Using Standard Java Packages

Firoz Ahmed

Associate Professor

Department of ICE, RU

A package is a namespace that organizes a set of related classes and interfaces

A **Java package** is a mechanism for organizing Java Classes into namespaces

Java packages can be stored in compressed files called JAR files

Outlines of Presentation

- Creating Graphical User Interfaces with AWT
- Managing Graphics Objects with GUI Layout Managers
- Event Handling of Various Components

Creating Graphical User Interfaces with AWT

- AWT Packages
- Containers and Components
- AWT Container Classes
- AWT Component Classes
- Example

AWT Packages

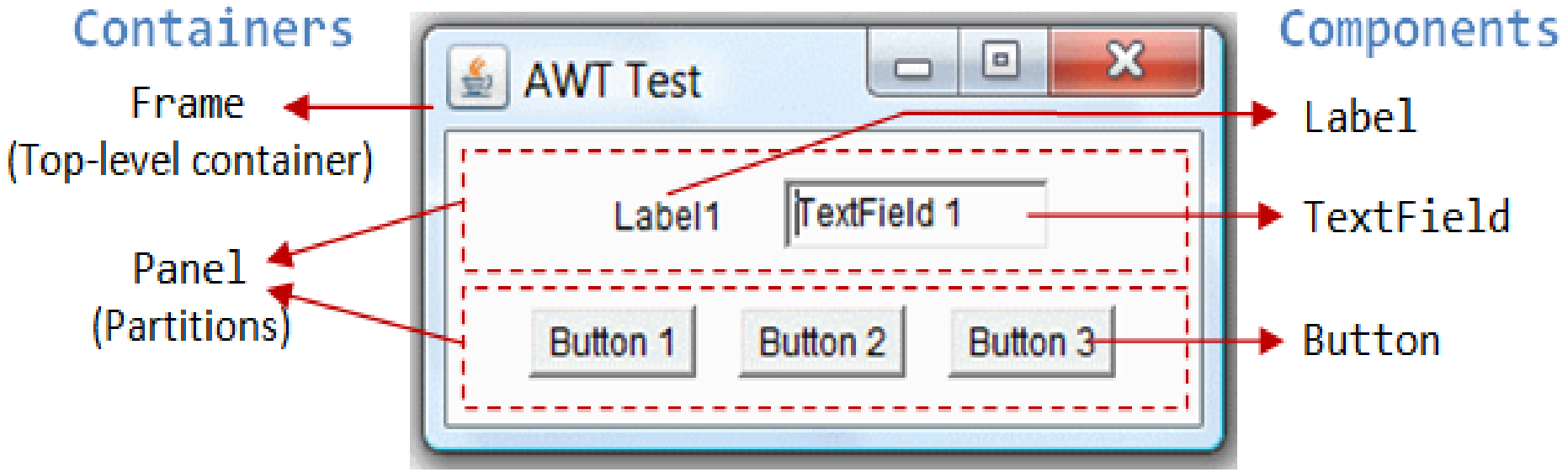
- Abstract Window Toolkit (AWT) is a set of Application Program Interfaces (API s) used by Java programmers to create Graphical User Interface(GUI) objects, such as *buttons, scroll bars,* and *windows*
- AWT provides a platform-independent and device-independent interface to develop graphic programs that runs on all platforms
  - Windows, Mac, Unix, etc.
- AWT is huge! It consists of 12 packages
- Fortunately, only 2 packages are commonly-used
  - java.awt
  - java.awt.event

AWT Packages

- 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*, *Panel*, *Dialog* and *ScrollPane*)
  - 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 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*, *Panel* and *Applet*) are used to hold components in a specific layout



Containers and Components

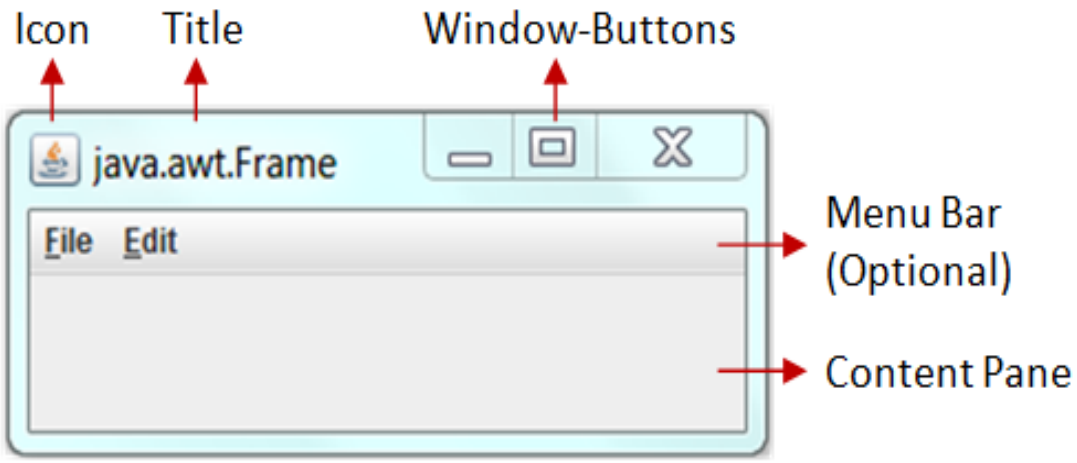
- A component must be kept in a container.
- Every container has a method called add(Component c)

```
Panel panel = new Panel();     // Panel is a Container
Button btn = new Button();     // Button is a Component
panel.add(btn);                // The Panel Container adds a Button Component
```

AWT Container Classes

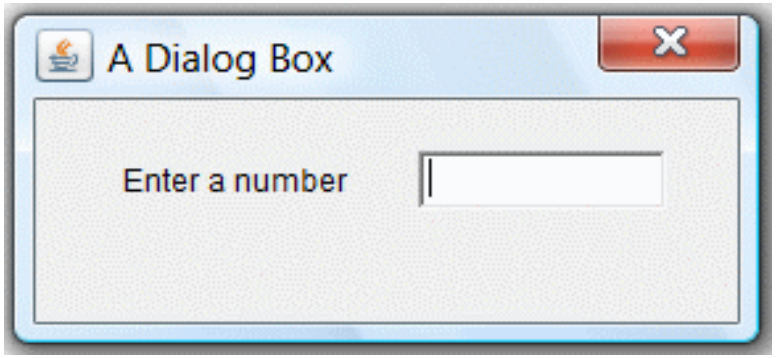
- Top-level containers
  - Each GUI program has a top-level container such as *Frame, Dialog and Applet*
- Secondary containers
  - They are placed inside a top-level container such as *Panel and ScrollPane*

Top-level containers



A Frame provides main window

- A title bar
- An optional menu bar,
- The content display area.



An AWT Dialog is a *"pop-up window"* used for interacting with the users.

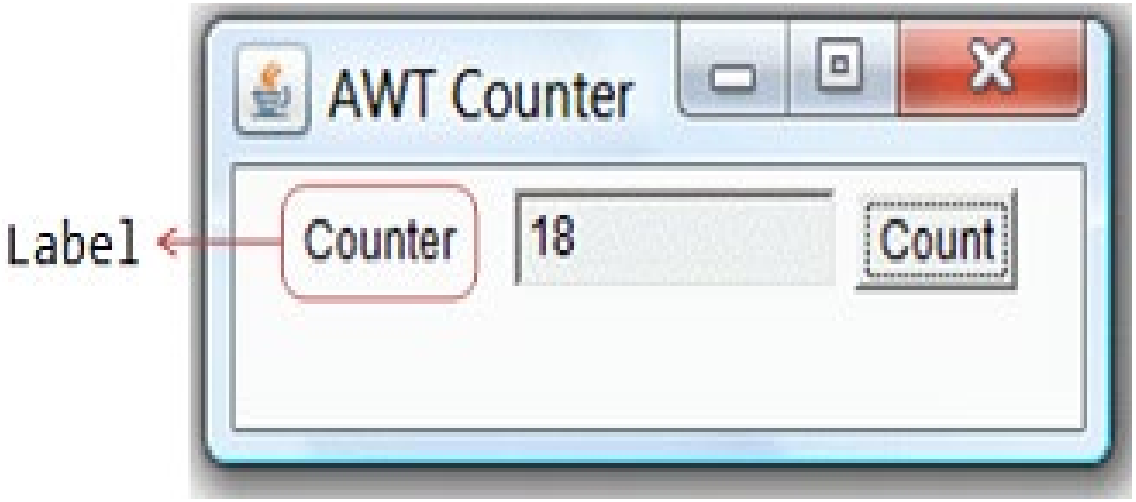
- A title bar
- The content display area.

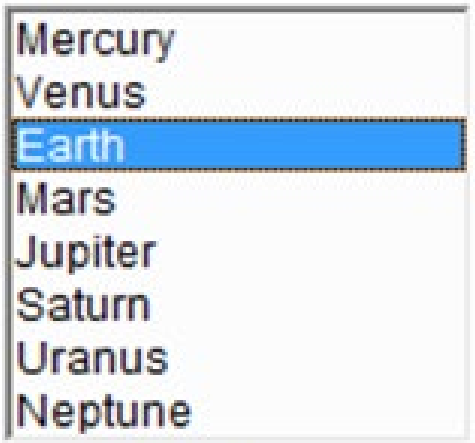
An AWT Applet is the top-level container for an applet, which is a Java program running inside a browser.

Secondary containers

- A Panel is a rectangular box under a container, used to *layout* a set of related GUI components
- ScrollPane (which provides automatic horizontal and/or vertical scrolling for a single child component)

AWT Component Classes





Example

```
import java.awt.* ;
public class hello
   {
      public static void main(String args[ ] )
           {
              Frame f = new Frame ("My Frame");
              Button b = new Button("OK");
              TextField tf = new TextField("Programming in Java", 20);
              f.setLayout(new FlowLayout());
              f.add(b);
              f.add(tf);
              f.setSize(300, 300);
              f.setVisible(true);
           }
   }
```

Example

```
import java.awt.* ;
import java.awt.event.* ;
public class hello
{
public static void main(String args[ ] )
{
Frame f = new Frame ("My Frame");
MyGuiAction ga = new MyGuiAction(f);
}
}
class MyGuiAction implements
ActionListener
{
static int count = 0;
Button b;
TextField tf;
MyGuiAction(Frame f)
{
b = new Button("OK");
b.addActionListener(this);
tf = new TextField("Hello Java", 20);
f.setLayout(new FlowLayout());
f.add(b);
f.add(tf);
f.setSize(300, 300);
f.setVisible(true);
}
public void actionPerformed( ActionEvent e)
{
if(e.getSource() == b)
{
count++ ;
System.out.println("Button is Pressed");
tf.setText("Hello Java Click "+ count);
}
}
}
```

Outlines of Presentation

- Creating Graphical User Interfaces with AWT
- Managing Graphics Objects with GUI Layout Managers
- Event Handling of Various Components

Layout Managers

- Java's layout managers provide a level of abstraction to automatically map your user interface on all window systems
- The UI components are placed in containers
- Each container has a layout manager to arrange the UI components within the container
- A container has a setLayout() method to set its layout manager

```
// java.awt.Container
  public void setLayout(LayoutManager mgr)
```

Layout Managers

- There are several layout manager classes in the AWT
- The Java platform supplies six commonly used layout managers
  - Flow Layout
  - Grid Layout
  - Border Layout

  } Basic Layout Managers

  - Box Layout
  - Card Layout
  - GridBag Layout

  } Advanced Layout Managers

Flow Layout

- It is the default layout manager for panels and applets.
- It always arranges the components in horizontal rows.
- The components always appear left to right in the order.

- To change this default behavior , we can use setLayout( )
  - setLayout(new FlowLayout(FlowLayout.RIGHT))

Flow Layout

public FlowLayout();
public FlowLayout(int *align*);
public FlowLayout(int *align*, int *hgap*, int *vgap*);

// *align*: FlowLayout.LEFT (or LEADING),
FlowLayout.RIGHT (or TRAILING), or
FlowLayout.CENTER
// *hgap*, *vgap*: horizontal/vertical gap between the components

// By default: hgap=5, vgap=5, align=CENTER

Example (Flow Layout)

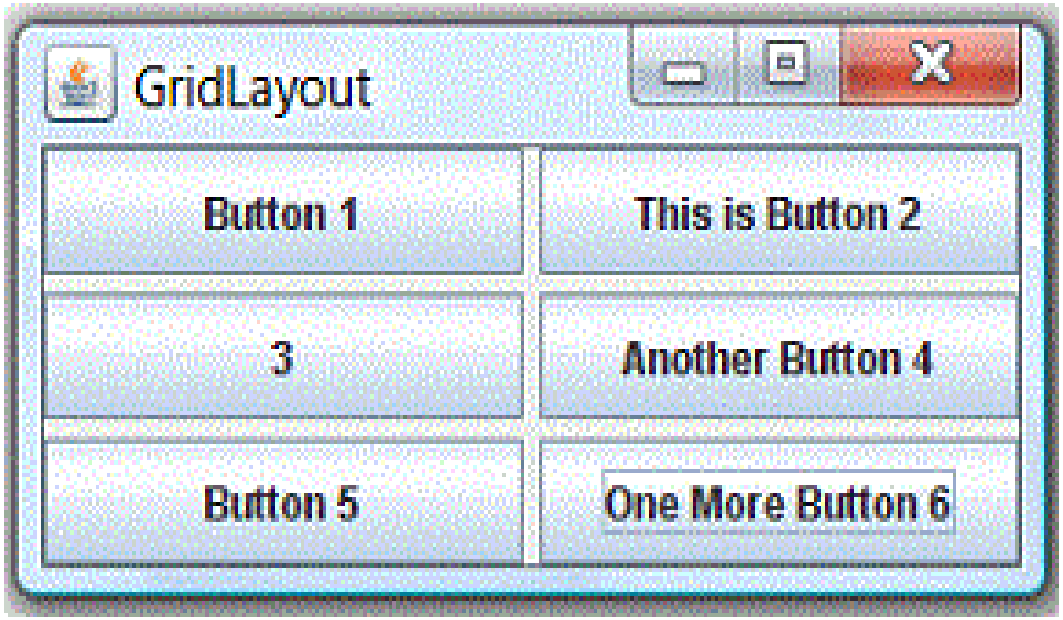
```
import java.awt.*;
import java.awt.event.*;
 // An AWT GUI program inherits the
//top-level container java.awt.Frame
public class AWTFlowLayout extends Frame
{
/** Constructor to setup GUI components */
public AWTFlowLayout () {
     setLayout(new FlowLayout());
// Frame sets layout to FlowLayout,
//which arranges the components
// from left-to-right, and flow from
//top-to-bottom.
     add(new Button("Button 1"));
     add(new Button("This is Button 2"));
     add(new Button("3"));
     add(new Button("Another Button 4"));
```

```
add(new Button("Button 5"));
add(new Button("One More Button 6"));
setTitle("FlowLayout");
// Frame sets title
setSize(280, 150);
// Frame sets initial size
setVisible(true);
 // "this" Frame shows
  }

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

Grid Layout

- In 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
- Every component in the applet in this case is exactly the same size



Grid Layout

public GridLayout (int *rows*, int *columns*);
public GridLayout (int *rows*, int *columns*, int *hgap*, int *vgap*);

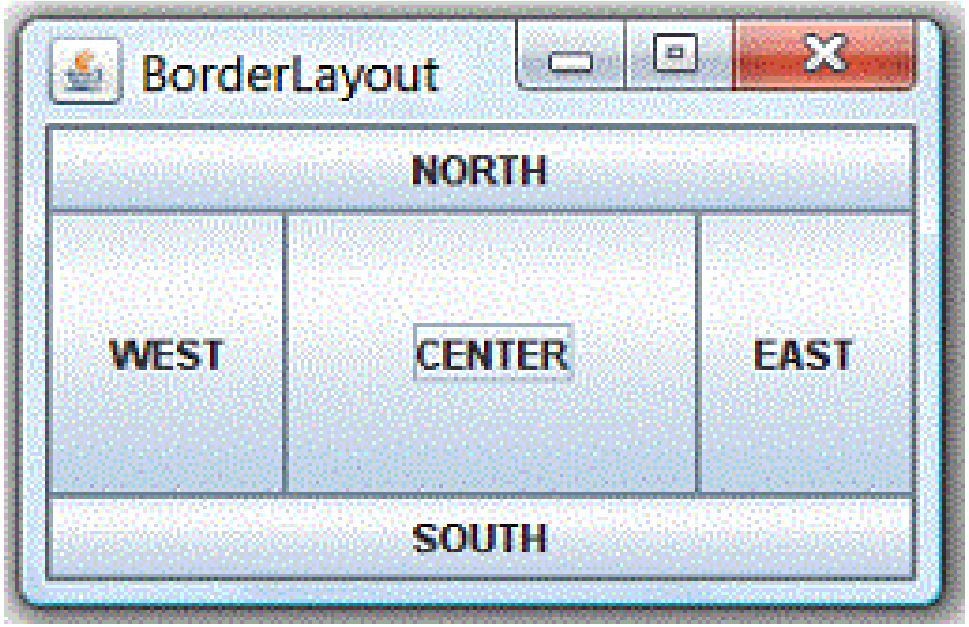
// By default: rows=1, cols=0, hgap=0, vgap=0

Example (Grid Layout)

```
import java.awt.*;
import java.awt.event.*;
public class AWTGridLayout extends Frame
{
public AWTGridLayout ()
{
    setLayout(new GridLayout(3, 2, 3, 3));
     add(new Button("Button 1"));
     add(new Button("This is Button 2"));
     add(new Button("3"));
     add(new Button("Another Button 4"));
     add(new Button("Another Button 4"));
     add(new Button("Button 5"));
     add(new Button("One More Button 6"));
     setTitle("GridLayout");
     setSize(280, 150);
     setVisible(true);
     }
      public static void main(String[] args) {
     new AWTGridLayout();
     }
     }
```

Border Layout

- In BorderLayout, the container is divided into 5 zones: EAST, WEST, SOUTH, NORTH, and CENTER
- A component at east or west extends vertically up to the bottom of the North component or to the top of the container (if there is no North component)
- Similarly for south component



Border Layout

public BorderLayout();
public BorderLayout(int *hgap*, int *vgap*);

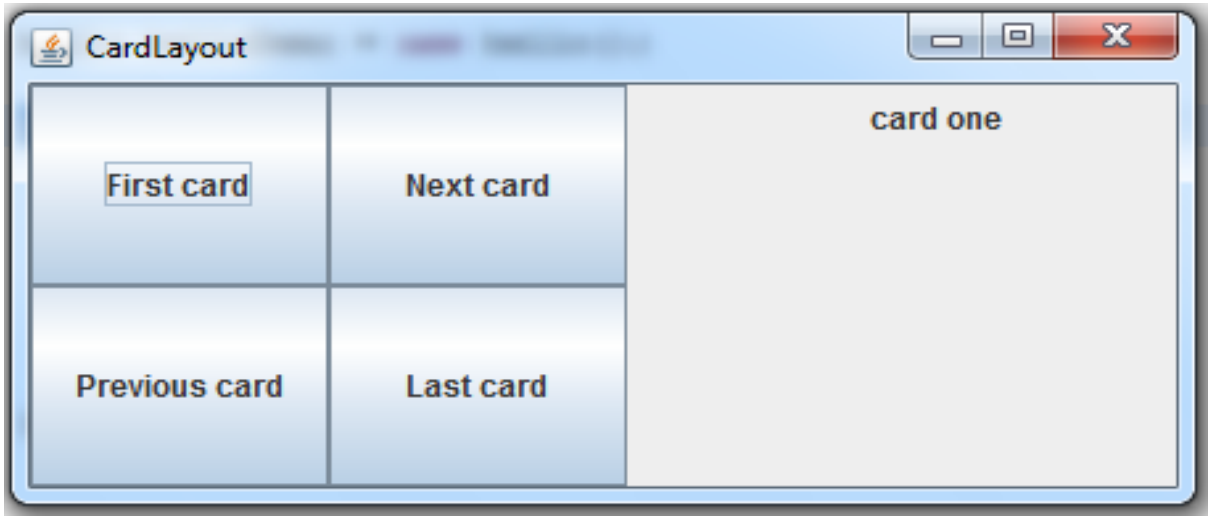
// By default: hgap=0, vgap=0

Example (Border Layout)

```
import java.awt.*;
import java.awt.event.*;
public class AWTBorderLayout extends Frame{
public AWTBorderLayout () {
setLayout(new BorderLayout(3, 3));
add(new Button("NORTH"),
            BorderLayout.NORTH);
add(new Button("SOUTH"),
            BorderLayout.SOUTH);
add(new Button("CENTER"),
            BorderLayout.CENTER);
add(new Button("EAST"),
            BorderLayout.EAST);
add(new Button("WEST"),
            BorderLayout.WEST);
setTitle("BorderLayout");
setSize(280, 150);
setVisible(true);
}
public static void main(String[] args) {
 new AWTBorderLayout();
}}
```

Card Layout

- A CardLayout object is a layout manager for a container, such as a panel
- Conceptually, each component that a CardLayout manages is like a playing card or trading card in a stack, where only the top card is visible at any time
- The CardLayout class manages two or more components that share the same display space



Example (Card Layout)

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class CardDeck extends JFrame
implements ActionListener {
  private CardLayout cardManager;
  private JPanel deck;
  private JButton controls[];
  private String names[] = { "First card", "Next
card", "Previous card", "Last card" };
  public CardDeck(){
    super( "CardLayout " );
     Container c = getContentPane();
deck = new JPanel();
    cardManager = new CardLayout();
    deck.setLayout( cardManager );
JLabel label1 = new JLabel( "card one",
SwingConstants.CENTER );
 JPanel card1 = new JPanel();
 card1.add( label1 );
```

```
deck.add( card1, label1.getText() ); JLabel
label2 = new JLabel( "card two",
SwingConstants.CENTER );
    JPanel card2 = new JPanel();
    card2.setBackground( Color.yellow );
    card2.add( label2 );
    deck.add( card2, label2.getText() );
JLabel label3 = new JLabel( "card three" );
    JPanel card3 = new JPanel();
card3.setLayout( new BorderLayout() );
    card3.add( new JButton( "North" ),
BorderLayout.NORTH );
    card3.add( new JButton( "West" ),
BorderLayout.WEST );
card3.add( new JButton( "East" ),
BorderLayout.EAST );
    card3.add( new JButton( "South" ),
BorderLayout.SOUTH );
    card3.add( label3,
BorderLayout.CENTER );
```

Example (Card Layout)

```
deck.add( card3, label3.getText() );
      JPanel buttons = new JPanel();
      buttons.setLayout( new GridLayout( 2, 2 ) );
      controls = new JButton[ names.length ];
      for ( int i = 0; i < controls.length; i++ ) {
        controls[ i ] = new JButton( names[ i ] );
        controls[ i ].addActionListener( this );
        buttons.add( controls[ i ] );
      }
      c.add( buttons, BorderLayout.WEST );
      c.add( deck, BorderLayout.EAST );
      setSize( 450, 200 );
      show();
   }
   public void actionPerformed( ActionEvent e )
{
 if ( e.getSource() == controls[ 0 ] )
        cardManager.first( deck ); // show first card
      else if ( e.getSource() == controls[ 1 ] )
        cardManager.next( deck );  // show next card
else if ( e.getSource() == controls[ 2 ] )
        cardManager.previous( deck );  //
show previous card
      else if ( e.getSource() == controls[ 3 ] )
        cardManager.last( deck );  // show last
card
   }
   public static void main( String args[] )
   {
     CardDeck cardDeckDemo = new
CardDeck();

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

Frame and JFrame

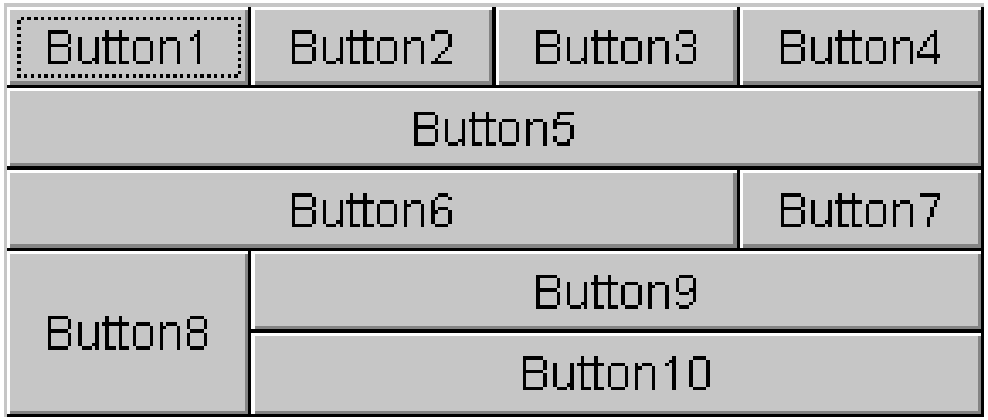
- Frame is part of java.awt package and exists since JDK1.0
- JFrame is part of javax.swing package and exists since JDK1.1.3 or something
- Frame extends Window
- JFrame extends Frame
- We can directly add components to Frame
- We add components to JFrame.getContentPane()

JPanel

- JPanel is a public java swing class which is used to create a general-purpose container
- JPanel panel objects can add color to their background and also can be customized
- JPanel inherits methods from it's super classes namely JComponent, Container, and Component java classes
- JPanel is an extension of java swing JComponent class. JPanel

GridBag Layout

- The GridBagLayout class is a flexible layout manager that aligns components vertically and horizontally, without requiring that the components be of the same size
- Each GridBagLayout object maintains a dynamic, rectangular grid of cells, with each component occupying one or more cells
- The position and behavior of each element is specified by an instance of the GridBagConstraints class



GridBag Layout

```
JPanel pane = new JPanel(new GridBagLayout());
GridBagConstraints c = new GridBagConstraints();
pane.add(theComponent, c);
```

Example (GridBag Layout)

```
import java.applet.*;
import java.awt.*;
public class MyGridBag extends Applet{
    TextArea ObjTa;
    TextField ObjTf;
    Button butta, buttf;
    CheckboxGroup cbg;
    Checkbox cbbold, cbitalic, cbplain, cbboth;
    GridBagLayout gb;
    GridBagConstraints gbc;
    public void init(){
        gb = new GridBagLayout();
        setLayout(gb);
gbc = new GridBagConstraints();
        ObjTa = new TextArea("TextArea ", 5, 10);
        ObjTf = new TextField("enter your Name");
        butta = new Button("TextArea");
        buttf = new Button("TextField");
 cbg = new CheckboxGroup();
cbbold = new Checkbox("Bold", cbg, false);
cbitalic = new Checkbox("Italic", cbg, false);
cbplain = new Checkbox("Plain", cbg, false);
cbboth = new Checkbox("Bold/Italic", cbg, true);
gbc.fill = GridBagConstraints.BOTH;
```

```
addComponent(ObjTa, 0,0,4,1);
 gbc.fill = GridBagConstraints.HORIZONTAL;
        addComponent(butta, 0,1,1,1);
 gbc.fill = GridBagConstraints.HORIZONTAL;
        addComponent(buttf, 0,2,1,1);
  gbc.fill = GridBagConstraints.HORIZONTAL;
        addComponent(cbbold, 2,1,1,1);
  gbc.fill = GridBagConstraints.HORIZONTAL;
        addComponent(cbitalic, 2,2,1,1);
  gbc.fill = GridBagConstraints.HORIZONTAL;
        addComponent(cbplain, 3,1,1,1);
   gbc.fill = GridBagConstraints.HORIZONTAL;
   addComponent(cbboth, 3,2,1,1);
        gbc.fill = GridBagConstraints.HORIZONTAL;
        addComponent(ObjTf, 4,0,1,3);    }
 public void addComponent(Component comp, int row, int
col, int nrow, int ncol){
        gbc.gridx = col;
        gbc.gridy = row;
        gbc.gridwidth = ncol;
        gbc.gridheight = nrow;
        gb.setConstraints(comp,gbc);
        add(comp); }
}
```

Outlines of Presentation

- Creating Graphical User Interfaces with AWT
- Managing Graphics Objects with GUI Layout Managers
- Event Handling of Various Components

Events and Listeners

- In *event-driven programming*, code is executed upon *activation of events*
- An *event* can be defined as a type of signal to the program that something has happened
- The event is generated by external user actions such as:
  - Moving the mouse
  - Clicking the button
  - Pressing a key
  - Sliding the scrollbar thumb
  - Choosing an item from a menu
- Events are responded to by event *listeners*

Event Handling Model

- To process an event
  - Register an event listener
  - Implement event handler
    - Method that is called in response to an event
    - Each event handling interface has one or more event handling methods that must be defined

The Event Handling process

- When an event is triggered, the JAVA runtime first determines its source and type
- If a listener for this type of event is registered with the source, an event object is created
- For each listener to this type of an event
  - The JAVA runtime invokes the appropriate event handling method to the listener and passes the event object as the parameter

Selected User Actions

User Action	Source Object	Event Type Generated
Click a button	Jbutton	ActionEvent
Select a new item	JComboBox	ItemEvent, ActionEvent
Select an item from a List	Jlist	ListSelectionEvent
Window opened, closed	Window	WindowEvent
Mouse pressed, released	MouseEvent	MouseEvent
Key released, pressed	KeyEvent	KeyEvent

Java AWT Event Listener Interfaces

- ActionListener
- ItemListener
- KeyListener
- MouseListener
- WindowListener
- ListSelectionListener

EventListner and ActionListener

- An Event Listener, once set to an applet object waits for some action to be performed on it
  - It mouse click, mouse hover, pressing of keys, click of button, etc
- *ActionListener* is an interface that could be implemented in order to determine how certain event should be handled
- When implementing an interface, all methods in that interface should be implemented, *ActionListener* interface has one method to implement named *actionPerformed()*

Selected Event Handlers

Event Class	Listener Interface	Listener Methods (Handlers)
ActionEvent	ActionListener	actionPerformed(ActionEvent)
ItemEvent	ItemListener	itemStateChanged(ItemEvent)
ListSelection Event	ListSelection Listener	valueChanged (ListSelectionEvent)

How to Implement a Listener Interface

- Use the **implements** keyword in the class declaration
- Register the object as a listener for a component's event, using the component's *addXListener* method. (where X is the type of event).
- Declare and fully define all methods for the interface that you are implementing

Example

```
import java.awt.* ;
import java.awt.event.* ;
public class hello
{
public static void main(String args[ ] )
{
Frame f = new Frame ("My Frame");
MyGuiAction ga = new MyGuiAction(f);
}
}
class MyGuiAction implements
ActionListener
{
static int count = 0;
Button b;
TextField tf;
MyGuiAction(Frame f)
{
b = new Button("OK");
b.addActionListener(this);
tf = new TextField("Hello Java", 20);
f.setLayout(new FlowLayout());
f.add(b);
f.add(tf);
f.setSize(300, 300);
f.setVisible(true);
}
public void actionPerformed( ActionEvent e)
{
if(e.getSource() == b)
{
count++ ;
System.out.println("Button is Pressed");
tf.setText("Hello Java Click "+ count);
}
}
}
```

Three Steps of Event Handling

- Prepare to accept events

  import package java.awt.event

- Start listening for events

  include appropriate methods

- Respond to events

  implement appropriate abstract method

Prepare to accept events

- Import package java.awt.event
- Applet manifests its desire to accept events by promising to "implement" certain methods
- Example:
  - "ActionListener" for Button events
  - "AdjustmentListener" for Scrollbar events

Start listening for events

- To make the applet "listen" to a particular event, include the appropriate "addxxxListener".
- Examples:

  addActionListener(this)

  - shows that the applet is interested in listening to events generated by the pushing of a certain button

  addAdjustmentListener(this)

  - shows that the applet is interested in listening to events generated by the sliding of a certain scroll bar thumb

Respond to events

- The appropriate abstract methods are implemented.
- Example:
  - *actionPerformed()* is automatically called whenever the user clicks the button
    - Thus, implement *actionPerformed()* to respond to the button event
  - *adjustmentValueChanged()* is automatically invoked whenever the user slides the scroll bar thumb
    - So *adjustmentValueChanged()* needs to be implemented
- In *actionPerformed(ActionEvent evt),* ActionEvent is a class in java.awt.event

ActionEvent

- In Java, most components have a special event called an *ActionEvent*
- This is loosely speaking the most common or canonical event for that component
- A good example is a click for a button
- To have any component listen for ActionEvents, you must register the component with an ActionListener

  e.g. *button.addActionListener(new MyAL());*

ActionPerformed

- The actionPerformed method has the following signature:

  void actionPerformed(ActionEvent)

- The object of type ActionEvent passed to the event handler is used to query information about the event
- Some common methods are:
  - getSource()
    - object reference to component generating event
  - getActionCommand()
    - some text associated with event (text on button, etc)

Event Handler Code

```
class MyActionListener implements ActionListener{
   public void actionPerformed(ActionEvent ae){
        JOptionPane.showMessageDialog("I got clicked", null);
   }
}
```

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

public class Sample extends Applet implements
ActionListener{
      TextField text1,text2,output;
      Label label1,label2,label3,title;
      Button button,clear;
      public void init(){
      setLayout(null);
```

```
title = new Label("Addition of Two Numbers");
title.setBounds(80,10,140,20);
add(title);
title.setAlignment(title.CENTER);

      label1 = new Label("Enter Number 1: ");
      label1.setBounds(20,50,100,20);
      add(label1);

      text1 = new TextField(5);
      text1.setBounds(150,50,100,20);
      add(text1);

      label2 = new Label("Enter Number 2: ");
      label2.setBounds(20,90,100,20);
      add(label2);
```

```
text2 = new TextField(5);
text2.setBounds(150,90,100,20);
add(text2);

label3 = new Label("Sum of Two Numbers: ");
label3.setBounds(20,130,130,20);
add(label3);

output = new TextField(5);
output.setBounds(150,130,100,20);
add(output);

button = new Button("Sum");
button.setBounds(150,170,100,20);
add(button);

clear = new Button("Clear");
clear.setBounds(280,170,100,20);
add(clear);
```

```
button.addActionListener(this);
      clear.addActionListener(this);
       }
       public void actionPerformed(ActionEvent ae){
       int num1=Integer.parseInt(text1.getText());
       int num2=Integer.parseInt(text2.getText());
       int sum=num1+num2;
       output.setText(Integer.toString(sum));
}
```