UNIT-V

S.No	Course Outcome	Intended Learning Outcomes (ILO)	Knowledge Level of ILO
1	CO 5	Explain about applet class	K2
2		Discuss about Applet Lifecycle	K2
		Discuss about AWT ,Components and Containers of AWT	K2
		Illustrate various AWT Controls like Button,label,Checkbox, RadioButton,List box, Menu and Scrollbar with example programs	K3
		Interpret different types of layout managers with examples	K3

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

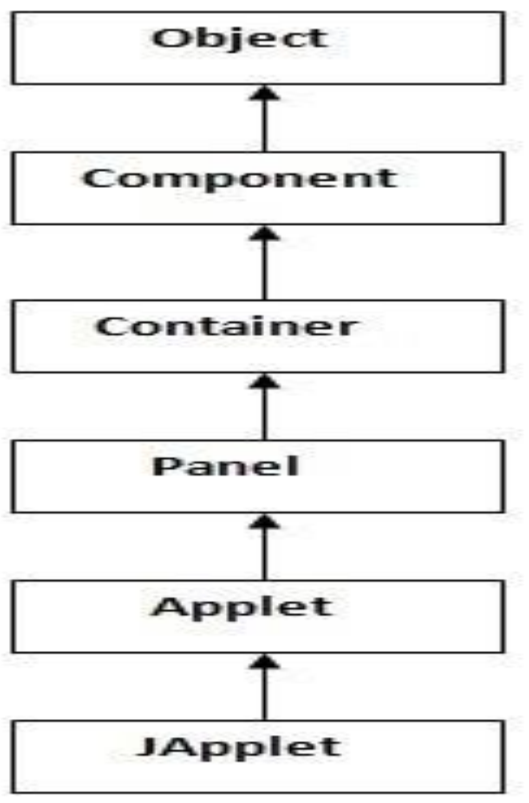
There are many advantages of applet. They are as follows:

- 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.

Hierarchy of Applet



Lifecycle of Java Applet

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

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.

java.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.

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.

<u>myapplet.html</u>

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

<u>Simple example of Applet by appletviewer tool:</u>

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
```

Displaying Graphics in Applet

java.awt.Graphics class provides many methods for graphics programming.

Commonly used methods of Graphics class:

- **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
- **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
- **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.
- **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
- **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.
- **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).
- **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
- **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
- **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.

- **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.

- **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(170,100,30,30);
        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>
```

Displaying Image in Applet

Applet is mostly used in games and animation. For this purpose image is required to be displayed. The java.awt.Graphics class provide a method drawImage() to display the image.

Syntax of drawImage() method:

public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer): is used draw the specified image.

How to get the object of Image:

The java.applet.Applet class provides getImage() method that returns the object of Image. Syntax:

public Image getImage(URL u, String image){}

Other required methods of Applet class to display image:

public URL getDocumentBase(): is used to return the URL of the document in which applet is embedded.

public URL getCodeBase(): is used to return the base URL.

Example of displaying image in applet:

```
import java.awt.*;
import java.applet.*;
public class DisplayImage extends Applet
{
        Image picture;
        public void init()
        {
          picture = getImage(getDocumentBase(),"img1.jpg");
        }

        public void paint(Graphics g)
       {
          g.drawImage(picture, 30,30, this);
        }

  }
```

In the above example, drawImage() method of Graphics class is used to display the image. The 4th argument of drawImage() method of is ImageObserver object. The Component class implements ImageObserver interface. So current class object would also be treated as ImageObserver because Applet class indirectly extends the Component class.

myapplet.html

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

Animation in Applet

Applet is mostly used in games and animation. For this purpose image is required to be moved.

Example of animation in applet:

```
import java.awt.*;
import java.applet.*;
public class AnimationExample extends Applet
{

  Image picture;

  public void init()
   {
      picture =getImage(getDocumentBase(),"bike_1.gif");
    }

  public void paint(Graphics g)
   {
     for(int i=0;i<500;i++)
     {
       g.drawImage(picture, i,30, this);

       try{ Thread.sleep(100); }  catch(Exception e){}
    }
  }
}
```

In the above example, drawImage() method of Graphics class is used to display the image. The 4th argument of drawImage() method of is ImageObserver object. The Component class implements ImageObserver interface. So current class object would also be treated as ImageObserver because Applet class indirectly extends the Component class.

myapplet.html

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

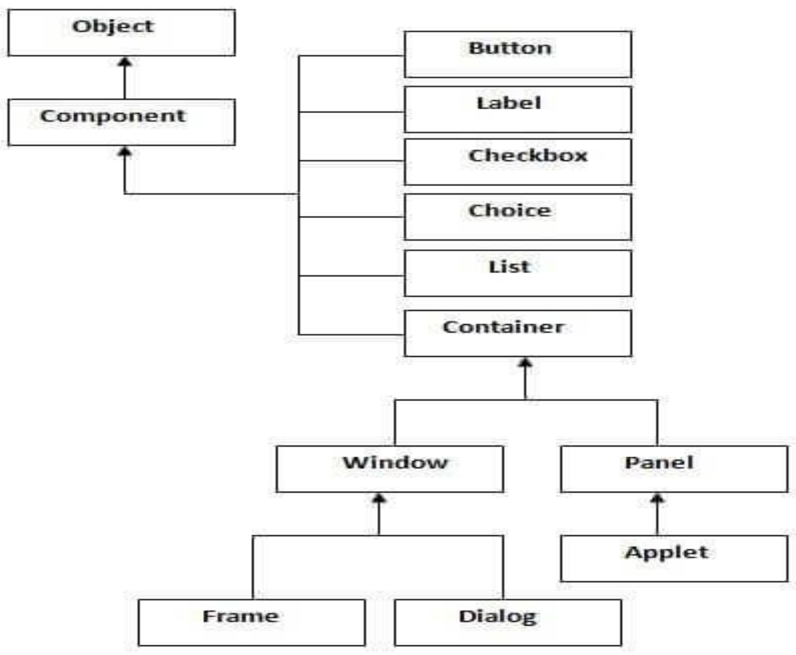
Java AWT

- Java AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java.

- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

- The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT Hierarchy



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

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

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

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

Useful Methods of Component class:

Method	Description
public void add(Component c)	inserts a component on this component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

Java AWT Example
To create simple awt example, you need a frame. There are two ways to create a frame in AWT.

- By extending Frame class (inheritance)
- By creating the object of Frame class (association)

AWT Example by Inheritance

Let's see a simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

```
import java.awt.*;
class First extends Frame
     {
        First()
        {
             Button b=new Button("click me");
             b.setBounds(30,100,80,30);// setting button position
             add(b);//adding button into frame
             setSize(300,300);//frame size 300 width and 300 height
             setLayout(null);//no layout manager
             setVisible(true);//now frame will be visible, by default not visible
        }
        public static void main(String args[])
        {
             First f=new First();
        }
    }
```

The **setBounds(int xaxis, int yaxis, int width, int height)** method is used in the above example that sets the position of the awt button.

AWT Example by Association:

Let's see a simple example of AWT where we are creating instance of Frame class. Here, we are showing Button component on the Frame.

```
import java.awt.*;
class First2
{
   First2()
    {
      Frame f=new Frame();
      Button b=new Button("click me");
      b.setBounds(30,50,80,30);
      f.add(b);
      f.setSize(300,300);
      f.setLayout(null);
      f.setVisible(true);
    }
   public static void main(String args[])
   {  First2 f=new First2();  }}
```

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

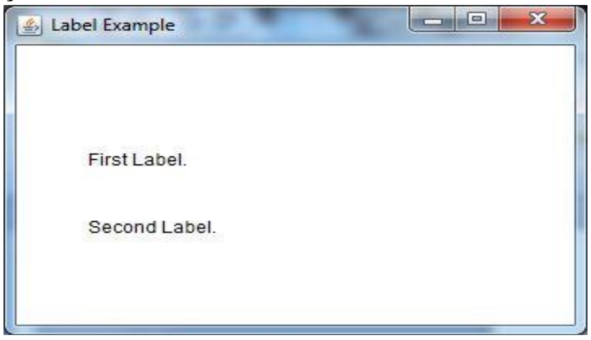
public class Button extends Component

AWT Label Class Declaration

public class Label extends Component

Java Label Example

```
import java.awt.*;
class LabelExample
{
 public static void main(String args[])
 {
    Frame f= new Frame("Label Example");
    Label l1,l2;
    l1=new Label("First Label.");
    l1.setBounds(50,100, 100,30);
    l2=new Label("Second Label.");
    l2.setBounds(50,150, 100,30);
    f.add(l1); f.add(l2);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
 }
}
```



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.

Components of Event Handling

Event handling has three main components,

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

Important Event Classes and Interface

Event Classes	Description	Listener Interface
ActionEvent	generated when button is pressed, menu-item is selected, list-item is double clicked	ActionListener
MouseEvent	generated when mouse is dragged, moved,clicked,pressed or released and also when it enters or exits a component	MouseListener
KeyEvent	generated when input is received from keyboard	KeyListener

Event Classes	**Description**	**Listener Interface**
ItemEvent	generated when check-box or list item is clicked	ItemListener
TextEvent	generated when value of textarea or textfield is changed	TextListener
MouseWheelEvent	generated when mouse wheel is moved	MouseWheelListener
WindowEvent	generated when window is activated, deactivated, deiconified, iconified, opened or closed	WindowListener
ComponentEvent	generated when component is hidden, moved, resized or set visible	ComponentEventListener
ContainerEvent	generated when component is added or removed from container	ContainerListener

Event Classes	Description	Listener Interface
AdjustmentEvent	generated when scroll bar is manipulated	AdjustmentListener
FocusEvent	generated when component gains or loses keyboard focus	FocusListener

<u>Steps to perform Event Handling:</u>

- 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){}

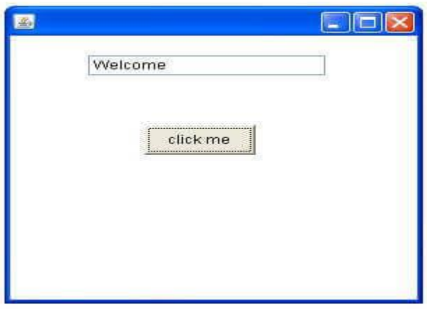
Java Event Handling Code

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

- Within class
- Other class
- 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();
    }
}
```



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

Java WindowListener Interface

The Java WindowListener is notified whenever you change the state of window. It is notified against WindowEvent. The WindowListener interface is found in java.awt.event package. It has three methods.

Methods of WindowListener interface

The signature of 7 methods found in WindowListener interface are given below:

```
public abstract void windowActivated(WindowEvent e);
public abstract void windowClosed(WindowEvent e);
public abstract void windowClosing(WindowEvent e);
public abstract void windowDeactivated(WindowEvent e);
public abstract void windowDeiconified(WindowEvent e);
public abstract void windowIconified(WindowEvent e);
public abstract void windowOpened(WindowEvent e);
```

Java WindowListener Example

```
import java.awt.*;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
public class WindowExample extends Frame implements WindowListener{
   WindowExample(){
      addWindowListener(this);

      setSize(400,400);
      setLayout(null);
      setVisible(true);
   }

public static void main(String[] args) {
   new WindowExample();
}
public void windowActivated(WindowEvent arg0) {
   System.out.println("activated");
}
public void windowClosed(WindowEvent arg0) {
   System.out.println("closed");
}
public void windowClosing(WindowEvent arg0) {
   System.out.println("closing");
   dispose();
}
public void windowDeactivated(WindowEvent arg0) {
   System.out.println("deactivated");
}
public void windowDeiconified(WindowEvent arg0) {
   System.out.println("deiconified");
}
public void windowIconified(WindowEvent arg0) {
   System.out.println("iconified");
}
public void windowOpened(WindowEvent arg0) {
   System.out.println("opened");
}
}
```

Java ActionListener Interface

The Java ActionListener is notified whenever you click on the button or menu item. It is notified against ActionEvent. The ActionListener interface is found in java.awt.event package. It has only one method: actionPerformed().

actionPerformed() method

The actionPerformed() method is invoked automatically whenever you click on the registered component.

```
public abstract void actionPerformed(ActionEvent e);
```

How to write ActionListener

The common approach is to implement the ActionListener. If you implement the ActionListener class, you need to follow 3 steps:

1) Implement the ActionListener interface in the class:

```
    public class ActionListenerExample Implements ActionListener
```

2) Register the component with the Listener:

```
   component.addActionListener(instanceOfListenerclass);
```

3) Override the actionPerformed() method:

```
  public void actionPerformed(ActionEvent e)
  {
       //Write the code here
   }
```

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:

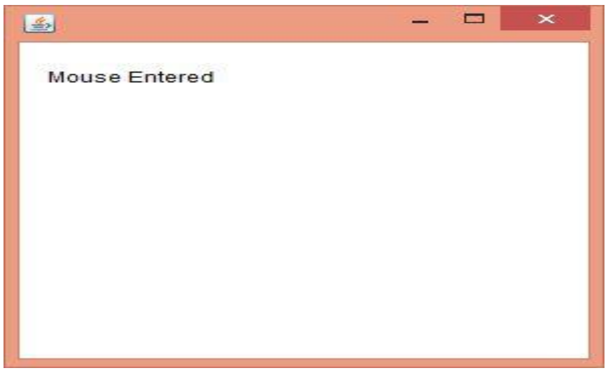
```
public abstract void mouseClicked(MouseEvent e);
public abstract void mouseEntered(MouseEvent e);
public abstract void mouseExited(MouseEvent e);
public abstract void mousePressed(MouseEvent e);
public abstract void mouseReleased(MouseEvent e);
```

Java MouseListener Example1

```
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();
}
}
```



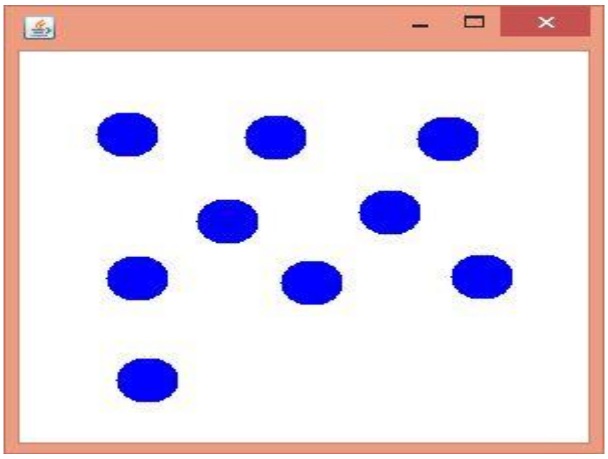
```
Mouse Entered
```

Java MouseListener Example 2

```
import java.awt.*;
import java.awt.event.*;
public class MouseListenerExample2 extends Frame implements MouseListener{
   MouseListenerExample2(){
      addMouseListener(this);

      setSize(300,300);
      setLayout(null);
      setVisible(true);
   }
   public void mouseClicked(MouseEvent e) {
      Graphics g=getGraphics();
      g.setColor(Color.BLUE);
      g.fillOval(e.getX(),e.getY(),30,30);
   }
   public void mouseEntered(MouseEvent e) {}
   public void mouseExited(MouseEvent e) {}
   public void mousePressed(MouseEvent e) {}
   public void mouseReleased(MouseEvent e) {}

public static void main(String[] args) {
   new MouseListenerExample2();
}
}
```



Java MouseMotionListener Interface

The Java MouseMotionListener is notified whenever you move or drag mouse. It is notified against MouseEvent. The MouseMotionListener interface is found in java.awt.event package. It has two methods.

Methods of MouseMotionListener interface

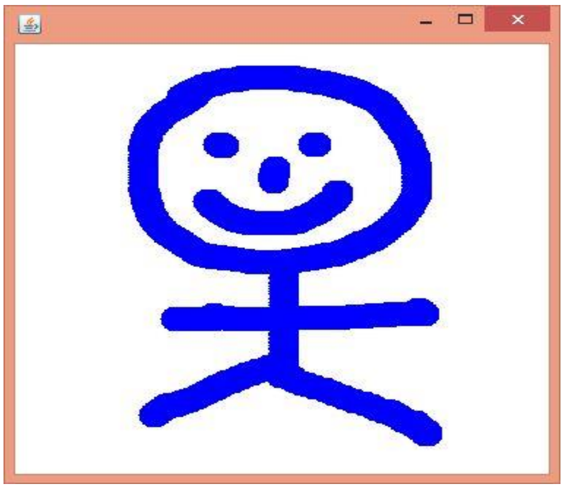
The signature of 2 methods found in MouseMotionListener interface are given below:

```
public abstract void mouseDragged(MouseEvent e);
public abstract void mouseMoved(MouseEvent e);
```

Java MouseMotionListener Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseMotionListenerExample extends Frame implements MouseMotionListener{
   MouseMotionListenerExample(){
      addMouseMotionListener(this);

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



Java KeyListener Interface

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

Methods of KeyListener interface

The signature of 3 methods found in KeyListener interface are given below:

```
public abstract void keyPressed(KeyEvent e);
public abstract void keyReleased(KeyEvent e);
public abstract void keyTyped(KeyEvent e);
```

Java KeyListener Example

```
import java.awt.*;
import java.awt.event.*;
public class KeyListenerExample extends Frame implements KeyListener{
   Label l;
   TextArea area;
   KeyListenerExample(){
        l=new Label();
        l.setBounds(20,50,100,20);
        area=new TextArea();
        area.setBounds(20,80,300, 300);
        area.addKeyListener(this);

        add(l);add(area);
        setSize(400,400);
        setLayout(null);
        setVisible(true);
    }
    public void keyPressed(KeyEvent e) {
        l.setText("Key Pressed");
    }
    public void keyReleased(KeyEvent e) {
        l.setText("Key Released");
    }
    public void keyTyped(KeyEvent e) {
        l.setText("Key Typed");
    }

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

Java Adapter Classes

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

The adapter classes are found in **java.awt.event**, **java.awt.dnd** and **javax.swing.event** packages. The Adapter classes with their corresponding listener interfaces are given below.

java.awt.event Adapter classes

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

Java WindowAdapter Example

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

      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
   }
public static void main(String[] args) {
   new AdapterExample();
}  }
```

Java 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.

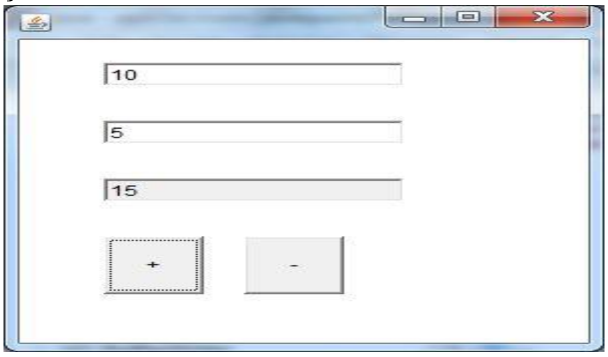
AWT TextField Class Declaration

public class TextField extends TextComponent

Java 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("+");
      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();
 }
}
```

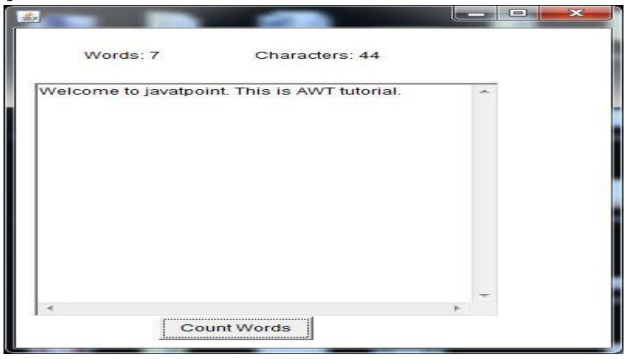


Java AWT TextArea
The object of a TextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits TextComponent class.

AWT TextArea Class Declaration
public class TextArea extends TextComponent

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.
```

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".

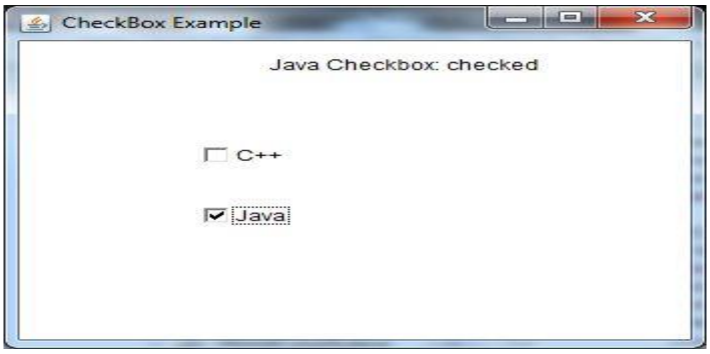
AWT Checkbox Class Declaration

public class Checkbox extends Component

Java AWT Checkbox Example with ItemListener

```
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

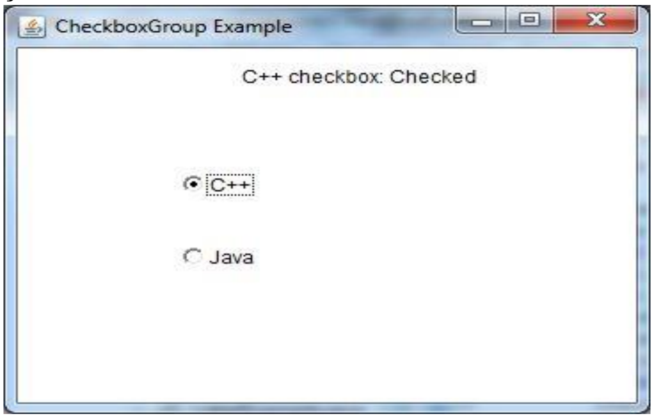
The object of CheckboxGroup class is used to group together a set of Checkbox. At a time only one check box button is allowed to be in "on" state and remaining check box button in "off" state. It inherits the object class.

AWT CheckboxGroup Class Declaration

public class CheckboxGroup extends Object

Java AWT CheckboxGroup Example with ItemListener

```
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();
}
}
```



C++ checkbox: Checked

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.

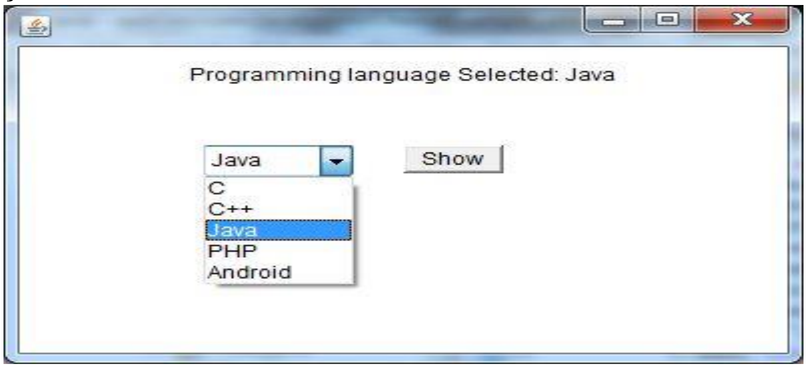
AWT Choice Class Declaration

public class Choice extends Component

Java AWT Choice Example with ActionListener

```
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();
}
}
```



Java AWT List

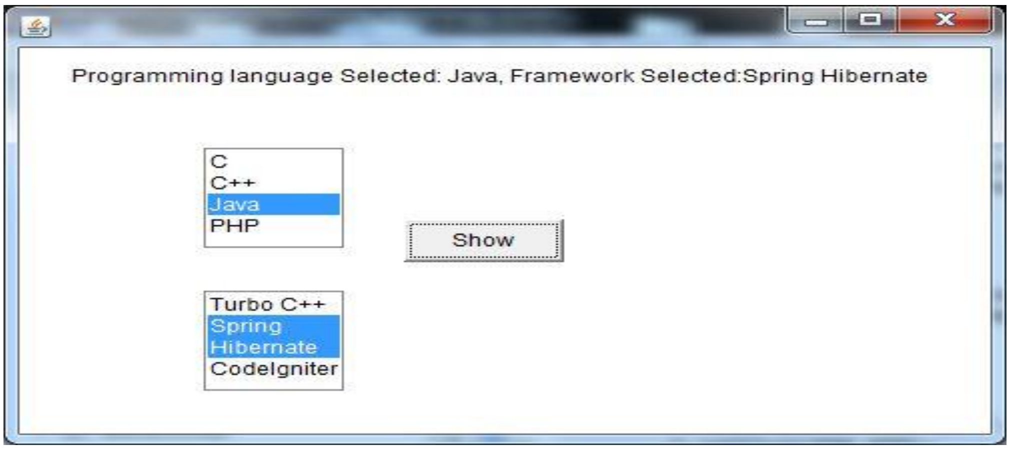
The object of List class represents a list of text items. By the help of list, user can choose either one item or multiple items. It inherits Component class.

AWT List class Declaration

```
public class List extends Component
```

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


Programming language Selected: Java, Framework Selected:Spring Hibernate

Java AWT Scrollbar

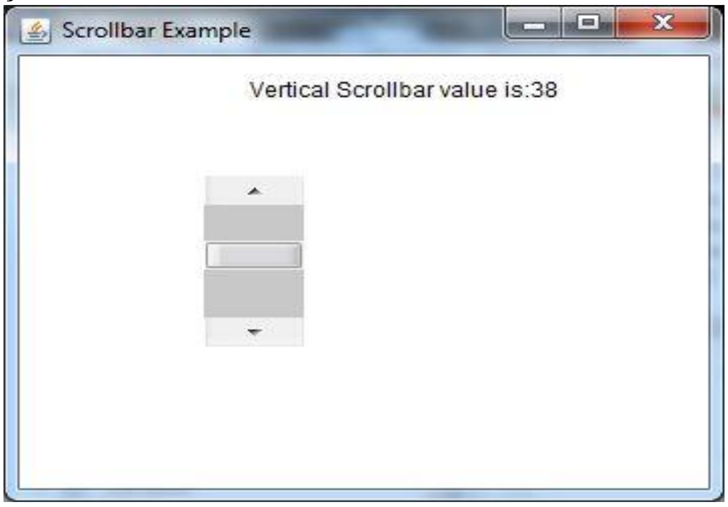
The object of Scrollbar class is used to add horizontal and vertical scrollbar. Scrollbar is a GUI component allows us to see invisible number of rows and columns.

AWT Scrollbar class declaration

public class Scrollbar extends Component

Java AWT Scrollbar Example with AdjustmentListener

```
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();
}
}
```



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

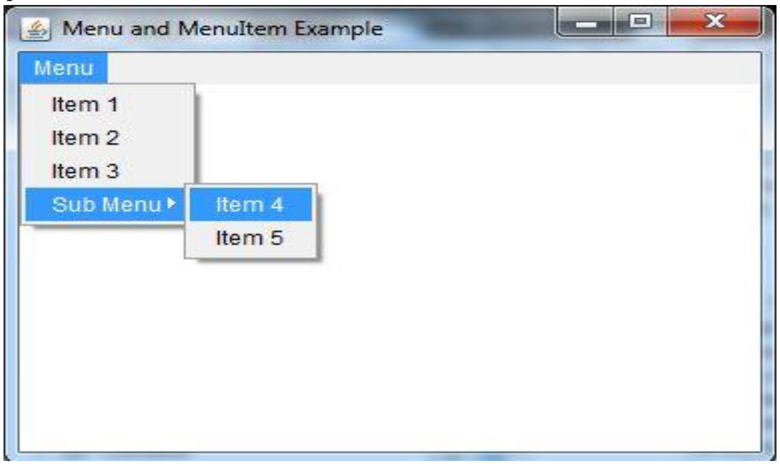
public class MenuItem extends MenuComponent

AWT Menu class declaration

public class Menu extends MenuItem

Java AWT MenuItem and Menu Example

```
import java.awt.*;
class MenuExample
{
    MenuExample(){
        Frame f= new Frame("Menu and MenuItem Example");
        MenuBar mb=new MenuBar();
        Menu menu=new Menu("Menu");
        Menu submenu=new Menu("Sub Menu");
        MenuItem i1=new MenuItem("Item 1");
        MenuItem i2=new MenuItem("Item 2");
        MenuItem i3=new MenuItem("Item 3");
        MenuItem i4=new MenuItem("Item 4");
        MenuItem i5=new MenuItem("Item 5");
        menu.add(i1);
        menu.add(i2);
        menu.add(i3);
        submenu.add(i4);
        submenu.add(i5);
        menu.add(submenu);
        mb.add(menu);
        f.setMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
}
public static void main(String args[])
{
new MenuExample();
}
}
```



Java LayoutManagers

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

- java.awt.BorderLayout
- java.awt.FlowLayout
- java.awt.GridLayout
- java.awt.CardLayout
- java.awt.GridBagLayout
- javax.swing.BoxLayout
- javax.swing.GroupLayout
- javax.swing.ScrollPaneLayout
- javax.swing.SpringLayout etc.

Java 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:

- public static final int NORTH
- public static final int SOUTH
- public static final int EAST
- public static final int WEST
- public static final int CENTER
-

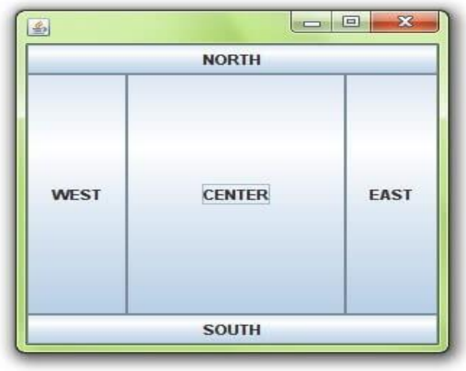
Constructors of BorderLayout class:

- **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.

BorderLayout Example:

```
import java.awt.*;
import java.awt.event.*;
public class Border {
Frame f;
Border()
{
   f=new Frame();
   Button b1=new Button("NORTH");
   Button b2=new Button("SOUTH");
   Button b3=new Button("EAST");
   Button b4=new Button("WEST");
   Button b5=new Button("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);
   f.addWindowListener(new WindowAdapter()
         {
              public void windowClosing(WindowEvent e)
              {
```

```
                        f.dispose();
                    }
                });
}
public static void main(String[] args) {
   new Border();
}
}
```



Java GridLayout

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

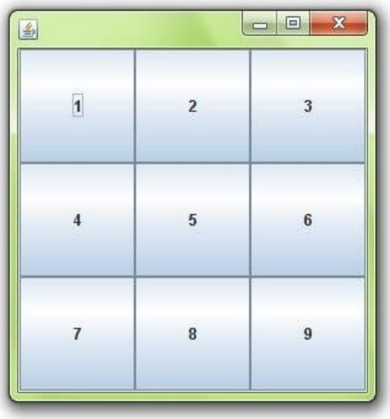
Constructors of GridLayout class:

- GridLayout(): creates a grid layout with one column per component in a row.
- GridLayout(int rows, int columns): creates a grid layout with the given rows and columns but no gaps between the components.
- 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

```
import java.awt.*;
class MyGridLayout
{
 Frame f;
 MyGridLayout()
  {
    f=new Frame();
   Button b1=new Button("1");
   Button b2=new Button("2");
   Button b3=new Button("3");
```

```
    Button b4=new Button("4");
    Button b5=new Button("5");
    Button b6=new Button("6");
    Button b7=new Button("7");
    Button b8=new Button("8");
    Button b9=new Button("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();
}
}
```



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

- public static final int LEFT
- public static final int RIGHT
- public static final int CENTER
- public static final int LEADING
- public static final int TRAILING

Constructors of FlowLayout class

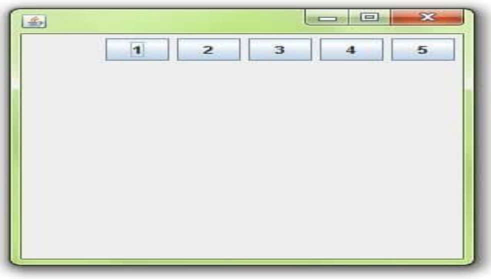
- FlowLayout(): creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
- FlowLayout(int align): creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
- FlowLayout(int align, int hgap, int vgap): creates a flow layout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

Example of FlowLayout class:

```
import java.awt.*;
class MyFlowLayout
{
 Frame f;
 MyFlowLayout()
  {
    f=new Frame();
    Button b1=new Button("1");
    Button b2=new Button("2");
    Button b3=new Button("3");
    Button b4=new Button("4");
    Button b5=new Button("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();
 }
}
```



Java BoxLayout

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

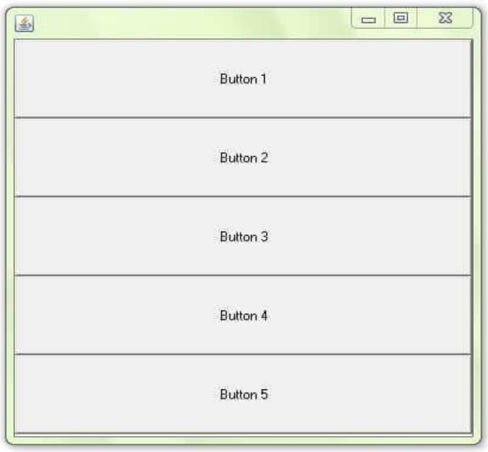
- public static final int X_AXIS
- public static final int Y_AXIS
- public static final int LINE_AXIS
- public static final int PAGE_AXIS

Constructor of BoxLayout class

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

Example of BoxLayout class with Y-AXIS:

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



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

- CardLayout(): creates a card layout with zero horizontal and vertical gap.
- 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

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
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);
    }
}
```