Java AWT Tutorial

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

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

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

Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.

Container

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

Window

The window is the container that have 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.

```
1. import java.awt.*;
2. class First extends Frame{
3. First(){
4. Button b=new Button("click me");
5. b.setBounds(30,100,80,30);// setting button position
6. add(b);//adding button into frame
7. setSize(300,300);//frame size 300 width and 300 height
8. setLayout(null);//no layout manager
9. setVisible(true);//now frame will be visible, by default not visible
10. }
11. public static void main(String args[]){
12. First f=new First();
13. } }
```

The setBounds(int x axis, 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.

```
1. import java.awt.*;
2. class First2{
3. First2(){
4. Frame f=new Frame();
5. Button b=new Button("click me");
6. b.setBounds(30,50,80,30);
7. f.add(b);
8. f.setSize(300,300);
9. f.setLayout(null);
10. f.setVisible(true);
11. }
12. public static void main(String args[]){
13. First2 f=new First2();
14. }}
```

AWT Controls:

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 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 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.

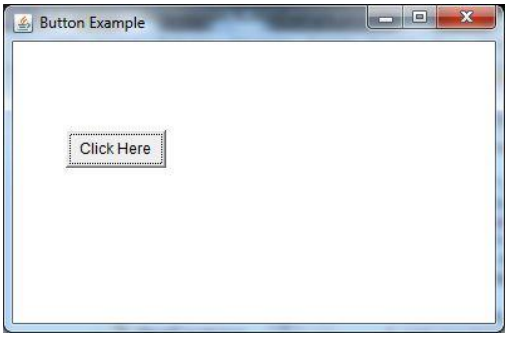
AWT Button Class declaration

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 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.

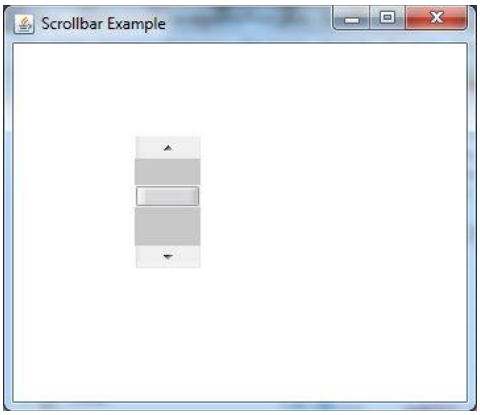
AWT Scrollbar class declaration

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 Frame("Scrollbar Example");
5.         Scrollbar s=new Scrollbar();
6.         s.setBounds(100,100, 50,100);
7.         f.add(s);
8.         f.setSize(400,400);
9.         f.setLayout(null);
10.        f.setVisible(true);
11. }
12. public static void main(String args[]){
13.     new ScrollbarExample();
14. }
15. }
```

Output:



Text Components:

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

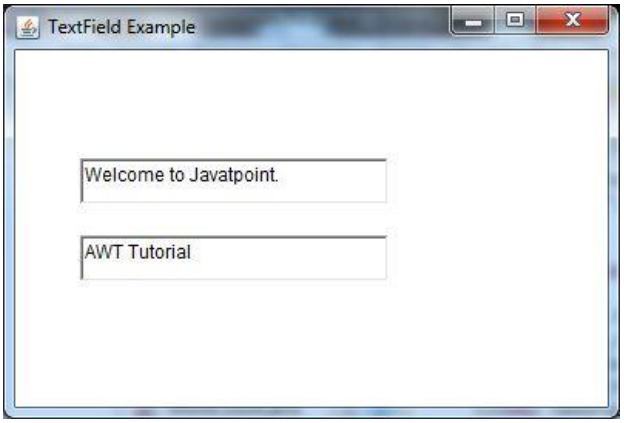
1. **public class** TextField **extends** TextComponent

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 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

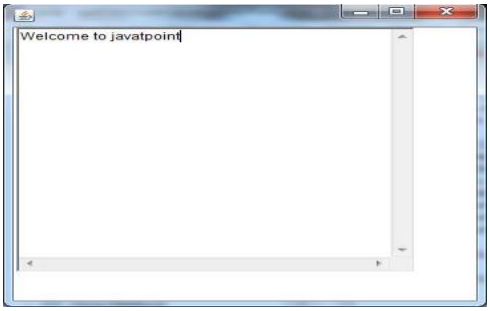
```
1. public class TextArea extends TextComponent
```

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 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

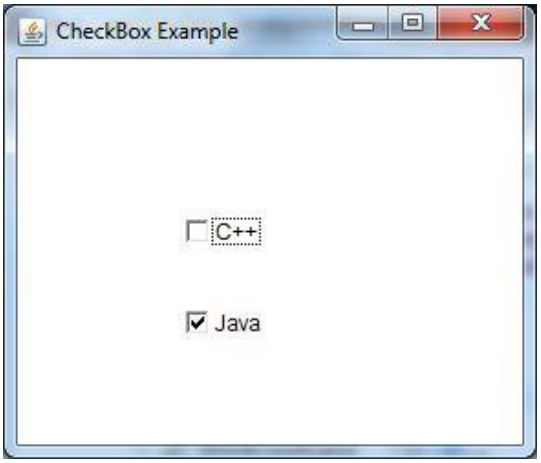
```
1. public class Checkbox extends Component implements ItemSelectable, Accessible
```

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 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

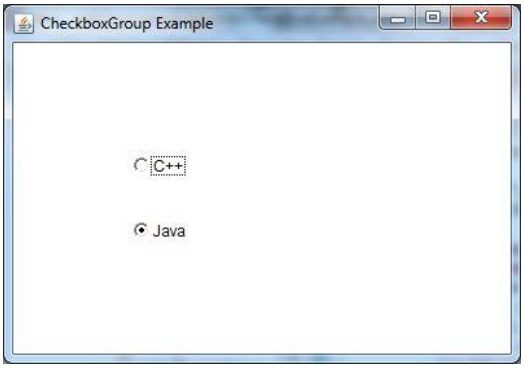
1. **public class** CheckboxGroup **extends** Object **implements** Serializable

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 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

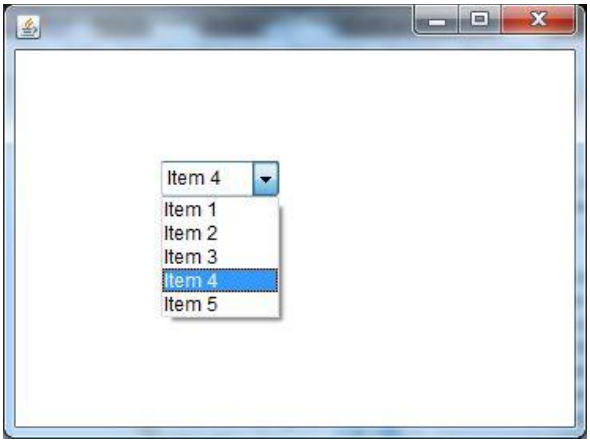
```
1. public class Choice extends Component implements ItemSelectable, Accessible
```

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 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.

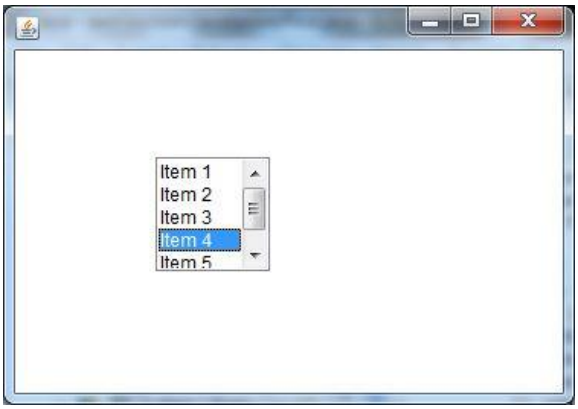
AWT List class Declaration

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 Dialog

The Dialog control represents a top level window with a border and a title used to take some form of input from the user. It inherits the Window class.

Unlike Frame, it doesn't have maximize and minimize buttons.

Frame vs Dialog

Frame and Dialog both inherits Window class. Frame has maximize and minimize buttons but Dialog doesn't have.

AWT Dialog class declaration

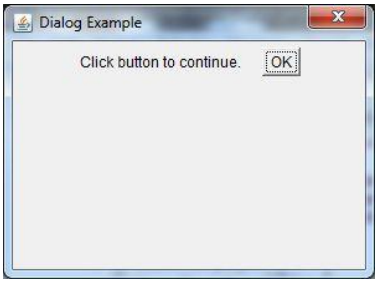
1. **public class** Dialog **extends** Window

Java AWT Dialog Example

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class DialogExample {
4.     private static Dialog d;
5.     DialogExample() {
6.         Frame f= new Frame();
7.         d = new Dialog(f , "Dialog Example", true);
8.         d.setLayout( new FlowLayout() );
9.         Button b = new Button ("OK");
10.        b.addActionListener ( new ActionListener()
11.        {
12.            public void actionPerformed( ActionEvent e )
13.            {
14.                DialogExample.d.setVisible(false);
15.            }
16.        });
```

```
17.         d.add( new Label ("Click button to continue."));
18.         d.add(b);
19.         d.setSize(300,300);
20.         d.setVisible(true);
21.     }
22.     public static void main(String args[])
23.     {
24.         new DialogExample();
25.     }
26. }
```

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 MenuComponent implements Accessible
```

AWT Menu class declaration

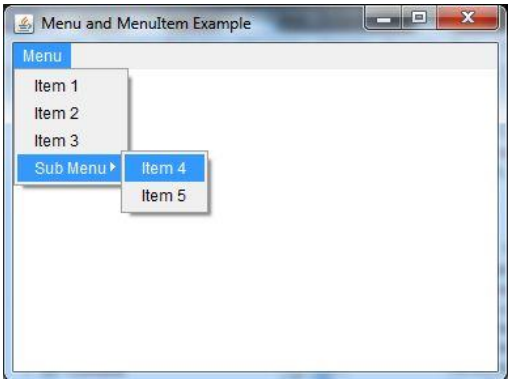
```
1. public class Menu extends MenuItem implements MenuContainer, Accessible
```

Java AWT MenuItem and Menu Example

```
1. import java.awt.*;
2. class MenuExample
3. {
4.     MenuExample(){
```

```
5.          Frame f= new Frame("Menu and MenuItem Example");
6.          MenuBar mb=new MenuBar();
7.          Menu menu=new Menu("Menu");
8.          Menu submenu=new Menu("Sub Menu");
9.          MenuItem i1=new MenuItem("Item 1");
10.         MenuItem i2=new MenuItem("Item 2");
11.         MenuItem i3=new MenuItem("Item 3");
12.         MenuItem i4=new MenuItem("Item 4");
13.         MenuItem i5=new MenuItem("Item 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:



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 by implementing ActionListener

```
1. import java.awt.*;
2. import java.awt.event.*;
3. class AEvent extends Frame implements ActionListener{
4. TextField tf;
5. AEvent()
6. {
```

```
7.  //create components
8.  tf=new TextField();
9.  tf.setBounds(60,50,170,20);
10. Button b=new Button("click me");
11. b.setBounds(100,120,80,30);
12.
13. //register listener
14. b.addActionListener(this);//passing current instance
15.
16. //add components and set size, layout and visibility
17. add(b);add(tf);
18. setSize(300,300);
19. setLayout(null);
20. setVisible(true);
21. }
22. public void actionPerformed(ActionEvent e){
23. tf.setText("Welcome");
24. }
25. public static void main(String args[]){
26. new AEvent();
27. }
28. }
```

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.

Output:

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:

1. **public abstract void** mouseClicked(MouseEvent e);
2. **public abstract void** mouseEntered(MouseEvent e);
3. **public abstract void** mouseExited(MouseEvent e);
4. **public abstract void** mousePressed(MouseEvent e);
5. **public abstract void** mouseReleased(MouseEvent e);

Java MouseListener

Example 1

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class MouseListenerExample extends Frame implements MouseListener{
4.     Label l;
```

```
5.    MouseListenerExample(){
6.        addMouseListener(this);
7.
8.        l=new Label();
9.        l.setBounds(20,50,100,20);
10.       add(l);
11.       setSize(300,300);
12.       setLayout(null);
13.       setVisible(true);
14.   }
15.   public void mouseClicked(MouseEvent e) {
16.       l.setText("Mouse Clicked");
17.   }
18.   public void mouseEntered(MouseEvent e) {
19.       l.setText("Mouse Entered");
20.   }
21.   public void mouseExited(MouseEvent e) {
22.       l.setText("Mouse Exited");
23.   }
24.   public void mousePressed(MouseEvent e) {
25.       l.setText("Mouse Pressed");
26.   }
27.   public void mouseReleased(MouseEvent e) {
28.       l.setText("Mouse Released");
29.   }
```

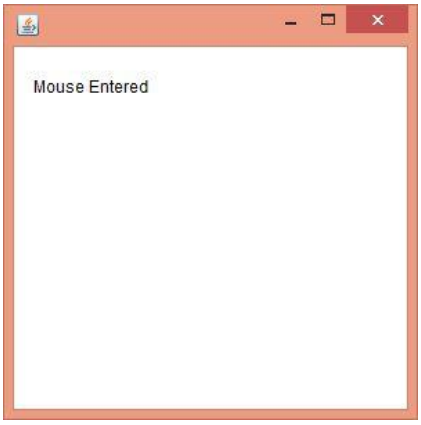
30. **public static void** main(String[] args) {

31. **new** MouseListenerExample();

32. }

33. }

Output:



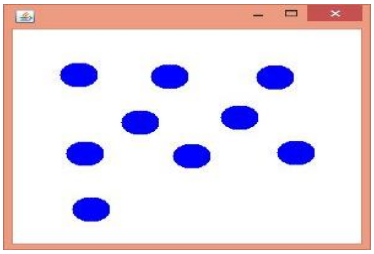
Java MouseListener

Example 2

1. **import** java.awt.*;
2. **import** java.awt.event.*;
3. **public class** MouseListenerExample2 **extends** Frame **implements** MouseListener{
4. MouseListenerExample2(){
5. addMouseListener(**this**);
6.
7. setSize(300,300);
8. setLayout(**null**);
9. setVisible(**true**);
10. }
11. **public void** mouseClicked(MouseEvent e) {

```
12.         Graphics g=getGraphics();
13.         g.setColor(Color.BLUE);
14.         g.fillOval(e.getX(),e.getY(),30,30);
15.     }
16.     public void mouseEntered(MouseEvent e) {}
17.     public void mouseExited(MouseEvent e) {}
18.     public void mousePressed(MouseEvent e) {}
19.     public void mouseReleased(MouseEvent e) {}
20.
21. public static void main(String[] args) {
22.     new MouseListenerExample2();
23. }
24. }
```

Output:



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:

1. **public abstract void** keyPressed(KeyEvent e);
2. **public abstract void** keyReleased(KeyEvent e);

3. **public abstract void** keyTyped(KeyEvent e);

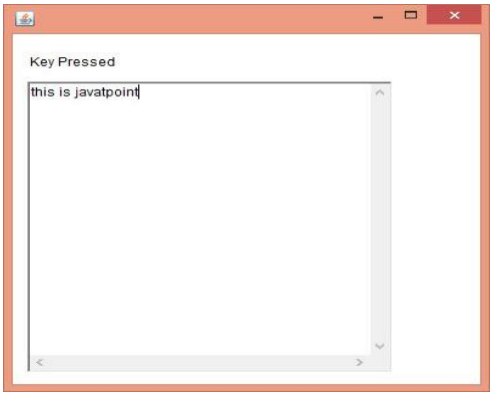
Java KeyListener

Example 1

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class KeyListenerExample extends Frame implements KeyListener{
4.     Label l;
5.     TextArea area;
6.     KeyListenerExample(){
7.
8.         l=new Label();
9.         l.setBounds(20,50,100,20);
10.        area=new TextArea();
11.        area.setBounds(20,80,300, 300);
12.        area.addKeyListener(this);
13.
14.        add(l);add(area);
15.        setSize(400,400);
16.        setLayout(null);
17.        setVisible(true);
18.    }
19.    public void keyPressed(KeyEvent e) {
20.        l.setText("Key Pressed");
21.    }
22.    public void keyReleased(KeyEvent e) {
```

```
23.         l.setText("Key Released");
24.     }
25.     public void keyTyped(KeyEvent e) {
26.         l.setText("Key Typed");
27.     }
28.
29.     public static void main(String[] args) {
30.         new KeyListenerExample();
31.     }
32. }
```

Output:



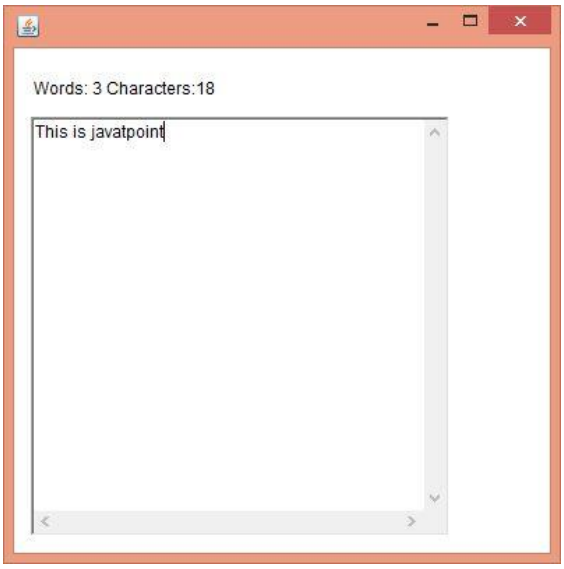
Java KeyListener

Example 2: Count Words & Characters

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class KeyListenerExample extends Frame implements KeyListener{
4.     Label l;
5.     TextArea area;
```

```
6.      KeyListenerExample(){
7.
8.          l=new Label();
9.          l.setBounds(20,50,200,20);
10.         area=new TextArea();
11.         area.setBounds(20,80,300, 300);
12.         area.addKeyListener(this);
13.
14.         add(l);add(area);
15.         setSize(400,400);
16.         setLayout(null);
17.         setVisible(true);
18.     }
19.     public void keyPressed(KeyEvent e) {}
20.     public void keyReleased(KeyEvent e) {
21.         String text=area.getText();
22.         String words[]=text.split("\\s");
23.         l.setText("Words: "+words.length+" Characters:"+text.length());
24.     }
25.     public void keyTyped(KeyEvent e) {}
26.
27.     public static void main(String[] args) {
28.         new KeyListenerExample();
29.     }
30. }
```

Output:



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** package. 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
HierarchyBoundsAdapter	HierarchyBoundsListener

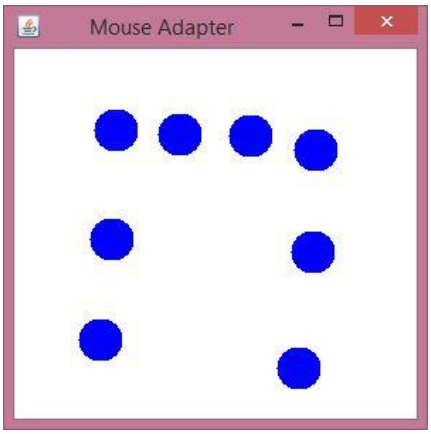
java.awt.dnd Adapter classes

Adapter class	Listener interface
DragSourceAdapter	DragSourceListener
DragTargetAdapter	DragTargetListener

Java MouseAdapter Example

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class MouseAdapterExample extends MouseAdapter{
4.     Frame f;
5.     MouseAdapterExample(){
6.         f=new Frame("Mouse Adapter");
7.         f.addMouseListener(this);
8.
9.         f.setSize(300,300);
10.        f.setLayout(null);
11.        f.setVisible(true);
12.    }
13.    public void mouseClicked(MouseEvent e) {
14.        Graphics g=f.getGraphics();
15.        g.setColor(Color.BLUE);
16.        g.fillOval(e.getX(),e.getY(),30,30);
17.    }
18.
19. public static void main(String[] args) {
20.    new MouseAdapterExample();
21. }
22. }
```

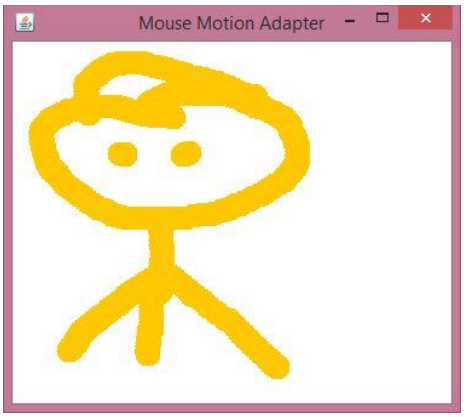
Output:



Java MouseMotionAdapter Example

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class MouseMotionAdapterExample extends MouseMotionAdapter{
4.     Frame f;
5.     MouseMotionAdapterExample(){
6.         f=new Frame("Mouse Motion Adapter");
7.         f.addMouseMotionListener(this);
8.
9.         f.setSize(300,300);
10.        f.setLayout(null);
11.        f.setVisible(true);
12.    }
13. public void mouseDragged(MouseEvent e) {
14.    Graphics g=f.getGraphics();
15.    g.setColor(Color.ORANGE);
16.    g.fillOval(e.getX(),e.getY(),20,20);
17. }
18. public static void main(String[] args) {
19.    new MouseMotionAdapterExample();
20. }
21. }
```

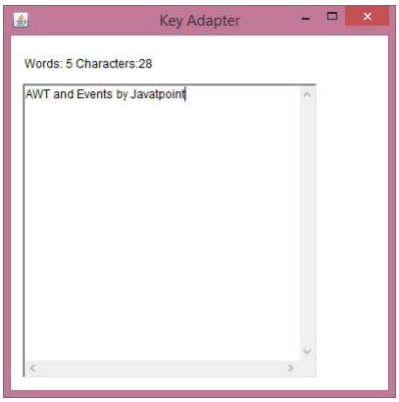
Output:



Java KeyAdapter Example

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.  public class KeyAdapterExample extends KeyAdapter{
4.     Label l;
5.     TextArea area;
6.     Frame f;
7.     KeyAdapterExample(){
8.        f=new Frame("Key Adapter");
9.        l=new Label();
10.       l.setBounds(20,50,200,20);
11.       area=new TextArea();
12.       area.setBounds(20,80,300, 300);
13.       area.addKeyListener(this);
14.
15.       f.add(l);f.add(area);
16.       f.setSize(400,400);
17.       f.setLayout(null);
18.       f.setVisible(true);
19.    }
20.    public void keyReleased(KeyEvent e) {
21.       String text=area.getText();
22.       String words[]=text.split("\\s");
23.       l.setText("Words: "+words.length+" Characters:"+text.length());
24.    }
25.
26.    public static void main(String[] args) {
27.       new KeyAdapterExample();
28.    }
29. }
```

Output:



UNIT V

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:

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.

Java BorderLayout

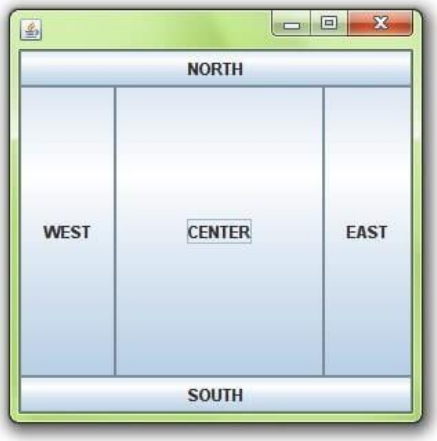
The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

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. }
```

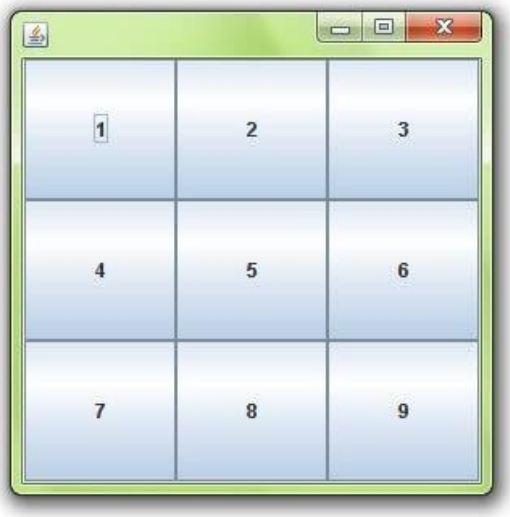
Java GridLayout

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. }
```

Java FlowLayout

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

Fields of FlowLayout class

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

Constructors of FlowLayout class

1. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.

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

Example of FlowLayout class


```
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. }
```

Java CardLayout

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

Constructors of CardLayout class

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

Commonly used methods of CardLayout class

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

Example of CardLayout class


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

Java GridBagLayout

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

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

Fields

Modifier and Type	Field	Description
double[]	columnWeights	It is used to hold the overrides to the column weights.
int[]	columnWidths	It is used to hold the overrides to the column minimum width.
protected Hashtable<Component,GridBagConstraints>	comptable	It is used to maintains the association between a component and its gridbag constraints.
protected GridBagConstraints	defaultConstraints	It is used to hold a gridbag constraints instance containing the default values.
protected GridBagLayoutInfo	layoutInfo	It is used to hold the layout information for the gridbag.
protected static int	MAXGRIDSIZE	No longer in use just for backward compatibility
protected static int	MINSIZE	It is smallest grid that can be laid out by the grid bag layout.
protected static int	PREFERREDSIZE	It is preferred grid size that can be laid out by the grid bag layout.
int[]	rowHeights	It is used to hold the overrides to the row minimum heights.
double[]	rowWeights	It is used to hold the overrides to the row weights.

Useful Methods

Modifier and Type	Method	Description
Void	addLayoutComponent(Component comp, Object constraints)	It adds specified component to the layout, using the specified constraints object.
Void	addLayoutComponent(String name, Component comp)	It has no effect, since this layout manager does not use a per-component string.
protected void	adjustForGravity(GridBagConstraints constraints, Rectangle r)	It adjusts the x, y, width, and height fields to the correct values depending on the constraint geometry and pads.
protected void	AdjustForGravity(GridBagConstraints constraints, Rectangle r)	This method is for backwards compatibility only
protected void	arrangeGrid(Container parent)	Lays out the grid.
protected void	ArrangeGrid(Container parent)	This method is obsolete and supplied for backwards compatibility
GridBagConstraints	getConstraints(Component comp)	It is for getting the constraints for the specified component.
Float	getLayoutAlignmentX(Container parent)	It returns the alignment along the x axis.
Float	getLayoutAlignmentY(Container parent)	It returns the alignment along the y axis.
int[][]	getLayoutDimensions()	It determines column widths and row heights for the layout grid.
protected GridBagLayoutInfo	getLayoutInfo(Container parent, int sizeflag)	This method is obsolete and supplied for backwards compatibility.
protected GridBagLayoutInfo	GetLayoutInfo(Container parent, int sizeflag)	This method is obsolete and supplied for backwards compatibility.

Point	getLayoutOrigin()	It determines the origin of the layout area, in the graphics coordinate space of the target container.
double[][]	getLayoutWeights()	It determines the weights of the layout grid's columns and rows.
protected Dimension	getMinSize(Container parent, GridBagLayoutInfo info)	It figures out the minimum size of the master based on the information from getLayoutInfo.
protected Dimension	GetMinSize(Container parent, GridBagLayoutInfo info)	This method is obsolete and supplied for backwards compatibility only

Example

```
1.  import java.awt.Button;
2.  import java.awt.GridBagConstraints;
3.  import java.awt.GridBagLayout;
4.
5.  import javax.swing.*;
6.  public class GridBagLayoutExample extends JFrame{
7.     public static void main(String[] args) {
8.           GridBagLayoutExample a = new GridBagLayoutExample();
9.        }
10.       public GridBagLayoutExample() {
11.    GridBagLayout grid = new GridBagLayout();
12.          GridBagConstraints gbc = new GridBagConstraints();
13.          setLayout(grid);
14.          setTitle("GridBag Layout Example");
15.          GridBagLayout layout = new GridBagLayout();
16.    this.setLayout(layout);
17.    gbc.fill = GridBagConstraints.HORIZONTAL;
18.    gbc.gridx = 0;
19.    gbc.gridy = 0;
20.    this.add(new Button("Button One"), gbc);
21.    gbc.gridx = 1;
22.    gbc.gridy = 0;
23.    this.add(new Button("Button two"), gbc);
24.    gbc.fill = GridBagConstraints.HORIZONTAL;
25.    gbc.ipady = 20;
26.    gbc.gridx = 0;
```

```
27.     gbc.gridy = 1;
28.     this.add(new Button("Button Three"), gbc);
29.     gbc.gridx = 1;
30.     gbc.gridy = 1;
31.     this.add(new Button("Button Four"), gbc);
32.     gbc.gridx = 0;
33.     gbc.gridy = 2;
34.     gbc.fill = GridBagConstraints.HORIZONTAL;
35.     gbc.gridwidth = 2;
36.     this.add(new Button("Button Five"), gbc);
37.             setSize(300, 300);
38.             setPreferredSize(getSize());
39.             setVisible(true);
40.             setDefaultCloseOperation(EXIT_ON_CLOSE);
41.
42.         }
43.
44. }
```

Output:

Applets

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:

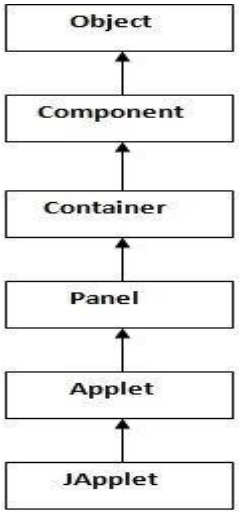
- o It works at client side so less response time.
- o Secured
- o It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

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

Lifecycle of Java Applet Hierarchy of 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 method 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.

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.

```
1. //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);
}
}
```

Simple example of Applet by appletviewer tool:

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

```
1. //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

Difference between Applet and Application programming

	Java Applet	Java Application
User graphics	Inherently graphical	Optional
Memory requirements	Java application requirements plus web browser requirements	Minimal java application requirements
Distribution	Linked via HTML and transported via HTTP	Loaded from the file system or by a custom class loading process
Environmental input	Browser client location and size; parameters embedded in the host HTML document	command-line parameters
Method expected by the virtual Machine	init- initialization method start-startup method stop pause/ deactive method destroy-termination method paint-drawing method	Main - startup method
Typical applications	public-access order-entry systems for the web, online multimedia persentations, web page animation	Network server, multimedia kiosks, developer tools, appliance and consumer electronics control and navigation.

Parameter in Applet

We can get any information from the HTML file as a parameter. For this purpose, Applet class provides a method named getParameter(). Syntax:

1. **public** String getParameter(String parameterName)

Example of using parameter in Applet:

1. **import** java.applet.Applet;
2. **import** java.awt.Graphics;
3. **public class** UseParam **extends** Applet
4. **{**
5. **public void** paint(Graphics g)

```
6. {
7. String str=getParameter("msg");
8. g.drawString(str,50, 50);
9. } }
```

myapplet.html

```
1. <html>
2. <body>
3. <applet code="UseParam.class" width="300" height="300">
4. <param name="msg" value="Welcome to applet">
5. </applet>
6. </body>
7. </html>
```

File name :file.txt Path: file.txt
Absolute path:C:\Users\akki\IdeaProjects\codewriting\src\file.txt Parent:null
Exists :true
Is writeable:true Is readabletrue
Is a directory:false File Size in bytes 20

Connceting to DB

What is JDBCDriver?

JDBC drivers implement the defined interfaces in the JDBC API, for interacting with your database server. For example, using JDBC drivers enable you to open database connections and to interact with it by sending SQL or database commands then receiving results with Java.

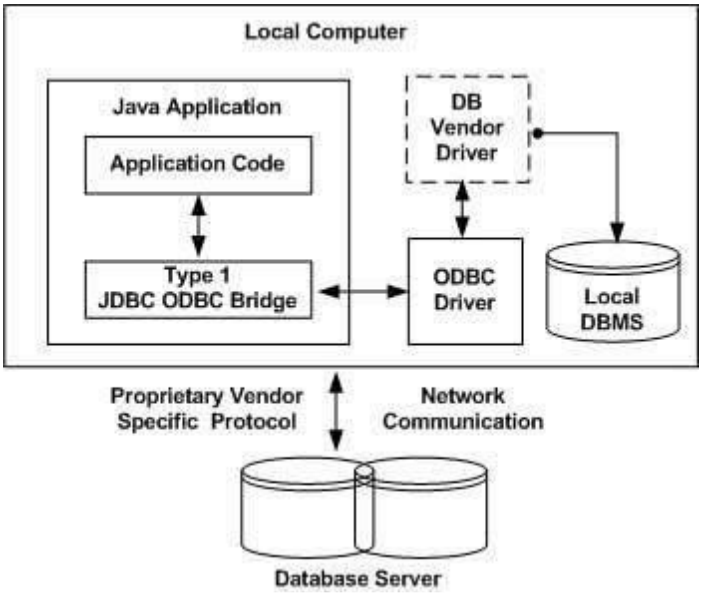
The *Java.sql* package that ships with JDK, contains various classes with their behaviours defined and their actual implementaions are done in third-party drivers. Third party vendors implements the *java.sql.Driver* interface in their database driver.

JDBC Drivers Types

JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates. Sun has divided the implementation types into four categories, Types 1, 2, 3, and 4, which is explained below

Type 1: JDBC-ODBCBridge Driver

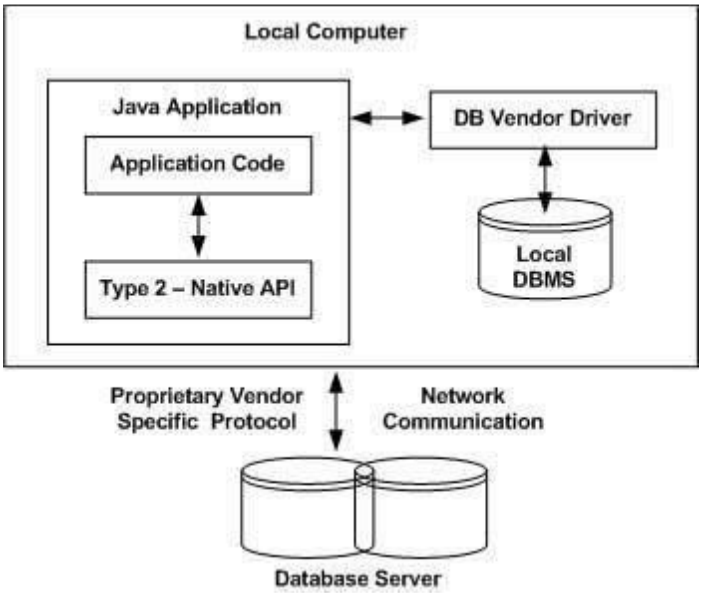
In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine. Using ODBC, requires configuring on your system a Data Source Name (DSN) that represents the target database.
When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.



The JDBC-ODBC Bridge that comes with JDK 1.2 is a good example of this kind of driver.

Type 2: JDBC-Native API

In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls, which are unique to the database. These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge. The vendor-specific driver must be installed on each client machine.

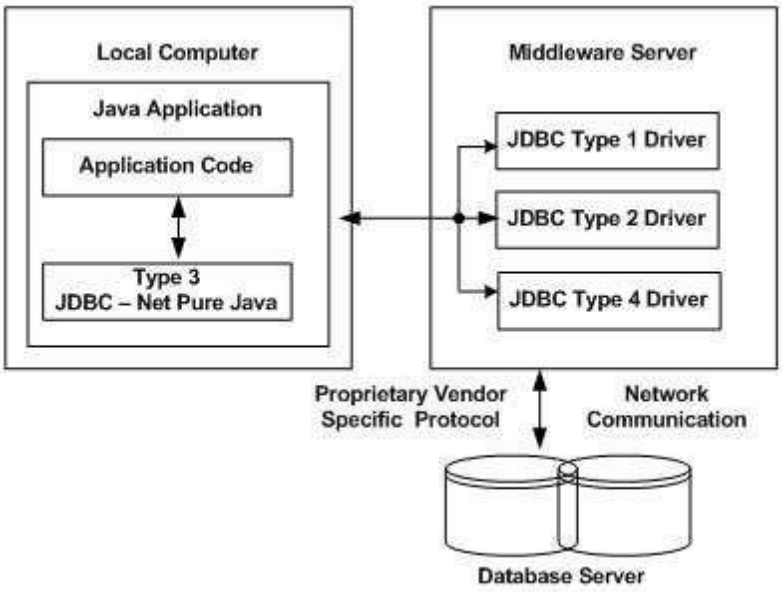


If we change the Database, we have to change the native API, as it is specific to a database and they are mostly obsolete now, but you may realize some speed increase with a Type 2 driver, because it eliminates ODBC's overhead.

The Oracle Call Interface (OCI) driver is an example of a Type 2 driver.

Type 3: JDBC-Net pure Java

In a Type 3 driver, a three-tier approach is used to access databases. The JDBC clients use standard network sockets to communicate with a middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.
This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.

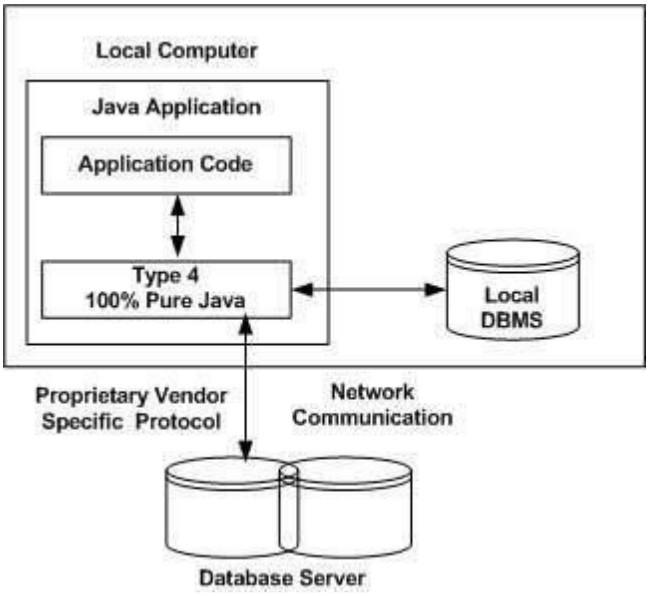


You can think of the application server as a JDBC "proxy," meaning that it makes calls for the client application. As a result, you need some knowledge of the application server's configuration in order to effectively use this driver type.

Your application server might use a Type 1, 2, or 4 driver to communicate with the database, understanding the nuances will prove helpful.

Type 4: 100% Pure Java

In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself.
This kind of driver is extremely flexible, you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.
MySQL's Connector/J driver is a Type 4 driver. Because of the proprietary nature of their network protocols, database vendors usually supply type 4 drivers.



Which Driver should be Used?

If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is 4.
If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.
Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.

The type 1 driver is not considered a deployment-level driver, and is typically used for development and testing purposes only.

Example to connect to the mysql database in java

For connecting java application with the mysql database, you need to follow few steps to perform database connectivity. In this example we are using MySql as the database. So we need to know following information for the mysql database:

1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.

2.**Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/sonoo** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, you need to replace the sonoo with your database name.

3. **Username:** The default username for the mysql database is **root**.

4. **Password:** Password is given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

Let's first create a table in the mysql database, but before creating table, we need to create database first.
1. create database sonoo;

2. use sonoo;

3. create table emp(id **int**(10),name varchar(40),age **int**(3)); **Example to Connect Java Application with mysql database**

In this example, sonoo is the database name, root is the username and password.

```
import java.sql.*;
class MysqlCon
{
public static void main(String args[])
{
Try
{ Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/sonoo","root","root");
//here sonoo is database name, root is username and password
```

```
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3)); con.close();
}
catch(Exception e){ System.out.println(e);
}
}
}
```

The above example will fetch all the records of emp table.