Session 9

Graphical User Interface (GUI)

Implementing GUIs in Java

- The Java Foundation Classes (JFC) are a set of packages encompassing the following APIs:
  - **Abstract Window Toolkit (AWT)**: native GUI components
  - **Swing**: lightweight GUI components

Abstract Window Toolkit (AWT)

Provides basic UI components:

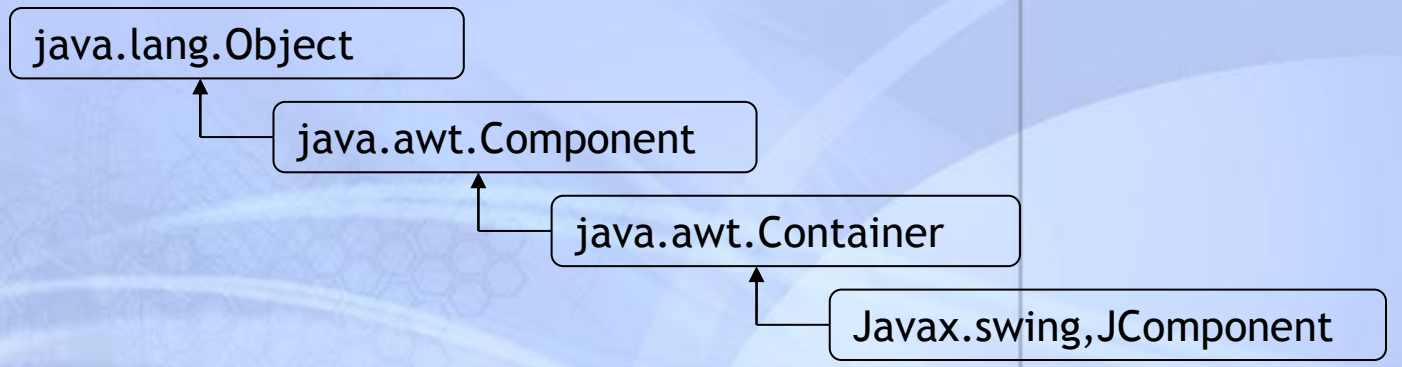
- Buttons, lists, menus, text fields, etc
- Event handling mechanism
- Clipboard and data transfer
- Image manipulation
- Font manipulation
- Graphics

AWT Packages

java.awt	Basic component functionality
java.awt.accessibility	Assistive technologies
java.awt.color	Colors and color spaces
java.awt.datatransfer	Clipboard and data transfer support
java.awt.dnd	Drag and drop
java.awt.event	Event classes and listeners
java.awt.font	2D API font package
java.awt.geom	2D API geometry package
java.awt.im	Input methods
java.awt.image	Fundamental image manipulation classes
java.awt.peer	Peer interfaces for component peers
java.awt.print	2D API support for printing
java.awt.swing	Swing components

Swing

- Common superclasses of many of the Swing components



- **`Component`** class
  - Operations common to most GUI components are found in `Component` class.
- **`Container`** class
  - Two important methods originates in this class
    - `add` — adds components to a `container`.
    - `setLayout` — enables a program to specify the layout manager that helps a `Container` position and size its components.

AWT Vs. Swing

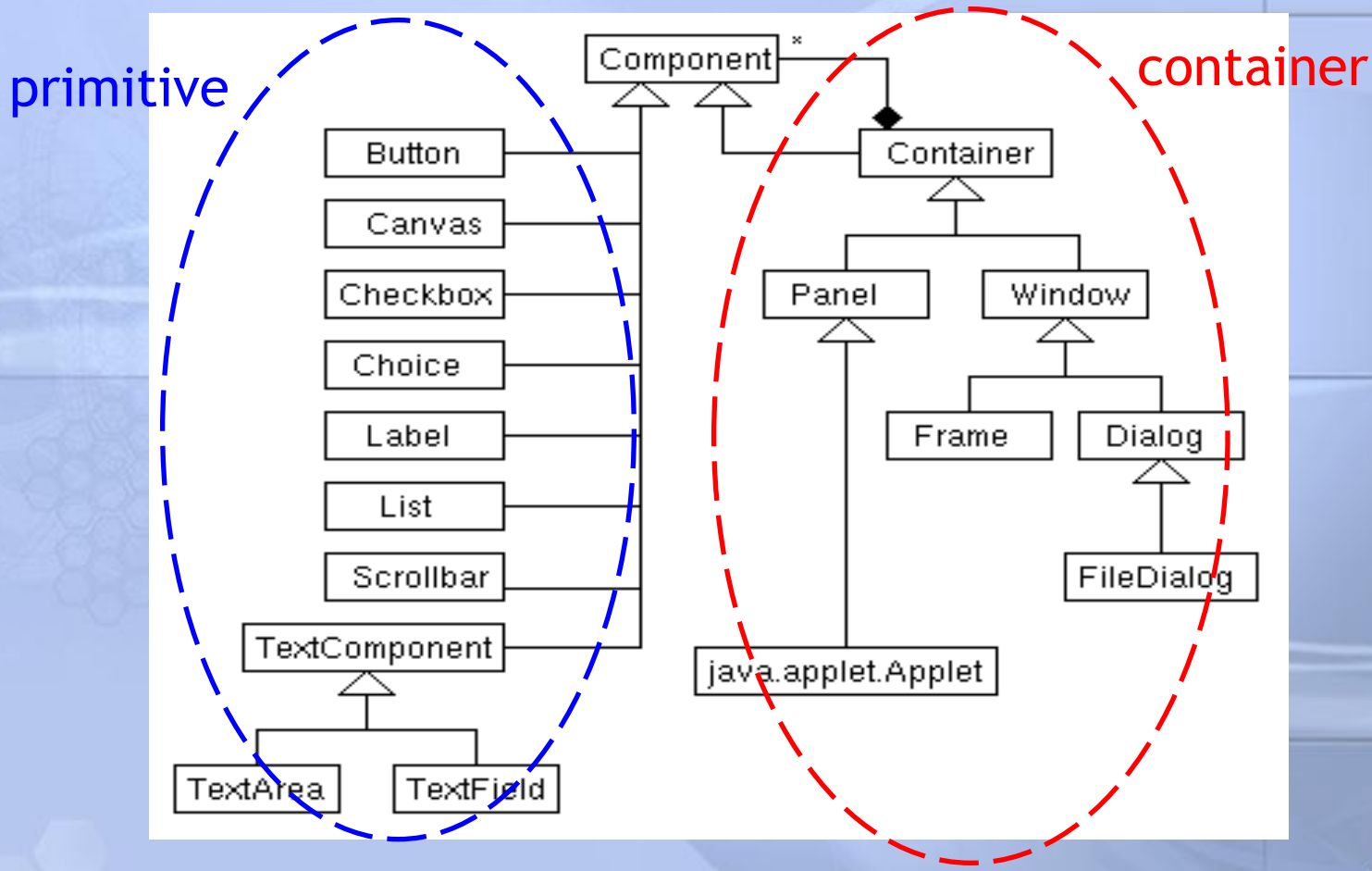
AWT

- *Heavyweight* components
- Associated with native components called *peers*
- Same behaviour, but platform-dependent look
- Package java.awt

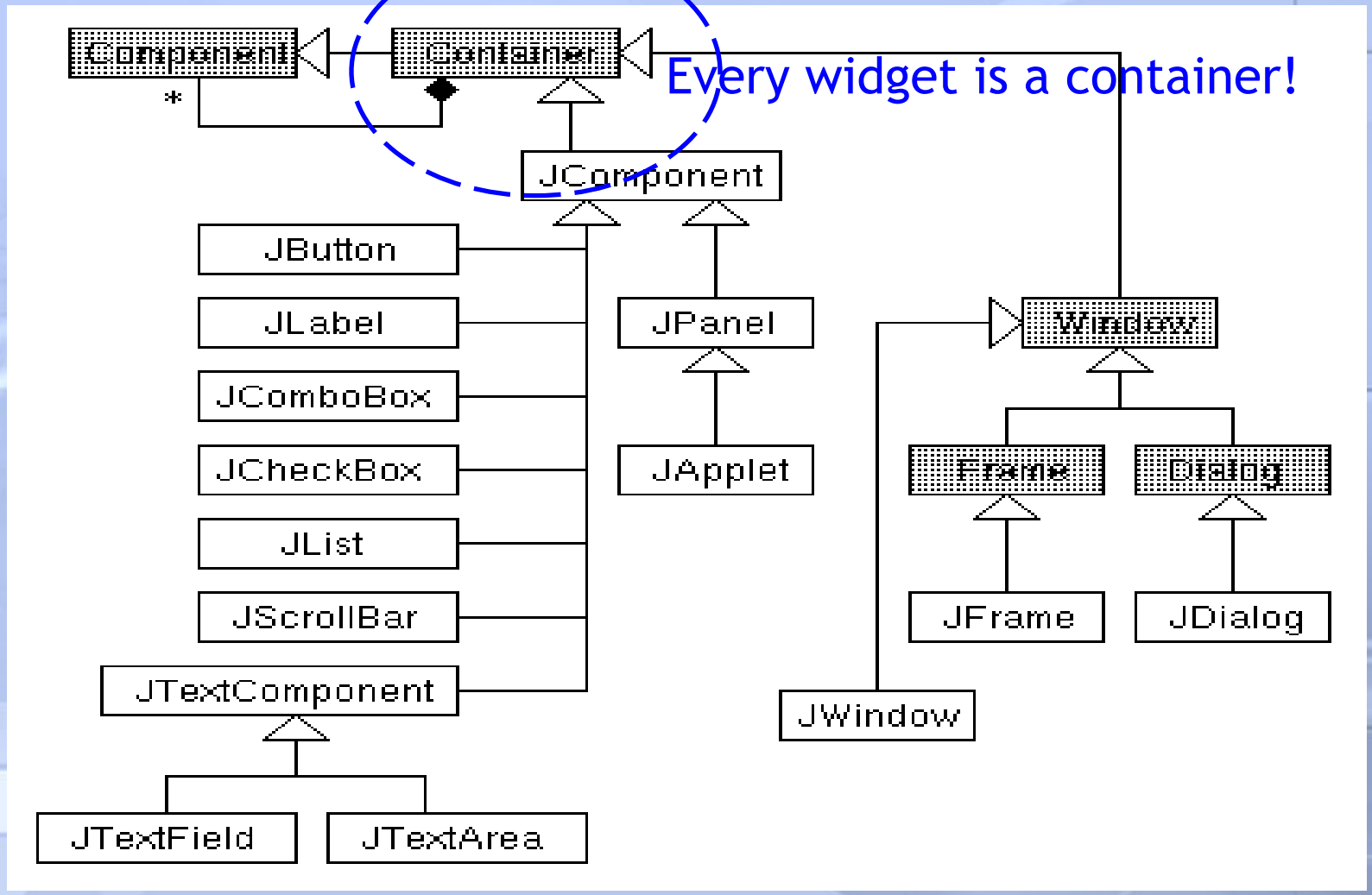
Swing

- *Lightweight* components, i.e., no peer components
- *Same look and feel* across platforms
- Support *pluggable* look and feel
- Package javax.swing

AWT Components



Swing Components



Containers

A **Container** maintains a list of Components that it manipulates as well as a **LayoutManager** to determine how components are displayed.

The **Container** class contains the following subclasses:

Window

Frame

Dialog

Panel

Applet

ScrollPane

Window

A **Window** is a 2-dimensional drawing surface that can be displayed on an output device.

A **Window** can be stacked on top of other Windows and can be moved to the front or back of the visible windows.

Methods in class Window include:

show ( )	Make the Window visible
toFront( )	Move Window to front
toBack( )	Move window to back

Frame

A **Frame** is a type of Window with a title bar, a menu bar, a border, and a cursor. It is used most frequently to display **applications** that contain a graphical user interface or an animation.

Methods defined in **Frame** include:

setTitle(String)	
getTitle(String)	
setCursor(int)	
setResizable( )	make the window resizable
setMenuBar(MenuBar)	include the menu bar for the window

Panel

A **Panel** is generally used as a component in a **Frame** or an **Applet**.

It is used **to group components** into a single unit. The Panel with its set of components is displayed as a single unit. It represents a rectangular region of the display.

A **Panel** holds its own LayoutManager which may be different from the LayoutManager of the Frame or Applet in which it resides.

Dialog & ScrollPane

Dialog

Always attached to an instance of Frame.

A window that is displayed for a short duration during execution.

Default LayoutManager is BorderLayoutManager (same as for Frame)

ScrollPane

It can hold only one Component.

It does not have a LayoutManager.

If size of the component held is larger than the size of the ScrollPane, scroll bars will be automatically generated.

Graphics Programming Using Pane

Four panes are layered in a *JFrame*.
They are

- **root pane**
- **layered pane**
- **glass pane and**
- **content pane**

The first three panes are of no interest to programmers. Because, they are required to organize the menu bar and content pane and to implement the look and feel. The part that most concerns Swing programmers is the ***content pane***. When designing a frame, you add components into the content pane, using code as follows:

```
Container contenetPane=getContenetPane();
Component c=…….;
contenetPane.add(c);
```

Swing Components

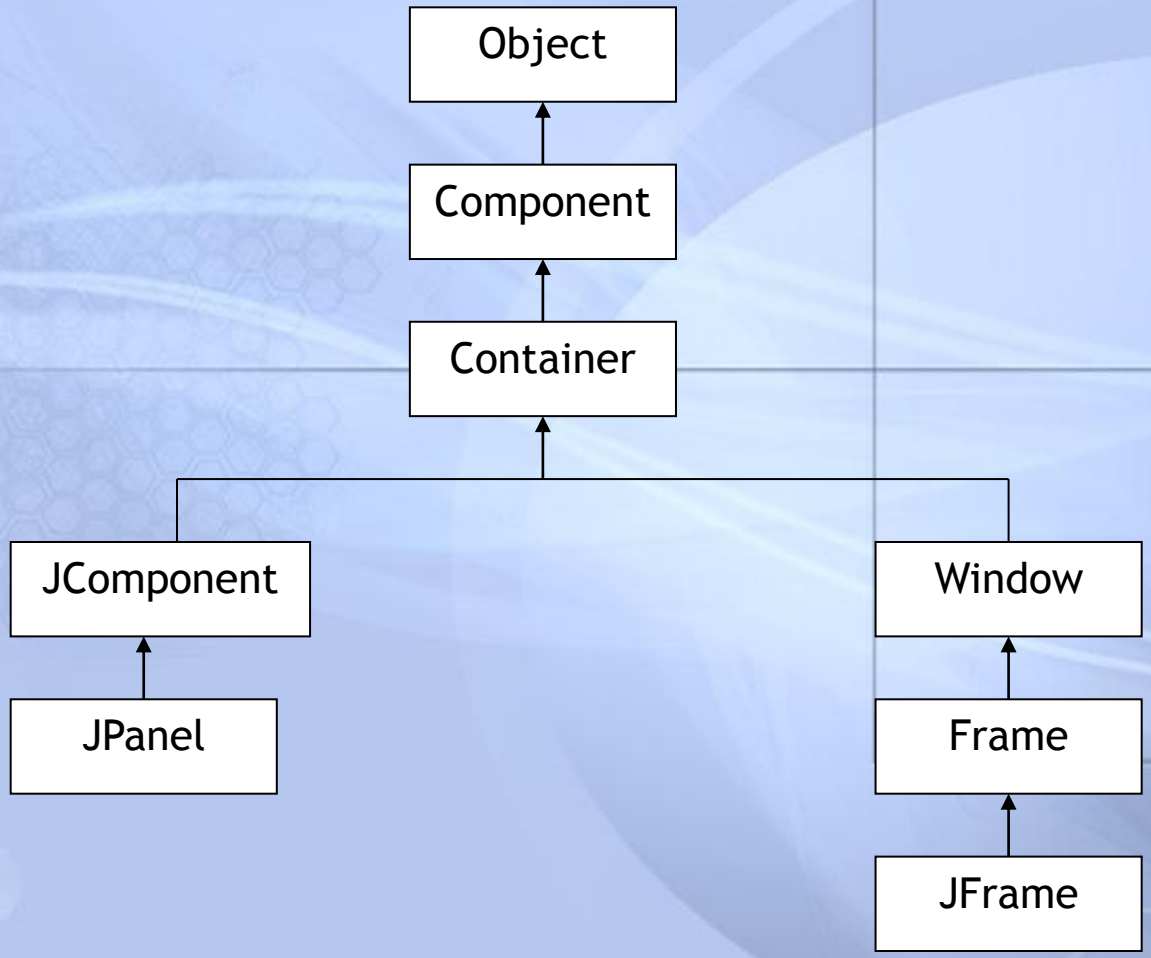
- Classes from package **`javax.swing`** defines various GUI components objects with which the user interacts via the mouse, the keyboard or another form of input.
- **Some basic GUI components**
  - **`JFrame`**
  - **`JPanel`**
  - **`Jlabel`**
  - **`JTextField`**
  - **`JButton`**
  - **`JCheckBox`**
  - **`JComboBox`**
  - **`Jlist`**
  - **`etc:,`**

JFrame

- A top-level window (that is, a window that is not contained inside another window) is called a frame in Java.
- The Swing library has a class **JFrame** for this top level.
- Frames are examples of containers.
- This means that a frame can contain other user interface components such as buttons and text fields.

Jframe (Cont.)

The following figure illustrates the inheritance chain for the JFrame class.



JFrame Example

```
import javax.swing.*;
class SimpleFrame extends JFrame
{   public SimpleFrame()
    {
      setSize(300,200);//WIDTH, HEIGHT
      setTitle("Simple Frame Test!");//Set Frame Title
      setLocation(100,100);//Set Position on the screen
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setVisible(true); //show window
     }
    public static void main(String[] args)
    {
      SimpleFrame frame = new SimpleFrame();
    }
}
```

Output

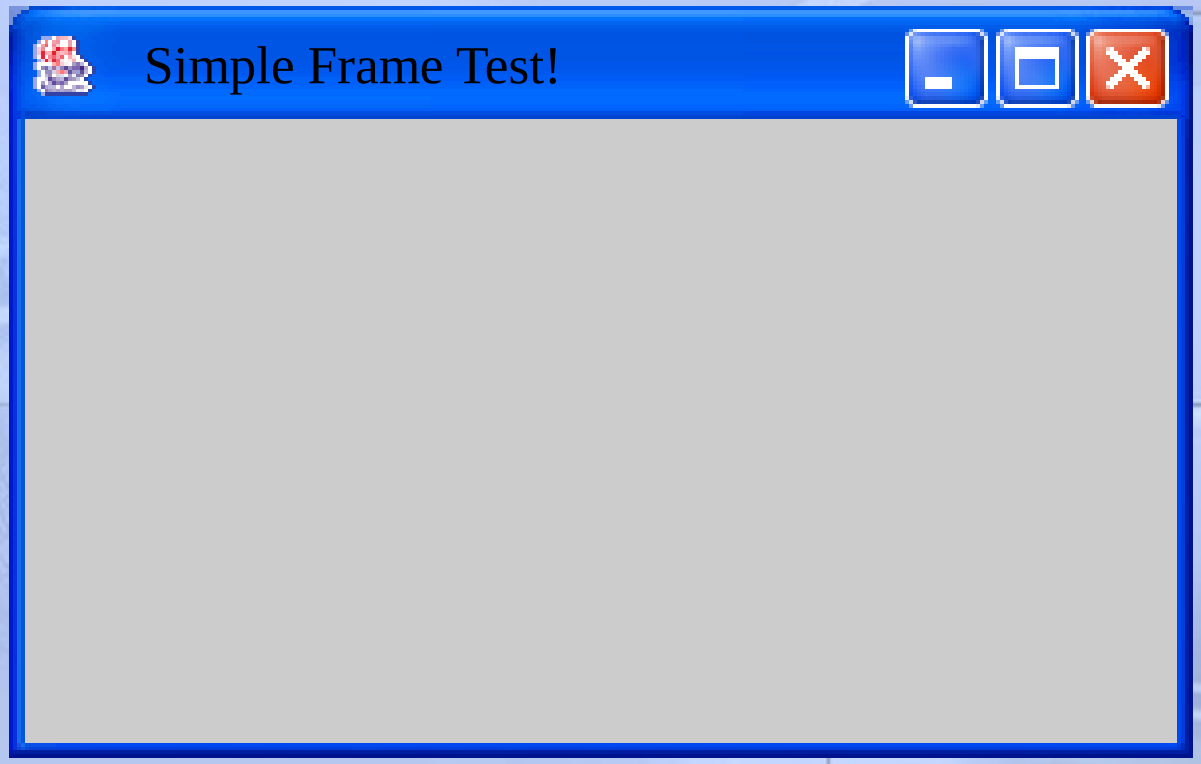


Figure : Output Frame of the previous program

JLabel

- A `JLabel` object provides text instructions or information on a GUI display a single line of *read-only* text, an image or both text and image
- One thing to be empasized: if you do not explicitly add a GUI component to a container, the GUI component will not be displayed when the container appears on the screen
  1. Construct a JLabel component with the correct text.
  2. Place it close enough to the component you want to identify so that the user can see that the label identifies the correct component.

JLabel

JLabel (String text)

JLabel (Icon icon)

JLabel (String text, int align)

JLabel (String text, Icon icon, int align)

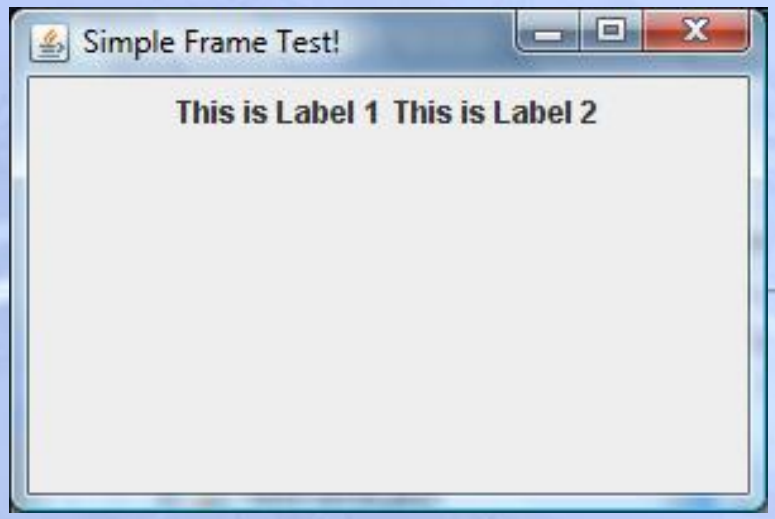
void setText (String text)

void setIcon (Icon icon)

JLabel

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class labelTest extends JFrame
{ private JLabel label1, label2;
   public labelTest()
   { super (“Simple Frame Test!”);
     Container c= getContentPane();
     c.setLayout(new FlowLayout());
     label1=new JLabel(“This is Label 1”);
     label1.setToolTipText(“This is ToolTip”);
     c.add(label1);
     label2=new JLabel();
     label1.setText(“This is Label 2”);
     c.add(label2);
   }
public static void main (String args[])
   { labelTest testObj= new labelTest();
     testObj.setVisible(true);
   }
}
```

Output



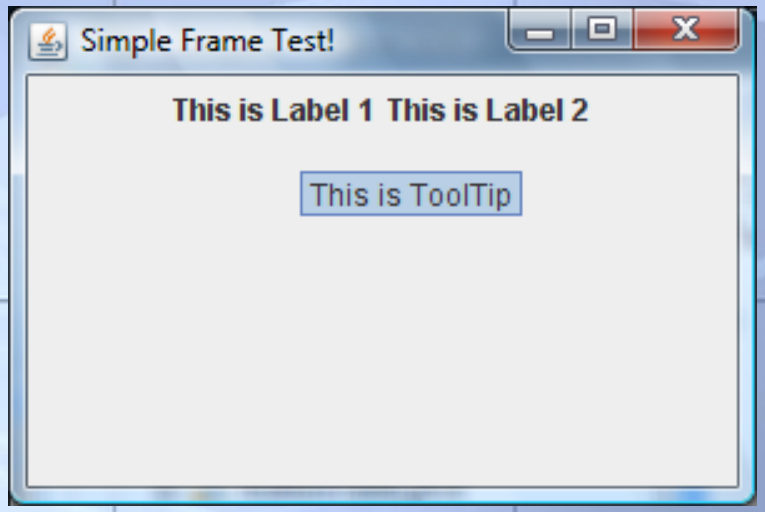


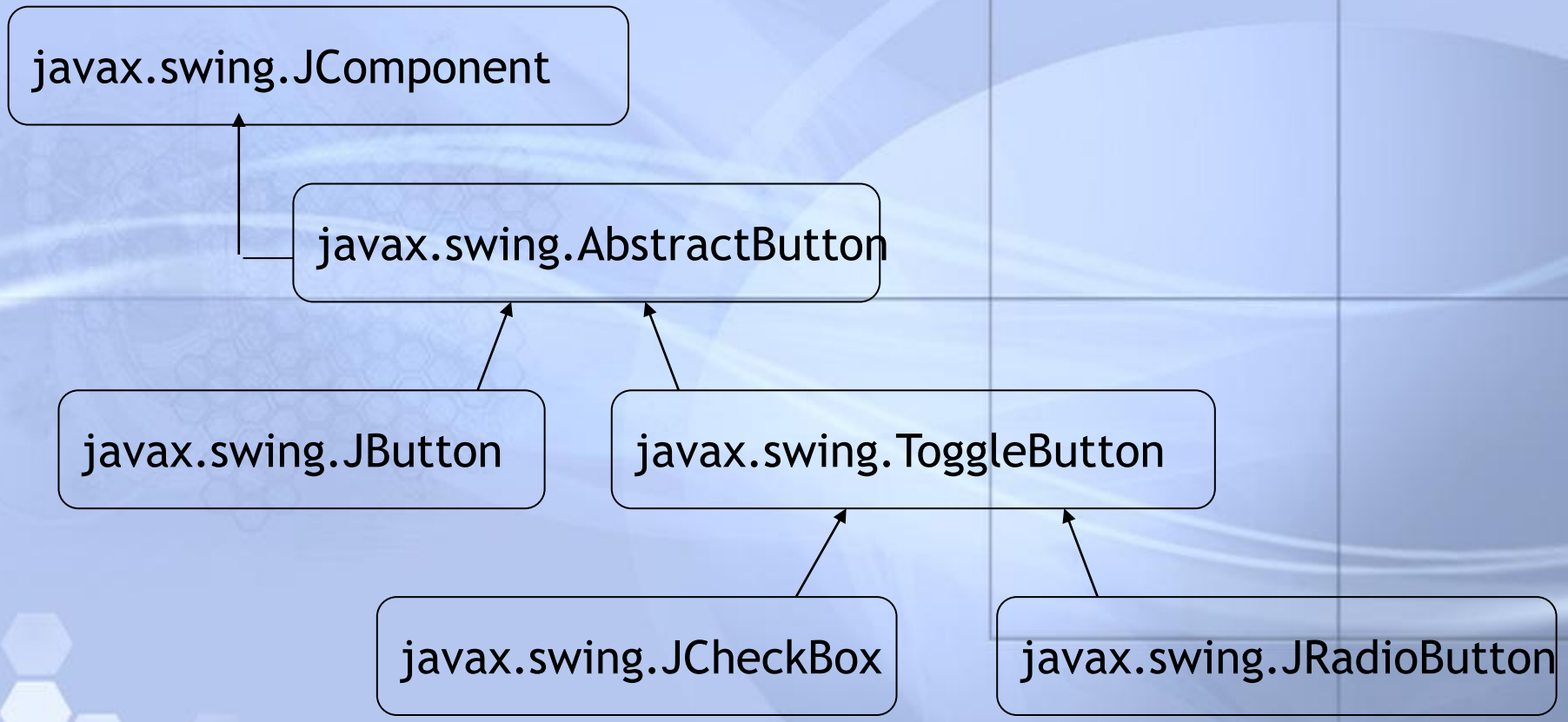
Figure : Output Frame of the previous program

JButton

- **Button**
  - Component user clicks to trigger an action
  - Several types of buttons
    - Command buttons, toggle buttons, check boxes, radio buttons
- **Command button**
  - Generates **ActionEvent** when clicked
  - Created with class **JButton**
    - Inherits from class **AbstractButton**
    - Defines many features of Swing buttons
- **JButton**
  - Text on face called button label
  - Each button should have a different label
  - Support display of **Icons**

JButton

Several types of buttons are subclasses of **`AbstractButton`**



JButton

- Methods of class **JButton**
  - Constructors

  JButton myButton = new JButton("Label");

  JButton myButton = new JButton("Label", myIcon);
  - **setRolloverIcon(myIcon)**
    - Sets image to display when mouse over button

- Class ActionEvent
  - **getActionCommand**
    - Returns label of button that generated event

Example

```
import java.awt.*;
import javax.swing.*;
Import java.awt.event.*;
public class buttonTest extends JFrame
{ private JButton button1, button2;
   public buttonTest()
   { super (“Testing JButton”);
      Container c= getContentPane();
      c.setLayout(new FlowLayout());
      button1=new JButton(“Click Me”);
      c.add(button1);
      show();
    }
   public static void main (String args[])
   {  buttonTest testobj= new buttonTest();
   }
}
```

Output

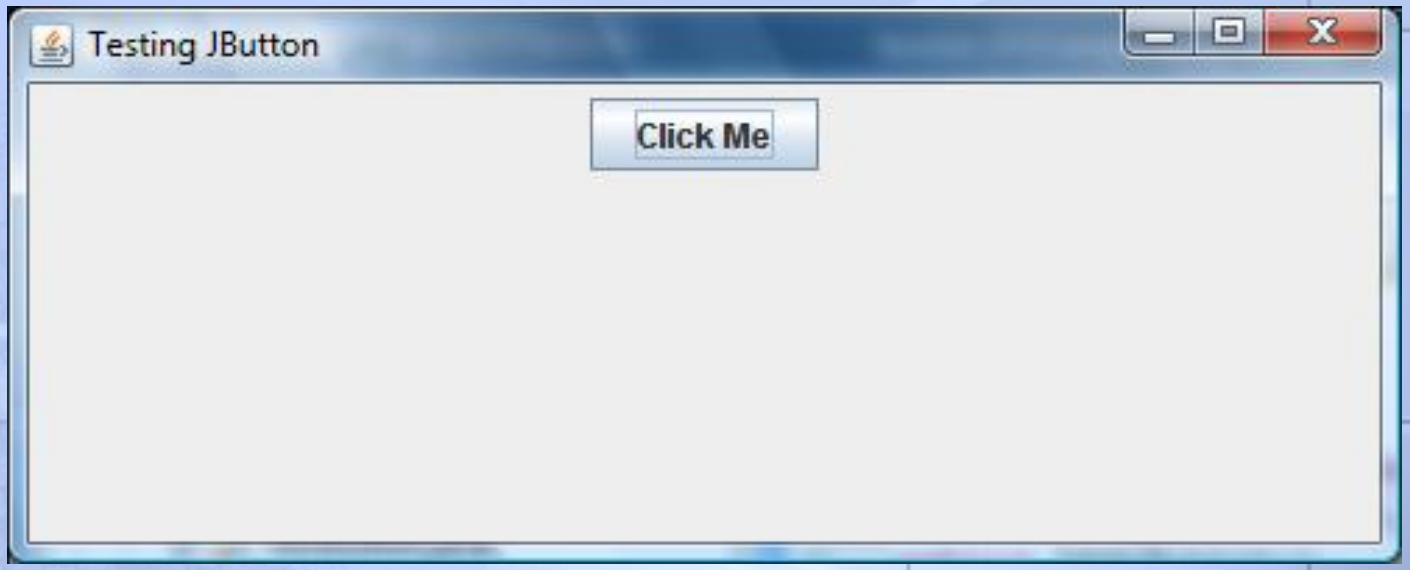


Figure : Output Frame of the previous program

To Make an Interactive GUI Program

To make an interactive GUI program, you need:

Components

buttons, windows, menus, etc.

Events

mouse clicked, window closed, button clicked, etc.

Event listeners (interfaces) and event handlers (methods)

listen for events to be trigged, and then perform actions to handle them

Event Handling Model

- **GUIs are event driven**
  - Generate events when user interacts with GUI
    - Mouse movements, mouse clicks, typing in a text field, etc.
  - Event information stored in object that extends **AWTEvent**
- To process an event
  - Register an event listener
    - Object from a class that implements an event-listener interface (from **java.awt.event** or **javax.swing.event**)
    - "Listens" for events
  - Implement event handler
    - Method that is called in response to an event
    - Event handling interface has one or more methods that must be defined

Event Handling Model

Event handling is of fundamental importance to programs with a graphical user interface.

Here's an overview of how event handling in the AWT works:

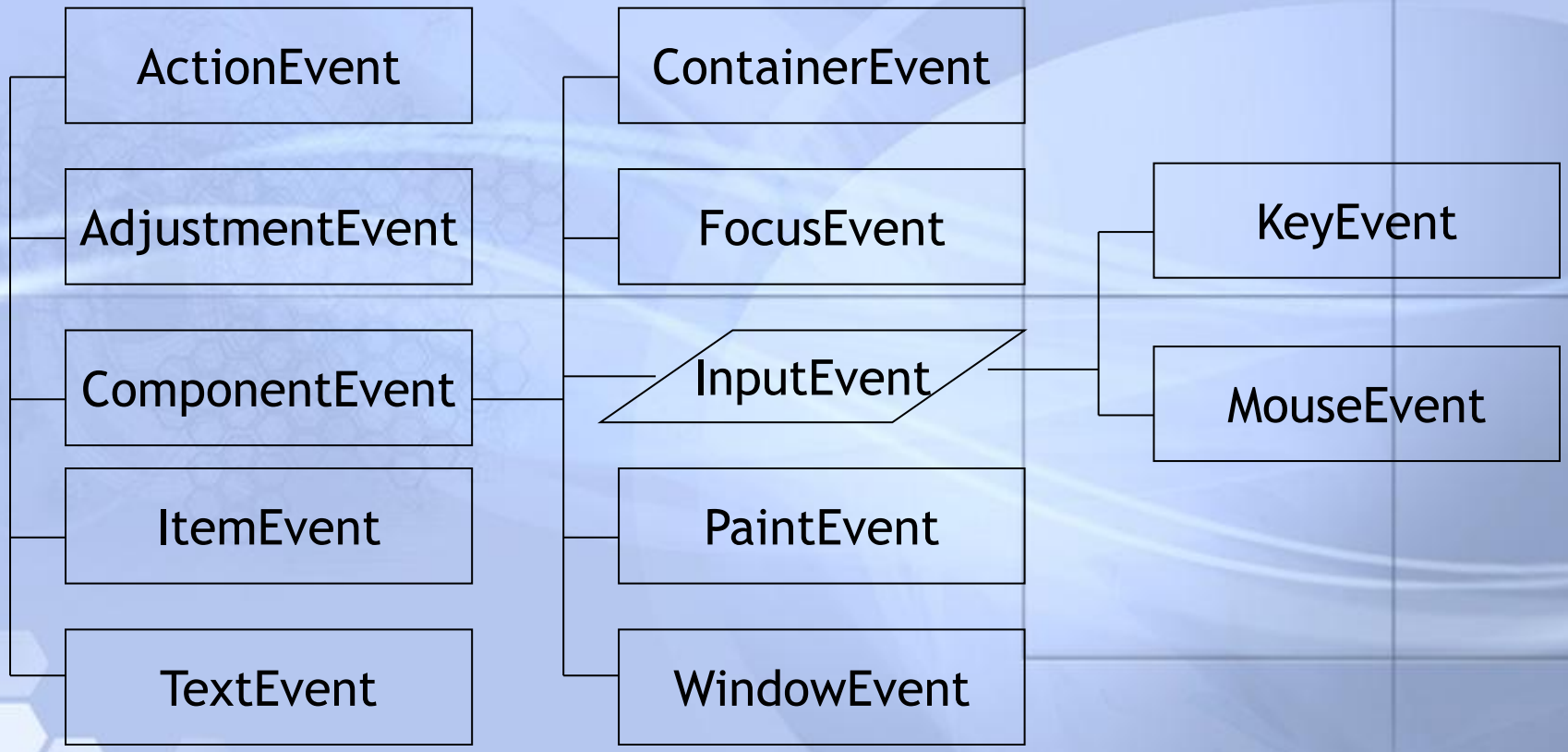
- A listener object is an instance of a class that implements a special interface called a *listener* interface.
- An event source is an object that can register listener objects and send them event objects.
- The event source sends out event objects to all registered listeners when that event occurs.
- The listener object will then use the information in the event object to determine their reaction to the event.

The code to register an event is

eventSourceObject.addEventListener(eventListenerObject);

Event-Handling Model (Cont'd)

some event classes in package java.awt.event



Event Handling Model

Some of the AWT event classes are of no practical use for the Java programmer. E.g., *PaintEvent* and *InputEvent*. There are eleven *listener* interfaces altogether in the ***java.awt.event*** package:

Interface	**Methods**	**Parameter**	**Events Generated By**
ActionListener	actionPerfomed	ActionEvent	AbstractButton JComboBox JTextField Timer
AdjustmentListener	adjustmentValueChanged	AdjustmentEvent	JScrollBar
ItemListener	itemStateChanged	ItemEvent	AbstractButton JComboBox
TextListener	textvalueChanged	TextEvent	Component

Event Handling Model

Interface	Methods	Parameter	Events Generated By
ComponentListener	componentMoved componentHidden componentResized componentShown	ComponentEvent	Container
ContainerListener	componentAdded componentremoved	ContainerEvent	Component
FocusListener	focusGained focusLost	FocusEvent	Component
KeyListener	keyPressed, keyReleased keyTyped	KeyEvent	Component

Event Handling Model

Interface	Methods	Parameter	Events Generated By
MouseListener	mousePressed mouseReleased mouseEntered mouseExited mouseClicked	MouseEvent	Component
MouseMotionListener	mouseDragged mouseMoved	MouseEvent	Component
WindowListener	windowOpened windowClosed windowActivated windowDeactivated	WindowEvent	Window

Example

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class buttonTest1 extends JFrame implements Action Listener
{  JButton button1, button2;
   JLabel label1=new JLabel();
   public buttonTest1()
   { super ("Testing JButton");
     Container c= getContentPane();
     c.setLayout(new FlowLayout());
     button1=new JButton("Click Me");
     c.add(button1);
     button2=new JButton("Nothing Done");
     c.add(button2); c.add(label1);
     button1.addActionListener(this);
     show();
    }
```

Example (Cont)

```
public void actionPerformed(ActionEvent e)
 { String msg= new String("Hello from Java");
   if(e.getSource()==button1)label1.setText(msg);
 }
 public static void main (String args[])
   {  buttonTest1 testobj= new buttonTest1();
   }
 }
```

Output

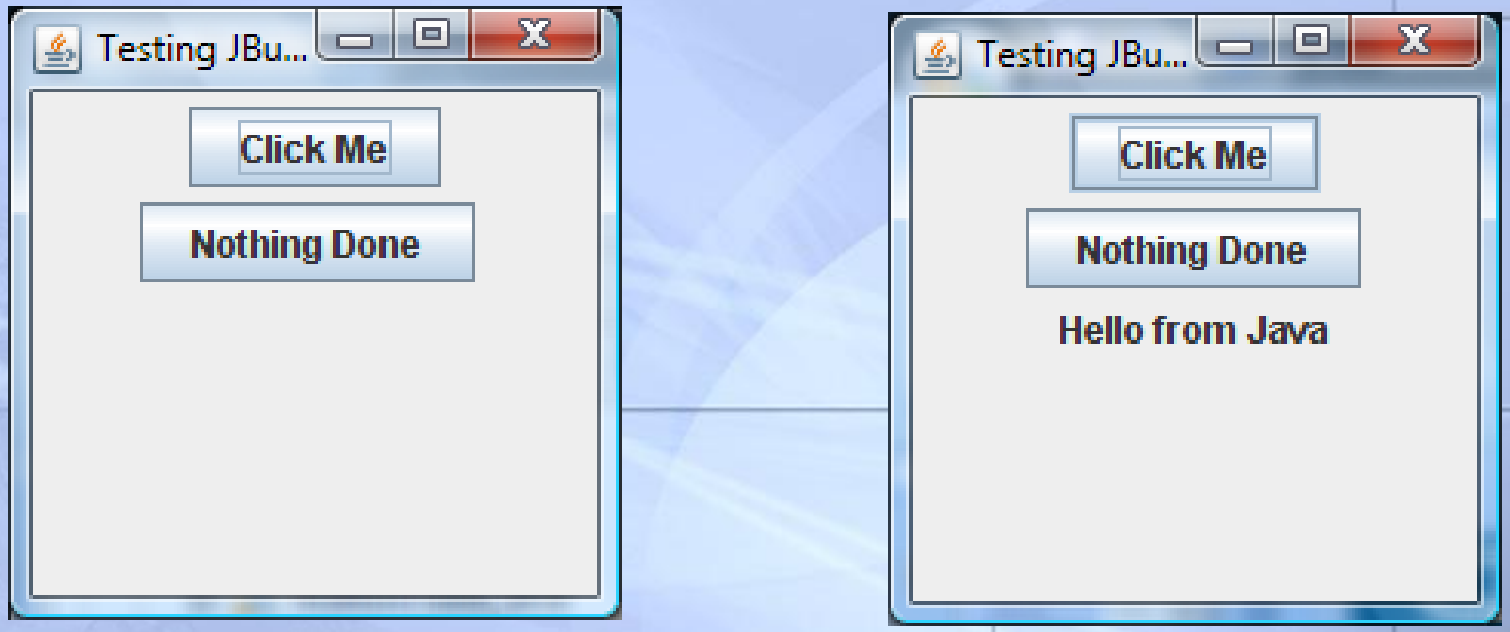


Figure : Output Frame of the previous program before and after Clicking

Example2

```
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.*;
import javax.swing.*;
public class btnTest extends JFrame implements ActionListener
{ JLabel label;
  JButton button;
  JPanel panel;
  private boolean clickMeMode = true;
 public btnTest()
 { this.setTitle("Example");
    setSize(150,90);
    setLocation(100,100);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    label = new JLabel("I'm a Simple Program");
    button = new JButton("Click Me");
```

Example 2

```
//Add button as an event listener
    button.addActionListener(this);
//Create panel
    panel = new JPanel();
    panel.add(label);
    panel.add(button);
//Add label and button to panel
    setContentPane(panel);
    setVisible(true);
  }
public void actionPerformed(ActionEvent event)
{  Object source = event.getSource();
    if (clickMeMode)
    { label.setText("Button Clicked");
      button.setText("Click Again");
      clickMeMode = false;
    }
```

Example 2

```
        else
        {
            label.setText("I'm a Simple Program");
            button.setText("Click Me");
            clickMeMode = true;
          }
    }
  public static void main(String[] args)
  {
    btnTest btnTest1 = new btnTest();
  }
}
```

Output

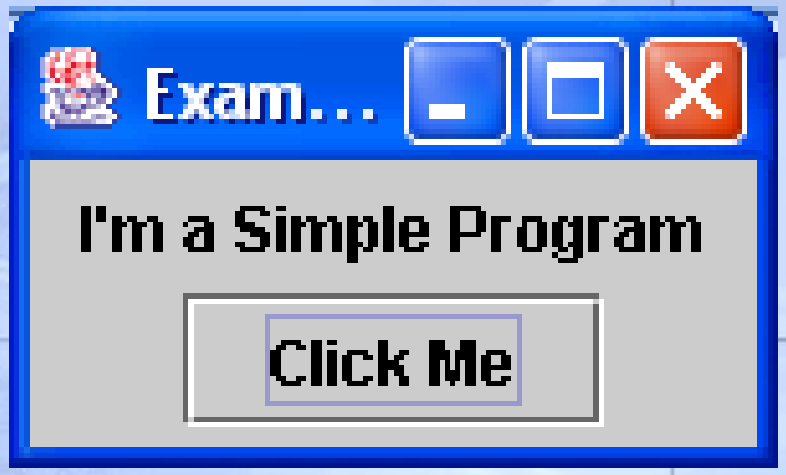


Figure : Output Frame of the previous program

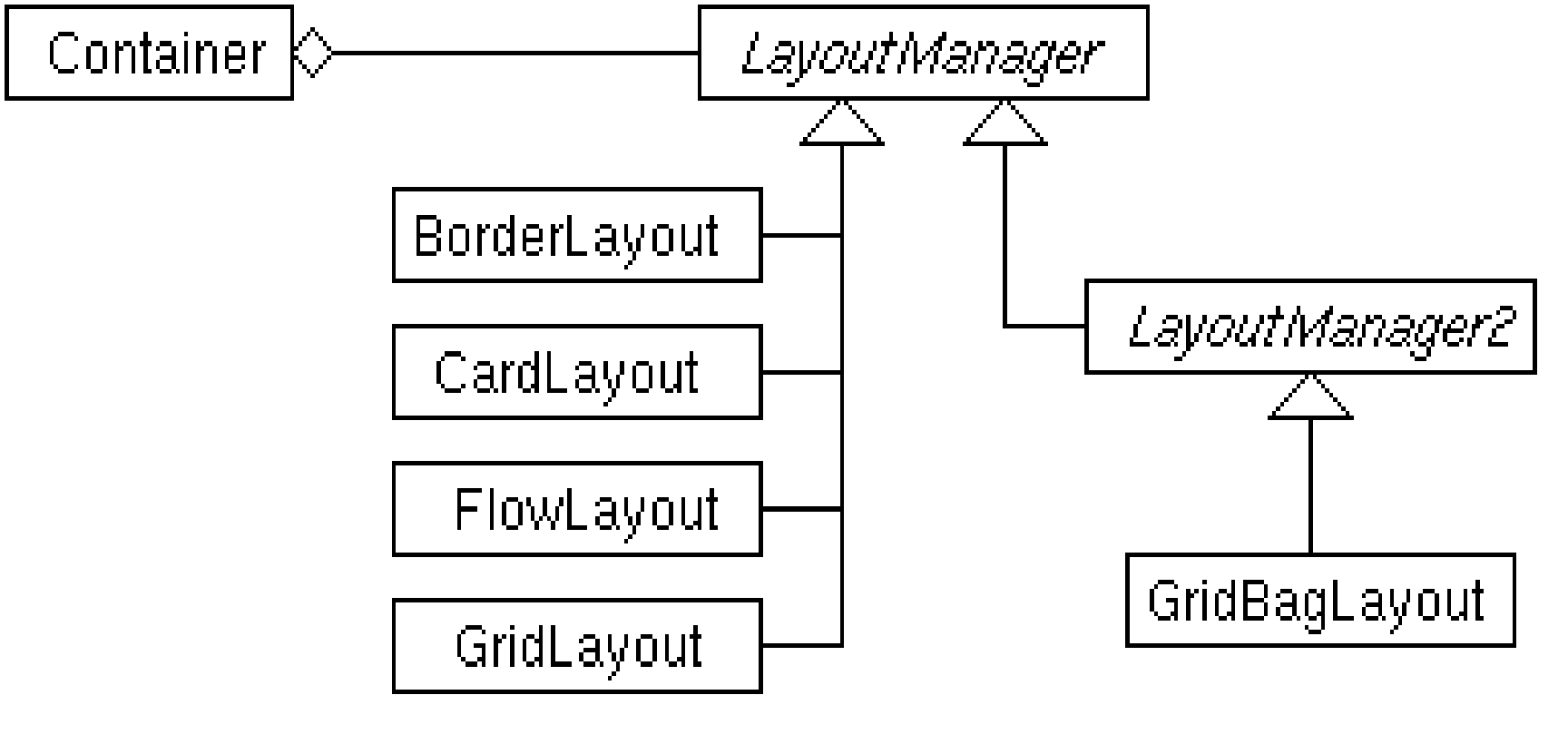
Layout Managers

A layout manager allows the java programmer to develop graphical interfaces that will have a common appearance across the heterogeneous internet.

These are layout managers that the programmer may choose from:

- **Flow Layout**
- **Grid Layout**
- **Border Layout**
- **Card Layout**
- **Gridbag Layout**
- **Box Layout**
- **Circle Layout**

Hierarchy of Layout Managers



Flow Layout

- Flow layout is layout components row by row from left to right.
- It is the ***default*** layout manager for a *panel*.
- The flow layout manager lines the components horizontally until there is no more room and then starts a new row of components.
- When the user resizes the container, the layout manager automatically reflows the components to fill the available space.
- The following example shows the demonstration of flow layout of components.

Usage:

setLayout(new FlowLayout());

Example

```
import javax.swing.*;
import java.awt.*;
public class layoutTest extends JFrame
{JButton b1=new JButton("Yellow");
  JButton b2=new JButton("Blue");
  JButton b3=new JButton("Red");
  JButton b4=new JButton("Green");
  JButton b5=new JButton("Orange");
  JButton b6=new JButton("Cyan");
  public layoutTest()
  {this.setSize(290,200);
    JPanel p=new JPanel();
    //  p.setLayout(new FlowLayout(FlowLayout.LEFT));        (OR)
    //  p.setLayout(new FlowLayout(FlowLayout.RIGHT ));      (OR)
    //  p.setLayout(new FlowLayout(FlowLayout.CENTER ));   // Default is CENTER
```

Example

```
    p.add(b1);
    p.add(b2);
    p.add(b3);
    p.add(b4);
    p.add(b5);
    p.add(b6);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setContentPane(p);
    this.show();
  }
  public static void main(String[] args)
  {
            new layoutTest();
  }
}
```

Output

BorderLayout

Properties of a Border Layout Manager

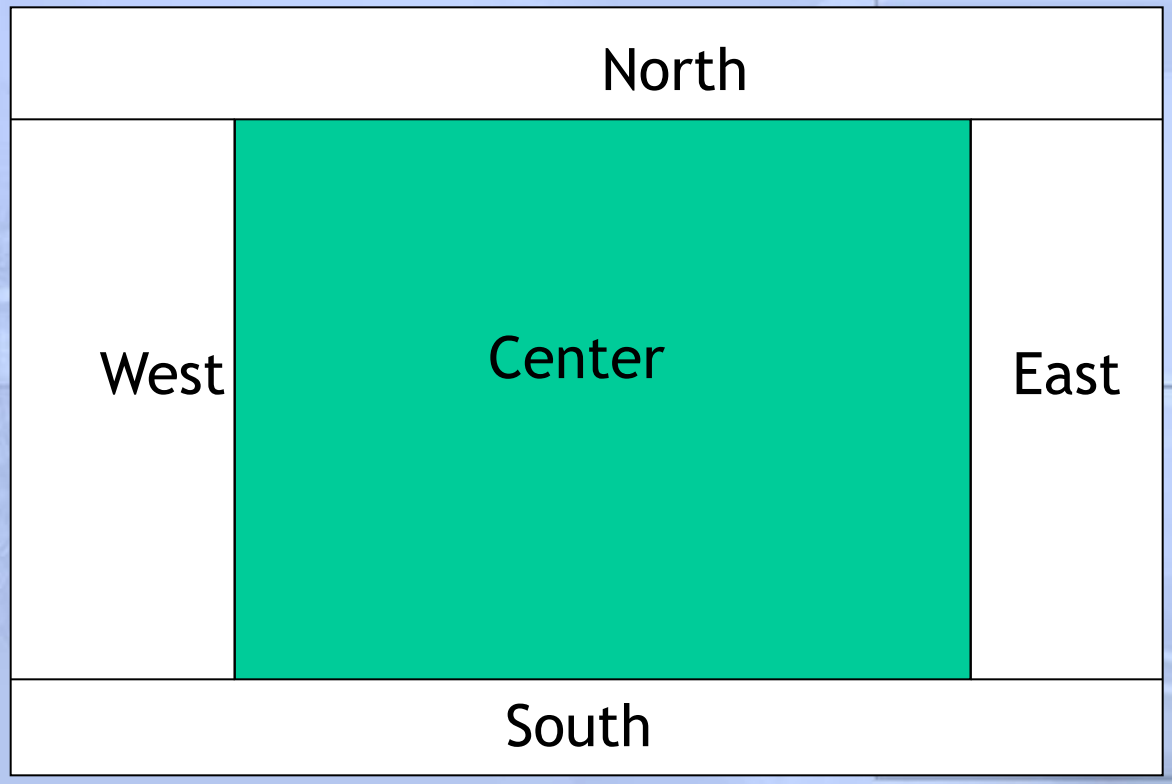
More flexible in permitting programmer placement of components of different sizes (such as panels, canvases and text areas).

Border areas adjust to accommodate the placement of a component. The center consists of what's left over.

Usage:

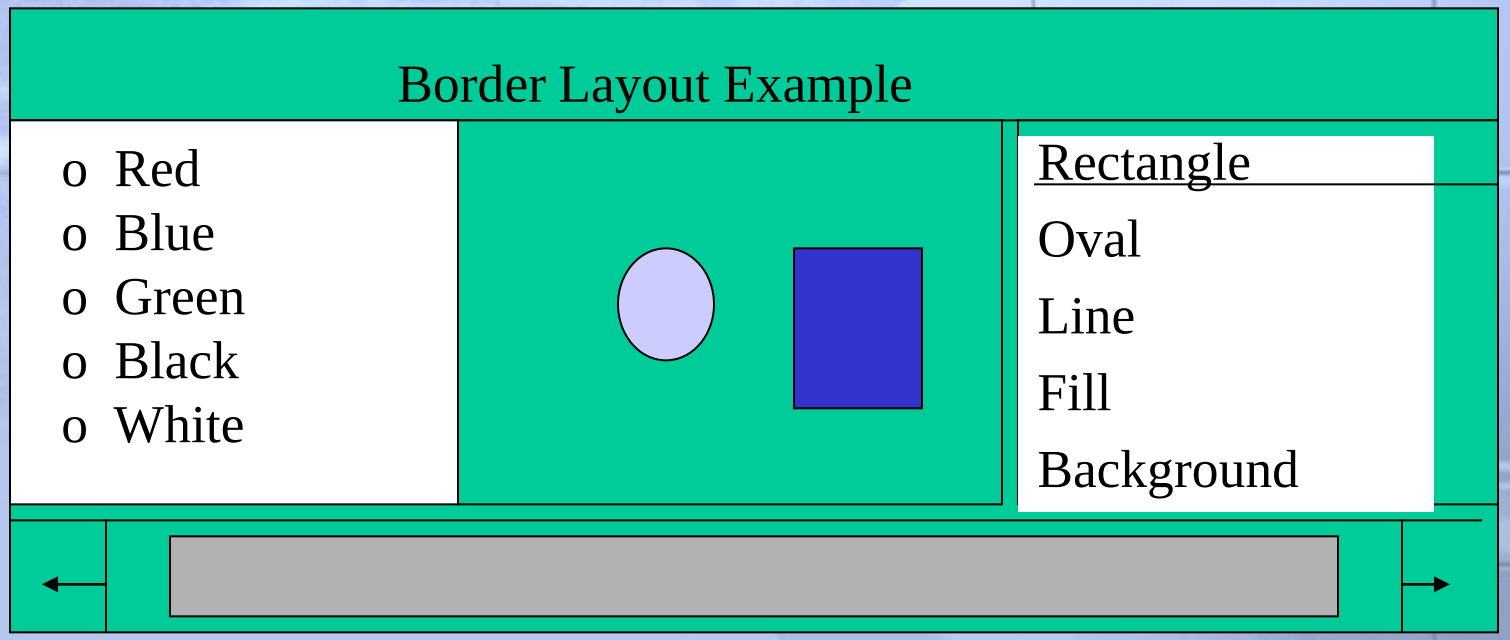
```
setLayout(new BorderLayout());
add(new JButton("North"), BorderLayout.NORTH);
```

BorderLayout



Border Layout

Example – A BorderLayout with a Label in North, a Scrollbar in South, a Panel containing a Checkboxgroup in West, a List in East, and a Canvas in Center.



GridLayout

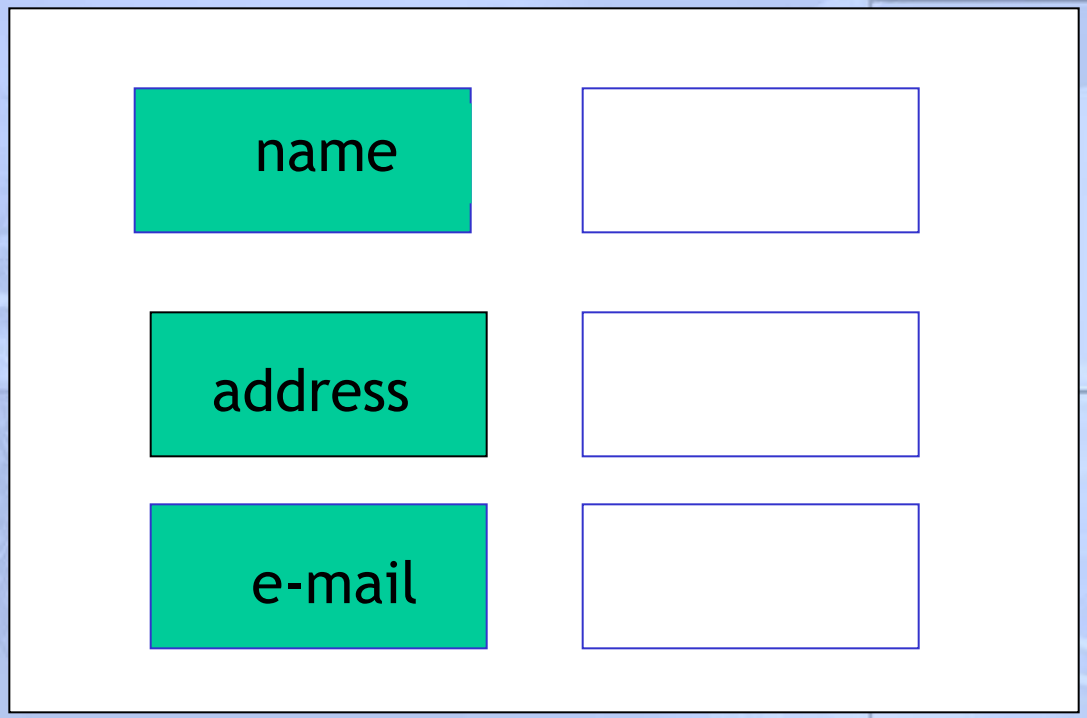
Components expand or contract in size to fill a single grid cell. Panels, canvases, and Lists are resized to fit into a grid cell.

Components are placed in the container in order from left to right and top to bottom.

Usage:

setLayout(new GridLayout(row, col));

Grid Layout



CardLayout

A card layout is a group of components or containers that are displayed one at a time.

Usage:

void add(Component com, String name)

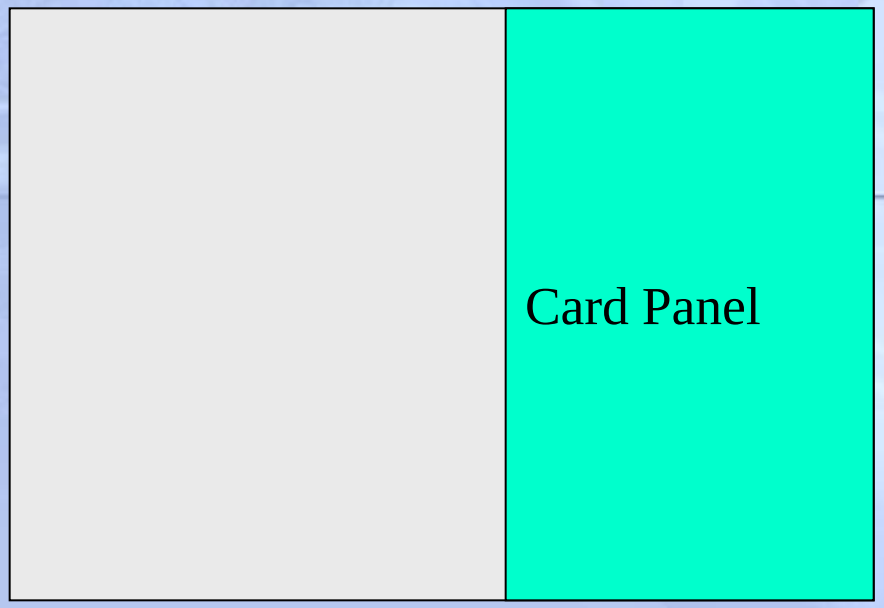
CardLayout View Components are

- void first(container)
- void last(container)
- void next(container)
- void previous(container)
- void show(Container, String name)

Card Layout

Card Layout Construction

The Applet contains an ordered listing (array) of cards in one of its panels



GridBag Layout

The grid bag layout manager is a flexible layout manager that aligns components horizontally and vertically, without requiring that the components be the same size.

Each grid bag layout manager uses a rectangular grid of cells, with each component occupying one or more cells (called its *display area*).

Each component in a grid bag layout is associated with a set of constraints contained within a GridBag-Constraints instance that specifies how the component is to be laid out within its display area.

Example

```
import  java.awt.*;
import  java.util.*;
import  java.applet.Applet;
public  class  GridBagEx1  extends  Applet  {
     protected  void  makebutton(String  name,
                    GridBagLayout  gridbag,
                     GridBagConstraints  c)  {
          Button  button  =  new  Button(name);
          gridbag.setConstraints(button,  c);
          add(button);
     }
     public  void  init()  {
          GridBagLayout  gridbag  =  new  GridBagLayout();
          GridBagConstraints  c  =  new  GridBagConstraints();
          setFont(new  Font("Helvetica",  Font.PLAIN,  14));
          setLayout(gridbag);
          c.fill  =  GridBagConstraints.BOTH;
          c.weightx  =  1.0;
          makebutton("Button1",  gridbag,  c);
          makebutton("Button2",  gridbag,  c);
          makebutton("Button3",  gridbag,  c);
```

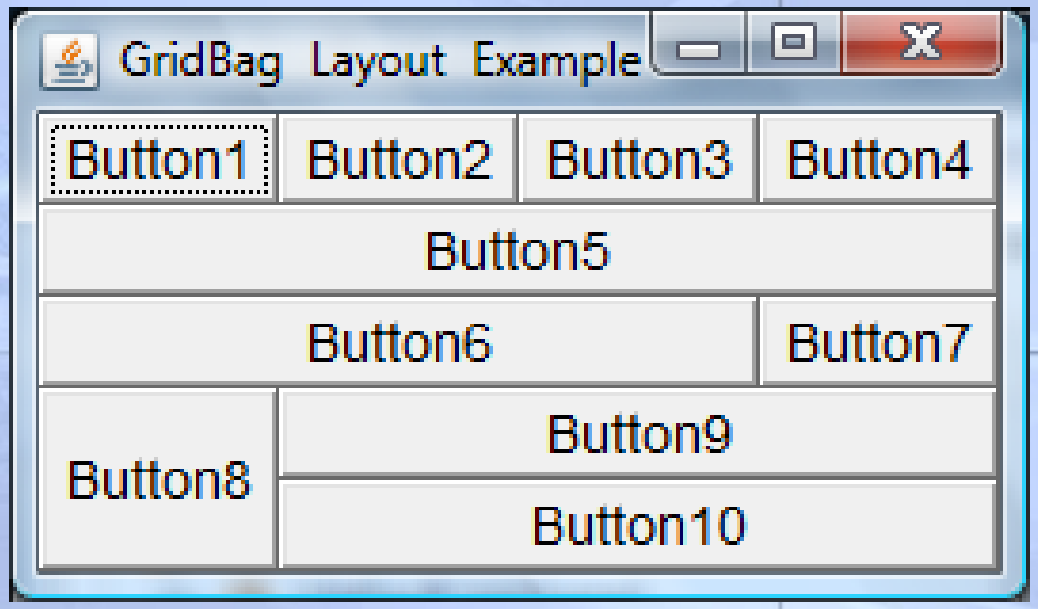
Example

```
    c.gridwidth = GridBagConstraints.REMAINDER;
    makebutton("Button4", gridbag, c);
    //end first row
       c.weightx = 0.0;                    //reset to the default
       makebutton("Button5", gridbag, c); //another row
    // next-to last in row
    c.gridwidth = GridBagConstraints.RELATIVE;
       makebutton("Button6", gridbag, c);
    c.gridwidth = GridBagConstraints.REMAINDER; //end row
       makebutton("Button7", gridbag, c);
    c.gridwidth = 1;                       // reset to the default
    c.gridheight = 2;
       c.weighty = 1.0;
       makebutton("Button8", gridbag, c);
       c.weighty = 0.0;                    //reset to the default
    // end row
    c.gridwidth = GridBagConstraints.REMAINDER;
    c.gridheight = 1;                      // reset to the default
       makebutton("Button9", gridbag, c);
       makebutton("Button10", gridbag, c);
       resize(300, 100);
}
```

Example

```
public static void main(String args[]) {
        Frame f = new Frame("GridBag Layout Example");
          GridBagEx1 ex1 = new GridBagEx1();
        ex1.init();
        f.add("Center", ex1);
        f.pack();
      f.show();
      }
}
```

Output


GridBag Layout Example
Button1
Button2
Button3
Button4
Button5
Button6
Button7
Button8
Button9
Button10

JTextField and JPasswordField

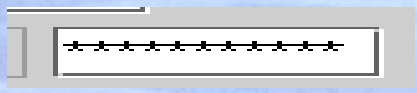
They are single-line areas in which text can be entered by the user from the keyboard or text can simply be displayed

When the user types data into them and presses the *Enter* key, an action event occurs.

If the program registers an event listener, the listener processes the event and can use the data in the text field at the time of the event in the program.

JTextField and JPasswordField

- **JTextField**s and **JPasswordField**s
  - Single line areas in which text can be entered or displayed
  - **JPasswordField**s show inputted text as an asterisk *



  - **JTextField** extends **JTextComponent**
    - **JPasswordField** extends **JTextField**

- When **Enter** pressed
  - **ActionEvent** occurs
  - Currently active field "has the focus"

JTextField and JPasswordField

- Methods
  - Constructors
    - **JTextField(10)**
      - Textfield with 10 columns of text
      - Takes average character width, multiplies by 10
    - **JTextField("Hi")**
      - Sets text, width determined automatically
    - **JTextField("Hi", 20)**
  - **setEditable(boolean)**
    - If **false**, user cannot edit text
    - Can still generate events
  - **getPassword**
    - Class **JPasswordField**
    - Returns password as an **array** of type **char**

JTextField and JPasswordField

- Class ActionEvent
  - Method **getActionCommand**
    - Returns text in **JTextField** that generated event
  - Method **getSource**
    - **getSource** returns a **Component** reference to component that generated event
- Example
  - Create **JTextField**s and a **JPasswordField**
  - Create and register an event handler
    - Use **getSource** to determine which component had event
    - Display a dialog box when *Enter* pressed

Example

```
import java.awt.*;
import javax.swing.*;
import.java.awt.event.*;
public class TextfieldTest extends JFrame
{
    final JTextField  tfield= new JTextField(“Press Here”, 20);
    public TextfieldTest()
    {      super ("Testing JTextField");
           setSize(500,300);
            Container c= getContentPane();
           c.setLayout(new FlowLayout());
            c.add(tfield);
           tfield.setHorizontalAlignment(JTextField.RIGHT);
```

Example

```
    tfield.addActionListener(new ActionListener()
            {public void actionPerformed(ActionEvent e)
              { int old= tfield.getHorizontalAlignment();
                if(old== JTextField.LEFT)
                        tfield.setHorizontalAlignment(JTextField.RIGHT);
                if(old== JTextField.RIGHT)
                        tfield.setHorizontalAlignment(JTextField.CENTER);
                if(old== JTextField.CENTER)
                        tfield.setHorizontalAlignment(JTextField.LEFT);
                }
            });
     tfield.requestFocus();
     show();
}
  Public static void main(String args[])
  { TextfieldTest testobj= new TextfieldTest ();
   }
}
```

Figure : Output Frame of the previous program

JPasswordField

- The JPasswordField class is a subclass of JTextField, provides text fields specialized for password entry.
- Good for security reason.
- Store its value as an array of characters, rather than a string.
- Following is the code which will create and set up the password field:

```
JPasswordField pwdField= new JPasswordField(10);
pwdField.setEchoChar('#');
pwdField.setActionCommand("OK");
pwdField.addActionListener(this);
```

Example

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class pwdTest extends JFrame implements ActionListener
{  JPasswordField pwdField= new JPasswordField(10);
   public pwdTest()
   { super ("Testing JTextField");
     setSize(500,300);
     Container c= getContentPane();
     c.setLayout(new FlowLayout());
     c.add(pwdField);
     pwdField.setEchoChar('#');
     pwdField.setActionCommand("OK");
     pwdField.addActionListener(this);
     show();
    }
```

Example

```
public void actionPerformed(ActionEvent e)
  { String strcmd= e.getActionCommand();
    if("OK".equals(strcmd))
    { char[] input= pwdField.getPassword();
      if(isPasswordCorrect(input))
      {JOptionPane.showMessageDialog(this,"Your Password is correct");
      }
      else
      {JOptionPane.showMessageDialog(this,"Invalid Password!Try Again");
      }
    for (int i=0; i<input.length;i++)
    { input[i]=0;
    }
    pwdField.selectAll();
    pwdField.setText(null);
    }
  }
```

Example

```
public static boolean isPasswordCorrect(char[] input)
   { boolean isCorrect = true;
     char[] c= {'i','m','c','e','i','t','s'};
     if (input.length != c.length)isCorrect=false;
     else
     { for(int i=0;i<input.length;i++)
       {if(input[i]!=c[i])isCorrect=false;}
     }
     for (int i=0; i<c.length;i++)
     { c[i]=0;
     }
     return isCorrect;
   }
   public static void main(String args[])
   { pwdTest testobj= new pwdTest();
   }
}
```

Example

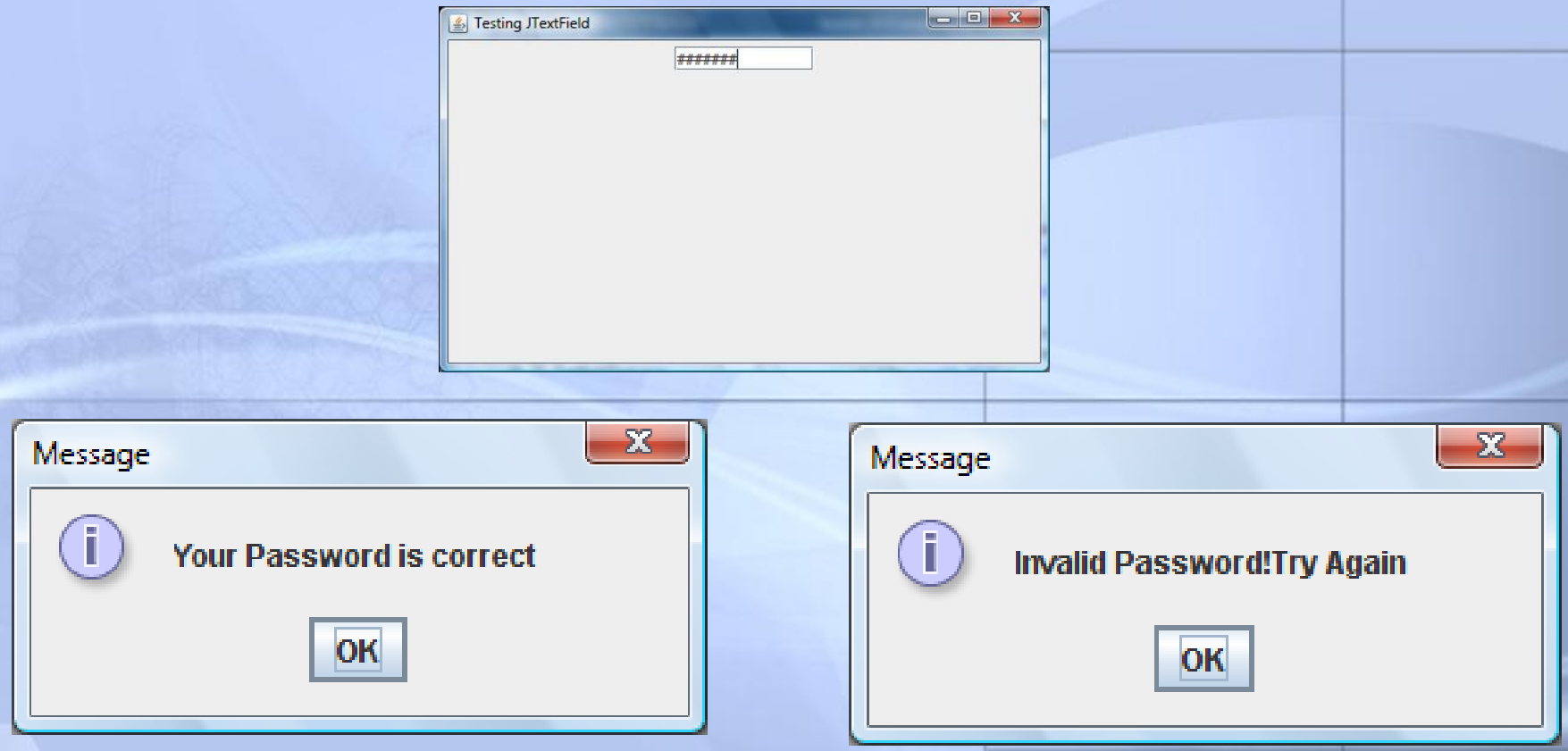


Figure : Output Frame of the previous program

Thank You!