UNIT-V

GUI programming

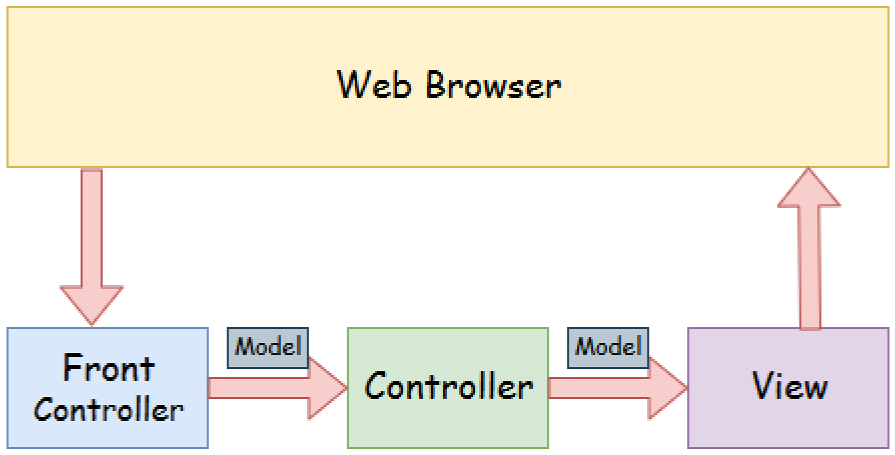
A **graphical** user interface (**GUI**) allows a user to interact with a computer **program** using a pointing device that manipulates small pictures on a computer screen. This style of **programming** is called "event driven **programming**." In fact, by definition, all **GUI** programs are event-driven programs.
Limitations of AWT: The AWT defines a basic set of controls, windows, and dialog boxes that support a usable, but limited graphical interface. One reason for the limited nature of the AWT is that it translates its various visual components into their corresponding, platform-specific equivalents or peers. This means that the look and feel of a component is defined by the platform, not by java. Because the AWT components use native code resources, they are referred to as heavy weight. The use of native peers led to several problems. First, because of variations between operating systems, a component might look, or even act, differently on different platforms. This variability threatened java's philosophy: write once, run anywhere. Second, the look and feel of each component was fixed and could not be changed. Third, the use of heavyweight components caused some frustrating restrictions. Due to these limitations Swing came and was integrated to java. Swing is built on the AWT. Two key Swing features are: Swing components are light weight, Swing supports a pluggable look and feel.

Swing MVC Architecture

A Swing MVC is a Java framework which is used to build web applications. It follows the Model-View-Controller design pattern. It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection.

A Swing MVC provides an elegant solution to use MVC in swing framework by the help of **DispatcherServlet**. Here, **DispatcherServlet** is a class that receives the incoming request and maps it to the right resource such as controllers, models, and views.

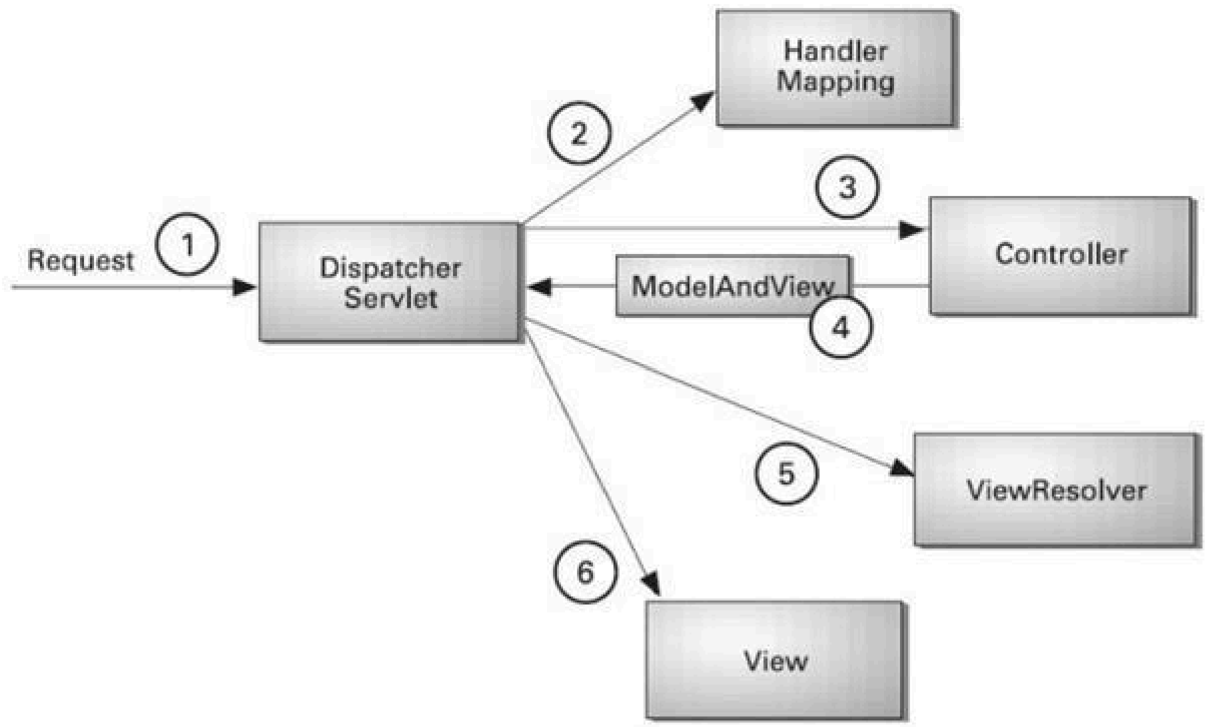
Swing Web Model-View-Controller



- **Model** - A model contains the data of the application. A data can be a single object or a collection of objects.

- **Controller** - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.
- **View** - A view represents the provided information in a particular format. Generally, JSP+JSTL is used to create a view page. Although spring also supports other view technologies such as Apache Velocity, Thymeleaf and FreeMarker.
- **Front Controller** - In Spring Web MVC, the DispatcherServlet class works as the front controller. It is responsible to manage the flow of the Spring MVC application.

Understanding the flow of Swing Web MVC



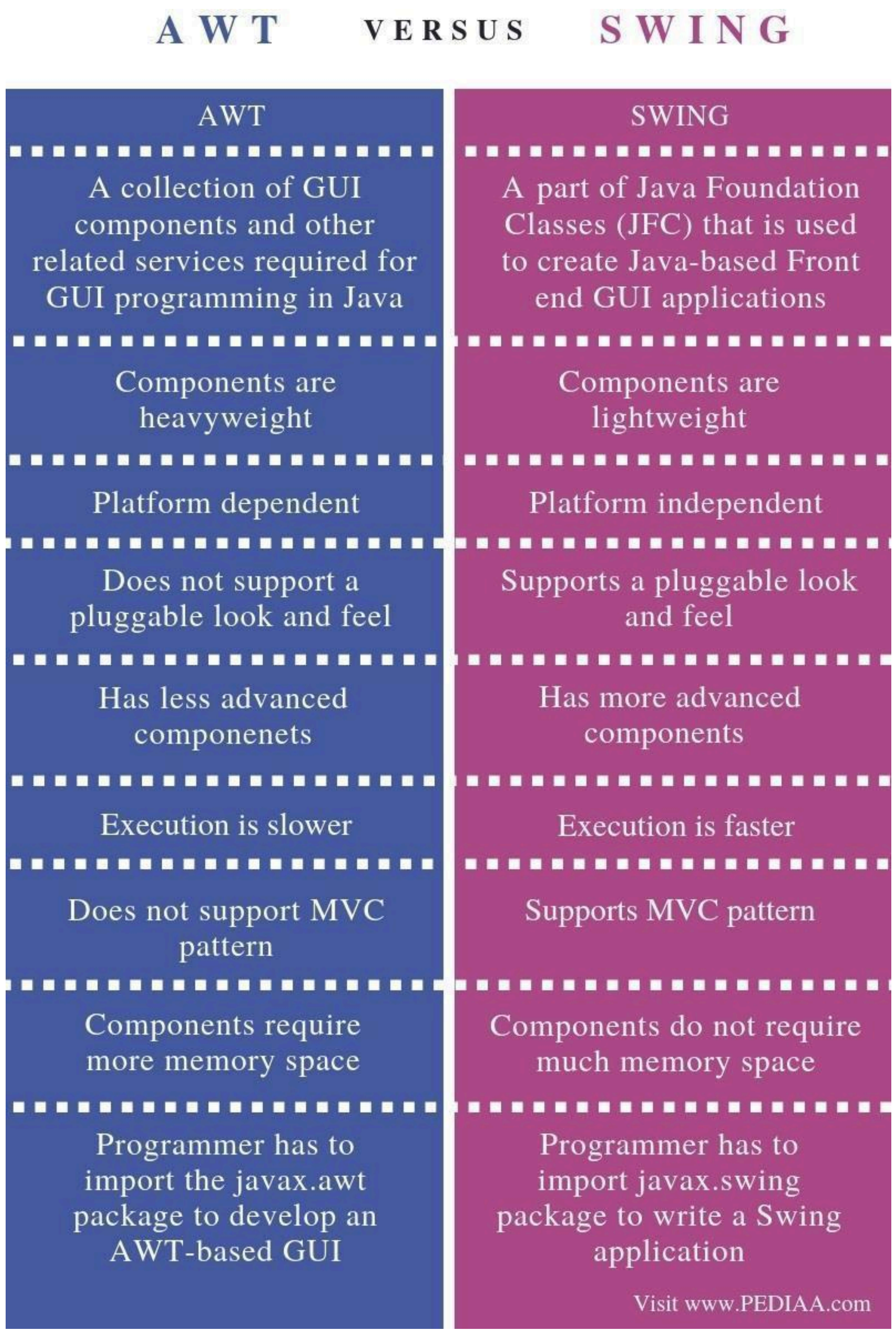
- As displayed in the figure, all the incoming request is intercepted by the DispatcherServlet that works as the front controller.
- The DispatcherServlet gets an entry of handler mapping from the XML file and forwards the request to the controller.
- The controller returns an object of ModelAndView.
- The DispatcherServlet checks the entry of view resolver in the XML file and invokes the specified view component.

Advantages of Spring MVC Framework

Let's see some of the advantages of Spring MVC Framework:-

- **Separate roles** - The Spring MVC separates each role, where the model object, controller, command object, view resolver, DispatcherServlet, validator, etc. can be fulfilled by a specialized object.
- **Light-weight** - It uses light-weight servlet container to develop and deploy your application.
- **Powerful Configuration** - It provides a robust configuration for both framework and application classes that includes easy referencing across contexts, such as from web controllers to business objects and validators.
- **Rapid development** - The Spring MVC facilitates fast and parallel development.
- **Reusable business code** - Instead of creating new objects, it allows us to use the existing business objects.
- **Easy to test** - In Spring, generally we create JavaBeans classes that enable you to inject test data using the setter methods.
- **Flexible Mapping** - It provides the specific annotations that easily redirect the page.

Difference between awt and swing



AWT VERSUS SWING

AWT	SWING
A collection of GUI components and other related services required for GUI programming in Java	A part of Java Foundation Classes (JFC) that is used to create Java-based Front end GUI applications
Components are heavyweight	Components are lightweight
Platform dependent	Platform independent
Does not support a pluggable look and feel	Supports a pluggable look and feel
Has less advanced componenets	Has more advanced components
Execution is slower	Execution is faster
Does not support MVC pattern	Supports MVC pattern
Components require more memory space	Components do not require much memory space
Programmer has to import the javax.awt package to develop an AWT-based GUI	Programmer has to import javax.swing package to write a Swing application

Visit www.PEDIAA.com

<u>Componenets and Containers</u>

The Component class is found under java.awt package. The container class is the subclass of Component class. All non-menu-related elements that comprise a graphical user interface are derived from the abstract class Component. The Component class defines a number of of methods for handling events, changing window bounds, controlling fonts and colors, and drawing components and their content.

A container is a component that can accommodate other components and also other containers. Containers provide the support for building complex hierarchical graphical user interface. Container provides the overloaded method add () to include components in the container.

LAYOUTManagers

There are following 5 classes that represents the layout managers:

- **java**. awt. BorderLayout.
- **java**. awt. FlowLayout.
- **java**. awt. GridLayout.
- **java**. awt. CardLayout.
- **java**. awt. GridBagLayout.

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:

1)BorderLayout

The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**
4. **public static final int WEST**
5. **public static final int CENTER**

Constructors of BorderLayout class:

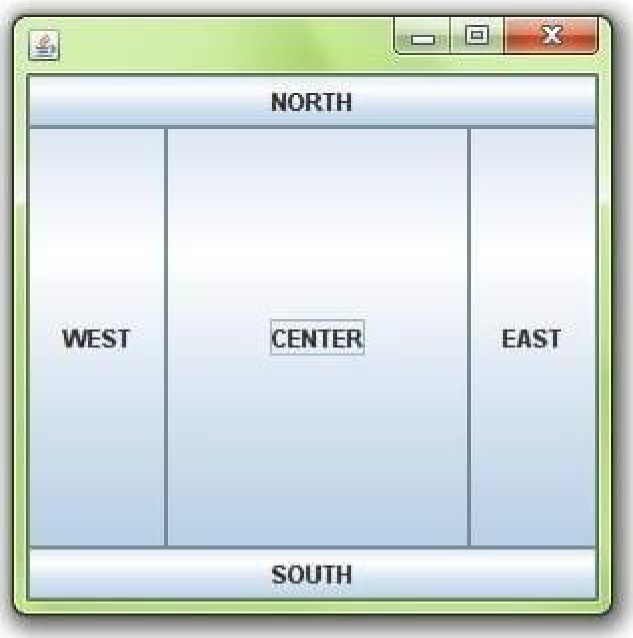
- **BorderLayout():** creates a border layout but with no gaps between the components.
- **JBorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.
- import java.awt.*;
- import javax.swing.*;
-
- public class Border {
- JFrame f;
- 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();
●   }
●   }
```

Here in above example we are using frame class to compile java programs

```
javac Border.java
java Border
```

output:



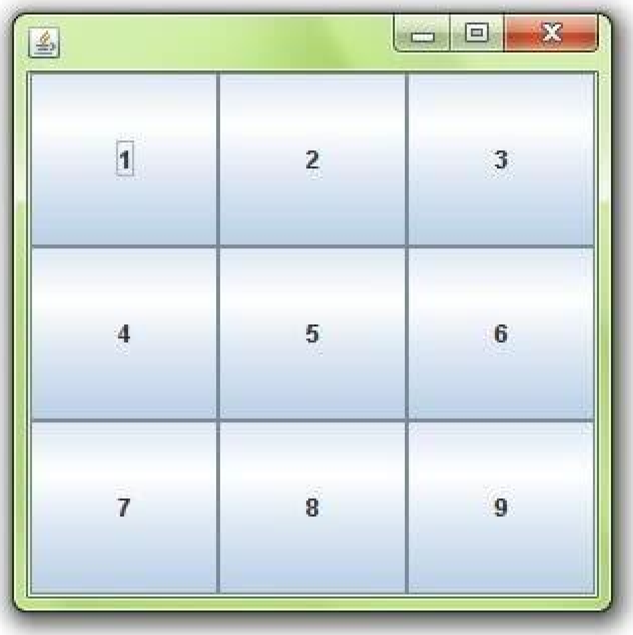
2)GridLayout:

The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.

Constructors of GridLayout class

1. **GridLayout():** creates a grid layout with one column per component in a row.
2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
3. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.

```
import java.awt.*;
import javax.swing.*;
public class MyGridLayout{
JFrame f;
MyGridLayout(){
    f=new JFrame();
      JButton b1=new JButton("1");
    JButton b2=new JButton("2");
    JButton b3=new JButton("3");
    JButton b4=new JButton("4");
    JButton b5=new JButton("5");
        JButton b6=new JButton("6");
        JButton b7=new JButton("7");
    JButton b8=new JButton("8");
        JButton b9=new JButton("9");
     f.add(b1);
f.add(b2);
f.add(b3);
f.add(b4);
f.add(b5);
    f.add(b6);
f.add(b7);
f.add(b8);
f.add(b9);
 f.setLayout(new GridLayout(3,3));
    //setting grid layout of 3 rows and 3 columns
    f.setSize(300,300);
    f.setVisible(true);
}
public static void main(String[] args) {
    new MyGridLayout();
}
}
```



3) FlowLayout:

Java FlowLayout

The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel.

Fields of FlowLayout class

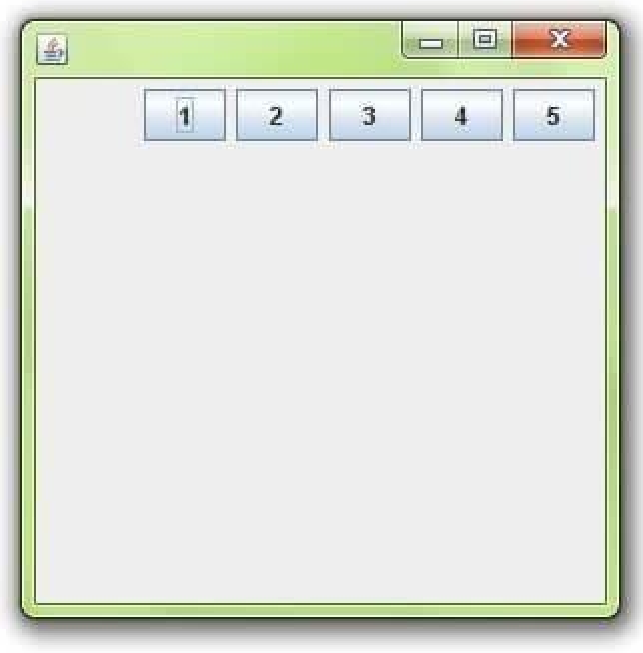
1. **public static final int LEFT**
2. **public static final int RIGHT**
3. **public static final int CENTER**
4. **public static final int LEADING**
5. **public static final int TRAILING**

Constructors of FlowLayout class

1. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
2. **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
3. **FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap.

Example of FlowLayout class

```java
import java.awt.*;
import javax.swing.*;

public class MyFlowLayout{
JFrame f;
MyFlowLayout(){
    f=new JFrame();

    JButton b1=new JButton("1");
    JButton b2=new JButton("2");
    JButton b3=new JButton("3");
    JButton b4=new JButton("4");
     JButton b5=new JButton("5");

    f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);

    f.setLayout(new FlowLayout(FlowLayout.RIGHT));
     //setting flow layout of right alignment

    f.setSize(300,300);
    f.setVisible(true);
}
public static void main(String[] args) {
    new MyFlowLayout();
}
}
```



4) Java CardLayout

The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

Constructors of CardLayout class

1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.
2. **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

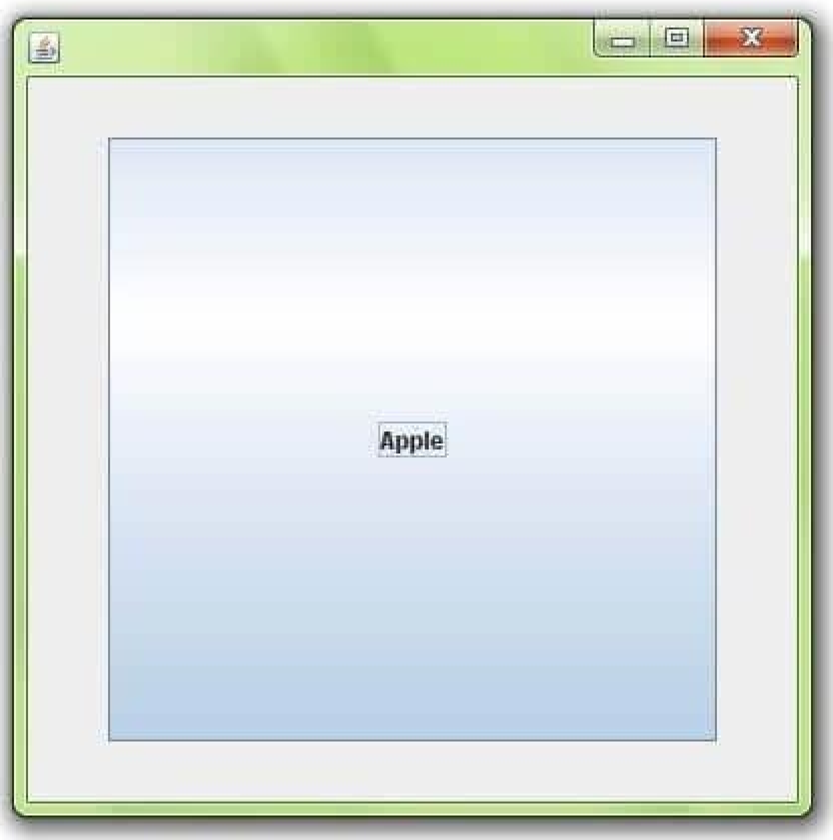
Commonly used methods of CardLayout class

- **public void next(Container parent):** is used to flip to the next card of the given container.
- **public void previous(Container parent):** is used to flip to the previous card of the given container.
- **public void first(Container parent):** is used to flip to the first card of the given container.
- **public void last(Container parent):** is used to flip to the last card of the given container.
- **public void show(Container parent, String name):** is used to flip to the specified card with the given name.

Example of CardLayout class

1. import java.awt.*;

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

import javax.swing.*;

public class CardLayoutExample extends JFrame implements ActionListener{
CardLayout card;
JButton b1,b2,b3;
Container c;
    CardLayoutExample(){

        c=getContentPane();
        card=new CardLayout(40,30);
//create CardLayout object with 40 hor space and 30 ver space
        c.setLayout(card);

        b1=new JButton("Apple");
        b2=new JButton("Boy");
        b3=new JButton("Cat");
        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);

        c.add("a",b1);c.add("b",b2);c.add("c",b3);

    }
    public void actionPerformed(ActionEvent e) {
    card.next(c);
    }

    public static void main(String[] args) {
        CardLayoutExample cl=new CardLayoutExample();
        cl.setSize(400,400);
        cl.setVisible(true);
        cl.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

```

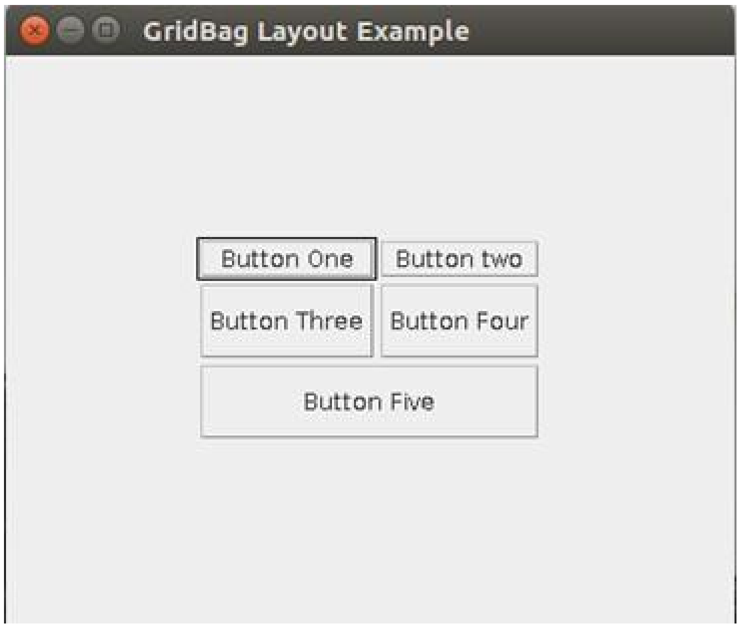


Java GridBagLayout

The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.

The components may not be of same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells. Each component occupies one or more cells known as its display area. Each component associates an instance of GridBagConstraints. With the help of constraints object we arrange component's display area on the grid. The GridBagLayout manages each component's minimum and preferred sizes in order to determine component's size.

```
import java.awt.Button;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;

import javax.swing.*;
public class GridBagLayoutExample extends JFrame{
    public static void main(String[] args) {
            GridBagLayoutExample a = new GridBagLayoutExample();
        }
    public GridBagLayoutExample() {
GridBagLayoutgrid = new GridBagLayout();
        GridBagConstraints gbc = new GridBagConstraints();
        setLayout(grid);
        setTitle("GridBag Layout Example");
        GridBagLayout layout = new GridBagLayout();
this.setLayout(layout);
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.gridx = 0;
gbc.gridy = 0;
this.add(new Button("Button One"), gbc);
gbc.gridx = 1;
gbc.gridy = 0;
this.add(new Button("Button two"), gbc);
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.ipady = 20;
gbc.gridx = 0;
gbc.gridy = 1;
this.add(new Button("Button Three"), gbc);
gbc.gridx = 1;
gbc.gridy = 1;
this.add(new Button("Button Four"), gbc);
gbc.gridx = 0;
gbc.gridy = 2;
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.gridwidth = 2;
this.add(new Button("Button Five"), gbc);
        setSize(300, 300);
        setPreferredSize(getSize());
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);

    }

}
```



Event and Listener (Java 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 Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Steps to perform Event Handling

Following steps are required to perform event handling:

1. Register the component with the Listener

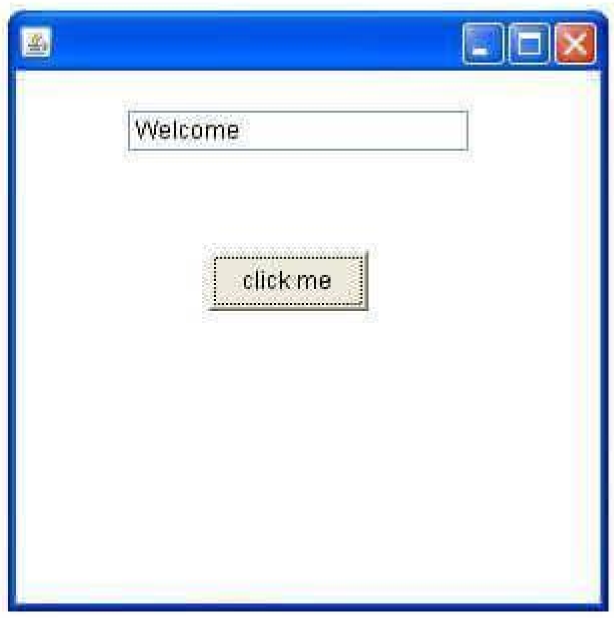
Registration Methods

For registering the component with the Listener, many classes provide the registration methods. For example:

- **Button**
  - public void addActionListener(ActionListener a){}
- **MenuItem**
  - public void addActionListener(ActionListener a){}
- **TextField**
  - public void addActionListener(ActionListener a){}
  - public void addTextListener(TextListener a){}
- **TextArea**
  - public void addTextListener(TextListener a){}
- **Checkbox**
  - public void addItemListener(ItemListener a){}
- **Choice**
  - public void addItemListener(ItemListener a){}
- **List**
  - public void addActionListener(ActionListener a){}
  - public void addItemListener(ItemListener a){}

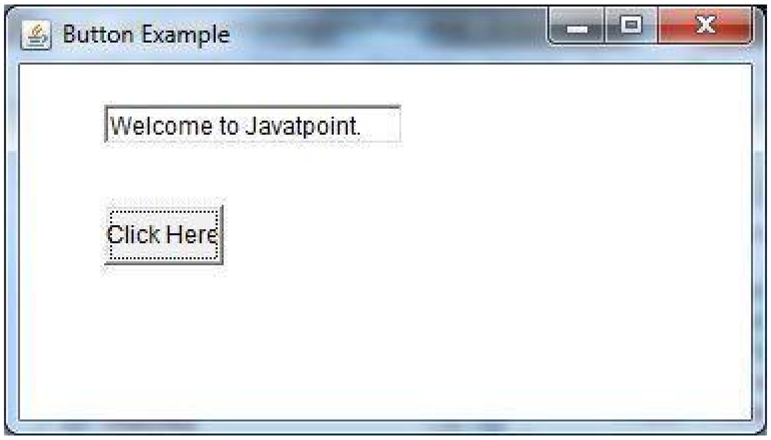
```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame implements ActionListener{
TextField tf;
AEvent(){

//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);

//register listener
b.addActionListener(this);//passing current instance

//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}
public static void main(String args[]){
new AEvent();
}
}
```

public void setBounds(int xaxis, int yaxis, int width, int height); have been used in the above example that sets the position of the component it may be button, textfield etc.



ava AWT Button Example with ActionListener

```
import java.awt.*;
import java.awt.event.*;
public class ButtonExample {
public static void main(String[] args) {
    Frame f=new Frame("Button Example");
    final TextField tf=new TextField();
    tf.setBounds(50,50, 150,20);

  Button b=new Button("Click Here");
    b.setBounds(50,100,60,30);
    b.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
            tf.setText("Welcome to Javatpoint.");
        }
    });
    f.add(b);f.add(tf);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

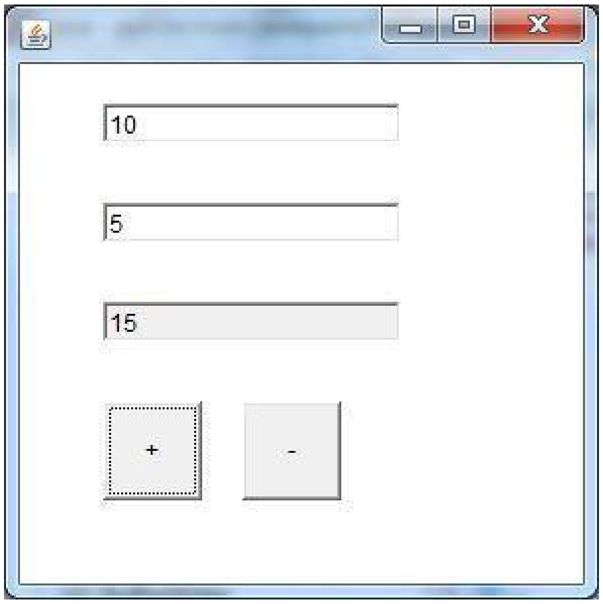


AWT TextField Example with ActionListener

```
import java.awt.*;
import java.awt.event.*;
public class TextFieldExample extends Frame implements ActionListener{
    TextField tf1,tf2,tf3;
    Button b1,b2;
    TextFieldExample(){
        tf1=new TextField();
        tf1.setBounds(50,50,150,20);
        tf2=new TextField();
        tf2.setBounds(50,100,150,20);
        tf3=new TextField();
        tf3.setBounds(50,150,150,20);
        tf3.setEditable(false);
        b1=new Button("+");
```

```java
        b1.setBounds(50,200,50,50);
        b2=new Button("-");
        b2.setBounds(120,200,50,50);
        b1.addActionListener(this);
        b2.addActionListener(this);
        add(tf1);add(tf2);add(tf3);add(b1);add(b2);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e) {
        String s1=tf1.getText();
        String s2=tf2.getText();
        int a=Integer.parseInt(s1);
        int b=Integer.parseInt(s2);
        int c=0;
        if(e.getSource()==b1){
            c=a+b;
        }else if(e.getSource()==b2){
            c=a-b;
        }
        String result=String.valueOf(c);
        tf3.setText(result);
    }
public static void main(String[] args) {
    new TextFieldExample();
}
}
```

Output:



10

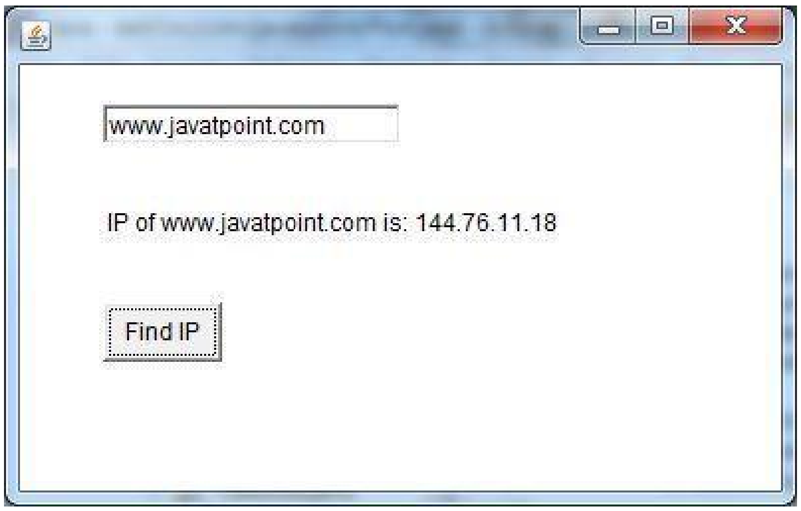
5

15

Java AWT Label Example with ActionListener

```
import java.awt.*;
import java.awt.event.*;
public class LabelExample extends Frame implements ActionListener{
    TextField tf; Label l; Button b;
    LabelExample(){
        tf=new TextField();
        tf.setBounds(50,50, 150,20);
        l=new Label();
        l.setBounds(50,100, 250,20);
        b=new Button("Find IP");
        b.setBounds(50,150,60,30);
        b.addActionListener(this);
        add(b);add(tf);add(l);
        setSize(400,400);
        setLayout(null);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e) {
        try{
        String host=tf.getText();
        String ip=java.net.InetAddress.getByName(host).getHostAddress();
        l.setText("IP of "+host+" is: "+ip);
        }catch(Exception ex){System.out.println(ex);}
    }
    public static void main(String[] args) {
        new LabelExample();
    }
}
```

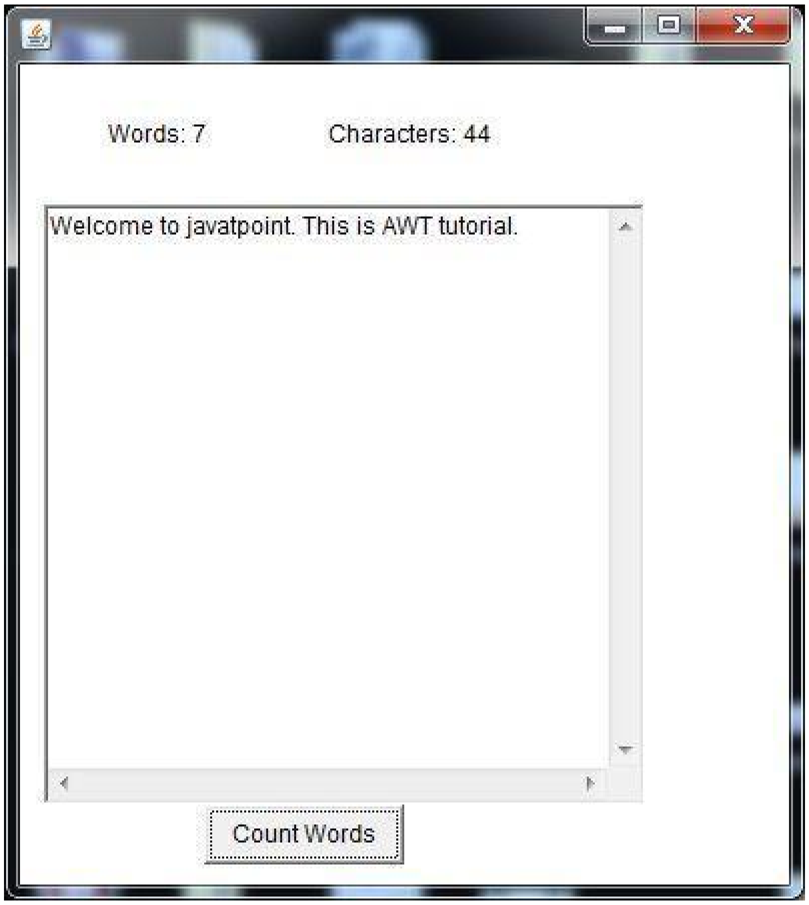
Output:



```
www.javatpoint.com

IP of www.javatpoint.com is: 144.76.11.18
```

Java AWT TextArea Example with ActionListener

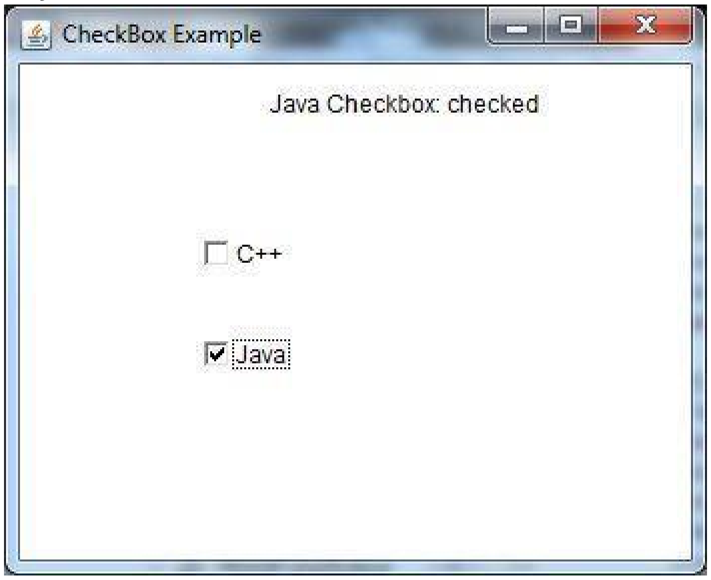
```
import java.awt.*;
import java.awt.event.*;
public class TextAreaExample extends Frame implements ActionListener{
Label l1,l2;
TextArea area;
Button b;
TextAreaExample(){
    l1=new Label();
    l1.setBounds(50,50,100,30);
    l2=new Label();
    l2.setBounds(160,50,100,30);
    area=new TextArea();
    area.setBounds(20,100,300,300);
    b=new Button("Count Words");
    b.setBounds(100,400,100,30);
    b.addActionListener(this);
    add(l1);add(l2);add(area);add(b);
    setSize(400,450);
    setLayout(null);
    setVisible(true);
}
public void actionPerformed(ActionEvent e){
    String text=area.getText();
    String words[]=text.split("\\s");
    l1.setText("Words: "+words.length);
    l2.setText("Characters: "+text.length());
}
public static void main(String[] args) {
    new TextAreaExample();
}
}
```



```
Words: 7        Characters: 44

Welcome to javatpoint. This is AWT tutorial.
```

Example on Checkbox

```java
import java.awt.*;
import java.awt.event.*;
public class CheckboxExample
{
     CheckboxExample(){
        Frame f= new Frame("CheckBox Example");
        final Label label = new Label();
        label.setAlignment(Label.CENTER);
        label.setSize(400,100);
        Checkbox checkbox1 = new Checkbox("C++");
        checkbox1.setBounds(100,100, 50,50);
        Checkbox checkbox2 = new Checkbox("Java");
        checkbox2.setBounds(100,150, 50,50);
        f.add(checkbox1); f.add(checkbox2); f.add(label);
        checkbox1.addItemListener(new ItemListener() {
             public void itemStateChanged(ItemEvent e) {
                label.setText("C++ Checkbox: "
                + (e.getStateChange()==1?"checked":"unchecked"));
             }
          });
        checkbox2.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
               label.setText("Java Checkbox: "
               + (e.getStateChange()==1?"checked":"unchecked"));
            }
          });
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
     }
public static void main(String args[])
{
    new CheckboxExample();
}
}
```



Java AWT CheckboxGroup Example with ItemListener

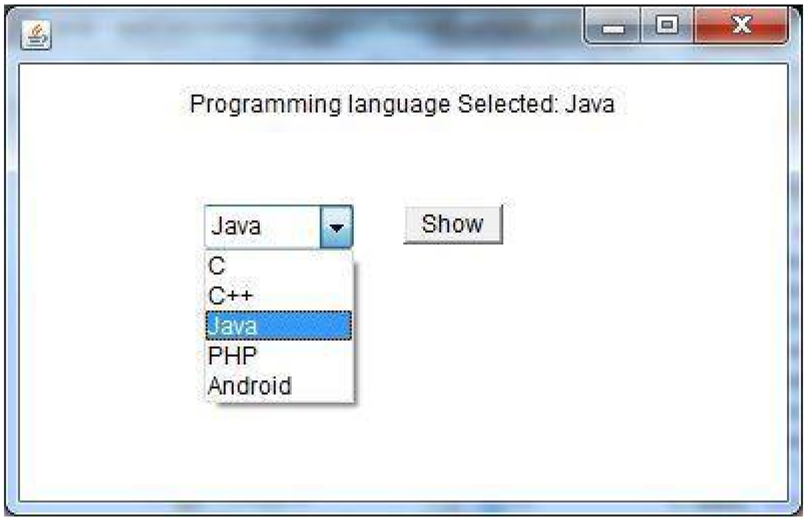
```java
import java.awt.*;
import java.awt.event.*;
public class CheckboxGroupExample
{
     CheckboxGroupExample(){
       Frame f= new Frame("CheckboxGroup Example");
       final Label label = new Label();
       label.setAlignment(Label.CENTER);
       label.setSize(400,100);
        CheckboxGroup cbg = new CheckboxGroup();
        Checkbox checkBox1 = new Checkbox("C++", cbg, false);
        checkBox1.setBounds(100,100, 50,50);
        Checkbox checkBox2 = new Checkbox("Java", cbg, false);
        checkBox2.setBounds(100,150, 50,50);
        f.add(checkBox1); f.add(checkBox2); f.add(label);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
        checkBox1.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
               label.setText("C++ checkbox: Checked");
            }
         });
        checkBox2.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
               label.setText("Java checkbox: Checked");
            }
         });
     }
public static void main(String args[])
{
    new CheckboxGroupExample();
}
}
```

```
mport java.awt.*;
import java.awt.event.*;
public class CheckboxGroupExample
{
     CheckboxGroupExample(){
       Frame f= new Frame("CheckboxGroup Example");
       final Label label = new Label();
       label.setAlignment(Label.CENTER);
       label.setSize(400,100);
        CheckboxGroup cbg = new CheckboxGroup();
        Checkbox checkBox1 = new Checkbox("C++", cbg, false);
        checkBox1.setBounds(100,100, 50,50);
        Checkbox checkBox2 = new Checkbox("Java", cbg, false);
        checkBox2.setBounds(100,150, 50,50);
        f.add(checkBox1); f.add(checkBox2); f.add(label);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
        checkBox1.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
               label.setText("C++ checkbox: Checked");
            }
         });
        checkBox2.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
               label.setText("Java checkbox: Checked");
            }
         });
```

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

Java AWT Choice Example with ActionListener

```java
import java.awt.*;
import java.awt.event.*;
public class ChoiceExample
{
        ChoiceExample(){
        Frame f= new Frame();
        final Label label = new Label();
        label.setAlignment(Label.CENTER);
        label.setSize(400,100);
        Button b=new Button("Show");
        b.setBounds(200,100,50,20);
        final Choice c=new Choice();
        c.setBounds(100,100, 75,75);
        c.add("C");
        c.add("C++");
        c.add("Java");
        c.add("PHP");
        c.add("Android");
        f.add(c);f.add(label); f.add(b);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
         String data = "Programming language Selected: "+ c.getItem(c.getSelectedIndex());
         label.setText(data);
        }
        });
        }
public static void main(String args[])
{
   new ChoiceExample();
}
}
```


Programming language Selected: Java

Java AWT List Example with ActionListener

```
import java.awt.*;
import java.awt.event.*;
public class ListExample
{
     ListExample(){
        Frame f= new Frame();
        final Label label = new Label();
        label.setAlignment(Label.CENTER);
        label.setSize(500,100);
        Button b=new Button("Show");
        b.setBounds(200,150,80,30);
        final List l1=new List(4, false);
        l1.setBounds(100,100, 70,70);
        l1.add("C");
        l1.add("C++");
        l1.add("Java");
        l1.add("PHP");
        final List l2=new List(4, true);
        l2.setBounds(100,200, 70,70);
        l2.add("Turbo C++");
        l2.add("Spring");
        l2.add("Hibernate");
        l2.add("CodeIgniter");
        f.add(l1); f.add(l2); f.add(label); f.add(b);
        f.setSize(450,450);
        f.setLayout(null);
        f.setVisible(true);
        b.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
          String data = "Programming language Selected: "+l1.getItem(l1.getSelectedIndex());
          data += ", Framework Selected:";
```

```java
            for(String frame:l2.getSelectedItems()){
                     data += frame + " ";
            }
            label.setText(data);
            }
           });
}
public static void main(String args[])
{
   new ListExample();
}
}
```

Output:

Java AWT Scrollbar Example with AdjustmentListener

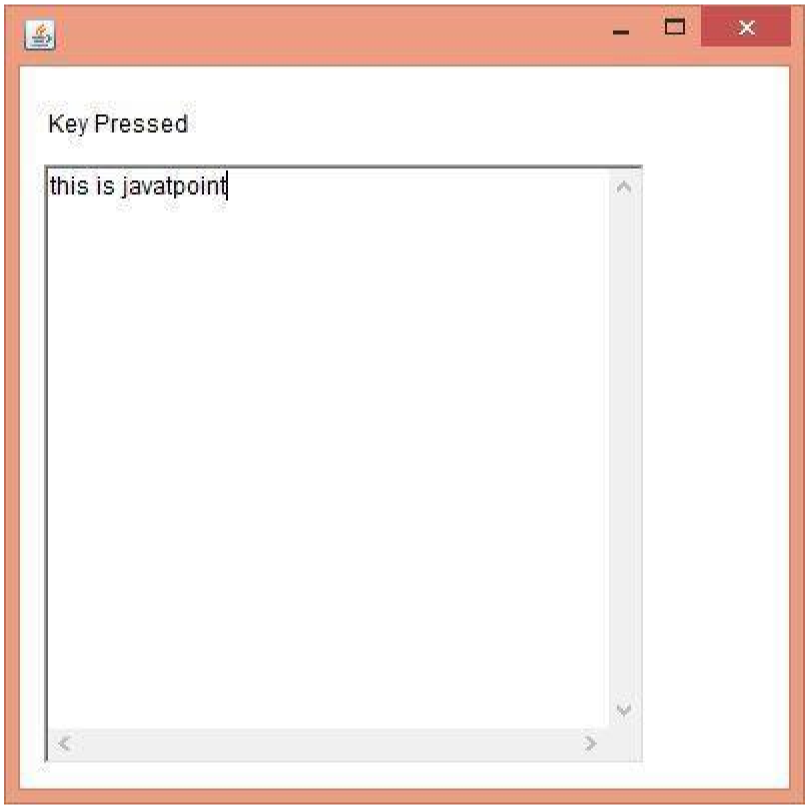
```java
import java.awt.*;
import java.awt.event.*;
class ScrollbarExample{
     ScrollbarExample(){
            Frame f= new Frame("Scrollbar Example");
            final Label label = new Label();
            label.setAlignment(Label.CENTER);
            label.setSize(400,100);
            final Scrollbar s=new Scrollbar();
            s.setBounds(100,100, 50,100);
            f.add(s);f.add(label);
            f.setSize(400,400);
            f.setLayout(null);
            f.setVisible(true);
            s.addAdjustmentListener(new AdjustmentListener() {
                public void adjustmentValueChanged(AdjustmentEvent e) {
                   label.setText("Vertical Scrollbar value is:"+ s.getValue());
                }
            });
         }
public static void main(String args[]){
new ScrollbarExample();
}
}
```

Output:

Java KeyListener Example

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

```java
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();
    }
}
```


Key Pressed

this is javatpoint

Java MouseListener Interface

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.

Methods of MouseListener interface

The signature of 5 methods found in MouseListener interface are given below:

```java
public abstract void mouseClicked(MouseEvent e);
public abstract void mouseEntered(MouseEvent e);
public abstract void mouseExited(MouseEvent e);
public abstract void mousePressed(MouseEvent e);
public abstract void mouseReleased(MouseEvent e);
```

Java MouseListener Example

```java
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();
}
}
```

Java Inner Classes

Java inner class or nested class is a class which is declared inside the class or interface.

We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.

Additionally, it can access all the members of outer class including private data members and methods.

Syntax of Inner class

```
class Java_Outer_class{
 //code
 class Java_Inner_class{
```

```
 //code
 }
}
```

Advantage of java inner classes

There are basically three advantages of inner classes in java. They are as follows:

1) Nested classes represent a special type of relationship that is **it can access all the members (data members and methods) of outer class** including private.

2) Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.

3) **Code Optimization**: It requires less code to write.

Difference between nested class and inner class in Java

Inner class is a part of nested class. Non-static nested classes are known as inner classes.

Types of Nested classes

There are two types of nested classes non-static and static nested classes.The non-static nested classes are also known as inner classes.

- Non-static nested class (inner class)
    1. Member inner class
    2. Anonymous inner class
    3. Local inner class
- Static nested class

Type	**Description**
Member Inner Class	A class created within class and outside method.
Anonymous Inner Class	A class created for implementing interface or extending class. Its name is decided by the java compiler.
Local Inner Class	A class created within method.
Static Nested Class	A static class created within class.
Nested Interface	An interface created within class or interfac

Java Member inner class

A non-static class that is created inside a class but outside a method is called member inner class.

Syntax:

```
class Outer{
 //code
 class Inner{
  //code
 }
}
```

Java Member inner class example

In this example, we are creating msg() method in member inner class that is accessing the private data member of outer class.

```
class TestMemberOuter1{
 private int data=30;
 class Inner{
  void msg(){System.out.println("data is "+data);}
 }
 public static void main(String args[]){
  TestMemberOuter1 obj=new TestMemberOuter1();
  TestMemberOuter1.Inner in=obj.new Inner();
  in.msg();
 }
}
```

Internal working of Java member inner class

The java compiler creates two class files in case of inner class. The class file name of inner class is "Outer$Inner". If you want to instantiate inner class, you must have to create the instance of outer class. In such case, instance of inner class is created inside the instance of outer class.

Internal code generated by the compiler

The java compiler creates a class file named Outer$Inner in this case. The Member inner class have the reference of Outer class that is why it can access all the data members of Outer class including private.

```
import java.io.PrintStream;
class Outer$Inner
{
    final Outer this$0;
    Outer$Inner()
    {  super();
        this$0 = Outer.this;
    }
    void msg()
    {
        System.out.println((new StringBuilder()).append("data is ")
                    .append(Outer.access$000(Outer.this)).toString());
    }
}

```

Java Anonymous inner class

A class that have no name is known as anonymous inner class in java. It should be used if you have to override method of class or interface. Java Anonymous inner class can be created by two ways:

1. Class (may be abstract or concrete).
2. Interface

Java anonymous inner class example using class

```
abstract class Person{
  abstract void eat();
}
class TestAnonymousInner{
 public static void main(String args[]){
  Person p=new Person(){
  void eat(){System.out.println("nice fruits");}
  };
  p.eat();
 }
}
```

Test it Now

Output:

```
nice fruits
```

Internal working of given code

```
Person p=new Person(){
```

```
void eat(){System.out.println("nice fruits");}
};
```

1. A class is created but its name is decided by the compiler which extends the Person class and provides the implementation of the eat() method.
2. An object of Anonymous class is created that is referred by p reference variable of Person type.

Internal class generated by the compiler

```
import java.io.PrintStream;
static class TestAnonymousInner$1 extends Person
{
  TestAnonymousInner$1(){}
  void eat()
   {
      System.out.println("nice fruits");
   }
}
```

Java anonymous inner class example using interface

```
interface Eatable{
 void eat();
}
class TestAnnonymousInner1{
 public static void main(String args[]){
 Eatable e=new Eatable(){
  public void eat(){System.out.println("nice fruits");}
 };
 e.eat();
 }
}
```

Output:

```
nice fruits
```

Internal working of given code

It performs two main tasks behind this code:

```
Eatable p=new Eatable(){
void eat(){System.out.println("nice fruits");}
};
```

1. A class is created but its name is decided by the compiler which implements the Eatable interface and provides the implementation of the eat() method.
2. An object of Anonymous class is created that is referred by p reference variable of Eatable type.

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.

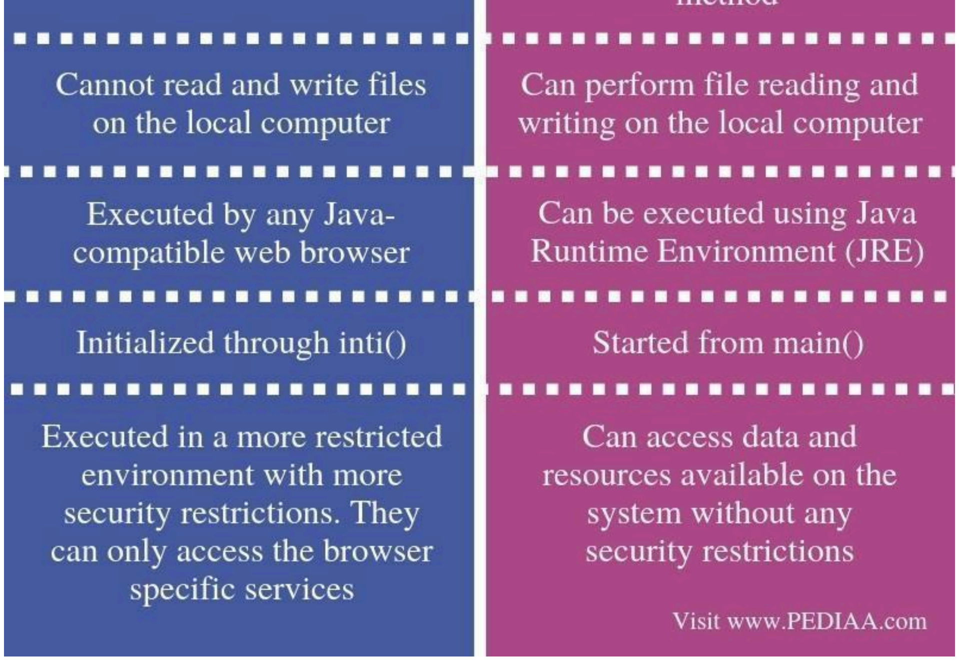
Advantage of Applet

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 plateforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

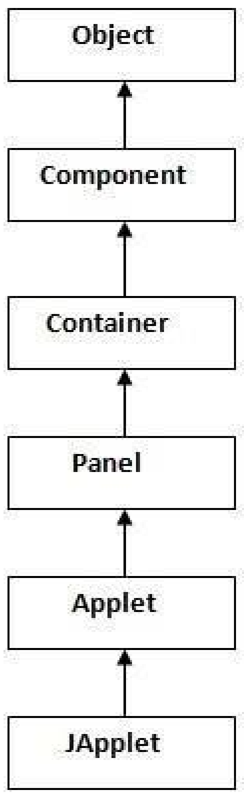
- Plugin is required at client browser to execute applet.
-



Cannot read and write files on the local computer	Can perform file reading and writing on the local computer
Executed by any Java-compatible web browser	Can be executed using Java Runtime Environment (JRE)
Initialized through inti()	Started from main()
Executed in a more restricted environment with more security restrictions. They can only access the browser specific services	Can access data and resources available on the system without any security restrictions

Visit www.PEDIAA.com

-

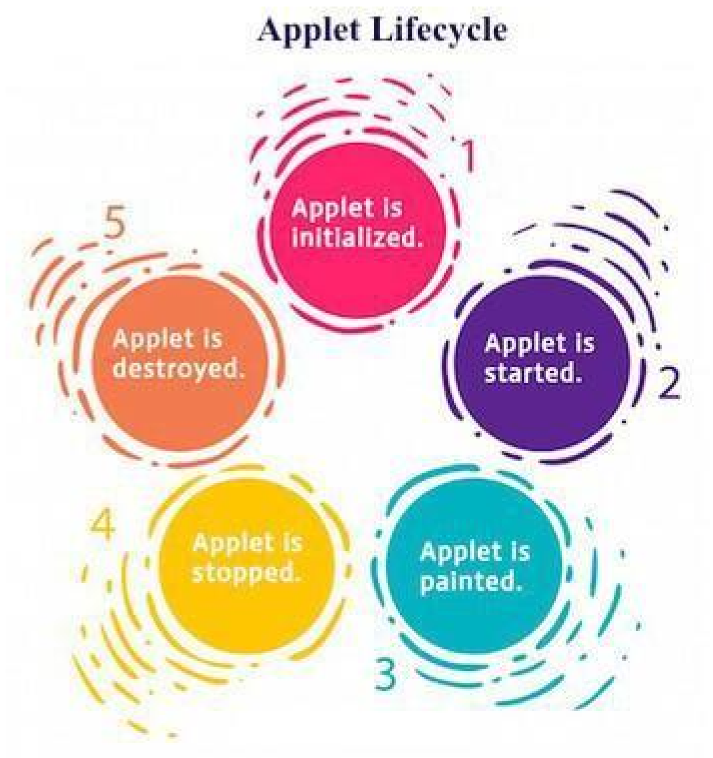


•

As displayed in the above diagram, Applet class extends Panel. Panel class extends Container which is the subclass of Component.

Lifecycle of Java Applet

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



Lifecycle methods for Applet:

The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.

ava.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

1. **public void init():** is used to initialized the Applet. It is invoked only once.
2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. **public void destroy():** is used to destroy the Applet. It is invoked only once.

java.awt.Component class

The Component class provides 1 life cycle method of applet.

1. **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

Who is responsible to manage the life cycle of an applet?

Java Plug-in software.

How to run an Applet?

There are two ways to run an applet

1. By html file.
2. By appletViewer tool (for testing purpose).

Simple example of Applet by html file:

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
    public class First extends Applet{

public void paint(Graphics g){
g.drawString("welcome",150,150);
}

}
```

Note: class must be public because its object is created by Java Plugin software that resides on the browser.

myapplet.html

```
<html>
<body>
<applet code="First.class" width="300" height="300">
</applet>
</body>
</html>
```

Simple example of Applet by appletviewer tool:

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
        public class First extends
Applet{
public void paint(Graphics g){
g.drawString("welcome to applet",150,150);
}

}
/*
<applet code="First.class" width="300" height="300">
</applet>
*/
```

To execute the applet by appletviewer tool, write in command prompt:

```
c:\>javac First.java
c:\>appletviewer First.java
```

Commonly used methods of Graphics class:

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.

10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

Example of Graphics in applet:

```
import java.applet.Applet;
import java.awt.*;

        public class GraphicsDemo extends
Applet{
public void paint(Graphics g){
g.setColor(Color.red);
g.drawString("Welcome",50, 50);
g.drawLine(20,30,20,300);
g.drawRect(70,100,30,30);
g.fillRect(100,100,30,30);
g.drawOval(70,200,30,30);

g.setColor(Color.pink);
g.fillOval(170,200,30,30);
g.drawArc(90,150,30,30,30,270);
g.fillArc(270,150,30,30,0,180);

}
}
```

myapplet.html

```
<html>
<body>
<applet code="GraphicsDemo.class" width="300" height="300">
</applet>
</body>
</html>
```

EventHandling in Applet

As we perform event handling in AWT or Swing, we can perform it in applet also. Let's see the simple example of event handling in applet that prints a message by click on the button.

Example of EventHandling in applet:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class EventApplet extends Applet implements ActionListener{
Button b;
TextField tf;
public void init(){
tf=new TextField();
tf.setBounds(30,40,150,20);
b=new Button("Click");
b.setBounds(80,150,60,50);
add(b);add(tf);
b.addActionListener(this);
setLayout(null);
} public void actionPerformed(ActionEvent e){
  tf.setText("Welcome");
 }
}
```

Seurity Issues with the Applet

Java applet is run inside a web browser. But an applet is restricted in some areas, until it has been deemed trustworthy by the end user. The security restriction is provided for protecting the user by malicious code, like copy important information from the hard disk or deleting the files. Generally, applets are loaded from the Internet and they are prevented from: the writing and reading the files on client side. Some security issues to applet are following :

- Applets are loaded over the internet and they are prevented to make open network connection to any computer, except for the host, which provided the .class file. Because the html page come from the host or the host specified codebase parameter in the applet tag, with codebase taking precedence.

- They are also prevented from starting other programs on the client. That means any applet, which you visited, cannot start any rogue process on you computer. In UNIX, applets cannot start any exec or fork processes. Applets are not allowed to invoke any program to list the contents of your file system that means it cant invoke **System.exit()** function to terminate you web browser. And they are not allowed to manipulate the threads outside the applets own thread group.

- Applets are loaded over the net. A web browser uses only one class loader that?s established at start up. Then the system class loader can not be overloaded, overridden, extended, replaced. Applet is not allowed to create the reference of their own class loader.

- They cant load the libraries or define the native method calls. But if it can define native method calls then that would give the applet direct access to underlying computer.

Passing Parameters to Applets

Parameters specify extra information that can be passed to an applet from the HTML page. Parameters are specified using the HTML's *param* tag.

Param Tag

The <param> tag is a sub tag of the <applet> tag. The <param> tag contains two attributes: *name* and *value* which are used to specify the name of the parameter and the value of the parameter respectively. For example, the param tags for passing name and age parameters looks as shown below:

```
<param name="name" value="Ramesh" />
<param name="age" value="25″ />
```

Now, these two parameters can be accessed in the applet program using the *getParameter()* method of the *Applet* class.

getParameter() Method

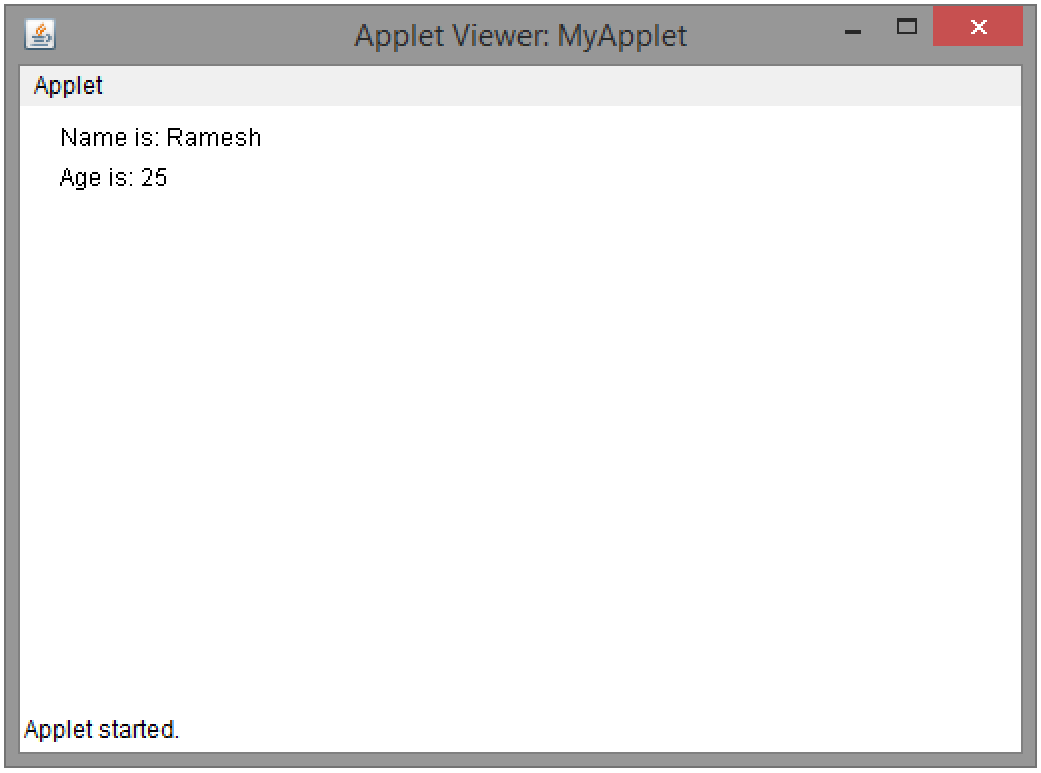
The *getParameter()* method of the *Applet* class can be used to retrieve the parameters passed from the HTML page. The syntax of *getParameter()* method is as follows:

String getParameter(String param-name)

Let’s look at a sample program which demonstrates the <param> HTML tag and the *getParameter()* method:

```
import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet
{
String
n;
String a;
public void init()
{
n = getParameter("name");
a = getParameter("age");
}
public void paint(Graphics g)
{
g.drawString("Name is: " + n, 20, 20);
g.drawString("Age is: " + a, 20, 40);
}
}
/*
<applet code="MyApplet" height="300" width="500">
<param name="name" value="Ramesh" />
<param name="age" value="25" />
</applet>
*/
```

Output of the above program is as follows:



```
Name is: Ramesh
Age is: 25
```

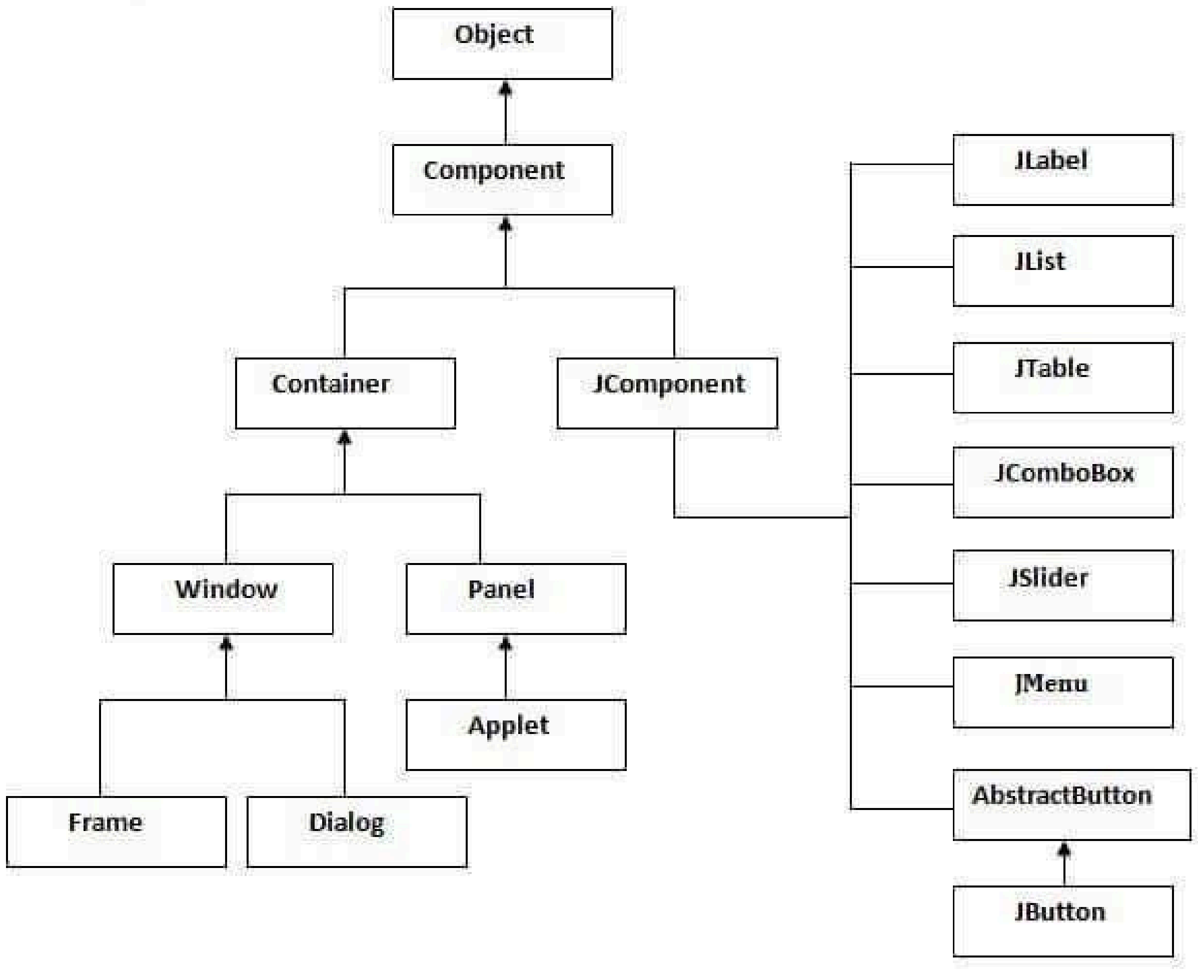
Java Swing tutorial is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

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.

Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.



Swing controls are

1)JLabel

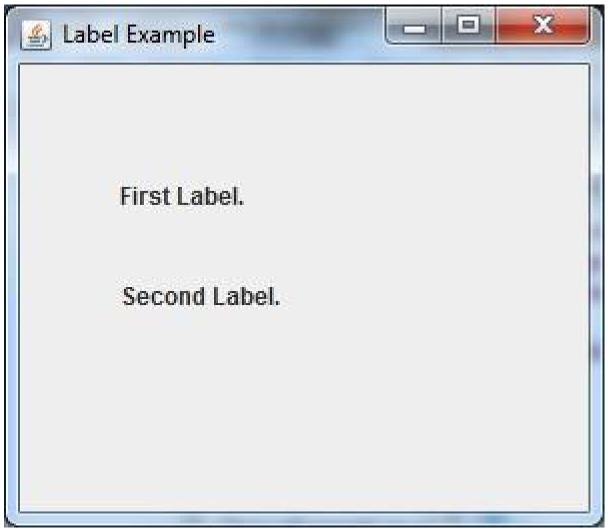
2)ImageIcon

3)JTextField

Sample program on JLabel

```
import javax.swing.*;
class LabelExample
{
public static void main(String args[])
    {
```

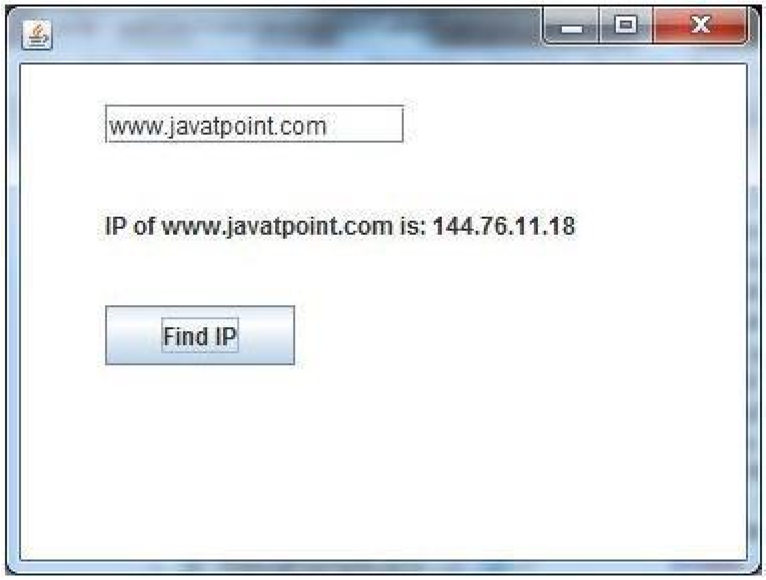
```
JFrame f= new JFrame("Label Example");
JLabel l1,l2;
l1=new JLabel("First Label.");
l1.setBounds(50,50, 100,30);
l2=new JLabel("Second Label.");
l2.setBounds(50,100, 100,30);
f.add(l1); f.add(l2);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
}

```



Java JLabel Example with ActionListener

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class LabelExample extends Frame implements ActionListener{
    JTextField tf; JLabel l; JButton b;
    LabelExample(){
        tf=new JTextField();
        tf.setBounds(50,50, 150,20);
        l=new JLabel();
        l.setBounds(50,100, 250,20);
        b=new JButton("Find IP");
        b.setBounds(50,150,95,30);
        b.addActionListener(this);
        add(b);add(tf);add(l);
        setSize(400,400);
        setLayout(null);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e) {
        try{
        String host=tf.getText();
        String ip=java.net.InetAddress.getByName(host).getHostAddress();
        l.setText("IP of "+host+" is: "+ip);
        }catch(Exception ex){System.out.println(ex);}
    }
    public static void main(String[] args) {
        new LabelExample();
    } }
```

Output:



```
www.javatpoint.com

IP of www.javatpoint.com is: 144.76.11.18
```

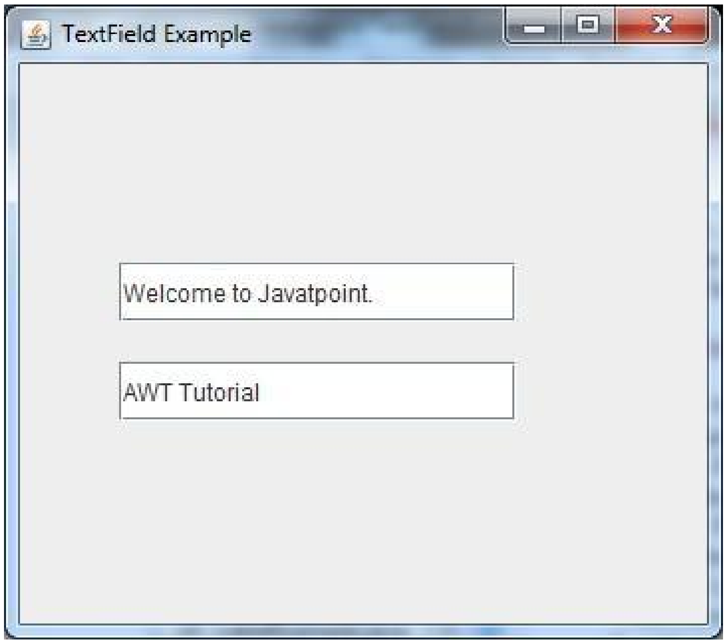
ImageIcon

```
import javax.swing.*;
class LabelExample
{
public static void main(String args[])
    {
    JFrame f= new JFrame("Example");
    ImageIcon im=new ImageIcon("hell.jpg");
    f.add(im);
    f.setSize(300,300);
    f.setLayout(null);
    f.setVisible(true);
    }
    }
```

JTextField program

Java JTextField Example

```
import javax.swing.*;
class TextFieldExample
{
public static void main(String args[])
    {
    JFrame f= new JFrame("TextField Example");
    JTextField t1,t2;
    t1=new JTextField("Welcome to Javatpoint.");
    t1.setBounds(50,100, 200,30);
    t2=new JTextField("AWT Tutorial");
    t2.setBounds(50,150, 200,30);
    f.add(t1); f.add(t2);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
    }
    }
```



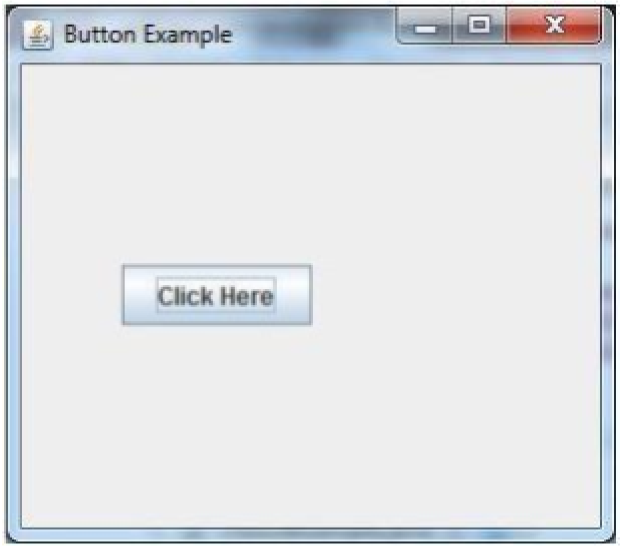
Welcome to Javatpoint.

AWT Tutorial

Java JButton Example

```
import javax.swing.*;
public class ButtonExample {
public static void main(String[] args) {
    JFrame f=new JFrame("Button Example");
    JButton b=new JButton("Click Here");
    b.setBounds(50,100,95,30);
    f.add(b);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

Output:

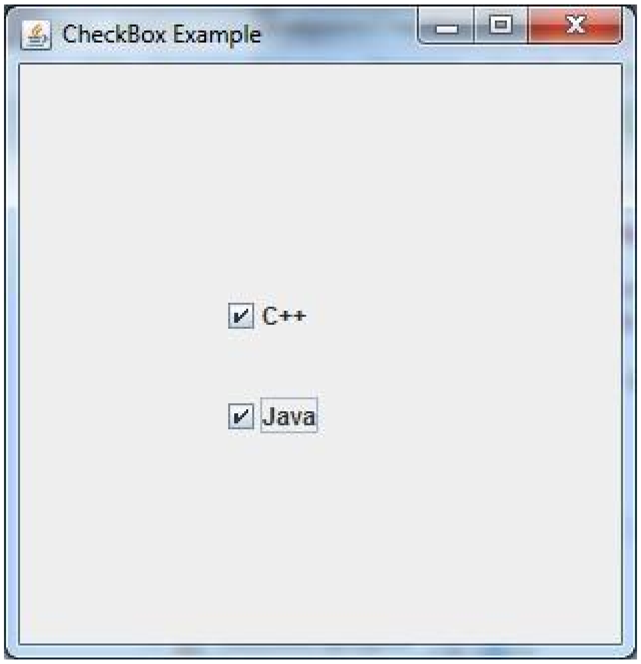


Java JCheckBox

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.

```
mport 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();

    }}
```



Java JRadioButton

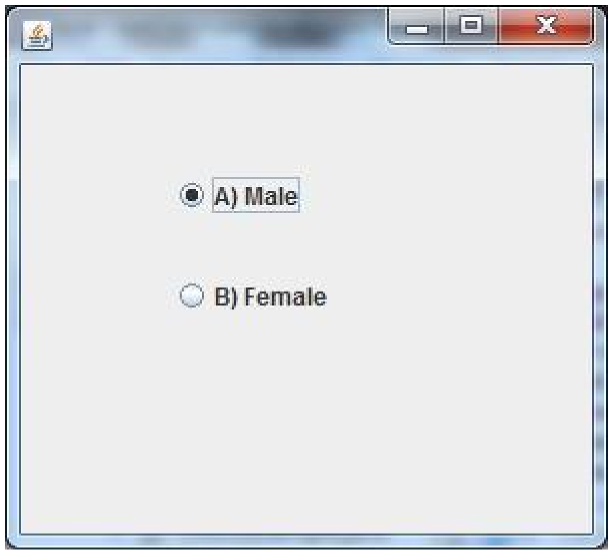
The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

It should be added in ButtonGroup to select one radio button only.

JRadioButton class Example

```
import javax.swing.*;
```

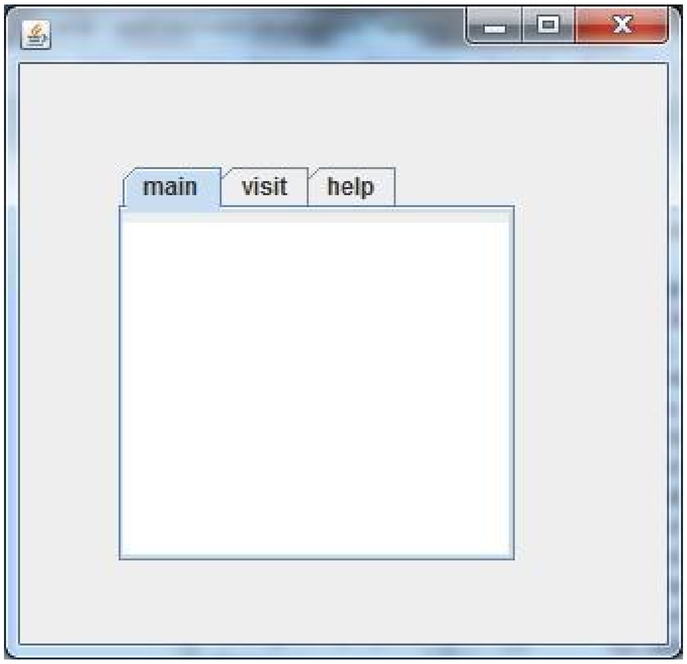
```
public class RadioButtonExample {
JFrame f;
RadioButtonExample(){
f=new JFrame();
JRadioButton r1=new JRadioButton("A) Male");
JRadioButton r2=new JRadioButton("B) Female");
r1.setBounds(75,50,100,30);
r2.setBounds(75,100,100,30);
ButtonGroup bg=new ButtonGroup();
bg.add(r1);bg.add(r2);
f.add(r1);f.add(r2);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String[] args) {
    new RadioButtonExample();
}
}
```



Java JTabbedPane

The JTabbedPane class is used to switch between a group of components by clicking on a tab with a given title or icon. It inherits JComponent class.

```
import javax.swing.*;
public class TabbedPaneExample {
JFrame f;
TabbedPaneExample(){
    f=new JFrame();
    JTextArea ta=new JTextArea(200,200);
    JPanel p1=new JPanel();
    p1.add(ta);
    JPanel p2=new JPanel();
    JPanel p3=new JPanel();
    JTabbedPane tp=new JTabbedPane();
    tp.setBounds(50,50,200,200);
    tp.add("main",p1);
    tp.add("visit",p2);
    tp.add("help",p3);
    f.add(tp);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
public static void main(String[] args) {
    new TabbedPaneExample();
}}
```



Java JScrollPane

A JscrollPane is used to make scrollable view of a component. When screen size is limited, we use a scroll pane to display a large component or a component whose size can change dynamically.

Constructor	Purpose

JScrollPane()
JScrollPane(Component)
JScrollPane(int, int)
JScrollPane(Component, int, int)
	It creates a scroll pane. The Component parameter, when present, sets the scroll pane's client. The two int parameters, when present, set the vertical and horizontal scroll bar policies (respectively).

Useful Methods

Modifier	Method	Description
void	setColumnHeaderView(Component)	It sets the column header for the scroll pane.
void	setRowHeaderView(Component)	It sets the row header for the scroll pane.

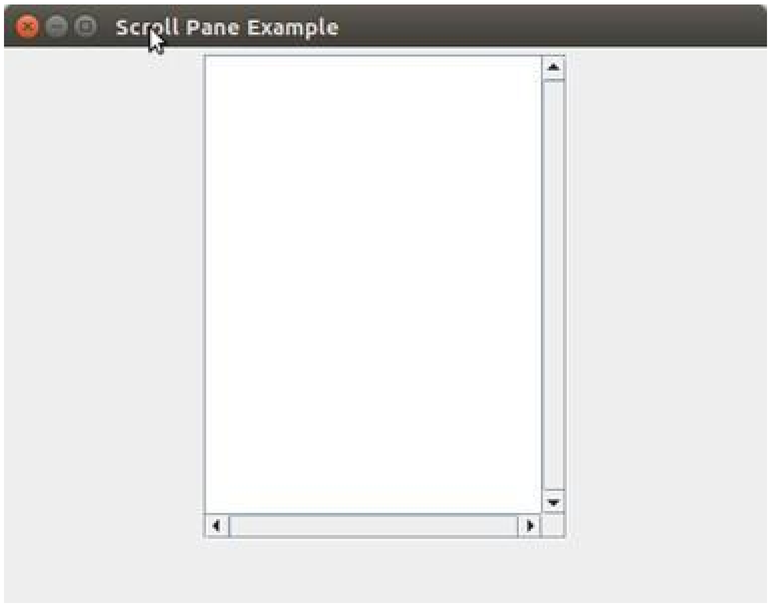
void
Component
	setCorner(String, Component)
	getCorner(String)
		It sets or gets the specified corner. The int parameter specifies which corner and must be one of the following constants defined in ScrollPaneConstants: UPPER_LEFT_CORNER, UPPER_RIGHT_CORNER, LOWER_LEFT_CORNER, LOWER_RIGHT_CORNER, LOWER_LEADING_CORNER, LOWER_TRAILING_CORNER, UPPER_LEADING_CORNER, UPPER_TRAILING_CORNER.

void	setViewportView(Component)	Set the scroll pane's client.

JScrollPane Example

```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JtextArea;
```

```

public class JScrollPaneExample {
    private static final long serialVersionUID = 1L;

  private static void createAndShowGUI() {

    // Create and set up the window.
    final JFrame frame = new JFrame("Scroll Pane Example");

    // Display the window.
    frame.setSize(500, 500);
    frame.setVisible(true);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    // set flow layout for the frame
    frame.getContentPane().setLayout(new FlowLayout());

    JTextArea textArea = new JTextArea(20, 20);
    JScrollPane scrollableTextArea = new JScrollPane(textArea);

    scrollableTextArea.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
    scrollableTextArea.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

    frame.getContentPane().add(scrollableTextArea);
  }
  public static void main(String[] args) {


    javax.swing.SwingUtilities.invokeLater(new Runnable() {

      public void run() {
        createAndShowGUI();
      }
    });
  }
}
```



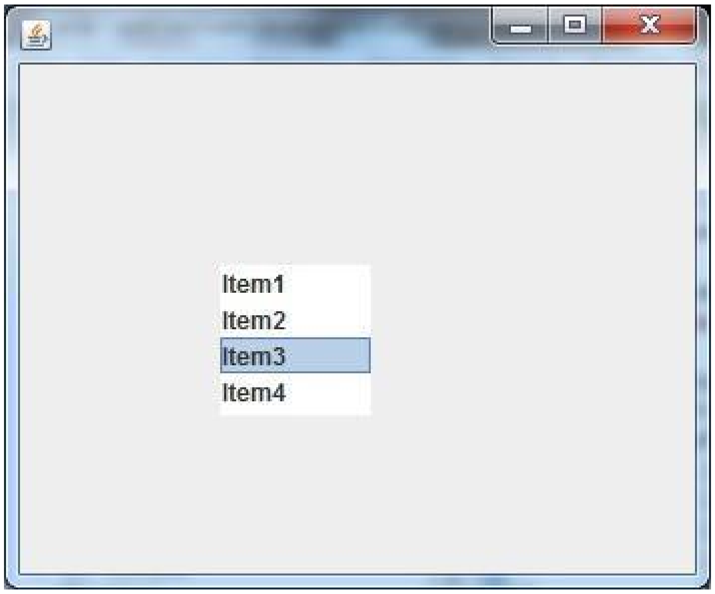
Java JList

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.

Java JList Example

```
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();
}}
```

Output:

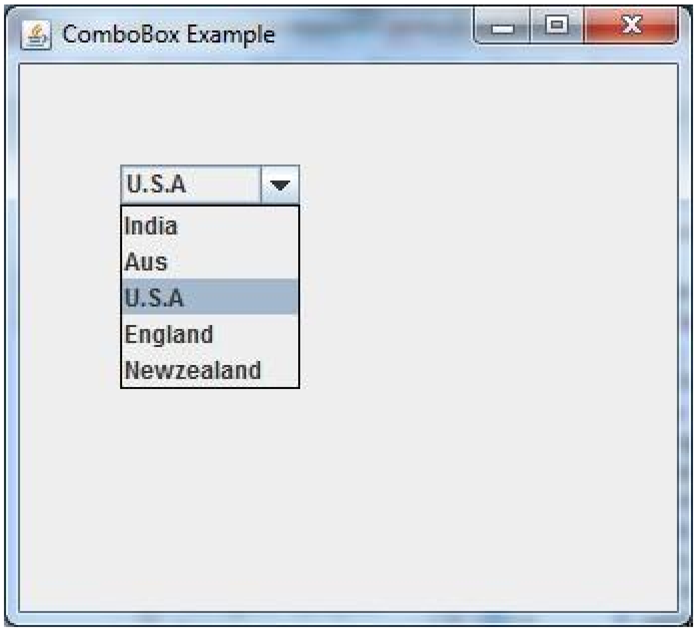


Java JComboBox

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.

Java JComboBox Example

```
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();
}
}
```



Java JMenuBar, JMenu and JMenuItem

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 class declaration

```
public class JMenuBar extends JComponent implements MenuElement, Accessible
```

JMenu class declaration

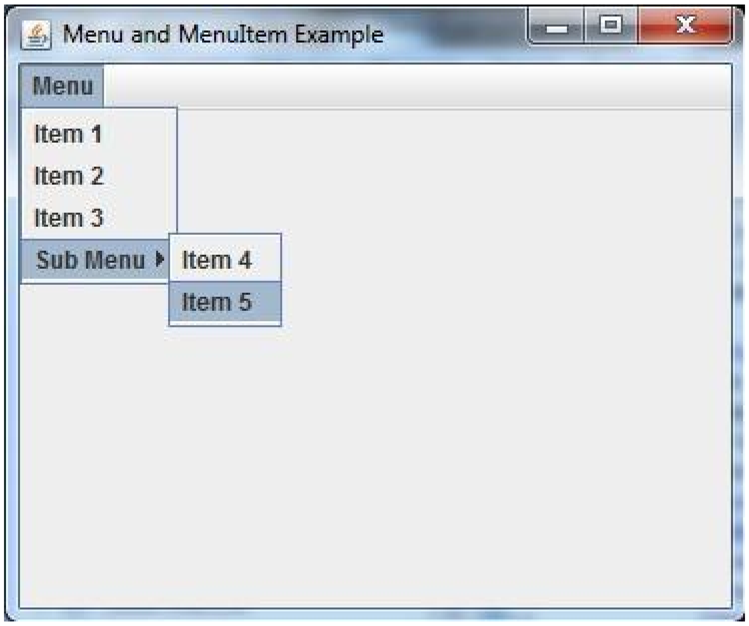
```
public class JMenu extends JMenuItem implements MenuElement, Accessible
```

JMenuItem class declaration

```
public class JMenuItem extends AbstractButton implements Accessible, MenuElement
```

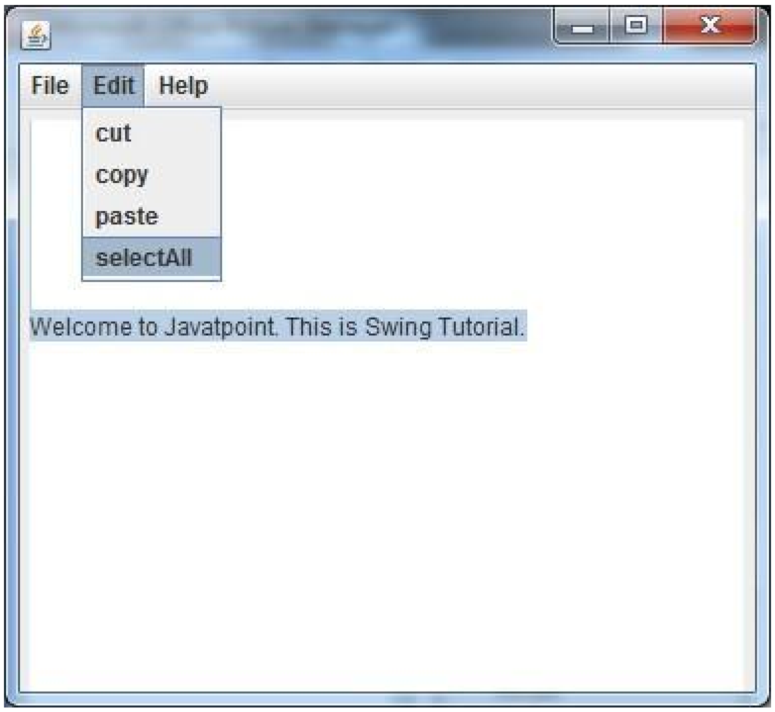
Java JMenuItem and JMenu Example

```
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();
}}
```



Example of creating Edit menu for Notepad:

```
import javax.swing.*;
import java.awt.event.*;
public class MenuExample implements ActionListener{
JFrame f;
JMenuBar mb;
JMenu file,edit,help;
JMenuItem cut,copy,paste,selectAll;
JTextArea ta;
MenuExample(){
f=new JFrame();
cut=new JMenuItem("cut");
copy=new JMenuItem("copy");
paste=new JMenuItem("paste");
selectAll=new JMenuItem("selectAll");
cut.addActionListener(this);
copy.addActionListener(this);
paste.addActionListener(this);
selectAll.addActionListener(this);
mb=new JMenuBar();
file=new JMenu("File");
edit=new JMenu("Edit");
help=new JMenu("Help");
edit.add(cut);edit.add(copy);edit.add(paste);edit.add(selectAll);
mb.add(file);mb.add(edit);mb.add(help);
ta=new JTextArea();
ta.setBounds(5,5,360,320);
f.add(mb);f.add(ta);
f.setJMenuBar(mb);
f.setLayout(null);
f.setSize(400,400);
f.setVisible(true);
}
public void actionPerformed(ActionEvent e) {
if(e.getSource()==cut)
ta.cut();
if(e.getSource()==paste)
ta.paste();
if(e.getSource()==copy)
ta.copy();
if(e.getSource()==selectAll)
ta.selectAll();
}
public static void main(String[] args) {
    new MenuExample();
}
}

```



Welcome to Javatpoint. This is Swing Tutorial.

Java JDialog

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.

JDialog class declaration

Let's see the declaration for javax.swing.JDialog class.

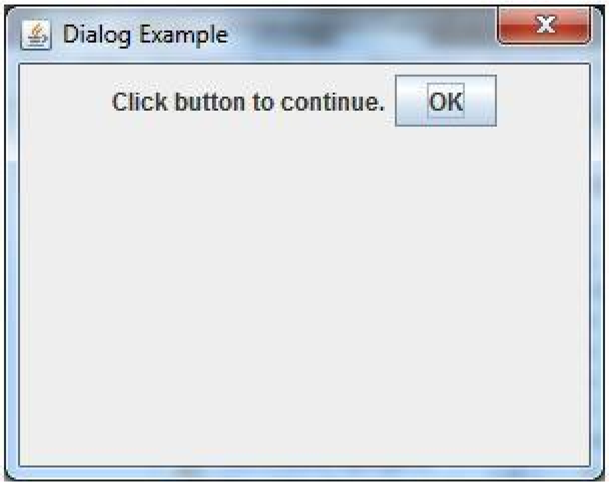
```
public class JDialog extends Dialog implements WindowConstants, Accessible, RootPaneContainer
```

Commonly used Constructors:

Constructor	Description
JDialog()	It is used to create a modeless dialog without a title and without a specified Frame owner.
JDialog(Frame owner)	It is used to create a modeless dialog with specified Frame as its owner and an empty title.
JDialog(Frame owner, String title, boolean modal)	It is used to create a dialog with the specified title, owner Frame and modality.

Java JDialog 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();
    }
}

```



Click button to continue.