Java AWT Tutorial

AWT contains large number of classes and methods that allows you to create and manage graphical user interface (GUI) applications, such as windows, buttons, scroll bars,etc. The AWT was designed to provide a common set of tools for GUI design that could work on a variety of platforms. The tools provided by the AWT are implemented using each platform's native GUI toolkit, hence preserving the look and feel of each platform. This is an advantage of using AWT.But the disadvantage of such an approach is that GUI designed on one platform may look different when displayed on another platform.

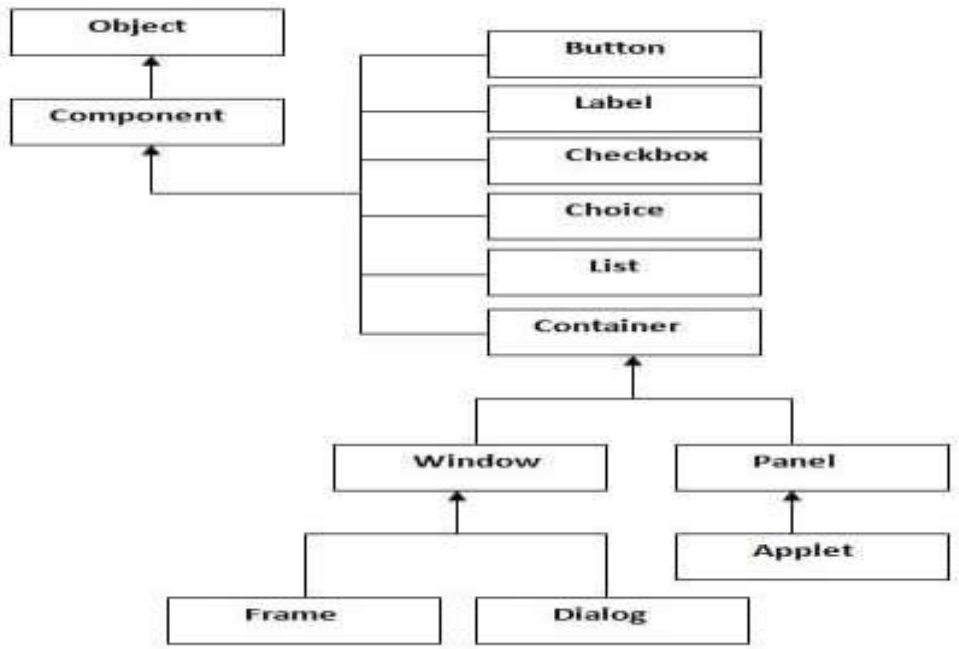
AWT is the foundation upon which Swing is made i.e Swing is a set of GUI interfaces that extends the AWT. But now a days AWT is merely used because most GUI Java programs are implemented using Swing because of its rich implementation of GUI controls and light-weighted nature.

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)

```
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();
}}
```

AWT UI Elements:

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

Sr. No.	Control & Description
1	**Label** A Label object is a component for placing text in a container.
2	**Button** This class creates a labeled button.
3	**Check Box** A check box is a graphical component that can be in either an **on**(true) or **off** (false) state.
4	**Check Box Group** The CheckboxGroup class is used to group the set of checkbox.
5	**List** The List component presents the user with a scrolling list of text items.

6	**Text Field** A TextField object is a text component that allows for the editing of a single line of text.
7	**Text Area** A TextArea object is a text component that allows for the editing of a multiple lines of text.
8	**Choice** A Choice control is used to show pop up menu of choices. Selected choice is shown on the top of the menu.
9	**Canvas** A Canvas control represents a rectangular area where application can draw something or can receive inputs created by user.
10	**Image** An Image control is superclass for all image classes representing graphical images.
11	**Scroll Bar** A Scrollbar control represents a scroll bar component in order to enable user to select from range of values.
12	**Dialog** A Dialog control represents a top-level window with a title and a border used to take some form of input from the user.
13	**File Dialog** A FileDialog control represents a dialog window from which the user can select a file.

Java Swing

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

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

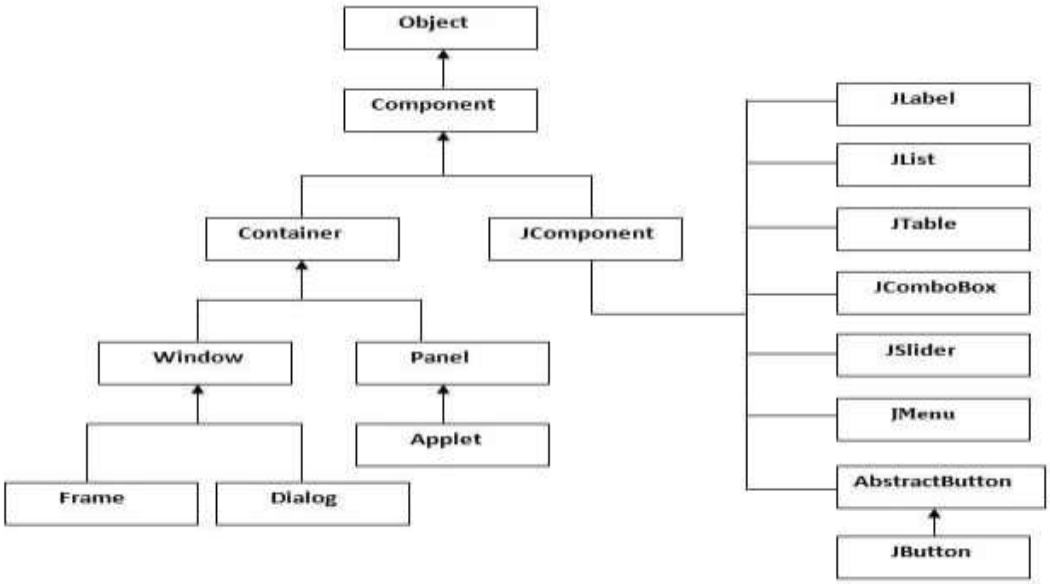
Difference between AWT and Swing

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

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

Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.



Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

Java Swing Examples

There are two ways to create a frame:

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

We can write the code of swing inside the main(), constructor or any other method.

Simple Java Swing Example

Let's see a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

File: FirstSwingExample.java

```
1. import javax.swing.*;
2. public class FirstSwingExample {
3. public static void main(String[] args) {
4. JFrame f=new JFrame();//creating instance of JFrame
5.
6. JButton b=new JButton("click");//creating instance of JButton
7. b.setBounds(130,100,100, 40);//x axis, y axis, width, height
8.
9. f.add(b);//adding button in JFrame
10.
11. f.setSize(400,500);//400 width and 500 height
12. f.setLayout(null);//using no layout managers
13. f.setVisible(true);//making the frame visible
```

```
14. } }
```

Java JButton

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

JButton class declaration

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

```
1. public class JButton extends AbstractButton implements Accessible
```

Commonly used Constructors:

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

Commonly used Methods of JButton class:

Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.

void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

JLabel Control

JLabel is a passive control because it does not create any event when accessed by the user. The label control is an object of Label. A label displays a single line of read-only text. JLabel is created by using javax.JLabel Class.

1	**JLabel()** Constructs an empty label.
2	**JLabel(String text)** Constructs a new label with the specified string of text, left justified.
3	**JLabel(String text, int alignment)** Constructs a new label that presents the specified string of text with the specified alignment.

S.N.	Method & Description
1	**String getText()** Gets the text of this label.
2	**protected String paramString()** Returns a string representing the state of this Label.
3	**void setAlignment(int alignment)** Sets the alignment for this label to the specified alignment.
4	**void setText(String text)** Sets the text for this label to the specified text.

Java JTextField

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

JTextField class declaration

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

1. **public class** JTextField **extends** JTextComponent **implements** SwingConstants

Commonly used Constructors:

Constructor	Description
JTextField()	Creates a new TextField
JTextField(String text)	Creates a new TextField initialized with the specified text.
JTextField(String text, int columns)	Creates a new TextField initialized with the specified text and columns.
JTextField(int columns)	Creates a new empty TextField with the specified number of columns.

Commonly used Methods:

Methods	Description
void addActionListener(ActionListener l)	It is used to add the specified action listener to receive action events from this textfield.
Action getAction()	It returns the currently set Action for this ActionEvent source, or null if no Action is set.
void setFont(Font f)	It is used to set the current font.
void removeActionListener(ActionListener l)	It is used to remove the specified action listener so that it no longer receives action events from this textfield.

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

Java JTextArea

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

JTextArea class declaration

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

1. **public class** JTextArea **extends** JTextComponent

Commonly used Constructors:

Constructor	Description
JTextArea()	Creates a text area that displays no text initially.

JTextArea(String s)	Creates a text area that displays specified text initially.
JTextArea(int row, int column)	Creates a text area with the specified number of rows and columns that displays no text initially.
JTextArea(String s, int row, int column)	Creates a text area with the specified number of rows and columns that displays specified text.

Commonly used Methods:

Methods	Description
void setRows(int rows)	It is used to set specified number of rows.
void setColumns(int cols)	It is used to set specified number of columns.
void setFont(Font f)	It is used to set the specified font.
void insert(String s, int position)	It is used to insert the specified text on the specified position.
void append(String s)	It is used to append the given text to the end of the document.

```
1. import javax.swing.*;
2. public class TextAreaExample
3. {
4.     TextAreaExample(){
5.       JFrame f= new JFrame();
6.       JTextArea area=new JTextArea("Welcome to javatpoint");
7.       area.setBounds(10,30, 200,200);
8.       f.add(area);
9.       f.setSize(300,300);
10.      f.setLayout(null);
11.      f.setVisible(true);
```

```
12.     }
13. public static void main(String args[])
14.     {
15.    new TextAreaExample();
16.     }}
```

Java JCheckBox

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

JCheckBox class declaration

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

Commonly used Constructors:

Constructor	Description
JJCheckBox()	Creates an initially unselected check box button with no text, no icon.
JChechBox(String s)	Creates an initially unselected check box with text.
JCheckBox(String text, boolean selected)	Creates a check box with text and specifies whether or not it is initially selected.
JCheckBox(Action a)	Creates a check box where properties are taken from the Action supplied.

Java JRadioButton

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

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

JRadioButton class declaration

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

1. public class JRadioButton extends JToggleButton implements Accessible

Commonly used Constructors:

Constructor	**Description**
JRadioButton()	Creates an unselected radio button with no text.
JRadioButton(String s)	Creates an unselected radio button with specified text.
JRadioButton(String s, boolean selected)	Creates a radio button with the specified text and selected status.

Commonly used Methods:

Methods	Description
void setText(String s)	It is used to set specified text on button.
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the
Icon getIcon()	It is used to get the Icon of the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

Java JComboBox

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

Commonly used Constructors:

Constructor	**Description**
JComboBox()	Creates a JComboBox with a default data model.

JComboBox(Object[] items)	Creates a JComboBox that contains the elements in the specified array.
JComboBox(Vector<?> items)	Creates a JComboBox that contains the elements in the specified Vector.

Commonly used Methods:

Methods	**Description**
void addItem(Object anObject)	It is used to add an item to the item list.
void removeItem(Object anObject)	It is used to delete an item to the item list.
void removeAllItems()	It is used to remove all the items from the list.
void setEditable(boolean b)	It is used to determine whether the JComboBox is editable.
void addActionListener(ActionListener a)	It is used to add the ActionListener.
void addItemListener(ItemListener i)	It is used to add the ItemListener.

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

Event Handling

Any program that uses GUI (graphical user interface) such as Java application written for windows, is event driven. Event describes the change in state of any object. **For Example :** Pressing a button, Entering a character in Textbox, Clicking or Dragging a mouse, etc.

Components of Event Handling

Event handling has three main components,

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

How Events are handled ?

A source generates an Event and send it to one or more listeners registered with the source. Once event is received by the listener, they process the event and then return. Events are supported by a number of Java packages, like **java.util**, **java.awt** and **java.awt.event**.

Important Event Classes and Interface

Event Classes	Description	Listener Interface
ActionEvent	generated when button is pressed, menu-item is selected, list-item is double clicked	ActionListener
MouseEvent	generated when mouse is dragged, moved,clicked,pressed or released and also when it enters or exit a component	MouseListener
KeyEvent	generated when input is received from keyboard	KeyListener
ItemEvent	generated when check-box or list item is clicked	ItemListener
TextEvent	generated when value of textarea or textfield is changed	TextListener
MouseWheelEvent	generated when mouse wheel is moved	MouseWheelListener
WindowEvent	generated when window is activated, deactivated, deiconified, iconified, opened or closed	WindowListener
ComponentEvent	generated when component is hidden, moved, resized or set visible	ComponentEventListener

ContainerEvent	generated when component is added or removed from container	ContainerListener
AdjustmentEvent	generated when scroll bar is manipulated	AdjustmentListener
FocusEvent	generated when component gains or loses keyboard focus	FocusListener

Event listeners represent the interfaces responsible to handle events. Java provides various Event listener classes, however, only those which are more frequently used will be discussed. Every method of an event listener method has a single argument as an object which is the subclass of EventObject class. For example, mouse event listener methods will accept instance of MouseEvent, where MouseEvent derives from EventObject.

EventListner Interface

It is a marker interface which every listener interface has to extend. This class is defined in **java.util** package.

SWING Event Listener Interfaces

Following is the list of commonly used event listeners.

Sr.No.	Class & Description
1	**ActionListener** This interface is used for receiving the action events.
2	**ComponentListener** This interface is used for receiving the component events.
3	**ItemListener** This interface is used for receiving the item events.
4	**KeyListener** This interface is used for receiving the key events.
5	**MouseListener** This interface is used for receiving the mouse events.
6	**WindowListener** This interface is used for receiving the window events.
7	**AdjustmentListener** This interface is used for receiving the adjustment events.

8	**ContainerListener** This interface is used for receiving the container events.
9	**MouseMotionListener** This interface is used for receiving the mouse motion events.
10	**FocusListener** This interface is used for receiving the focus events.

ActionListener:

The class which processes the ActionEvent should implement this interface. The object of that class must be registered with a component. The object can be registered using the **addActionListener()** method. When the action event occurs, that object's actionPerformed method is invoked

Sr.No.	Method & Description
1	**void actionPerformed(ActionEvent e)** Invoked when an action occurs.

ItemListener

The class which processes the ItemEvent should implement this interface.The object of that class must be registered with a component. The object can be registered using the **addItemListener()** method. When the action event occurs, that object's itemStateChanged method is invoked.

Sr.No.	Method & Description
1	**void itemStateChanged(ItemEvent e)** Invoked when an item has been selected or deselected by the user.

KeyListener

The class which processes the KeyEvent should implement this interface. The object of that class must be registered with a component. The object can be registered using the **addKeyListener()** method.

nterface Methods

Sr.No.	Method & Description
1	**void keyPressed(KeyEvent e)** Invoked when a key has been pressed.
2	**void keyReleased(KeyEvent e)** Invoked when a key has been released.
3	**void keyTyped(KeyEvent e)** Invoked when a key has been typed.

Window Listener

The class which processes the WindowEvent should implement this interface. The object of that class must be registered with a component. The object can be registered using the **addWindowListener()** method.

Interface Methods

Sr.No.	Method & Description
1	**void windowActivated(WindowEvent e)** Invoked when the Window is set to be the active Window.
2	**void windowClosed(WindowEvent e)** Invoked when a window has been closed as a result of calling dispose on the Window.
3	**void windowClosing(WindowEvent e)** Invoked when the user attempts to close the Window from the Window's system menu.
4	**void windowDeactivated(WindowEvent e)** Invoked when a Window is no longer the active Window.
5	**void windowDeiconified(WindowEvent e)** Invoked when a Window is changed from a minimized to a normal state.

6	**void windowIconified(WindowEvent e)** Invoked when a Window is changed from a normal to a minimized state.
7	**void windowOpened(WindowEvent e)** Invoked the first time a Window is made visible

FocusListener

The interface **FocusListener** is used for receiving keyboard focus events. The class that processes focus events needs to implements this interface.

Sr.No.	Method & Description
1	**void focusGained(FocusEvent e)** Invoked when a component gains the keyboard focus.
2	**void focusLost(FocusEvent e)** Invoked when a component loses the keyboard focus.

Adapter Classes

Adapters are abstract classes for receiving various events. The methods in these classes are empty. These classes exist as convenience for creating listener objects.

SWING Adapters

Following is the list of commonly used adapters while listening GUI events in SWING.

Sr.No.	Adapter & Description
1	**FocusAdapter** An abstract adapter class for receiving focus events.
2	**KeyAdapter** An abstract adapter class for receiving key events.
3	**MouseAdapter** An abstract adapter class for receiving mouse events.
4	**MouseMotionAdapter** An abstract adapter class for receiving mouse motion events.

5	**WindowAdapter** An abstract adapter class for receiving window events.

The class **FocusAdapter** is an abstract (adapter) class for receiving keyboard focus events. All methods of this class are empty. This class is convenience class for creating listener objects.

Class Constructors

Sr.No.	Constructor & Description
1	**FocusAdapter()**

Class Methods

Sr.No.	Method & Description
1	**void focusGained(FocusEvent e)** Invoked when a component gains the keyboard focus.

The class **KeyAdapter** is an abstract (adapter) class for receiving keyboard events. All methods of this class are empty. This class is convenience class for creating listener objects.

Class Constructors

Sr.No.	Constructor & Description
1	**KeyAdapter()**

Class Methods

Sr.No.	Method & Description
1	**void keyPressed(KeyEvent e)** Invoked when a key has been pressed.
2	**void keyReleased(KeyEvent e)** Invoked when a key has been released.
3	**void keyTyped(KeyEvent e)** Invoked when a key has been typed.

The class **WindowAdapter** is an abstract (adapter) class for receiving window events. All methods of this class are empty. This class is convenience class for creating listener objects.

Class Constructors

Sr.No.	Constructor & Description
1	**WindowAdapter()**

Class Methods

Sr.No.	Method & Description
1	**void windowActivated(WindowEvent e)** Invoked when a Window is activated.
2	**void windowClosed(WindowEvent e)** Invoked when a Window has been closed.
3	**void windowClosing(WindowEvent e)** Invoked when a Window is in the process of being closed.
4	**void windowDeactivated(WindowEvent e)** Invoked when a Window is de-activated.
5	**void windowDeiconified(WindowEvent e)** Invoked when a Window is de-iconified.
6	**void windowGainedFocus(WindowEvent e)** Invoked when the Window is set to be the focused Window, which means that the Window, or one of its subcomponents, will receive keyboard events.
7	**void windowIconified(WindowEvent e)** Invoked when a Window is iconified.
8	**void windowLostFocus(WindowEvent e)**

	Invoked when the Window is no longer the focused Window, which means that keyboard events will no longer be delivered to the Window or any of its subcomponents.
9	**void windowOpened(WindowEvent e)** Invoked when a Window has been opened.
10	**void windowStateChanged(WindowEvent e)** Invoked when a Window state is changed.

Applets:

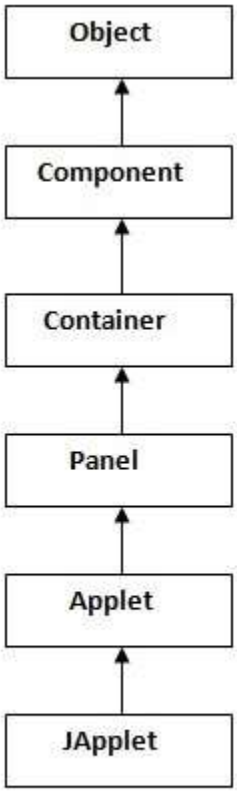
Applet is a Java program that can be embedded into a web page. It runs inside the web browser and works at client side. Applet is embedded in a HTML page using the APPLET or **OBJECT** tag and hosted on a web server. Applets are used to make the web site more dynamic and entertaining.

Java Applets are usually used to add small, **interactive** components or enhancements to a webpage. These may consist of buttons, **scrolling** text, or stock tickers, but they can also be used to **display** larger programs like word processors or games.

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

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

Hierarchy of Applet



Life Cycle of an Applet

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

- **init** − This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.
- **start** − This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.
- **stop** − This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.
- **destroy** − This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.
- **paint** − Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

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.

"Hello, World" Applet

Following is a simple applet named HelloWorldApplet.java −

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


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

Invoking an Applet

An applet may be invoked by embedding directives in an HTML file and viewing the file through an applet viewer or Java-enabled browser.

The <applet> tag is the basis for embedding an applet in an HTML file. Following is an example that invokes the "Hello, World" applet −

```
<html>
  <title>The Hello, World Applet</title>
  <hr>
  <applet code = "HelloWorldApplet.class" width = "320" height = "120">
    If your browser was Java-enabled, a "Hello, World"
    message would appear here.
  </applet>
  <hr>
</html>
```

Passing parameters to Applet

Java allows users to pass user-defined parameters to an applet with the help of <PARAM>tags. The <PARAM>tag has a NAME attribute which defines the name of the parameter and a VALUE attribute which specifies the value of the parameter. In the applet source code, the applet can refer to the parameter by its NAME to find its value. The syntax of the <PARAM>tag is:

<APPLET code="appletclass.class" width="" height="" >

<PARAMNAME=parameter1_name VALUE=parameter1_value>

<PARAMNAME=parameter2_name VALUE=parameter2_value>

<PARAMNAME=parametern_name VALUE=parametern_value>

</APPLET>

getParameter() Method

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

```
/*
<html>
<applet code="HelloWorldApplet.class" width="300" height="300">
<param name="msg" value="Welcome to Mca 1st Sem Bindhu applet">
</applet>
</html>
*/


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

public class HelloWorldApplet extends JApplet{
    public void paint (Graphics g) {
            Font myFont2 = new Font(Font.SERIF, Font.BOLD | Font.ITALIC, 26);
            g.setFont(myFont2);
            String str=getParameter("msg");
            g.drawString(str,50, 50);                }}
```

Displaying Graphics in Applet

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

Commonly used methods of Graphics class:

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

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

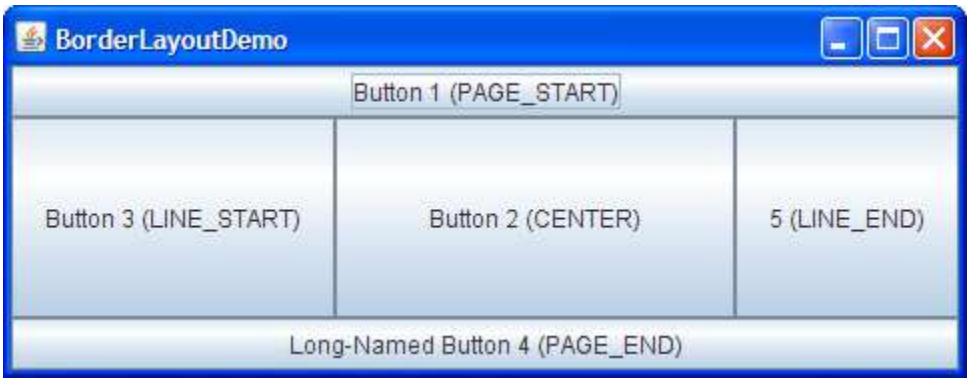
Java BorderLayout

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.



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.

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.

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.

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

- Ex:-

```
layout = new BorderLayout( 5, 5 );

        Container c = getContentPane();
        c.setLayout( layout );
```