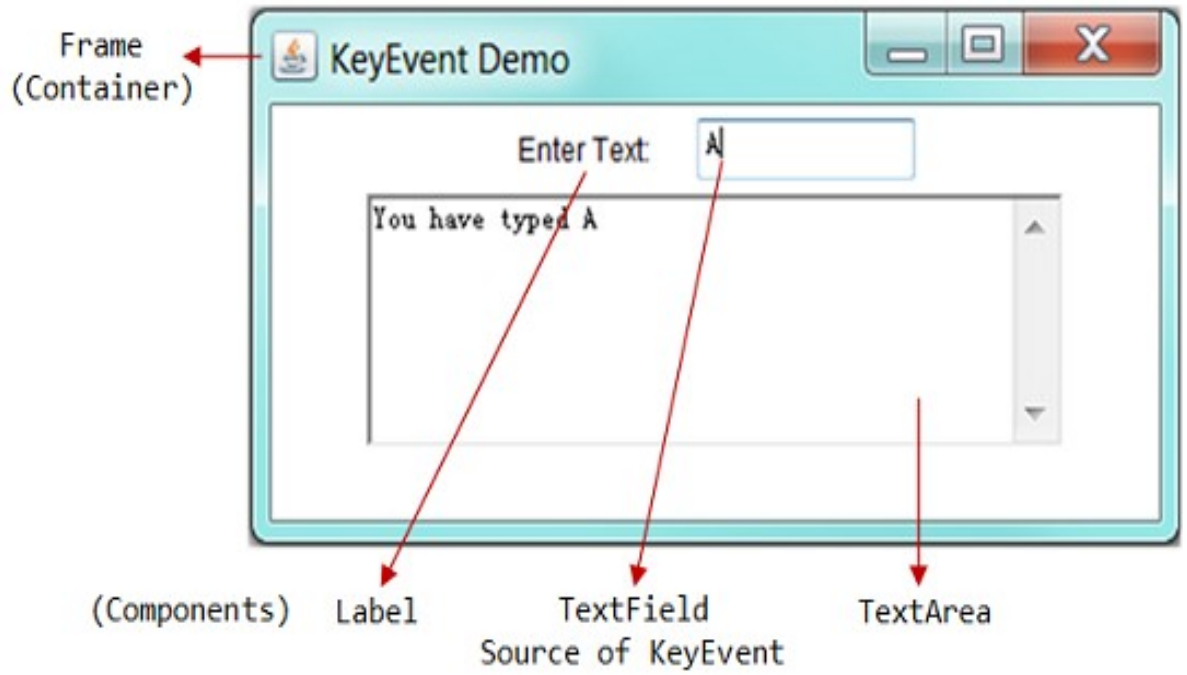


```java
import java.awt.*;
import java.awt.event.*;
// An AWT GUI program inherits from the top-level container java.awt.Frame
public class MouseMotionDemo extends Frame
      implements MouseListener, MouseMotionListener {
      // This class acts as MouseListener and MouseMotionListener

   // To display the (x, y) of the mouse-clicked
   private TextField tfMouseClickX;
   private TextField tfMouseClickY;
   // To display the (x, y) of the current mouse-pointer position
   private TextField tfMousePositionX;
   private TextField tfMousePositionY;

   // Constructor to setup the GUI components and event handlers
   public MouseMotionDemo() {
      setLayout(new FlowLayout()); // "super" frame sets to FlowLayout

      add(new Label("X-Click: "));
      tfMouseClickX = new TextField(10);
      tfMouseClickX.setEditable(false);
      add(tfMouseClickX);
      add(new Label("Y-Click: "));
      tfMouseClickY = new TextField(10);
      tfMouseClickY.setEditable(false);
      add(tfMouseClickY);

      add(new Label("X-Position: "));
      tfMousePositionX = new TextField(10);
      tfMousePositionX.setEditable(false);
      add(tfMousePositionX);
      add(new Label("Y-Position: "));
      tfMousePositionY = new TextField(10);
      tfMousePositionY.setEditable(false);
      add(tfMousePositionY);

      addMouseListener(this);
      addMouseMotionListener(this);
        // "super" frame (source) fires MouseEvent.
        // "super" frame adds "this" object as MouseListener and
        //MouseMotionListener.
      setTitle("MouseMotion Demo"); // "super" Frame sets title
      setSize(400, 120);            // "super" Frame sets initial size
      setVisible(true);             // "super" Frame shows
   }

   // The entry main() method
   public static void main(String[] args) {
      new MouseMotionDemo();  // Let the constructor do the job
   }

   /** MouseListener handlers */
   // Called back when a mouse-button has been clicked
   @Override
   public void mouseClicked(MouseEvent evt) {
      tfMouseClickX.setText(evt.getX() + "");
      tfMouseClickY.setText(evt.getY() + "");
   }

   // Not Used, but need to provide an empty body for compilation
   @Override public void mousePressed(MouseEvent evt) { }
   @Override public void mouseReleased(MouseEvent evt) { }
   @Override public void mouseEntered(MouseEvent evt) { }
   @Override public void mouseExited(MouseEvent evt) { }

   /** MouseMotionEvent handlers */
   // Called back when the mouse-pointer has been moved
   @Override
   public void mouseMoved(MouseEvent evt) {
      tfMousePositionX.setText(evt.getX() + "");
      tfMousePositionY.setText(evt.getY() + "");
   }
   // Not Used, but need to provide an empty body for compilation
   @Override public void mouseDragged(MouseEvent evt) { }
}
```

# KeyEvent and KeyListener Interface

- A KeyEvent is fired when you pressed, released, and typed (pressed followed by released) a key on the source object.
- A KeyEvent listener must implement KeyListener interface, which declares three abstract methods:
  - public void keyTyped(KeyEvent e) // Called-back when a key has been typed (pressed and released).
  - public void keyPressed(KeyEvent e)
  - public void keyReleased(KeyEvent e) // Called-back when a key has been pressed or released.

# Example4: KeyEventDemo



# KeyEventDemo

```java
import java.awt.*;
import java.awt.event.*;
public class KeyEventDemo extends Frame implements KeyListener {
   private TextField tfInput;
   private TextArea taDisplay;
   // Constructor to setup the GUI components and event handlers
   public KeyEventDemo() {
      setLayout(new FlowLayout());

      add(new Label("Enter Text: "));
      tfInput = new TextField(10);
      add(tfInput);
      taDisplay = new TextArea(5, 40);
      add(taDisplay);

      tfInput.addKeyListener(this);
         // tfInput TextField (source) fires KeyEvent.
         // tfInput adds "this" object as a KeyEvent listener.
      setTitle("KeyEvent Demo");
      setSize(400, 200);
      setVisible(true);
   }
   // The entry main() method
   public static void main(String[] args) {
      new KeyEventDemo();
   }
   /** KeyEvent handlers */
   // Called back when a key has been typed (pressed and released)
   @Override
   public void keyTyped(KeyEvent evt) {
      taDisplay.append("You have typed " + evt.getKeyChar() + "\n");
   }
   // Not Used, but need to provide an empty body for compilation
   @Override public void keyPressed(KeyEvent evt) { }
   @Override public void keyReleased(KeyEvent evt) { }
}
```

# Nested (Inner) Classes

- A nested class (or commonly called inner class) is a class defined inside another class - introduced in JDK 1.1.

```
public class MyOuterClass {   // outer class defined here

   ......

   private class MyNestedClass1 { ..... }  // an nested class defined inside the outer class

   public static class MyNestedClass2 { ..... }  // an "static" nested class defined inside the outer class

   ......
}
```

# A nested class has these properties:

- A nested class is a proper class. That is, it could contain constructors, member variables and member methods.
- A nested class is a member of the outer class, just like any member variables and methods defined inside a class.
- Most importantly, a nested class can access the private members (variables/methods) of the enclosing outer class.
- A nested class can have private, public, protected, or the default access, just like any member variables and methods defined inside a class.
- A nested class can also be declared static, final or abstract, just like any ordinary class.
- A nested class is NOT a subclass of the outer class. That is, the nested class does not inherit the variables and methods of the outer class.

# The usages of nested class are:

- To control visibilities (of the member variables and methods) between inner/outer class. The nested class, being defined inside an outer class, can access private members of the outer class.
- To place a piece of class definition codes closer to where it is going to be used, to make the program clearer and easier to understand.
- For namespace management.

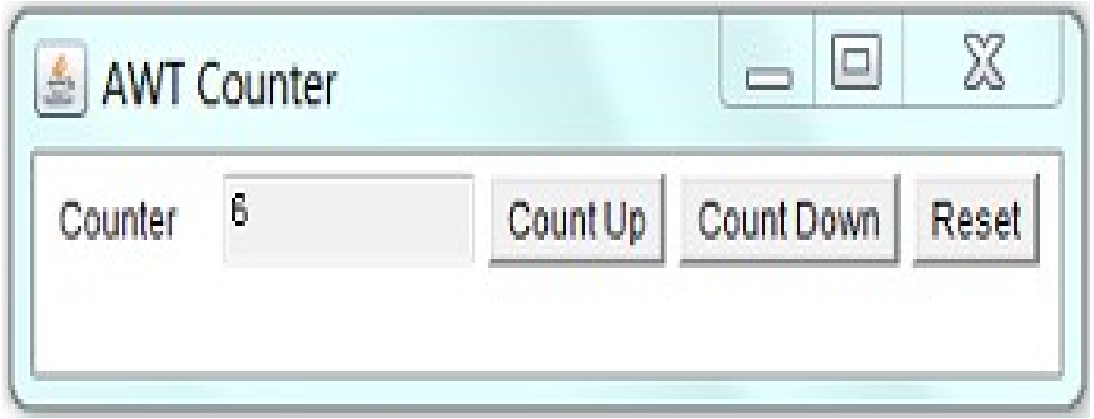
Event Handling using Nested Class

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

// An AWT GUI program inherits from the top-level container java.awt.Frame
public class AWTCounterNamedInnerClass extends Frame {
   // This class is NOT a ActionListener, hence, it does not implement ActionListener interface

   // The event-handler actionPerformed() needs to access these "private" variables
   private TextField tfCount;
   private Button btnCount;
   private int count = 0;

   // Constructor to setup the GUI components and event handlers
   public AWTCounterNamedInnerClass () {
      setLayout(new FlowLayout());  // "super" Frame sets to FlowLayout
      add(new Label("Counter"));    // An anonymous instance of Label
      tfCount = new TextField("0", 10);
      tfCount.setEditable(false);   // read-only
      add(tfCount);                 // "super" Frame adds tfCount

      btnCount = new Button("Count");
      add(btnCount);                // "super" Frame adds btnCount

      // Construct an anonymous instance of BtnCountListener (a named inner class).
      // btnCount adds this instance as a ActionListener.
      btnCount.addActionListener(new BtnCountListener());

      setTitle("AWT Counter");
      setSize(250, 100);
      setVisible(true);
   }

   // The entry main method
   public static void main(String[] args) {
      new AWTCounterNamedInnerClass(); // Let the constructor do the job
   }

   /**
    * BtnCountListener is a "named inner class" used as ActionListener.
    * This inner class can access private variables of the outer class.
    */
   private class BtnCountListener implements ActionListener {
      @Override
      public void actionPerformed(ActionEvent evt) {
         ++count;
         tfCount.setText(count + "");
      }
   }
}
```

## An Anonymous Inner Class as Event Listener

```
1  import java.awt.*;
2  import java.awt.event.*;
3
4  // An AWT GUI program inherits from the top-level container java.awt.Frame
5  public class AWTCounterAnonymousInnerClass extends Frame {
6     // This class is NOT a ActionListener, hence, it does not implement ActionListener interface
7
8     // The event-handler actionPerformed() needs to access these private variables
9     private TextField tfCount;
10    private Button btnCount;
11    private int count = 0;
12
13    // Constructor to setup the GUI components and event handlers
14    public AWTCounterAnonymousInnerClass () {
15       setLayout(new FlowLayout());  // "super" Frame sets to FlowLayout
16       add(new Label("Counter"));    // An anonymous instance of Label
17       tfCount = new TextField("0", 10);
18       tfCount.setEditable(false);   // read-only
19       add(tfCount);                 // "super" Frame adds tfCount
20
21       btnCount = new Button("Count");
22       add(btnCount);                // "super" Frame adds btnCount
23
24       // Construct an anonymous instance of an anonymous class.
25       // btnCount adds this instance as a ActionListener.
26       btnCount.addActionListener(new ActionListener() {
27          @Override
28          public void actionPerformed(ActionEvent evt) {
29             ++count;
30             tfCount.setText(count + "");
31          }
32       });
33
34       setTitle("AWT Counter");
35       setSize(250, 100);
36       setVisible(true);
37    }
38
39    // The entry main method
40    public static void main(String[] args) {
41       new AWTCounterAnonymousInnerClass(); // Let the constructor do the job
42    }
43 }
```

# Modifying the Counter Application using an Anonymous Inner Class




Each of the Buttons uses one anonymous instance of an anonymous inner class as its ActionEvent listener.

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

// An AWT GUI program inherits the top-level container java.awt.Frame
public class AWTCounter3Buttons extends Frame {
   private TextField tfCount;
   private Button btnCountUp, btnCountDown, btnReset;
   private int count = 0;

   // Constructor to setup the GUI components and event handlers
   public AWTCounter3Buttons () {
      setLayout(new FlowLayout());
      add(new Label("Counter"));   // an anonymous instance of Label
      tfCount = new TextField("0", 10);
      tfCount.setEditable(false);  // read-only
      add(tfCount);                // "super" Frame adds tfCount

      btnCountUp = new Button("Count Up");
      add(btnCountUp);
      // Construct an anonymous instance of an anonymous inner class.
      // The source Button adds the anonymous instance as ActionEvent listener
      btnCountUp.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent evt) {
            ++count;
            tfCount.setText(count + "");
         }
      });

      btnCountDown = new Button("Count Down");
      add(btnCountDown);
      btnCountDown.addActionListener(new ActionListener() {
          @Override
          public void actionPerformed(ActionEvent evt) {
             count--;
             tfCount.setText(count + "");
          }
       });

       btnReset = new Button("Reset");
       add(btnReset);
       btnReset.addActionListener(new ActionListener() {
          @Override
          public void actionPerformed(ActionEvent evt) {
             count = 0;
             tfCount.setText("0");
          }
       });

       setTitle("AWT Counter");
       setSize(400, 100);
       setVisible(true);
    }

    // The entry main method
    public static void main(String[] args) {
       new AWTCounter3Buttons();  // Let the constructor do the job
    }
 }
```

# Using the Same Listener Instance for All the Buttons

```
1  import java.awt.*;
2  import java.awt.event.*;
3
4  // An AWT GUI program inherits the top-level container java.awt.Frame
5  public class AWTCounter3Buttons1Listener extends Frame {
6     private TextField tfCount;
7     private Button btnCountUp, btnCountDown, btnReset;
8     private int count = 0;
9
10    // Constructor to setup the GUI components and event handlers
11    public AWTCounter3Buttons1Listener () {
12       setLayout(new FlowLayout());
13       add(new Label("Counter"));
14       tfCount = new TextField("0", 10);
15       tfCount.setEditable(false);
16       add(tfCount);
17
18       // Construct Buttons
19       btnCountUp = new Button("Count Up");
20       add(btnCountUp);
21       btnCountDown = new Button("Count Down");
22       add(btnCountDown);
23       btnReset = new Button("Reset");
24       add(btnReset);
25
26       // Allocate an instance of the "named" inner class BtnListener.
27       BtnListener listener = new BtnListener();
28       // Use the same listener instance for all the 3 Buttons.
29       btnCountUp.addActionListener(listener);
30       btnCountDown.addActionListener(listener);
31       btnReset.addActionListener(listener);
32
33       setTitle("AWT Counter");
34       setSize(400, 100);
35       setVisible(true);
36    }
37
38    // The entry main method
39    public static void main(String[] args) {
40       new AWTCounter3Buttons1Listener();  // Let the constructor do the job
41    }
42
43    /**
44     * BtnListener is a named inner class used as ActionEvent listener for all the Buttons.
45     */
46    private class BtnListener implements ActionListener {
47       @Override
48       public void actionPerformed(ActionEvent evt) {
49          // Need to determine which button fired the event.
50          // the getActionCommand() returns the Button's label
51          String btnLabel = evt.getActionCommand();
52          if (btnLabel.equals("Count Up")) {
53             ++count;
54          } else if (btnLabel.equals("Count Down")) {
55             --count;
56          } else {
57             count = 0;
58          }
59          tfCount.setText(count + "");
60       }
61    }
62 }
```

# Using getSource() of EventObject

```
43      * BtnListener is a named inner class used as ActionEvent listener for all the Buttons.
44      */
45     private class BtnListener implements ActionListener {
46        @Override
47        public void actionPerformed(ActionEvent evt) {
48           // Need to determine which button has fired the event.
49           Button source = (Button)evt.getSource();
50                 // Get a reference of the source that has fired the event.
51                 // getSource() returns a java.lang.Object. Downcast back to Button.
52           if (source == btnCountUp) {
53              ++count;
54           } else if (source == btnCountDown) {
55              --count;
56           } else {
57              count = 0;
58           }
59           tfCount.setText(count + "");
60        }
```

# Event Listener's Adapter Classes

**Using WindowListener Interface**

- Refer to the WindowEventDemo, a WindowEvent listener is required to implement the WindowListener interface, which declares 7 abstract methods.
- Although we are only interested in windowClosing(), we need to provide an empty body to the other 6 abstract methods in order to compile the program.
- This is tedious, e.g., we can rewrite the WindowEventDemo using an inner class implementing ActionListener as follows:

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

// An AWT GUI program inherits the top-level container java.awt.Frame
public class WindowEventDemoWithInnerClass extends Frame {
   private TextField tfCount;
   private Button btnCount;
   private int count = 0;

   // Constructor to setup the GUI components and event handlers
   public WindowEventDemoWithInnerClass () {
      setLayout(new FlowLayout());
      add(new Label("Counter"));
      tfCount = new TextField("0", 10);
      tfCount.setEditable(false);
      add(tfCount);

      btnCount = new Button("Count");
      add(btnCount);
      btnCount.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent evt) {
            ++count;
            tfCount.setText(count + "");
         }
      });
```

# Event Listener's Adapter Classes

```
      // Allocate an anonymous instance of an anonymous inner class
      // that implements WindowListener.
      // "super" Frame adds this instance as WindowEvent listener.
      addWindowListener(new WindowListener() {
         @Override
         public void windowClosing(WindowEvent evt) {
            System.exit(0);  // terminate the program
         }
         // Need to provide an empty body for compilation
         @Override public void windowOpened(WindowEvent evt) { }
         @Override public void windowClosed(WindowEvent evt) { }
         @Override public void windowIconified(WindowEvent evt) { }
         @Override public void windowDeiconified(WindowEvent evt) { }
         @Override public void windowActivated(WindowEvent evt) { }
         @Override public void windowDeactivated(WindowEvent evt) { }
      });

      setTitle("WindowEvent Demo");
      setSize(250, 100);
      setVisible(true);
   }

   // The entry main method
   public static void main(String[] args) {
      new WindowEventDemoWithInnerClass();   // Let the constructor do the job
   }
}
```

## Using WindowAdapter Superclass

- An adapter class called WindowAdapter is therefore provided, which implements the WindowListener interface and provides default implementations to all the 7 abstract methods.
- You can then derive a subclass from WindowAdapter and override only methods of interest and leave the rest to their default implementation. For example,

# Event Listener's Adapter Classes

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

// An AWT GUI program inherits the top-level container java.awt.Frame
public class WindowEventDemoAdapter extends Frame {
   private TextField tfCount;
   private Button btnCount;
   private int count = 0;

   // Constructor to setup the GUI components and event handlers
   public WindowEventDemoAdapter () {
      setLayout(new FlowLayout());
      add(new Label("Counter"));
      tfCount = new TextField("0", 10);
      tfCount.setEditable(false);
      add(tfCount);

      btnCount = new Button("Count");
      add(btnCount);
      btnCount.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent evt) {
            ++count;
            tfCount.setText(count + "");
         }
      });

      // Allocate an anonymous instance of an anonymous inner class
      // that extends WindowAdapter.
      // "super" Frame adds the instance as WindowEvent listener.
      addWindowListener(new WindowAdapter() {
         @Override
         public void windowClosing(WindowEvent evt) {
            System.exit(0);  // Terminate the program
         }
      });

      setTitle("WindowEvent Demo");
      setSize(250, 100);
      setVisible(true);
   }

   /** The entry main method */
   public static void main(String[] args) {
      new WindowEventDemoAdapter();   // Let the constructor do the job
   }
}
```

Clearly, the adapter greatly simplifies the codes.

# Other Event-Listener Adapter Classes

- Similarly, adapter classes such as MouseAdapter, MouseMotionAdapter, KeyAdapter, FocusAdapter are available for MouseListener, MouseMotionListener, KeyListener, and FocusListener, respectively.
- There is no ActionAdapter for ActionListener, because there is only one abstract method (i.e. actionPerformed()) declared in the ActionListener interface. This method has to be overridden and there is no need for an adapter.

# Layout Manager

- A container has a so-called layout manager to arrange its components.
- AWT provides the following layout managers (in package java.awt): FlowLayout, GridLayout, BorderLayout, GridBagLayout, BoxLayout, CardLayout, and others.
- A container has a setLayout() method to set its layout manager
- For Example

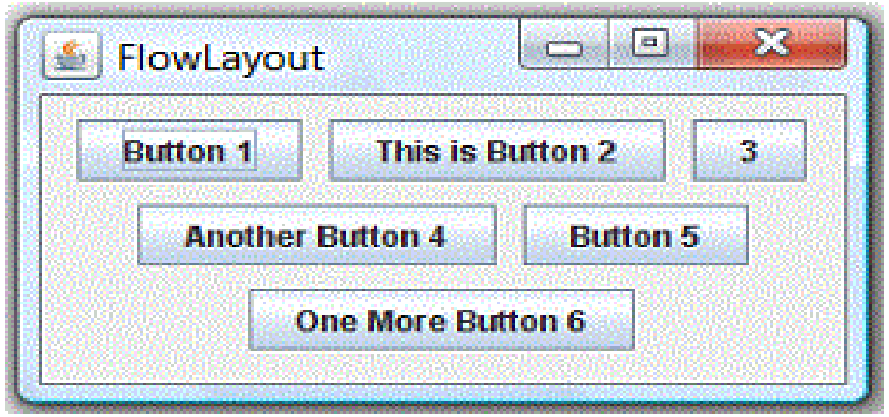
```
Panel pnl = new Panel();
pnl.setLayout(new FlowLayout());
```

- You can get the current layout via Container's getLayout() method.
- For Example

```
Panel pnl = new Panel();
 System.out.println(pnl.getLayout());
```

# Layout Manager: FlowLayout

- In the java.awt.FlowLayout, components are arranged from left-to-right inside the container in the order that they are added (via method aContainer.add(aComponent)).
- When one row is filled, a new row will be started.
- The actual appearance depends on the width of the display window.

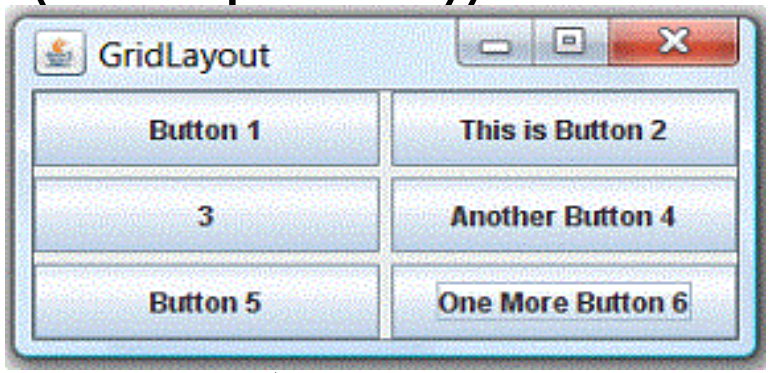



**Constructors**

- public FlowLayout();
- public FlowLayout(int alignment);
- public FlowLayout(int alignment, int hgap, int vgap);

# Layout Manager: GridLayout

- In java.awt.GridLayout, components are arranged in a grid (matrix) of rows and columns inside the Container.
- Components are added in a left-to-right, top-to-bottom manner in the order they are added (via method aContainer.add(aComponent)).




**Constructors**

- public GridLayout(int rows, int columns);
- public GridLayout(int rows, int columns, int hgap, int vgap);
- // By default: rows = 1, cols = 0, hgap = 0, vgap = 0

# Layout Manager: BorderLayout

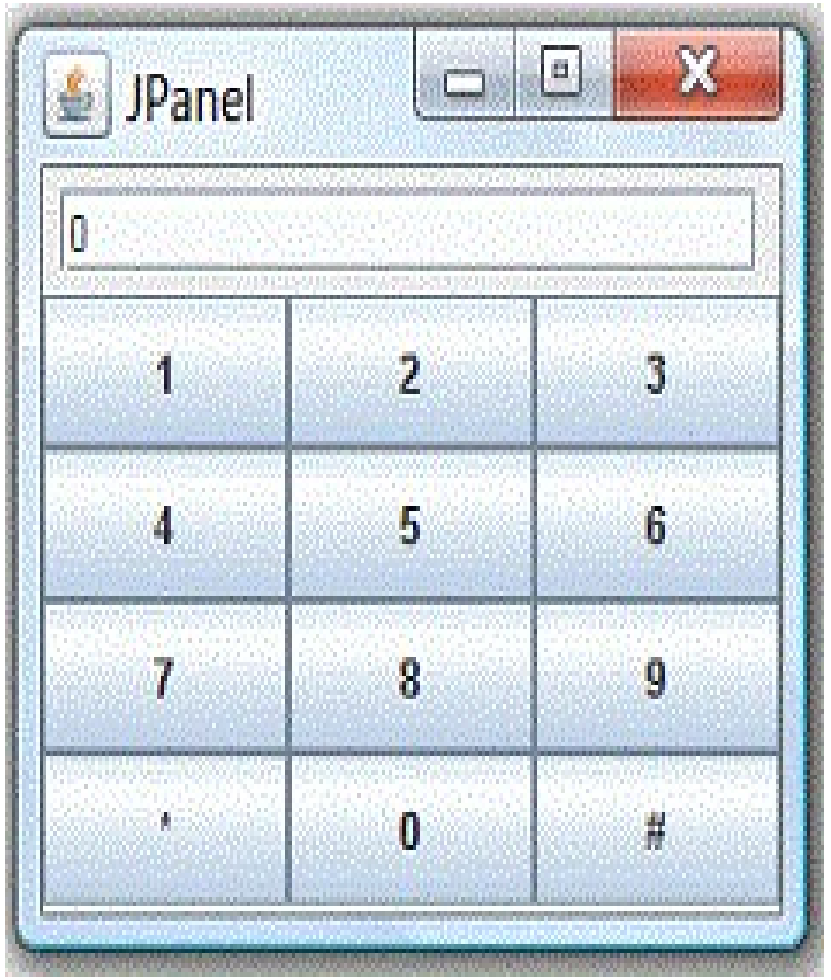
- In java.awt.BorderLayout, the container is divided into 5 zones: EAST, WEST, SOUTH, NORTH, and CENTER.
- Components are added using method aContainer.add(aComponent, zone), where zone is either BorderLayout.NORTH (or PAGE_START), BorderLayout.SOUTH (or PAGE_END), BorderLayout.WEST (or LINE_START), BorderLayout.EAST (or LINE_END), or BorderLayout.CENTER.
- You need not place components to all the 5 zones.
- The NORTH and SOUTH components may be stretched horizontally;
- the EAST and WEST components may be stretched vertically; the CENTER component may stretch both horizontally and vertically to fill any space left over.

**Constructors**

- public BorderLayout();
- public BorderLayout(int hgap, int vgap);
- // By default hgap = 0, vgap = 0

# Using Panels as Sub-Container to Organize Components

- An AWT Panel is a rectangular pane, which can be used as sub-container to organized a group of related components in a specific layout (e.g., FlowLayout, BorderLayout).
- Panels are secondary containers, which shall be added into a top-level container (such as Frame), or another Panel.
- For example, the figure shows a Frame in BorderLayout containing two Panels - panelResult in FlowLayout and panelButtons in GridLayout.
- panelResult is added to the NORTH, and panelButtons is added to the CENTER.



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

// An AWT GUI program inherits the top-level container java.awt.Frame
public class AWTPanelDemo extends Frame {
   private Button[] btnNumbers;  // Array of 10 numeric Buttons
   private Button btnHash, btnStar;
   private TextField tfDisplay;

   // Constructor to setup GUI components and event handlers
   public AWTPanelDemo () {
      // Set up display panel
      Panel panelDisplay = new Panel(new FlowLayout());
      tfDisplay = new TextField("0", 20);
      panelDisplay.add(tfDisplay);

      // Set up button panel
      Panel panelButtons = new Panel(new GridLayout(4, 3));
      btnNumbers = new Button[10];  // Construct an array of 10 numeric Buttons
      btnNumbers[1] = new Button("1");  // Construct Button "1"
      panelButtons.add(btnNumbers[1]);  // The Panel adds this Button
      btnNumbers[2] = new Button("2");
      panelButtons.add(btnNumbers[2]);
      btnNumbers[3] = new Button("3");
      panelButtons.add(btnNumbers[3]);
      btnNumbers[4] = new Button("4");
      panelButtons.add(btnNumbers[4]);
      btnNumbers[5] = new Button("5");
      panelButtons.add(btnNumbers[5]);
      btnNumbers[6] = new Button("6");
      panelButtons.add(btnNumbers[6]);
      btnNumbers[7] = new Button("7");
      panelButtons.add(btnNumbers[7]);
      btnNumbers[8] = new Button("8");
      panelButtons.add(btnNumbers[8]);
      btnNumbers[9] = new Button("9");
      panelButtons.add(btnNumbers[9]);
            // You should use a loop for
      btnStar = new Button("*");
      panelButtons.add(btnStar);
      btnNumbers[0] = new Button("0");
      panelButtons.add(btnNumbers[0]);
      btnHash = new Button("#");
      panelButtons.add(btnHash);

      setLayout(new BorderLayout());  // "super" Frame sets to BorderLayout
      add(panelDisplay, BorderLayout.NORTH);
      add(panelButtons, BorderLayout.CENTER);

      setTitle("BorderLayout Demo"); // "super" Frame sets title
      setSize(200, 200);             // "super" Frame sets initial size
      setVisible(true);              // "super" Frame shows
   }

   // The entry main() method
   public static void main(String[] args) {
      new AWTPanelDemo();  // Let the constructor do the job
   }
}
```

# Exercise

- Re-create all the AWT examples using Adapter classes.

# Any Question

?

# What you have learned?