# Unit – 2 & 3 User Interface Components with Swing and Event Handling

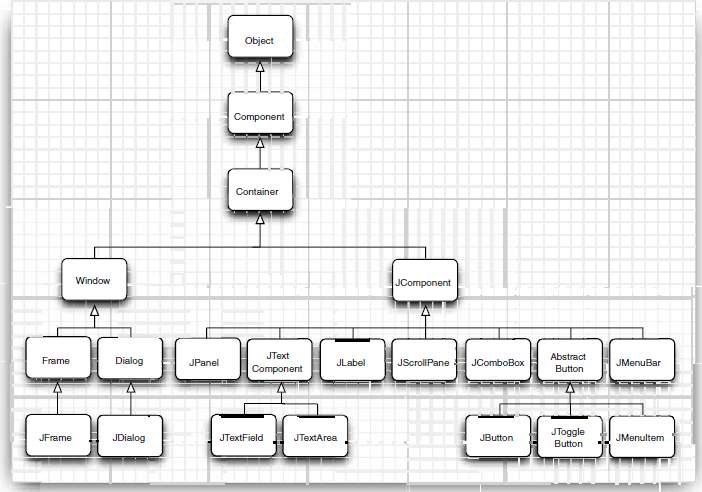
# About AWT & Swing

AWT and Swing both are used to create GUI in Java. Both of them are used to perform almost same work, still they differ from each other.

Following are the differences between AWT and Swing:

|  |  |
| --- | --- |
| **AWT** | **Swing** |
| AWT stands for Abstract Window Toolkit. | Swing is a part of Java Foundation Class (JFC). |
| AWT components are heavy weight. | Swing components are light weight. |
| AWT components are platform dependent so there look and feel changes according to OS. | Swing components are platform independent so there look and feel remains constant. |
| AWT components are not very good in look and feel as compared to Swing components. See the button in below image, its look is not good as button created using Swing.  Difference between AWT and Swing | Swing components are better in look and feel as compared to AWT. See the button in below image, its look is better than button created using AWT.  Difference between AWT and Swing |
| AWT were built using C language. | Swing built using java language. |

# Overview of Swing Components



# Java Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

## Advantage of Applet

There are many advantages of applet. They are as follows:

* It works at client side so less response time.
* Secured
* It can be executed by browsers running under many platforms, including Linux, Windows, Mac Os etc.

## Drawback of Applet

* Plugin is required at client browser to execute applet.

# Lifecycle of Java Applet

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.

### JFrame(a top level window in Swing)

Whenever you create a graphical user interface with Java Swing functionality, you will need a container for your application. In the case of Swing, this container is called a JFrame. All GUI applications require a JFrame. In fact, some Applets even use a JFrame. Why?

You can't build a house without a foundation. The same is true in Java: Without a container in which to put all other elements, you won't have a GUI application. In other words, the JFrame is required as the foundation or base container for all other graphical components.

Java Swing applications can be run on any system that supports Java. These applications are lightweight. This means that don't take up much space or use many system resources.

JFrame is a class in Java and has its own methods and constructors. Methods are functions that impact the JFrame, such as setting the size or visibility. Constructors are run when the instance is created: One constructor can create a blank JFrame, while another can create it with a default title.

When a new JFrame is created, you actually create an instance of the JFrame class. You can create an empty one, or one with a title. If you pass a string into the constructor, a title is created as follows:

1. JFrame f = new JFrame();
2. // Or overload the constructor and give it a title:
3. JFrame f2 = new JFrame("The Twilight Zone");

## JPanel in Swing

The JPanel is a simplest container class. It provides space in which an application can attach any other component. It inherits the JComponents class. It doesn't have title bar.

JFrame f= **new** JFrame("Panel Example"); JPanel panel=**new** JPanel();

JLabel lbl=new JLabel(“Testing label”); panel.add(lbl);

f.add(panel);

## Swing Components and Containers

A component is an independent visual control. Swing Framework contains a large set of components which provide rich functionalities and allow high level of customization. They all are derived from JComponent class. All these components are lightweight components. This class provides some common functionality like pluggable look and feel, support for accessibility, drag and drop, layout, etc.

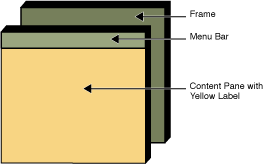
A container holds a group of components. It provides a space where a component can be managed and displayed. Containers are of two types:

##### Top level Containers

* + - It inherits Component and Container of AWT.
    - It cannot be contained within other containers.
    - Heavyweight.
    - Example: JFrame, JDialog, JApplet

##### Lightweight Containers

* + - It inherits JComponent class.
    - It is a general purpose container.
    - It can be used to organize related components together.
    - Example: JPanel



## JButton

JButton is in implementation of a push button. It is used to trigger an action if the user clicks on it. JButton can display a text, an icon, or both.

JButton btn=new JButton(“Click Me”);

### JLabel

Label is a simple component for displaying text, images or both. It does not react to input events.

JLabel lbl=new Label(“This is a label”);

### JTextField

JTextField is a text component that allows editing of a single line of non-formatted text. JTextField txt=new JTextField();

## JPasswordField

Same as JTextField, but with special symbols shown instead of actual texts.

### JCheckBox

JCheckBox is a box with a label that has two states: on and off. If the check box is selected, it is represented by a tick in a box. A check box can be used to show or hide a splashscreen at startup, toggle visibility of a toolbar etc.

JCheckBox chk1=new JCheckBox(“BCA”); JCheckBox chk2=new JCheckBox(“BBA”);

### JRadioButton

JRadioButton allows the user to select a single exclusive choice from a group of options. It is used with the ButtonGroup component.

JRadioButton rad1=new JRadioButton (“Male”);

JRadioButton red2=new JRadioButton (“Female”);

ButtonGroup grp=new ButtonGroup(); grp.add(rad1);

grp.add(rad2);

### JComboBox

JComboBox is a component that combines a button or editable field and a drop-down list. The user can select a value from the drop-down list, which appears at the user's request. If you make the combo box editable, then the combo box includes an editable field into which the user can type a value.

String arr[]={“BCA”,”BBA”,”MCA”,”MBA”};

JComboBox cmb=new JComboBox(arr);

//or

JComboBox<String> cmb=new JComboBox<String>(); cmb.addItem(“BCA”);

cmb.addItem(“BBA”);

cmb.addItem(“MCA”);

cmb.addItem(“MBA”);

### JList

JList is a component that displays a list of objects. It allows the user to select one or more items.

String arr[]={“BCA”,”BBA”,”MCA”,”MBA”};

JList cmb=new JList(arr);

### JTextArea

A JTextArea is a multiline text area that displays plain text. It is lightweight component for working with text. The component does not handle scrolling. For this task, we use JScrollPane component.

JTextArea txt=new JTextArea();

### JTable

The JTable class is a part of Java Swing Package and is generally used to display or edit two- dimensional data that is having both rows and columns. It is similar to a spreadsheet. This arranges data in a tabular form.

// Data to be displayed in the JTable String[][] data = {

{ "Kundan Kumar Jha", "4031", "CSE" },

{ "Anand Jha", "6014", "IT" }

};

// Column Names

String[] columnNames = { "Name", "Roll Number", "Department" };

// Initializing the JTable

JTable j = new JTable(data, columnNames);

### JMenu

The JMenuBar class is used to display menubar on the window or frame. It may have several menus.

The object of JMenu class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class.

The object of JMenuItem class adds a simple labeled menu item. The items used in a menu must belong to the JMenuItem or any of its subclass.

JMenuBar mb=new JMenuBar();

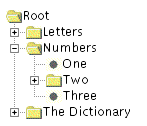
JMenu menu1=new JMenu(“Courses”);

JMenuItem item1=new JMenuItem(“BCA”); JMenuItem item2=new JMenuItem(“BBA”); menu1.add(item1);

menu1.add(item2); mb.add(menu1);

## JTree

With the JTree class, you can display hierarchical data. A JTree object does not actually contain your data; it simply provides a view of the data. Like any non-trivial Swing component, the tree gets data by querying its data model. Here is a picture of a tree:



## JSlider

The Java JSlider class is used to create the slider. By using JSlider, a user can select a value from a specific range.

JSlider slider = **new** JSlider(JSlider.HORIZONTAL, 0, 50, 25);

### Dialog Boxes in Swing

Dialog windows or dialogs are an indispensable part of most modern GUI applications. A dialog is defined as a conversation between two or more persons. In a computer application a dialog is a window which is used to "talk" to the application. A dialog is used to input data, modify data, change the application settings etc. Dialogs are important means of communication between a user and a computer program.

In Java Swing, we can create two kinds of dialogs: **standard dialogs and custom dialogs**. Custom dialogs are created by programmers. They are based on the JDialog class. Standard dialogs are predefined dialogs available in the Swing toolkit, for example the JColorChooser or the JFileChooser. These are dialogs for common programming tasks like showing text, receiving input, loading and saving files. They save programmer's time and enhance using some standard behaviour.

##### Modal and Modeless Dialog

There are two basic types of dialogs: modal and modeless. Modal dialogs block input to other top-level windows. Modeless dialogs allow input to other windows. An open file dialog is a good example of a modal dialog. While choosing a file to open, no other operation should be permitted. A typical modeless dialog is a find text dialog. It is handy to have the ability to move the cursor in the text control and define, where to start the finding of the particular text.

##### Standard Dialog Example – Message Box

Message dialogs are simple dialogs that provide information to the user. Message dialogs are created with the JOptionPane.showMessageDialog() method.

JOptionPane.showMessageDialog(null,”This is a test message”);

##### Custom Dialog Creation

JDialog jd=**new** JDialog(); jd.setTitle("This is a Test Dialog");

JLabel lbl=**new** JLabel("Do you want to exit? "); jd.add(lbl);

JButton yes=**new** JButton("Yes"); jd.add(yes);

# Addition program that uses JOptionPane for input and output.

import javax.swing.JOptionPane; // program uses JOptionPane public class Addition

{

public static void main( String[] args )

{

// obtain user input from JOptionPane input dialogs String firstNumber =

JOptionPane.showInputDialog( "Enter first integer" ); String secondNumber =

JOptionPane.showInputDialog( "Enter second integer" );

// convert String inputs to int values for use in a calculation int number1 = Integer.parseInt( firstNumber );

int number2 = Integer.parseInt( secondNumber ); int sum = number1 + number2; // add numbers

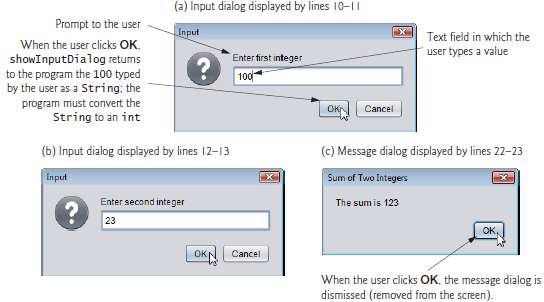
// display result in a JOptionPane message dialog JOptionPane.showMessageDialog( null, "The sum is " + sum,

"Sum of Two Integers", JOptionPane.PLAIN\_MESSAGE );

} // end method main

} // end class Addition

# Output



# Input Dialogs

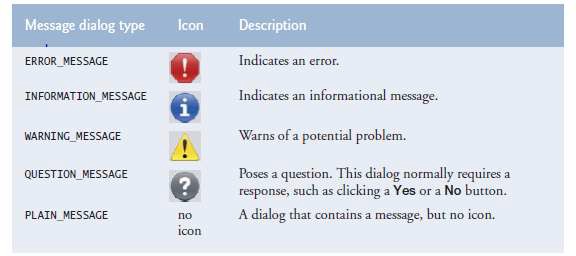
Line 3 imports class JOptionPane. Lines 10–11 declare the local String variable first- Number and assign it the result of the call to **JOptionPane static method showInputDialog**. This method displays an input dialog using the method’s String argument ("Enter first integer") as a prompt.

# Message Dailogs

Lines 22–23 use JOptionPane static method showMessageDialog to display a message dialog (the last screen of Fig. 1) containing the sum. The **first argument** helps the Java application determine **where to position the dialog box.** A dialog is typically displayed from a GUI application with its own window. The first argument refers to that window (known as the parent window) and causes the dialog to appear centered over the parent. If the first argument is null, the dialog box is displayed at the center of your screen. The **second argument** is the message to display—in this case, **the result of concatenating the String "The sum is " and the value of sum**. The **third argument**—**"Sum of Two Integers"—is the String that should appear in the title bar at the top of the dialog. The fourth argument— JOptionPane.PLAIN\_MESSAGE**—**is the type ofmessage dialog to display.** A PLAIN\_MESSAGE dialog does not display an icon to the left of the message.

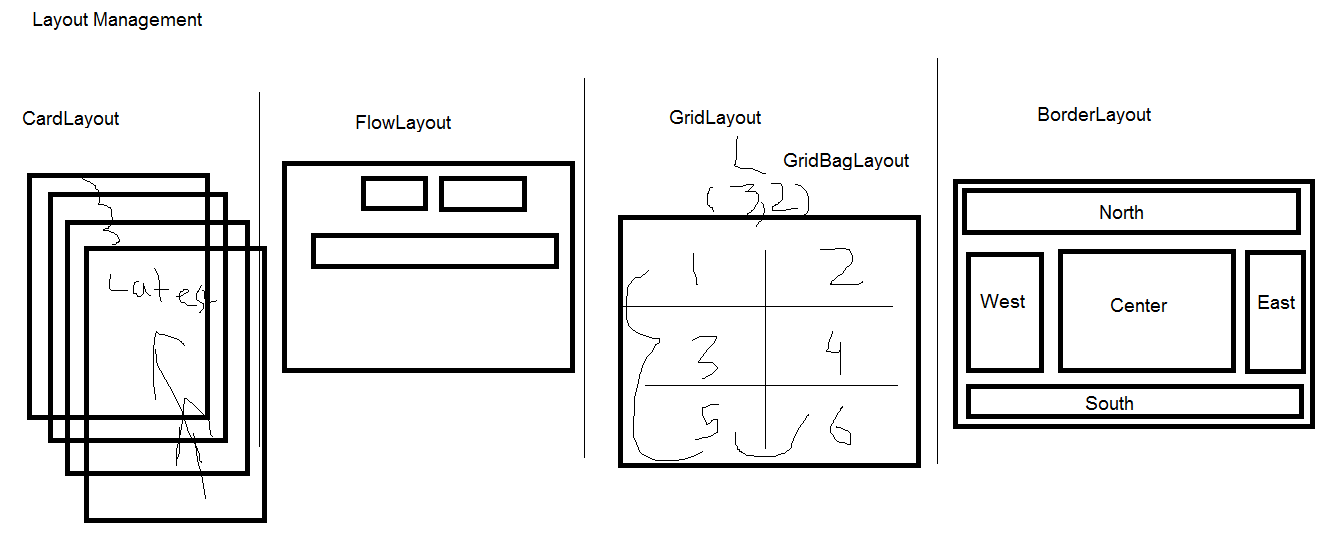
# JOptionPane Message Dialog Constants

The constants that represent the message dialog types are shown in Fig. 2. All message dialog types except PLAIN\_MESSAGE display an icon to the left of the message. These icons provide a visual indication of the message’s importance to the user. A QUESTION\_MESSAGE icon is the default icon for an input dialog box (see Fig. 2).



## Layout Management

In Java swing, Layout manager is used to position all its components, with setting properties, such as the size, the shape and the arrangement.



Following are the different types of layout managers:

1. Flow Layout
2. Border layout
3. Grid Layout

### Flow Layout

The FlowLayout arranges the components in a directional flow, either from left to right or from right to left. Normally all components are set to one row, according to the order of different components. If all components cannot be fit into one row, it will start a new row and fit the rest in.

import javax.swing.\*; import java.awt.FlowLayout;

public class FlowLayoutExample {

public static void main(String[] args) {

// Create and set up a frame window JFrame frame = new JFrame("Layout");

// Define new buttons

JButton jb1 = new JButton("Button 1"); JButton jb2 = new JButton("Button 2"); JButton jb3 = new JButton("Button 3");

// Define the panel to hold the buttons JPanel panel = new JPanel();

panel.setLayout(new FlowLayout()); //setting flowlayout panel.add(jb1);

panel.add(jb2); panel.add(jb3);

// Set the window to be visible as the default to be false frame.add(panel);

frame.pack(); frame.setVisible(true);

}

}



### Border Layout

A BorderLayout lays out a container, arranging its components to fit into five regions: NORTH, SOUTH, EAST, WEST and CENTER. For each region, it may contain no more than one component. When adding different components, you need to specify the orientation of it to be the one of the five regions.

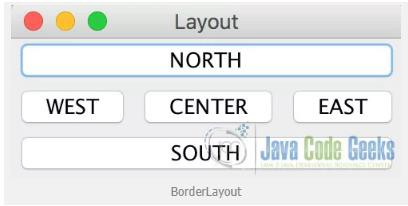
For BorderLayout, it can be constructed like below:

* **BorderLayout():** construct a border layout with no gaps between components.
* **BorderLayout(int hgap, int vgap):** construct a border layout with specified gaps between components.

// Define new buttons with different regions JButton jb1 = new JButton("NORTH");

JButton jb2 = new JButton("SOUTH"); JButton jb3 = new JButton("WEST"); JButton jb4 = new JButton("EAST"); JButton jb5 = new JButton("CENTER");

// Define the panel to hold the buttons JPanel panel = new JPanel(); panel.setLayout(new BorderLayout()); panel.add(jb1, BorderLayout.NORTH); panel.add(jb2, BorderLayout.SOUTH); panel.add(jb3, BorderLayout.WEST); panel.add(jb4, BorderLayout.EAST); panel.add(jb5, BorderLayout.CENTER);



### Grid Layout

The GridLayout manager is used to lay out the components in a rectangle grid, which has been divided into equal-sized rectangles and one component is placed in each rectangle. It can constructed with following methods:

* **GridLayout():** construct a grid layout with one column per component in a single row.
* **GridLayout(int row, int col):** construct a grid layout with specified numbers of rows and columns.
* **GridLayout(int row, int col, int hgap, int vgap):** construct a grid layout with specified rows, columns and gaps between components.

// Define new buttons

JButton jb1 = new JButton("Button 1"); JButton jb2 = new JButton("Button 2"); JButton jb3 = new JButton("Button 3"); JButton jb4 = new JButton("Button 4"); JButton jb5 = new JButton("Button 5");

// Define the panel to hold the buttons JPanel panel = new JPanel(); panel.setLayout(new GridLayout(3, 2)); panel.add(jb1);

panel.add(jb2); panel.add(jb3); panel.add(jb4); panel.add(jb5);



## MDI using JDesktop Pane and JInternal Frame

MDI- Multi Document Interface

The JDesktopPane class, can be used to create "multi-document" applications. A multi- document application can have many windows included in it. The JDesktopPane is a container which is used to create a multiple-document interface or a virtual desktop.The JFrame inside the desktop becomes JInternalFrame. JInternalFrame is used just like the JFrame but is added to JDesktopPane object.

JDesktopPane jd=**new** JDesktopPane();

JInternalFrame frame1=**new** JInternalFrame("Internal Frame1",**true**,**true**,**true**,**true**);

frame1.setLayout(**null**); frame1.setSize(200,100); frame1.setVisible(**true**);

JInternalFrame frame2=**new** JInternalFrame("Internal Frame2",**true**,**true**,**true**,**true**);

frame2.setLayout(**null**); frame2.setSize(200,100); frame2.setVisible(**true**);

JInternalFrame frame3=**new** JInternalFrame("Internal Frame3",**true**,**true**,**true**,**true**);

frame3.setLayout(**null**); frame3.setSize(200,100); frame3.setVisible(**true**); jd.add(frame1); jd.add(frame2); jd.add(frame3);

# Event Handling

Any program that uses GUI (graphical user interface) such as Java application written for windows, is event driven. Event describes the change in state of any object. For Example: Pressing a button, entering a character in Textbox, Clicking or Dragging a mouse, etc.

Event handling has three main components,

* **Events:** An event is a change in state of an object.
* **Events Source:** Event source is an object that generates an event.
* **Listeners:** A listener is an object that listens to the event. A listener gets notified when an event occurs.

##### How Events are handled?

A source generates an Event and send it to one or more listeners registered with the source. Once event is received by the listener, they process the event and then return. Events are supported by a number of Java packages, like **java.util**, **java.awt** and **java.awt.event**.

**Important Event Classes and Interface**

|  |  |  |
| --- | --- | --- |
| **Event Classes** | **Description** | **Listener Interface** |
| **ActionEvent** | generated when button is pressed, menu-item is selected, list-item is double clicked | ActionListener |
| **MouseEvent** | generated when mouse is dragged, moved,clicked,pressed or released and also when it enters or exit a component | MouseListener |
| **KeyEvent** | generated when input is received from keyboard | KeyListener |
| **ItemEvent** | generated when check-box or list item is clicked | ItemListener |
| **TextEvent** | generated when value of textarea or textfield is changed | TextListener |
| **MouseWheelEvent** | generated when mouse wheel is moved | MouseWheelListener |
| **WindowEvent** | generated when window is activated, deactivated, deiconified, iconified, opened or closed | WindowListener |
| **ComponentEvent** | generated when component is hidden, moved, resized or set visible | ComponentEventListener |
| **ContainerEvent** | generated when component is added or removed from container | ContainerListener |
| **AdjustmentEvent** | generated when scroll bar is manipulated | AdjustmentListener |
| **FocusEvent** | generated when component gains or loses keyboard focus | FocusListener |

#### Example of ActionListener

JButton btn=new JButton(“Click Here”); btn.addActionListener(new ActionListener(){

pubic void actionPerformed(ActionEvent ae){

//event handing code here

}

});

#### Example of ItemListener

JComboBox cmb=new JComboBox(arr); cmb.addItemListener(new ItemListener(){

pubic void actionPerformed(ItemEvent ie){

if (ie.getStateChange() == ItemEvent.SELECTED)

//event handing code here

}

});

### Keyboard and Mouse Events

The Java **MouseListener** is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package. It has five methods.

**void** mouseClicked(MouseEvent e); **void** mouseEntered(MouseEvent e); **void** mouseExited(MouseEvent e); **void** mousePressed(MouseEvent e); **void** mouseReleased(MouseEvent e);

##### Mouse Event Example

**import** java.awt.\*;

**import** java.awt.event.\*;

**public class** MouseListenerExample **extends** Frame **implements** MouseListener

{

Label l; MouseListenerExample(){

addMouseListener(**this**);

l=**new** Label(); l.setBounds(20,50,100,20); add(l);

setSize(300,300); setLayout(**null**); setVisible(**true**);

}

**public void** mouseClicked(MouseEvent e) { l.setText("Mouse Clicked");

}

**public void** mouseEntered(MouseEvent e) { l.setText("Mouse Entered");

}

**public void** mouseExited(MouseEvent e) { l.setText("Mouse Exited");

}

**public void** mousePressed(MouseEvent e) { l.setText("Mouse Pressed");

}

**public void** mouseReleased(MouseEvent e) { l.setText("Mouse Released");

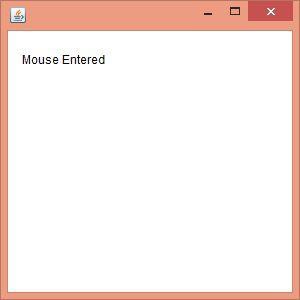
}

**public static void** main(String[] args) {

**new** MouseListenerExample();

}

}



The Java **KeyListener** is notified whenever you change the state of key. It is notified against KeyEvent. The KeyListener interface is found in java.awt.event package. It has three methods.

**void** keyPressed(KeyEvent e); **void** keyReleased(KeyEvent e); **void** keyTyped(KeyEvent e);

##### Key Event Example

**import** java.awt.\*;

**import** java.awt.event.\*;

**public class** KeyListenerExample **extends** Frame **implements** KeyListener{ Label l;

TextArea area; KeyListenerExample(){

l=**new** Label(); l.setBounds(20,50,100,20); area=**new** TextArea(); area.setBounds(20,80,300, 300); area.addKeyListener(**this**); add(l);add(area); setSize(400,400); setLayout(**null**); setVisible(**true**);

}

**public void** keyPressed(KeyEvent e) { l.setText("Key Pressed");

}

**public void** keyReleased(KeyEvent e) { l.setText("Key Released");

}

**public void** keyTyped(KeyEvent e) { l.setText("Key Typed");

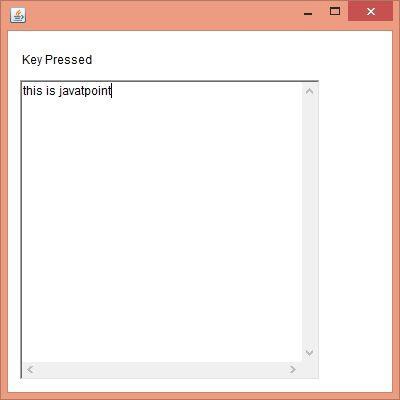
}

**public static void** main(String[] args) {

**new** KeyListenerExample();

}

}



### Java Swing Examples

##### Example 1

**import** java.awt.event.\*; **import** javax.swing.\*; **public class** SwingExample{

SwingExample(){

JFrame jframe=**new** JFrame("This App contains almost all Elements of Swing");

jframe.setSize(600, 500); jframe.setLocationRelativeTo(**null**); jframe.setLayout(**null**); jframe.setVisible(**true**);

//Name

JLabel lblName=**new** JLabel("Name: "); lblName.setBounds(30, 12, 150, 10); jframe.add(lblName);

JTextField txtName=**new** JTextField(); txtName.setBounds(100, 10, 150, 20); jframe.add(txtName);

//Address

JLabel lblAddress=**new** JLabel("Address: "); lblAddress.setBounds(300, 12, 150, 10); jframe.add(lblAddress);

JTextField txtAddress=**new** JTextField(); txtAddress.setBounds(370, 10, 150, 20); jframe.add(txtAddress);

//Class

JLabel lblClass=**new** JLabel("Class: "); lblClass.setBounds(30, 60, 150, 10); jframe.add(lblClass);

JComboBox<String> cmbClass=**new** JComboBox<String>(); cmbClass.addItem("Select Class"); cmbClass.addItem("BCA");

cmbClass.addItem("BBA"); cmbClass.addItem("MCA"); cmbClass.addItem("MBA");

/\*String[] clas= {"BCA","BBA"};

JComboBox cmbClass=new JComboBox(clas);\*/

cmbClass.setBounds(100, 55, 150, 20);

//cmbClass.setEditable(true); jframe.add(cmbClass);

//Gender

JLabel lblSex=**new** JLabel("Gender: "); lblSex.setBounds(300,60,80,10); jframe.add(lblSex);

ButtonGroup group1=**new** ButtonGroup();

JRadioButton chkMale=**new** JRadioButton("Male",**true**); chkMale.setBounds(360, 55, 70, 20); jframe.add(chkMale);

group1.add(chkMale);

JRadioButton chkFemale=**new** JRadioButton("Female"); chkFemale.setBounds(430, 55, 70, 20); jframe.add(chkFemale);

group1.add(chkFemale);

//Shift

JLabel lblShift=**new** JLabel("Shift: "); lblShift.setBounds(30,100,80,10); jframe.add(lblShift);

String[] shift= {"Morning","Day","Evening"}; JList list1=**new** JList(shift); list1.setBounds(100, 100, 100, 60);

jframe.add(list1);

//Remarks

JLabel lblRemarks=**new** JLabel("Remarks: "); lblRemarks.setBounds(300,100,80,10); jframe.add(lblRemarks);

JTextArea txtArea=**new** JTextArea(); txtArea.setBounds(400, 100, 150, 100); jframe.add(txtArea);

//JTable

String data[][]={ {"101","Amit","670000"},

{"102","Jai","780000"},

{"101","Sachin","700000"}};

String column[]={"ID","NAME","SALARY"};

JTable jt=**new** JTable(data,column); JScrollPane sp=**new** JScrollPane(jt); sp.setBounds(30,200,250,80);

jframe.add(sp);

JButton click=**new** JButton("Submit"); click.setBounds(200,300,100,30); jframe.add(click);

//creating menu

JMenuBar mb=**new** JMenuBar(); mb.setBounds(30, 350, 200, 20); JMenu menu1=**new** JMenu("File"); JMenuItem item1,item3; item1=**new** JMenuItem("New"); JMenu item2=**new** JMenu("Open"); JMenuItem i1,i2;

i1=**new** JMenuItem("Open Project"); i2=**new** JMenuItem("Open File"); item2.add(i1);

item2.add(i2);

item3=**new** JMenuItem("Close"); menu1.add(item1); menu1.add(item2); menu1.add(item3); mb.add(menu1);

JMenu menu2=**new** JMenu("Edit"); JMenuItem it1,it2;

it1=**new** JMenuItem("Copy"); it2=**new** JMenuItem("Paste"); menu2.add(it1); menu2.add(it2); mb.add(menu2); jframe.add(mb);

click.addActionListener(**new** ActionListener() {

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

//get textfield

String name=txtName.getText();

//get combo

String clas=cmbClass.getSelectedItem().toString();

//get radio + checkbox (same process) String gender=""; **if**(chkMale.isSelected())

gender=chkMale.getText();

**else**

gender=chkFemale.getText();

//getList

String shift=list1.getSelectedValuesList().toString();

//get textarea

String remarks=txtArea.getText(); JOptionPane.*showMessageDialog*(**null**, "Remarks: "+remarks);

}

});

//can use action listener also cmbClass.addItemListener(**new** ItemListener() {

**public void** itemStateChanged(ItemEvent e) { **if**(e.getStateChange()==ItemEvent.***SELECTED***) JOptionPane.*showMessageDialog*(**null**, cmbClass.getSelectedItem().toString());

}

}

});

// listener on menu item

//can use item listener also item1.addActionListener(**new** ActionListener() {

**public void** actionPerformed(ActionEvent ae) { JOptionPane.*showMessageDialog*(**null**, item1.getText().toString()+" Clicked!");

}

});

}

**public static void** main(String[] args) { SwingUtilities.*invokeLater*(**new** Runnable() {

**public void** run() {

**new** SwingExample();

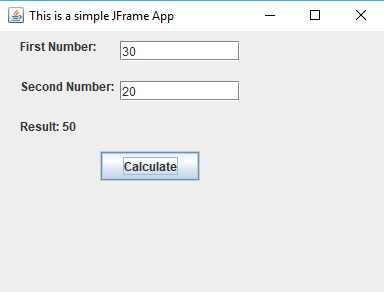
}

});

}

}

**Example 2**



**import** javax.swing.\*; **import** java.awt.event.\*; **public class** SwingDemo {

SwingDemo(){

JFrame jframe=**new** JFrame("This is a simple JFrame App"); jframe.setSize(400, 300); jframe.setLocationRelativeTo(**null**); jframe.getContentPane().setLayout(**null**); jframe.setVisible(**true**);

JLabel lbl1=**new** JLabel("First Number:"); lbl1.setBounds(20, 10, 100, 10); jframe.add(lbl1);

JTextField txt1=**new** JTextField(); txt1.setBounds(120, 10, 120, 20); jframe.add(txt1);

JLabel lbl2=**new** JLabel("Second Number:"); lbl2.setBounds(20, 50, 100, 10); jframe.add(lbl2);

JTextField txt2=**new** JTextField(); txt2.setBounds(120, 50, 120, 20); jframe.add(txt2);

JLabel lbl3=**new** JLabel("Result: ");

//lbl3.setText("Result: ");

lbl3.setBounds(20,80,100,30); jframe.add(lbl3);

JButton btn=**new** JButton("Calculate"); btn.setBounds(100, 120, 100, 30); jframe.add(btn);

btn.addActionListener(**new** ActionListener() {

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

String first1=txt1.getText().toString(); String second1=txt2.getText().toString(); **int** a,b,c;

a=Integer.*parseInt*(first1); b=Integer.*parseInt*(second1); c=a+b;

lbl3.setText("Result: "+c);

//JOptionPane.showMessageDialog(null, "Addition= "+c);

}

});

}

**public static void** main(String[] args) { SwingUtilities.*invokeLater*(**new** Runnable() {

**public void** run() {

**new** SwingDemo();

}

});

}

}