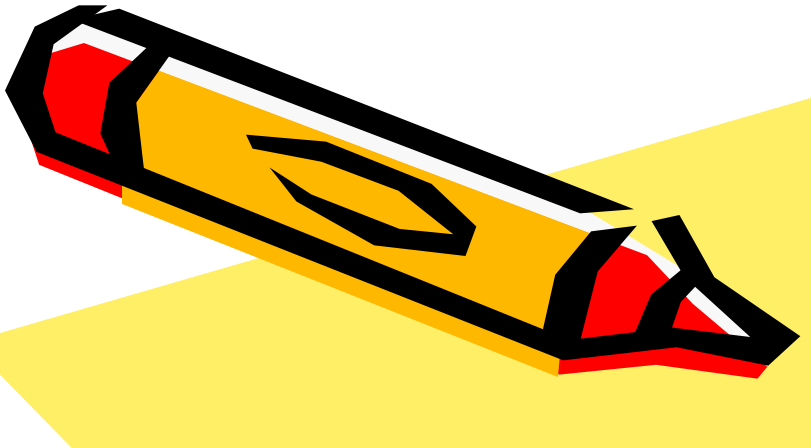
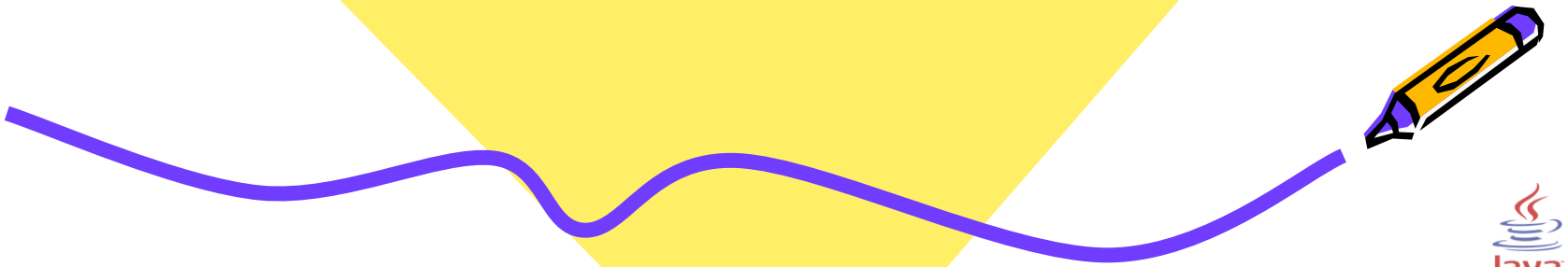


Understanding JSP



Agenda

- Introduction to JSP.
- JSP Comments.
- JSP directives.
- JSP Scripting element.
- JSP Actions
- Custom Tag

ABOUT REQUEST AND RESPONSE

All the pieces. On one page.

The user types a URL.

The browser creates an HTTP GET request.

```
GET /test1/Beer1.html HTTP/1.1
Host: www.wickedlysmart.com
User-Agent: Mozilla/5.0 (Macintosh...
...
```

The HTTP GET is sent to the server.

Server

The server finds the page...

Beer1.html

```
<html><body>
<h1 align=center>Beer Login Page</h1>
<form>
  Select a beer type or buy beer
  making supplies?<p>
  <input type=radio name=select
    value=Select> Select a beer<br>
  <input type=radio name=select
    value=Buy> Buy supplies<br><br>
  <center>
    <input type=SUBMIT>
  </center>
</form>
</body></html>
```

And generates an HTTP response.

```
HTTP/1.1 200 OK
Set-Cookie: . . .
. . .
<html><body>
<h1 align=center>Beer Login Page</h1>
<form>
  Select a beer type or buy beer . . .
```

The HTTP response is sent to the browser.

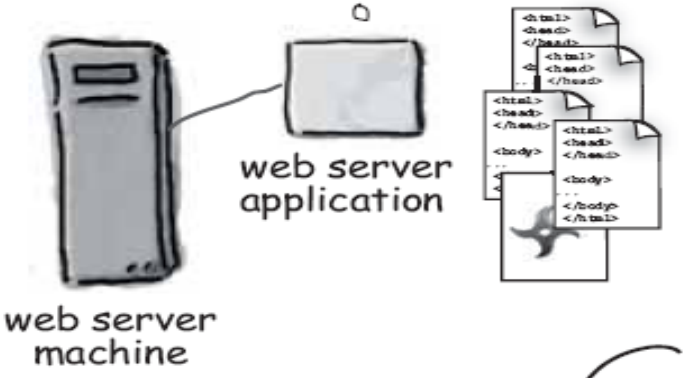
Client

The browser renders the HTML.

Client looks forward to a successful beer transaction.

This is what I do. Ask me for a page, I find it, and I hand it back. With a few headers. But that's it. Do NOT ask me to, like, *do* anything to the page.

A static page just sits there in a directory. The server finds it and hands it back to the client as-is. Every client sees the same thing.



These pages go straight to the client just exactly as they were put on the server.

Static Page Story

But what if I want, say, the current time to show up on my page? What if I want a page that has something *dynamic*? Can't I have something like a *variable* inside my HTML?

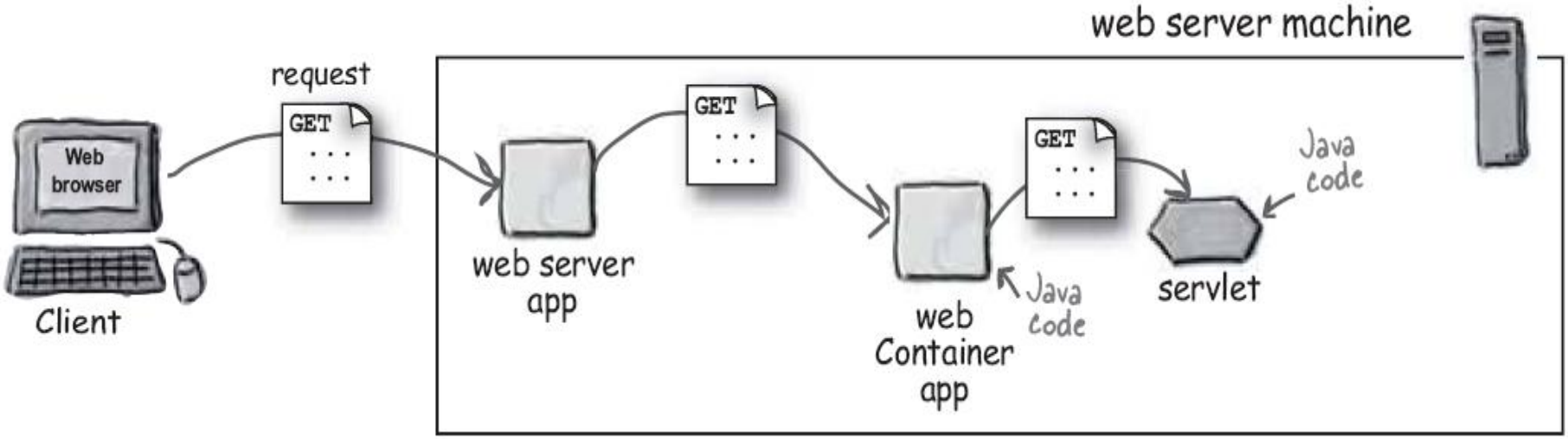
What if we want to stick something variable inside the HTML page?

```
<html>
<body>
The current time is [insertTimeOnServer].
</body>
</html>
```

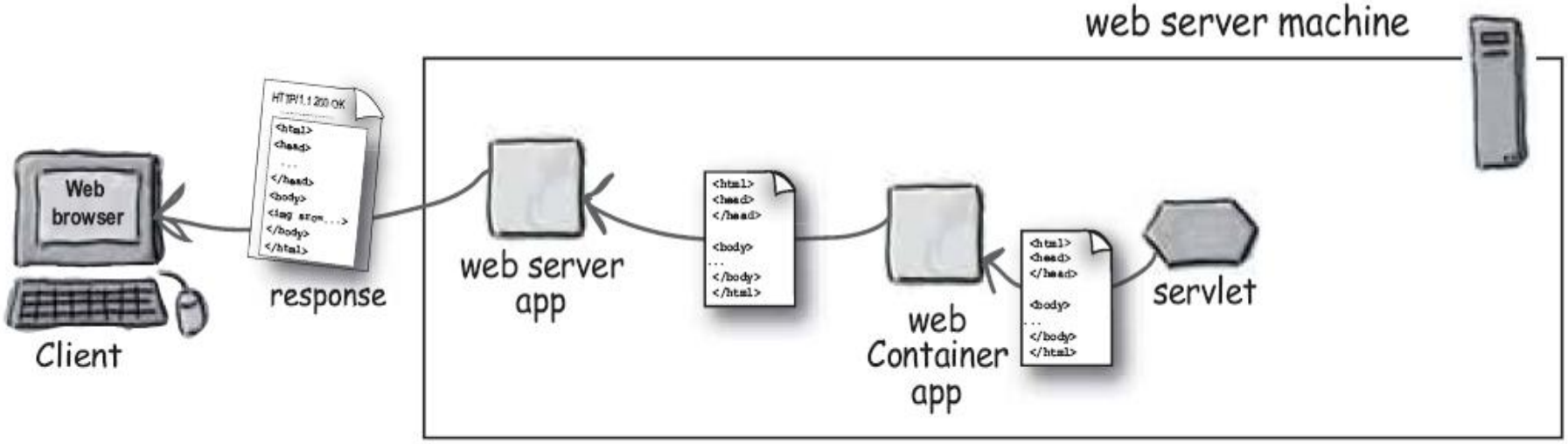
For the Dynamic Page

```
<html>
<body>
The Current time is <%=new java.util.Date() %>
</body>
</html>
```

Sample Example



Dynamic Web-Application Cycle



Container Services

1) Communication Support
2) LifeCycle Management
3) MultiThreading Support
4) Declarative Security
5) Jsp Support

What does the Container give you?

We know that it's the Container that manages and runs the servlet, but *why*? Is it worth the extra overhead?

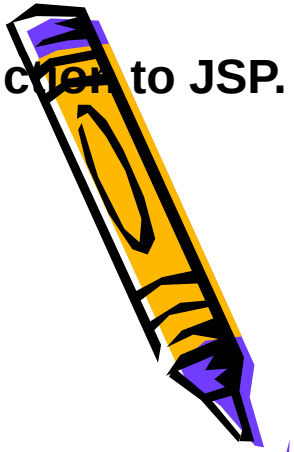
Communications support The container provides an easy way for your servlets to talk to your web server. You don't have to build a ServerSocket, listen on a port, create streams, etc. The Container knows the protocol between the web server and itself, so that your servlet doesn't have to worry about an API between, say, the Apache web server and your own web application code. All you have to worry about is your own business logic that goes in your Servlet (like accepting an order from your online store).

Lifecycle Management The Container controls the life and death of your servlets. It takes care of loading the classes, instantiating and initializing the servlets, invoking the servlet methods, and making servlet instances eligible for garbage collection. With the Container in control, *you* don't have to worry as much about resource management.

Multithreading Support The Container automatically creates a new Java thread for every servlet request it receives. When the servlet's done running the HTTP service method for that client's request, the thread completes (i.e. dies). This doesn't mean you're off the hook for thread safety—you can still run into synchronization issues. But having the server create and manage threads for multiple requests still saves you a lot of work.

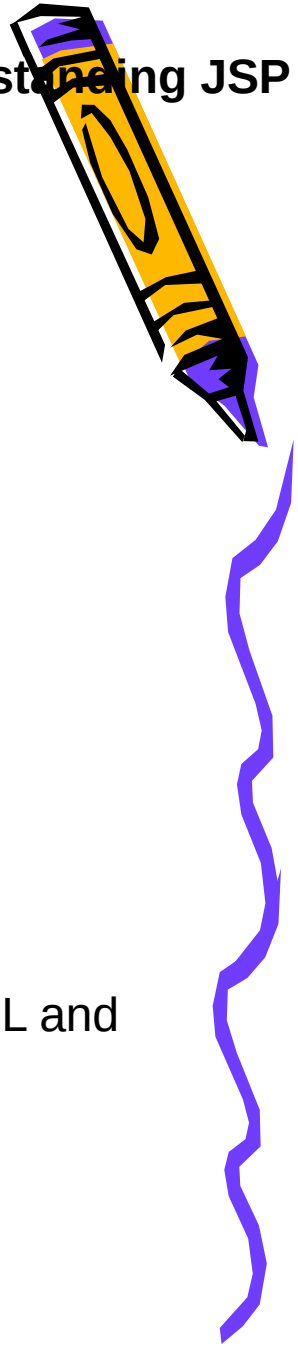
Declarative Security With a Container, you get to use an XML deployment descriptor to configure (and modify) security without having to hard-code it into your servlet (or any other) class code. Think about that! You can manage and change your security without touching and recompiling your Java source files.

JSP Support You already know how cool JSPs are. Well, who do you think takes care of translating that JSP



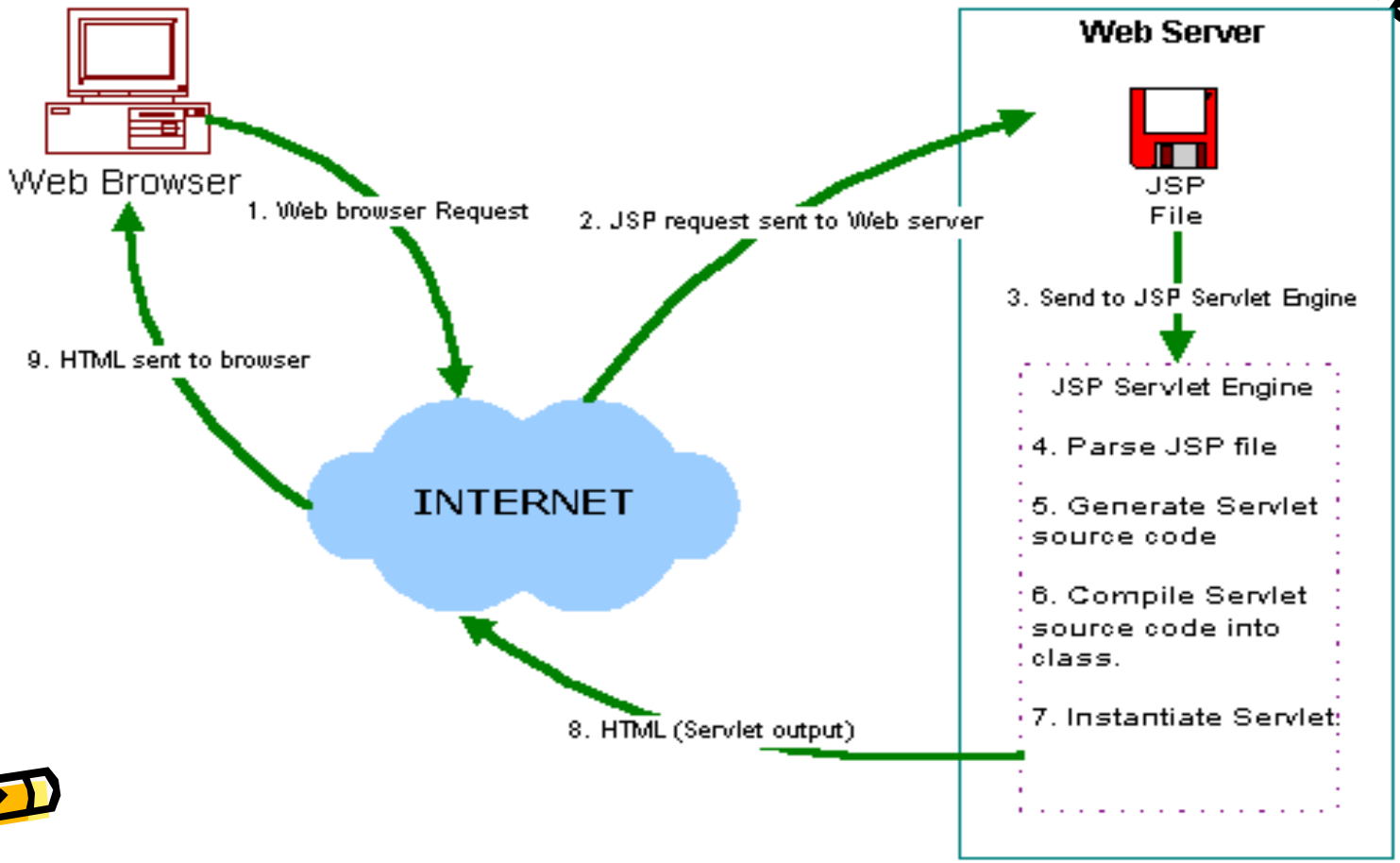
What Is a JSP ?

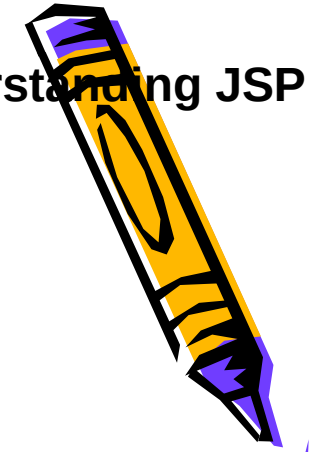
Understanding JSP



- JSP is a mixture of
  - standard HTML tags
  - web page content
  - some dynamic content using JSP constructs.
- Benefits
  - Write HTML
  - Read and Maintain HTML
  - Work Distribution , Different resources doing the HTML and layout programming
  - Separate Java Code and HTML Code

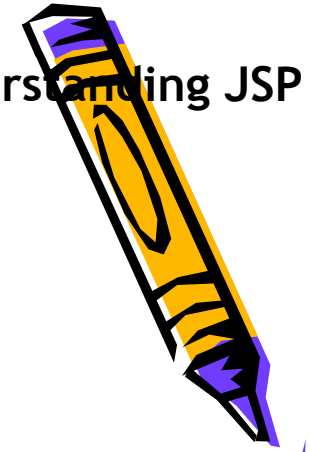
- What happens whenever a JSP Page is requested for the first time ?





- Loads the JSP page's servlet class
- Instantiates an instance of the servlet class
- Initializes the servlet instance by calling the jspInit method
- Invokes the _jspService method, passing a request and response object.
- If the container needs to remove the JSP page's servlet, it calls the jspDestroy method
- Unloads the servlet.

```
package hall;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet
 {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException
{
    PrintWriter out = response.getWriter();
    out.println("Hello World");
  }
}
```

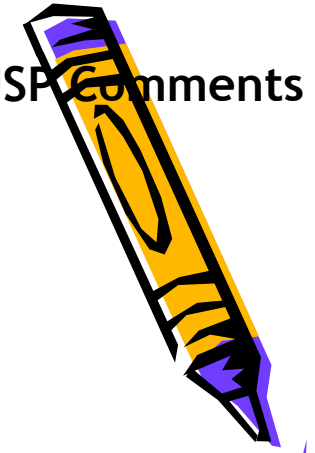


<%="Hello World "%>

Why use JSP?

- Ease of writing a servlet instead.
- Multi platform
- Advantages of Java.
- You can take one JSP file and move it to another platform,web server or JSP Servlet engine.

JSP Comments



Thumb Rules

- Purpose of the code and intention : WHY
- Code behaviour : How ?
- Clarify the code written and not state the obvious code written.
- Proper comments and not misleading comments
- Consistency in code format (location / format) .

JSP Comments

- Types of JSP Comments
  - Documenting the source JSP – Hidden Comment
    - <%-- Retrieve the rules from rule definition--%>
  - Documenting the HTML content. – Output Comment
    - <!-- The page was created on <%=new java.util.Date()%> -- >

JSP Directives

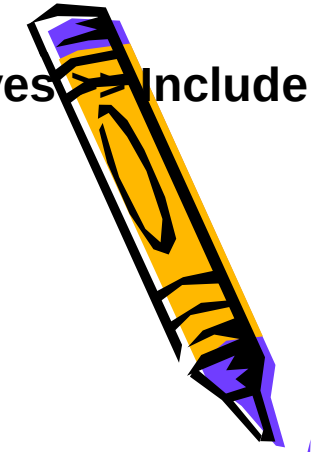
JSP Directives

JSP page which works for the entire JSP page. These directives apply different properties for the page like language support, page information and import etc.

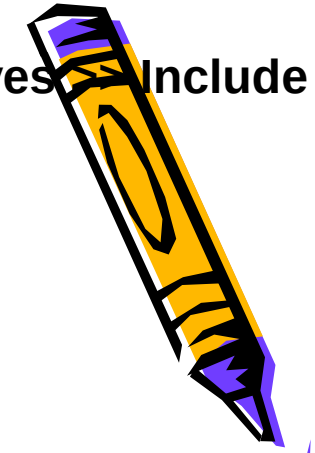
A JSP "directive" starts with <%@ characters

Types of directives

- Include
- Page
- Taglib



- Including Files at Page Translation Time
  - Format
    - <%@ include file="/include/Validation.js" %>
  - Purpose
    - To reuse JSP content in multiple pages, where JSP content affects main page
- Notes
  - Servers are not required to detect changes to the included file, and in practice many don't
  - Thus, you need to change the JSP files whenever the included file changes

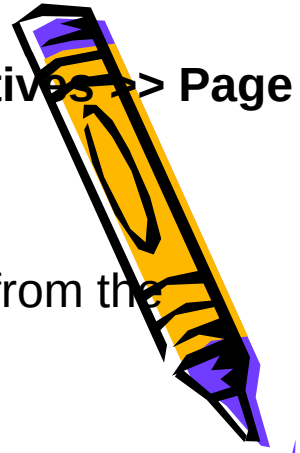


- Including Pages at Request Time
  - Format
    - <jsp:include page="/include/Visualattrib.jsp" flush="true">

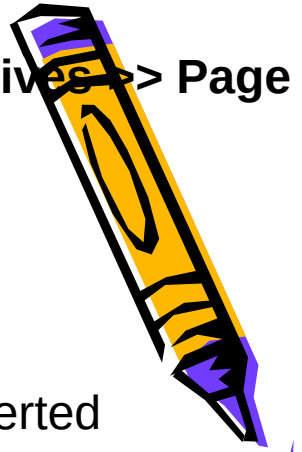
      <jsp:param name="tdFontWeight" value="lighter" />

      </jsp:include>
- Purpose
  - To reuse JSP, HTML, or plain text content
  - JSP content cannot affect main page:only output of included JSP page is used
  - To permit updates to the included content without changing the main JSP page(s)

JSP Directives -> Page



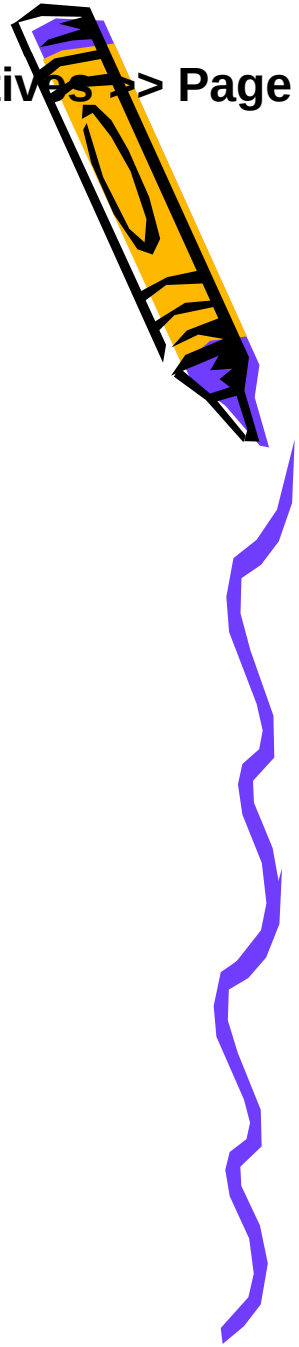
- Give high-level information about the servlet that will result from the JSP page
- Can control
  - Which classes are imported
  - What class the servlet extends
  - What MIME type is generated
  - How multithreading is handled
  - If the servlet participates in sessions
  - The size and behavior of the output buffer
  - What page handles unexpected errors



There are several options to the page directive.

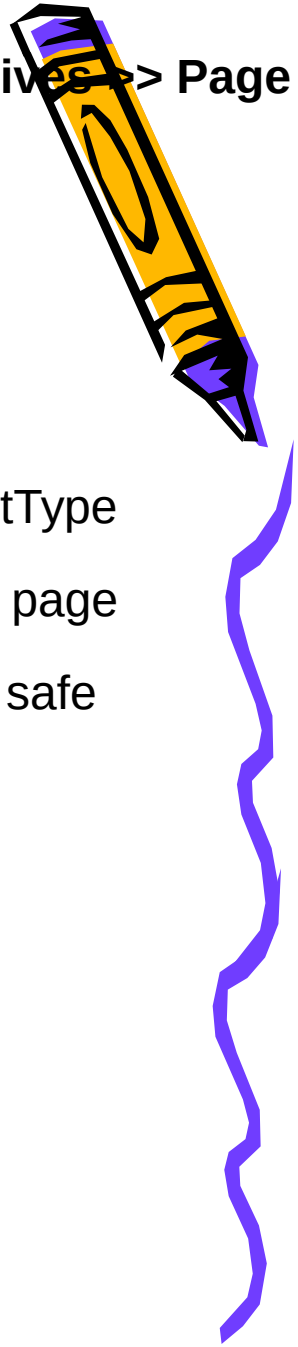
- import : Results in a Java import statement being inserted into the resulting file.
- contentType : specifies the content that is generated. This should be used if HTML is not used or if the character set is not the default character set.
- errorPage : Indicates the page that will be shown if an exception occurs while processing the HTTP request.
- isErrorPage If set to true, it indicates that this is the error page.
- isThreadSafe Indicates if the resulting servlet is thread safe.Default value is False

JSP Directives -> Page

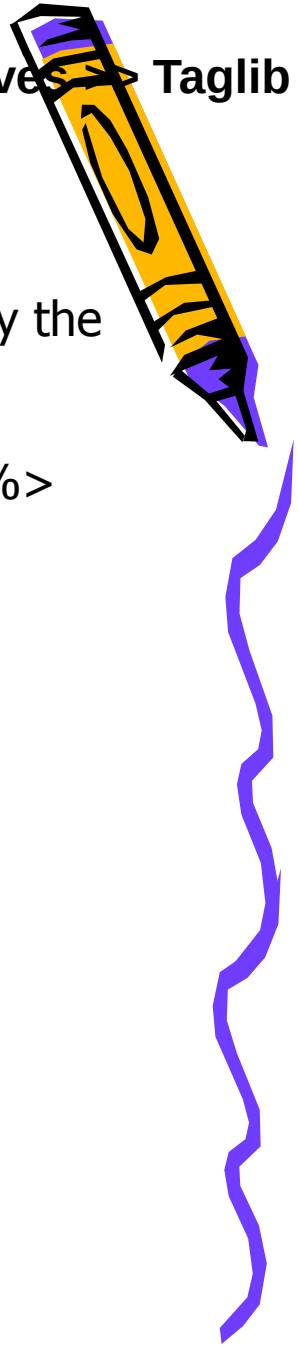


- Extends : Fully qualified class names
- Buffer : none or size(kb)
- autoFlush True or False
- Info : Information text such as author , version and copyright

JSP Directives -> Page

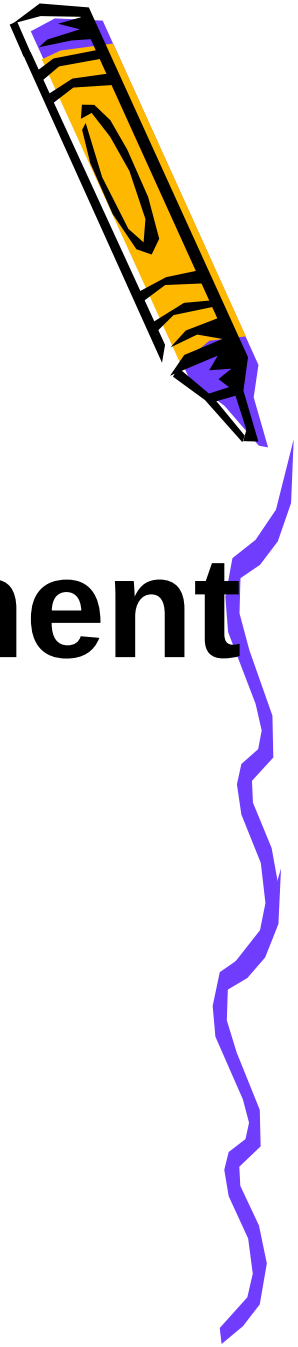


- Examples are like :
- <%@ page import="java.util.*" %> //example import
- <%@ page contentType="text/html" %> //example contentType
- <%@ page isErrorPage=false %> //example for non error page
- <%@ page isThreadSafe=true %> //example for a thread safe JSP
- <%@page language="java" session="true" errorPage="error.jsp" %>

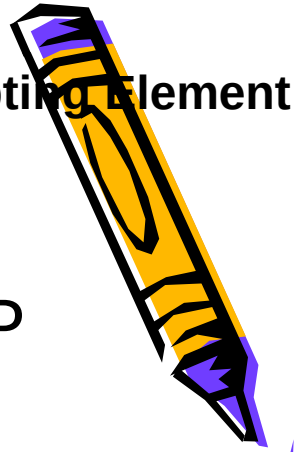


- A tag lib is a collection of custom tags that can be used by the page.

  <%@ taglib uri = "tag library URI" prefix = "tag Prefix" %>

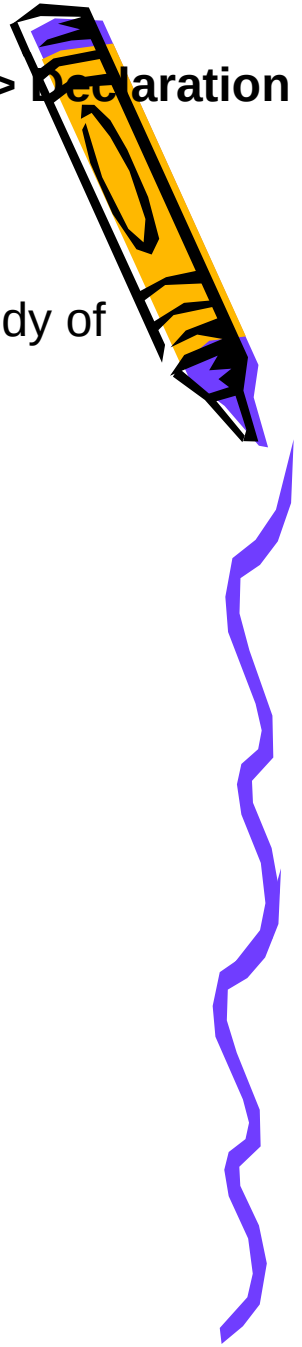


JSP Scripting Element

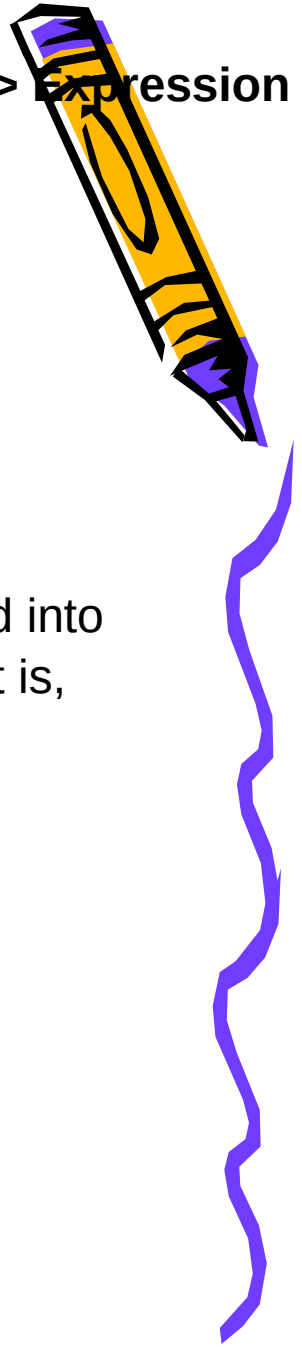


Types of Scripting Elements used in JSP

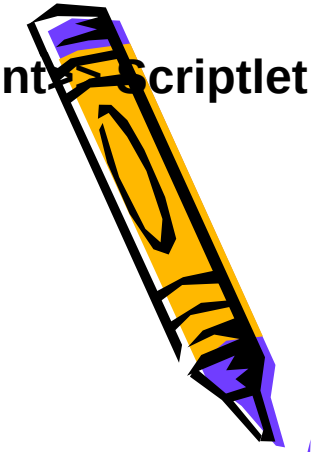
- Declarations
- Expressions
- Scriptlets



- A *declaration* tag places a variable definition inside the body of the java servlet class.
- Format: <%! code %>
- Result
    - Code is inserted verbatim into servlet's class definition,outside of any existing methods
- Examples
    - <%! private int someField = 5; %>
    - <%! private void someMethod(...) {...} %>



- Embed Java expressions in JSP pages
- Format: <%= expression %>
- Result
  - Expression evaluated, converted to String, and placed into HTML page at the place it occurred in JSP page, That is, expression placed in _jspService inside out.print
- Example : Hello_Expression.jsp



- Blocks of Java code inside the JSP.
- Format: <% code %>
- Result
  - Code is inserted verbatim into servlet's _jspService
  - By itself a scriptlet does not generate HTML
  - If a scriptlet wants to generate HTML, it can use a variable called "out".
- Example : name.jsp

Sample Example

JSP Implicit Objects

JSP provides certain Implicit Objects listed below

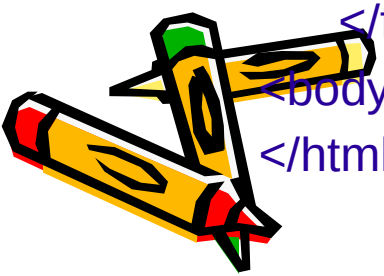
Object	Kind	Example
out	JSPWrier	out.println(“Hello Client “);
request	HttpServletRequest	String a= request.getParameter(“userid”);
response	HttpServletResponse	response.sendRedirect(“www.google.com”);
session	HttpSession	session.setAttribute(“userid”,a);
application	ServletContext	application.setAttribute(“global”,msg);
config	ServletConfig	String db=Config.getInitParameter(“db”);
page	Object	page.equals(Object o);
pageContext	PageContext	HttpSession s=pageContext.getSession(); ServletContext a=pageContext.getServletContext();
exception	JSPException	out.println(exception.getMessage());

Working on JSP

Login Check Using JSP Example

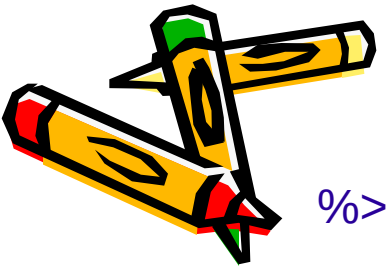
Step -1 Making User Interface

```
<html>
   <head>

      <title>JSP Page</title>
   </head>
<body>
<h1> Login Page </h1>
   <br>
    <form method="post" action="./logincheck.jsp">
       Enter the Userid <input type="text" name="t1"  ><br>
       Enter the Password<input type="password" name="t2"><br>
       <input type="submit" value="Submit" >
       <input type="reset" value="Reset" >
    </form>
   </body>
</html>
```



Step-2 Check Login Logic

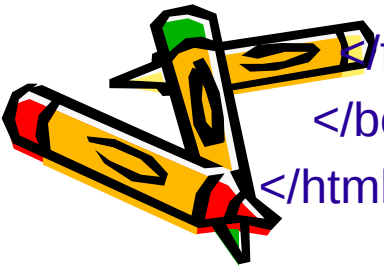
```
<%@page contentType="text/html"%>
<html>
    <head>
        <title>JSP Page</title>
    </head>
    <body>
<%!    String userid,password;    %>
    <%
    userid=request.getParameter("t1");
    password=request.getParameter("t2");
    if(userid.equals(password))
                    response.sendRedirect("./welcome.html");
     else
                {
                        out.println("Invalid Userid and Password !");
            }
     %> </body> </html>
```



Using Session API Example

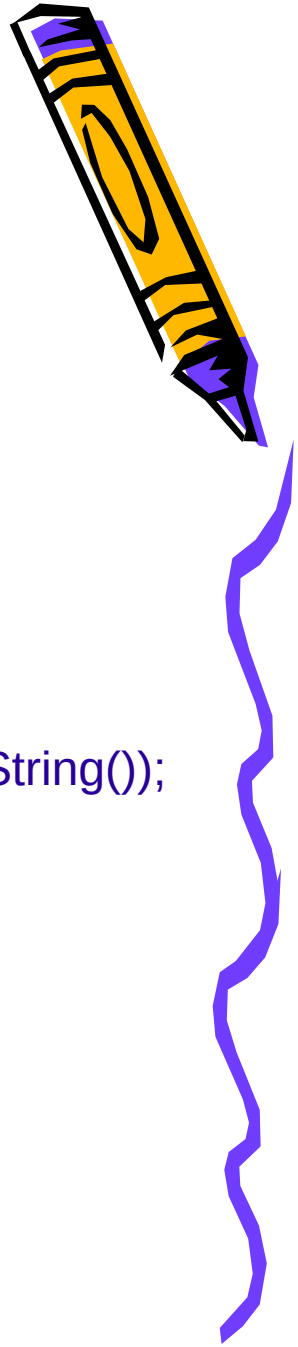
Step -1 index.jsp

```
<%@page contentType="text/html"%>
<html>
    <head>
        <title>JSP Page</title>
    </head>
    <body>
<h1> Login Page </h1>
    <br>
      <form method="post" action="./logincheck.jsp">
          Enter the Userid <input type="text" name="t1"  ><br>
          Enter the Password<input type="password" name="t2"><br>
          <input type="submit" value="Submit" >
          <input type="reset" value="Reset" >
      </form>
    </body>
</html>
```



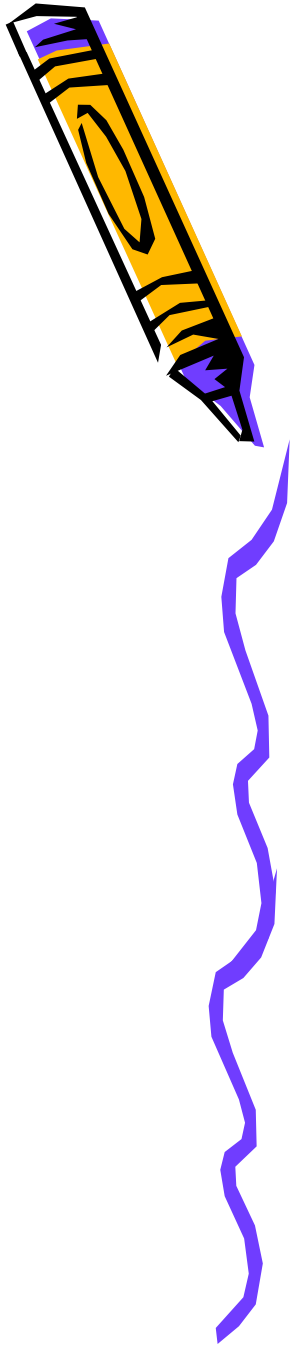
Step-2 logincheck.jsp

```
<%@page contentType="text/html"%>
<html>
   <body>
<%! String userid,password; %>
<%
   userid=request.getParameter("t1");
   password=request.getParameter("t2");
   if(userid.equals(password))
      {
      session.setAttribute("userid",userid);
            response.sendRedirect("./welcome.jsp");
      }
      else
         {
         out.println("Invalid Userid and Password !");
         }
%>
   </body>
</html>
```



Step-3 welcome.jsp

```
<%@page contentType="text/html"%>
<html>
    <body>
<%
    if(session.isNew())
        response.sendRedirect("./index.jsp");
        else
            out.println("Welcome "+session.getAttribute("userid").toString());
    %>

    </body>
</html>
```



JSP Actions

JSP actions

- Actions are XML tags that invoke built-in web server functionality
- Executed at runtime

JSP actions Conti..

jsp:include

- Java servlet temporarily hands the request and response off to the specified JSP.
- Control will then return to the current JSP, once the other JSP has finished.
- Attributes supported are page / flush.

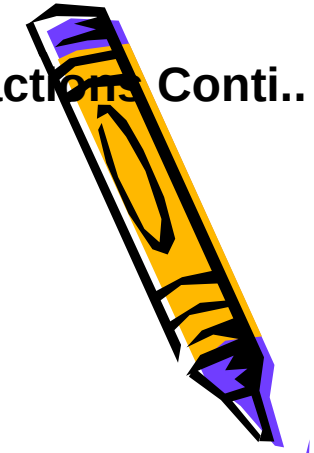
jsp:forward

- Used to hand off the request and response to another JSP or servlet.
- Control will never return to the current JSP.
- Attributes supported is page.

jsp:fallback

- The content to show if the browser does not support applets.

JSP actions Conti..



jsp:getProperty

- Gets a property from the specified JavaBean.

jsp:setProperty

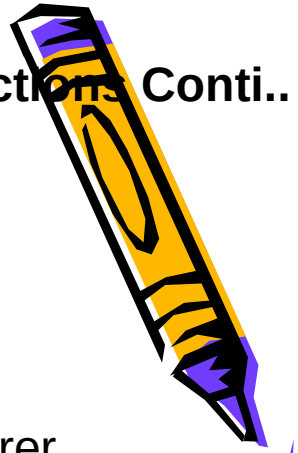
- Sets a property in the specified JavaBean.
- Attributes supported are name/ property / param /value.

jsp:param

- Can be used inside a jsp:include, jsp:forward or jsp:params block.
- Specifies a parameter that will be added to the request's current parameters.

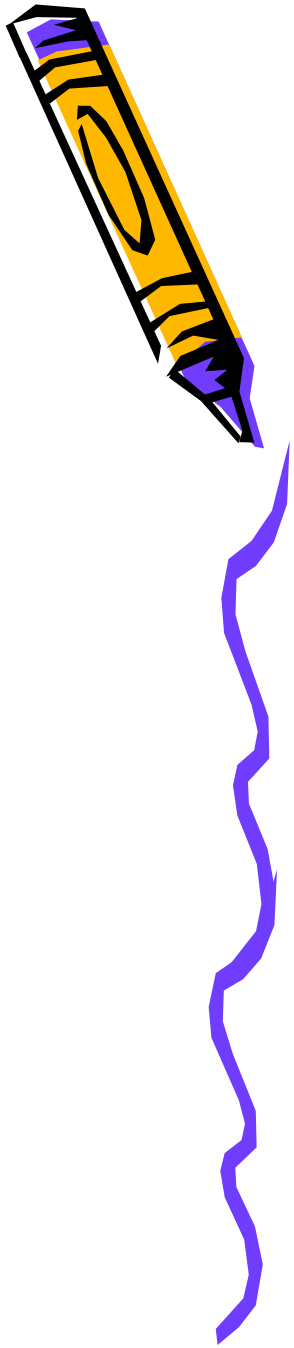
jsp:useBean

- Creates or re-uses a JavaBean available to the JSP page.
- Attributes supported are Id/ Class/Scope/Type/beanName



jsp:plugin

- Older versions of Netscape Navigator and Internet Explorer used different tags to embed an applet.
- This action generates the browser specific tag needed to include an applet.



Standard Action Example Codes

<jsp:include>

Example

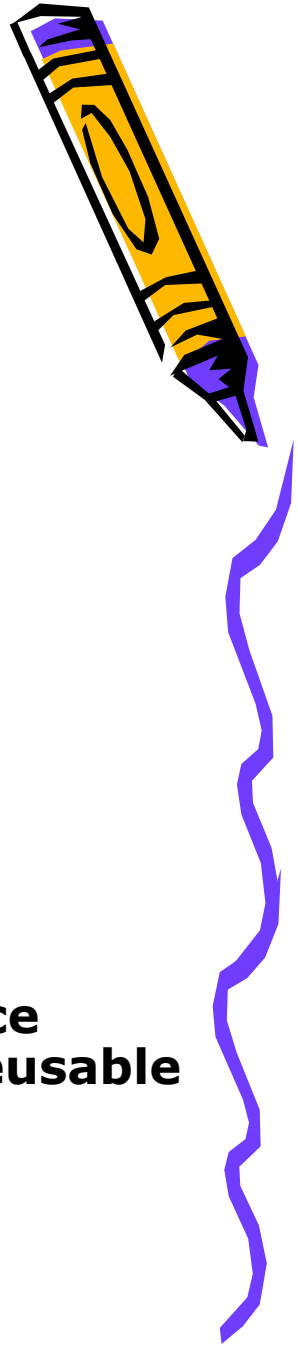
```
<jsp:include page=”header.jsp” flush=”true”/>
<jsp:include page=”footer.jsp” flush=”true” />
```

Passing Parameter in include

```
<jsp:include page=”header.jsp” flush=”true”>
<jsp:param name=”empno” value=”1001” />
<jsp:param name=”name” value=”Abcd” />
</jsp:include>
```

<jsp:forward> Example

```
<jsp:forward page=”result.jsp”>
<jsp:param name=”empno” value=”1001” />
</jsp:forward>
```



<jsp:usebean> Example

<jsp:usebean id=”obj” class=”pack.Emp” scope=”request”>
<jsp:setProperty name=”obj” property=”empno” value=”1001” param=”text1” />
or
<jsp:setProperty name=”obj” property=”*” />
This use when bean property and form field have same name.
</jsp:usebean>

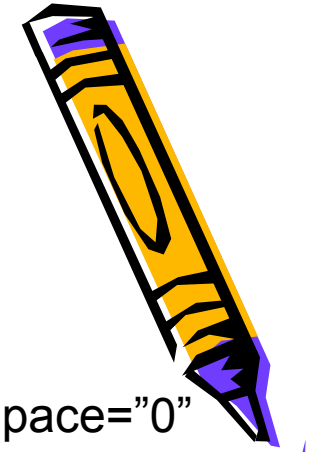
<jsp:usebean id=”obj” class=”pack.Emp” scope=”request”>
<jsp:getProperty name=”obj” property=”empno” />
</jsp:usebean>

Note: **Simple Java classes that have private instance members and setter and getter methods. It is a reusable component.**

Example: Emp Java Bean

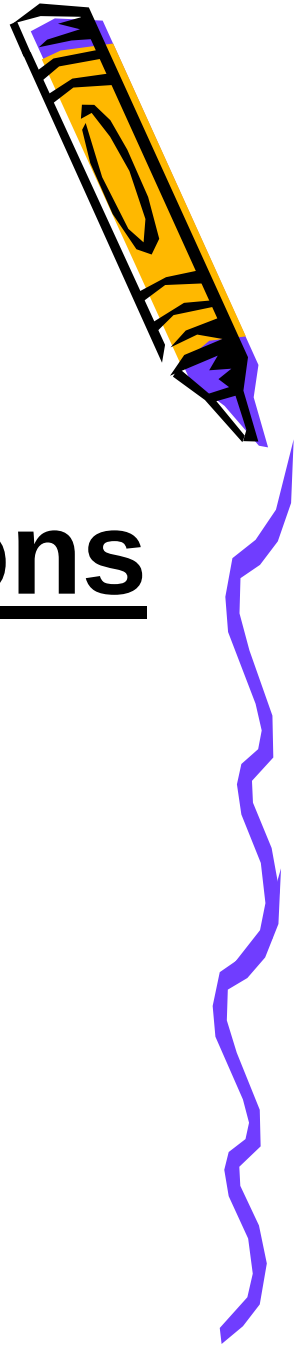
****

```
class Emp
{
	private int empno;
	private String name;
	public void setEmpno(int empno)  //Java Bean Rules
	{
		this.empno=empno;  //Shadowing
	}
public int getEmpno()
{
return this.empno;
}
public void setName(String name)
{
	this.name=name;
}
public String getName()
{
	return name;
}
}  //Class Close
```



<jsp:plugin> Example

<jsp:plugin type=”applet” code=”xyz.class” codebase=”/java” vspace=”0” hspace=”0” width =”60” height=”60”>
<jsp:fallback>
Cannot display this applet on your browser
</jsp:fallback>
</jsp:plugin>

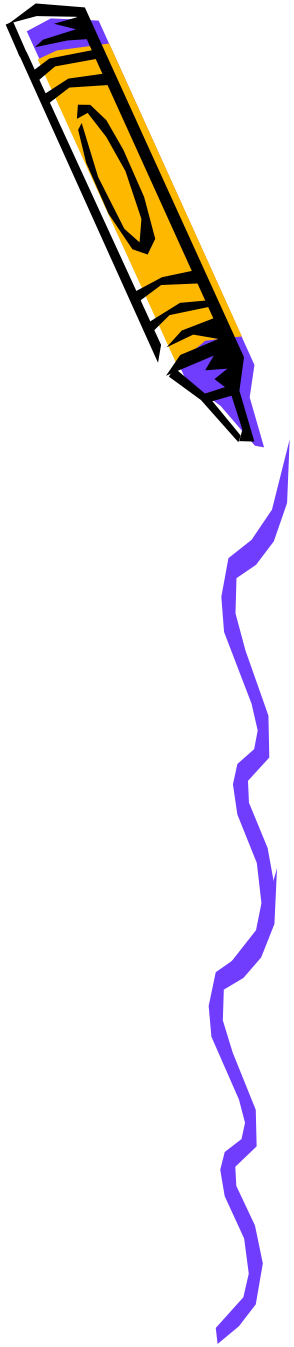


Working on JSP Actions

Using <jsp:include />

HEADER.JSP

```
<%@page contentType="text/html"%>

<html>
    <body>
    <h1>Welcome  <%=new java.util.Date() %></h1>
    <br>
    <hr>
    </body>
</html>
```

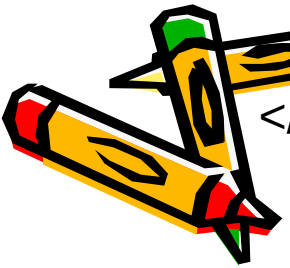


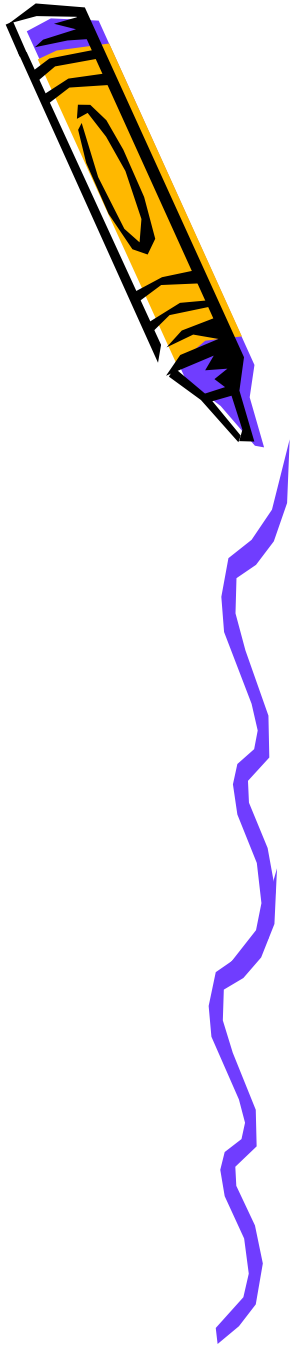
FOOTER.JSP

```
<%@page contentType="text/html"%>

<html>
  <body>
<hr>
<br>
<h1>Contact us Abcd@yahoo.com</h1>

    </body>
</html>
```

Main.jsp

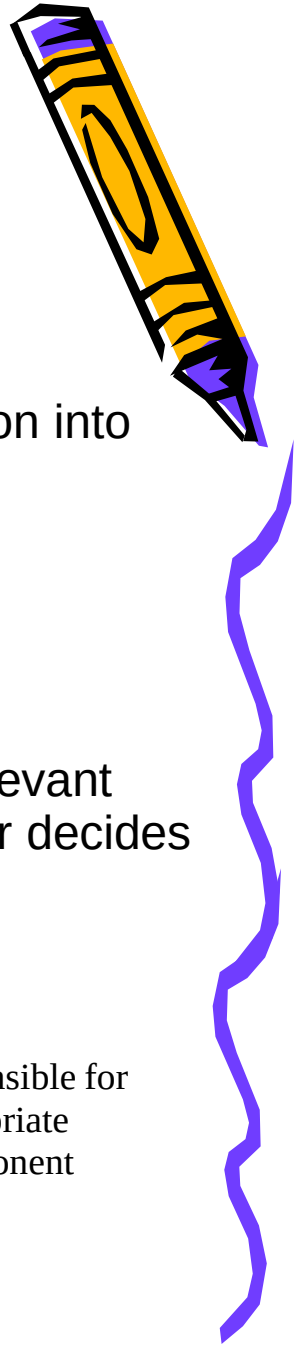
```
<%@page contentType="text/html"%>

<html>
    <body>
<jsp:include page="./Header.jsp" />
<br>

<h1> THIS IS Main Page</h1>
<br>
<img src=”abcd.jpg”>
<br>
 <jsp:include page="./Footer.jsp" />

    </body>
</html>
```





Making an MVC Application

MVC



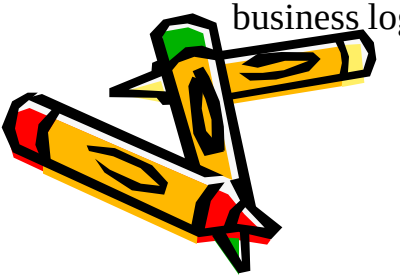
The basic idea of the MVC architecture is to divide the application into **three layers**:

Model data and logic (business logic methods).

View output displayed to the user

Controller takes the input from the user and passes it to the relevant model. The model returns and based on the result, the controller decides which view (page) to show the user

NOTE: Controller: Controller is intermediary between Model and View. Controller is responsible for receiving the request from client. Once request is received from client it executes the appropriate business logic from the Model and then produce the output to the user using the View component



Step-1 Creating View

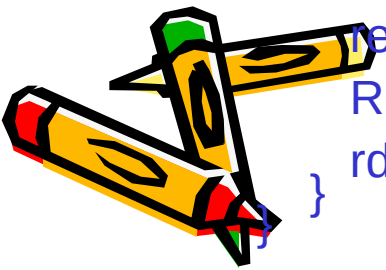
index.jsp

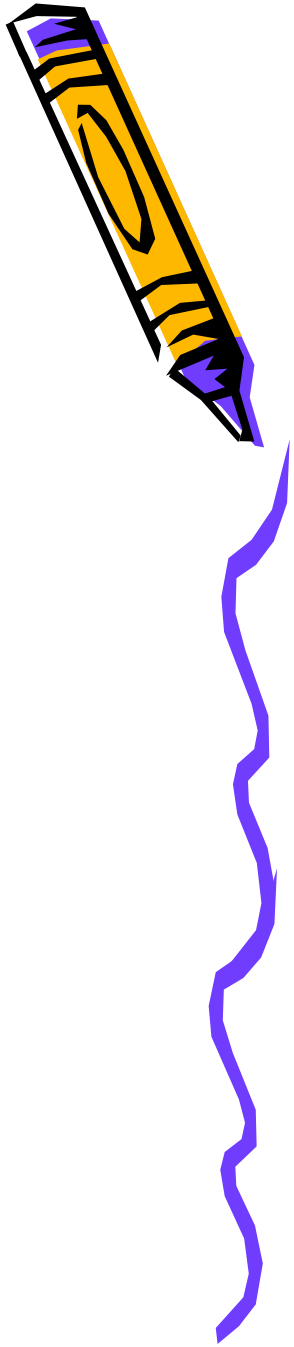
```
<%@page contentType="text/html"%>

<html>
    <body>

    <h1>View Page</h1>
    <br>
    <form method="post" action="./Controller">
        Enter Empno to Search <input type="text" name="t1"> <br>
        <input type="submit" >

    </form>

    </body>
</html>
```

Step-2 Creating Controller

Servlet File

```
import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Controller extends HttpServlet
{
protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException
    {
        String empno=request.getParameter("t1");
        Model obj=new Model();
        obj.setEmpno(empno);
        String temp=obj.fetch(empno);
        request.setAttribute("result",temp);
        RequestDispatcher rd=request.getRequestDispatcher("/result.jsp");
        rd.forward(request,response);
    }
}
```

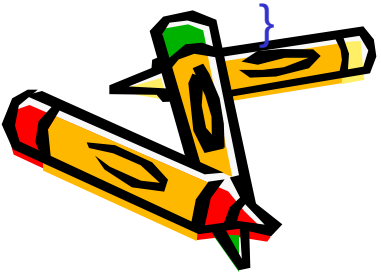




Step-3 Creating Model

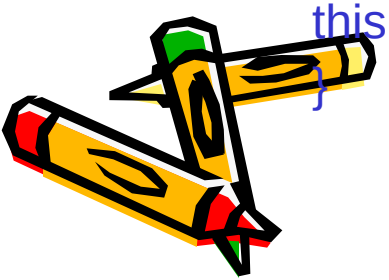
Simple Java Bean + Business Methods

```
import java.util.*;

public class Model
{
private String empno;
private String name;
private String address;
private boolean att;
private String gender;
ArrayList l=null;

public Model()
{
   l=new ArrayList();
}
```



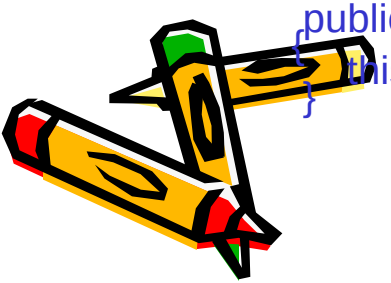
Model Continued.....

```
public String getEmpno()
{
return empno;
}

public void setEmpno(String empno)
{
 this.empno = empno;
}

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



Model Continued.....

```
public String getAddress()
{
return address;
}
 public void setAddress(String address)
{
    this.address = address;
}
 public boolean isAtt()
{
    return att;
}
 public void setAtt(boolean att)
{
 this.att = att;
}
 public String getGender()
{
 return gender;
}
 public void setGender(String gender)
{
 this.gender = gender;
}
```

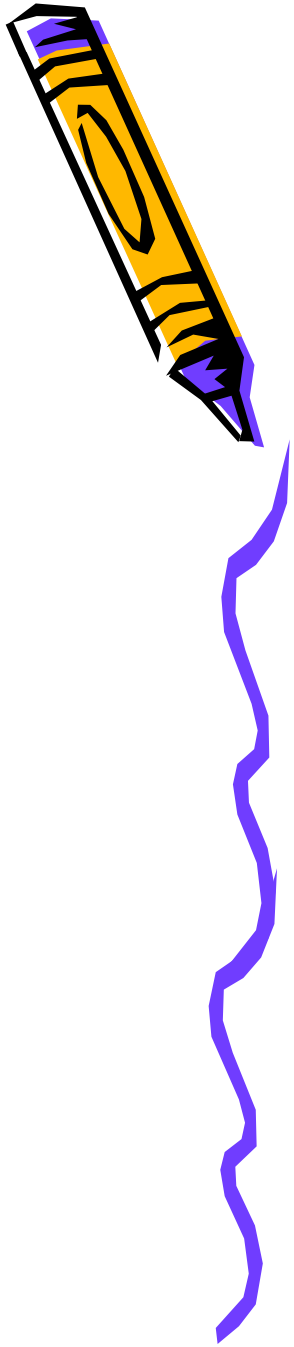


Model Continued.....

```
public void createList()
{
Model amit=new Model();
amit.setEmpno("1001");
amit.setAtt(true);\
amit.setGender("male");
amit.setName("Amit");
amit.setAddress("Shakti Nagar");
Model ram=new Model();
ram.setEmpno("1002");
ram.setAtt(true);
ram.setAddress("Roop Nagar");
ram.setGender("male");
ram.setName("Ram");
l.add(amit);
l.add(ram);
}
```

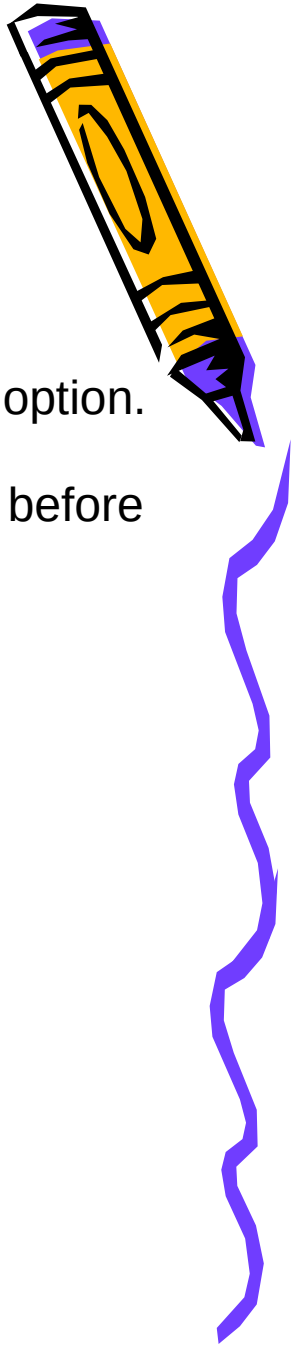
Model Continued.....

```
public String fetch(String empno)
  {
  createList();
  Model m=null;
  boolean b=false;
  for(int i=0;i<l.size();i++)
  {
  m=(Model)l.get(i);
  if(m.getEmpno().equalsIgnoreCase(empno))
  {
     b=true;
     break;
  }
  }
  if(b==true)
     return "Empno is " +m.getEmpno()+ "   Name is  "+m.getName()+" Address is
"+m.getAddress()+" Gender is     "+m.getGender()+ " Attendence is  ”+m.isAtt();
  else
     return "Record Not Found !";
```



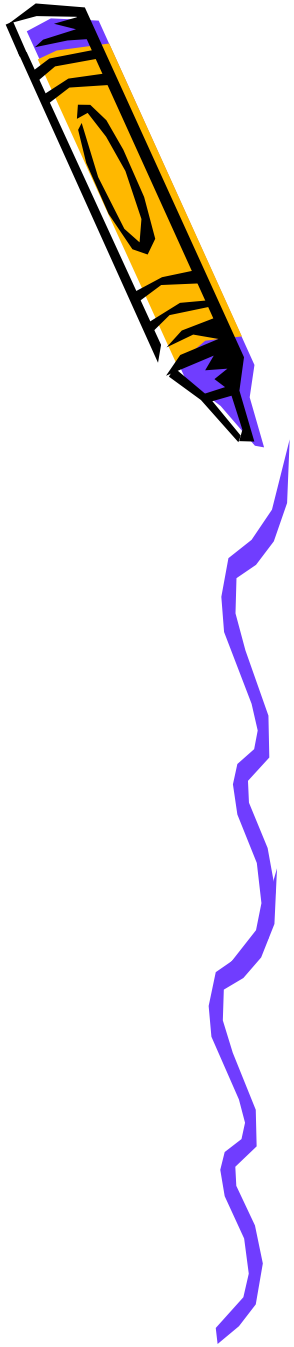
Result.jsp

```
<%@page contentType="text/html"%>

<html>

    <body>
        <br>
        <hr>
        <br>
        <b><%=request.getAttribute("result").toString()%></b>

    </body>
</html>
```



MVC exercise

Create a customer login page , user have deposit and withdraw option.
The existing user detail maintain inside a database.
User cannot withdraw more than it limit so also check it balance before withdraw.
Also user have logout button,to logout .
Maintain it session in session API



EL

Expression Language

EL- Expression Language

EL was added to the JSP 2.0

EL expression always in the curly braces, and prefixed with the dollar sign.

Example : ${person.name}

Syntax: {firstThing.secondThing}

EL Implicit Objects

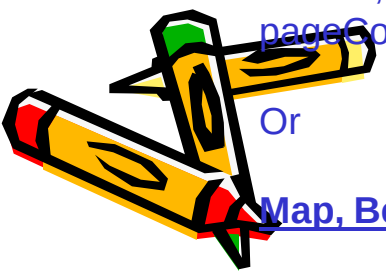
pageScope, requestScope, sessionScope, applicationScope, param, paramValues, header, headerValues, cookie, initParam, pageContext

Or

Map, Bean, List

Property , Index, Key

getName(), setName(), [0], “user” ← Key



EL Examples

${person.name} or ${person[“name”]

String [] names= {“Amit”, “Ram”, “Shyam”};

Request.setAttribute(“nameList”,names);

${nameList} // Will print entire name list

${nameList[0]}   or ${nameList[“0”]}

EL Examples

Map phones = new HashMap();

phones.put(“amit”,”1111”);

phones.put(“ram”,”2222”);

phones.put(“shyam”,”3333”);

${phones[“amit”]} // Can get Sub keys also

Or

${phones.amit} // this will give only value no sub key

EL Implicit Objects Examples

Implicit Object	Example	Remark
param	${param.name}	request.getParameter("name");
paramValues	${paramValues.hobbies[0]}	request.getParameterValues("hobbies)[0];
header	${header["host"]	request.getHeader("host");
request	${request.method} // through this object you can get the request properties	request.getMethod();
requestScope	${requestScope.phones[0]} // Note requestScope is used for set and get attribute it is not an request object	request.getAttribute("phones") ;
cookie	${cookie.userName.valu}	cookie[i].getValue();
initParam	${initParam.mainEmail}	config.getInitParameter("main Email");
sessionScope	${sessionScope.phones[0]);	session.getAttribute("phones") ;

<scripting-invalid> Tag

Example:

```
<jsp-config>
<jsp-property-group>
<url-pattern>*.jsp</url-pattern>
<scripting-invalid>true</scripting-invalid>
</jsp-property-group>
</jsp-config>
```

Or

<%@ page isScriptingEnabled="false" %> // Now in JSP 2.0 Onwards this attribute is no longer valid

<el-ignored> Tag

Example:

```
<jsp-config>
<jsp-property-group>
<url-pattern>*.jsp</url-pattern>
<el-ignored>true</el-ignored>
</jsp-property-group>
</jsp-config>
```

Or

```
<%@ page isELIgnored="true" %>
```

Note: The page directive takes priority over the DD setting

JSTL

JSTL (JSP Standard Tag Library)

<c:forEach>

Using JSTL Loop

```
<br>
<c:forEach var="a" items="${requestScope.nameList}“ varStatus="loopcount" >
${a}      Loop Count Value is ${loopcount.count}
</c:forEach>
```

JSTL (JSP Standard Tag Library)

<c:if>

Using JSTL If

Using JSTL If

```
<c:if test="${requestScope.nameList[0] == 'Amit'}">
Welcome ${requestScope.nameList[0]}
</c:if>
```

Using JSTL choose

Using JSTL Choose

<c:choose>
<c:when test="${requestScope.nameList[0] == 'Amit'}">
Welcome ${requestScope.nameList[0]}
</c:when>

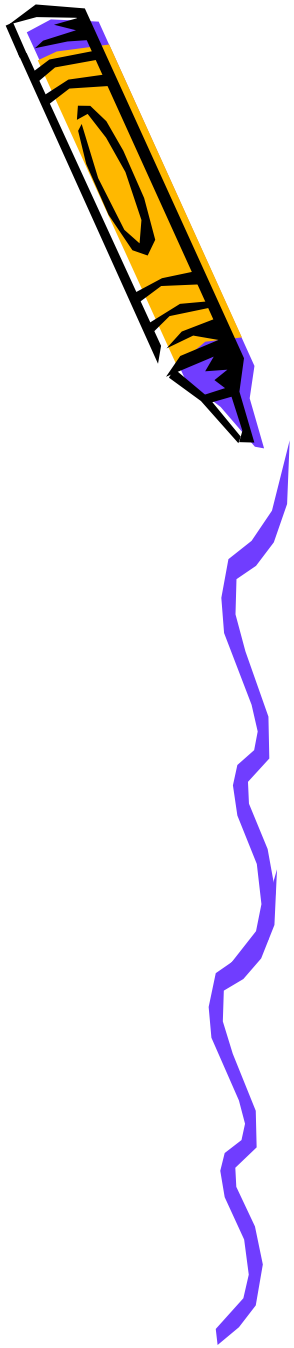
<c:when test="${requestScope.nameList[0] == 'Ram'}">
Welcome ${requestScope.nameList[0]}
</c:when>
<c:when test="${requestScope.nameList[0] == 'Shyam'}">
Welcome ${requestScope.nameList[0]}
</c:when>
<c:otherwise>
Wrong Choice
</c:otherwise>
</c:choose>

JSTL (JSP Standard Tag Library)

<c:set>

Using Set

```
Using set
 <br>
 <c:set var="user" scope="session" value="Amit" />
 <c:set var="user" scope="session" value="${requestScope.nameList[0]}" />
 <c:set var="user" scope="session">
 Amit, Ram, Shyam
 </c:set>
 <c:remove var="user" scope="session" />
```



JSTL (JSP Standard Tag Library)

<c:import>

Using import

```
<c:import url="http://localhost:8988/web/index.jsp" />
  <c:import url="http://localhost:8988/web/index.jsp" >
  <c:param name="a" value="Amit" />
  </c:import>
  <c:url value="/index.jsp" />
```

```
<error-page>
<error-code>404</error-code>
  <location>/result.jsp</location>
  </error-page>
```

```
<error-page>
   <exception-type>java.lang.ArithmeticException</exception-type>
   <location>/result.jsp</location>
   </error-page>
```

```
<%@ page isErrorPage="true" %>
<%@ page errorPage="/index.jsp" %>
```

```
<welcome-file-list>
   <welcome-file>/index.jsp</welcome-file>
   </welcome-file-list>
```

```
<servlet>
<servlet-name>index_jsp</servlet-name>
<jsp-file>/index.jsp</jsp-file>
<init-param>
<param-name>email</param-name>
<param-value>amit@yahoo.com</param-value>
</init-param>
```

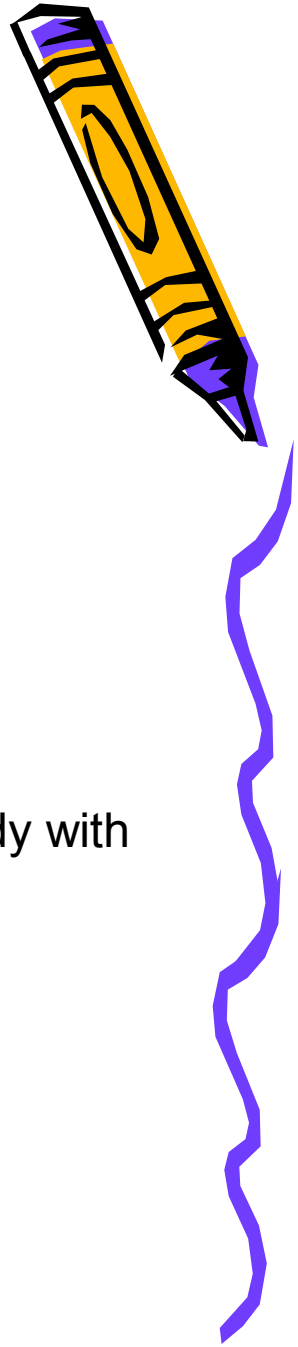
Custom Tags

JSP 1.1 support creation of custom tags that enables segregation of business logic from the content presentation.

The Structure of a custom tag in JSP, like those in XML and HTML, contain the start/end tag and a body.

Advantages of Using Custom tags

- They can reduce or eliminate scriptlets in your JSP application.
- They are reusable.
- They can improve the productivity of non-programmer , content-developers , by allowing them to perform tasks that cannot be done with HTML.



Types of Custom Tag

1)
 No Body , No Attribute

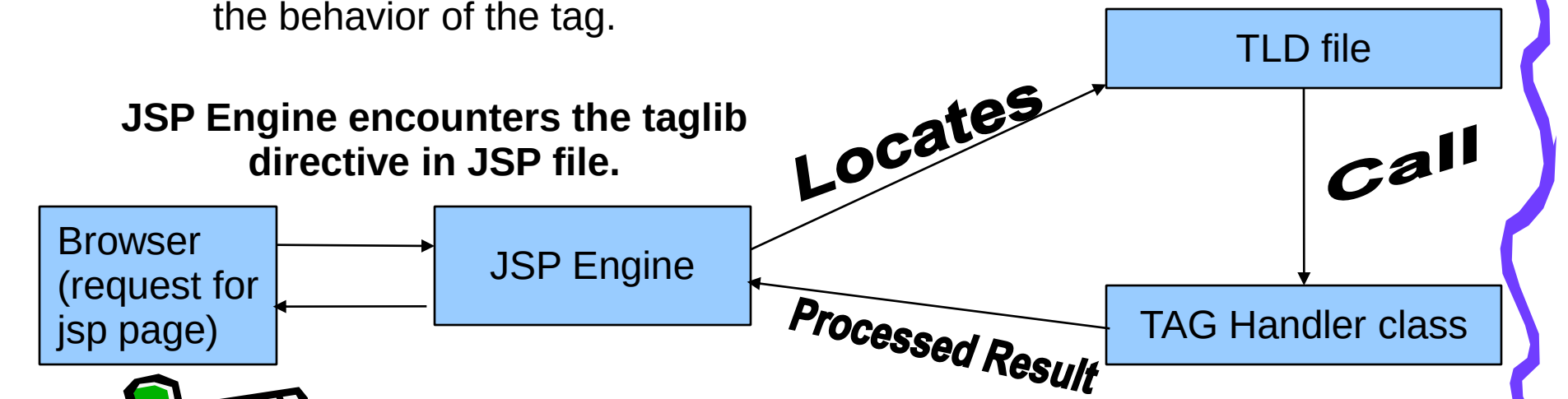
2) <hr size=”3” > No Body with Attribute

3) THIS IS TEXT   With body No Attribute

4) <Font face=”Arial” size=”10”> THIS IS TEXT With body with Attribute.

Components of a Tag Library

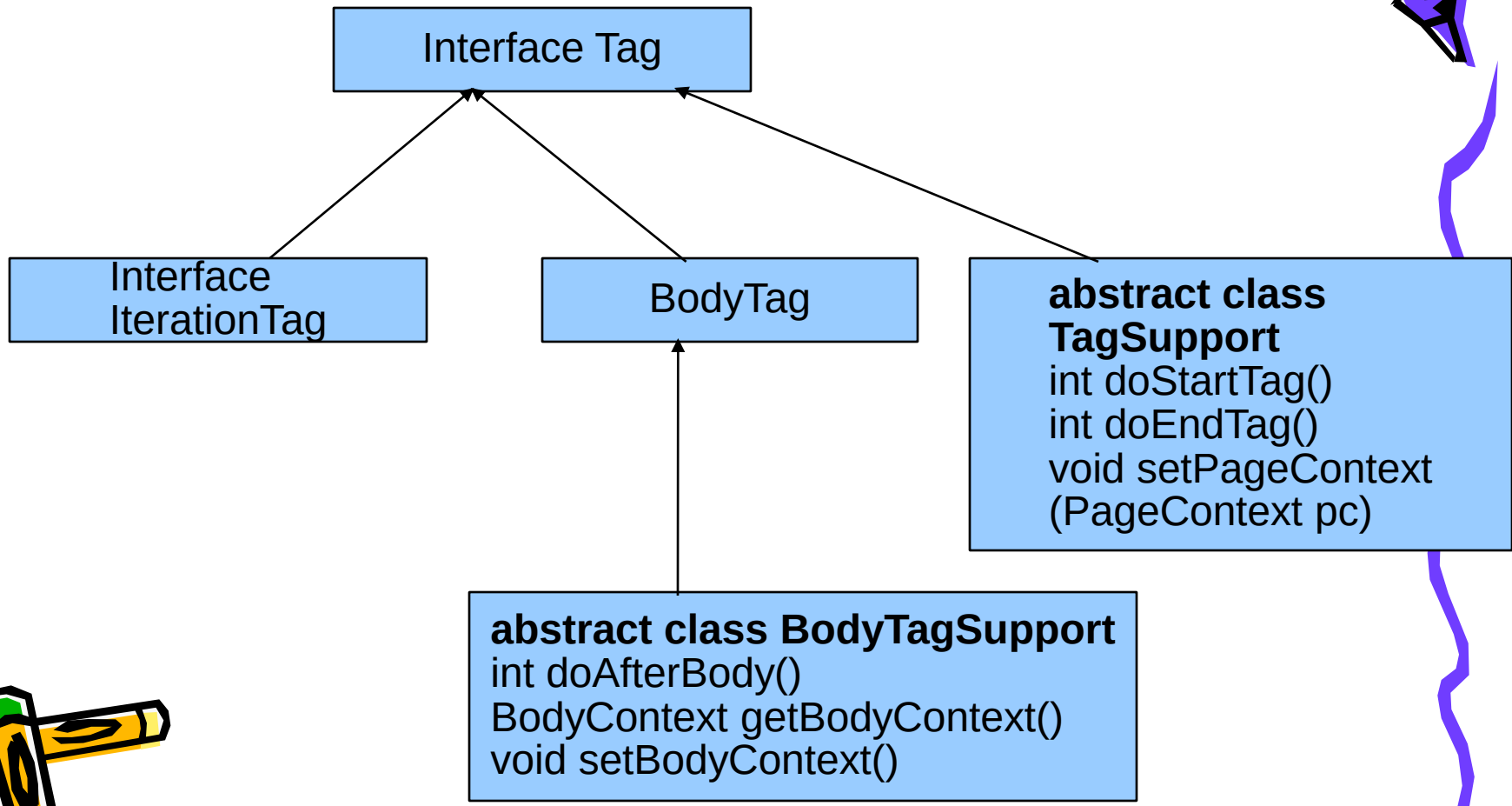
1) JSP File – Contains the Custom tag to add.
2) TLD (Tag Library Descriptor) – It is an XML file that contain a descriptive list of the custom tags.
3) Tag Handler class file- Use different methods and objects to define the behavior of the tag.



The Execution cycle of a JSP file with custom tags

Custom Tag API

javax.servlet.jsp.tagext.*



Custom Tag API Continued.....

Static Constants

1) final static int SKIP_BODY – Skip body evaluation.

2) final static int EVAL_BODY_TAG – request the creation of new BodyContent on which to evaluate the body of this tag.

3) final static int EVAL_BODY_INCLUDE- evaluate body into existing out stream.

4)final static int EVAL_PAGE- continue evaluating the page.

5) final static int SKIP_PAGE- Skip the rest of the page.

Steps to create a Custom Tag

Step-1 Develop a Tag Handler class

```
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.*;
public class mytag extends TagSupport
{
public int doStartTag() throws JspException
{
try
{
JspWriter out=pageContext().getOut();
out.println("Welcome to my website ");
out.close();
}
catch(Exception ee)
{
ee.printStackTrace();
}
return SKIP_BODY;
}
```

To develop a empty or body less tag use TagSupport class

This method is called when the container encounters the start tag
of a custom tag.

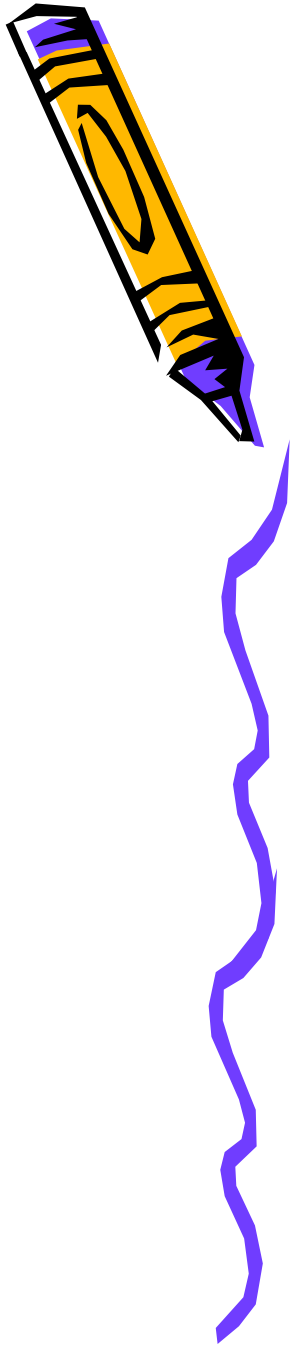
It means no need to evaluate body because body is empty.

Code Continued.....

```
public int doEndTag() throws JspException
{
return SKIP_PAGE;  //EVAL_PAGE
}
}
```

This method is invoke when the container encounters the end tag of a custom tag.

It indicates that the container should not process the remaining content of the JSP page.



Step-2 Develop the TLD file mytld.tld

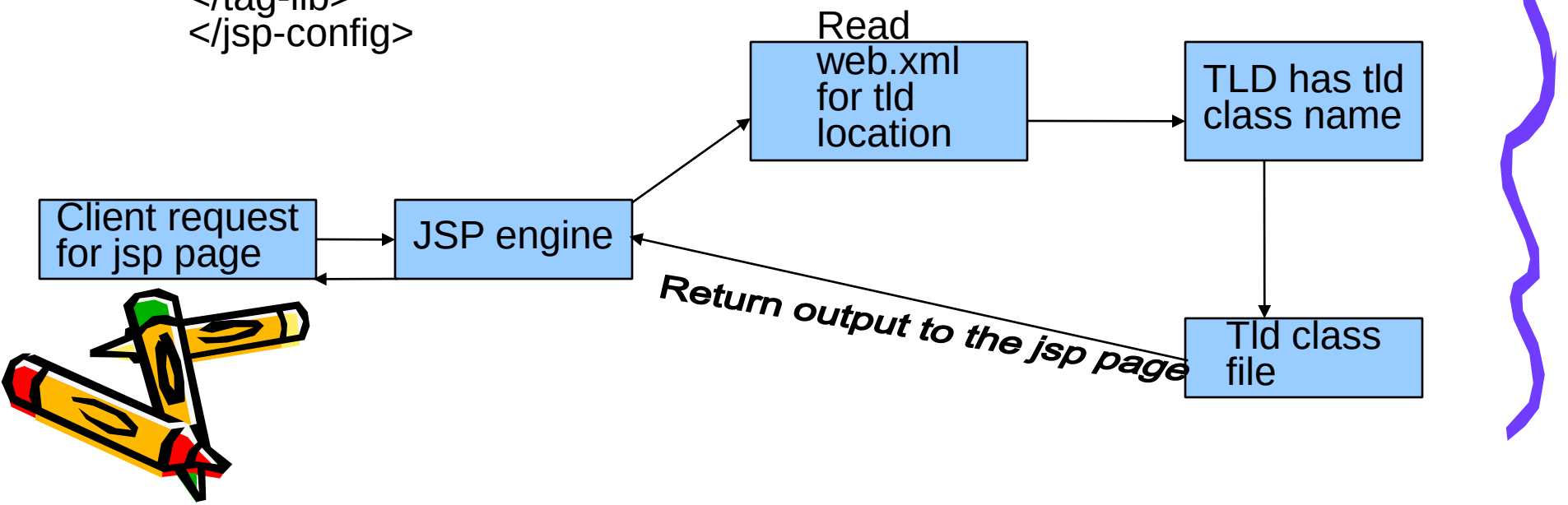
```
<tag-lib>
<tlib-version>1.0</tlib-version>
<jsp-version>1.2</jsp-version>
<short-name>A</short-name>
<description></description>
<tag>
        <name>mytag</name>
        <tag-class>mytag</tag-class>
        <body-content>Empty</body-content>
</tag>
</tag-lib>
```

Step-3 Include the tag library in jsp page

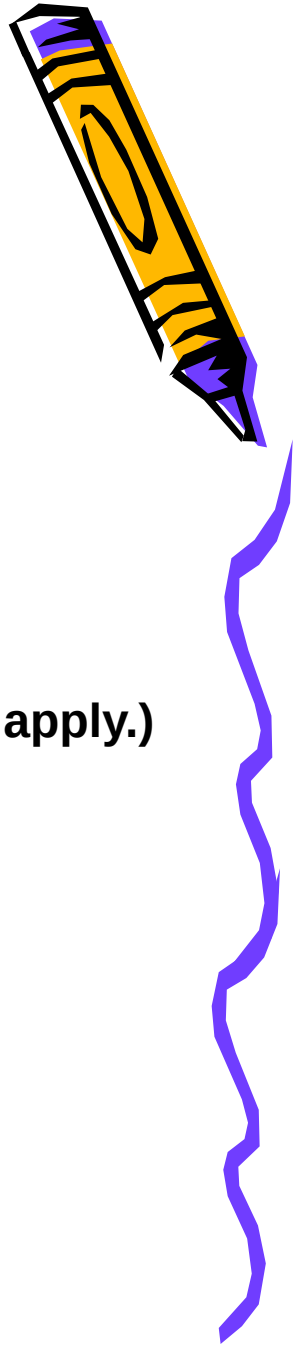
```
<%@ taglib uri=”/mytld.tld” prefix=”print” %>
<print:mytag />
```

Step-4 Change in web.xml

```
<jsp-config>
<tag-lib>
<taglib-uri>/mytld.tld</taglib-uri>
<taglib-location>/WEB-INF/mytld.tld</taglib-location>
</tag-lib>
</jsp-config>
```



TEST YOUR SELF



1) Which HTTP method is used by default , if not method is specified

a) GET
b) POST
c) OPTION
d) TRACE

2) Which are valid JSP implicit variables ? (Choose all that apply.)

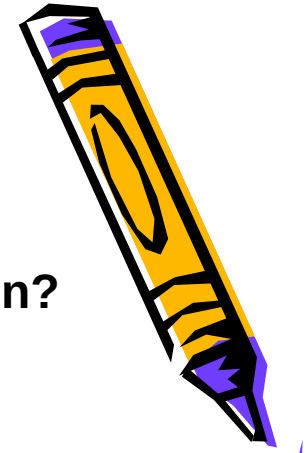
a) stream
b) context
c) exception
d) application

Q3. Which are valid JSP implicit variables ? (Choose all that apply.)

a) stream
b) context
c) exception
d) application

Q4. Which is an example of the syntax used to import a class in JSP?

a) <%page import =”java.util.Date “ %>
b) <%@ page import =”java.util.Date @%>
c) <%@ page import =”java.utl.Date” %>
d) <% import java.util.Date; %>



Q5. Which are true about the <jsp:useBean> standard action? (Choose all that apply.)

a) The id attribute is optional

b) The scope attribute is required

c) Either the class or type attributes may be specified, but at least one.

d) It is valid to include both the class attribute and the type attribute , even if their values are NOT the same.

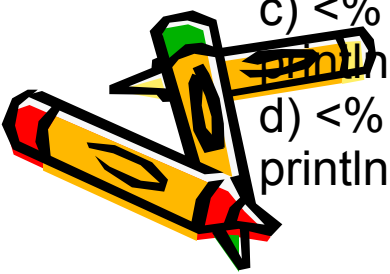
Q6. Given a request with two parameters: one named “first” represents a user's first name and another named “last” represents his last name.

a) <% out.println(request.getParameter(“first”);
out.println(request.getParameter(“second”); %>

b) <% out.println(application.getInitParameter(“first”);
out.println(application.getInitParameter(“second”) %>

c) <% println(request.getParameter(“first”);
println(request.getParameter(“second”); %>

d) <% println(application.getInitParameter(“first”);
println(application.getInitParameter(“second”); %>



Q7.Which tag in the jsp is used to define the error page.

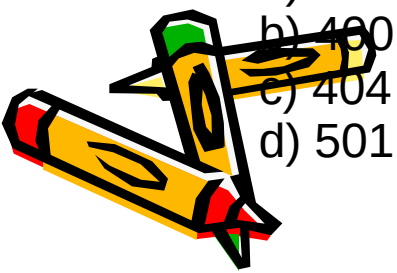
a) location
b) exception – type
c) error-page
d) content type

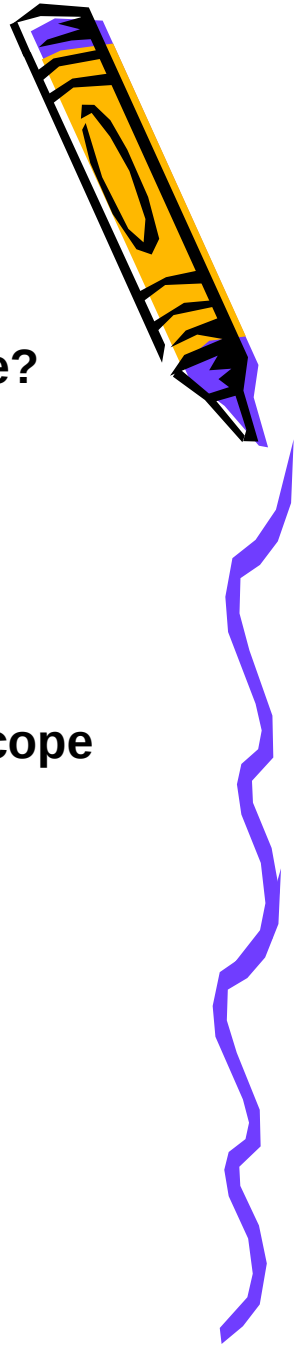
Q8. Implementing the tag <%@ page isThreadSafe=”false” %> implements the SingleThreadModel interface making the JSP.

a) content-safe
b) thread-safe
c) secure-safe
d) None of these

Q9.The error message displayed when the page is not found at the correct location.

a) 550
b) 400
c) 404
d) 501





Q10. What is the role played by JSP in the MVC architecture?

a) Model
b) View
c) Controller
d) None of the Above

Q11. Which of the following statements is true about the scope of the “application object “ in JSP

a) It is global and accessible with in the same web context.
b) It is global and can accessible outside the context.
c) It's scope one page to another page.
d) none of the above