UNIT-V

EVENT DRIVEN PROGRAMMING

Graphics programming-Frame-Components-working with 2D shapes-Using color, fonts, and images-Basics of event Handling-event handlers-adapter classes-actions mouse events-AWT event hierarchy-Introduction to Swing-layout management-Swing Components-Text Fields, Text Areas-Buttons-Check Boxes-Radio Buttons-Lists-choices-Scrollbars-windows-Menus-Dialog Boxes and Interfaces, Exception handling, Multithreaded programming, Strings, Input/output

Graphics programming

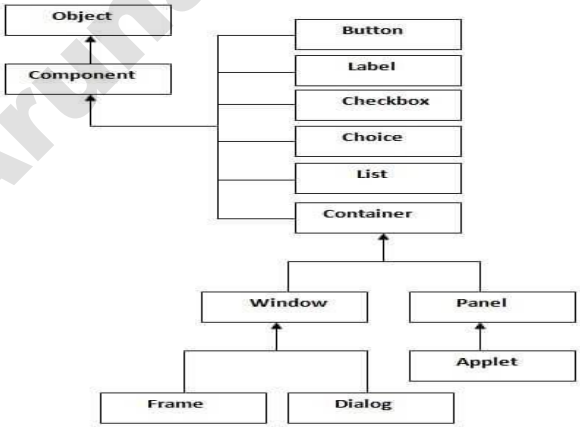
- Java contains support for graphics that enable programmers to visually enhance applications
- Java contains many more sophisticated drawing capabilities as part of the Java 2D API

AWT

- Java AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java.
- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system.
- AWT is heavyweight i.e. its components are using the resources of OS.The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.



Container

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extend Container class are known as container such as Frame, Dialog and Panel.

Window

The window is the container that has no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panel

The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

There are two ways to create a Frame. They are,

- By Instantiating Frame class
- By extending Frame class

Example:

```
import java.awt.*;
import java.awt.event.*;
class MyLoginWindow extends Frame
{
   TextField name,pass;
   Button b1,b2;
   MyLoginWindow()
   {
      setLayout(new FlowLayout());
      this.setLayout(null);
      Label n=new Label("Name:",Label.CENTER);
      Label p=new Label("password:",Label.CENTER);
      name=new TextField(20);
      pass=new TextField(20);
      pass.setEchoChar('#');
      b1=new Button("submit");
      b2=new Button("cancel");
      this.add(n);
      this.add(name);
      this.add(p);
      this.add(pass);
      this.add(b1);
      this.add(b2);
```

```
    n.setBounds(70,90,90,60);
    p.setBounds(70,130,90,60);
    name.setBounds(200,100,90,20);
    pass.setBounds(200,140,90,20);
    b1.setBounds(100,260,70,40);
    b2.setBounds(180,260,70,40);
  }
  public static void main(String args[])
  {
    MyLoginWindow ml=new MyLoginWindow();
    ml.setVisible(true);
    ml.setSize(400,400);
    ml.setTitle("my login window");
   }}
```

Output:

Event handling:

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling. 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 occur

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

Steps to handle events:

- Implement appropriate interface in the class.
- Register the component with the listener.

How to implement Listener

1. Declare an event handler class and specify that the class either implements an ActionListener(any listener) interface or extends a class that implements an ActionListener interface. For example:

```
public class MyClass implements ActionListener
{
// Set of Code
}
```

2. Register an instance of the event handler class as a listener on one or more components. For example:

```
someComponent.addActionListener(instanceOfMyClass);
```

3. Include code that implements the methods in listener interface. For example:

```
public void actionPerformed(ActionEvent e) {
   //code that reacts to the action
}
```

Mouse Listener

```
package Listener;
import java.awt.Frame;
import java.awt.Label;
import java.awt.TextArea;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
```

```
public class Mouse implements MouseListener {
        TextArea s;
public Mouse()
{
        Frame d=new Frame("kkkk");
        s=new TextArea("");
        d.add(s);
        s.addMouseListener(this);
        d.setSize(190, 190);
   d.show();
  }
   public void mousePressed(MouseEvent e) {
     System.out.println("MousePressed");
     int a=e.getX();
        int b=e.getY();
        System.out.println("X="+a+"Y="+b);
   }
   public void mouseReleased(MouseEvent e) {
        System.out.println("MouseReleased");
        }
   public void mouseEntered(MouseEvent e) {
        System.out.println("MouseEntered");
   }
   public void mouseExited(MouseEvent e) {
        System.out.println("MouseExited");
   }
   public void mouseClicked(MouseEvent e) {
        System.out.println("MouseClicked");
     }
   public static void main(String arg[])
   {
   Mouse a=new Mouse();
```

```
    }
}
```

Mouse Motion Listener

```
package Listener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextArea;
public class MouseMotionEventDemo extends JPanel implements MouseMotionListener {
MouseMotionEventDemo()
{
        JTextArea a=new JTextArea();
        a.addMouseMotionListener(this);
        JFrame b=new JFrame();
        b.add(a);
        b.setVisible(true);
}
public void mouseMoved(MouseEvent e) {
System.out.println("Mouse is Moving");
}
public void mouseDragged(MouseEvent e) {
        System.out.println("MouseDragged");
}
public static void main(String arg[])
{
        MouseMotionEventDemo a=new MouseMotionEventDemo();
}
}
```

KEY LISTENER

```
package Listener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.JFrame;
import javax.swing.JTextField;
public class KeyEventDemo implements KeyListener
{
   public KeyEventDemo()
   {
   JFrame s=new JFrame("hai");
         JTextField typingArea = new JTextField(20);
         typingArea.addKeyListener(this);
         s.add(typingArea);
         s.setVisible(true);
   }
        public void keyTyped(KeyEvent e){
         System.out.println("KeyTyped");
   }
   /** Handle the key-pressed event from the text field. */
   public void keyPressed(KeyEvent e) {
         System.out.println("KeyPressed");
   }
   /** Handle the key-released event from the text field. */
   public void keyReleased(KeyEvent e) {
         System.out.println("Keyreleased");
   }
  public static void main(String g[])
  {
           KeyEventDemo a=new KeyEventDemo();
  }
     }
```

ITEM LISTENER

```
package Listener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JFrame;
import javax.swing.JTextField;
public class itemlistener implements ItemListener
{
   public itemlistener()
   {
   JFrame s=new JFrame("hai");
         JCheckBox a=new JCheckBox("Ok");
         a.addItemListener(this);
         s.add(a);
         s.setVisible(true);
   }
     public static void main(String g[])
  {
            itemlistener l=new itemlistener();
  }
public void itemStateChanged(ItemEvent arg0) {
System.out.println("State changed");
}
   }
```

Window Listener

```
package Listener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextArea;
public class window extends JPanel implements WindowListener {
window()
{
        JFrame b=new JFrame();
        b.addWindowListener(this);
        b.setVisible(true);
}
public static void main(String arg[])
{
        window a=new window();
}
public void windowActivated(WindowEvent arg0) {
        System.out.println("Window activated");
        }
public void windowClosed(WindowEvent arg0) {
        // TODO Auto-generated method stub
        System.out.println("Window closed");
}
public void windowClosing(WindowEvent arg0) {
        // TODO Auto-generated method stub
        System.out.println("Window closing");
}
public void windowDeactivated(WindowEvent arg0) {
        // TODO Auto-generated method stub
```

```
        System.out.println("Window deactivated");
}
public void windowDeiconified(WindowEvent arg0) {
        // TODO Auto-generated method stub
        System.out.println("Window deiconified");
}
public void windowIconified(WindowEvent arg0) {
        // TODO Auto-generated method stub
        System.out.println("Window Iconified");
}
public void windowOpened(WindowEvent arg0) {
        // TODO Auto-generated method stub
        System.out.println("Window opened");
}}
```

WINDOW FOCUS LISTENER

```
package Listener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
import java.awt.event.WindowEvent;
import java.awt.event.WindowFocusListener;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextArea;
public class window1 extends JPanel implements WindowFocusListener {
window1()
{
        JFrame b=new JFrame();
        b.addWindowFocusListener(this);
        b.setVisible(true);
}
```

```
public static void main(String arg[])
{
        window1 b=new window1();
}
public void windowGainedFocus(WindowEvent e) {
        // TODO Auto-generated method stub
         System.out.println("Window  gained");
}
public void windowLostFocus(WindowEvent e) {
        // TODO Auto-generated method stub
        System.out.println("Windowlostfocus");
}}
```

WindowStateListener

```
package Listener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
import java.awt.event.WindowEvent;
import java.awt.event.WindowStateListener;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextArea;
public class window2 extends JPanel implements WindowStateListener {
window2()
{
        JFrame b=new JFrame();
        b.addWindowStateListener(this);
        b.setVisible(true);
}
public static void main(String arg[])
{
        window2 b=new window2();
```

```
}
public void windowStateChanged(WindowEvent e) {
        // TODO Auto-generated method stub
        System.out.println("State Changed");
}}
```

ACTION LISTENER

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.event.*;
public class A extends JFrame implements ActionListener {
      Scientific() {
          JPanel buttonpanel = new JPanel();
          JButton b1 = new JButton("Hai");
          buttonpanel.add(b1);
          b1.addActionListener(this);
     }
     public void actionPerformed(ActionEvent e) {
          System.out.println(“Hai button”);
      }
      public static void main(String args[]) {
            A f = new A();
          f.setTitle("ActionListener");
          f.setSize(500,500);
          f.setVisible(true);
     }}
```

Java adapter classes

Java adapter classes provide the default implementation of listener interfaces. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it saves code.

The adapter classes are found in java.awt.event, java.awt.dnd and javax.swing.event packages.

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

Java WindowAdapter Example

```
import java.awt.*;
import java.awt.event.*;
public class AdapterExample{
   Frame f;
   AdapterExample(){
      f=new Frame("Window Adapter");
      f.addWindowListener(new WindowAdapter(){
         public void windowClosing(WindowEvent e) {
            f.dispose();
         }
      });
        f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
```

```
    }
  public static void main(String[] args) {
    new AdapterExample();
  } }
```

Java MouseAdapter Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseAdapterExample extends MouseAdapter{
  Frame f;
  MouseAdapterExample(){
    f=new Frame("Mouse Adapter");
    f.addMouseListener(this);
    f.setSize(300,300);
    f.setLayout(null);
    f.setVisible(true);
  }
  public void mouseClicked(MouseEvent e) {
    Graphics g=f.getGraphics();
    g.setColor(Color.BLUE);
    g.fillOval(e.getX(),e.getY(),30,30);
  }
   public static void main(String[] args) {
  new MouseAdapterExample();
} }
```

Java MouseMotionAdapter Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseMotionAdapterExample extends MouseMotionAdapter{
  Frame f;
  MouseMotionAdapterExample(){
    f=new Frame("Mouse Motion Adapter");
```

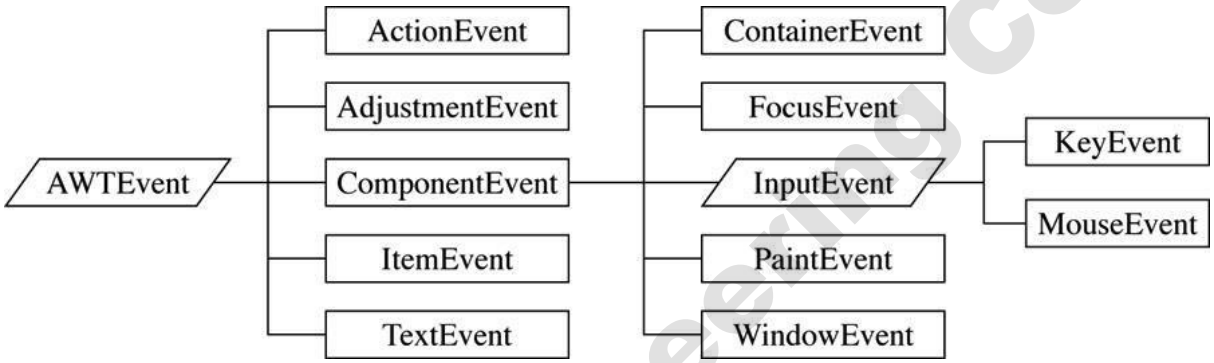
```
        f.addMouseMotionListener(this);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
public void mouseDragged(MouseEvent e) {
    Graphics g=f.getGraphics();
    g.setColor(Color.ORANGE);
    g.fillOval(e.getX(),e.getY(),20,20);
}
public static void main(String[] args) {
    new MouseMotionAdapterExample();
} }
```

Java KeyAdapter Example

```
import java.awt.*;
import java.awt.event.*;
public class KeyAdapterExample extends KeyAdapter{
    Label l;
    TextArea area;
    Frame f;
    KeyAdapterExample(){
        f=new Frame("Key Adapter");
        l=new Label();
        l.setBounds(20,50,200,20);
        area=new TextArea();
        area.setBounds(20,80,300, 300);
        area.addKeyListener(this);
        f.add(l);f.add(area);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
```

```
public void keyReleased(KeyEvent e) {
   String text=area.getText();
   String words[]=text.split("\\s");
   l.setText("Words: "+words.length+" Characters:"+text.length());
}
  public static void main(String[] args) {
   new KeyAdapterExample();
} }
```

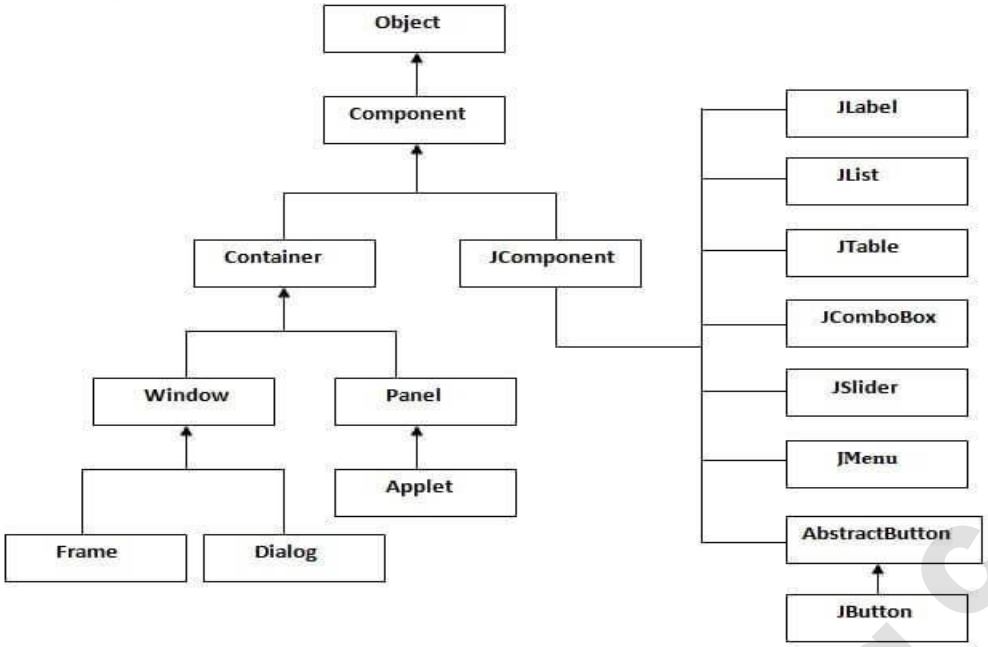
AWT EVENT HIERARCHY



Swing

- Java Swing tutorial is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.
- Unlike AWT, Java Swing provides platform-independent and lightweight components.
- The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

The hierarchy of java swing API is given below



Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components**such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC**.

Layout management

Java LayoutManagers

The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

AWT Layout Manager Classes

Following is the list of commonly used controls while designing GUI using AWT.

Sr.No.	LayoutManager & Description
1	**BorderLayout** The borderlayout arranges the components to fit in the five regions: east, west, north, south, and center.
2	**CardLayout** The CardLayout object treats each component in the container as a card. Only one card is visible at a time.
3	**FlowLayout** The FlowLayout is the default layout. It layout the components in a directional flow.
4	**GridLayout** The GridLayout manages the components in the form of a rectangular grid.
5	**GridBagLayout** This is the most flexible layout manager class. The object of GridBagLayout aligns the component vertically, horizontally, or along their baseline without requiring the components of the same size.
6	**GroupLayout** The GroupLayout hierarchically groups the components in order to position them in a Container.
7	**SpringLayout** A SpringLayout positions the children of its associated container according to a set of constraints.
8	**BoxLayout** The BoxLayout is used to arrange the components either vertically or horizontally.
9	**ScrollPaneLayout** The layout manager used by JScrollPane. JScrollPaneLayout is responsible for nine components: a viewport, two scrollbars, a row header, a column header, and four "corner" components.

Border layout:

Example:

```
import java.awt.*;
import javax.swing.*;
public class Border {
JFrame f;
Border(){
   f=new JFrame();
      JButton b1=new JButton("NORTH");;
   JButton b2=new JButton("SOUTH");;
   JButton b3=new JButton("EAST");;
   JButton b4=new JButton("WEST");;
   JButton b5=new JButton("CENTER");;
        f.add(b1,BorderLayout.NORTH);
   f.add(b2,BorderLayout.SOUTH);
   f.add(b3,BorderLayout.EAST);
   f.add(b4,BorderLayout.WEST);
   f.add(b5,BorderLayout.CENTER);
   f.setSize(300,300);
   f.setVisible(true);
}
public static void main(String[] args) {
   new Border();
} }
```

ScrollPaneLayout:

```
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
public class ScrollPaneDemo extends JFrame
```

```
{
public ScrollPaneDemo() {
super("ScrollPane Demo");
ImageIcon img = new ImageIcon("child.png");
  JScrollPane png = new JScrollPane(new JLabel(img));
  getContentPane().add(png);
setSize(300,250);
setVisible(true);
}
  public static void main(String[] args) {
new ScrollPaneDemo();
}  }
```

Boxlayout

```
import java.awt.*;
import javax.swing.*;
  public class BoxLayoutExample1 extends Frame {
 Button buttons[];
   public BoxLayoutExample1 () {
   buttons = new Button [5];
       for (int i = 0;i<5;i++) {
      buttons[i] = new Button ("Button " + (i + 1));
      add (buttons[i]);
    }
  setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));
setSize(400,400);
setVisible(true);
}
  public static void main(String args[]){
BoxLayoutExample1 b=new BoxLayoutExample1();
}
```

Group layout:

Example

```
public class GroupExample
{
   public static void main(String[] args)
{
      JFrame frame = new JFrame("GroupLayoutExample");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container contentPanel = frame.getContentPane();
      GroupLayout groupLayout = new GroupLayout(contentPanel);
      contentPanel.setLayout(groupLayout);
       JLabel clickMe = new JLabel("Click Here");
      JButton button = new JButton("This Button");
      groupLayout.setHorizontalGroup(
               groupLayout.createSequentialGroup()
                        .addComponent(clickMe)
                        .addGap(10, 20, 100)
                        .addComponent(button));
      groupLayout.setVerticalGroup(
               groupLayout.createParallelGroup(GroupLayout.Alignment.BASELINE)
                        .addComponent(clickMe)
                        .addComponent(button));
      frame.pack();
      frame.setVisible(true);
  } } }
```

Swing components:

Text Fields

The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class.

Text Areas

The object of a JTextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits JTextComponent class

Buttons

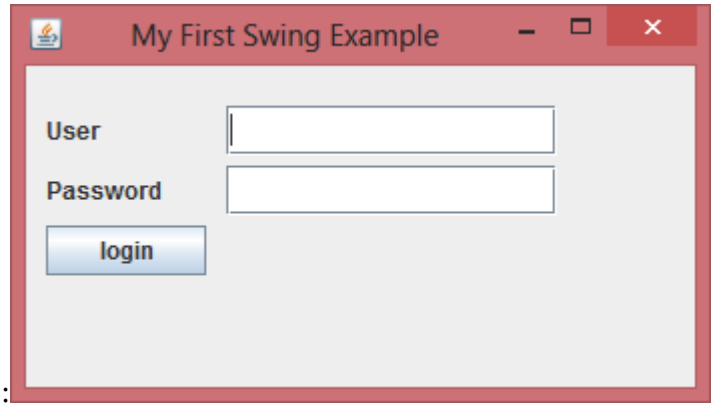
The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

```
import javax.swing.JButton;
import javax.swing.JFrame;
import  javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;
public class SwingFirstExample {
    public static void main(String[] args) {
    // Creating instance of JFrame
    JFrame frame = new JFrame("My First Swing Example");
    // Setting the width and height of frame
    frame.setSize(350, 200);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    /* Creating panel. This is same as a div tag in HTML
     * We can create several panels and add them to specific
     * positions in a JFrame. Inside panels we can add text
     * fields, buttons and other components.
     */
    JPanel panel = new JPanel();
    // adding panel to frame
    frame.add(panel);
    /* calling user defined method for adding components
     * to the panel.
     */
    placeComponents(panel);
    // Setting the frame visibility to true
    frame.setVisible(true);     }
  private static void placeComponents(JPanel panel) {
    /* We will discuss about layouts in the later sections          * of this tutorial. For now we are setting the
layout       * to null       */
```

```
    panel.setLayout(null);
    // Creating JLabel
    JLabel userLabel = new JLabel("User");
    /* This method specifies the location and size
     * of component. setBounds(x, y, width, height)
     * here (x,y) are cordinates from the top left
     * corner and remaining two arguments are the width
     * and height of the component.
     */
    userLabel.setBounds(10,20,80,25);
    panel.add(userLabel);
    /* Creating text field where user is supposed to
     * enter user name.
     */
    JTextField userText = new JTextField(20);
    userText.setBounds(100,20,165,25);
    panel.add(userText);
    // Same process for password label and text field.
    JLabel passwordLabel = new JLabel("Password");
    passwordLabel.setBounds(10,50,80,25);
    panel.add(passwordLabel);
    /*This is similar to text field but it hides the user
     * entered data and displays dots instead to protect
     * the password like we normally see on login screens.
     */
    JPasswordField passwordText = new JPasswordField(20);
    passwordText.setBounds(100,50,165,25);
    panel.add(passwordText);
    // Creating login button
    JButton loginButton = new JButton("login");
    loginButton.setBounds(10, 80, 80, 25);
    panel.add(loginButton);
}}
```

Output:

:



Check Boxes

The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".It inherits JToggleButton class.

Example:

```
import javax.swing.*;
public class CheckBoxExample
{
    CheckBoxExample(){
      JFrame f= new JFrame("CheckBox Example");
      JCheckBox checkBox1 = new JCheckBox("C++");
      checkBox1.setBounds(100,100, 50,50);
      JCheckBox checkBox2 = new JCheckBox("Java", true);
      checkBox2.setBounds(100,150, 50,50);
      f.add(checkBox1);
      f.add(checkBox2);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
    }
public static void main(String args[])
   {
   new CheckBoxExample();
```

}}

Radio Buttons

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz. It should be added in ButtonGroup to select one radio button only.

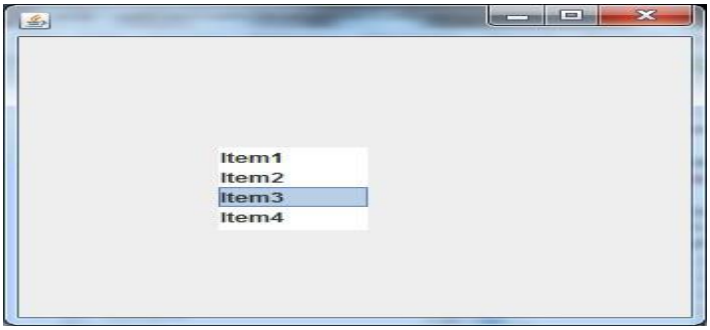
```
import javax.swing.*;
import java.awt.event.*;
class RadioButtonExample extends JFrame implements ActionListener{
JRadioButton rb1,rb2;
JButton b;
RadioButtonExample(){
rb1=new JRadioButton("Male");
rb1.setBounds(100,50,100,30);
rb2=new JRadioButton("Female");
rb2.setBounds(100,100,100,30);
ButtonGroup bg=new ButtonGroup();
bg.add(rb1);bg.add(rb2);
b=new JButton("click");
b.setBounds(100,150,80,30);
b.addActionListener(this);
add(rb1);add(rb2);add(b);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
if(rb1.isSelected()){
JOptionPane.showMessageDialog(this,"You are Male.");
}
if(rb2.isSelected()){
JOptionPane.showMessageDialog(this,"You are Female.");
}  }
```

```
public static void main(String args[]){
new RadioButtonExample();
}}
```

Lists

The object of JList class represents a list of text items. The list of text items can be set up so that the user can choose either one item or multiple items. It inherits JComponent class.

```
import javax.swing.*;
public class ListExample
{
    ListExample(){
       JFrame f= new JFrame();
       DefaultListModel<String> l1 = new DefaultListModel<>();
         l1.addElement("Item1");
         l1.addElement("Item2");
         l1.addElement("Item3");
         l1.addElement("Item4");
         JList<String> list = new JList<>(l1);
         list.setBounds(100,100, 75,75);
         f.add(list);
         f.setSize(400,400);
         f.setLayout(null);
         f.setVisible(true);
    }
public static void main(String args[])
   {
  new ListExample();
   }}
```

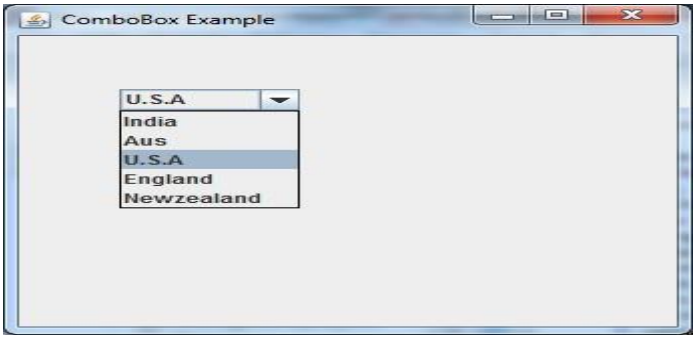


Choices (JComboBox)

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits JComponent class.

```
import javax.swing.*;
public class ComboBoxExample {
JFrame f;
ComboBoxExample(){
   f=new JFrame("ComboBox Example");
   String country[]={"India","Aus","U.S.A","England","Newzealand"};
   JComboBox cb=new JComboBox(country);
   cb.setBounds(50, 50,90,20);
   f.add(cb);
   f.setLayout(null);
   f.setSize(400,500);
   f.setVisible(true);
}
public static void main(String[] args) {
   new ComboBoxExample();
}   }
```

Output:

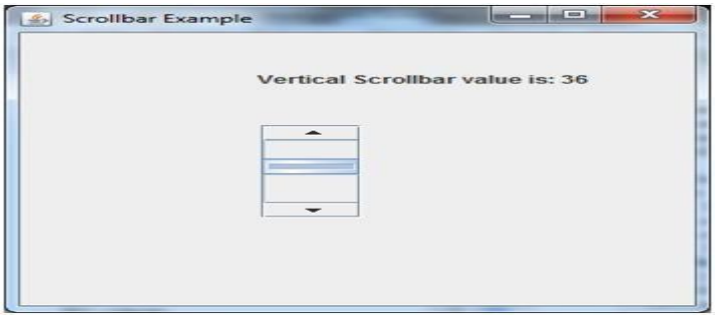


Scrollbars

The object of JScrollbar class is used to add horizontal and vertical scrollbar. It is an implementation of a scrollbar. It inherits JComponent class.

```
mport javax.swing.*;
class ScrollBarExample
{
ScrollBarExample(){
   JFrame f= new JFrame("Scrollbar Example");
JScrollBar s=new JScrollBar();
s.setBounds(100,100, 50,100);
f.add(s);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[])
{
new ScrollBarExample();
}}
```

Output:



Windows

The class JWindow is a container that can be displayed but does not have the title bar

Menus

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

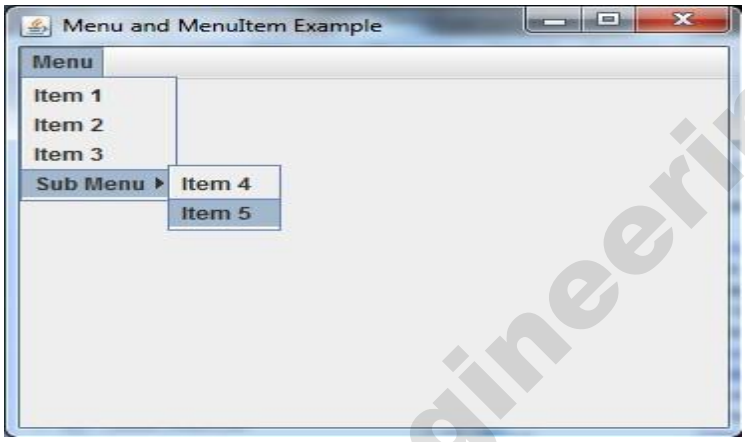
The object of JMenu class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class.

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.

```
import javax.swing.*;
class MenuExample
{
    JMenu menu, submenu;
    JMenuItem i1, i2, i3, i4, i5;
    MenuExample(){
    JFrame f= new JFrame("Menu and MenuItem Example");
    JMenuBar mb=new JMenuBar();
    menu=new JMenu("Menu");
    submenu=new JMenu("Sub Menu");
    i1=new JMenuItem("Item 1");
    i2=new JMenuItem("Item 2");
    i3=new JMenuItem("Item 3");
    i4=new JMenuItem("Item 4");
    i5=new JMenuItem("Item 5");
    menu.add(i1); menu.add(i2); menu.add(i3);
    submenu.add(i4); submenu.add(i5);
```

```
        menu.add(submenu);
        mb.add(menu);
        f.setJMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
    new MenuExample();
    }}
```

Output :



Dialog Boxes.

The JDialog control represents a top level window with a border and a title used to take some form of input from the user. It inherits the Dialog class.Unlike JFrame, it doesn't have maximize and minimize buttons.

Example:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class DialogExample {
   private static JDialog d;
   DialogExample() {
      JFrame f= new JFrame();
```

```
    d = new JDialog(f , "Dialog Example", true);
    d.setLayout( new FlowLayout() );
    JButton b = new JButton ("OK");
    b.addActionListener ( new ActionListener()
    {
       public void actionPerformed( ActionEvent e )
       {
          DialogExample.d.setVisible(false); }}};
    d.add( new JLabel ("Click button to continue."));
    d.add(b);
    d.setSize(300,300);
    d.setVisible(true);
  }
  public static void main(String args[])
  {
    new DialogExample(); }}
```

Output: