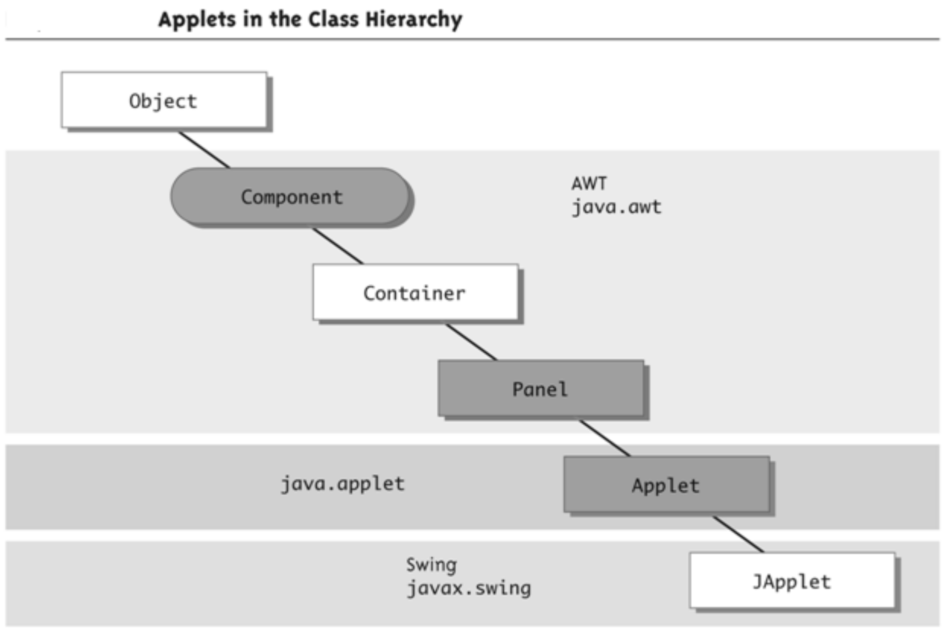
APPLETS:

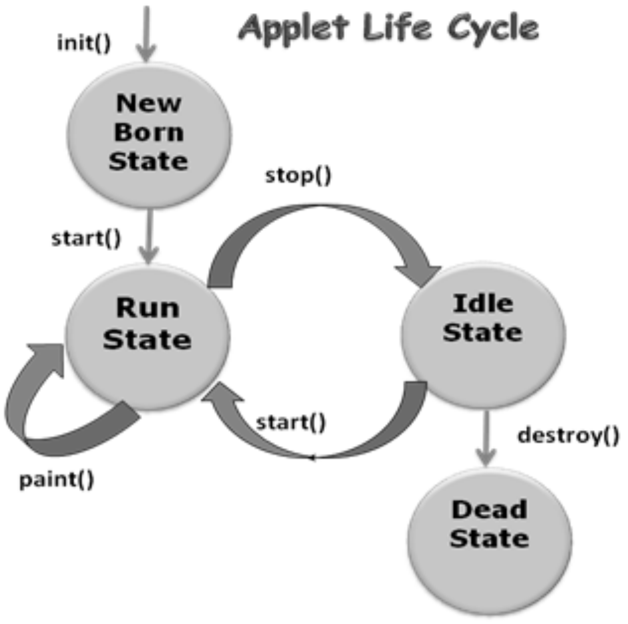
- An applet is a special Java program that can be embedded in HTML documents.
- A applet is typically embedded in a Web page and can be run from a browser
- It is automatically executed by (applet-enabled) web browsers.
    - In Java, non-applet programs are called applications.
- An applet class is normally defined as a derived class of the class Applet
    - The class Applet is in the package java.awt



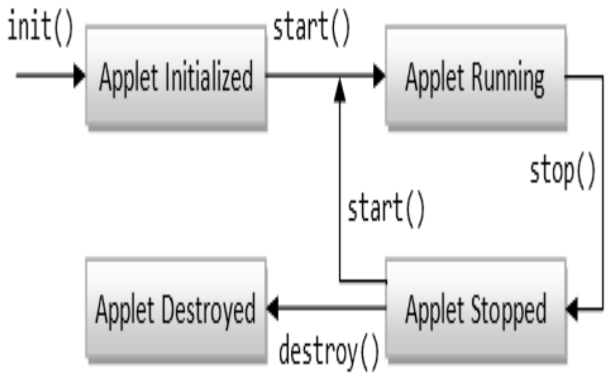
Life Cycle of an Applet:

An applet can react to major events in the following ways:

- It can *initialize* itself.
- It can *start* running.
- It can *stop* running.
- It can perform a *final cleanup*, in preparation for being unloaded.



Life cycle methods:



```
public class Simple extends Applet
{
public void init()
{ }
 public void start()
{ }
 public void stop()
{ }
 public void destroy()
{ }
 public void paint(Graphics g)
{ }
}
```

public void init ( )

- This is the first method to execute
- It is an ideal place to initialize variables

- It is the best place to define the GUI Components (buttons, text fields, scrollbars, etc.), lay them out, and add listeners to them
- Almost every applet you ever write will have an init( ) method

public void start ( ):

- Not always needed
- Called after init( )
- Called each time the page is loaded and restarted
- start() and stop( ) are used when the Applet is doing time-consuming calculations that you don't want to continue when the page is not in front

public void stop( ):

- Not always needed
- Called when the browser leaves the page
- Called just before destroy( )
- Use stop( ) if the applet is doing heavy computation that you don't want to continue when the browser is on some other page

public void destroy( ):

- Called after stop( )
- Use to explicitly release system resources (like threads)
- System resources are usually released automatically

public void paint(Graphics g):

- Needed if you do any drawing or painting other than just using standard GUI Components
- Any painting you want to do should be done here, or in a method you call from here
- Painting that you do in other methods may *or may not* happen
- *Never call* paint(*Graphics*)*, c*all repaint( )

repaint( ):

- Call repaint( ) when you have changed something and want your changes to show up on the screen
- repaint( ) is a *request*
- When you call repaint( ), Java schedules a call to update(Graphics g)

update( ):

- When you call repaint( ), Java schedules a call to update(Graphics g)
- Here's what update does:

```
public void update(Graphics g)
 {
   // Fills applet with background color, then   paint(g);
}
```

INVOKING AN APPLET:

```
<html>
<title>The Hello, World Applet</title>
<applet code="HelloWorldApplet.class" width="320" height="120">
</applet>
</html>
```

HelloWorldApplet.java:

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

NOTE:

getCodeBase():

- The code base, returned by the **Applet** getCodeBase() method, is a URL that specifies the directory from which the applet's classes were loaded.

getDocumentBase():

- The document base, returned by the **Applet** getDocumentBase() method, specifies the directory of the HTML page that contains the applet.

ADDING IMAGE TO AN APPLET:

Images must be in GIF or JPEG format.

Example:

```
        Image  I1 = new Image();
        I1= getImage(getCodeBase(),"images/sea.gif");
```

Code:

```
package applets;
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Image;
import java.net.URL;
public class ImageApplet extends Applet
{
@Override
   public void init() {
   }
@Override
  public void paint(Graphics g) {
   try
   {
      System.out.println(getDocumentBase());
      System.out.println(getCodeBase());
      //Image img = getImage(getDocumentBase(),"Desert.jpg");
      Image img = getImage(new URL("file:C://Users//Desert.jpg"));
       g.drawImage(img,50,50,this);
       g.drawString("Hello", 50, 50);
   }
   catch (Exception e)
   {
    }
  }
}
```

ADDING SOUND TO AN APPLET:

- The AudioClip interface in the Applet class provides basic support for playing sounds.
- Sound format: 8 bit, 8000 Hz, one-channel, Sun ".au" files.
- Methods in AudioClip that need to be implemented in the applet:
- loop(): starts playing the clip repeatedly
- play() & stop() to play & stop the clip.

Code:

```
package applets;
import java.applet.Applet;
import java.applet.AudioClip;
import java.net.URL;
public class AudioApplet extends Applet {
   @Override
   public void init() {
   try {
        System.out.println(getDocumentBase());
        System.out.println(getCodeBase());
        //AudioClip clip = getAudioClip(getCodeBase(),"Yamaha.wav");
        AudioClip clip = getAudioClip(new URL("file:C://Users//Yamaha.wav"));
        clip.play();
       }
   catch (Exception m)
        {
        System.out.println(m);
        }
   }
}
```

PASSING PARAMETERS TO APPLET:

```
package applets;
import java.applet.Applet;
import java.awt.*;
public class Passing extends Applet
{
    String str;
    int a,b,result;
    @Override
   public void init()
    {
        str=getParameter("a");
        a=Integer.parseInt(str);
        str=getParameter("b");
        b=Integer.parseInt(str);
        result=a+b;
        str=String.valueOf(result);
     }
    @Override
  public void paint(Graphics g){
      g.drawString(" Result of Addition is : "+str,20,100);
  }
}

/*<applet code=Passing.class width=400 height=400>
                              <param name=a value=10>
                              <param name=b value=20>
                            </applet>*/
```

EVENT DELEGATION MODEL

<u>EVENT :-</u>

An *event* is an object that describes a state change in a source.

The event delegation model comprises three elements:

- Event source
- Event listener
- Adapter

EVENT SOURCE

An event source is a component, such as a GUI component, that generates an event. The event source can be generated by any type of user interaction. You can also combine a number of different types of events into one event object.

In the event delegation model, a class represents each event type. Event objects are all defined in thejava.util.EventObject subclasses.

<u>EVENT OBIECT:-</u>

A generated event object

- provides the methods to add or remove the source event
- manages the list of registered event listeners
- provides the appropriate class type to the registered event listeners

<u>EVENT LISTENERS:-</u>

A *listener* is an object that is notified when an event occurs.
Components define the events they fire by registering objects called listeners for those event types. When an event is fired, an event object is passed as an argument to the relevant listener object method.
The listener object then handles the event.
All listeners are implementations of the EventListener interface or one of its subinterfaces.

<u>Event Listener Interfaces:-</u>

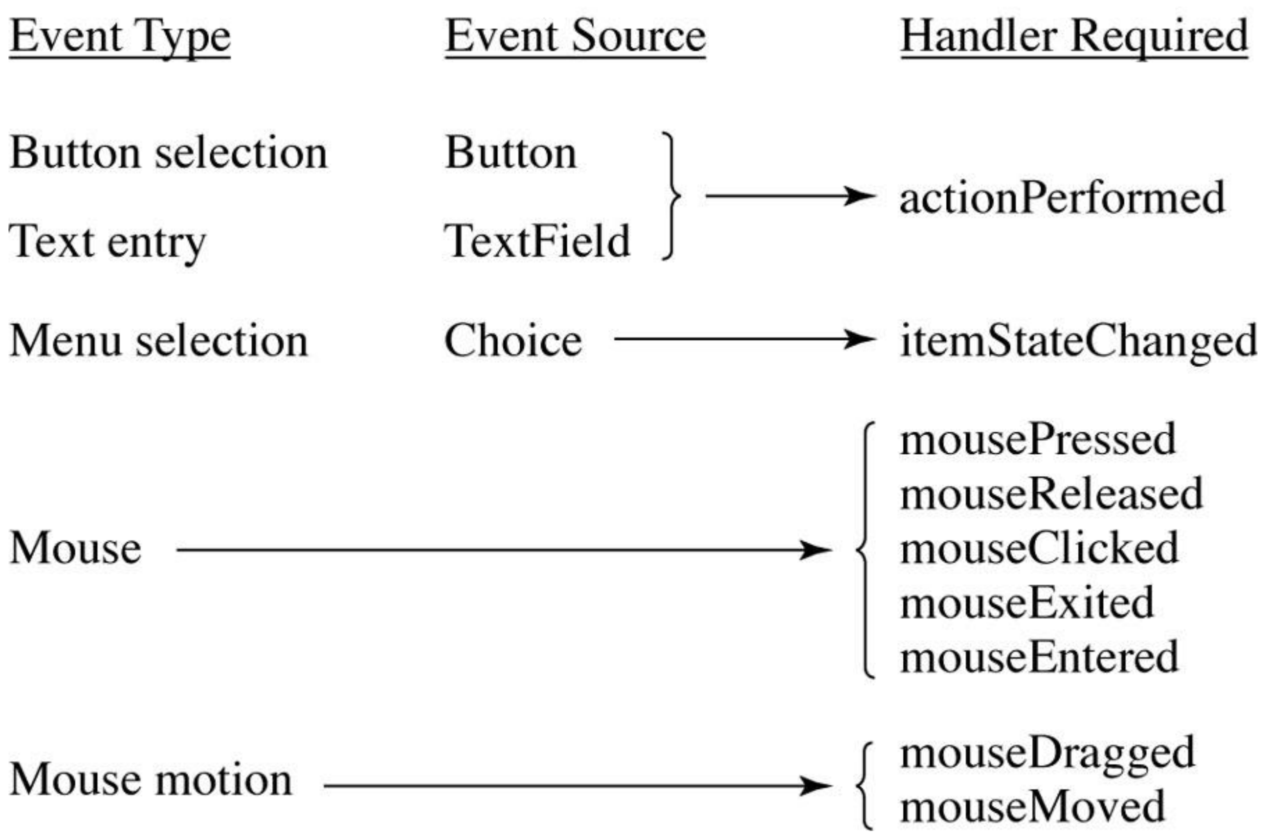
Listeners are created by implementing one or more of the interfaces defined by the **java.awt.event**package.

<u>ADAPTER CLASSES:-</u>

Adapters are abstract classes that implement listener interfaces using predefined methods. These are provided for convenience.

By sub classing MouseAdapter rather than implementing MouseListener directly, you avoid having to write the methods you don't actually need. You only override those that you plan to actually implement.
1. ComponentAdapter
2. ContainerAdapter

3. FocusAdapter
4. KeyAdapter
5. MouseAdapter
6. MouseMotionAdapter
7. WindowAdapter



LISTENER TYPES:-

LISTENERS	METHODS	ADAPTERS
MouseListener	void mouseClicked(MouseEvent*me*) void mouseEntered(MouseEvent*me*) void mouseExited(MouseEvent*me*) void mousePressed(MouseEvent*me*) void mouseReleased(MouseEvent*me*)	MouseAdapter
MouseMotionListener	void mouseDragged(MouseEvent*me*) void mouseMoved(MouseEvent*me*)	MouseMotionAdapter
KeyListener	void keyPressed(KeyEvent*ke*) void keyReleased(KeyEvent*ke*) void keyTyped(KeyEvent*ke*)	KeyAdapter
WindowListener	void windowActivated(WindowEvent*we*) void windowClosed(WindowEvent*we*)	

	void windowClosing(WindowEvent*we*) void windowDeactivated(WindowEvent*we*) void windowDeiconified(WindowEvent*we*) void windowIconified(WindowEvent*we*) void windowOpened(WindowEvent*we*)	WindowAdapter
WindowStateListener	void windowStateChanged(WindowEvent e)	
WindowFocusListener	void windowGainedFocus(WindowEvent*we*) void windowLostFocus(WindowEvent*we*)	
ContainerListener	void componentAdded(ContainerEvent*ce*) void componentRemoved(ContainerEvent*ce*)	ContainerAdapter
ComponentListener	void componentResized(ComponentEvent*ce*) void componentMoved(ComponentEvent*ce*) void componentShown(ComponentEvent*ce*) void componentHidden(ComponentEvent*ce*)	ComponentAdapter
ActionListener	void actionPerformed(ActionEvent*ae*)	NA
ItemListener	void itemStateChanged(ItemEvent*ie*)	NA

LISTENER CLASSES

MOUSE LISTENER:-(Using Listener class as a Separate Class)

Src.java:

```
package Mouse;
import javax.swing.*;
public class Src
{
        Public Src()
        {
        JFrame d=newJFrame("Frame1");
        JTextArea s=newJTextArea("");
        d.add(s);
        Listen L1=new Listen();
        s.addMouseListener(L1);
        d.setSize(190, 190);
        d.setVisible(true);
        }
public static void main(String arg[])
        {
        Srca=newSrc();
        }
}
```

Listen.java:

```
package Mouse;
import java.awt.event.MouseEvent;
```

ADAPTER CLASSES

MOUSE ADAPTER(Using Adapter class as a Separate Class)

Src.java:

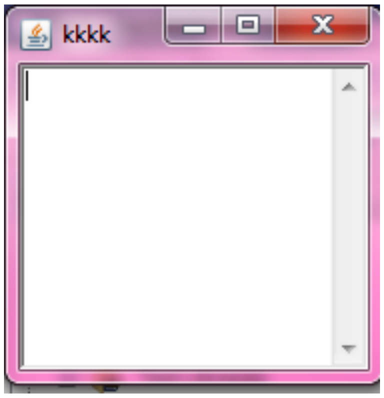
```
package Mouse;
importjavax.swing.*;
public class Src {
        public Src()
        {
        JFrame d=newJFrame("Frame1");
        JTextArea s=newJTextArea("");
        d.add(s);
        Listen L1=new Listen();
        s.addMouseListener(L1);
        d.setSize(190, 190);
        d.setVisible(true);
        }
public static void main(String arg[])
        {
        Srca=newSrc();
        }
}
```

Listen.java:

```
package Mouse;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
```

```java
import java.awt.event.MouseListener;

public class Listen implements MouseListener
{
public void mousePressed(MouseEvent e)
{
System.out.println("MousePressed");
int a=e.getX();
int b=e.getY();
System.out.println("X="+a+"Y="+b);
}
public void mouseReleased(MouseEvent e) {
System.out.println("MouseReleased");
System.exit(0);
}
public void mouseEntered(MouseEvent e) {
System.out.println("MouseEntered");
}
public void mouseExited(MouseEvent e) {
        System.out.println("MouseExited");
}

public void mouseClicked(MouseEvent e) {
System.out.println("MouseClicked");
System.exit(0);
}
}
```

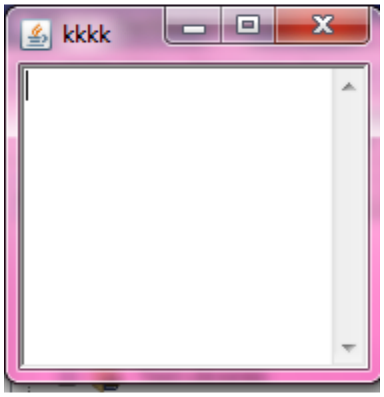
OUTPUT :-



```
MouseEntered
MouseExited
MousePressed
X=36Y=25
MouseReleased
MouseClicked
```

```java
public class Listen extends MouseAdapter
{

public void mousePressed(MouseEvent e) {
System.out.println("MousePressed");
int a=e.getX();
int b=e.getY();
System.out.println("X="+a+"Y="+b);
}
public void mouseReleased(MouseEvent e) {
System.out.println("MouseReleased");
System.exit(0);
}
public void mouseEntered(MouseEvent e) {
System.out.println("MouseEntered");
                    }
public void mouseExited(MouseEvent e) {
System.out.println("MouseExited");
}

public void mouseClicked(MouseEvent e) {
System.out.println("MouseClicked");
System.exit(0);
}
}
```

Output:-



```
MouseEntered
MouseExited
MousePressed
X=36Y=25
MouseReleased
MouseClicked
MouseExited
MouseEntered
```

```
MouseExited
MouseEntered
MouseExited
```

MOUSEMOTIONLISTENER:-(Using Listener class as a SAME Class)

```java
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
import javax.swing.*;
public class Src implements
MouseMotionListener
{
public Src()
{
JFrame d=newJFrame("Frame1");
JTextArea s=newJTextArea("");
d.add(s);
s.addMouseMotionListener(this);
d.setSize(190, 190);
d.setVisible(true);
}
public void mouseMoved(MouseEvent e) {
System.out.println("Mouse is Moving");
}
public void mouseDragged(MouseEvent e) {
        System.out.println("MouseDragged");
}
public static void main(String arg[])
{
Src a=new Src();
}
}
```

```
MouseExited
```

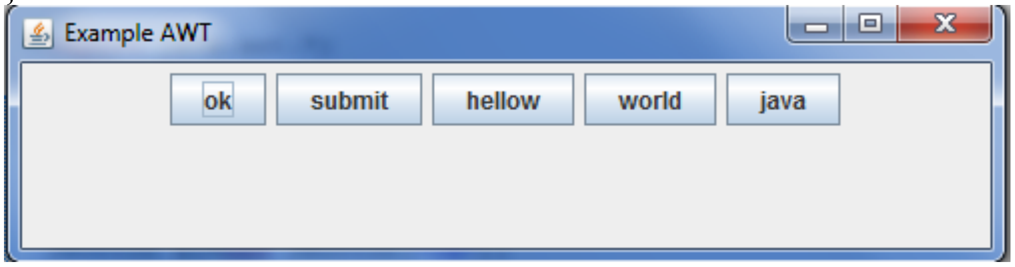
MOUSEMOTIONLISTENER:-(Using Adapter class as a SAME Class)

```java
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionAdapter;
import javax.swing.*;
public class Src extends
MouseMotionAdapter
{
public Src()
{
JFrame d=newJFrame("Frame1");
JTextArea s=newJTextArea("");
d.add(s);
s.addMouseMotionListener(this);
d.setSize(190, 190);
d.setVisible(true);
}
public void mouseMoved(MouseEvent e) {
System.out.println("Mouse is Moving");
}

public void mouseDragged(MouseEvent e) {
System.out.println("MouseDragged");
}
public static void main(String arg[])
{
Src a=new Src();
}
}
```

LAYOUT MANAGEMENT:

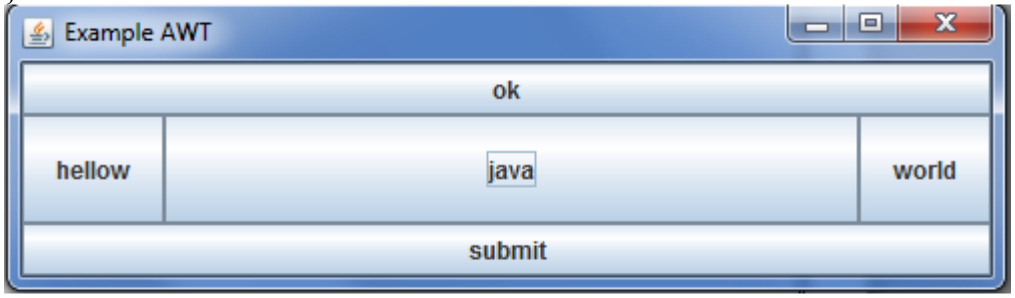
FlowLayout:

```
import java.awt.*;
import javax.swing.*;
public class Flow
{
public static void main(String[] args)
{
JButton a1=new JButton("ok");
JButton b1=new JButton("submit");
JButton c1=new JButton("hellow");
JButton d1=new JButton("world");
JButton e1=new JButton("java");
JFrame a=new JFrame("Example AWT");
Container panel = new JPanel(new FlowLayout(FlowLayout.CENTER));
panel.add(a1);
panel.add(b1);
panel.add(c1);
panel.add(d1);
panel.add(e1);
a.add(panel);
a.setSize(500,500);
a.setVisible(true);
}
}
```



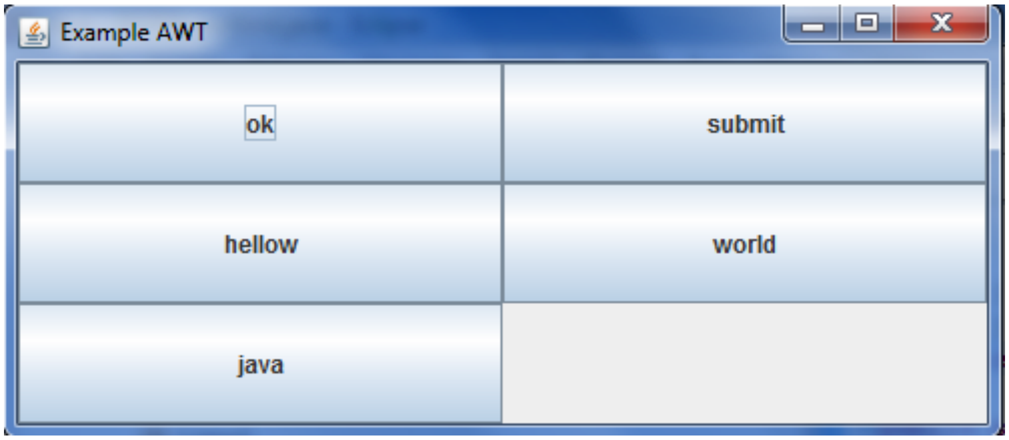
BorderLayout

```
import java.awt.*;
import javax.swing.*;
public class Border
{
public static void main(String[] args)
{
        JButton a1=newJButton("ok");
        JButton b1=newJButton("submit");
        JButton c1=newJButton("hellow");
        JButton d1=newJButton("world");
        JButton e1=newJButton("java");
        JFrame a=newJFrame("Example AWT");
        Container panel = newJPanel(newBorderLayout());
        panel.add(a1,BorderLayout.NORTH);
        panel.add(b1,BorderLayout.SOUTH);
        panel.add(c1,BorderLayout.WEST);
        panel.add(d1,BorderLayout.EAST);
        panel.add(e1,BorderLayout.CENTER);
        a.add(panel);
        a.setSize(500,500);
        a.setVisible(true);
}
}
```



GridLayout:

```
import java.awt.*;
import javax.swing.*;
public class Grid
{
public static void main(String[] args)
{
JButton a1=newJButton("ok");
JButton b1=newJButton("submit");
JButton c1=newJButton("hellow");
JButton d1=newJButton("world");
JButton e1=newJButton("java");
JFrame a=newJFrame("Example AWT");
Container panel = newJPanel(newGridLayout(3,2));
panel.add(a1);
panel.add(b1);
panel.add(c1);
panel.add(d1);
panel.add(e1);
a.add(panel);
a.setSize(500,500);
a.setVisible(true);
}
}
```

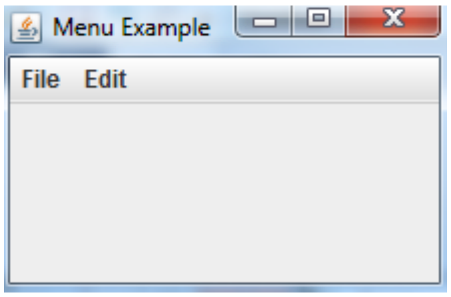


BOXLAYOUT:

```
import java.awt.*;
import javax.swing.*;
public class Box1
{
public static void main(String[] args)
{
JButton a1=newJButton("ok");
JButton b1=newJButton("submit");
JButton c1=newJButton("hellow");
JButton d1=newJButton("world");
JButton e1=newJButton("java");
JFrame a=newJFrame("Example AWT");
//Box n=Box.createHorizontalBox();
Box n=Box.createVerticalBox();
n.add(a1);
n.add(b1);
n.add(c1);
n.add(d1);
n.add(e1);
a.add(n);
a.setSize(500,500);
a.setVisible(true);
}
}
```

MENUS:

```
package layout;

import javax.swing.JFrame;

import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;

public class MenuExp extends JFrame {

    public MenuExp() {

        setTitle("Menu Example");
        setSize(150, 150);

        // Creates a menubar for a JFrame
        JMenuBar menuBar = new JMenuBar();

        // Add the menubar to the frame
        setJMenuBar(menuBar);

        // Define and add two drop down menu to the menubar
        JMenu fileMenu = new JMenu("File");
        JMenu editMenu = new JMenu("Edit");
        menuBar.add(fileMenu);
        menuBar.add(editMenu);

        // Create and add simple menu item to one of the drop down menu
        JMenuItem f1 = new JMenuItem("New");
        JMenuItem f2 = new JMenuItem("Open");
        JMenuItem f3 = new JMenuItem("Exit");
        JMenuItem e1 = new JMenuItem("Cut");
        JMenuItem e2 = new JMenuItem("Copy");
        JMenuItem e3 = new JMenuItem("Paste");

        fileMenu.add(f1);
        fileMenu.add(f2);
        fileMenu.add(f3);

        editMenu.add(e1);
        editMenu.add(e2);
        editMenu.add(e3);

    }
    public static void main(String[] args) {
        MenuExp me = new MenuExp();
        me.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        me.setVisible(true);
    }
}
```

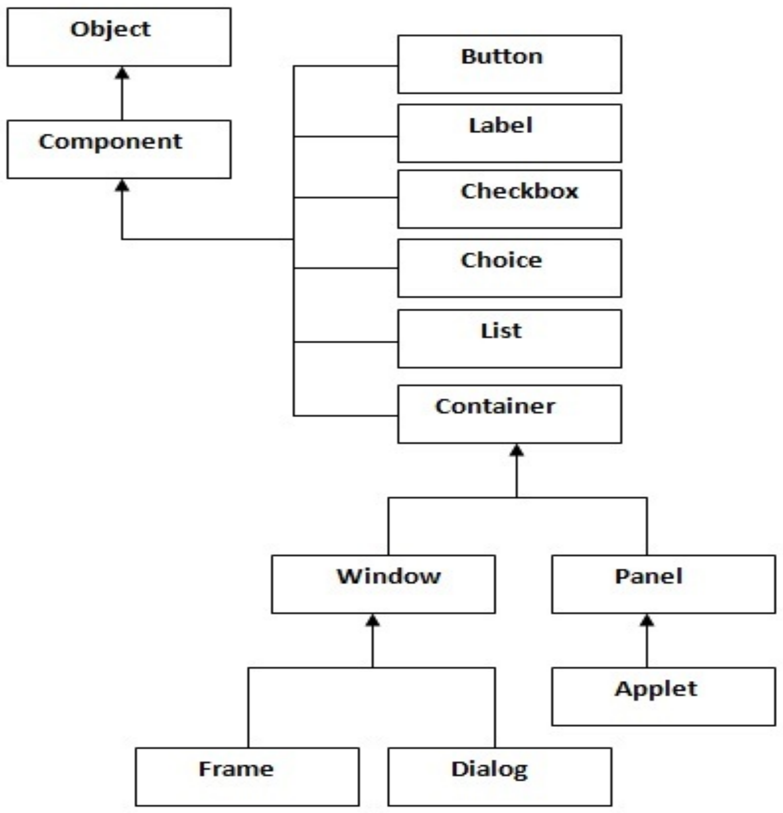


JAVA AWT :

- 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.



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.

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

Code:

```
package layout;
import java.awt.*;
import java.awt.event.*;
class Awt1 extends Frame implements ActionListener{
TextField tf;
Awt1(){
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);
b.addActionListener(this);
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}

public static void main(String args[]){
new Awt1();
```

}

}

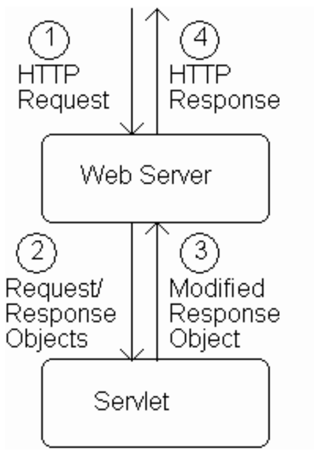
SERVLETS:

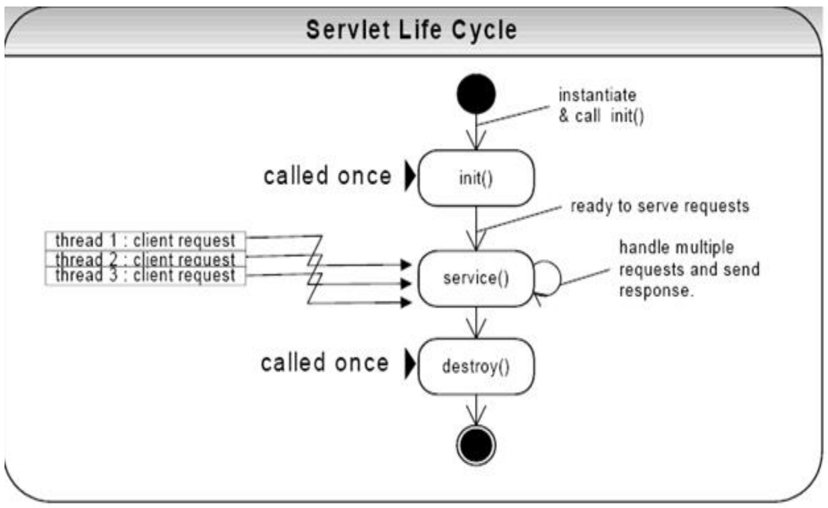
Difference Between Static and Dynamic HTML?

- Static: HTML document is retrieved from the file system and returned to the client
- Dynamic: HTML document is generated by a program in response to an HTTP request

– Java servlets are one technology for producing dynamic server responses

- Servlet is a class instantiated by the server to produce a dynamic response




Servlet Life Cycle
instantiate & call init()
called once
init()
ready to serve requests
thread 1 : client request
thread 2 : client request
thread 3 : client request
service()
handle multiple requests and send response.
called once
destroy()

What are all the Servlet API life cycle methods

Servlet API life cycle methods

- init(): called when servlet is instantiated; must return before any other methods will be called
- service(): method called directly by server when an HTTP request is received; default service() method calls doGet() (or related methods covered later)
- destroy(): called when server shuts down

PARAMETER DATA:

- The request object (which implements HttpServletRequest) provides information from the HTTP request to the servlet
- One type of information is parameter data, which is information from the query string portion of the HTTP request

query string with one parameter

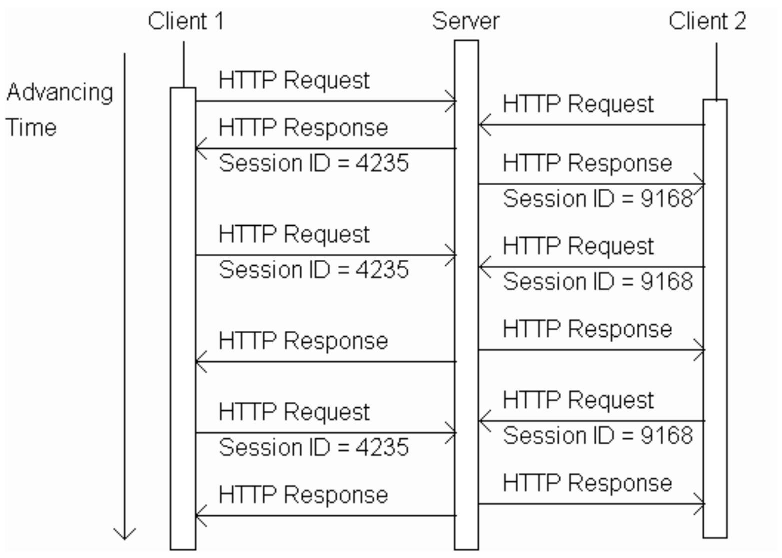
```
http://www.example.com/servlet/PrintThis?arg=aString
```

parameter name: arg
parameter value: aString
GET vs. POST method for forms:

- GET:
  - Query string is part of URL
  - Length of query string may be limited
  - Recommended when parameter data is not stored but used only to request information (*e.g.*, search engine query)
- POST:
  - Query string is sent as body of HTTP request
  - Length of query string is unlimited
  - Recommended if parameter data is intended to cause the server to update stored data
  - Most browsers will warn you if they are about to resubmit POST data to avoid duplicate updates

2. SESSIONS

- A session refers to the entire interaction between a client and a server from the time of the client's first request, which generally begins the session, to the time the session is terminated.
- The session could be terminated by the client's request, or the server could automatically close it after a certain period of time.
- Without session management, each time a client makes a request to a server, it's a brand new user with a brand new request from the server's point of view.



.

Server knows that all of these requests are from the same client. The set of requests is known as a *session*

SESSION TRACKING CODE:

Login.HTML:

```
<html>
  <head>
    <title>TODO supply a title</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <form name="f1" action="Serv12" method="post">
      Enter empid:<input type="text" name="eid"/>
      Enter the name:<input type="text" name="name"/>
      <input type="submit" value="click">
    </form>
  </body>
</html>
```

Serv12.java

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.DriverManager;
import javax.servlet.RequestDispatcher;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

@WebServlet(urlPatterns = {"/Serv12"})
```

```
public class Serv12 extends HttpServlet
{
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
IOException
{
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try
    {
      int e_id;
      String Name;
      e_id=Integer.parseInt(request.getParameter("eid"));
      Name=request.getParameter("name");

      HttpSession session = request.getSession();
      session.setAttribute("ses_attr",Name);

       //PREPARED STATEMENT
      String url="jdbc:mysql://localhost:3306/emp?zeroDateTimeBehavior=convertToNull";
      String query2="SELECT * FROM employee where empid="+e_id;
      Class.forName("com.mysql.jdbc.Driver");
      java.sql.Connection connect=DriverManager.getConnection(url,"root","focus");
      java.sql.PreparedStatement stmt2=connect.prepareStatement(query2);
      java.sql.ResultSet rs=stmt2.executeQuery();

      while(rs.next())
      {
      if(rs.getInt(1)==e_id && rs.getString(2).equals(Name) )
      {
      RequestDispatcher rd=request.getRequestDispatcher("Success.jsp");
```

```
            rd.forward(request, response);
            }
            else
            {
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet Serv12</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1> Invalid User</h1>");
            out.println("</body>");
            out.println("</html>");
            }
         }
        }
       catch(Exception e)
       {
       out.println(e);
       }
    }
}
```

Success.JSP:

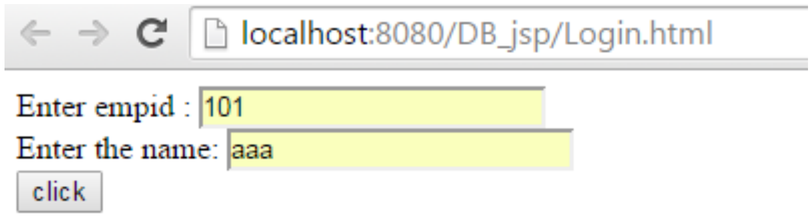
```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
```

```
  <body>
     <%
    String ses_name=(String) session.getAttribute("ses_attr");
    if(ses_name!="")
    {
    out.println("Welcome:"+ses_name);
    out.println("You are successfully logged in!");
    }
    else
    {
      out.println("Page cannot be found");
    }
   %>
   </body>
</html>
```

Query:

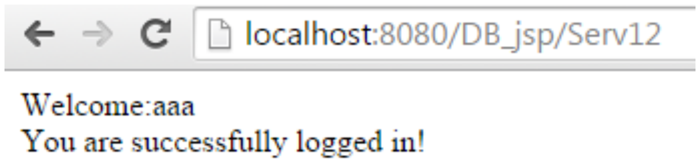
create table Student(empid numeric(20),name varchar(20));

OUTPUT:



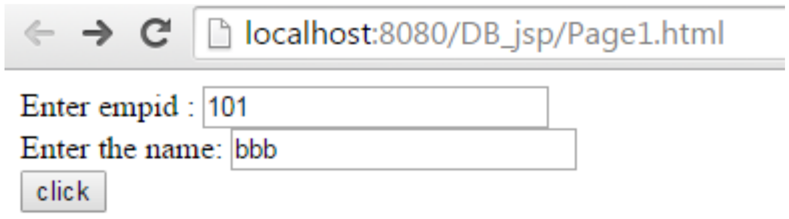
localhost:8080/DB_jsp/Login.html

Enter empid : 101
Enter the name: aaa
click



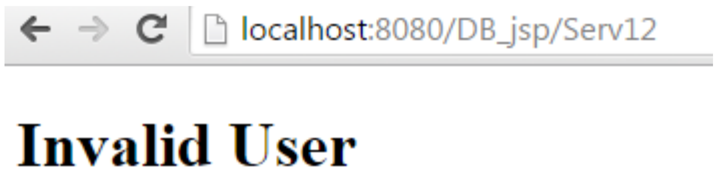
localhost:8080/DB_jsp/Serv12

Welcome:aaa
You are successfully logged in!



localhost:8080/DB_jsp/Page1.html

Enter empid : 101
Enter the name: bbb
click



localhost:8080/DB_jsp/Serv12

Invalid User

HttpSession interface:

METHODS

Method	Description
public HttpSession getSession()	Will cause one session to be created.

public HttpSession getSession(boolean)	true = will cause one to be created; false = will return null (no session)
public Object getAttribute(String name)	Returns the object bound with the specified name in this session, or null if no object is bound under the name.
public void setAttribute(String name, Object value)	Binds an object to this session, using the name specified.
public void removeAttribute(String name)	Removes the object bound with the specified name from this session.

Session termination:

- By default, each session expires if a server-determined length of time elapses between a session's HTTP requests
  - Server destroys the corresponding session object
- Servlet code can:
  - Terminate a session by calling invalidate() method on session object
  - Set the expiration time-out duration (secs) by calling setMaxInactiveInterval(int)

COOKIES

- A cookie is a name/value pair in the Set-Cookie header field of an HTTP response
- The main difference between cookies and sessions is that cookies are stored in the user's browser, and sessions are not.
- A cookie can keep information in the user's browser until deleted.
- If a person has a login and password, this can be set as a cookie in their browser so they do not have to re-login to your website every time they visit.

TABLE 6.3: Key `Cookie` class methods.

Method	Purpose
`Cookie(String name, String value)`	Constructor to create a cookie with given name and value
`String getName()`	Return name of this cookie
`String getValue()`	Return value of this cookie
`void setMaxAge(int seconds)`	Set delay until cookie expires. Positive value is delay in seconds, negative value means that the cookie expires when the browser closes, and 0 means delete the cookie.

Servlets can set cookies explicitly

- Cookie class used to represent cookies
- **request.getCookies()** returns an array of Cookie instances representing cookie data in HTTP request
- **response.addCookie(Cookie)** adds a cookie to the HTTP response

Cookies.java:

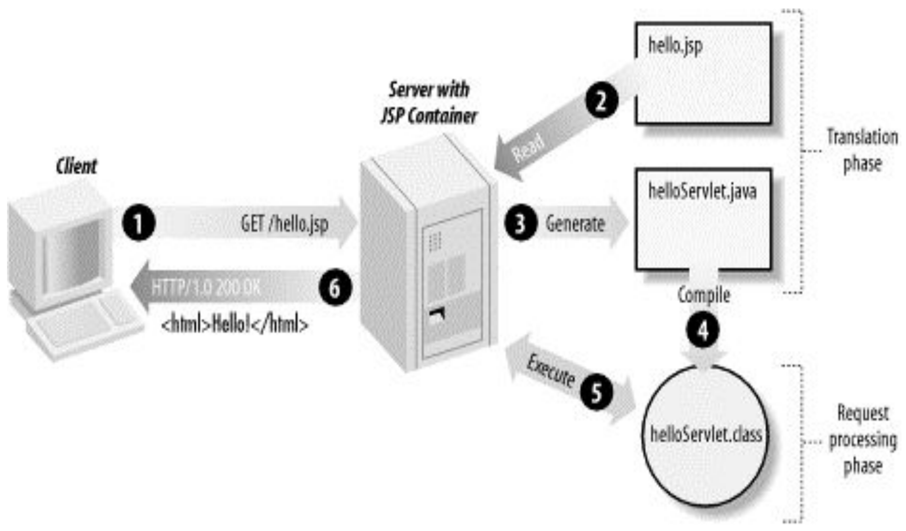
```
public class Cookies extends HttpServlet
{
  private static final int oneYear=60*60*24*365;
  protected void processRequest(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
                              int count = 0;
                              Cookie[] cookies = request.getCookies();
                                if(cookies != null)
                                {
                                        for(int i=0; (i<cookies.length) && (count==0); i++)
                                        {
                                                if(cookies[i].getName().equals("COUNT"))
                                                {
```

```
                                        count = Integer.parseInt(cookies[i].getValue());
                                        }
                                }
                        }
count++;
Cookie cookie = new Cookie("COUNT", new Integer(count).toString());
cookie.setMaxAge(oneYear);
response.addCookie(cookie);
out.println("<p>You have visited content type page " +count+" time(s) \n in the past year , or
since clearing your cookies</p>");
out.close();
}
finally
{
 out.close();
 }
 }
```

JAVA SERVER PAGES

vs. Pure Servlets: It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.

- **Servlet is html in java**
- **JSP is java in html**



JSP CODE:

```
<HTML>
<HEAD><TITLE>Welcome</TITLE></HEAD>        }  Template Data
<BODY>
<H3>Welcome!</H3>
<jsp:declaration>
        String st = "Hai";
        String st2 = "World";
</jsp:declaration>

<jsp:scriptlet>
  out.println("st + st2 is " + (st+st2));
</jsp:scriptlet>

<P>Today is <%= new java.util.Date() %>.</P>-----   →Java Code
</BODY>
</HTML>
```

Conversion of JSP to SERVLET Code:

This conversion is very simple in which all **template text is converted to println( ) statements** and all JSP elements are converted to Java code that implements the corresponding dynamic behavior of the page.

```
out.println("<HTML>");
out.println("<HEAD><TITLE>Welcome</TITLE></HEAD>");
out.println("<BODY>");
out.println("<H3>Welcome!</H3>");
   String st = "Hai";
   String st2 = "World";
out.println("st + st2 is " + (st+st2));
out.println("<P>Today is "+new java.util.Date()+".</P>");
out.println("</BODY>");
out.println("</HTML>");
```

JSP MARKUP

- Three types of markup elements:
  - Scripting
  - Directive
  - Action

SCRIPTING:

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

- Scriptlet tag (<% …%>)
- Expression tag (<%= statement %>)
- Declaration tag (<%! int data=50; %>)

DIRECTIVE

- Instructs JSP translator

PAGE directive:

The **page** directive is used to provide instructions to the container that pertain to the current JSP page. You may code page directives anywhere in your JSP page. By convention, page directives are coded at the top of the JSP page.

Syntax of page directive:

<%@ page attribute="value" %>

You can write XML equivalent of the above syntax as follows:

<jsp:directive.page attribute="value" />

INCLUDE DIRECTIVE :

The **include** directive is used to includes a file during the translation phase. This directive tells the container to merge the content of other external files with the current JSP during the translation phase. You may code *include* directives anywhere in your JSP page.

The general usage form of this directive is as follows:

<%@ include file="relative url" >

You can write XML equivalent of the above syntax as follows:

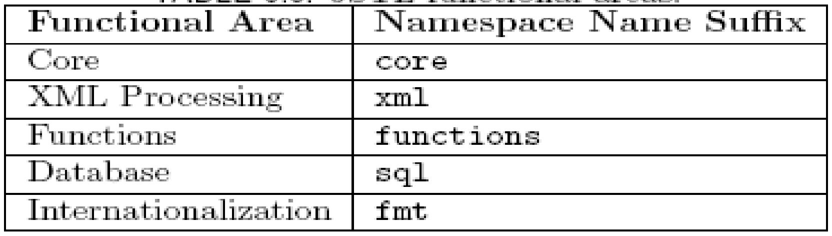
<jsp:directive.include file="relative url" />

ACTION:

- Standard: provided by JSP itself
- Custom: provided by a tag library such as JSTL.

- **JSTL** is divided into several functional areas, each with its own namespace:

TABLE 8.6: JSTL functional areas.



Functional Area	Namespace Name Suffix
Core	`core`
XML Processing	`xml`
Functions	`functions`
Database	`sql`
Internationalization	`fmt`

Namespace prefix is
`http://java.sun.com/jsp/jstl/`

JSTL CODE:

```
<%@ page import="java.io.*,java.util.*,java.sql.*"%>

<%@ page import="javax.servlet.http.*,javax.servlet.*" %>

<%@ tagliburi="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<%@ tagliburi="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>

<html>

<head>

<title>Displaying Employee list</title>

</head>

<body>

<sql:setDataSourcevar="snapshot" driver="com.mysql.jdbc.Driver"

url="jdbc:mysql://localhost:3306/mysql?zeroDateTimeBehavior=convertToNull"

user="root"  password="focus"/>

<sql:querydataSource="${snapshot}" var="result">

SELECT * from Emp

</sql:query>

<table border="1" width="100%">
```

```
        <tr>
        <th>Emp ID</th>
        <th>Name</th>
        <th>Age</th>
        </tr>
        <c:forEachvar="row" items="${result.rows}">
        <tr>
        <td><c:out value="${row.EmpId}"/></td>
        <td><c:out value="${row.Name}"/></td>
        <td><c:out value="${row.Age}"/></td>
        </tr>
        </c:forEach>
        </table>
        </body>
</html>
```