Applets Unit-I

Applets are small applications that are accessed on an Internet server, transported over the Internet, automatically installed, and run as part of a web document.
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

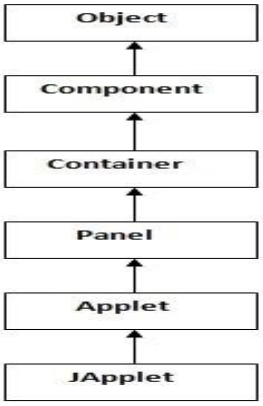
There are many advantages of applet. They are as follows:

- 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



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

The **Applet** class is contained in the **java.applet** package. **Applet** contains several methods that give you detailed control over the execution of your applet.
It is important to state at the outset that there are two varieties of **Applets.** The first are those based directly on the **Applet** class, which uses the Abstract Window Toolkit (AWT) to provide the graphic user interface (or use no GUI at all).The second type of applets are those based on the Swing class **JApplet**. Swing applets use the Swing classes to provide the GUI.

All applets are subclasses (either directly or indirectly) of **Applet**. Applets are not stand-alone programs. Instead, they run within either a web browser or an applet viewer.
Execution of an applet does not begin at **main ( )**. (Few applets even have **main ( )** methods) Instead, execution of an applet is started and controlled with an entirely different mechanism. Output to your applet's window is not performed by **System.out.println( )**. Rather, in non-Swing applets, output is handled with various AWT methods, such as **drawString( )**, which outputs a string to a specified X,Y location. Input is also handled differently than in a console application.

Simple Example:

Applets differ from console-based applications in several key areas.

```
import java.awt.*;
import java.applet.*;
public class SimpleApplet extends Applet {
public void paint(Graphics g) {
g.drawString("A Simple Applet", 20, 20);
}
}
```

This applet begins with two **import** statements. The first imports the Abstract Window Toolkit (AWT) classes. Applets interact with the user (either directly or indirectly) through the AWT, not through the console-based I/O classes. The AWT contains support for a window-based, graphical user interface. The second **import** statement imports the **applet** package, which contains the class **Applet**. Every applet that you create must be a subclass of **Applet**. The next line in the program declares the class **SimpleApplet**. This class must be declared as **public**, because it will be accessed by code that is outside the program. Inside **SimpleApplet**, **paint ( )** is declared. This method is defined by the AWT and must be overridden by the applet. **Paint ( )** is called each time that the applet must redisplay its output. **Paint ( )** is also called when the applet begins execution. The **paint ( )** method has one parameter of type **Graphics**. This parameter contains the graphics context, which describes the graphics environment in which the applet is running. This context is used whenever output to the applet is required. Inside **paint ( )** is a call to **drawString( )**, which is a member of the **Graphics** class. This method outputs a string beginning at the specified X,Y location. It has the following general form:

void drawString(String *message*, int *x*, int *y*)

Here, *message* is the string to be output beginning at *x,y.*
Notice that the applet does not have a **main ( )** method. Unlike Java programs, applets do not begin execution at **main ( )**. In fact, most applets don't even have a **main ( )** method. Instead, an applet begins execution when the name of its class is passed to an applet viewer or to a network browser.
There are two ways in which you can run an applet:

- Executing the applet within a Java-compatible web browser.
- Using an applet viewer, such as the standard tool, **appletviewer**. An applet viewer executes your applet in a window. This is generally the fastest and easiest way to test your applet.

To execute an applet in a web browser, you need to write a short HTML text file that contains a tag that loads the applet. Currently, Sun recommends using the APPLET tag for this purpose. Here is the HTML file that executes **SimpleApplet**:

```
<applet code="SimpleApplet" width=200 height=60>
</applet>
```

The **width** and **height** statements specify the dimensions of the display area used by the applet. After you create this file, you can execute your browser and then load this file, which causes **SimpleApplet** to be executed.

To execute **SimpleApplet** with an applet viewer, you may also execute the HTML file shown earlier. For example, if the preceding HTML file is called **RunApp.html**, then the following command line will run **SimpleApplet**:

```
C:\>appletviewer RunApp.html
```

However, a more convenient method exists that you can use to speed up testing. Simply include a comment at the head of your Java source code file that contains the APPLET tag. By doing so, your code is documented with a prototype of the necessary HTML statements, and you can test your compiled applet merely by starting the applet viewer with your Java source code file. If you use this method, the **SimpleApplet** source file looks like this:

```
import java.awt.*;
import java.applet.*;
/*
<applet code="SimpleApplet" width=200 height=60>
</applet>
*/
public class SimpleApplet extends Applet {
public void paint(Graphics g) {
g.drawString("A Simple Applet", 20, 20);
}
}
```

With this approach, you can quickly iterate through applet development by using these three steps:

1. Edit a Java source file.
2. Compile your program.
3. Execute the applet viewer, specifying the name of your applet's source file. The applet viewer will encounter the APPLET tag within the comment and execute your applet.

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

Key Points:

- Applets do not need a **main ( )** method.

- Applets must be run under an applet viewer or a Java-compatible browser.
- User I/O is not accomplished with Java's stream I/O classes. Instead, applets use the interface provided by the AWT or Swing.

The **Applet** class defines the methods. **Applet** provides all necessary support for applet execution, such as starting and stopping. It also provides methods that load and display images, and methods that load and play audio clips.

Method	Description
void destroy( )	Called by the browser just before an applet is terminated. Your applet will override this method if it needs to perform any cleanup prior to its destruction.
AccessibleContext getAccessibleContext( )	Returns the accessibility context for the invoking object.
AppletContext getAppletContext( )	Returns the context associated with the applet.
String getAppletInfo( )	Returns a string that describes the applet.
AudioClip getAudioClip(URL url)	Returns an AudioClip object that encapsulates the audio clip found at the location specified by url.
AudioClip getAudioClip(URL url, String clipName)	Returns an AudioClip object that encapsulates the audio clip found at the location specified by url and having the name specified by clipName.
URL getCodeBase( )	Returns the URL associated with the invoking applet.
URL getDocumentBase( )	Returns the URL of the HTML document that invokes the applet.
Image getImage(URL url)	Returns an Image object that encapsulates the image found at the location specified by url.
Image getImage(URL url, String imageName)	Returns an Image object that encapsulates the image found at the location specified by url and having the name specified by imageName.
String getParameter(String paramName)	Returns the parameter associated with paramName. **null** is returned if the specified parameter is not found.
String[ ] [ ] getParameterInfo( )	Returns a String table that describes the parameters recognized by the applet. Each entry in the table must consist of three strings that contain the name of the parameter, a description of its type and/or range, and an explanation of its purpose.
void init()	Called when an applet begins execution. It is the first method called for any applet.
boolean isActive( )	Returns true if the applet has been started. It returns false if the applet has been stopped.
void play(URL url)	If an audio clip is found at the location specified by url, the clip is played.
void play(URL url, String clipName)	If an audio clip is found at the location specified by url with the name specified by clipName, the clip is played.
void resize(Dimension dim)	Resizes the applet according to the dimensions specified by dim. Dimension is a class stored inside java.awt. It contains two integer fields: width and height.
void resize(int width, int height)	Resizes the applet according to the dimensions specified by width and height.
void showStatus(String str)	Displays str in the status window of the browser or applet viewer. If the browser does not support a status window, then no action takes place.
void start( )	Called by the browser when an applet should start (or resume) execution. It is automatically called after init( ) when an applet first begins.

void stop( )	Called by the browser to suspend execution of the applet. Once stopped, an applet is restarted when the browser calls start( ).

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.

Applet Initialization and Termination

It is important to understand the order in which the various methods shown in the skeleton are called. When an applet begins, the following methods are called, in this sequence:

1. **init( )**
2. **start( )**
3. **paint( )**

When an applet is terminated, the following sequence of method calls takes place:

1. **stop( )**
2. **destroy( )**

init( ) : The **init( )** method is the first method to be called. This is where you should initialize variables. This method is called only once during the run time of your applet.

start( ): The **start( )** method is called after **init( )**. It is also called to restart an applet after it has been stopped. Whereas **init( )** is called once—the first time an applet is loaded—**start( )** is called each time an applet's HTML document is displayed onscreen. So, if a user leaves a web page and comes back, the applet resumes execution at **start( )**.

paint( ): The **paint( )** method is called each time your applet's output must be redrawn. This situation can occur for several reasons. For example, the window in which the applet is running may be overwritten by another window and then uncovered. Or the applet window may be minimized and then restored. **paint( )** is also called when the applet begins execution. Whatever the cause, whenever the applet must redraw its output, **paint( )** is called. The **paint( )** method has one parameter of type **Graphics**. This parameter will contain the graphics context, which describes the graphics environment in which the applet is running. This context is used whenever output to the applet is required.

stop( ): The **stop( )** method is called when a web browser leaves the HTML document containing the applet—when it goes to another page, for example. When **stop( )** is called, the applet is probably running. You should use **stop( )** to suspend threads that don't need to run when the applet is not visible. You can restart them when **start( )** is called if the user returns to the page.

destroy( ): The **destroy( )** method is called when the environment determines that your applet needs to be removed completely from memory. At this point, you should free up any resources the applet may be using. The **stop( )** method is always called before **destroy( )**.

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.

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>
```

To set the background color of an applet's window, use **setBackground( )**. To set the foreground color use **setForeground( )**. These methods are defined by **Component**, and they have the following general forms:

```
void setBackground(Color newColor)
void setForeground(Color newColor)
```

Ex:

```
setBackground(Color.green);
setForeground(Color.red);
```

Requesting Repainting

As a general rule, an applet writes to its window only when its **update ( )** or **paint ( )** method is called by the AWT. Whenever your applet needs to update the information displayed in its window, it simply calls **repaint ( )**. The **repaint( )** method is defined by the AWT. It causes the AWT run-time system to execute a call to your applet's **update( )** method, which, in its default implementation, calls **paint( )**.
The **repaint( )** method has four forms.

void repaint( ): This version causes the entire window to be repainted.
void repaint(int *left*, int *top*, int *width*, int *height*): This version specifies a region that will be repainted. Here, the coordinates of the upper-left corner of the region are specified by *left* and *top,* and the width and height of the region are passed in *width* and *height*.

void repaint(long *maxDelay*)
void repaint(long *maxDelay*, int *x*, int *y*, int *width*, int *height*)

Here, *maxDelay* specifies the maximum number of milliseconds that can elapse before **update( )** is called.

A Simple Banner Applet

```
/* A simple banner applet.
This applet creates a thread that scrolls
the message contained in msg right to left
across the applet's window.
*/
import java.awt.*;
import java.applet.*;
/*
<applet code="SimpleBanner" width=300 height=50>
</applet>
*/
public class SimpleBanner extends Applet implements Runnable {
String msg = " A Simple Moving Banner.";
```

```
Thread t = null;
int state;
boolean stopFlag;
// Set colors and initialize thread.
public void init() {
setBackground(Color.cyan);
setForeground(Color.red);
}
// Start thread
public void start() {
t = new Thread(this);
stopFlag = false;
t.start();
}
// Entry point for the thread that runs the banner.
public void run() {
char ch;
// Display banner
for( ; ; ) {
try {
repaint();
Thread.sleep(250);
ch = msg.charAt(0);
msg = msg.substring(1, msg.length());
msg += ch;
if(stopFlag)
break;
} catch(InterruptedException e) {}
}
}
// Pause the banner.
public void stop() {
stopFlag = true;
t = null;
}
// Display the banner.
public void paint(Graphics g) {
g.drawString(msg, 50, 30);
}
}
```

Following is sample output:

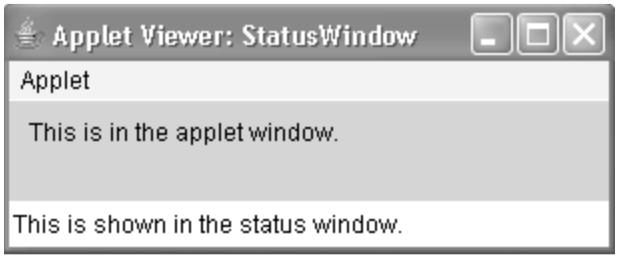
Using the Status Window

In addition to displaying information in its window, an applet can also output a message to the status window of the browser or applet viewer on which it is running. To do so, call **showStatus( )** with the string that you want displayed. The status window is a good place

to give the user feedback about what is occurring in the applet, suggest options, or possibly report some types of errors. The status window also makes an excellent debugging aid, because it gives you an easy way to output information about your applet.

```
// Using the Status Window.
import java.awt.*;
import java.applet.*;
/*
<applet code="StatusWindow" width=300 height=50>
</applet>
*/
public class StatusWindow extends Applet{
public void init() {
setBackground(Color.cyan);
}
// Display msg in applet window.
public void paint(Graphics g) {
g.drawString("This is in the applet window.", 10, 20);
showStatus("This is shown in the status window.");
}
}
```

Sample output from this program is shown here:



The HTML APPLET Tag

APPLET tag is used to start an applet from both an HTML document and from an applet viewer. An applet viewer will execute each APPLET tag that it finds in a separate window, while web browsers will allow many applets on a single page.

< APPLET
[CODEBASE = *codebaseURL*]
CODE = *appletFile*
[ALT = *alternateText*]
[NAME = *appletInstanceName*]
WIDTH = *pixels* HEIGHT = *pixels*
[ALIGN = *alignment*]
[VSPACE = *pixels*] [HSPACE = *pixels*]
>
[< PARAM NAME = *AttributeName* VALUE = *AttributeValue*>]
[< PARAM NAME = *AttributeName2* VALUE = *AttributeValue*>]
. . .
[*HTML Displayed in the absence of Java*]
</APPLET>

CODEBASE It is an optional attribute that specifies the base URL of the applet code, which is the directory that will be searched for the applet's executable class file (specified by the CODE tag). The HTML document's URL directory is used as the CODEBASE if this attribute is not specified. The CODEBASE does not have to be on the host from which the HTML document was read.

CODE It is a required attribute that gives the name of the file containing your applet's compiled **.class** file. This file is relative to the code base URL of the applet, which is the directory that the HTML file was in or the directory indicated by CODEBASE if set.

ALT The ALT tag is an optional attribute used to specify a short text message that should be displayed if the browser recognizes the APPLET tag but can't currently run Java applets. This is distinct from the alternate HTML you provide for browsers that don't support applets.

NAME It is an optional attribute used to specify a name for the applet instance. Applets must be named in order for other applets on the same page to find them by name and communicate with them. To obtain an applet by name, use **getApplet( )**, which is defined by the **AppletContext** interface.

WIDTH and HEIGHT WIDTH and HEIGHT are required attributes that give the size (in pixels) of the applet display area.

ALIGN It is an optional attribute that specifies the alignment of the applet. This attribute is treated the same as the HTML IMG tag with these possible values: LEFT, RIGHT, TOP, BOTTOM, MIDDLE, BASELINE, TEXTTOP, ABSMIDDLE, and ABSBOTTOM.

VSPACE and HSPACE these attributes are optional. VSPACE specifies the space, in pixels, above and below the applet. HSPACE specifies the space, in pixels, on each side of the applet. They're treated the same as the IMG tag's VSPACE and HSPACE attributes.

PARAM NAME and VALUE The PARAM tag allows you to specify applet-specific arguments in an HTML page. Applets access their attributes with the **getParameter( )** method.

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

/*<html>
<Body>
<applet code="UseParam.class" width="300" height="300">
<param name="msg" value="Welcome to applet">
</applet>
</body>
</html> */

public class UseParam extends Applet{

public void paint(Graphics g){
String str=getParameter("msg");
g.drawString(str,50, 50);
}

}
```

getDocumentBase( ) and getCodeBase( )

Java will allow the applet to load data from the directory holding the HTML file that started the applet (the *document base*) and the directory from which the applet's class file was loaded (the *code base*). These directories are returned as **URL** objects by **getDocumentBase( )** and **getCodeBase( )**.

// Display code and document bases.

```
import java.awt.*;
import java.applet.*;
import java.net.*;
/*
<applet code="Bases" width=300 height=50>
</applet>
*/
public class Bases extends Applet{
// Display code and document bases.
public void paint(Graphics g) {
String msg;
URL url = getCodeBase(); // get code base
msg = "Code base: " + url.toString();
g.drawString(msg, 10, 20);
url = getDocumentBase(); // get document base
msg = "Document base: " + url.toString();
g.drawString(msg, 10, 40);
}
}
```

Sample output from this program is shown here:

AppletContext and showDocument( )

One application of Java is to use active images and animation to provide a graphical means of navigating the Web that is more interesting than simple text-based links. To allow your applet to transfer control to another URL, you must use the **showDocument( )** method defined by the **AppletContext** interface. **AppletContext** is an interface that lets you get information from the applet's execution environment. The context of the currently executing applet is obtained by a call to the **getAppletContext( )** method defined by **Applet**.
There are two **showDocument( )** methods. The method **showDocument(URL)** displays the document at the specified **URL**. The method **showDocument(URL, String)** displays the specified document at the specified location within the browser window. Valid arguments for *where* are "_self" (show in current frame), "_parent" (show in parent frame), "_top" (show in topmost frame), and "_blank" (show in new browser window).

```
/* Using an applet context, getCodeBase(),
and showDocument() to display an HTML file.
*/
import java.awt.*;
import java.applet.*;
import java.net.*;
/*
<applet code="ACDemo" width=300 height=50>
</applet>
```

```
*/
public class ACDemo extends Applet{
public void start() {
AppletContext ac = getAppletContext();
URL url = getCodeBase(); // get url of this applet
try {
ac.showDocument(new URL(url+"Test.html"));
} catch(MalformedURLException e) {
showStatus("URL not found");
}
}
}
```

java.applet.AppletContext class provides the facility of communication between applets. We provide the name of applet through the HTML file. It provides getApplet() method that returns the object of Applet. Syntax:

```
public Applet getApplet(String name){}
```

Example of Applet Communication

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class ContextApplet extends Applet implements ActionListener{
Button b;

public void init(){
b=new Button("Click");
b.setBounds(50,50,60,50);

add(b);
b.addActionListener(this);
}

public void actionPerformed(ActionEvent e){

AppletContext ctx=getAppletContext();
Applet a=ctx.getApplet("app2");
a.setBackground(Color.yellow);
}
}
```

myapplet.html

```
<html>
<body>
<applet code="ContextApplet.class" width="150" height="150" name="app1">
</applet>

<applet code="First.class" width="150" height="150" name="app2">
</applet>
```

```
</body>
</html>
```

Java AWT Tutorial

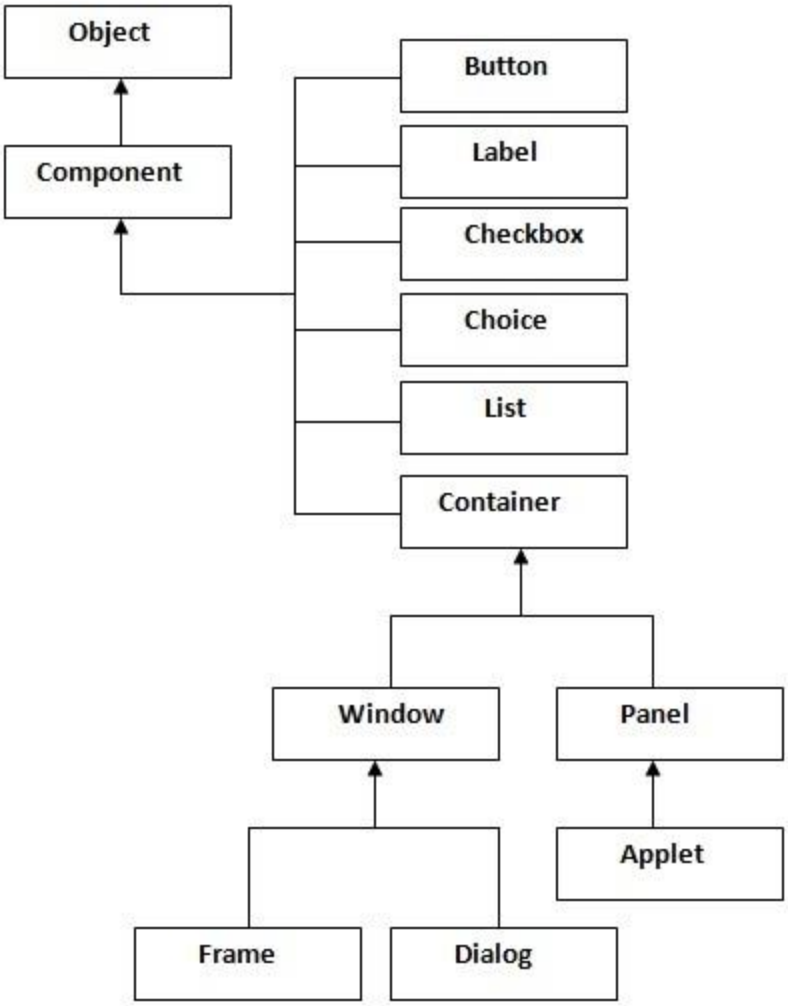
Java AWT (Abstract Windowing Toolkit) is *an API to develop GUI or window-based application in java*.

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 uses the resources of system.

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 fal

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)

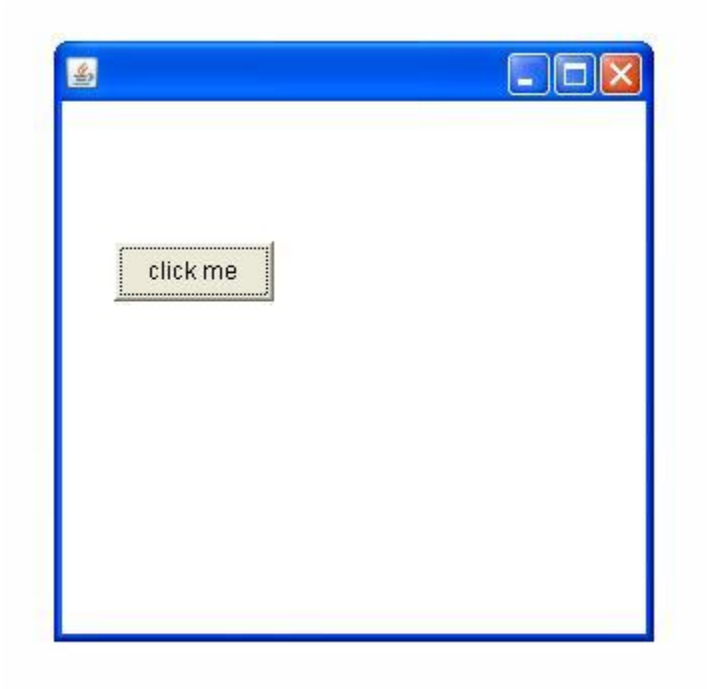
Simple example of AWT by inheritance

```
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.
7.  add(b);//adding button into frame
8.  setSize(300,300);//frame size 300 width and 300 height
9.  setLayout(null);//no layout manager
10. setVisible(true);//now frame will be visible, by default not visible
11. }
12. public static void main(String args[]){
13. First f=new First();
14. }}
```

download this example

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.



Simple example of AWT by association

```
1.  import java.awt.*;
2.  class First2{
3.  First2(){
4.  Frame f=new Frame();
5.
6.  Button b=new Button("click me");
7.  b.setBounds(30,50,80,30);
8.
```

```
9.  f.add(b);
10. f.setSize(300,300);
11. f.setLayout(null);
12. f.setVisible(true);
13. }
14. public static void main(String args[]){
15. First2 f=new First2();
16. }}
```

Event and Listener (Java Event Handling)

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

Event classes 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. Implement the Listener interface and overrides its methods

2. Register the component with the Listener

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){}

EventHandling Codes:

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

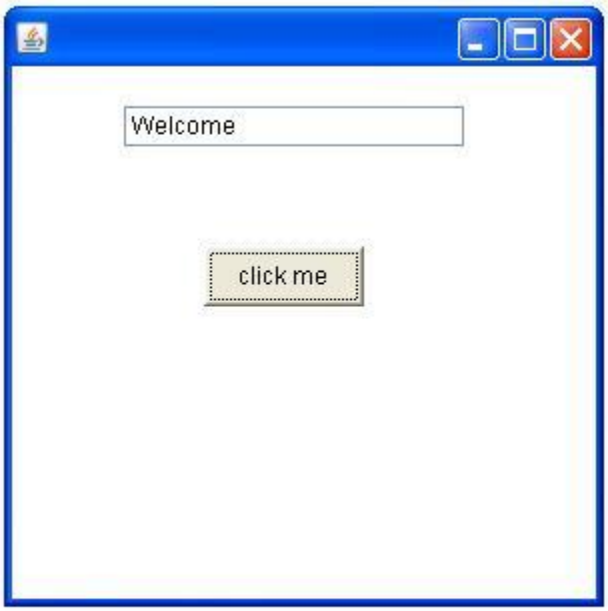
1. Same class
2. Other class
3. Annonymous class

Example of event handling within class:

```
1. import java.awt.*;
2. import java.awt.event.*;
3.
4. class AEvent extends Frame implements ActionListener{
5. TextField tf;
6. AEvent(){
7.
8. tf=new TextField();
9. tf.setBounds(60,50,170,20);
```

```
10.
11. Button b=new Button("click me");
12. b.setBounds(100,120,80,30);
13.
14. b.addActionListener(this);
15.
16. add(b);add(tf);
17.
18. setSize(300,300);
19. setLayout(null);
20. setVisible(true);
21.
22. }
23.
24. public void actionPerformed(ActionEvent e){
25. tf.setText("Welcome");
26. }
27.
28. public static void main(String args[]){
29. new AEvent();
30. }
31. }
```

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



2) Example of event handling by Outer class:

```
1. import java.awt.*;
2. import java.awt.event.*;
3. class AEvent2 extends Frame{
4. TextField tf;
5. AEvent2(){
6. 
7. tf=new TextField();
8. tf.setBounds(60,50,170,20);
9. 
10. Button b=new Button("click me");
11. b.setBounds(100,120,80,30);
12. 
13. Outer o=new Outer(this);
14. b.addActionListener(o);//passing outer class instance
15. 
16. add(b);add(tf);
17. 
18. setSize(300,300);
19. setLayout(null);
20. setVisible(true);
21. }
22. public static void main(String args[]){
23. new AEvent2();
24. }
25. }
```

```
1. import java.awt.event.*;
2. class Outer implements ActionListener{
3. AEvent2 obj;
4. Outer(AEvent2 obj){
5. this.obj=obj;
6. }
7. public void actionPerformed(ActionEvent e){
8. obj.tf.setText("welcome");
9. }
10. }
```

---

3) Example of event handling by Annonymous class:

```
1. import java.awt.*;
2. import java.awt.event.*;
3. class AEvent3 extends Frame{
4. TextField tf;
5. AEvent3(){
6. tf=new TextField();
7. tf.setBounds(60,50,170,20);
8. Button b=new Button("click me");
9. b.setBounds(50,120,80,30);
```

```
10.
11. b.addActionListener(new ActionListener(){
12. public void actionPerformed(){
13. tf.setText("hello");
14. }
15. });
16. add(b);add(tf);
17. setSize(300,300);
18. setLayout(null);
19. setVisible(true);
20. }
21. public static void main(String args[]){
22. new AEvent3();
23. }
24. }
```

UNIT-2

GUI PROGRAMMING WITH JAVA

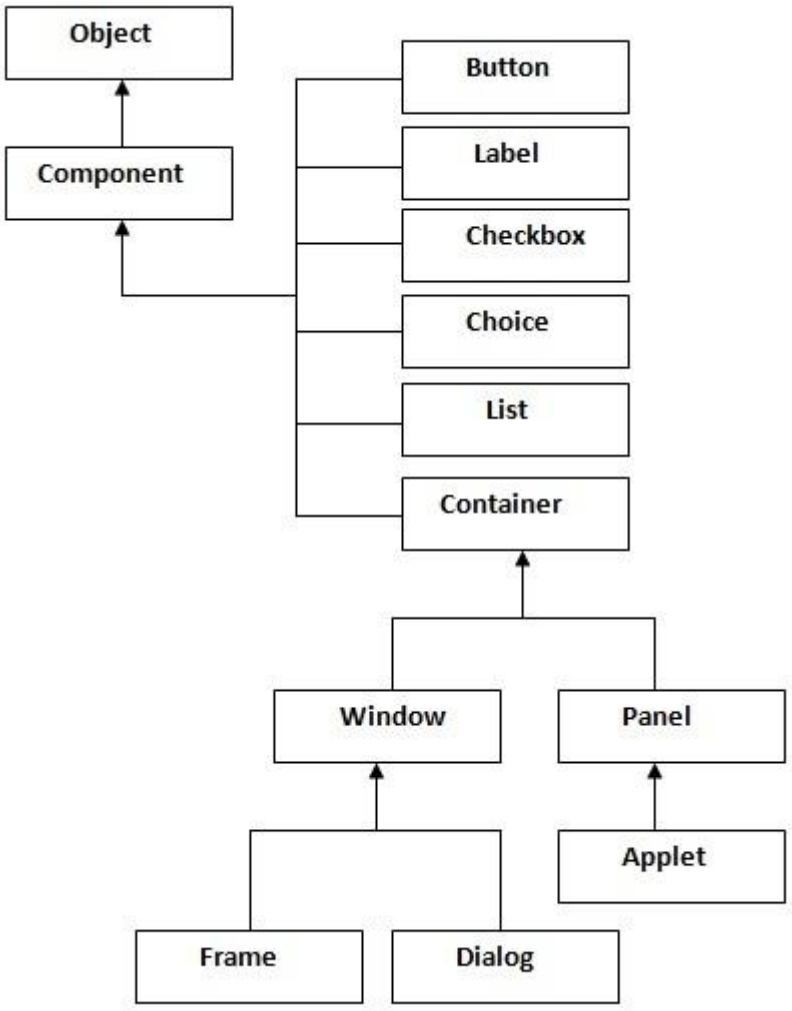
THE AWT CLASS HIERARCHY:

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.

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.

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**	Swing provides **more powerful**

	components than Swing.	**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**.

MVC ARCHITECTURE:

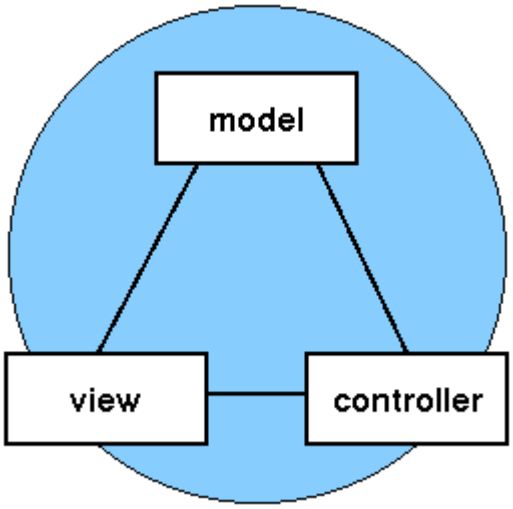
The *model* is the piece that represents the state and low-level behavior of the component. It manages the state and conducts all transformations on that state. The model has no specific knowledge of either its controllers or its views. The system itself maintains links between model and views and notifies the views when the model changes state.

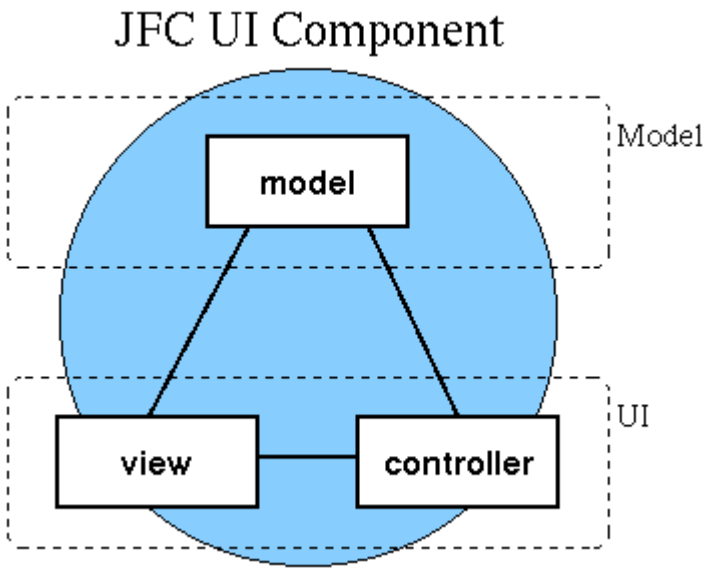
The *view* is the piece that manages the visual display of the state represented by the model. A model can have more than one view, but that is typically not the case in the Swing set.

The *controller* is the piece that manages user interaction with the model. It provides the mechanism by which changes are made to the state of the model.

Using the keyboard key example, the model corresponds to the key's mechanism, and the view and controller correspond to the key's façade.

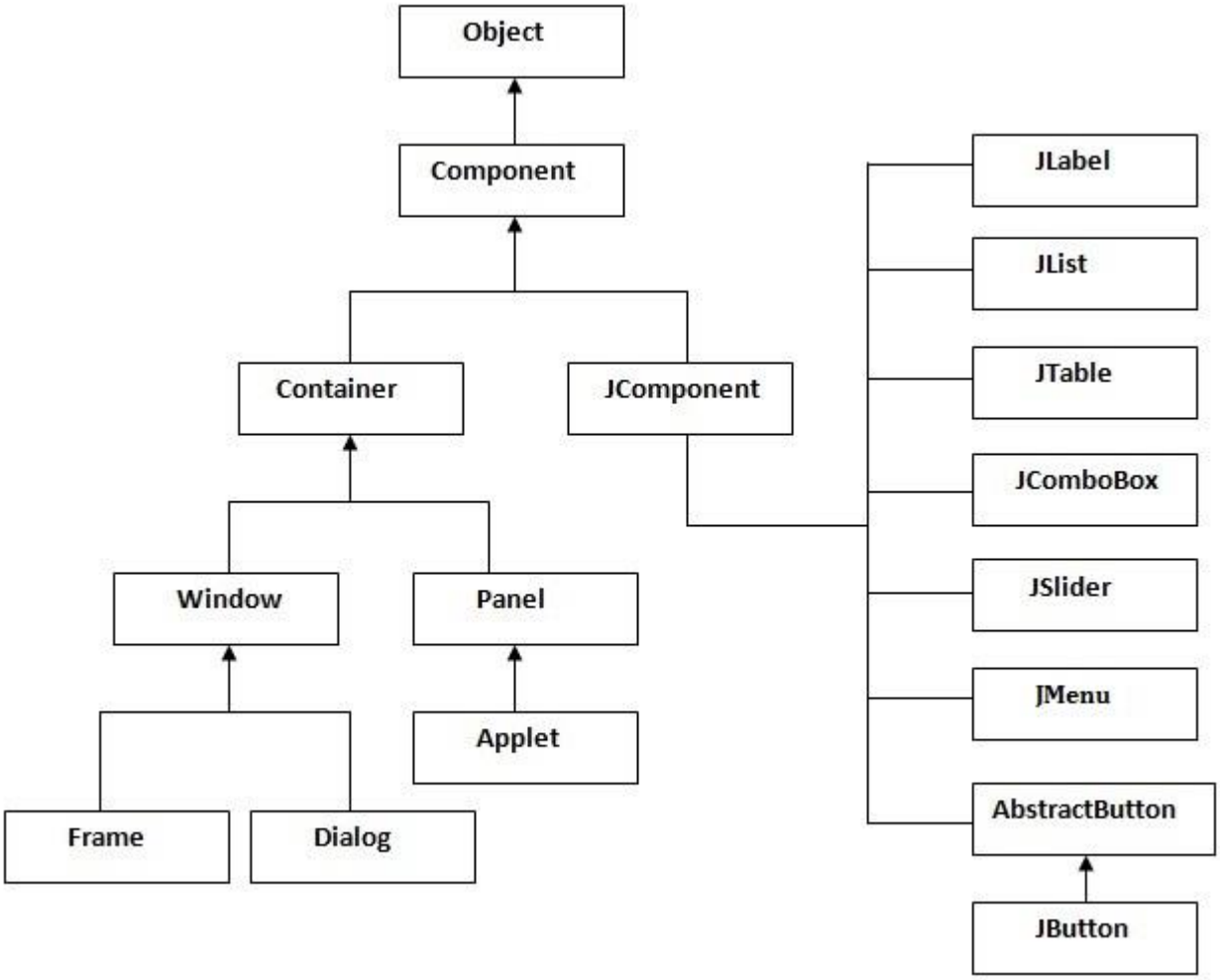
The following figure illustrates how to break a JFC user interface component into a model, view, and controller. Note that the view and controller are combined into one piece, a common adaptation of the basic MVC pattern. They form the user interface for the component





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.

CONTAINERS:

Containers are an integral part of SWING GUI components. A container provides a space where a component can be located. A Container in AWT is a component itself and it provides the capability to add a component to itself. Following are certain noticable points to be considered.

- Sub classes of Container are called as Container. For example, JPanel, JFrame and JWindow.
- Container can add only a Component to itself.
- A default layout is present in each container which can be overridden using **setLayout** method.

SWING Containers

Following is the list of commonly used containers while designed GUI using SWING.

Sr.No.	Container & Description
1	**Panel** JPanel is the simplest container. It provides space in which any other

	component can be placed, including other panels.
2	**Frame** A JFrame is a top-level window with a title and a border.
3	**Window** A JWindow object is a top-level window with no borders and no menubar.

SWING COMPONENTS:

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 AbstractButton 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.

Java JButton Example

```
1.  import javax.swing.*;
2.  public class ButtonExample {
3.  public static void main(String[] args) {
4.      JFrame f=new JFrame("Button Example");
5.      JButton b=new JButton("Click Here");
6.      b.setBounds(50,100,95,30);
7.      f.add(b);
8.      f.setSize(400,400);
9.      f.setLayout(null);
10.     f.setVisible(true);
11. }
12. }
```

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.

```
1.  public class JCheckBox extends JToggleButton implements Accessible
```

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.

Commonly used Methods:

Methods	Description
AccessibleContext getAccessibleContext()	It is used to get the AccessibleContext associated with this JCheckBox.
protected String paramString()	It returns a string representation of this JCheckBox.

Java JCheckBox Example

```
1.  import javax.swing.*;
2.  public class CheckBoxExample
3.  {
4.      CheckBoxExample(){
5.        JFrame f= new JFrame("CheckBox Example");
6.        JCheckBox checkBox1 = new JCheckBox("C++");
7.        checkBox1.setBounds(100,100, 50,50);
```

```
8.              JCheckBox checkBox2 = new JCheckBox("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.          }}
```

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 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.

Java JRadioButton Example

```
1.  import javax.swing.*;
2.  public class RadioButtonExample {
3.  JFrame f;
4.  RadioButtonExample(){
5.  f=new JFrame();
6.  JRadioButton r1=new JRadioButton("A) Male");
7.  JRadioButton r2=new JRadioButton("B) Female");
8.  r1.setBounds(75,50,100,30);
9.  r2.setBounds(75,100,100,30);
10. ButtonGroup bg=new ButtonGroup();
11. bg.add(r1);bg.add(r2);
12. f.add(r1);f.add(r2);
13. f.setSize(300,300);
14. f.setLayout(null);
15. f.setVisible(true);
16. }
17. public static void main(String[] args) {
18.    new RadioButtonExample();
19. }
20. }
```

Java JLabel

The object of JLabel 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. It inherits JComponent class.

JLabel class declaration

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

1. **public class** JLabel **extends** JComponent **implements** SwingConstants, Accessible

Commonly used Constructors:

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty string for the title.
JLabel(String s)	Creates a JLabel instance with the specified text.
JLabel(Icon i)	Creates a JLabel instance with the specified image.
JLabel(String s, Icon i, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment.

Commonly used Methods:

Methods	Description
String getText()	It returns the text string that a label displays.
void setText(String text)	It defines the single line of text this component will display.
void setHorizontalAlignment(int alignment)	It sets the alignment of the label's contents along the X axis.

Icon getIcon()	It returns the graphic image that the label displays.
int getHorizontalAlignment()	It returns the alignment of the label's contents along the X axis.

Java JLabel Example

```
1.  import javax.swing.*;
2.  class LabelExample
3.  {
4.  public static void main(String args[])
5.     {
6.     JFrame f= new JFrame("Label Example");
7.     JLabel l1,l2;
8.     l1=new JLabel("First Label.");
9.     l1.setBounds(50,50, 100,30);
10.    l2=new JLabel("Second Label.");
11.    l2.setBounds(50,100, 100,30);
12.    f.add(l1); f.add(l2);
13.    f.setSize(300,300);
14.    f.setLayout(null);
15.    f.setVisible(true);
16.    }
17.    }
```

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.

Java JTextField Example

```
1.  import javax.swing.*;
2.  class TextFieldExample
3.  {
4.  public static void main(String args[])
5.    {
6.    JFrame f= new JFrame("TextField Example");
7.    JTextField t1,t2;
8.    t1=new JTextField("Welcome to Javatpoint.");
```

```
9.          t1.setBounds(50,100, 200,30);
10.         t2=new JTextField("AWT Tutorial");
11.         t2.setBounds(50,150, 200,30);
12.         f.add(t1); f.add(t2);
13.         f.setSize(400,400);
14.         f.setLayout(null);
15.         f.setVisible(true);
16.         }
17.         }
```

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.

Java JTextArea Example

```
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 JList

The object of JList class represents a list of text items. The list of text items can be set up so that the user can choose either one item or multiple items. It inherits JComponent class.

JList class declaration

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

1. **public class** JList **extends** JComponent **implements** Scrollable, Accessible

Commonly used Constructors:

Constructor	Description
JList()	Creates a JList with an empty, read-only, model.
JList(ary[] listData)	Creates a JList that displays the elements in the specified array.
JList(ListModel<ary> dataModel)	Creates a JList that displays elements from the specified, non-null, model.

Commonly used Methods:

Methods	Description
Void addListSelectionListener(ListSelectionListener listener)	It is used to add a listener to the list, to be notified each time a change to the selection occurs.
int getSelectedIndex()	It is used to return the smallest selected cell index.
ListModel getModel()	It is used to return the data model that holds a list of items displayed by the JList component.
void setListData(Object[] listData)	It is used to create a read-only ListModel from an array of objects.

Java JList Example

```
1.  import javax.swing.*;
2.  public class ListExample
3.  {
4.      ListExample(){
5.        JFrame f= new JFrame();
6.        DefaultListModel<String> l1 = new DefaultListModel<>();
7.         l1.addElement("Item1");
8.         l1.addElement("Item2");
9.         l1.addElement("Item3");
10.        l1.addElement("Item4");
11.        JList<String> list = new JList<>(l1);
12.        list.setBounds(100,100, 75,75);
13.        f.add(list);
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.    }}
```

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.

JComboBox class declaration

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

```
1.  public class JComboBox extends JComponent implements ItemSelectable, List
```

DataListener, ActionListener, Accessible

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.

Java JComboBox Example

```
1.  import javax.swing.*;
2.  public class ComboBoxExample {
3.  JFrame f;
4.  ComboBoxExample(){
5.     f=new JFrame("ComboBox Example");
6.     String country[]={"India","Aus","U.S.A","England","Newzealand"};
7.     JComboBox cb=new JComboBox(country);
8.     cb.setBounds(50, 50,90,20);
9.     f.add(cb);
```

```
10.             f.setLayout(null);
11.             f.setSize(400,500);
12.             f.setVisible(true);
13.         }
14.         public static void main(String[] args) {
15.             new ComboBoxExample();
16.         }
17.         }
```

Java JMenuBar, JMenu and JMenuItem

The JMenuBar class is used to display menubar on the window or frame. It may have several menus.

The object of JMenu class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class.

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

JMenuBar class declaration

```
1.          public class JMenuBar extends JComponent implements MenuElement, Access
    ible
```

JMenu class declaration

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

JMenuItem class declaration

```
1.          public class JMenuItem extends AbstractButton implements Accessible, Menu
    Element
```

Java JMenuItem and JMenu Example

```
1.          import javax.swing.*;
2.          class MenuExample
3.          {
4.                  JMenu menu, submenu;
```

```
5.              JMenuItem i1, i2, i3, i4, i5;
6.              MenuExample(){
7.              JFrame f= new JFrame("Menu and MenuItem Example");
8.              JMenuBar mb=new JMenuBar();
9.              menu=new JMenu("Menu");
10.             submenu=new JMenu("Sub Menu");
11.             i1=new JMenuItem("Item 1");
12.             i2=new JMenuItem("Item 2");
13.             i3=new JMenuItem("Item 3");
14.             i4=new JMenuItem("Item 4");
15.             i5=new JMenuItem("Item 5");
16.             menu.add(i1); menu.add(i2); menu.add(i3);
17.             submenu.add(i4); submenu.add(i5);
18.             menu.add(submenu);
19.             mb.add(menu);
20.             f.setJMenuBar(mb);
21.             f.setSize(400,400);
22.             f.setLayout(null);
23.             f.setVisible(true);
24.     }
25.     public static void main(String args[])
26.     {
27.     new MenuExample();
28.     }}
```

LayoutManagers:

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

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

9. javax.swing.SpringLayout etc.

BorderLayout:

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

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

Constructors of BorderLayout class:

- **BorderLayout():** creates a border layout but with no gaps between the components.
- **JBorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

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

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

   JButton b1=new JButton("NORTH");;
   JButton b2=new JButton("SOUTH");;
   JButton b3=new JButton("EAST");;
   JButton b4=new JButton("WEST");;
   JButton b5=new JButton("CENTER");;

   f.add(b1,BorderLayout.NORTH);
   f.add(b2,BorderLayout.SOUTH);
   f.add(b3,BorderLayout.EAST);
   f.add(b4,BorderLayout.WEST);
   f.add(b5,BorderLayout.CENTER);
```

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

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.

```
1.  import java.awt.*;
2.  import javax.swing.*;
3.
4.  public class MyGridLayout{
5.  JFrame f;
6.  MyGridLayout(){
7.     f=new JFrame();
8.
9.     JButton b1=new JButton("1");
10.    JButton b2=new JButton("2");
11.    JButton b3=new JButton("3");
12.    JButton b4=new JButton("4");
13.    JButton b5=new JButton("5");
14.       JButton b6=new JButton("6");
15.       JButton b7=new JButton("7");
16.    JButton b8=new JButton("8");
17.       JButton b9=new JButton("9");
```

```
18.
19.             f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
20.             f.add(b6);f.add(b7);f.add(b8);f.add(b9);
21.
22.             f.setLayout(new GridLayout(3,3));
23.             //setting grid layout of 3 rows and 3 columns
24.
25.             f.setSize(300,300);
26.             f.setVisible(true);
27.         }
28.         public static void main(String[] args) {
29.             new MyGridLayout();
30.         }
31.         }
```

FlowLayout

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

Fields of FlowLayout class:

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

Constructors of FlowLayout class:

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

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

```
3.
4.          public class MyFlowLayout{
5.          JFrame f;
6.          MyFlowLayout(){
7.              f=new JFrame();
8.
9.              JButton b1=new JButton("1");
10.             JButton b2=new JButton("2");
11.             JButton b3=new JButton("3");
12.             JButton b4=new JButton("4");
13.             JButton b5=new JButton("5");
14.
15.             f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
16.
17.             f.setLayout(new FlowLayout(FlowLayout.RIGHT));
18.             //setting flow layout of right alignment
19.
20.             f.setSize(300,300);
21.             f.setVisible(true);
22.         }
23.         public static void main(String[] args) {
24.             new MyFlowLayout();
25.         }
26.         }
```

BoxLayout class:

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

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

Fields of BoxLayout class:

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

4. **public static final int PAGE_AXIS**

Constructor of BoxLayout class:

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

```
1.  import java.awt.*;
2.  import javax.swing.*;
3.
4.  public class BoxLayoutExample1 extends Frame {
5.   Button buttons[];
6.
7.   public BoxLayoutExample1 () {
8.    buttons = new Button [5];
9.
10.   for (int i = 0;i<5;i++) {
11.     buttons[i] = new Button ("Button " + (i + 1));
12.     add (buttons[i]);
13.    }
14.
15.  setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));
16.  setSize(400,400);
17.  setVisible(true);
18.  }
19.
20.  public static void main(String args[]){
21.  BoxLayoutExample1 b=new BoxLayoutExample1();
22.  }
23.  }
```

Displaying graphics in swing:

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.

```
1.  import java.awt.*;
2.  import javax.swing.JFrame;
3.
4.  public class DisplayGraphics extends Canvas{
5.
```

```
6.          public void paint(Graphics g) {
7.             g.drawString("Hello",40,40);
8.             setBackground(Color.WHITE);
9.             g.fillRect(130, 30,100, 80);
10.            g.drawOval(30,130,50, 60);
11.            setForeground(Color.RED);
12.            g.fillOval(130,130,50, 60);
13.            g.drawArc(30, 200, 40,50,90,60);
14.            g.fillArc(30, 130, 40,50,180,40);
15.
16.         }
17.            public static void main(String[] args) {
18.            DisplayGraphics m=new DisplayGraphics();
19.            JFrame f=new JFrame();
20.            f.add(m);
21.            f.setSize(400,400);
22.            //f.setLayout(null);
23.            f.setVisible(true);
24.         }
25.
26.      }
```

BorderLayout (LayoutManagers):

LayoutManagers:

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

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

BorderLayout:

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

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

Constructors of BorderLayout class:

- **BorderLayout():** creates a border layout but with no gaps between the components.
- **JBorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

Example of BorderLayout class:


```
1.  import java.awt.*;
2.  import javax.swing.*;
3.
4.  public class Border {
5.  JFrame f;
6.  Border(){
7.      f=new JFrame();
8.
9.      JButton b1=new JButton("NORTH");;
10.     JButton b2=new JButton("SOUTH");;
11.     JButton b3=new JButton("EAST");;
12.     JButton b4=new JButton("WEST");;
13.     JButton b5=new JButton("CENTER");;
14.
15.     f.add(b1,BorderLayout.NORTH);
16.     f.add(b2,BorderLayout.SOUTH);
17.     f.add(b3,BorderLayout.EAST);
18.     f.add(b4,BorderLayout.WEST);
19.     f.add(b5,BorderLayout.CENTER);
20.
21.     f.setSize(300,300);
22.     f.setVisible(true);
23. }
24. public static void main(String[] args) {
25.     new Border();
26. }
27. }
```

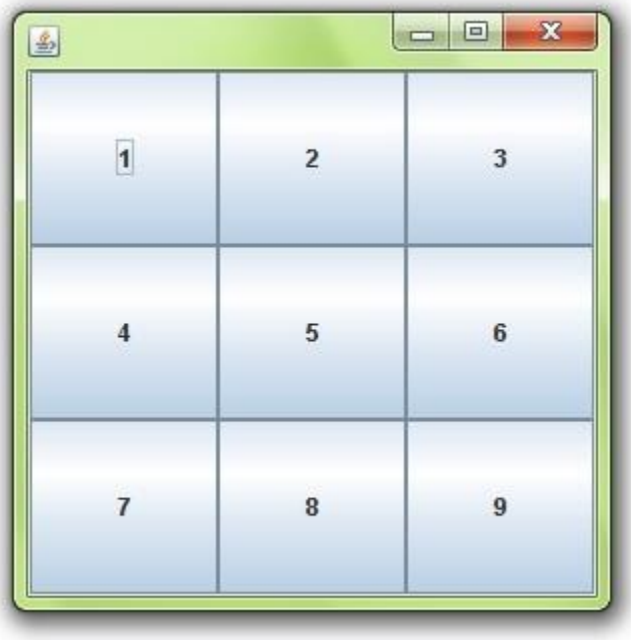
GridLayout

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

Constructors of GridLayout class:

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

Example of GridLayout class:




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

```
11.    JButton b3=new JButton("3");
12.    JButton b4=new JButton("4");
13.    JButton b5=new JButton("5");
14.       JButton b6=new JButton("6");
15.       JButton b7=new JButton("7");
16.    JButton b8=new JButton("8");
17.       JButton b9=new JButton("9");
18.
19.    f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
20.    f.add(b6);f.add(b7);f.add(b8);f.add(b9);
21.
22.    f.setLayout(new GridLayout(3,3));
23.    //setting grid layout of 3 rows and 3 columns
24.
25.    f.setSize(300,300);
26.    f.setVisible(true);
27. }
28. public static void main(String[] args) {
29.    new MyGridLayout();
30. }
31. }
```

FlowLayout

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

Fields of FlowLayout class:

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

Constructors of FlowLayout class:

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

Example of FlowLayout class:


```
1.  import java.awt.*;
2.  import javax.swing.*;
3.
4.  public class MyFlowLayout{
5.  JFrame f;
6.  MyFlowLayout(){
7.      f=new JFrame();
8.
9.      JButton b1=new JButton("1");
10.     JButton b2=new JButton("2");
11.     JButton b3=new JButton("3");
12.     JButton b4=new JButton("4");
13.     JButton b5=new JButton("5");
14.
15.     f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
16.
17.     f.setLayout(new FlowLayout(FlowLayout.RIGHT));
18.     //setting flow layout of right alignment
19.
20.     f.setSize(300,300);
21.     f.setVisible(true);
22. }
23. public static void main(String[] args) {
24.     new MyFlowLayout();
25. }
26. }
```

BoxLayout class:

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

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

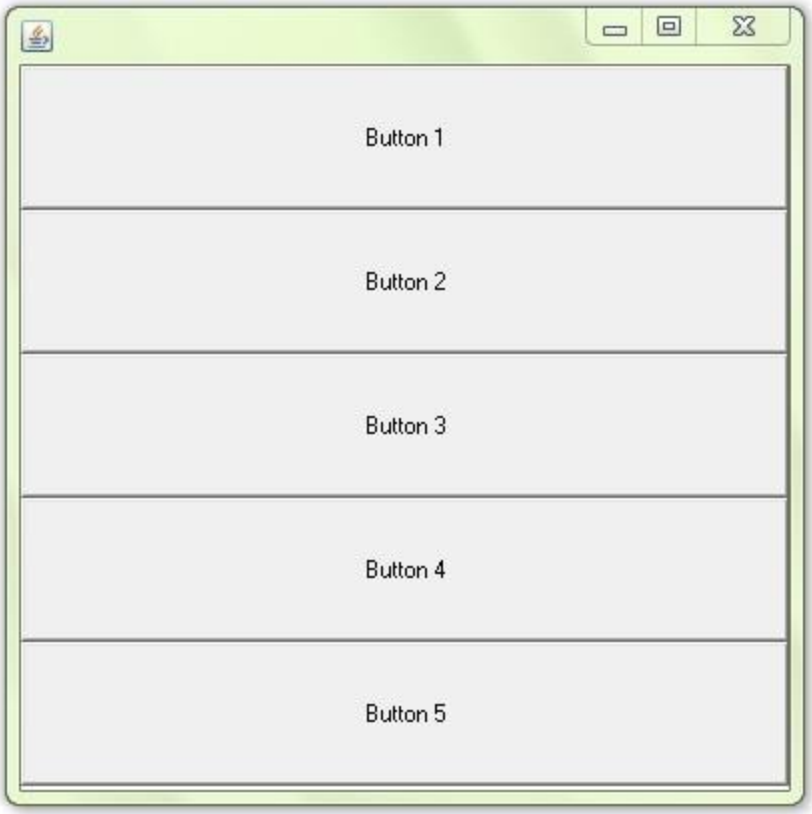
Fields of BoxLayout class:

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

Constructor of BoxLayout class:

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

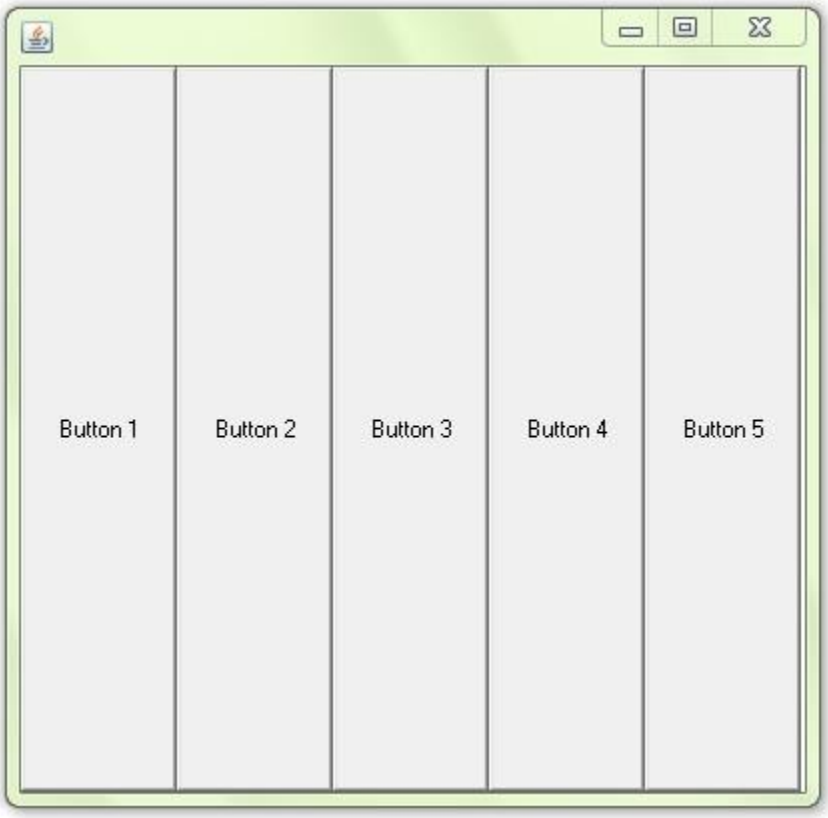
Example of BoxLayout class with Y-AXIS:




```
1. import java.awt.*;
2. import javax.swing.*;
3.
4. public class BoxLayoutExample1 extends Frame {
5.  Button buttons[];
6.
7.  public BoxLayoutExample1 () {
8.    buttons = new Button [5];
9.
10.   for (int i = 0;i<5;i++) {
11.     buttons[i] = new Button ("Button " + (i + 1));
12.     add (buttons[i]);
13.    }
14.
15. setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));
16. setSize(400,400);
17. setVisible(true);
18. }
19.
20. public static void main(String args[]){
21. BoxLayoutExample1 b=new BoxLayoutExample1();
22. }
23. }
```

download this example

Example of BoxLayout class with X-AXIS:




```
1. import java.awt.*;
2. import javax.swing.*;
3.
4. public class BoxLayoutExample2 extends Frame {
5.  Button buttons[];
6.
7.  public BoxLayoutExample2() {
8.    buttons = new Button [5];
9.
10.   for (int i = 0;i<5;i++) {
11.     buttons[i] = new Button ("Button " + (i + 1));
12.     add (buttons[i]);
13.    }
14.
15. setLayout (new BoxLayout(this, BoxLayout.X_AXIS));
16. setSize(400,400);
17. setVisible(true);
18. }
19.
20. public static void main(String args[]){
```

```
21. BoxLayoutExample2 b=new BoxLayoutExample2();
22. }
23. }
```

CardLayout class

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

Constructors of CardLayout class:

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

Commonly used methods of CardLayout class:

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

Example of CardLayout class:


```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.
4.  import javax.swing.*;
5.
6.  public class CardLayoutExample extends JFrame implements ActionListener{
7.  CardLayout card;
8.  JButton b1,b2,b3;
9.  Container c;
10.     CardLayoutExample(){
11.
12.         c=getContentPane();
13.         card=new CardLayout(40,30);
14. //create CardLayout object with 40 hor space and 30 ver space
15.         c.setLayout(card);
16.
17.         b1=new JButton("Apple");
18.         b2=new JButton("Boy");
19.         b3=new JButton("Cat");
20.         b1.addActionListener(this);
21.         b2.addActionListener(this);
22.         b3.addActionListener(this);
23.
24.         c.add("a",b1);c.add("b",b2);c.add("c",b3);
25.
```

```
    }
    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);
    }
}
```

Unit 2 Swings

Contents

- Swings
- Two key Swing features
- Components and containers

- The Abstract Window Toolkit (AWT) is the part of Java, used for creating user interfaces and painting graphics and images.

- AWT components are derived from the java.awt.Component class.

- The Java Foundation Classes (JFC) consist of five major parts: AWT, Swing, Accessibility, Java 2D, and Drag and Drop.

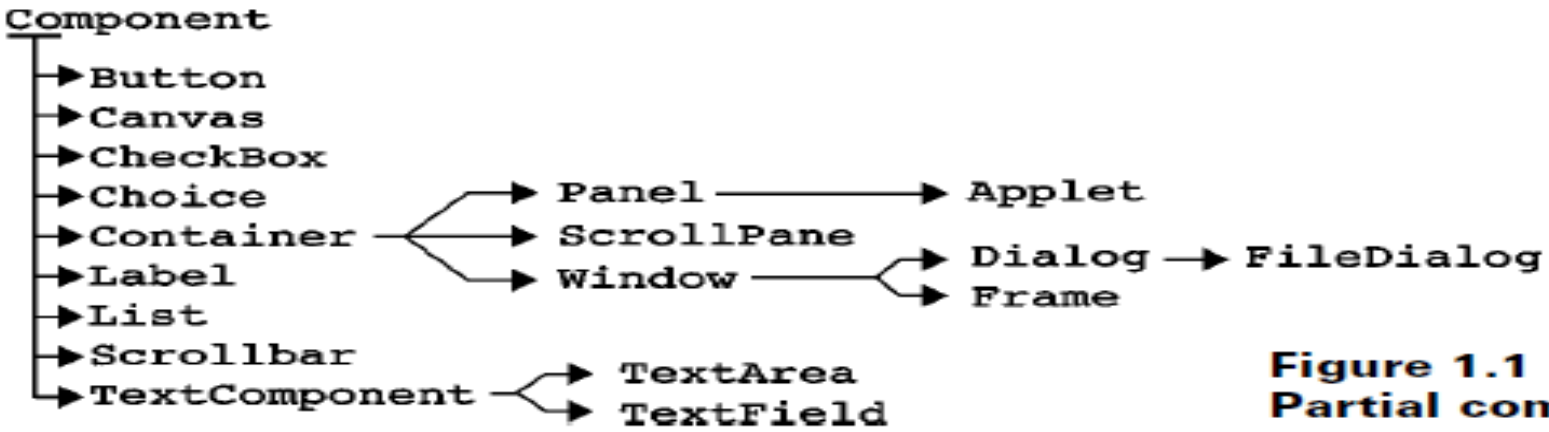


Figure 1.1
Partial comp

- **Limitation of AWT**:

- it translates its various visual components into their corresponding, platform-specific equivalents.

-It means that the look and feel of a component is defined by the platform, not by java.

- it uses the native code resources, because of variations between operating system,a component might look differently on different platforms.

- AWT components are referred as heavyweight components.

- heavyweight components have some retrictions.(it will be always rectangular and opaque).

- Swing was introduced in the year of 1997.

- Swing is built on the foundation of the AWT.

- Swing also uses the same event handling mechanism as the AWT.

Two key Swing Features

- Swing components are Lightweight
- Swing supports a Pluggable Look and Feel

- **Swing components are Lightweight**
- This means that they are written entirely in java and do not map directly to platform-specific peers.

- Lightweight components are rendered using graphics primitives.

- The look and feel of each component is determined by swing not by the underlying operating system.

- **Swing supports a Pluggable Look and Feel**
- Because each swing component is rendered by java code,the look and feel of a component is under the control of swing.

- The look and feel of a component can be separated from the logic of the component.

- This has got an advantage that a component is rendered without affecting any of its other aspects.

- It is possible to "plug in" a new look and feel for any given component.

Components and Containers

- A swing GUI consists of two key items:

Components and Containers

- A component is an independent visual control, such as push button,text etc.
- A container holds a group of components.
- To display a component, it must be held within a container.
- Thus, all swing GUIs will have at least one container.
- A container can hold other containers.
- This enables swing to define what is called a containment hierarchy, at the top of which must be a top-level container.

Components

- Swing components are derived from the JComponent class.
- Jcomponent provides the functionality that is common to all components.
- Jcomponent inherits the AWT classes Container and Component.
- Thus, a swing component is built on and compatible with an AWT component.

- All of swing's components are represented by classes defined within package javax.swing.
- All the component classes begin with the letter J.
- For e.g. the class for a label is JLabel, the class for a push button is JButton.

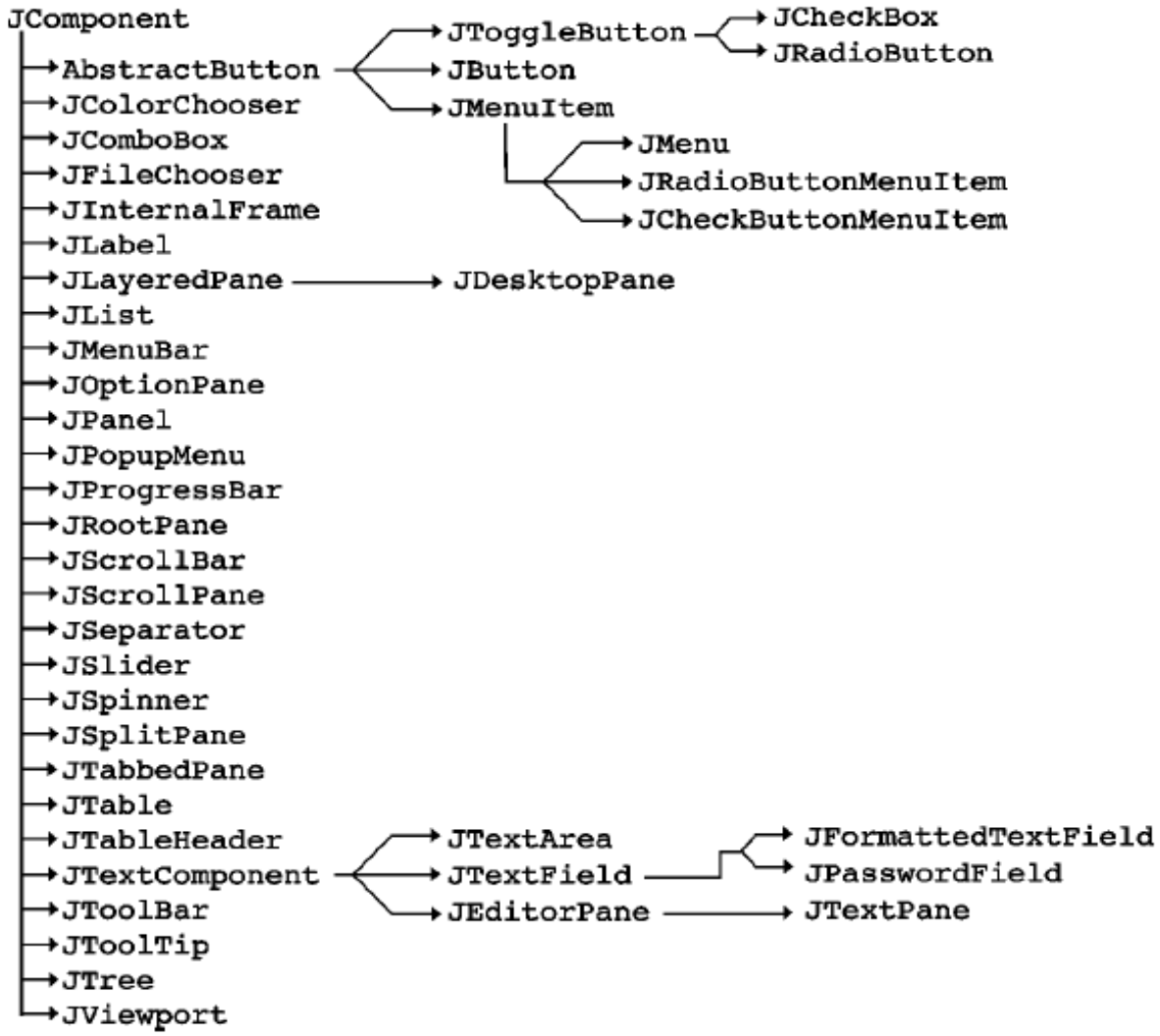


Figure 1.2 Partial `JComponent` hierarchy

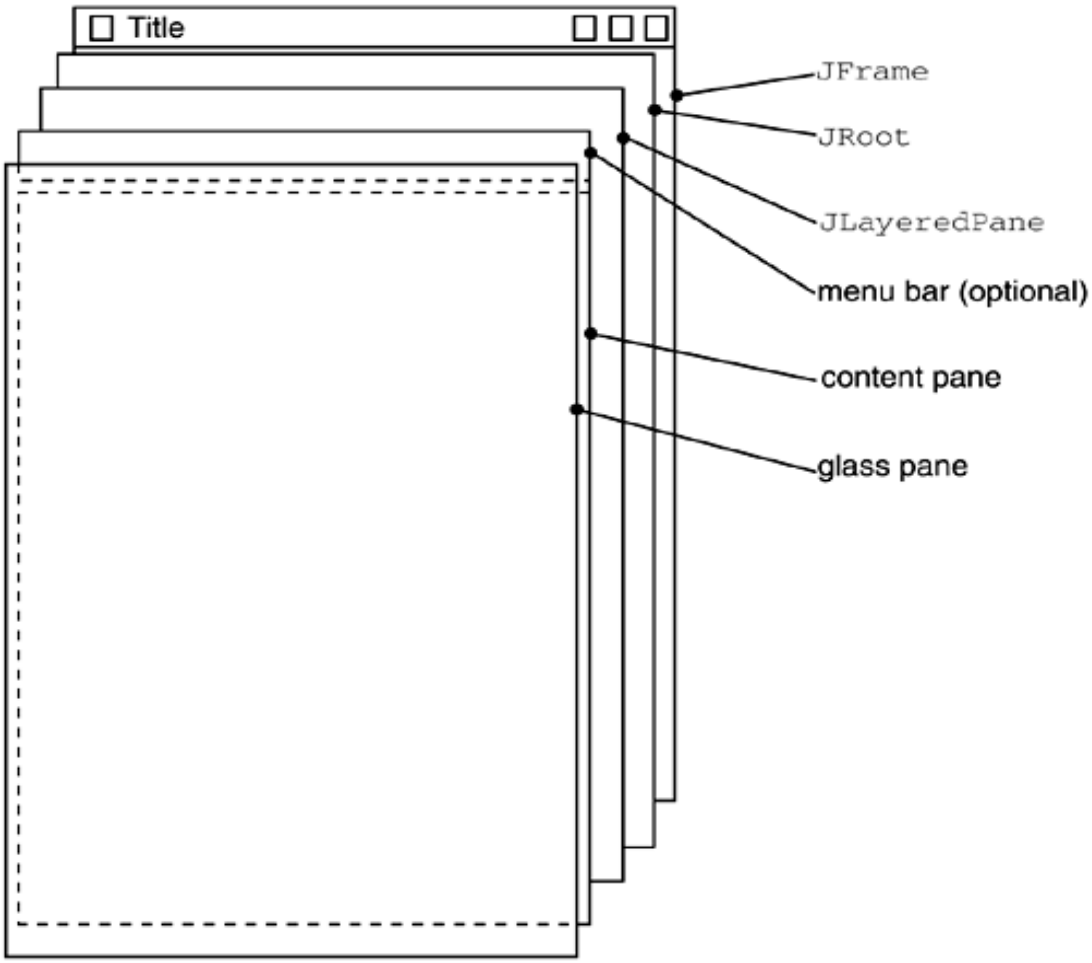
Containers

- Swing defines two types of containers.
- The first are top-level containers: JFrame,JApplet,JWindow and JDialog.
- These containers do not inherit JComponent.
- These top-level containers are heavyweight.
- A top-level container must be at the top of a containment hierarchy.
- A top-level container is not contained within any other container.
- Every containment hierarchy must begin with a top –level container.

- The second type of containers are lightweight containers.
  - it inherits JComponent. An example of a lightweight container is JPanel, it's a general purpose container.
  - it is used to organize and manage groups of related components because a lightweight container can be contained within another container.

Top-level Container Panes

- Each top-level container defines a set of panes.
- At the top of the hierarchy is an instance of JRootPane.
- JRootPane is a lightweight container which is used to manage the other panes.
- The panes that comprise the root pane are called the glass pane, the content pane and the layered pane.
- The glass pane is the top level pane. It sits above and completely covers all other panes.
- It is used to manage mouse events that affect the entire container or to paint over any other component.
- The layered pane is an instance of JLayeredPane.
- It allows components to be given a depth value.
- This value determines which component overlays another.
- It holds the content pane and the (optional) menu bar.
- The glass pane and the layered pane are used to serve important purposes and mostly these occur behind the scene.
- The content pane is the one where the applications will interact mostly.
- This is the pane to which the visual components are added.
- For e.g. if we add a button to a top-level container, we add it to the content pane.
- By default, the content pane is an opaque instance of JPanel.



A Simple Swing Application:

There are two types of Java programs in which Swing is typically used. The first is a desktop application. The second is the applet.

It uses two Swing components: JFrame and JLabel. JFrame is the top-level container that is commonly used for Swing applications. JLabel is the Swing component that creates a label,which is a component that displays information. The label is Swing's simplest component because it is passive. That is, a label does not respond to user input. It just displays output. The program uses a JFrame container to hold an instance of a JLabel. The label displays a short txt message.

```
Import javax.swing.*;
Class SwingDemo {
  Swingdemo() {
    JFrame jfrm=new JFrame("A simple Swing Application");
    jfrm.setSize(275,100);
    jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JLabel jlab=new JLabel("Swing means powerful GUIs.");
```

```
        jfrm.add(jlab);
        jfrm.setVisible(true);
    }
    public static void main(string args[])
    {
         SwingUtilities.invokeLater(new Runnable() {
            Public void run() {
                 New SwingDemo();
              }
          });
    }
}
```

The program begins by importing javax.swing. this package contains the component and models defined by Swing.

Next ,the program declares Swing Demo lass and a constructor for that class. The constructor is where most of the action of the program occurs.

The title of the window is passed to the constructor. Next, the window is sized using this statement:

Jfrm.setSize(275,100);

The setSize() method(which is inherited by JFrame from the AWT class Component) sets the dimensions of the window, which are specified in pixels.

Its general form is:

Void setSize(int width,int height)

In this example the width of the window is set to 275 and the height is set to 100.

INTRODUCTION TO JAVA BEANS

Software components are self-contained software units developed according to the motto "Developed them once, run and reused them everywhere". Or in other words, reusability is the main concern behind the component model. A software component is a reusable object that can be plugged into any target software application. You can develop software components using various programming languages, such as C, C++, Java, and Visual Basic.

- A "Bean" is a reusable software component model based on sun's java bean specification that can be manipulated visually in a builder tool.
- The term software component model describe how to create and use reusable software components to build an application
- Builder tool is nothing but an application development tool which lets you both to create new beans or use existing beans to create an application.
- To enrich the software systems by adopting component technology JAVA came up with the concept called Java Beans.
- Java provides the facility of creating some user defined components by means of Bean programming.
- We create simple components using java beans.
- We can directly embed these beans into the software.

Advantages of Java Beans:

- The java beans possess the property of "Write once and run anywhere".
- Beans can work in different local platforms.
- Beans have the capability of capturing the events sent by other objects and vice versa enabling object communication.
- The properties, events and methods of the bean can be controlled by the application developer.(ex. Add new properties)
- Beans can be configured with the help of auxiliary software during design time.(no hassle at runtime)
- The configuration setting can be made persistent.(reused)
- Configuration setting of a bean can be saved in persistent storage and restored later.

What can we do/create by using JavaBean?

There is no restriction on the capability of a Bean.

- It may perform a simple function, such as checking the spelling of a document, or a complex function, such as forecasting the performance of a stock portfolio. A Bean may be visible to an end user. One example of this is a button on a graphical user interface.

- Software to generate a pie chart from a set of data points is an example of a Bean that can execute locally.
- Bean that provides real-time price information from a stock or commodities exchange.

Definition of a builder tool:

Builder tools allow a developer to work with JavaBeans in a convenient way. By examining a JavaBean by a process known as Introspection, a builder tool exposes the discovered features of the JavaBean for visual manipulation. A builder tool maintains a list of all JavaBeans available. It allows you to compose the Bean into applets, application, servlets and composite components (e.g. a JFrame), customize its behavior and appearance by modifying its properties and connect other components to the event of the Bean or vice versa.

Some Examples of Application Builder tools:

TOOL	VENDOR	DESCRIPTION
Java Workshop2.0	Sun MicroSystems., Inc.,	Complete IDE that support applet, application and bean development
Visual age for java	IBM	Bean Oriented visual development toolset.
Jbuilder	Borland Inc.	Suit of bean oriented java development tool
Beans Development Kit	SunMicroSystems., Inc.,	Supports only Beans development

JavaBeans basic rules

A JavaBean should:

- ✓ be public
- ✓ implement the Serializable interface
- ✓ have a no-arg constructor
- ✓ be derived from javax.swing.JComponent or java.awt.Component if it is visual

The classes and interfaces defined in the java.beans package enable you to create JavaBeans. The Java Bean components can exist in one of the following three phases of development:

- Construction phase
- Build phase
- Execution phase

It supports the standard **component architecture** features of

- ✓ Properties
- ✓ Events
- ✓ Methods
- ✓ Persistence.

In addition Java Beans provides support for

- ✓ Introspection (Allows Automatic Analysis of a java beans)
- ✓ Customization (To make it easy to configure a java beans component)

Elements of a JavaBean:

- **Properties:** It is similar to instance variables. A bean *property* is a named attribute of a bean that can affect its behavior or appearance. Examples of bean properties include color, label, font, font size, and display size.
- **Methods**
  - **Same as normal Java methods.**
  - **Every property should have accessor (get) and mutator (set) method.**
  - All Public methods can be identified by the introspection mechanism.
  - There is no specific naming standard for these methods.
- **Events**
  - **Similar to Swing/AWT event handling.**

The JavaBean Component Specification:

Customization: Is the ability of JavaBean to allow its properties to be changed in build and execution phase.
Persistence: Is the ability of JavaBean to save its state to disk or storage device and restore the saved state when the JavaBean is reloaded.
Communication: Is the ability of JavaBean to notify change in its properties to other JavaBeans or the container.

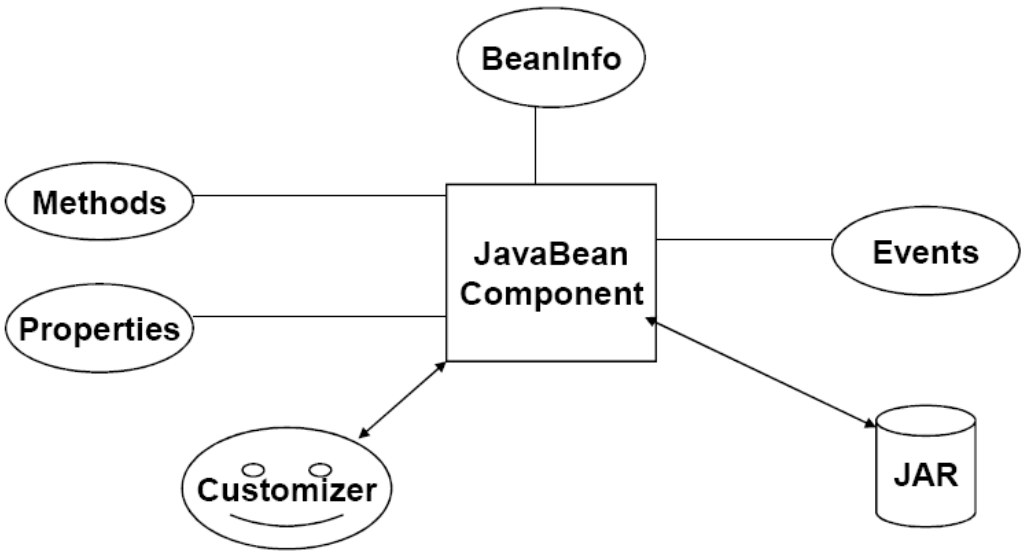
Introspection: Is the ability of a JavaBean to allow an external application to query the properties, methods, and events supported by it.

Services of JavaBean Components

Builder support: Enables you to create and group multiple JavaBeans in an application.
Layout: Allows multiple JavaBeans to be arranged in a development environment.
Interface publishing: Enables multiple JavaBeans in an application to communicate with each other.
Event handling: Refers to firing and handling of events associated with a JavaBean.
Persistence: Enables you to save the last state of JavaBean.

Features of a JavaBean

- Support for "introspection" so that a builder tool can analyze how a bean works.
- Support for "customization" to allow the customization of the appearance and behavior of a bean.
- Support for "events" as a simple communication metaphor than can be used to connect up beans.
- Support for "properties", both for customization and for programmatic use.
- Support for "persistence", so that a bean can save and restore its customized state.



Beans Development Kit

It is a development environment to create, configure, and test Java Beans. The features of BDK environment are:

- Provides a GUI to create, configure, and test JavaBeans.
- Enables you to modify JavaBean properties and link multiple JavaBeans in an application using BDK.
- Provides a set of sample JavaBeans.
- Enables you to associate pre-defined events with sample JavaBeans.

Identifying BDK Components

- Execute the run.bat file of BDK to start the BDK development environment.
- The components of BDK development environment are:
  - **ToolBox**
  - **BeanBox**
  - **Properties**
  - **Method Tracer**

ToolBox window: Lists the sample JavaBeans of BDK. The following figure shows the **ToolBox** window:

BeanBox window:

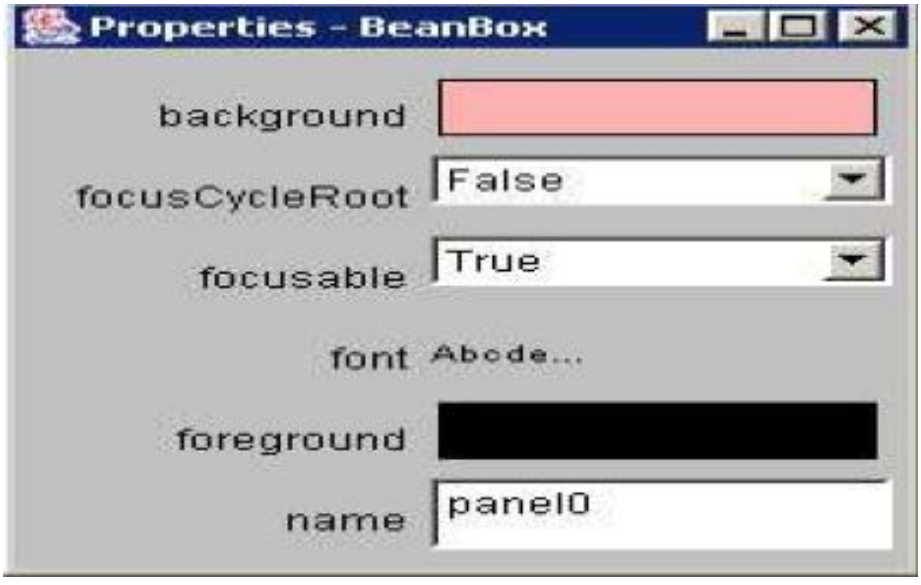
It is a workspace for creating the layout of JavaBean application.

- The following figure shows the **BeanBox** window:

Properties window:

It is used to displays all the exposed properties of a JavaBean. You can modify JavaBean properties in the properties window.

The following figure shows the **Properties** window:



Method Tracer window:

It is used to displays the debugging messages and method calls for a JavaBean application.

The following figure shows the **Method Tracer** window:

Steps to Develop a User-Defined JavaBean:

1. Create a directory for the new bean
2. Create the java bean source file(s)
3. Compile the source file(s)
4. Create a manifest file
5. Generate a JAR file
6. Start BDK
7. Load Jar file
8. Test.

1. Create a directory for the new bean

Create a directory/folder like C:\Beans

2. Create bean source file - MyBean.java

```
import java.awt.*;
public class MyBean extends Canvas
{
 public MyBean()
 {
        setSize(70,50);
        setBackground(Color.green);
  }

}
```

3. Compile the source file(s)

C:\Beans >Javac MyBean.java

4. Create a manifest file

- The manifest file for a JavaBean application contains a list of all the class files that make up a JavaBean.
- The entry in the manifest file enables the target application to recognize the JavaBean classes for an application.
- For example, the entry for the MyBean JavaBean in the manifest file is as shown:

Name: MyBean.class
Java-Bean: true

Note: write that 2 lines code in the notepad and save that file as MyBean.mf

The rules to create a manifest file are:

- Press the Enter key after typing each line in the manifest file.
- Leave a space after the colon.
- Type a hyphen between Java and Bean.
- No blank line between the Name and the Java-Bean entry.

5. Generate a JAR file

Syntax for creating jar file using manifest file

C:\Beans **>jar cfm MyBean.jar MyBean.mf MyBean.class**

JAR file:

- JAR file allows you to efficiently deploy a set of classes and their associated resources.
- JAR file makes it much easier to deliver, install, and download. It is compressed.

Java Archive File

- The files of a JavaBean application are compressed and grouped as JAR files to reduce the size and the download time of the files.
- The syntax to create a JAR file from the command prompt is:
  - jar <options> <file_names>
- The file_names is a list of files for a JavaBean application that are stored in the JAR file.

The various options that you can specify while creating a JAR file are:

- c: Indicates the new JAR file is created.
- f: Indicates that the first file in the file_names list is the name of the JAR file.
- m: Indicates that the second file in the file_names list is the name of the manifest file.
- t: Indicates that all the files and resources in the JAR file are to be displayed in a tabular format.
- v: Indicates that the JAR file should generate a verbose output.
- x: Indicates that the files and resources of a JAR file are to be extracted.
- o: Indicates that the JAR file should not be compressed.
- m: Indicates that the manifest file is not created.

6. Start BDK

Go to->
C:\bdk1_1\beans\beanbox

Click on **run.bat** file. When we click on run.bat file the BDK software automatically started.

7. Load Jar file

Go to
Beanbox->File->Load jar. Here we have to select our created jar file when we click on ok, our bean (user defined) MyBean appear in the ToolBox.

8. Test our created user defined bean

Select the MyBean from the ToolBox when we select that bean one + simple appear then drag that Bean in to the Beanbox.
If you want to apply events for that bean, now we apply the events for that Bean.

Introspection:

- Introspection can be defined as the technique of obtaining information about bean properties, events and methods.
- Basically introspection means analysis of bean capabilities.
- Introspection is the automatic process by which a builder tool finds out which properties, methods, and events a bean supports.
- Introspection describes how methods, properties, and events are discovered in the beans that you write.
- This process controls the publishing and discovery of bean operations and properties
- Without introspection, the JavaBeans technology could not operate.

BDK Introspection:

- Allows automatic analysis of a java beans component
- Enables a builder tool to analyze how a bean works.

(Or)

- A mechanism that allows classes to publish the operations and properties they support and a mechanism to support the discovery of such mechanism.
- Introspection can be defined as the technique of obtaining information about bean properties, events and methods.
- Basically introspection means analysis of bean capabilities.

There are two ways in which the developer of a Bean can indicate which of its properties, events, and methods should be exposed by a builder tool. With the first method, simple naming conventions are used. These allow the introspection mechanisms to infer information about a Bean. In the second way, an additional class is provided that explicitly supplies this information.

Design patterns for JavaBean Properties:-

- A property is a subset of a Bean's state.
- A bean *property* is a named attribute of a bean that can affect its behavior or appearance. Examples of bean properties include color, label, font, font size, and display size.
- Properties are the private data members of the JavaBean classes.
- Properties are used to accept input from an end user in order to customize a JavaBean.
- Properties can retrieve and specify the values of various attributes, which determine the behavior of a JavaBean.
- Types of JavaBeans Properties
    - Simple properties
    - Boolean properties
    - Indexed properties

Simple Properties:

Simple properties refer to the private variables of a JavaBean that can have only a single value. Simple properties are retrieved and specified using the get and set methods respectively.

- A read/write property has both of these methods to access its values. The **get method** used to read the value of the property .The **set method** that sets the value of the property.
- The setXXX() and getXXX() methods are the heart of the java beans properties mechanism. This is also called getters and setters. These accessor methods are used to set the property.

The syntax of get method is:

public return_type get<PropertyName>()
public T getN();
public void setN(T arg)

where N is the name of the property and T is its type

Ex:

```
public double getDepth()
{
return depth;
}
```

Read only property has only a get method.

The syntax of set method is:

public void set<PropertyName>(data_type value)

Ex:

```
public void setDepth(double d)
{
Depth=d;
}
```

Write only property has only a set method.

Boolean Properties:

A Boolean property is a property which is used to represent the values True or False. Have either of the two values, TRUE or FALSE.

It can identify by the following methods:

Let N be the name of the property and T be the type of the value then

public boolean isN();
public void setN(boolean parameter);
public Boolean getN();

- public boolean is<PropertyName>()
- public boolean get<PropertyName>()

First or second pattern can be used to retrieve the value of a Boolean.

- public void set<PropertyName>(boolean value)

For getting the values isN() and getN() methods are used and for setting the Boolean values setN() method is used.

Ex:

```
public boolean dotted=false;
public boolean isDotted()
{
return dotted;
}
public void setDotted(boolean dotted)
{
this.dotted=dotted;
}
```

Indexed Properties:

Indexed Properties are consists of multiple values. If a simple property can hold an array of value they are no longer called simple but instead indexed properties. The method's signature has to be adapted accordingly. An indexed property may expose set/get methods to read/write one element in the array (so-called 'index getter/setter') and/or so-called 'array getter/setter' which read/write the entire array. Indexed Properties enable you to set or retrieve the values from an array of property values. Indexed Properties are retrieved using the following get methods:

Syntax: public int[] get<PropertyName>()

Ex:

```
private double data[];
public double getData(int index)
{
return data[index];
}
```

Syntax: public property_datatype get<PropertyName>(int index)

Ex:

```
public void setData(int index,double value)
{
Data[index]=value;
}
```

Indexed Properties are specified using the following set methods:

Syntax:

```
public void set<PropertyName>(int index, property_datatype value)
```

Ex:

```
public double[] getData()
{
return data;
}
```

Syntax :

public void set<PropertyName>(property_datatype[] property_array)

Ex:

```
public void setData(double[] values)
{
}
```

The properties window of BDK does not handle indexed properties. Hence the output can not be displayed here.

Bound Properties:

A bean that has a bound property generates an event when the property is changed.
Bound Properties are the properties of a JavaBean that inform its listeners about changes in its values.
Bound Properties are implemented using the **PropertyChangeSupport** class and its methods.
Bound Properties are always registered with an external event listener.
The event is of type ***PropertyChangeEvent*** and is sent to objects that previously registered an interest in receiving such notifications

- Bean with bound property - Event source
- Bean implementing listener -- event target

In order to provide this notification service a JavaBean needs to have the following two methods:

```
public void addPropertyChangeListener(PropertyChangeListener p) {
changes.addPropertyChangeListener(p);
}
public void removePropertyChangeListener(PropertyChangeListener p) {
changes.removePropertyChangeListener(p);
}
```

PropertyChangeListener is an interface declared in the java.beans package. Observers which want to be notified of property changes have to implement this interface, which consists of only one method:

```
public interface PropertyChangeListener extends EventListener {
public void propertyChange(PropertyChangeEvent e );
}
```

Constrained Properties:

It generates an event when an attempt is made to change it value.
Constrained Properties are implemented using the ***PropertyChangeEvent*** class.
The event is sent to objects that previously registered an interest in receiving such notification

Those other objects have the ability to veto the proposed change
This allows a bean to operate differently according to the runtime environment
A bean property for which a change to the property results in validation by another bean. The other bean may reject the change if it is not appropriate.
Constrained Properties are the properties that are protected from being changed by other JavaBeans.
Constrained Properties are registered with an external event listener that has the ability to either accept or reject the change in the value of a constrained property.
Constrained Properties can be retrieved using the get method. The prototype of the get method is:
Syntax: public string get<ConstrainedPropertyName> ()
Can be specified using the set method. The prototype of the set method is:

Syntax: public string set<ConstrainedPropertyName> (String str) throws PropertyVetoException

Design Patterns for Events:

Handling Events in JavaBeans:
Enables Beans to communicate and connect together.
Beans generate events and these events can be sent to other objects.
Event means any activity that interrupts the current ongoing activity.
Example: mouse clicks, pressing key…
User-defined JavaBeans interact with the help of user-defined events, which are also called custom events.
You can use the Java event delegation model to handle these custom events.

The components of the event delegation model are:

- **Event Source**: Generates the event and informs all the event listeners that are registered with it.
- **Event Listener**: Receives the notification, when an event source generates an event.
- **Event Object**: Represents the various types of events that can be generated by the event sources.

Creating Custom Events:

The classes and interfaces that you need to define to create the custom JavaBean events are:

- An event class to define a custom JavaBean event.
- An event listener interface for the custom JavaBean event.
- An event handler to process the custom JavaBean event.
- A target Java application that implements the custom event.

Creating the Event Class:

The event class that defines the custom event extends the EventObject class of the java.util package. For example,

```
public class NumberEvent extends EventObject
{
public int number1,number2;
public NumberEvent(Object o,int number1,int number2)
{
super(o);
this.number1=number1;
this.number2=number2;
}
}
```

Beans can generate events and send them to other objects.

Creating Event Listeners

When the event source triggers an event, it sends a notification to the event listener interface. The event listener interface implements the java.util.EventListener interface.
Syntax:

```
public void addTListener(TListener eventListener);
public void addTListener(TListener eventListener)throws TooManyListeners;
public void removeTListener(TListener eventListener);
```

The target application that uses the custom event implements the custom listener. For example,

```
public interface NumberEnteredListener extends EventListener
{
public void arithmeticPerformed(NumberEvent mec);
}
```

Creating Event Handler

Custom event handlers should define the following methods:

addXXListener(): Registers listeners of a JavaBean event.
fireXX(): Notifies the listeners of the occurrence of a JavaBean event.
removeXXListener(): Removes a listener from the list of registered listeners of a JavaBean.

The code snippet to define an event handler for the custom event NumberEvent is:

```
public class NumberBean extends JPanel implements ActionListener
{
public NumberBean()
{}
NumberEnteredListener mel;
public void addNumberListener(NumberEnteredListener mel)
{
this.mel = mel;
}
}
```

Persistence

Persistence means an ability to save properties and events of our beans to non-volatile storage and retrieve later.

It has the ability to save a bean to storage and retrieve it at a later time Configuration settings are saved. It is implemented by Java serialization.

If a bean inherits directly or indirectly from Component class it is automatically Serializable.

Transient keyword can be used to designate data members of a Bean that should not be serialized.

- Enables developers to customize Beans in an application builder, and then retrieve those Beans, with customized features intact, for future use, perhaps in another environment.
- Java Beans supports two forms of persistence:
    - Automatic persistence
    - External persistence

Automatic Persistence:

Automatic persistence is java's built-in serialization mechanism to save and restore the state of a bean.

External Persistence:

External persistence, on the other hand, gives you the option of supplying your own custom classes to control precisely how a bean state is stored and retrieved.

- Juggle Bean.
- Building an applet
- Your own bean

Customizers:

- The Properties window of the BDK allows a developer to modify the several properties of the Bean.
- Property sheet may not be the best user interface for a complex component
- It can provide step-by-step wizard guide to use component
- It can provide a GUI frame with image which visually tells what is changed such as radio button, check box...
- It can customize the appearance and behavior of the properties
- Online documentation can also be provided. A Bean developer has great flexibility to develop a customizer that can differentiate his or her product in the marketplace.
- To make it easy to configure a java beans component
- Enables a developer to use an application builder tool to customize the appearance and behavior of a bean

The Java Beans API:

The Java Beans functionality is provided by a set of classes and interfaces in the **java.beans** package.

Package java.beans

Contains classes related to developing *beans* -- components based on the JavaBeans architecture

Interface	Description
AppletInitializer	Methods in this interface are used to initialize Beans that are also applets.
BeanInfo	This interface allows a designer to specify information about the properties, events, and methods of a Bean.
Customizer	This interface allows a designer to provide a graphical user interface through which a Bean may be configured.
DesignMode	Methods in this interface determine if a Bean is executing in design mode.
ExceptionListener	A method in this interface is invoked when an exception has occurred.
PropertyChangeListener	A method in this interface is invoked when a bound property is changed.
PropertyEditor	Objects that implement this interface allow designers to change and display property values.
VetoableChangeListener	A method in this interface is invoked when a constrained property is changed.
Visibility	Methods in this interface allow a Bean to execute in environments where a graphical user interface is not available.

Class	Description
BeanDescriptor	This class provides information about a Bean. It also allows you to associate a customizer with a Bean.
Beans	This class is used to obtain information about a Bean.
DefaultPersistenceDelegate	A concrete subclass of **PersistenceDelegate**.
Encoder	Encodes the state of a set of Beans. Can be used to write this information to a stream.
EventHandler	Supports dynamic event listener creation.
EventSetDescriptor	Instances of this class describe an event that can be generated by a Bean.
Expression	Encapsulates a call to a method that returns a result.
FeatureDescriptor	This is the superclass of the **PropertyDescriptor**, **EventSetDescriptor**, and **MethodDescriptor** classes.
IndexedPropertyChangeEvent	A subclass of **PropertyChangeEvent** that represents a change to an indexed property.
IndexedPropertyDescriptor	Instances of this class describe an indexed property of a Bean.
IntrospectionException	An exception of this type is generated if a problem occurs when analyzing a Bean.
Introspector	This class analyzes a Bean and constructs a **BeanInfo** object that describes the component.

Class	Description
MethodDescriptor	Instances of this class describe a method of a Bean.
ParameterDescriptor	Instances of this class describe a method parameter.
PersistenceDelegate	Handles the state information of an object.
PropertyChangeEvent	This event is generated when bound or constrained properties are changed. It is sent to objects that registered an interest in these events and that implement either the **PropertyChangeListener** or **VetoableChangeListener** interfaces.
PropertyChangeListenerProxy	Extends **EventListenerProxy** and implements **PropertyChangeListener**.
PropertyChangeSupport	Beans that support bound properties can use this class to notify **PropertyChangeListener** objects.
PropertyDescriptor	Instances of this class describe a property of a Bean.
PropertyEditorManager	This class locates a **PropertyEditor** object for a given type.
PropertyEditorSupport	This class provides functionality that can be used when writing property editors.
PropertyVetoException	An exception of this type is generated if a change to a constrained property is vetoed.
SimpleBeanInfo	This class provides functionality that can be used when writing **BeanInfo** classes.
Statement	Encapsulates a call to a method.
VetoableChangeListenerProxy	Extends **EventListenerProxy** and implements **VetoableChangeListener**.
VetoableChangeSupport	Beans that support constrained properties can use this class to notify **VetoableChangeListener** objects.
XMLDecoder	Used to read a Bean from an XML document.
XMLEncoder	Used to write a Bean to an XML document.

The BeanInfo **Interface:**

By default an Introspector uses the Reflection API to determine the features of a JavaBean. However, a JavaBean can provide its own BeanInfo which will be used instead by the Introspector to determine the discussed information. This allows a developer hiding specific properties, events and methods from a builder tool or from any other tool which uses the Introspector class. Moreover it allows supplying further details about events/properties/methods as you are in charge of creating the descriptor objects. Hence you can, for example, call the setShortDescription() method to set a descriptive description. A BeanInfo class has to be derived from the SimpleBeanInfo class and its name has to start with the name of the associated JavaBean. At this point it has to be underlined that the name of the BeanInfo class is the only relation between a JavaBean and its BeanInfo class.
The BeanInfo interface provides the methods that enable you to specify and retrieve the information about a JavaBean.

The following table lists some of the methods of BeanInfo interface:

Method	*Description*
`MethodDescriptor[] getMethodDescriptors()`	Returns an array of the method descriptor objects of a JavaBean. The method descriptor objects are used to determine information about the various methods defined in a JavaBean.
`EventDescriptor[] getEventDescriptors()`	Returns an array of the event descriptor objects of a JavaBean. The event descriptor objects determine the information about the events associated with a JavaBean.

Method	Description
PropertyDescriptor[] getPropertyDescriptors()	Returns an array of the property descriptor objects of a JavaBean. The property descriptor objects are used to determine information about the various custom properties of a JavaBean.
Image getIcon(int icon_type)	Returns a corresponding image object for one of the fields of the BeanInfo interface passed as a parameter to this method. The BeanInfo interface defines int fields, such as ICON_COLOR_32x32 and ICON_MONO_32x32 to represent icons.

Introspector

The **Introspector** class provides several static methods that support introspection. Of most interest is **getBeanInfo( )**. This method returns a **BeanInfo** object that can be used to obtain information about the Bean. The **getBeanInfo( )** method has several forms, including the one shown here:

static BeanInfo getBeanInfo(Class<?> *bean*) throws IntrospectionException

The returned object contains information about the Bean specified by *bean.*

PropertyDescriptor

The **PropertyDescriptor** class describes a Bean property. It supports several methods that manage and describe properties. For example, you can determine if a property is bound by calling **isBound( )**. To determine if a property is constrained, call **isConstrained( )**. You can obtain the name of property by calling **getName( )**.

EventSetDescriptor

The **EventSetDescriptor** class represents a Bean event. It supports several methods that obtain the methods that a Bean uses to add or remove event listeners, and to otherwise manage events. For example, to obtain the method used to add listeners, call **getAddListenerMethod( )**. To

obtain the method used to remove listeners, call **getRemoveListenerMethod( )**. To obtain the type of a listener, call **getListenerType( )**. You can obtain the name of an event by calling **getName( )**.

MethodDescriptor

The **MethodDescriptor** class represents a Bean method. To obtain the name of the method, call **getName( )**. You can obtain information about the method by calling **getMethod( )**, shown here:

Method getMethod( )

An object of type **Method** that describes the method is returned.

A Bean Example

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

public class Colors extends Canvas implements Serializable
 {
        transient private Color color; // not persistent
        private boolean rectangular; // is persistent
        public Colors()
        {
                addMouseListener(new MouseAdapter() {
                public void mousePressed(MouseEvent me) {
                change();
                }
                });
        rectangular = false;
        setSize(200, 100);
        change();

        }

        public boolean getRectangular() {
                return rectangular;
        }
        public void setRectangular(boolean flag) {
                this.rectangular = flag;
        repaint();
        }
        public void change() {
                color = randomColor();
```

```
            repaint();
        }
        private Color randomColor() {
            int r = (int)(255*Math.random());
            int g = (int)(255*Math.random());
            int b = (int)(255*Math.random());
            return new Color(r, g, b);
        }
        public void paint(Graphics g) {
            Dimension d = getSize();
            int h = d.height;
            int w = d.width;
            g.setColor(color);
            if(rectangular) {
                g.fillRect(0, 0, w-1, h-1);
            }
            else {
                g.fillOval(0, 0, w-1, h-1);
            }
        }
}
```

The **Colors** Bean displays a colored object within a frame. The color of the component is determined by the private **Color** variable **color**, and its shape is determined by the private **Boolean** variable **rectangular**. The constructor defines an anonymous inner class that extends **MouseAdapter** and overrides its **mousePressed( )** method. The **change( )** method is invoked in response to mouse presses. It selects a random color and then repaints the component. The **getRectangular( )** and **setRectangular( )** methods provide access to the one property of this Bean. The **change( )** method calls **randomColor( )** to choose a color and then calls **repaint( )** to make the change visible. Notice that the **paint( )** method uses the **rectangular** and **color** variables to determine how to present the Bean.

Java Bean

A Java Bean is a java class that should follow following conventions:

- It should have a no-arg constructor.
- It should be Serializable.
- It should provide methods to set and get the values of the properties, known as getter and setter methods.

Why use Java Bean?

According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object, so we can access this object from multiple places. Moreover, it provides the easy maintenance.

Simple example of java bean class

```
//Employee.java

package mypack;
public class Employee implements java.io.Serializable{
private int id;
private String name;

public Employee(){}

public void setId(int id){this.id=id;}

public int getId(){return id;}

public void setName(String name){this.name=name;}

public String getName(){return name;}

}
```

How to access the java bean class?

To access the java bean class, we should use getter and setter methods

```
package mypack;
public class Test{
public static void main(String args[]){

Employee e=new Employee();//object is created

e.setName("Arjun");//setting value to the object

System.out.println(e.getName());

}}
```

Note: There are two ways to provide values to the object, one way is by constructor and second is by setter method.

Java Networking

Java Networking is a concept of connecting two or more computing devices together so that we can share resources.

Java socket programming provides facility to share data between different computing devices.

Advantage of Java Networking

1. sharing resources
2. centralize software management

Java Networking Terminology

The widely used java networking terminologies are given below:

1. IP Address
2. Protocol
3. Port Number
4. MAC Address
5. Connection-oriented and connection-less protocol
6. Socket

1) IP Address

IP address is a unique number assigned to a node of a network e.g. 192.168.0.1 . It is composed of octets that range from 0 to 255.

It is a logical address that can be changed.

2) Protocol

A protocol is a set of rules basically that is followed for communication. For example:

- TCP
- FTP
- Telnet
- SMTP
- POP etc.

3) Port Number

The port number is used to uniquely identify different applications. It acts as a communication endpoint between applications.

The port number is associated with the IP address for communication between two applications.

4) MAC Address

MAC (Media Access Control) Address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC.

5) Connection-oriented and connection-less protocol

In connection-oriented protocol, acknowledgement is sent by the receiver. So it is reliable but slow. The example of connection-oriented protocol is TCP.

But, in connection-less protocol, acknowledgement is not sent by the receiver. So it is not reliable but fast. The example of connection-less protocol is UDP.

6) Socket

A socket is an endpoint between two way communications.

Sockets are at the foundation of modern networking because a socket allows a single computer to serve many different clients at once, as well as to serve many different types of information. This is accomplished through the use of a port, which is a numbered socket on a particular machine. A server process is said to “listen” to a port until a client connects to it. A server is allowed to accept multiple clients connected to the same port number, although each session is unique. To manage multiple client connections, a server process must be multithreaded or have some other means of multiplexing the simultaneous I/O.

Visit next page for java socket programming.

Java Socket Programming

Java Socket programming is used for communication between the applications running on different JRE.

Java Socket programming can be connection-oriented or connection-less.

Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

The client in socket programming must know two information:

1. IP Address of Server, and
2. Port number.

Socket class

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

Important methods

Method	Description
1) public InputStream getInputStream()	returns the InputStream attached with this socket.
2) public OutputStream getOutputStream()	returns the OutputStream attached with this socket.
3) public synchronized void close()	closes this socket

ServerSocket class

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

Important methods

Method	Description

1) public Socket accept()	returns the socket and establish a connection between server and client.
2) public synchronized void close()	closes the server socket.

Example of Java Socket Programming

Let's see a simple of java socket programming in which client sends a text and server receives it.

File: MyServer.java

```
1. import java.io.*;
2. import java.net.*;
3. public class MyServer {
4. public static void main(String[] args){
5. try{
6. ServerSocket ss=new ServerSocket(6666);
7. Socket s=ss.accept();//establishes connection
8. DataInputStream dis=new DataInputStream(s.getInputStream());
9. String  str=(String)dis.readUTF();
10. System.out.println("message= "+str);
11. ss.close();
12. }catch(Exception e){System.out.println(e);}
13. }
14. }
```

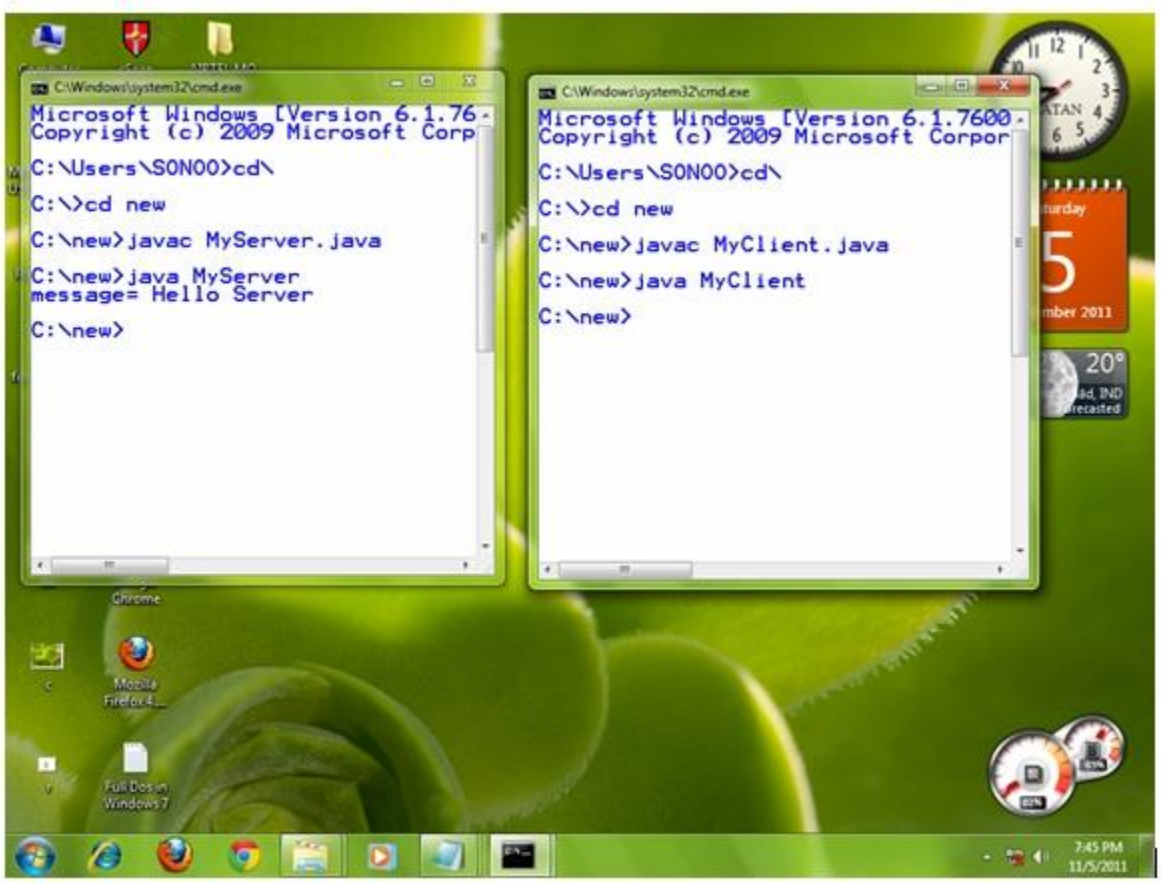
File: MyClient.java

```
1. import java.io.*;
2. import java.net.*;
3. public class MyClient {
4. public static void main(String[] args) {
5. try{
6. Socket s=new Socket("localhost",6666);
7. DataOutputStream dout=new DataOutputStream(s.getOutputStream());
8. dout.writeUTF("Hello Server");
9. dout.flush();
10. dout.close();
11. s.close();
12. }catch(Exception e){System.out.println(e);}
13. }
14. }
```

download this example

To execute this program open two command prompts and execute each program at each command prompt as displayed in the below figure.

After running the client application, a message will be displayed on the server console.



Example of Java Socket Programming (Read-Write both side)

In this example, client will write first to the server then server will receive and print the text. Then server will write to the client and client will receive and print the text. The step goes on.

File: MyServer.java

```
import java.net.*;
import java.io.*;
class MyServer{
```

```
public static void main(String args[])throws Exception{
ServerSocket ss=new ServerSocket(3333);
Socket s=ss.accept();
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

String str="",str2="";
while(!str.equals("stop")){
str=din.readUTF();
System.out.println("client says: "+str);
str2=br.readLine();
dout.writeUTF(str2);
dout.flush();
}
din.close();
s.close();
ss.close();
}}
```

File: MyClient.java

```
import java.net.*;
import java.io.*;
class MyClient{
public static void main(String args[])throws Exception{
Socket s=new Socket("localhost",3333);
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

String str="",str2="";
while(!str.equals("stop")){
str=br.readLine();
dout.writeUTF(str);
dout.flush();
str2=din.readUTF();
System.out.println("Server says: "+str2);
}

dout.close();
s.close();
}}
```

Java URL

The **Java URL** class represents an URL. URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web. For example:

1. http://www.javatpoint.com/java-tutorial

   A URL contains many information:

   1. **Protocol:** In this case, http is the protocol.
   2. **Server name or IP Address:** In this case, www.javatpoint.com is the server name.
   3. **Port Number:** It is an optional attribute. If we write http//ww.javatpoint.com:80/sonoojaiswal/ , 80 is the port number. If port number is not mentioned in the URL, it returns -1.
   4. **File Name or directory name:** In this case, index.jsp is the file name.

Commonly used methods of Java URL class

The java.net.URL class provides many methods. The important methods of URL class are given below.

Method	Description
public String getProtocol()	it returns the protocol of the URL.
public String getHost()	it returns the host name of the URL.
public String getPort()	it returns the Port Number of the URL.
public String getFile()	it returns the file name of the URL.
public URLConnection openConnection()	it returns the instance of URLConnection i.e. associated with this URL.

Example of Java URL class

```
1. //URLDemo.java
2. import java.io.*;
3. import java.net.*;
4. public class URLDemo{
5. public static void main(String[] args){
6. try{
7. URL url=new URL("http://www.javatpoint.com/java-tutorial");
8.
9. System.out.println("Protocol: "+url.getProtocol());
10. System.out.println("Host Name: "+url.getHost());
11. System.out.println("Port Number: "+url.getPort());
```

```
12. System.out.println("File Name: "+url.getFile());
13.
14. }catch(Exception e){System.out.println(e);}
15. }
16. }
```

Test it Now

Output:

```
Protocol: http
Host Name: www.javatpoint.com
Port Number: -1
File Name: /java-tutorial
```

Java URLConnection class

The **Java URLConnection** class represents a communication link between the URL and the application. This class can be used to read and write data to the specified resource referred by the URL.

How to get the object of URLConnection class

The openConnection() method of URL class returns the object of URLConnection class. Syntax:

```
1. public URLConnection openConnection()throws IOException{}
```

Displaying source code of a webpage by URLConnecton class

The URLConnection class provides many methods, we can display all the data of a webpage by using the getInputStream() method. The getInputStream() method returns all the data of the specified URL in the stream that can be read and displayed.

Example of Java URLConnecton class

```
1. import java.io.*;
2. import java.net.*;
3. public class URLConnectionExample {
4. public static void main(String[] args){
5. try{
6. URL url=new URL("http://www.javatpoint.com/java-tutorial");
```

```
7. URLConnection urlcon=url.openConnection();
8. InputStream stream=urlcon.getInputStream();
9. int i;
10. while((i=stream.read())!=-1){
11. System.out.print((char)i);
12. }
13. }catch(Exception e){System.out.println(e);}
14. }
15. }
```

Java HttpURLConnection class

The **Java HttpURLConnection** class is http specific URLConnection. It works for HTTP protocol only.

By the help of HttpURLConnection class, you can information of any HTTP URL such as header information, status code, response code etc.

The java.net.HttpURLConnection is subclass of URLConnection class.

How to get the object of HttpURLConnection class

The openConnection() method of URL class returns the object of URLConnection class. Syntax:

```
1. public URLConnection openConnection()throws IOException{}
```

You can typecast it to HttpURLConnection type as given below.

```
1. URL url=new URL("http://www.javatpoint.com/java-tutorial");
2. HttpURLConnection huc=(HttpURLConnection)url.openConnection();
```

Java HttpURLConnecton Example

```
1. import java.io.*;
2. import java.net.*;
3. public class HttpURLConnectionDemo{
4. public static void main(String[] args){
5. try{
6. URL url=new URL("http://www.javatpoint.com/java-tutorial");
7. HttpURLConnection huc=(HttpURLConnection)url.openConnection();
8. for(int i=1;i<=8;i++){
9. System.out.println(huc.getHeaderFieldKey(i)+" = "+huc.getHeaderField(i));
10. }
11. huc.disconnect();
12. }catch(Exception e){System.out.println(e);}
```

```
13. }
14. }
```

Test it Now

Output:

```
Date = Wed, 10 Dec 2014 19:31:14 GMT
Set-Cookie = JSESSIONID=D70B87DBB832820CACA5998C90939D48; Path=/
Content-Type = text/html
Cache-Control = max-age=2592000
Expires = Fri, 09 Jan 2015 19:31:14 GMT
Vary = Accept-Encoding,User-Agent
Connection = close
Transfer-Encoding = chunked
```

Java InetAddress class

Java InetAddress class represents an IP address. The java.net.InetAddress class provides methods to get the IP of any host name *for example* www.javatpoint.com, www.google.com, www.facebook.com etc.

Commonly used methods of InetAddress class

Method	Description
public static InetAddress getByName(String host) throws UnknownHostException	it returns the instance of InetAddress containing LocalHost IP and name.
public static InetAddress getLocalHost() throws UnknownHostException	it returns the instance of InetAdddress containing local host name and address.
public String getHostName()	it returns the host name of the IP address.
public String getHostAddress()	it returns the IP address in string format.

Example of Java InetAddress class

Let's see a simple example of InetAddress class to get ip address of www.javatpoint.com website.

```
1. import java.io.*;
2. import java.net.*;
3. public class InetDemo{
4. public static void main(String[] args){
5. try{
6. InetAddress ip=InetAddress.getByName("www.javatpoint.com");
7.
8. System.out.println("Host Name: "+ip.getHostName());
9. System.out.println("IP Address: "+ip.getHostAddress());
10. }catch(Exception e){System.out.println(e);}
11. }
12. }
```

Test it Now

Output:

```
Host Name: www.javatpoint.com
IP Address: 206.51.231.148
```

Java DatagramSocket and DatagramPacket

Java DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

Java DatagramSocket class

Java DatagramSocket class represents a connection-less socket for sending and receiving datagram packets.

A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

Commonly used Constructors of DatagramSocket class

- **DatagramSocket() throws SocketEeption:** it creates a datagram socket and binds it with the available Port Number on the localhost machine.
- **DatagramSocket(int port) throws SocketEeption:** it creates a datagram socket and binds it with the given Port Number.
- **DatagramSocket(int port, InetAddress address) throws SocketEeption:** it creates a datagram socket and binds it with the specified port number and host address.

Java DatagramPacket class

Java DatagramPacket is a message that can be sent or received. If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed.

Commonly used Constructors of DatagramPacket class

- **DatagramPacket(byte[] barr, int length):** it creates a datagram packet. This constructor is used to receive the packets.
- **DatagramPacket(byte[] barr, int length, InetAddress address, int port):** it creates a datagram packet. This constructor is used to send the packets.

Example of Sending DatagramPacket by DatagramSocket

```
1. //DSender.java
2. import java.net.*;
3. public class DSender{
4.   public static void main(String[] args) throws Exception {
5.     DatagramSocket ds = new DatagramSocket();
6.     String str = "Welcome java";
7.     InetAddress ip = InetAddress.getByName("127.0.0.1");
8.
9.     DatagramPacket dp = new DatagramPacket(str.getBytes(), str.length(), ip, 3000);
10.    ds.send(dp);
11.    ds.close();
12.  }
13. }
```

Example of Receiving DatagramPacket by DatagramSocket

```
1. //DReceiver.java
2. import java.net.*;
3. public class DReceiver{
4.   public static void main(String[] args) throws Exception {
5.     DatagramSocket ds = new DatagramSocket(3000);
6.     byte[] buf = new byte[1024];
7.     DatagramPacket dp = new DatagramPacket(buf, 1024);
8.     ds.receive(dp);
9.     String str = new String(dp.getData(), 0, dp.getLength());
10.    System.out.println(str);
11.    ds.close();
12.  }
13. }
```

Unit-V

Servlets

Servlet technology is used to create web application (resides at server side and generates dynamic web page).

Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was popular as a server-side programming language. But there was many disadvantages of this technology. We have discussed these disadvantages below.

There are many interfaces and classes in the servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse etc.

What is a Servlet?

Servlet can be described in many ways, depending on the context.

- Servlet is a technology i.e. used to create web application.
- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extend the capabilities of the servers and respond to the incoming request. It can respond to any type of requests.
- Servlet is a web component that is deployed on the server to create dynamic web page.

What is web application?

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter etc. and other components such as HTML. The web components typically execute in Web Server and respond to HTTP request.

CGI (Common Gateway Interface)

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.

Disadvantages of CGI

There are many problems in CGI technology:

1. If number of clients increases, it takes more time for sending response.
2. For each request, it starts a process and Web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, Perl.

Advantage of Servlets

There are many advantages of servlet over CGI. The web container creates threads for handling the multiple requests to the servlet. Threads have a lot of benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The basic benefits of servlet are as follows:

1. better performance: because it creates a thread for each request not process.
2. Portability: because it uses java language.
3. Robust: Servlets are managed by JVM so no need to worry about memory leak, garbage collection etc.

4. Secure: because it uses java language.

Servlet Terminology

1. Basics of Servlet
2. HTTP
3. Http Request Methods
4. Difference between Get and Post
5. Anatomy of Get Request
6. Anatomy of Post Request
7. Content Type

There are some key points that must be known by the servlet programmer like server, container, get request, post request etc. Let's first discuss these points before starting the servlet technology.

The basic terminology used in servlet is given below:

1. HTTP
2. HTTP Request Types
3. Difference between Get and Post method
4. Container
5. Server and Difference between web server and application server
6. Content Type
7. Introduction of XML
8. Deployment

HTTP (Hyper Text Transfer Protocol)

1. Http is the protocol that allows web servers and browsers to exchange data over the web.

2. It is a request response protocol.

3. Http uses reliable TCP connections by default on TCP port 80.

4. It is stateless means each request is considered as the new request. In other words, server doesn't recognize the user by default.

Http Request Methods

Every request has a header that tells the status of the client. There are many request methods. Get and Post requests are mostly used.

The http request methods are:

- GET
- POST
- HEAD
- PUT
- DELETE
- OPTIONS
- TRACE

HTTP RequestDescription

GET Asks to get the resource at the requested URL.

POST Asks the server to accept the body info attached. It is like GET request with extra info sent with the request.

HEAD Asks for only the header part of whatever a GET would return. Just like GET but with no body.

TRACE Asks for the loopback of the request message, for testing or troubleshooting.

PUT Says to put the enclosed info (the body) at the requested URL.

DELETE Says to delete the resource at the requested URL.

OPTIONS Asks for a list of the HTTP methods to which the thing at the request URL can respond

---

What is the difference between Get and Post?

There are many differences between the Get and Post request. Let's see these differences:

GET POST

1) In case of Get request, only limited amount of data can be sent because data is sent in header. In case of post request, large amount of datacan be sent because data is sent in body.

2) Get request is not secured because data is exposed in URL bar. Post request is secured because data is not exposed in URL bar.

3) Get request can be bookmarked Post request cannot be bookmarked

4) Get request is idempotent. It means second request will be ignored until response of first request is delivered. Post request is non-idempotent

5) Get request is more efficient and used more than Post Post request is less efficient and used less than get.

Anatomy of Get Request

As we know that data is sent in request header in case of get request. It is the default request type. Let's see what informations are sent to the server.

Container

It provides runtime environment for JavaEE (j2ee) applications.

It performs many operations that are given below:

1. Life Cycle Management
2. Multithreaded support
3. Object Pooling
4. Security etc.

Server

It is a running program or software that provides services.

There are two types of servers:

1. Web Server
2. Application Server

Web Server

Web server contains only web or servlet container. It can be used for servlet, jsp, struts, jsf etc. It can't be used for EJB.

Example of Web Servers are: **Apache Tomcat** and **Resin**.

Application Server

Application server contains Web and EJB containers. It can be used for servlet, jsp, struts, jsf, ejb etc.

Example of Application Servers are:

1. **JBoss** Open-source server from JBoss community.
2. **Glassfish** provided by Sun Microsystem. Now acquired by Oracle.
3. **Weblogic** provided by Oracle. It more secured.
4. **Websphere** provided by IBM.

Content Type

Content Type is also known as MIME (Multipurpose internet Mail Extension) Type. It is a **HTTP header** that provides the description about what are you sending to the browser.

There are many content types:

- text/html
- text/plain
- application/msword
- application/vnd.ms-excel
- application/jar
- application/pdf
- application/octet-stream
- application/x-zip
- images/jpeg
- video/quicktime etc.

Servlet API

1. Servlet API
2. Interfaces in javax.servlet package
3. Classes in javax.servlet package
4. Interfaces in javax.servlet.http package
5. Classes in javax.servlet.http package

The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api.

The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.

The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

Let's see what are the interfaces of javax.servlet package.

Interfaces in javax.servlet package

There are many interfaces in javax.servlet package. They are as follows:

1. Servlet
2. ServletRequest
3. ServletResponse
4. RequestDispatcher
5. ServletConfig
6. ServletContext
7. SingleThreadModel
8. Filter
9. FilterConfig
10. FilterChain
11. ServletRequestListener

12. ServletRequestAttributeListener
13. ServletContextListener
14. ServletContextAttributeListener

Classes in javax.servlet package

There are many classes in javax.servlet package. They are as follows:

1. GenericServlet
2. ServletInputStream
3. ServletOutputStream
4. ServletRequestWrapper
5. ServletResponseWrapper
6. ServletRequestEvent
7. ServletContextEvent
8. ServletRequestAttributeEvent
9. ServletContextAttributeEvent
10. ServletException
11. UnavailableException

Interfaces in javax.servlet.http package

There are many interfaces in javax.servlet.http package. They are as follows:

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession
4. HttpSessionListener
5. HttpSessionAttributeListener
6. HttpSessionBindingListener

7. HttpSessionActivationListener
8. HttpSessionContext (deprecated now)

Classes in javax.servlet.http package

There are many classes in javax.servlet.http package. They are as follows:

1. HttpServlet
2. Cookie
3. HttpServletRequestWrapper
4. HttpServletResponseWrapper
5. HttpSessionEvent
6. HttpSessionBindingEvent
7. HttpUtils (deprecated now)

Servlet Interface

1. Servlet Interface
2. Methods of Servlet interface

Servlet interface provides common behaviour to all the servlets.

Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

Methods of Servlet interface

There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.

Method	Description
public void init(ServletConfig config)	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
public void service(ServletRequest request,ServletResponse response)	provides response for the incoming request. It is invoked at each request by the web container.
public void destroy()	is invoked only once and indicates that servlet is being destroyed.
public ServletConfig getServletConfig()	returns the object of ServletConfig.
public String getServletInfo()	returns information about servlet such as writer, copyright, version etc.

Servlet Example by implementing Servlet interface

Let's see the simple example of servlet by implementing the servlet interface.

> ***It will be better if you learn it after visiting steps to create a servlet.***

File: First.java

```
1. import java.io.*;
2. import javax.servlet.*;
3.
4. public class First implements Servlet{
5. ServletConfig config=null;
6.
7. public void init(ServletConfig config){
8. this.config=config;
```

```
9.  System.out.println("servlet is initialized");
10. }
11.
12. public void service(ServletRequest req,ServletResponse res)
13. throws IOException,ServletException{
14.
15. res.setContentType("text/html");
16.
17. PrintWriter out=res.getWriter();
18. out.print("<html><body>");
19. out.print("<b>hello simple servlet</b>");
20. out.print("</body></html>");
21.
22. }
23. public void destroy(){System.out.println("servlet is destroyed");}
24. public ServletConfig getServletConfig(){return config;}
25. public String getServletInfo(){return "copyright 2007-1010";}
26.
27. }
```

GenericServlet class

1. GenericServlet class
2. Methods of GenericServlet class
3. Example of GenericServlet class

GenericServlet class implements **Servlet**, **ServletConfig** and**Serializable** interfaces. It provides the implementaion of all the methods of these interfaces except the service method.

GenericServlet class can handle any type of request so it is protocol-independent.

You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

Methods of GenericServlet class

There are many methods in GenericServlet class. They are as follows:

1. **public void init(ServletConfig config)** is used to initialize the servlet.
2. **public abstract void service(ServletRequest request, ServletResponse response)** provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. **public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. **public ServletConfig getServletConfig()** returns the object of ServletConfig.
5. **public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.
6. **public void init()** it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
7. **public ServletContext getServletContext()** returns the object of ServletContext.
8. **public String getInitParameter(String name)** returns the parameter value for the given parameter name.
9. **public Enumeration getInitParameterNames()** returns all the parameters defined in the web.xml file.
10. **public String getServletName()** returns the name of the servlet object.
11. **public void log(String msg)** writes the given message in the servlet log file.

12. **public void log(String msg,Throwable t)** writes the explanatory message in the servlet log file and a stack trace.

Servlet Example by inheriting the GenericServlet class

Let's see the simple example of servlet by inheriting the GenericServlet class.

> ***It will be better if you learn it after visiting steps to create a servlet.***

File: First.java

```
1. import java.io.*;
2. import javax.servlet.*;
3.
4. public class First extends GenericServlet{
5. public void service(ServletRequest req,ServletResponse res)
6. throws IOException,ServletException{
7.
8. res.setContentType("text/html");
9.
10. PrintWriter out=res.getWriter();
11. out.print("<html><body>");
12. out.print("<b>hello generic servlet</b>");
13. out.print("</body></html>");
14.
15. }
16. }
```

HttpServlet class

The HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

Methods of HttpServlet class

There are many methods in HttpServlet class. They are as follows:

1. **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
4. **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.
5. **protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.
6. **protected void doOptions(HttpServletRequest req, HttpServletResponse res)** handles the OPTIONS request. It is invoked by the web container.
7. **protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. It is invoked by the web container.
8. **protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.
9. **protected void doDelete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.

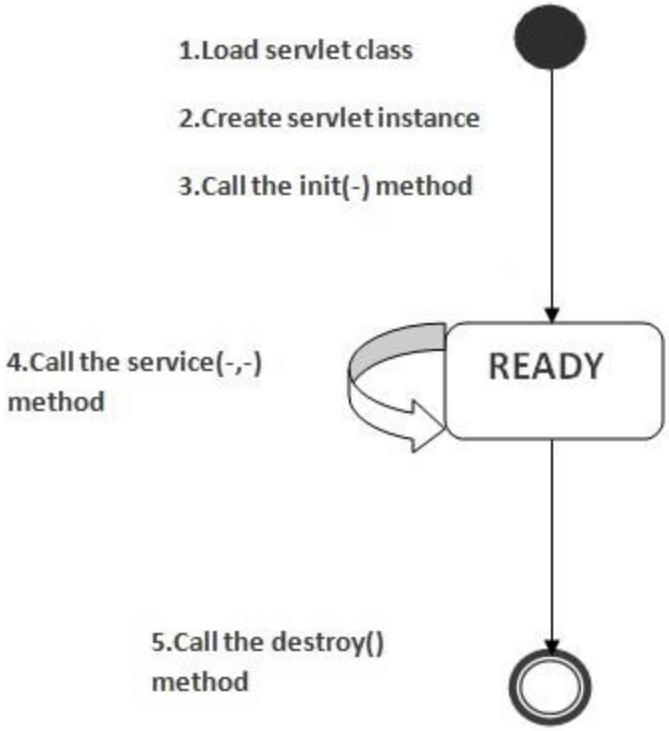
10. **protected long getLastModified(HttpServletRequest req)** returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

Life Cycle of a Servlet (Servlet Life Cycle)

1. Life Cycle of a Servlet
1. Servlet class is loaded
2. Servlet instance is created
3. init method is invoked
4. service method is invoked
5. destroy method is invoked

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.



As displayed in the above diagram, there are three states of a servlet: new, ready and end. The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

---

1) Servlet class is loaded

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

---

2) Servlet instance is created

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

---

3) init method is invoked

The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:

```
public void init(ServletConfig config) throws ServletException
```

---

4) service method is invoked

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

```
public void service(ServletRequest request, ServletResponse response) throws Servlet Exception, IOException
```

---

5) destroy method is invoked

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:

```
public void destroy()
```

Steps to create a servlet example

Steps to create the servlet using Tomcat server

1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the application

There are given 6 steps to create a **servlet example**. These steps are required for all the servers.

The servlet example can be created by three ways:

1. By implementing Servlet interface,
2. By inheriting GenericServlet class, (or)
3. By inheriting HttpServlet class

The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.

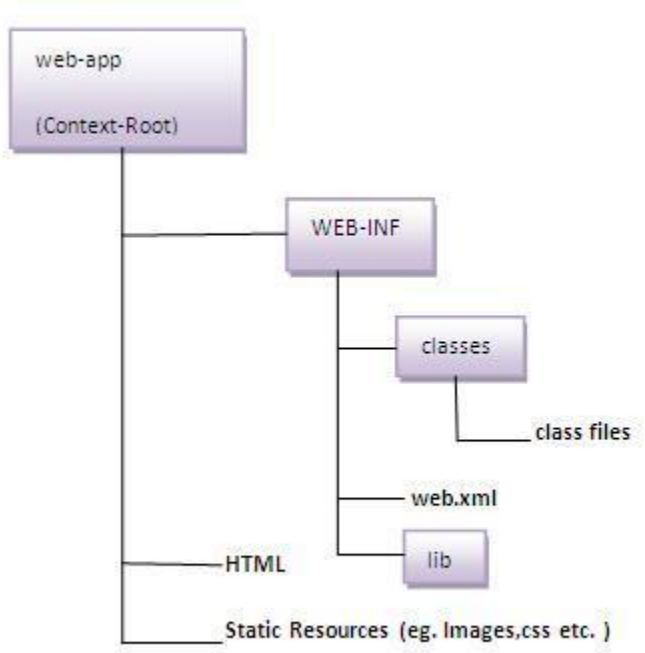
Here, we are going to use **apache tomcat server** in this example. The steps are as follows:

1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the project
6. Access the servlet

1)Create a directory structures

The **directory structure** defines that where to put the different types of files so that web container may get the information and respond to the client.

The Sun Microsystem defines a unique standard to be followed by all the server vendors. Let's see the directory structure that must be followed to create the servlet.



As you can see that the servlet class file must be in the classes folder. The web.xml file must be under the WEB-INF folder.

---

2)Create a Servlet

There are three ways to create the servlet.

1. By implementing the Servlet interface
2. By inheriting the GenericServlet class

3. By inheriting the HttpServlet class

The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost, doHead() etc.

In this example we are going to create a servlet that extends the HttpServlet class. In this example, we are inheriting the HttpServlet class and providing the implementation of the doGet() method. Notice that get request is the default request.

DemoServlet.java

```
1. import javax.servlet.http.*;
2. import javax.servlet.*;
3. import java.io.*;
4. public class DemoServlet extends HttpServlet{
5. public void doGet(HttpServletRequest req,HttpServletResponse res)
6. throws ServletException,IOException
7. {
8. res.setContentType("text/html");//setting the content type
9. PrintWriter pw=res.getWriter();//get the stream to write the data
10.
11. //writing html in the stream
12. pw.println("<html><body>");
13. pw.println("Welcome to servlet");
14. pw.println("</body></html>");
15.
16. pw.close();//closing the stream
17. }}
```

3)Compile the servlet

For compiling the Servlet, jar file is required to be loaded. Different Servers provide different jar files:

Jar file	Server
1) servlet-api.jar	Apache Tomcat
2) weblogic.jar	Weblogic
3) javaee.jar	Glassfish
4) javaee.jar	JBoss

Two ways to load the jar file

1. set classpath
2. paste the jar file in JRE/lib/ext folder

Put the java file in any folder. After compiling the java file, paste the class file of servlet in **WEB-INF/classes**directory.

---

4)Create the deployment descriptor (web.xml file)

The **deployment descriptor** is an xml file, from which Web Container gets the information about the servet to be invoked.

The web container uses the Parser to get the information from the web.xml file. There are many xml parsers such as SAX, DOM and Pull.

There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

web.xml file

```
1. <web-app>
2.
3. <servlet>
4. <servlet-name>sonoojaiswal</servlet-name>
5. <servlet-class>DemoServlet</servlet-class>
6. </servlet>
7.
8. <servlet-mapping>
9. <servlet-name>sonoojaiswal</servlet-name>
10. <url-pattern>/welcome</url-pattern>
11. </servlet-mapping>
12.
13. </web-app>
```

Description of the elements of web.xml file

There are too many elements in the web.xml file. Here is the illustration of some elements that is used in the above web.xml file. The elements are as follows:

<web-app> represents the whole application.

<servlet> is sub element of <web-app> and represents the servlet.

<servlet-name> is sub element of <servlet> represents the name of the servlet.

<servlet-class> is sub element of <servlet> represents the class of the servlet.

<servlet-mapping> is sub element of <web-app>. It is used to map the servlet.

<url-pattern> is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet.

---

5)Start the Server and deploy the project

To start Apache Tomcat server, double click on the startup.bat file under apache-tomcat/bin directory.

Unit-VI

JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to servlet because it provides more functionality than servlet such as expression language, jstl etc.

A JSP page consists of HTML tags and JSP tags. The jsp pages are easier to maintain than servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tag etc.

Advantage of JSP over Servlet

There are many advantages of JSP over servlet. They are as follows:

1) Extension to Servlet

JSP technology is the extension to servlet technology. We can use all the features of servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP that makes JSP development easy.

2) Easy to maintain

JSP can be easily managed because we can easily separate our business logic with presentation logic. In servlet technology, we mix our business logic with the presentation logic.

3) Fast Development: No need to recompile and redeploy

If JSP page is modified, we don't need to recompile and redeploy the project. The servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

4) Less code than Servlet

In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc. that reduces the code. Moreover, we can use EL, implicit objects etc.

The Anatomy of a JSP Page

A JSP page is simply a regular web page with JSP elements for generating the parts of the page that differ for each request, as shown in Figure 3.2.

Everything in the page that is not a JSP element is called *template text* . Template text can really be any text: HTML, WML, XML, or even plain text. Since HTML is by far the most common web page language in use today, most of the descriptions and examples in this book are HTML-based, but keep in mind that JSP has no dependency on HTML; it can be used with any markup language. Template text is always passed straight through to the browser.

Figure 3-2. Template text and JSP elements

When a JSP page request is processed, the template text and the dynamic content generated by the JSP elements are merged, and the result is sent as the response to the browser.

JSP Elements

There are three types of elements with JavaServer Pages: *directive*, *action*, and *scripting* elements.

The directive elements, shown in Table 3.1, are used to specify information about the page itself that remains the same between page requests, for example, the scripting language used in the page, whether session tracking is required, and the name of a page that should be used to report errors, if any.

Table 3-1. Directive Elements

Element	Description
<%@ page ... %>	Defines page-dependent attributes, such as scripting language, error page, and buffer

JSP Processing

A JSP page cannot be sent as-is to the browser; all JSP elements must first be processed by the server. This is done by turning the JSP page into a servlet, and then executing the servlet.

Just as a web server needs a servlet container to provide an interface to servlets, the server needs a JSP container to process JSP pages. The JSP container is often implemented as a servlet configured to handle all requests for JSP pages. In fact, these two containers—a servlet container and a JSP container—are often combined into one package under the name *web container* (as it is referred to in the J2EE documentation).

A JSP container is responsible for converting the JSP page into a servlet (known as the *JSP page implementation class*) and compiling the servlet. These two steps form the *translation phase* . The JSP container automatically initiates the translation phase for a page when the first request for the page is received. The translation phase takes a bit of time, of course, so a user notices a slight delay the first time a JSP page is requested. The translation phase can also be initiated explicitly; this is referred to as *precompilation*of a JSP page. Precompiling a JSP page avoids hitting the user with this delay, and is discussed in more detail in Chapter 12.

The JSP container is also responsible for invoking the JSP page implementation class to process each request and generate the response. This is called the *request processing phase*. The two phases are illustrated ...

JSP Application Design with MVC

JSP technology can play a part in everything from the simplest web application, such as an online phone list or an employee vacation planner, to full-fledged enterprise applications, such as a human resource application or a sophisticated online shopping site. How large a part JSP plays differs in each case, of course. In this section, we introduce a design model suitable for both simple and complex applications called Model-View-Controller (MVC).

MVC was first described by Xerox in a number of papers published in the late 1980s. The key point of using MVC is to separate components into three distinct units: the Model, the View, and the Controller. In a server application, we commonly classify the parts of the application as: business logic, presentation, and request processing. *Business logic* is the term used for the manipulation of an application's data, i.e., customer, product, and order information. *Presentation* refers to how the application is displayed to the user, i.e., the position, font, and size. And finally, *request processing* is what ties the business logic and presentation parts together. In MVC terms, the Model corresponds to business logic and data, the View to the presentation logic, and the Controller to the request processing.

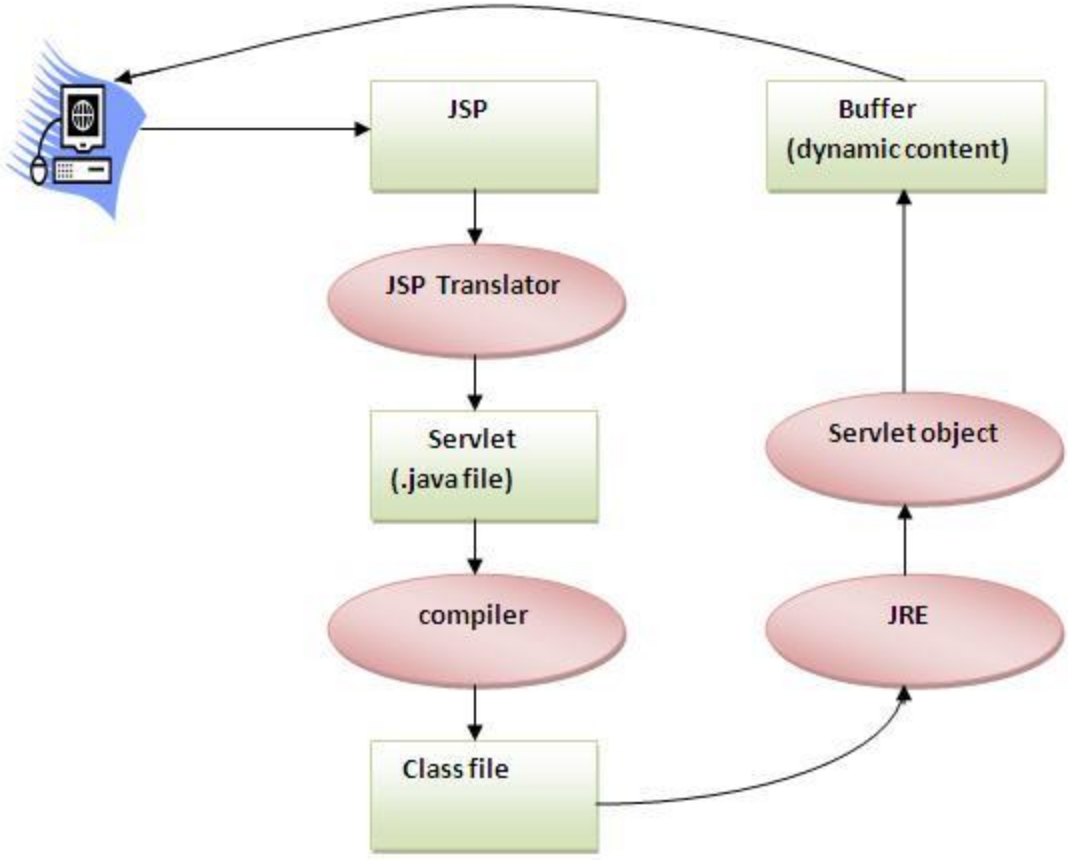
Why use this design with JSP? The answer lies primarily in the first two elements. Remember that an application data structure and logic (the Model) is typically the most stable part of an application, ...

Life cycle of a JSP Page

The JSP pages follow these phases:

- Translation of JSP Page
- Compilation of JSP Page
- Class loading (class file is loaded by the class loader)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (jspInit() method is invoked by the container).
- Request processing (_jspService() method is invoked by the container).
- Destroy (jspDestroy() method is invoked by the container).

Note: jspInit(), _jspService() and jspDestroy() are the life cycle methods of JSP.



As depicted in the above diagram, JSP page is translated into servlet by the help of JSP translator. The JSP translator is a part of webserver that is responsible to translate the JSP page into servlet. Afterthat Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happens in servlet is performed on JSP later like initialization, committing response to the browser and destroy.

Creating a simple JSP Page

To create the first jsp page, write some html code as given below, and save it by .jsp extension. We have save this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the jsp page.

index.jsp

Let's see the simple example of JSP, here we are using the scriptlet tag to put java code in the JSP page. We will learn scriptlet tag later.

```
1. <html>
2. <body>
3. <% out.print(2*5); %>
4. </body>
5. </html>
```

It will print **10** on the browser.

How to run a simple JSP Page ?

Follow the following steps to execute this JSP page:

- Start the server
- put the jsp file in a folder and deploy on the server
- visit the browser by the url http://localhost:portno/contextRoot/jspfile e.g. http://localhost:8888/myapplication/index.jsp

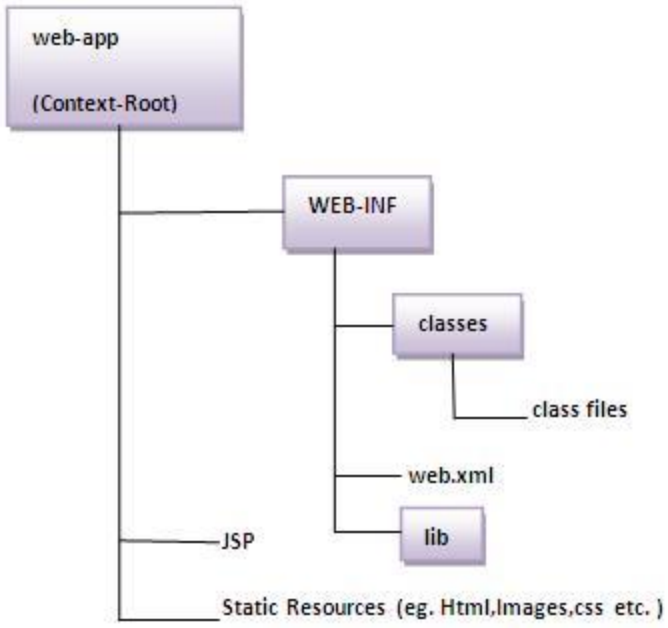
---

Do I need to follow directory structure to run a simple JSP ?

No, there is no need of directory structure if you don't have class files or tld files. For example, put jsp files in a folder directly and deploy that folder.It will be running fine.But if you are using bean class, Servlet or tld file then directory structure is required.

Directory structure of JSP

The directory structure of JSP page is same as servlet. We contain the jsp page outside the WEB-INF folder or in any directory.



JSP Scriptlet tag (Scripting elements)

1. Scripting elements
2. JSP scriptlet tag
3. Simple Example of JSP scriptlet tag
4. Example of JSP scriptlet tag that prints the user name

In JSP, java code can be written inside the jsp page using the scriptlet tag. Let's see what are the scripting elements first.

JSP Scripting elements

The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

- scriptlet tag
- expression tag
- declaration tag

JSP scriptlet tag

A scriptlet tag is used to execute java source code in JSP. Syntax is as follows:

```
<%  java source code %>
```

Example of JSP scriptlet tag

In this example, we are displaying a welcome message.

```
<html>
<body>
<% out.print("welcome to jsp"); %>
</body>
</html>
```

Example of JSP scriptlet tag that prints the user name

In this example, we have created two files index.html and welcome.jsp. The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message.

File: index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

File: welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
</form>
</body>
</html>
```

JSP expression tag

The code placed within **JSP expression tag** is *written to the output stream of the response*. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.

Syntax of JSP expression tag

```
<%=  statement %>
```

Example of JSP expression tag

In this example of jsp expression tag, we are simply displaying a welcome message.

```
<html>
<body>
<%= "welcome to jsp" %>
</body>
</html>
```

Note: Do not end your statement with semicolon in case of expression tag.

Example of JSP expression tag that prints current time

To display the current time, we have used the getTime() method of Calendar class. The getTime() is an instance method of Calendar class, so we have called it after getting the instance of Calendar class by the getInstance() method.

index.jsp

```
<html>
<body>
Current Time: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

Example of JSP expression tag that prints the user name

In this example, we are printing the username using the expression tag. The index.html file gets the username and sends the request to the welcome.jsp file, which displays the username.

File: index.jsp

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname"><br/>
<input type="submit" value="go">
</form>
</body>
</html>
```

File: welcome.jsp

```
<html>
<body>
<%= "Welcome "+request.getParameter("uname") %>
</body>
</html>
```

JSP Declaration Tag

1. JSP declaration tag
2. Difference between JSP scriptlet tag and JSP declaration tag
3. Example of JSP declaration tag that declares field
4. Example of JSP declaration tag that declares method

The **JSP declaration tag** is used *to declare fields and methods*.

The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.

So it doesn't get memory at each request.

Syntax of JSP declaration tag

The syntax of the declaration tag is as follows:

```
<%!  field or method declaration %>
```

Difference between JSP Scriptlet tag and Declaration tag

Jsp Scriptlet Tag	Jsp Declaration Tag
The jsp scriptlet tag can only declare variables not	The jsp declaration tag can declare variables as

methods.	methods.
The declaration of scriptlet tag is placed inside the _jspService() method.	The declaration of jsp declaration tag is placed the _jspService() method.

Example of JSP declaration tag that declares field

In this example of JSP declaration tag, we are declaring the field and printing the value of the declared field using the jsp expression tag.

index.jsp

```
<html>
<body>
<%! int data=50; %>
<%= "Value of the variable is:"+data %>
</body>
</html>
```

Example of JSP declaration tag that declares method

In this example of JSP declaration tag, we are defining the method which returns the cube of given number and calling this method from the jsp expression tag. But we can also use jsp scriptlet tag to call the declared method.

index.jsp

```
<html>
<body>
<%!
int cube(int n){
return n*n*n*;
}
%>
<%= "Cube of 3 is:"+cube(3) %>
</body>
</html>
```