APPLETS:

An **applet** is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

There are some important differences between an applet and a standalone Java application, including the following −

- An applet is a Java class that extends the java.applet.Applet class.
- A main() method is not invoked on an applet, and an applet class will not define main().
- Applets are designed to be embedded within an HTML page.
- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
- A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
- The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
- Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.
- Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file.

Life Cycle of an Applet

Four methods in the Applet class gives you the framework on which you build any serious applet −

- **public void init()** – This method is the first method to be called by the browser and it is executed only once. So, the programmer should use this method to initialize any variables, creating components and creating threads, etc. It is called after the param tags inside the applet tag have been processed.
- **public void start()** − This method is called after init() method and each time the applet is revisited by the user. For example, the user minimized and the webpage that contains the applet is moved to another page then this method's execution is stopped. Any calculations and processing of data should be done in this method and result is displayed.

- **public void stop()** − This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.
- **public void destroy()** − This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.

Creation of applet:

To create an applet that displays “Hello, World” in the applet frame. We can take the help of paint() method of component class of *java.awt* package.

- **paint** − Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

A "Hello, World" Applet

Following is a simple applet named HelloWorldApplet.java −

```
import java.applet.*;
import java.awt.*;
public class HelloWorldApplet extends Applet {
  public void paint (Graphics g) {
    g.drawString ("Hello, World", 25, 50);
  }
}
```

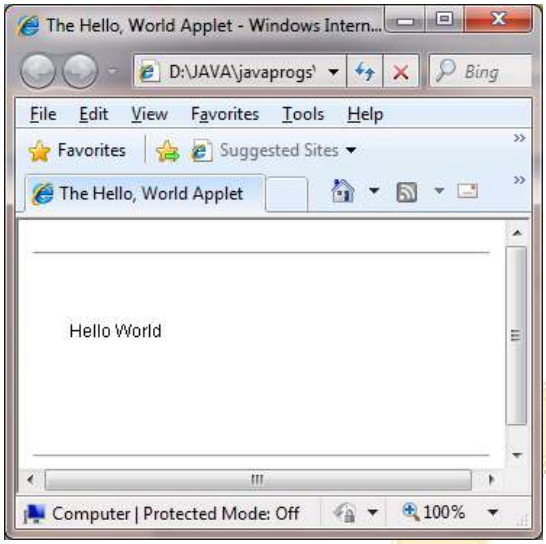
After creation of applet save the program with class name. Because the class is public. Then compile the above program “HelloWorldApplet.class” file created.

Invoking the applet in web browser:

```
<html>
<head>   <title>Hello World </title> </head>
<body>
<applet code=”HelloWorldApplet.class” width=”500”  height=”400” ></applet>
</body>
</html>
```

Save the above code with “hello.html”. This HTML page Contains the applet which can be opened in the browser, or an “appletviewer” can be used to test the applet.

> appletviewer hello.html



Example 2: Write a java applet program describe about life cycle of applet.

```
import java.applet.*;
import java.awt.*;
public class HelloWorldApplet extends Applet {
		String msg="";
		public void init()
		{
			msg+="INIT ";
		}
		public void start()
		{
			msg+="START ";
		}
		public void stop()
		{
			msg+="STOP ";
		}
		public void destroy()
		{
			msg+="DESTROY ";
		}
  public void paint (Graphics g) {
			System.out.println(msg);
    g.drawString (msg, 25, 50);
  }
}
```

Uses of applet:

- ✓ Applets are used on internet for creating dynamic web pages. There are two types of web pages: Static and Dynamic. Static web pages provide some information to the user but the user cannot interact with the web page other than viewing the information. Dynamic web pages interact with the user at runtime. For example, a student can type his hall ticket number in a text field and click retrieve button to get back his results from the university server.
- ✓ Another use of applets is for creating animation and games where images can be displayed or, moved giving a visual impression that they are alive.

These import statements bring the classes into the scope of our applet class −

- java.applet.Applet
- java.awt.Graphics

Without those import statements, the Java compiler would not recognize the classes Applet and Graphics, which the applet class refers to.

The Applet Class

Every applet is an extension of the *java.applet.Applet class*. The base Applet class provides methods that a derived Applet class may call to obtain information and services from the browser context.

These include methods that do the following −

- Get applet parameters
- Get the network location of the HTML file that contains the applet
- Get the network location of the applet class directory
- Print a status message in the browser
- Fetch an image
- Fetch an audio clip
- Play an audio clip
- Resize the applet

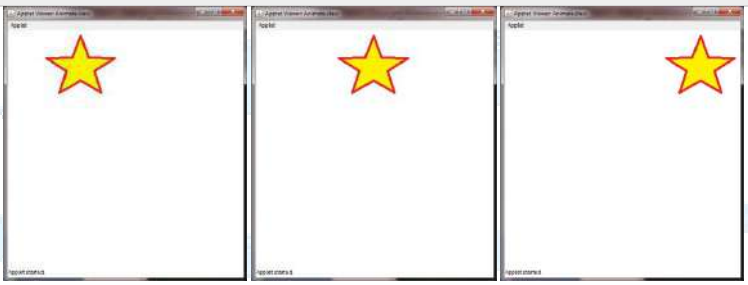
Additionally, the Applet class provides an interface by which the viewer or browser obtains information about the applet and controls the applet's execution. The viewer may −

- Request information about the author, version, and copyright of the applet
- Request a description of the parameters the applet recognizes
- Initialize the applet
- Destroy the applet
- Start the applet's execution
- Stop the applet's execution

Note:

- The code attribute of the <applet> tag is required. It specifies the Applet class to run. Width and height are also required to specify the initial size of the panel in which an applet runs. The applet directive must be closed with an </applet> tag.
- If an applet takes parameters, values may be passed for the parameters by adding <param> tags between <applet> and </applet>. The browser ignores text and other tags between the applet tags.
- Non-Java-enabled browsers do not process <applet> and </applet>. Therefore, anything that appears between the tags, not related to the applet, is visible in non-Java-enabled browsers.

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

public class Animate extends Applet {
  public void paint (Graphics g) {
            Image img=getImage(getDocumentBase(),"hello.jpg");
            for(int i=0;i<800;i++)
            {
                        g.drawImage(img,i,0,null);
                        try{
                                    Thread.sleep(20);
                        }catch(Exception e){}
            }
  }
}
```



repaint():-

The **repaint()** method is sent to a Component when it needs to be repainted. This happens when a window is moved, or resized, or unhidden. It also happens when a webpage contains an image and the pixels of the image are arriving slowly down the wire.

update(Graphics):-

The repaint () method causes then AWT runtime system to execute the update () method of the Component class which clears the window with the background color of the applet and then calls the paint () method. But sometimes erasing the background is undesirable. For example: If user wants to display the x and y coordinates of each location where the mouse is clicked instead of only the currently clicked location then it is not possible using the default version of the update () method. Each time the user clicks in the applets's window, the coordinates of the previously clicked location are erased and only the coordinates of the currently clicked location are displayed. To overcome this problem, it is necessary to override the update () method to invoke paint () directly, thus avoiding the erasure of the component's background.

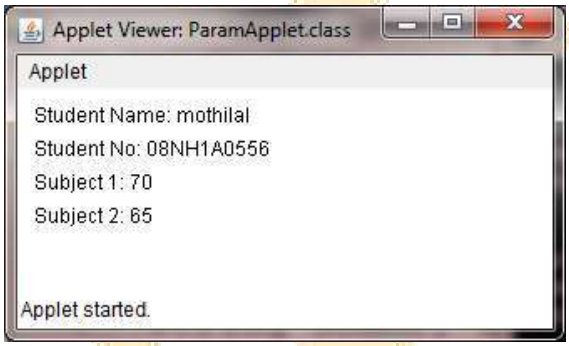
Passing Parameters to Applets:

We can pass the parameters to the applet. When we create we use those parameters by using getParameter() method. This method returns the parameters in the form of String object.

Ex: Write an applet to take student name, student no, and two subject's marks.

```
import java.applet.*;
import java.awt.*;
public class ParamApplet extends Applet {
String sname, sno, s1, s2;
public void start(){
sname=getParameter("stdname");
sno=getParameter("stdno");
s1=getParameter("sub1");
s2=getParameter("sub2");
}
  public void paint (Graphics g) {
            g.drawString("Student Name: "+sname,10,20);
            g.drawString("Student No: "+sno,10,40);
            g.drawString("Subject 1: "+s1,10,60);
            g.drawString("Subject 2: "+s2,10,80);

  }
}
```

```
<html>
  <title>The Parameters Applet</title>
  <hr>
  <applet code = "ParamApplet.class" width = "320" height = "120">
  <param name="stdname" value="mothilal">
  <param name="stdno" value="08NH1A0556">
  <param name="sub1" value="70">
  <param name="sub2" value="65">
  </applet>
  <hr>
</html>
```



What is an Event?

Change in the state of an object is known as **Event**, i.e., event describes the change in the state of the source. Events are generated as a result of user interaction with the graphical user interface components. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from the list, and scrolling the page are the activities that cause an event to occur.

Types of Event:

The events can be broadly classified into two categories −

Foreground Events − these events require direct interaction of the user. They are generated as consequences of a person interacting with the graphical components in the Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page, etc.

Background Events − These events require the interaction of the end user. Operating system interrupts, hardware or software failure, timer expiration, and operation completion are some examples of background events.

What is Event Handling?

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has a code which is known as an event handler that is executed when an event occurs.

Java uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events.

The Delegation Event Model has the following key participants.

Source − the source is an object on which the event occurs. Source is responsible for providing information of the occurred event to its handler. Java provides us with classes for the source object.

Listener − It is also known as event handler. The listener is responsible for generating a response to an event. From the point of view of Java implementation, the listener is also an object. The listener waits till it receives an event. Once the event is received, the listener processes the event and then returns.

The benefit of this approach is that the user interface logic is completely separated from the logic that generates the event. The user interface element is able to delegate the processing of an event to a separate piece of code.

In this model, the listener needs to be registered with the source object so that the listener can receive the event notification. This is an efficient way of handling the event because the event notifications are sent only to those listeners who want to receive them.

Steps Involved in Event Handling

Step 1 − The user clicks the button and the event is generated.

Step 2 − The object of concerned event class is created automatically and information about the source and the event get populated within the same object.

Step 3 − Event object is forwarded to the method of the registered listener class.

Step 4 − The method is gets executed and returns.

Points to Remember About the Listener

In order to design a listener class, you have to develop some listener interfaces. These Listener interfaces forecast some public abstract callback methods, which must be implemented by the listener class.

If you do not implement any of the predefined interfaces, then your class cannot act as a listener class for a source object.

Callback Methods

These are the methods that are provided by API provider and are defined by the application programmer and invoked by the application developer. Here the callback methods represent an event method. In response to an event, java jre will fire callback method. All such callback methods are provided in listener interfaces.

If a component wants some listener to listen ot its events, the source must register itself to the listener.

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.

Java Event classes and Listener interfaces:

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){}

Java Event Handling Code

We can put the event handling code into one of the following places:

1. Within class
2. Other class
3. Anonymous class

Java event handling by implementing ActionListener

```
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.

2) Java event handling by outer class

```
import java.awt.*;
import java.awt.event.*;
class AEvent2 extends Frame
{
	TextField tf;
	AEvent2()
	{
		//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
		Outer o=new Outer(this);
		b.addActionListener(o);//passing outer class instance
		//add components and set size, layout and visibility
		add(b);add(tf);
		setSize(300,300);
		setLayout(null);
		setVisible(true);
	}
	public static void main(String args[]){
		new AEvent2();
	}
}
import java.awt.event.*;
class Outer implements ActionListener
{
	AEvent2 obj;
	Outer(AEvent2 obj)
	{
		this.obj=obj;
	}
	public void actionPerformed(ActionEvent e)
	{
		obj.tf.setText("welcome");
	}
}
```

3) Java event handling by anonymous class

```
import java.awt.*;
import java.awt.event.*;
class AEvent3 extends Frame
{
	TextField tf;
	AEvent3()
	{
		tf=new TextField();
		tf.setBounds(60,50,170,20);
```

```
		Button b=new Button("click me");
		b.setBounds(50,120,80,30);

		b.addActionListener(new ActionListener(){
		public void actionPerformed(ActionEvent e){
		tf.setText("hello");
		}
		});
		add(b);add(tf);
		setSize(300,300);
		setLayout(null);
		setVisible(true);
	}
	public static void main(String args[]){
		new AEvent3();
	}
}
```

Java AWT Button

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

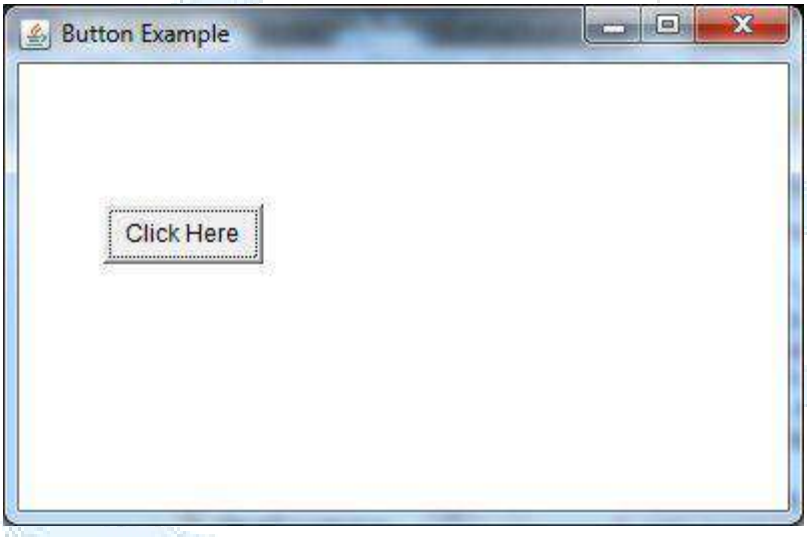
AWT Button Class declaration

```
1. public class Button extends Component implements Accessible
```

Java AWT Button Example

```
1. import java.awt.*;
2. public class ButtonExample {
3. public static void main(String[] args) {
4.     Frame f=new Frame("Button Example");
5.     Button b=new Button("Click Here");
6.     b.setBounds(50,100,80,30);
7.     f.add(b);
8.     f.setSize(400,400);
9.     f.setLayout(null);
10.    f.setVisible(true);
11. }
12. }
```

Output:

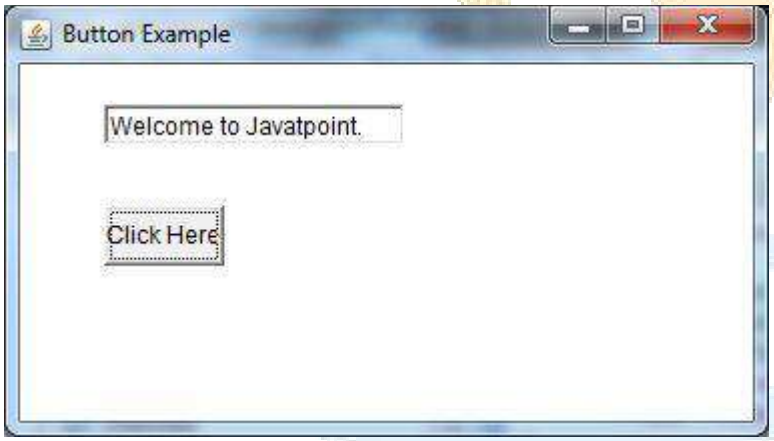


Java AWT Button Example with ActionListener

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

```
3. public class ButtonExample {
4. public static void main(String[] args) {
5.     Frame f=new Frame("Button Example");
6.     final TextField tf=new TextField();
7.     tf.setBounds(50,50, 150,20);
8.     Button b=new Button("Click Here");
9.     b.setBounds(50,100,60,30);
10.    b.addActionListener(new ActionListener(){
11.    public void actionPerformed(ActionEvent e){
12.            tf.setText("Welcome to Javatpoint.");
13.        }
14.    });
15.    f.add(b);f.add(tf);
16.    f.setSize(400,400);
17.    f.setLayout(null);
18.    f.setVisible(true);
19. }
20. }
```

Output:



Java AWT Label

The object of Label class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly.

AWT Label Class Declaration

```
1. public class Label extends Component implements Accessible
```

Java Label Button Example

```
1. import java.awt.*;
2. class LabelExample{
3. public static void main(String args[]){
4.     Frame f= new Frame("Label Example");
5.     Label l1,l2;
6.     l1=new Label("First Label.");
7.     l1.setBounds(50,100, 100,30);
8.     l2=new Label("Second Label.");
9.     l2.setBounds(50,150, 100,30);
10.    f.add(l1); f.add(l2);
11.    f.setSize(400,400);
12.    f.setLayout(null);
13.    f.setVisible(true);
14. }
15. }
```

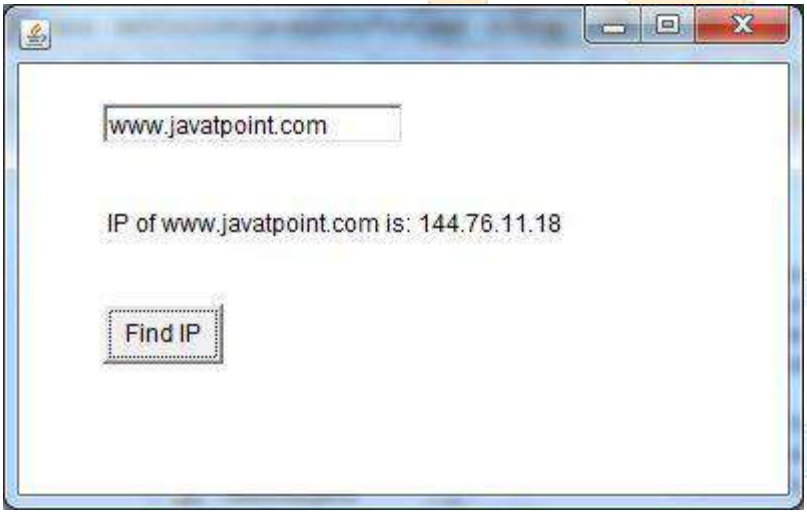
Output:

Java AWT Label Example with ActionListener

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class LabelExample extends Frame implements ActionListener{
4.     TextField tf; Label l; Button b;
5.     LabelExample(){
6.         tf=new TextField();
7.         tf.setBounds(50,50, 150,20);
8.         l=new Label();
9.         l.setBounds(50,100, 250,20);
10.        b=new Button("Find IP");
11.        b.setBounds(50,150,60,30);
```

```
12.         b.addActionListener(this);
13.         add(b);add(tf);add(l);
14.         setSize(400,400);
15.         setLayout(null);
16.         setVisible(true);
17.     }
18.     public void actionPerformed(ActionEvent e) {
19.         try{
20.         String host=tf.getText();
21.         String ip=java.net.InetAddress.getByName(host).getHostAddress();
22.         l.setText("IP of "+host+" is: "+ip);
23.         }catch(Exception ex){System.out.println(ex);}
24.     }
25.     public static void main(String[] args) {
26.         new LabelExample();
27.     }
28. }
```

Output:



Java AWT TextField

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

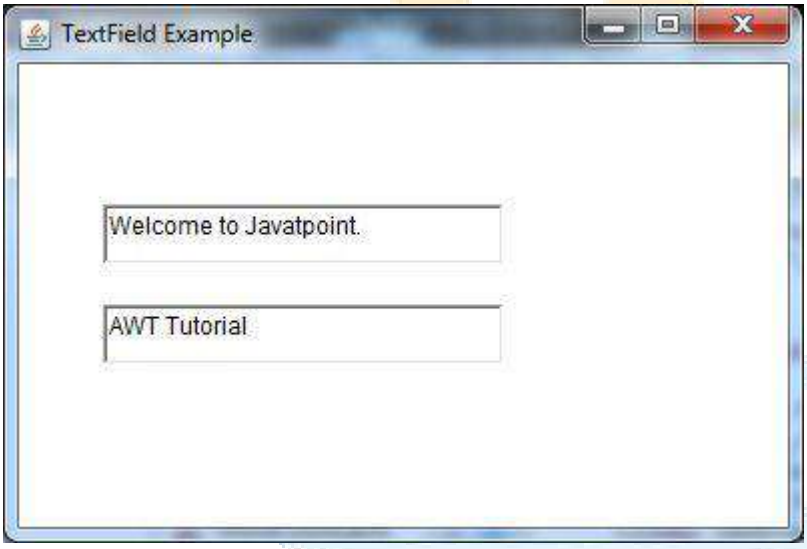
AWT TextField Class Declaration

```
1. public class TextField extends TextComponent
```

Java AWT TextField Example

```
1. import java.awt.*;
2. class TextFieldExample{
3. public static void main(String args[]){
4.     Frame f= new Frame("TextField Example");
5.     TextField t1,t2;
6.     t1=new TextField("Welcome to Javatpoint.");
7.     t1.setBounds(50,100, 200,30);
8.     t2=new TextField("AWT Tutorial");
9.     t2.setBounds(50,150, 200,30);
10.    f.add(t1); f.add(t2);
11.    f.setSize(400,400);
12.    f.setLayout(null);
13.    f.setVisible(true);
14. }
15. }
```

Output:

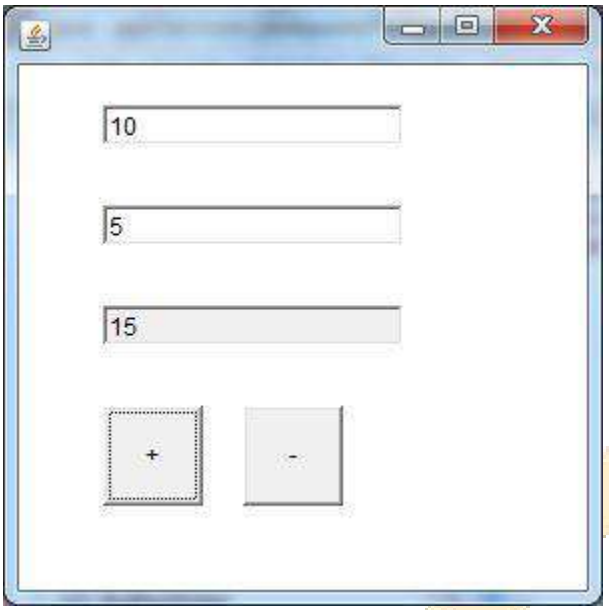


Java AWT TextField Example with ActionListener

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class TextFieldExample extends Frame implements ActionListener{
4.     TextField tf1,tf2,tf3;
5.     Button b1,b2;
6.     TextFieldExample(){
7.         tf1=new TextField();
8.         tf1.setBounds(50,50,150,20);
9.         tf2=new TextField();
10.        tf2.setBounds(50,100,150,20);
```

```
11.       tf3=new TextField();
12.       tf3.setBounds(50,150,150,20);
13.       tf3.setEditable(false);
14.       b1=new Button("+");
15.       b1.setBounds(50,200,50,50);
16.       b2=new Button("-");
17.       b2.setBounds(120,200,50,50);
18.       b1.addActionListener(this);
19.       b2.addActionListener(this);
20.       add(tf1);add(tf2);add(tf3);add(b1);add(b2);
21.       setSize(300,300);
22.       setLayout(null);
23.       setVisible(true);
24.    }
25.    public void actionPerformed(ActionEvent e) {
26.       String s1=tf1.getText();
27.       String s2=tf2.getText();
28.       int a=Integer.parseInt(s1);
29.       int b=Integer.parseInt(s2);
30.       int c=0;
31.       if(e.getSource()==b1){
32.          c=a+b;
33.       }else if(e.getSource()==b2){
34.          c=a-b;
35.       }
36.       String result=String.valueOf(c);
37.       tf3.setText(result);
38.    }
39. public static void main(String[] args) {
40.    new TextFieldExample();
41. }
42. }
```

Output:



Java AWT TextArea

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

AWT TextArea Class Declaration

```
1. public class TextArea extends TextComponent
```

Java AWT TextArea Example

```
1. import java.awt.*;
2. public class TextAreaExample
3. {
4.     TextAreaExample(){
5.       Frame f= new Frame();
6.          TextArea area=new TextArea("Welcome to javatpoint");
7.       area.setBounds(10,30, 300,300);
8.       f.add(area);
9.       f.setSize(400,400);
10.      f.setLayout(null);
11.      f.setVisible(true);
12.   }
```

```
13. public static void main(String args[])
14. {
15.    new TextAreaExample();
16. }
17. }
```

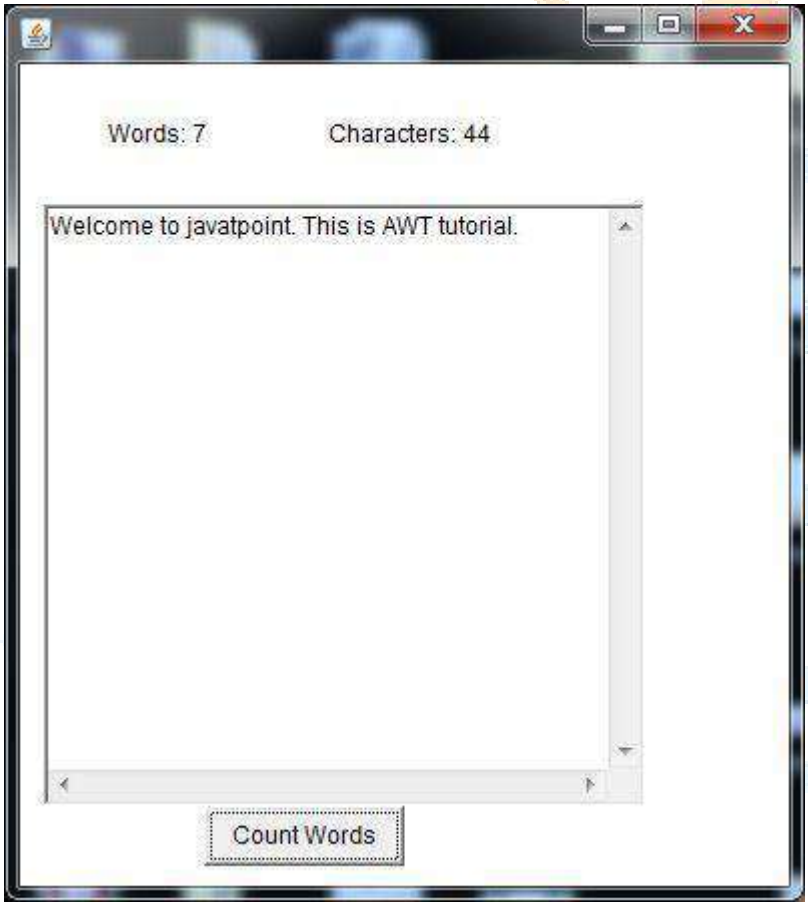
Output:

Java AWT TextArea Example with ActionListener

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.  public class TextAreaExample extends Frame implements ActionListener{
4.  Label l1,l2;
5.  TextArea area;
6.  Button b;
7.  TextAreaExample(){
8.     l1=new Label();
9.     l1.setBounds(50,50,100,30);
10.    l2=new Label();
11.    l2.setBounds(160,50,100,30);
12.    area=new TextArea();
13.    area.setBounds(20,100,300,300);
```

```
14.     b=new Button("Count Words");
15.     b.setBounds(100,400,100,30);
16.     b.addActionListener(this);
17.     add(l1);add(l2);add(area);add(b);
18.     setSize(400,450);
19.     setLayout(null);
20.     setVisible(true);
21. }
22. public void actionPerformed(ActionEvent e){
23.     String text=area.getText();
24.     String words[]=text.split("\\s");
25.     l1.setText("Words: "+words.length);
26.     l2.setText("Characters: "+text.length());
27. }
28. public static void main(String[] args) {
29.     new TextAreaExample();
30. }
31. }
```

Output:



Java AWT Checkbox

The Checkbox 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".

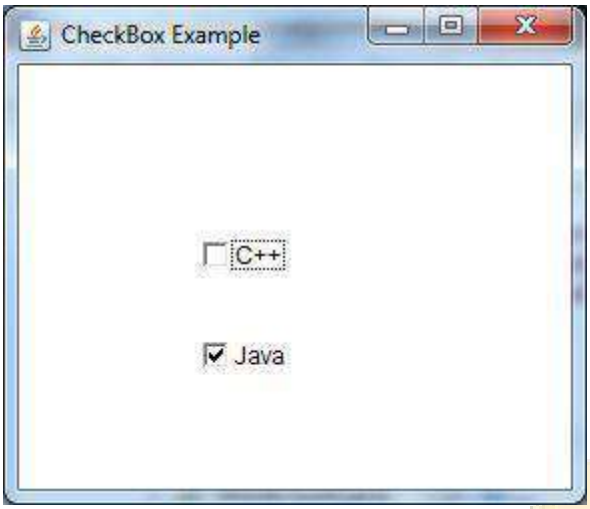
AWT Checkbox Class Declaration

```
1. public class Checkbox extends Component implements ItemSelectable, Accessible
```

Java AWT Checkbox Example

```
1. import java.awt.*;
2. public class CheckboxExample
3. {
4.     CheckboxExample(){
5.       Frame f= new Frame("Checkbox Example");
6.        Checkbox checkbox1 = new Checkbox("C++");
7.        checkbox1.setBounds(100,100, 50,50);
8.        Checkbox checkbox2 = new Checkbox("Java", true);
9.        checkbox2.setBounds(100,150, 50,50);
10.       f.add(checkbox1);
11.       f.add(checkbox2);
12.       f.setSize(400,400);
13.       f.setLayout(null);
14.       f.setVisible(true);
15.    }
16. public static void main(String args[])
17. {
18.    new CheckboxExample();
19. }
20. }
```

Output:

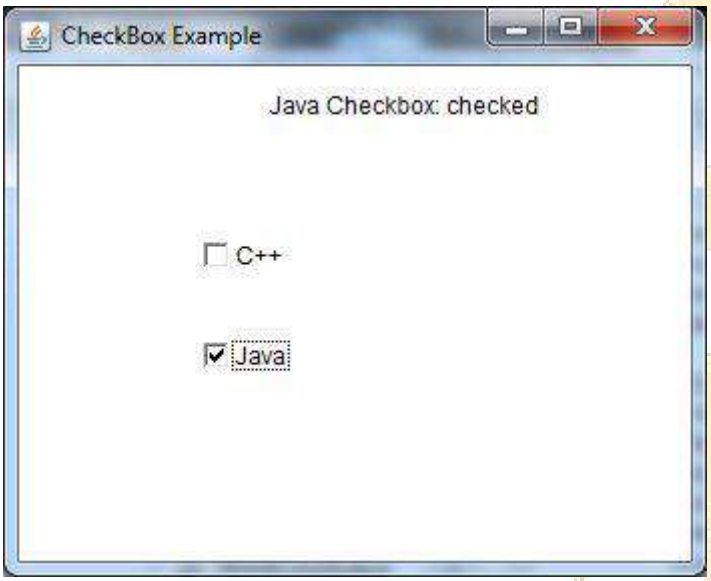


Java AWT Checkbox Example with ItemListener

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.  public class CheckboxExample
4.  {
5.      CheckboxExample(){
6.        Frame f= new Frame("CheckBox Example");
7.        final Label label = new Label();
8.        label.setAlignment(Label.CENTER);
9.        label.setSize(400,100);
10.       Checkbox checkbox1 = new Checkbox("C++");
11.       checkbox1.setBounds(100,100, 50,50);
12.       Checkbox checkbox2 = new Checkbox("Java");
13.       checkbox2.setBounds(100,150, 50,50);
14.       f.add(checkbox1); f.add(checkbox2); f.add(label);
15.       checkbox1.addItemListener(new ItemListener() {
16.           public void itemStateChanged(ItemEvent e) {
17.             label.setText("C++ Checkbox: "
18.             + (e.getStateChange()==1?"checked":"unchecked"));
19.           }
20.         });
21.       checkbox2.addItemListener(new ItemListener() {
22.           public void itemStateChanged(ItemEvent e) {
23.             label.setText("Java Checkbox: "
24.             + (e.getStateChange()==1?"checked":"unchecked"));
25.           }
26.         });
27.       f.setSize(400,400);
28.       f.setLayout(null);
```

```
29.         f.setVisible(true);
30.      }
31. public static void main(String args[])
32. {
33.     new CheckboxExample();
34. }
35. }
```

Output:



Java AWT CheckboxGroup

The object of CheckboxGroup class is used to group together a set of Checkbox. At a time only one check box button is allowed to be in "on" state and remaining check box button in "off" state. It inherits the object class.

Note: CheckboxGroup enables you to create radio buttons in AWT. There is no special control for creating radio buttons in AWT.

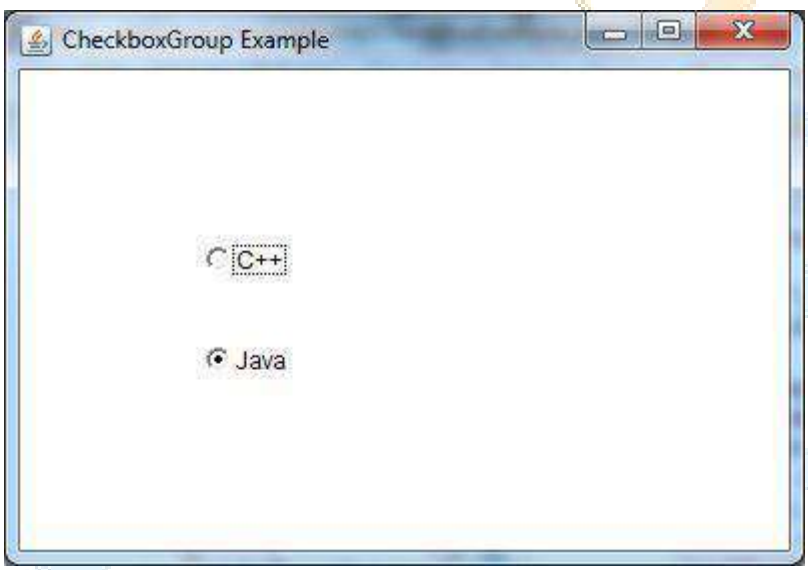
AWT CheckboxGroup Class Declaration

```
1. public class CheckboxGroup extends Object implements Serializable
```

Java AWT CheckboxGroup Example

```
1.  import java.awt.*;
2.  public class CheckboxGroupExample
3.  {
4.        CheckboxGroupExample(){
5.        Frame f= new Frame("CheckboxGroup Example");
6.         CheckboxGroup cbg = new CheckboxGroup();
7.         Checkbox checkBox1 = new Checkbox("C++", cbg, false);
8.         checkBox1.setBounds(100,100, 50,50);
9.         Checkbox checkBox2 = new Checkbox("Java", cbg, true);
10.        checkBox2.setBounds(100,150, 50,50);
11.        f.add(checkBox1);
12.        f.add(checkBox2);
13.        f.setSize(400,400);
14.        f.setLayout(null);
15.        f.setVisible(true);
16.     }
17. public static void main(String args[])
18. {
19.    new CheckboxGroupExample();
20. }
21. }
```

Output:

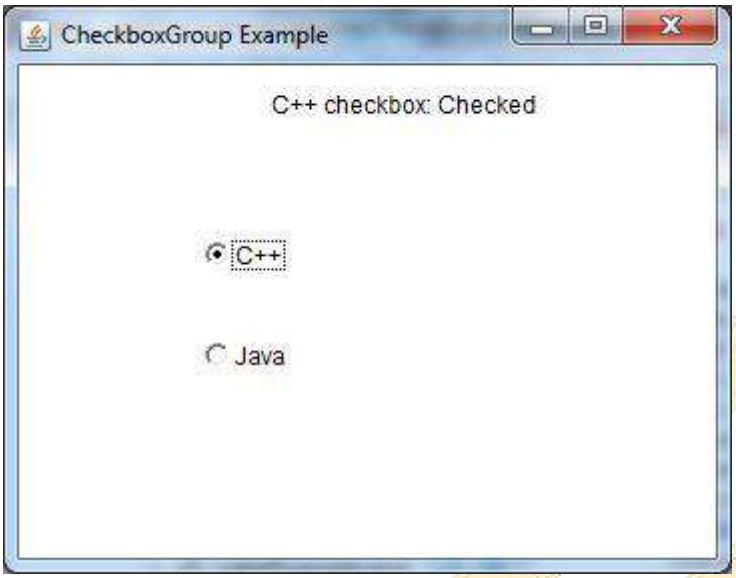


Java AWT CheckboxGroup Example with ItemListener

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.  public class CheckboxGroupExample
4.  {
```

```
5.      CheckboxGroupExample(){
6.        Frame f= new Frame("CheckboxGroup Example");
7.        final Label label = new Label();
8.        label.setAlignment(Label.CENTER);
9.        label.setSize(400,100);
10.       CheckboxGroup cbg = new CheckboxGroup();
11.       Checkbox checkBox1 = new Checkbox("C++", cbg, false);
12.       checkBox1.setBounds(100,100, 50,50);
13.       Checkbox checkBox2 = new Checkbox("Java", cbg, false);
14.       checkBox2.setBounds(100,150, 50,50);
15.       f.add(checkBox1); f.add(checkBox2); f.add(label);
16.       f.setSize(400,400);
17.       f.setLayout(null);
18.       f.setVisible(true);
19.       checkBox1.addItemListener(new ItemListener() {
20.          public void itemStateChanged(ItemEvent e) {
21.            label.setText("C++ checkbox: Checked");
22.          }
23.       });
24.       checkBox2.addItemListener(new ItemListener() {
25.          public void itemStateChanged(ItemEvent e) {
26.            label.setText("Java checkbox: Checked");
27.          }
28.       });
29.    }
30. public static void main(String args[])
31. {
32.    new CheckboxGroupExample();
33. }
34. }
```

Output:



Java AWT Choice

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 Component class.

AWT Choice Class Declaration

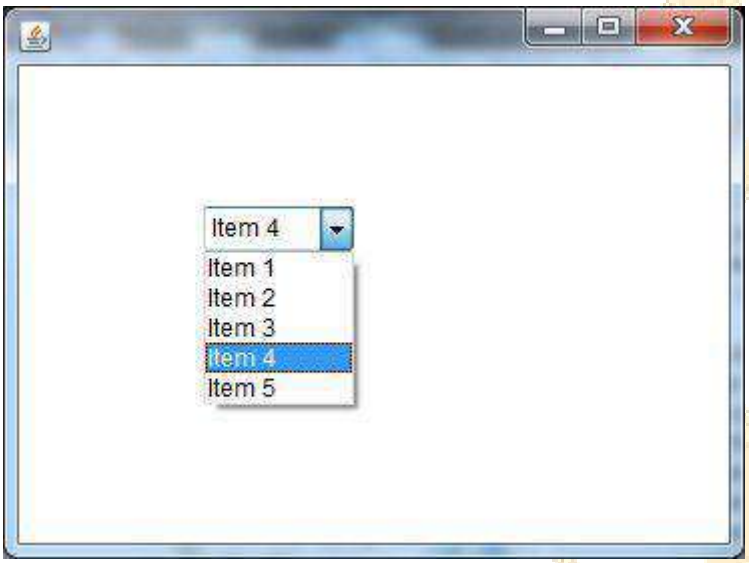
```
1. public class Choice extends Component implements ItemSelectable, Accessible
```

Java AWT Choice Example

```
1. import java.awt.*;
2. public class ChoiceExample
3. {
4.         ChoiceExample(){
5.         Frame f= new Frame();
6.         Choice c=new Choice();
7.         c.setBounds(100,100, 75,75);
8.         c.add("Item 1");
9.         c.add("Item 2");
10.        c.add("Item 3");
11.        c.add("Item 4");
12.        c.add("Item 5");
13.        f.add(c);
14.        f.setSize(400,400);
15.        f.setLayout(null);
```

```
16.         f.setVisible(true);
17.      }
18. public static void main(String args[])
19. {
20.    new ChoiceExample();
21. }
22. }
```

Output:



Java AWT Choice Example with ActionListener

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.  public class ChoiceExample
4.  {
5.          ChoiceExample(){
6.          Frame f= new Frame();
7.          final Label label = new Label();
8.          label.setAlignment(Label.CENTER);
9.          label.setSize(400,100);
10.         Button b=new Button("Show");
11.         b.setBounds(200,100,50,20);
12.         final Choice c=new Choice();
13.         c.setBounds(100,100, 75,75);
14.         c.add("C");
15.         c.add("C++");
16.         c.add("Java");
17.         c.add("PHP");
18.         c.add("Android");
```

```
19.         f.add(c);f.add(label); f.add(b);
20.         f.setSize(400,400);
21.         f.setLayout(null);
22.         f.setVisible(true);
23.         b.addActionListener(new ActionListener() {
24.             public void actionPerformed(ActionEvent e) {
25.          String data = "Programming language Selected: "+ c.getItem(c.getSelectedIndex());
26.          label.setText(data);
27.         }
28.         });
29.         }
30. public static void main(String args[])
31. {
32.    new ChoiceExample();
33. }
34. }
```

Output:

Java AWT List

The object of List class represents a list of text items. By the help of list, user can choose either one item or multiple items. It inherits Component class.

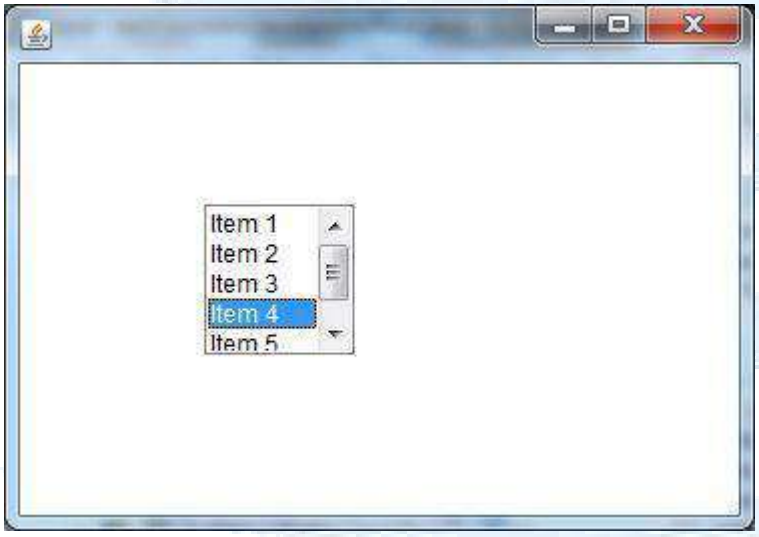
AWT List class Declaration

```
1. public class List extends Component implements ItemSelectable, Accessible
```

Java AWT List Example

```
1.  import java.awt.*;
2.  public class ListExample
3.  {
4.      ListExample(){
5.          Frame f= new Frame();
6.          List l1=new List(5);
7.          l1.setBounds(100,100, 75,75);
8.          l1.add("Item 1");
9.          l1.add("Item 2");
10.         l1.add("Item 3");
11.         l1.add("Item 4");
12.         l1.add("Item 5");
13.         f.add(l1);
14.         f.setSize(400,400);
15.         f.setLayout(null);
16.         f.setVisible(true);
17.     }
18. public static void main(String args[])
19. {
20.   new ListExample();
21. }
22. }
```

Output:

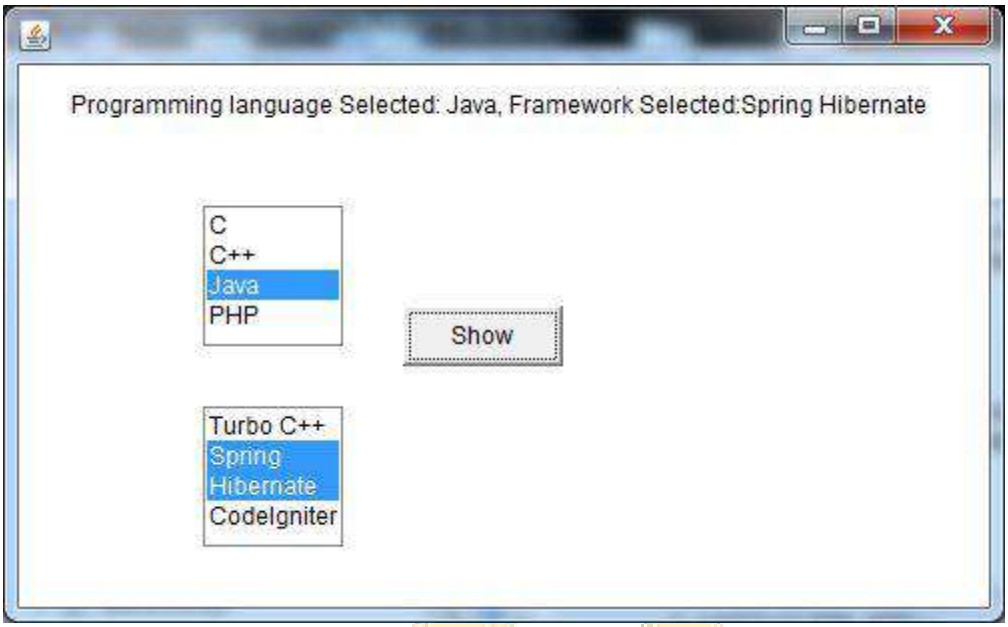


Java AWT List Example with ActionListener

```
1.  import java.awt.*;
```

```
2.  import java.awt.event.*;
3.  public class ListExample
4.  {
5.      ListExample(){
6.        Frame f= new Frame();
7.        final Label label = new Label();
8.        label.setAlignment(Label.CENTER);
9.        label.setSize(500,100);
10.       Button b=new Button("Show");
11.       b.setBounds(200,150,80,30);
12.       final List l1=new List(4, false);
13.       l1.setBounds(100,100, 70,70);
14.       l1.add("C");
15.       l1.add("C++");
16.       l1.add("Java");
17.       l1.add("PHP");
18.       final List l2=new List(4, true);
19.       l2.setBounds(100,200, 70,70);
20.       l2.add("Turbo C++");
21.       l2.add("Spring");
22.       l2.add("Hibernate");
23.       l2.add("CodeIgniter");
24.       f.add(l1); f.add(l2); f.add(label); f.add(b);
25.       f.setSize(450,450);
26.       f.setLayout(null);
27.       f.setVisible(true);
28.       b.addActionListener(new ActionListener() {
29.        public void actionPerformed(ActionEvent e) {
30.         String data = "Programming language Selected: "+l1.getItem(l1.getSelectedIndex());
31.         data += ", Framework Selected:";
32.         for(String frame:l2.getSelectedItems()){
33.                 data += frame + " ";
34.         }
35.         label.setText(data);
36.         }
37.        });
38. }
39. public static void main(String args[])
40. {
41.   new ListExample();
42. }
43. }
```

Output:



Java AWT MenuItem and Menu

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

The object of Menu class is a pull down menu component which is displayed on the menu bar. It inherits the MenuItem class.

AWT MenuItem class declaration

```
1.  public class MenuItem extends MenuCo
    mponent implements Accessible
```

AWT Menu class declaration

```
1.  public class Menu extends MenuItem i
    mplements MenuContainer, Accessible
```

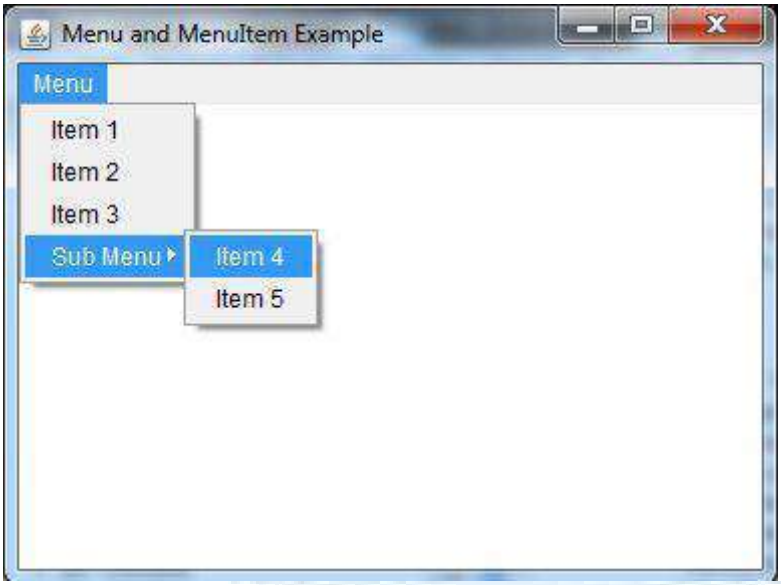
Java AWT MenuItem and Menu Example

```
1.  import java.awt.*;
2.  class MenuExample
3.  {
4.      MenuExample(){
5.          Frame f= new Frame("Menu and M
    enuItem Example");
6.          MenuBar mb=new MenuBar();
7.          Menu menu=new Menu("Menu");
```

```
8.          Menu submenu=new Menu("Sub M
  enu");
9.          MenuItem i1=new MenuItem("Ite
  m 1");
10.         MenuItem i2=new MenuItem("Ite
  m 2");
11.         MenuItem i3=new MenuItem("Ite
  m 3");
12.         MenuItem i4=new MenuItem("Ite
  m 4");
13.         MenuItem i5=new MenuItem("Ite
  m 5");
14.         menu.add(i1);
15.         menu.add(i2);
16.         menu.add(i3);
17.         submenu.add(i4);
18.         submenu.add(i5);
19.         menu.add(submenu);
20.         mb.add(menu);
```

```
21.             f.setMenuBar(mb);
22.             f.setSize(400,400);
23.             f.setLayout(null);
24.             f.setVisible(true);
25.     }
26.     public static void main(String args[])
27.     {
28.     new MenuExample();
29.     }
30.     }
```

Output:



Java AWT PopupMenu

PopupMenu can be dynamically popped up at specific position within a component. It inherits the Menu class.

AWT PopupMenu class declaration

```
1. public class PopupMenu
   extends Menu implemen
   ts MenuContainer, Accessi
   ble
```

Java AWT PopupMenu Example

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

```
3.  class PopupMenuExampl
  e
4.  {
5.       PopupMenuExample()
  {
6.            final Frame f= ne
  w Frame("PopupMenu Exa
  mple");
7.            final PopupMenu p
  opupmenu = new PopupM
  enu("Edit");
```

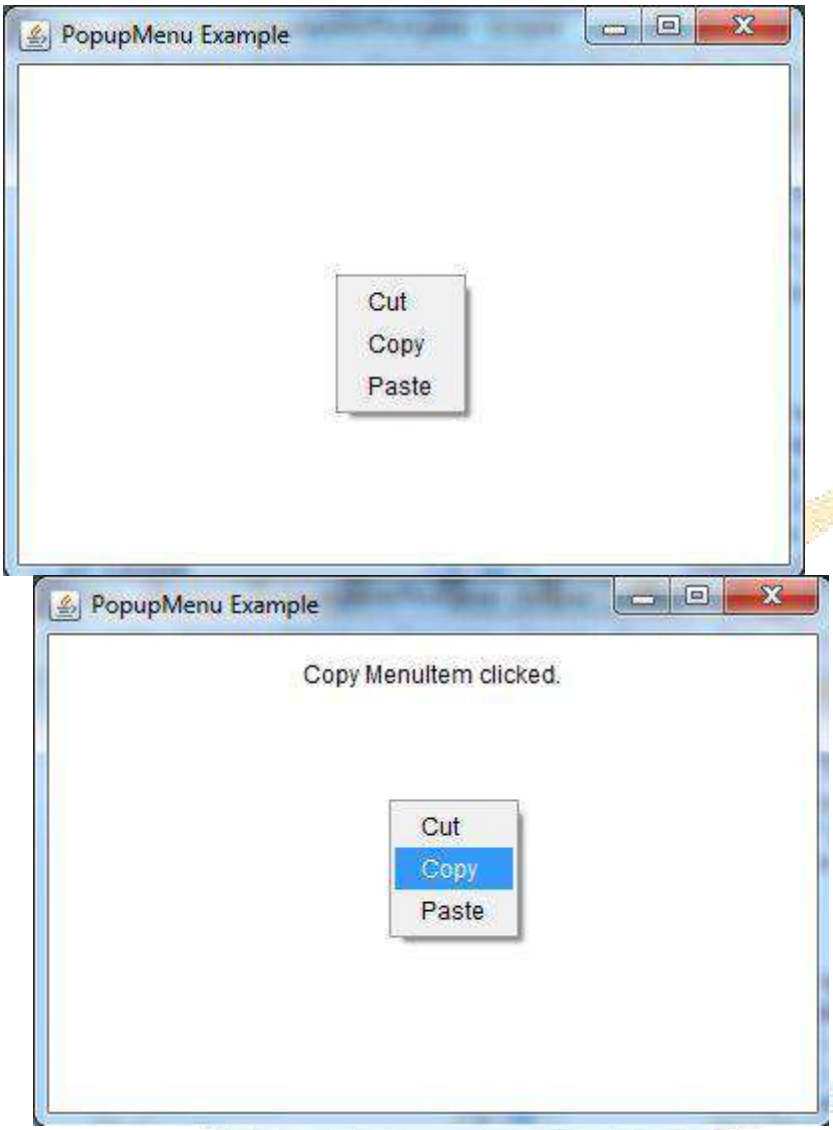
```
8.          MenuItem cut = n
 ew MenuItem("Cut");
9.          cut.setActionCom
 mand("Cut");
10.         MenuItem copy =
 new MenuItem("Copy");
11.         copy.setActionCom
 mand("Copy");
12.         MenuItem paste =
  new MenuItem("Paste");
13.         paste.setActionCo
 mmand("Paste");
```

```
14.         popupmenu.add(c
 ut);
15.         popupmenu.add(c
 opy);
16.         popupmenu.add(p
 aste);
17.         f.addMouseListene
 r(new MouseAdapter() {
18.             public void mo
 useClicked(MouseEvent e)
 {
```

```
19.                 popupmenu.s
 how(f , e.getX(), e.getY());

20.                   }
21.             });
22.             f.add(popupmenu)
  ;
23.             f.setSize(400,400)
  ;
24.             f.setLayout(null);
```

```
25.         f.setVisible(true);


26.     }
27.public static void main
 (String args[])
28.{
29.         new PopupMenuEx
 ample();
30.}
31.}
```

Output:



Java AWT Scrollbar

The object of Scrollbar class is used to add horizontal and vertical scrollbar. Scrollbar is a GUI component allows us to see invisible number of rows and columns.

AWT Scrollbar class declaration

```
1. public class Scrollbar extends Component implements Adjustable, Accessible
```

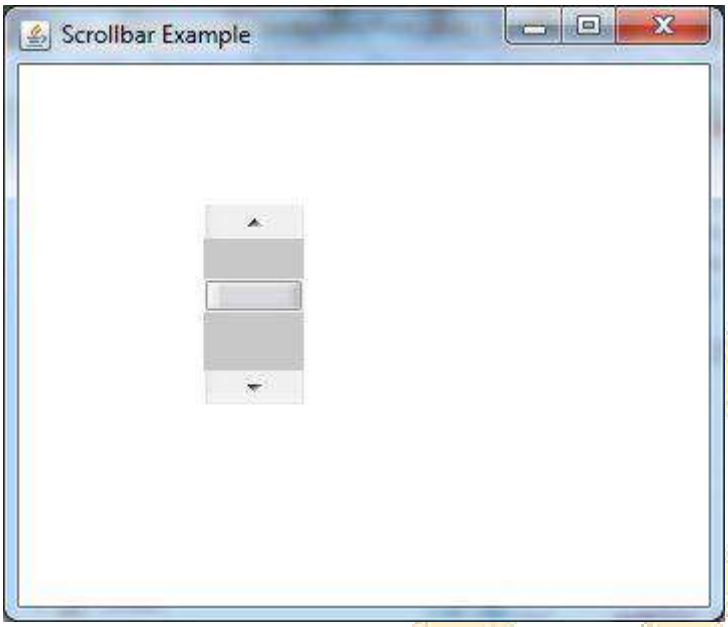
Java AWT Scrollbar Example

```
1. import java.awt.*;
2. class ScrollbarExample{
```

```
3.  ScrollbarExample(){
4.              Frame f= new F
 rame("Scrollbar Example")
 ;
5.              Scrollbar s=new
  Scrollbar();
6.              s.setBounds(100
 ,100, 50,100);
7.              f.add(s);
8.              f.setSize(400,40
 0);
```

```
9.              f.setLayout(null
 );
10.             f.setVisible(true
 );
11.}
12.public static void main
 (String args[]){
13.       new ScrollbarExam
 ple();
14.}
15.}
```

Output:



Java AWT Scrollbar Example with AdjustmentListener

```
1. import java.awt.*;
```

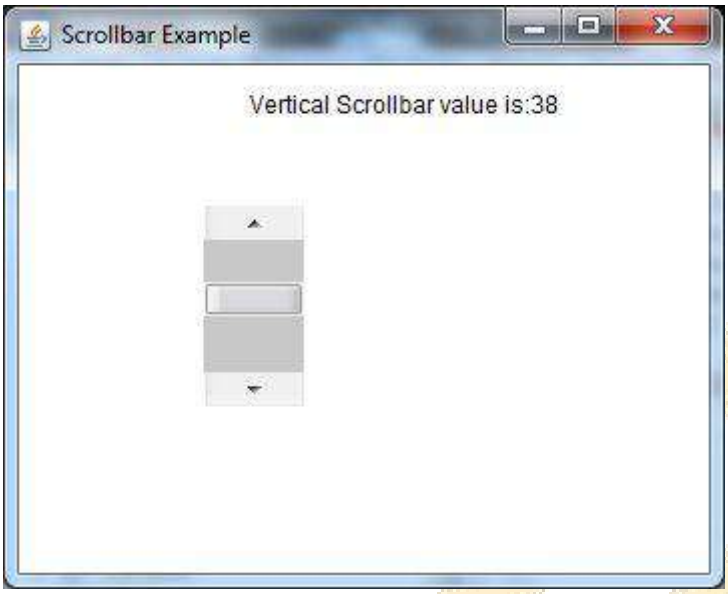
```
2.  import java.awt.event.*
  ;
3.  class ScrollbarExample{

4.      ScrollbarExample(){
5.          Frame f= new F
  rame("Scrollbar Example")
  ;
6.          final Label label
  = new Label();
7.          label.setAlignme
  nt(Label.CENTER);
```

```
8.              label.setSize(40
 0,100);
9.              final Scrollbar s
 =new Scrollbar();
10.             s.setBounds(100
 ,100, 50,100);
11.             f.add(s);f.add(la
 bel);
12.             f.setSize(400,40
 0);
13.             f.setLayout(null
 );
```

```
14.             f.setVisible(true
 );
15.             s.addAdjustment
 Listener(new AdjustmentLi
 stener() {
16.                 public void a
 djustmentValueChanged(A
 djustmentEvent e) {
17.                     label.setTex
 t("Vertical Scrollbar value i
 s:"+ s.getValue());
18.                 }
```

```
19.         });
20.     }
21.public static void main
 (String args[]){
22.new ScrollbarExample()
 ;
23.}
24.}
```

Output:



BorderLayout (LayoutManagers):

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:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout etc.

BorderLayout:

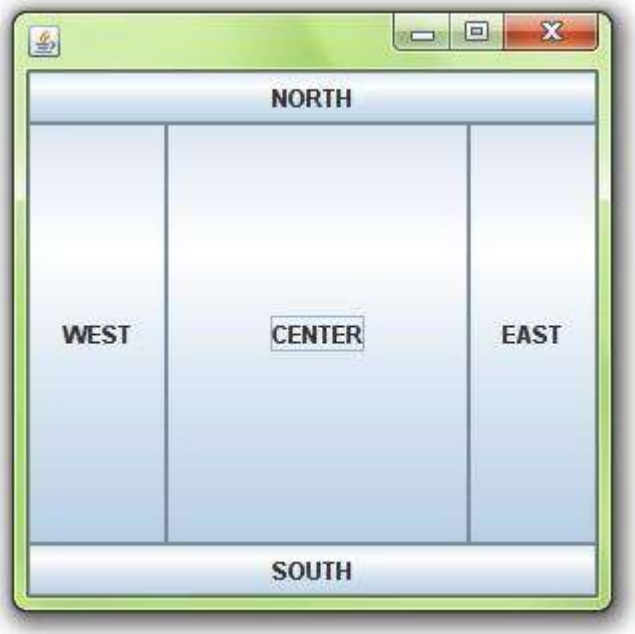
The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

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.

Example of BorderLayout class:




```
1.  import java.awt.*;
2.  import javax.swing.*;
3.
4.  public class Border {
5.  JFrame f;
6.  Border(){
7.      f=new JFrame();
8.
9.      JButton b1=new JButton("NORTH");;
10.     JButton b2=new JButton("SOUTH");;
11.     JButton b3=new JButton("EAST");;
```

```
12.     JButton b4=new JButton("WEST");;
13.     JButton b5=new JButton("CENTER");;
14.
15.     f.add(b1,BorderLayout.NORTH);
16.     f.add(b2,BorderLayout.SOUTH);
17.     f.add(b3,BorderLayout.EAST);
18.     f.add(b4,BorderLayout.WEST);
19.     f.add(b5,BorderLayout.CENTER);
20.
21.     f.setSize(300,300);
22.     f.setVisible(true);
23. }
24. public static void main(String[] args) {
25.     new Border();
26. }
27. }
```

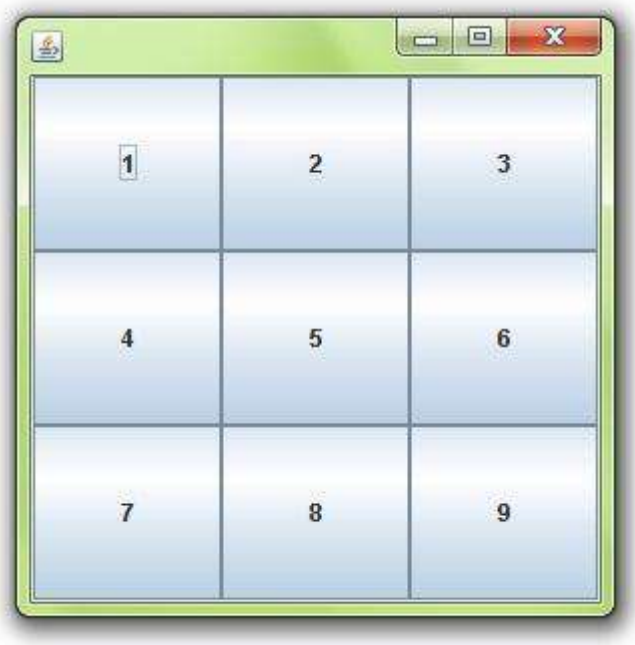
GridLayout

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.

Example of GridLayout class:




```
1.  import java.awt.*;
2.  import javax.swing.*;
3.
4.  public class MyGridLayout{
5.  JFrame f;
6.  MyGridLayout(){
7.      f=new JFrame();
8.
9.      JButton b1=new JButton("1");
10.     JButton b2=new JButton("2");
11.     JButton b3=new JButton("3");
12.     JButton b4=new JButton("4");
13.     JButton b5=new JButton("5");
14.         JButton b6=new JButton("6");
15.         JButton b7=new JButton("7");
16.     JButton b8=new JButton("8");
17.         JButton b9=new JButton("9");
18.
19.     f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
20.     f.add(b6);f.add(b7);f.add(b8);f.add(b9);
21.
22.     f.setLayout(new GridLayout(3,3));
23.     //setting grid layout of 3 rows and 3 columns
24.
25.     f.setSize(300,300);
```

```
26.     f.setVisible(true);
27. }
28. public static void main(String[] args) {
29.     new MyGridLayout();
30. }
31. }
```

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:


```
1.  import java.awt.*;
2.  import javax.swing.*;
3.
4.  public class MyFlowLayout{
5.  JFrame f;
6.  MyFlowLayout(){
7.      f=new JFrame();
8.
9.      JButton b1=new JButton("1");
10.     JButton b2=new JButton("2");
11.     JButton b3=new JButton("3");
12.     JButton b4=new JButton("4");
13.     JButton b5=new JButton("5");
14.
15.     f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
16.
17.     f.setLayout(new FlowLayout(FlowLayout.RIGHT));
18.     //setting flow layout of right alignment
19.
20.     f.setSize(300,300);
21.     f.setVisible(true);
22. }
23. public static void main(String[] args) {
24.     new MyFlowLayout();
25. }
```

```
26. }
```

BoxLayout class:

The BoxLayout is used to arrange the components either vertically or horizontally. For this purpose, BoxLayout provides four constants. They are as follows:

Note: BoxLayout class is found in javax.swing package.

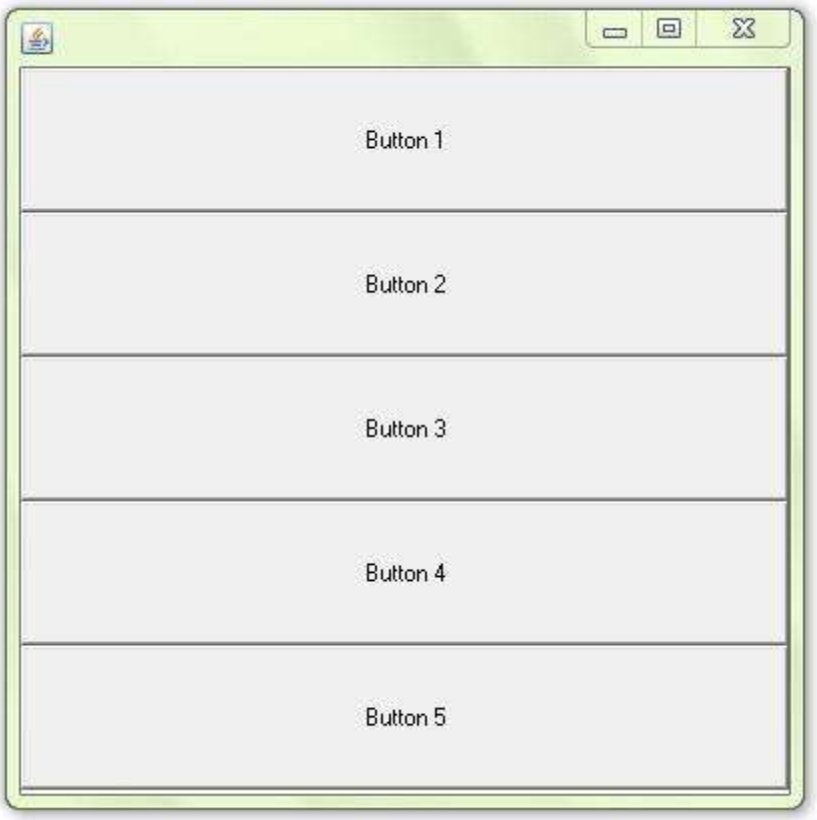
Fields of BoxLayout class:

1. **public static final int X_AXIS**
2. **public static final int Y_AXIS**
3. **public static final int LINE_AXIS**
4. **public static final int PAGE_AXIS**

Constructor of BoxLayout class:

1. **BoxLayout(Container c, int axis):** creates a box layout that arranges the components with the given axis.

Example of BoxLayout class with Y-AXIS:

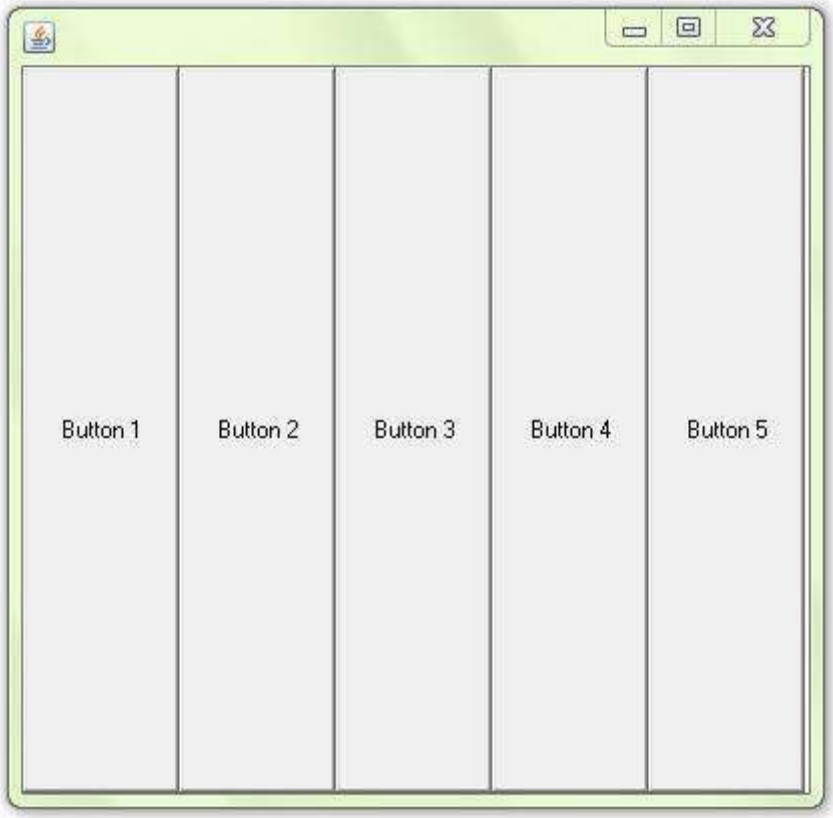



```
1.  import java.awt.*;
2.  import javax.swing.*;
3.
4.  public class BoxLayoutExample1 extends Frame {
5.   Button buttons[];
6.
7.   public BoxLayoutExample1 () {
8.     buttons = new Button [5];
9.
10.    for (int i = 0;i<5;i++) {
11.       buttons[i] = new Button ("Button " + (i + 1));
12.       add (buttons[i]);
13.     }
14.
15. setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));
16. setSize(400,400);
17. setVisible(true);
18. }
19.
20. public static void main(String args[]){
21. BoxLayoutExample1 b=new BoxLayoutExample1();
```

```
22. }
23. }
```

download this example

Example of BoxLayout class with X-AXIS:




```
1.  import java.awt.*;
2.  import javax.swing.*;
3.
4.  public class BoxLayoutExample2 extends Frame {
5.   Button buttons[];
6.
7.   public BoxLayoutExample2() {
8.     buttons = new Button [5];
9.
10.    for (int i = 0;i<5;i++) {
11.      buttons[i] = new Button ("Button " + (i + 1));
12.      add (buttons[i]);
13.     }
14.
15. setLayout (new BoxLayout(this, BoxLayout.X_AXIS));
16. setSize(400,400);
```

```
17. setVisible(true);
18. }
19.
20. public static void main(String args[]){
21. BoxLayoutExample2 b=new BoxLayoutExample2();
22. }
23. }
```

CardLayout class

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.

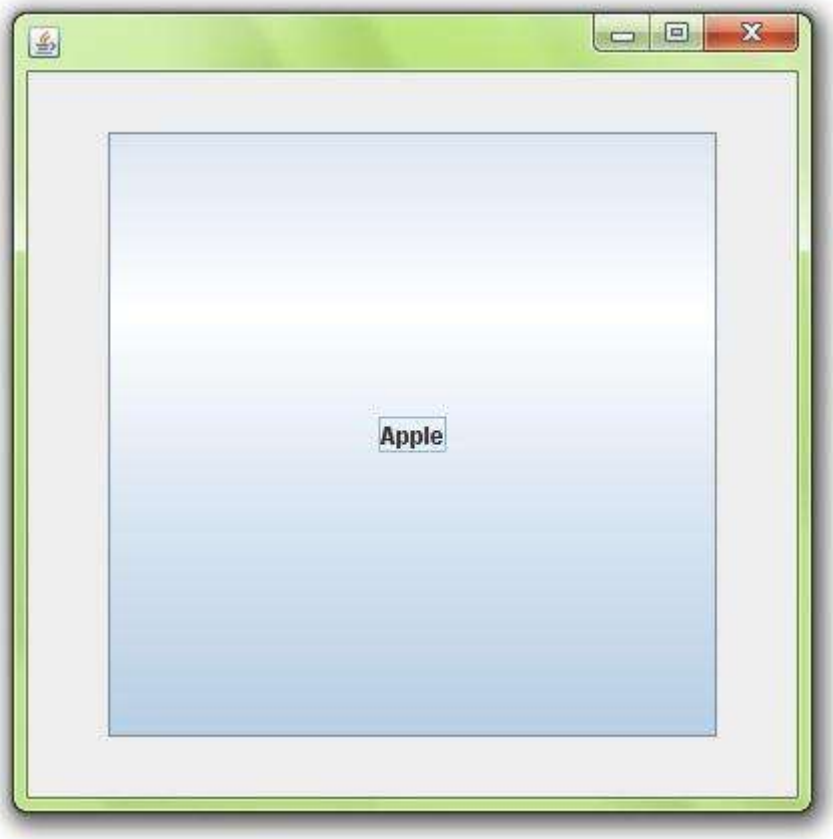
Constructors of CardLayout class:

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.*;
2.  import java.awt.event.*;
3.
4.  import javax.swing.*;
5.
6.  public class CardLayoutExample extends JFrame implements ActionListener{
7.  CardLayout card;
8.  JButton b1,b2,b3;
9.  Container c;
10.     CardLayoutExample(){
11.
12.         c=getContentPane();
13.         card=new CardLayout(40,30);
14. //create CardLayout object with 40 hor space and 30 ver space
15.         c.setLayout(card);
16.
17.         b1=new JButton("Apple");
18.         b2=new JButton("Boy");
19.         b3=new JButton("Cat");
20.         b1.addActionListener(this);
```

```
21.         b2.addActionListener(this);
22.         b3.addActionListener(this);
23.
24.         c.add("a",b1);c.add("b",b2);c.add("c",b3);
25.
26.     }
27.     public void actionPerformed(ActionEvent e) {
28.     card.next(c);
29.     }
30.
31.     public static void main(String[] args) {
32.         CardLayoutExample cl=new CardLayoutExample();
33.         cl.setSize(400,400);
34.         cl.setVisible(true);
35.         cl.setDefaultCloseOperation(EXIT_ON_CLOSE);
36.     }
37. }
```