Hi Friends,

Just go through the following small story.. (taken from ***"You Can Win "*** - by Shiv Khera)

Once upon a time a very strong woodcutter asks for a job in a timber merchant, and he got it. The paid was really good and so were the work conditions. For that reason, the woodcutter was determined to do his best. His boss gave him an axe and showed him the area where he was supposed to work. The first day, the woodcutter brought 18 trees "Congratulations," the boss said. "Go on that way!" Very motivated for the boss' words, the woodcutter try harder the next day, but he only could bring 15 trees. The third day he try even harder, but he only could bring 10 trees. Day after day he was bringing less and less trees. The woodcutter thought that "I must be losing my strength". He went to the boss and apologized, saying that he could not understand what was going on.

The boss asked, "When was the last time you sharpened your axe?"

"Sharpen? I had no time to sharpen my axe. I have been very busy trying to cut trees.

If we are just busy in applying for jobs & work, when we will sharpen our skills to chase the job selection process?

My aim is to provide good and quality content to readers to understand easily. All contents are written and practically tested by me before publishing. If you have any query or questions regarding any article feel free to leave a comment or you can get in touch with me on venus.kumaar@gmail.com.

This is just a start, I have achieved till now is just 0.000001n% .

With Warm Regards

Venu Kumaar.S
venus.kumaar@gmail.com

Jsp: Web application Resource programs are 2 types 1. server side programs 2. client side programs

→ serverside programs resides and executes from the server.
        Ex: servlet prog, jsp prog , ASP, asp.net, PHP, coldFusion etc.
→ client side programs resides in web server but they comes to browser window for execution.
        Ex: html program, javaScrit program, VBScript program , Ajax etc.
→ Server side programs generate dynamic web pages
→ client side programs generate static web pages.
→ every web application is a collection of both client side and server side web resource programs.
→ while working with asp, php technologies we can go for tags based programming. to work with servlets strong java knowledge is required and tags based programming is not possible so no programmer had liked servlets in its days. To attract normal-java programmers like asp programmers the sun micro system has given a tag based server side technology called "JSP" having all the features of servlets.
→ **Drawbacks of servlet programming:**
1. strong java knowledge is required so not suitable for non-java programmers.
2. The presentation logic (html code) will be mixed up with java code(pw.println()) so modifications done in one code may effect another code.
3. writing HTML code in servlet programming is a complex process.
4. In servlet programming implicit objects are there but we need to write some additional code to get access to those objects.
5. modifications done in source code of servlet program will be effected only after compilation of servlet program and reloading of web application in most of the servers.
6. programmer should explicitly take care of exception handling.
7. the session object required for session tracking should be created by the programmer manually.
8. servlet programming is not thread safe by default so we need to write additional code to make our servlet program as thread safe.

→ **Features of JSP:**
1. allows tag based programming so strong java knowledge is not required.
2. suitable for both java and non-java programmers.
3. use 9 implicit objects and we can use them directly in our jsp program.
4.modifications done in jsp program will be recognized by underplaying server automatically without reloading of web application.
5. takes care of exception handling.
6. allows use to separate presentation logic from(html code) java code (business logic).
7. easy to learn and utilize.
8. allows us to work with jsp supplied built-in tags and third party supplied jsp tags and even allows to develop custom jsp tags.
10. use all the features of servlet.

→ jsp, asp asp.net, php programs are there to execute based on npage compilation process.
→ the process of converting one form of program source code to another form of source code is called as page compilation.
→ In jsp page compilation the source code of jsp program will be converted into an equivalent servlet program source code. for this jsp page compiler is required.
→ every web server/app server supplies one jsp page compiler.

→ servlet container executes servlet program and takes care of the life cycle of servlet programs.
→ jsp container executes jsp programs by executing the jsp equivalent servlet programs and also takes care of the life cycle of jsp programs.
→ every jsp container is enhancement of servlet container.
→ every web server and application server supplied servlet container and jsp container.
→ the servlet container is developed based on server-api specification. In tomcat server servlet container name is "catalina.jar".
→ jsp container will be developed based on servlet and jsp api specification. in tomcat server jsp container name is jasper(tomcat_home\linb\jasper.jar).

→ the 9 implicit object of jsp programming:
out
config → it is ServletConfig
application → it is ServletContext obj
request
response
page
pageContext
exception
session

→ once jsp equivalent servlet is generated all these objects will be created in that servlet automatically so we can call them as implicit objects or built-in objects of jsp.

→ **The difference between html and jsp:**

→ **HTML**	**JSP**
→ client side technology	→ serverside tech
→ needs html interpreter for execution	→ needs jsp page compiler and jsp container for execution
→ html program generates static web pages web	→ jsp programs can generate static & dynamic pages.
→ given by w3c	→ given by sun micro systems
→ can be used in any kind of web appl dev applications.	→ should be used only in java based web
→ not allows to create custom tags	→ allows to create custom tags.

Java Server Pages(JSP) is a J2EE component.

- JSP is Server side program which is similar to Java servlet in functionality and Design.
- JSP provide same features found in a Java Servlet because a JSP is internally converted into Servlet when Client requests the JSP.
- A Java Server Page is nothing but a servlet that is automatically generated by a JSP container

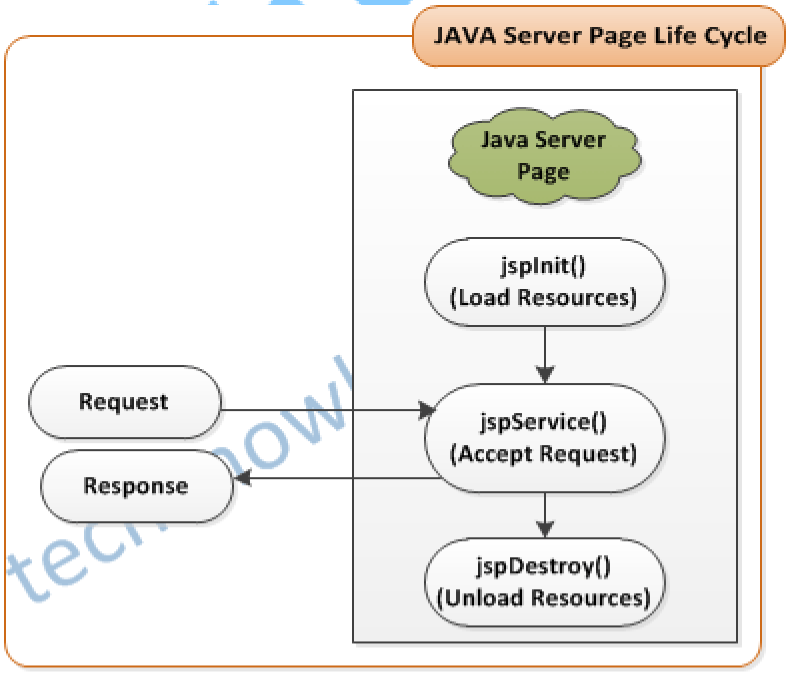
from a file containing valid HTML and JSP tags.

- The JSP specification defines JSP as "a technology for creating the applications for generate dynamic Web content such as HTML, DHTML, SHTML, and XML."
- A Real Web application can generate a lot of Web pages and each of them may need to be modified or maintenance.
- JSP provide you to separate design from business logic, so a Web designer can take care of the HTML part while the Java developer concentrates on the programming of business functions required by the application.

JSP Life Cycle

There are three methods in JSP:-

1. jspInt()
2. jspService()
3. jspDestroy()



jspInt():-

4. Calls the jspInit() to initialize te servlet instance.
5. It is called only once for a servlet instance.
6. It is called before any other method by Container.

jspService():-

7. jspService() method in the new servlet automatically generate by container.
8. Which have the same functionality and arguments as the servlet's service() method.

jspDestroy():-

9. It automatically called when JSP terminates normally.
10. It used for cleanup and disconnect from th database where resources used during the execution of JSP.

JSP container will generate, compile, and deploy a servlet automatically that produces the output based on the content of the file.

JSP vs Servlet:-

JSP	Servlet
JSP is better to generate pages that consist of large sections of fairly well structured HTML or other character data.	Servlets are good for generate binary data, building pages with highly variable structure, and performing tasks (such as redirection).
JSP uses java code in Html.	Servlet uses Html in Java code.
JSP is easy to maintenance or modified.	Servlet is not easy to maintenance or modify.
JSP is slow compare to Servlet.	Servlet is fast compare to JSP.
JSP uses both the method(doPost and doGet)in service().	Servlet uses only one method(doPost or doGet) in service().
JSP uses for both design and functionality.	Servlet uses only for functionality.

page 397

→ <%---%> is called as scriplet.
→ Template text = html code + ordinary code(text)
→ jsp program can generate dynamic web page. the fixed content of this dynamic web page will be generated through template text of jsp program. similarly the dynamic values/ content of that dynamic web page will be generated through java code of scriptlet placed in jsp program.

Ex: <b><i> Date and time:</b></i>
<% java.ujtil.Date d=new java.util.Date();
out.println(d.toString());%>

step1: create deployment directory structure by taking the jsp program as web source program.
step2: deploy the above jsp program in tomcat server (copy this to <tomcat-home>\webapps folder)
step3: start the tomcat server

step4: test the web application. give request to abc.jsp
→ we can hide the jsp program file name & extension from end users to hide technologies details from end users by providing url pattern to jspm program in web.xml file.
→ we can provide URL-pattern to jsp program by containing jsp program in web.xml

```
        <web-app>
        <servlet>
                <servlet-name> xyz</servlet>
                <jsp-file> /abc.jsp</>
        </s>
<servlet-mapping>
<s-n>xyz</s-n>
<url-pattern>/test1</up>
</s-m>
</wa>
```

→ WEB-INF is a private directory of java web application so only web server can recognize that directory.
→ the web resource programs that are placed inside WEB-INF or its sub folders will be recognized by the webserver only when they are configured in web.xml file.
Ex: servlet programs

→ the web resource programs that resides outside the WEB-INF folder can be recognized by the webserver directly so there is no need of configuring those programs in web.xml file **Ex:** html, jsp programs

→ **the jsp program can have two types of objects:**
- → **implicit objects:** the 9 implicit objects
- → **explicit objects:** objects created by programmer manually.
  <% java.util.Date d=new java.util.Date()%> --> d is explicit obj

→ this, super are implicit objs of standalone java programs
→ request, response, ServletConfig & ServletContext objects are implicit objects of servlet programming and we need to write code to set access to those objects.

→ Jsp execution process: two phases: 1) translation phase 2. Request processing phase.

→ in translation phase the jsp program will be converted into an equivalent servlet program.
→ in request processing phase the compile code of jsp equivalent servlet executes and the generated response goes to browser window.
fig page 400

→ in this phase the _jspService(-,-) of jsp equivalent servlet executes and generates response goes to browser window as web page.

→ the life cycle of servlet program:
init(ServletConfig cg)
service(ServletRequest req, ServletResponse res)
destroy()

→ The life cycle of jsp program:

→ _jspInit→ latest version jspInit→ old version
→ _jspService(-,-)
→ _jspDestroy() → latest version jspDestroy → old version

jspInit() --> execute when jsp equivalent Servlet class is instantiated
_jspService(-,-) → executes when jsp program gets req from clients.
jspDestroy() → executes when the ServletContainer is about to destroy object of jsp equivalent servletClass.

in tomcat server abc.jsp belongs to JSPAPP web app comes in <tomcat-home>\work\catalina\localhost\JspApp\org\Apache\Jsp folder as ABC_jsp.java, ABC_JSP.class

→ the generated jsp equivalent servlet is HttpServlet.

→ to deploy above abc.jsp in web appl in advjavabatcahdomain of weblogic
copy JSPAPP folder to <weblogic-home>\middleware\user_projects\domains\advjavaBA€tchdomain\autodeploy floder.

→ to generates req to abc.jsp of above JSPAPP web app user the following url:
http://localhost:7001/JSPAPP/ABC.jsp

Jsp tags:
built-in tags/ elements of jsp programming:
1) scripting elements: allows to place java code.
1.1) scriptlet <%....%>
1.2) declaration<%!..%>
1.3) expression<%=...%>
2) directive tags: provides global information to jsp page.
2.1) page directive <%@page attributes %>
2.2) include directive<%@include attributes %>
2.3) taglib directive<%@ taglib attribute %>
jsp comments: <%-- .... --%>

3) standard action tags: these are given to perform dynamic operations in jsp programming at request processing phase.

<jsp:forward>, <jsp:include>,<jsp:useBean>, <jsp:setProperty>, <jsp:getProperty>, <jsp:param>, <jsp:plugin>, <jsp:callback>

→ **scriptlets:** two types of syn 1) standard <%.......%>
2) xml syntax <jsp:scriptlet>.............</jsp:scriptlet>
→ the java code placed in this goes to _jspService(-,-) method of jsp equivalent servlet program as it is.
→ programmers can place BL or requesting processing logic code in the scriptlets of jsp program.
→ implicit objects of jsp are visible in the scriptlet. In one jsp program we can keep multiple scriptlets.
→ variable declared in jsp comes as local variables of _jspService(-,-) method in jsp equivalent servlet so they must be initialized.
Ex: With standard syntax

```
<% int a=10;
int b=20;
int c=a+b;
out.println("Result:"+c);
%>
```

Ex: with the support of xml standard

```
<jsp:scriptlet>
java.util.,Date d = new java.util.Date();
out.println(d.toString());
</jsp:scriptlet>
```

Ex:

```
<jsp:scriptlet>
 int a=10;
int b=20;
if (a<b>)
        out.println("a is small");
else
        out.println("b is small");

</jsp:scriptlet>
```

→ the above code execution gives exception because in (a<b) statement "<" symbol is taken as beginning of sub-tag having xml or jsp meaning. That means it will not be taken as the conditional operator of java code.

Solution1

Ex: with standard syntax

```
<% int a=10;
int b=20;
if (a<b>)
        out.println("a is small");
else
        out.println("b is small");

%>
```

Solution2

xml syntax with CDATA

```
<jsp:scriptlet> int a=10;
<![CDATA[ int b=20;
if (a<b>)
        out.println("a is small");
else
         out.println("a is b");
]]>
</jsp:scriptlet>
```

→ this CDATA makes page compiler to take the body of the tag as text information(java code) and suppress the meaning of xml and jsp for that code.

→ While working with xml syntax of declaration tag take care of "<" symbol problem with the support of "CDATA".

→ The scriptlet not allowed to declare methods. the following code is invalid

```
 <% public int sum(int x, int y) {return x+y;}%>
```

→ but we can place method calls code in scriptlets.

```
<% String s="Hello";
int len=s.length();
out.println("length="+len);
%>
```

→ **Declaration**

```
<%! declaration of instance var;
definitions of user defined methods;
definitions of jspInit() & jspDestroy() life cycle methods
%>

<jsp:declaration>

</jsp:declaration>
```

→ the java code placed in declaration tag comes outside the _jspService(-,-) of jsp equivalent. so implicit objects are not visible in this declaration tag.

→ variables declared in declaration tag will come as the instance variables of the jsp equivalent servlet class.

→ use "this" or implicit object "page" to differentiate declaration tag variable from the scriptlet tag variable.

```
<%! int a=10; %>
<% int a=20;
out.println("local var scriptlet a valueis :"+a);
out.println("instance vardeclarative  a valueis :"+this.a);
%>
```

Ex: keeping user defined java method definitions:

```
<%! public int sum(int x, int y) {return x+y;
} %>
<% out.println("Result="+sum(10,20));%>
```

→ we can also use declaration tag to place jspInit(), jspDestroy() life cycle method definitions in out jsp program.

Ex: <%!public void jspInit()

```
{
System.out.println("JspInit() method can contain initialization");
}%>
```

Ex:
```
<%!public void jspDestroy()
{
System.out.println("JspDestroy() method can contain Un-initialization");
}%>
```

→ we Can't declare _jspService(-,-) in declaration tag of jsp progs. by default the jsp equalent servlet class does not provide this life cycle method. Jsp equalent servlet automatically gives this method, so our method becomes duplicate method to it and java does not support to have duplicate methods in a java class.

```
<%! public void _jspService(HttpServletREquest req, HttpServletResponse res)
{
}
%>
```

Expression tag:

```
<%=java expr%>
<jsp:expression>
java expr
</jsp:expr>
```

→evaluates given expression and writes generated result to browser window.
→ code placed in this tag goes to _jspService(-,-) of jsp equivalent servlet so implicit objects are visible in expression tag.
→ expressions are nothing but arithmetic operations, logical operations, and method calls.

```
<%
int a=10;
int b=20;
out.println(a+b);
%>
```

the above code is equals to the following code:

```
<%
int a=10;
int b=20;
%>
```
<%=a+b> → evaluates a+b expression and writes the result to browser window.

→ we can also use the expression tag to print the variable values:

```
<%
int a=10;
int b=20;
%>
A value is<%=a%>
B value is<%=b%>
Result is <%=a+b%>
```

→ we can use expression tag to call java methods both predefined and user defined.

```
<%! public int sum(int x, int y) { return x+y;} %>
The result is : <%=sum(10,20)%><br>

<%String s="Hello!jsp;%>
The subString of <%=s%> is: <%=s.substring(0,5)%>
```

→ If expression tag is used to instantiate a java class then the default data of the object will be written to browser as response.

Ex: Date and Time is: <%=new java.util.Date() %>

Ex: xml based expression tag:

```
<jsp:scriptlet>
int a=10;
int b=20;
</jsp:scriptlet>
Result is <jsp:expression>a+b</jsp:expression>
```

→ by using xml syntax of expression tag we can not pass expressions that are using "<" syntax, even-though CDATA is used.

```
<jsp:scriptlet>
int a=10;
int b=20;
</jsp:scriptlet>
Result is <jsp:expression>
                    <![CDATA[a<b]]>
          </jsp:expression>
```

the above code is invalid.

→ the above problem can be solved with standard syntax of expression tag.

```
<jsp:scriptlet>
int a=10;
int b=20;
</jsp:scriptlet>
Result is <%=a<b%>
```

→ it is always recommended to use scripting elements of jsp by following standard syntax.

→ The jsp expression tag can evaluate only one expression at a time. It can not be used to evaluate multiple expressions as shown below

```
<%
int a=10;
int b=20;
%>
Results are: <%=a+b, a-b, a*b%>
```

→ we cannot write one scripting tag of jsp inside another scripting tag that means we must use these tags as independent tags:

```
<%
int a=10;
```

int b=20;
Result is <%=a+b→ invalid statement
%>

→ **implicit objects: 09**
1)**request:** javax.servlet.HttpServletRequest → request scope
2)**response:** javax.servlet.HttpServletResponse → response scope
3)**out :** javax.servlet.jsp.JspWriter(Abstract Class) → Page scope
4)**session:** javax.servlet.HttpSession (Interface) → session scope
5)**page :** this (currently invoking jsp equalent servlet class object reference) → page scope
6)**pageContext:** javax.servlet.jsp.PageContext(class) → page scope
7)**application:** javax.servlet.ServletContext(interface) → web application scope
8)**config:** javax.servlet.ServletConfig(interface) → page scope
9)**exception:** → java.lang.Throwable(class) → page scope

→ all the above given 09 implicit objects of jsp are the objects of servlet container and jsp container supplied java classes implementing the above given interfaces or extending the classes and interfaces of servlet, jsp **api**s given by sun micro systems.
→ JSP programmers never worries and remembers the class names of these implicit objects because they will be changed based on the server we utilizes.
Ex: the implicit object "out" is not the object of javax.servlet.jsp.JspWriter class. It is the object of underlying JSPContainer supplied java class that extends from javax.servlet.jsp.JspWriter class.
In tomcat server that class name is "org.apache.jasper.runtime.JspWriterImpl".
In tomcat server that class name is "weblogic.servlet.jsp.JspWriterImpl".
In JSP program we can use 03 types of comments to comment the code.
page: 415

→ **Page Directive tag:** This tag is given to provide global information to jsp program like buffer size, package import statements, content type and etc.
Syntax:
standard: <%@page attributes%>
xml: <jsp:directive.page attributes/>
attributes:
→ language="java" → dafault → java only one language that you can pass here as a value.
It specifies the language taht we need to use to write code in scripting tags.
→ import="java.util.*I, java.net.*,..." → default is java.lang.*;
usefull to import the java packages that are required for java application jsp program.
→ extends="class name" → not recommended to use, no default value.
==> content type="text/html" ==> no default values. this is useful to specidy content type of the response that should be generated by jsp program.

Ex: <%@page language="java" import="java.net.*, java.sql.*" contentType="text/html"%>

→buffer="18kb" or more → default is 8Kb → specifies the size of the buffer at server side.
→ Even though the code of jsp program is executing part by part, because of underlying cpu algorithms (like round robin algorithm) the server preserves output of the executed statements in buffer of the jsp program until last statement of jsp program gets executed.

<%@page buffer="10kb"%>

<%@page buffer="none"%>

→ autoflush="true"- default → true → It cleans tghe buffer of jsp program and sends to browser window as web page.

<%@ page buffer="8kb" autoflush="true"%>

→session="true" default→ enables jsp equalent servlet create or not to create implicit object session.
<%@page session="false"%>
<%@page session="true"%>
→ when we don't enable session taracking we don't need session object in jsp program.

→ errorPage="url" //no default
→ isErrorPage="false" true → default → false
==> errorPage & isErrorPage are required to configure error pages for jsp program to handle the exceptions raised in jsp program.

→ isThreadSafe="true" default = true → useful to enable or disable thread safty on jsp equalent servlet.
Ex: <%@page isThreadSafe="false"> enables the threadsafe. make jsp equalent servlet as Thread safe servlet by implementing javax.servlet.SingleThreadModel(inteface)

→ info="String→ the short description about jsp page . no default value.
<%@ page info=basic program"%>
→ isELIgnored="true" default is false. → This can make page compiler to ignore or recognize EL.

Rules for page directive usage:
1) page directive tag name nad attribute names are case sensitive.
2) unknown attributes will not be recognized.
3) except import attribute no other attribute can have multiple values seraped with ",".
4) Except import attribute no other attribute can not occur for multiple times with differnt values either in the same page directive or in the multiple page directive tags of same jsp program.

→ The jsp program dynamically expands its size as need.
→ even though buffer=none is taken the jsp program is capable of expanding buffer size as needed at run-time so we can enable autoflush="true" even-though buffer is none.

→ **The difference between PrintWriter() and JspWriter():**
→ the process of result/output in buffer before delivering to the original destination is technically called as buffering operation.
→ PrintWriter can not perform buffering while sending its output to browser window where as JspWriter can perform buffering.
→ When jsp program is taken with out buffer then JspWriter internally uses PrintWriter to write the output to browser window.
→ JspWriter is the type of the implicit object called "out" in jsp program.
→ PrintWriter is useful in servlet program.
→ PrintWriter belongs to java.io package and JspWriter belongs to javax.servlet.jsp package.

→ **servlet API is given as**
javax.servlet pacakge, javax.servlet.http package.
latest version : 3.0 supports annotations based programming.
running version:2.4/2.5

→ **JSP api is given as:**
javax.serverlet.jsp package
javax.serverlet.jsp.el package
javax.serverlet.jsp.tagext package

latest version: 2.1
running version: 2.0

→ Jsp equivalent servlet automatically handles the exceptions, but displays the exceptions related error messages quite ugly on browser window when exception is raised. Since these are technical messages the non-technical end users can not understand. To overcome this problem configure errorpages for jsp program.
→ Error page is a jsp or html program that are capable of executing only when exception is raised in other jsp programs. These are also useful to display the exception related messages non technical messages guiding the end users.
→ Always develop web applications by keeping non-technical end users in mind. for that error pages configuration is important.

→ **Error page configuration in Jsp program**
1) **Local configuration:** → This error page configuration(cfg) is specific to one jsp program.
→ use errorPage, IsErrorPage attributes of <%@page..%>

→ **isErrorPage="true"** attribute of page directive tag can make a jsp program of web appl. as error page.
→ the implicit object exception is visible only in that Jsp page that acts as 'error page" and we can use this object to know the details of exceptions that are raised in main jsp programs:
→ in servlet program we take support of rd.forward(-,-) for error servlet/error page configuration.

→ **Test.jsp**
```
<%@page errorPage="err.jsp"%>
<b> from test.jsp</b><br>
<% int x=Integer.parseInt("@123");
out.println("x value is :"+x);%>
```

→ **Err.jsp**
```
<%@page isErrorPage="true"%>
<b>from err.jsp</b><br>
<font color="red">Internal problem</font><br>
The Exception that is raised is : <%=exception.toString()%>
```

2) **Global Configuration**→ This error page cfg is common for all the jsp programs of a web appl.

→ use <error-page> tag of web.xml file.

→ the error page cfg done in web.xml file for jsp programs will not work for servlet programs of web appl.

→ this cfg also allows to take .html program as error page.

→ **Test.jsp**
```
<%@page errorPage="err.jsp"%>
<b> from test.jsp</b><br>
<% int x=Integer.parseInt("@123");
out.println("x value is :"+x);%>
```

Err.jsp
```
<%@page isErrorPage="true"%>
<b>from err.jsp</b><br>
<font color="red">Internal problem</font><br>
The Exception that is raised is : <%=exception.toString()%>
```

web.xml
```
<web-app>
<error-page>
<exception-type>java.lang.Exception</exception-type>
<location>/err.jsp(/err.html)</location>
</error-page>
</web-app>
```

→ **Note:** Because of the above configuration done in web.xml for any exception raised in any jsp program of web application the control goes to error.jsp page.
→ global error page configuration also allows to take .html program as error page.

****→ if both error cfgs are done in the same page to handle same exception pointing to two different error pages, in this scenario the cfgs done in local error page configuration will be effected.

→ **directive include tag:**
standard syntax: <% @include file="destutl"%>

xml syntax: <jsp:directive.include file="dest url"/>

→ This tag is given to include the code or content of destination web resource program to the source code of jsp equivalent servlet belonging to the source jsp program.
→ Here B.jsp will not be executed separately but its code will be included to the code of A.jsp and executes along with A.jsp.
→ directive include(<%@include ...%>) does not perform output inclusion, but it performs code/content inclusion.
Ex:
A.jsp

```
        <b> from A.jsp</b><br>
        <%@ include file="B.jsp"%>
        <b> from end of A.jsp</b><br>
```
B.jsp
```
        <b> from B.jsp</b><br>
        <%=new java.util.Date()%>
```
web.xml
```
        <web-app/>
```
Request URL to test the application:
http://localhost:8080/jspApp5/A.jsp

→ **when the above A.jsp is requested**

→a) jsp page compiler generates jsp equivalent servlet for A.jsp

→b) Because of <%@include file="B.jsp"%> the code of B.jsp will be included to A.jsp related jsp equalent servlet program

→c) The java compiler compiles jsp equivalent servlet source code of A.jsp and executes that code then sends response to browser window.

→ From the above execution B.jsp program will not be executed separately, so its jsp equivalent servlet will not be generated separately.

→ Action include(<jsp:include>)

syntax: <jsp:inlcude page="dest url"/>

***→Note All standard action tags of jso like <jsp:include> are having only xml syntax and there is no standard syntax for them.

→ <jsp:include> is given to include the output of destination web resource program to the output of the source jsp program.

→ Here destination program(B.jsp) executes independently so its output will be included to the output of source program(A.jsp) dynamically. The response that comes to browser window contains the output of source(A.jsp) and destination(B.jsp) programs.

Ex:

A.jsp
```
        <b> from A.jsp</b><br>
        <jsp:include page="B.jsp"%>
        <b> from end of A.jsp</b><br>
```
B.jsp
```
        <b> from B.jsp</b><br>
        <%=new java.util.Date()%>
```
web.xml
```
        <web-app/>
```
Request URL to test the application:
http://localhost:8080/jspApp6/A.jsp

→ **<jsp:include>** performs output inclusion but not the code inclusion.

→ <jsp:include> is tag based alternate for "rd.include(-,-)" method of servlet programming.

→ When the above A.jsp is requested two separate jsp equivalent servlets will be generated for both A.jsp, B.jsp and the jsp equivalent servlet of A.jsp internally uses some rd.include(-,-) method to include the output of B.jsp.

→ the action tags of jsp can be used only with xml syntax compare to other tags of jsp(other tags can be having both xml and standard syntax).
→ standard action tag of jsp are given to perform more dynamic operations as runtime/request processing phase operation.
→The difference between "include directive" and "action include" tags:

Include Directive <%@inclscriptlets: DeclarationExpression tag:implicit objects: 09Page Directive tag: Rules for page directive usage:Error pagedirective include tag:ude..%>
→ given to perform code inclusion.
→ performs code inclusion at translation phase, so this work is called as compile time binding or static binding.
→ If destination program is JSP the separate jsp equivalent servlet will not be generated for that program.
→ can not include code and output of servlet program.
→ suitable to take static web resource programs as destination programs(like html programs).
→ does not use rd.include(-,-) internally
<%@include file="B.jsp"%>

Action Directive <jsp:include>
→ Given to perform output inclusion
→ performs output inclusion at run time/request processing phase so this work is called as dynamic/run-time binding.
→ will be generated.
→ can include(includes the output)
→ suitable to take dynamic web resource programs as destination program(like servlet program jsp program)
→ uses rd.include(-,-) internally.
<jsp:include page="B.jsp"/>
→ example application on jsp program including the output of servlet program:
→ request url → http://localhost:8080/jspapp7/Test.jsp

DateSrv.java

```
improt javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class DateSrv extends HttpServlet
{
	public void service(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
	{
	//general settings
	PrintWriter pw = res.getWriter();
	res.setContentType("text/html");
	//write BL
	java.util.Date d = new java.util.Date();
	pw.println(d.toString());
	} //service(-,-)
}//class
```

web.xml

configure DateSrv servlet program with test1 url pattern.

Test.jsp

```
<b> from begining of Test.jsp</b><br>
Date and Time is <jsp:include page="test1"/><br> → we can't replace this tag

  with indclude directive tag
<br>
<i>end of Test.jsp</i>
```

→ while working with directive include and action include don't commit the response in destination web resource program by using out.close() or pw.close() methods because it does not allow to add further output of source jsp program to the response.

→ Taking a request from a browser window and process the request by using multiple jsp programs as chaining. for this we can use <jsp:include> tag or <jsp:forward> tag.

→ all the jsp programs which participate in a jsp chaining will use same request and response objects.

→ Java program participating in jsp chaining, all the jsp programs of jsp chaining will use same request and response objects.

→ if the content to include is static content use directive include tag. If the content to include is dynamic content use action include tag.

→ in one jsp program we can see the utilization of both directive include, action include tags for multiple times.

Example appl based on the above:

header.jsp

```
<%@page import="java.util.*"%>
<% Calendar cl=Clendar.getInstance();
int date = cl.get(Calendar.DAY_OF_MONTH);
int month = cl.get(Calendar.MONTH);%>
<%=date%>/<%=month> &nbsp;&nbsp;&nbsp;&nbsp<%=cl.getTime()%>
<br><br>
<marquee><font color=red size=5>G U R U</font></marquee>
<be>
<br>
```

Footer.html

```
<hr>
<br><br><center>
&copy;copy rithts reserved 2012</center>
```

Main.jsp

```
<jsp:include page="header.jsp"/>
<br><br>
Sports news of the day <%=new java.util.Date()%>;
<font size=2> some text</font><br><br><br>
<%@include file="footer.html"%>
```

web.xml

```
<web-app/>
request URL to test the application ==> http://localhost:8080/jspApp8/main.jsp
```

→ <jsp:forward>

syntax: <jsp:forward page="dsturl"/>

→ given to perform forwarding request mode of jsp chaining. It internally uses rd.forward(-,-) method.

diagram in page no 432.

→ The html output of source jsp program (A.jsp) will be discarded and only the html output of destination program(B.jsp) goes to browser window as response through source program (A.jsp). Here two separate and independent equivalent servlet programs will be generated for A.jsp, B.jsp programs.

→ **Example application:**

A.jsp

```
<b>Start of A.jsp</b><br>
<jsp:forward page="B.jsp"/>
<b>End of A.jsp</b>
```

B.jsp

```
<br><%=new java.util.Date()%><br>
<i>From B.jsp</i>
```

→ when source jsp program is interacting with destination jsp program either by using <jsp:forward> or <jsp:include> then source jsp program can send additional request parameter values to destination jsp program by using <jsp:param>tag

<jsp:param> This tag is allowed to use only as sub tag of <jsp:include>, <jsp:foward> tags.

Eample program:

A.jsp

```
<b>Start of A.jsp</b><br>
<jsp:forward page="B.jsp"/>
    <jsp:param name="bookname" value="CRJ"/>
    <jsp:param name="price" value="300"/> → additional req pram name, values.
</jsp:foward>
<b>End of A.jsp</b>
```

B.jsp

```
<br><i>From B.jsp</i><br>
<br>Additonal request parameters given by A.jsp are
  <%=request.getParameter("bookname")%><br>
  <%=request.getParameter("price")%><br> → logic to read additional req
                                           param values.
<i>From B.jsp</i>
```

Example application on <jsp:forward>, <jsp:param> tags:

form.html

```
<form action="First.jsp" method="get">
Item name:<input type="text" name="tname"/><br>
Item Price:<input type="text" name="tprice"/><br>
Item Quantity:<input type="text" name="tqty"/><br>
<input type="submit" value="submit"/>
</form>
```

First.jsp

```
<% //read form data
String iname=request.getParameter("tname");
int price=Integer.parseInt(request.getParameter("tprice"));
int qty=Integer.parseInt(request.getParameter("tqty"));
//calculate bill amout
int bamt=price*qty;
//give 20% discount
float discount=bamt*0.2f;
if(discount>=1000)
{
out.println("Item name:"+iname);
out.println("Bill Amount:"+bamt);
out.println("Discount:"+dicount);
}
else
{
//forward the request to Second.jsp
<jsp:forward page="second.jsp">
        <jsp:param name="p1" value="<%=iname%>"/>
        <jsp:param name="p2" value="<%=bamt%>"/>
</jsp:forward>
}
%>
```

from the above→ sending iname, bamt details to second.jsp from first.jsp as additional request parameter values.

Second.jsp

```
<b> From second.jsp</b><br>
<% //read additional request parameters send by First.jsp
String name=request.getParameter("p1");
int bamt=Integer.parseInt(request.getParameter("p2"));
//give 30% discount
float discount=bamt*0.3f;
//display item details
out.println("Item name:<br>"+name);
out.println("Bill Amount:<br>"+bamt);
out.println("Discount:<br>"+dicount);%>
```

Request URL → http://localhost:8080/jspApp10/Form.html

→ There are directive include, action include tags. why there is no directive forward tag?(only action forward tag is available in jsp)

→ the basic work of forwarding request operation is discarding the out of source jsp program.

The directive tags perform their activity at translation phase by including the code of destination program to the code of jsp equivalent servlet generated for source jsp program.

In this process discarding the output of source jsp program is impossible. So there is no directive forward tag in jsp.

→ Difference between <jsp:forword> tag and response.sendRedirect(-) method:

<jsp:forword>

→source jsp program communicates with destination program directly.
→ both source, destination programs use same request and response objects so request data coming to source jsp program can be used in destination program
→ Both source and destination programs must be there in same web application.
→ A destination program must be a html program or jsp program or servlet program.
→ while forwarding request url in the address bar will not be changed.

response.sendRedirect(-)
→ source jsp program communicates with destination program by having one network round trip with browser window.
→ source jsp program and destination program will not use same request and response objects so the request data coming to source destination program is not visible and accessible in destination program.
→ both source and destination programs can be there in same web application or can be there in two different web applications of same server or different servers.
→Destination program can be html program or java based web resource or non-java based web resource program.
→ While redirecting the request the URL in the address bar will be changed into destination program related URL.

→ <jsp:forward>, <jsp:include> tags are the tag based alternates for rd.forward(-,-) methods and there is no jsp tag providing the functionality of response.sendRedirect(-) method. that means java code should be written directly in jsp program.

→ Difference between the implicit objects page and pageContext:
→The implicit object page holds "this" keyword is nothing but reference of currently invoking jsp equivalent servlet class object.
→ pageContext object holds multiple details of jsp page like this, request, response objects, errorPage name, buffersize, autoflush mode, session objects availability status.
→Jsp equivalent servlet creates/gets access to application, config, session, and out implicit object through this pageContext object.
→code in jsp equivalent servlet program
→**for implicit object page:** Object page=this;
→**for implicit object pageContext:**
pageContext=_jspxFactory.getPageContext(this, rquest, response, "Abc.jsp", true, 10240, true);
Abc.jsp → error page
true → availability of status of session object
10240 → buffer size
true→ autoflush mode

→To pass data between the JSP program of web application:
→ 1) if source and destination programs are using same req, res objects
a) use request attributes
b) use additional request parameters(<jsp:param>)
→ 2) If source and destination jsp programs are getting request from same browser window and not using same req and res objects ==> use session attributes.
→3) If source and destination programs are not getting request from same browser

window and not using same request and response objects:==> use application attributes.

→4) use pageContext attributes: we can create pageContext attribute having page Scope or request Scope or session Scope or application Scope.

diagram

→ In the above diagram the request attribute created in A.jsp is visible and accessible in B.jsp, C.jsp programs because A.jsp, B.jsp and c.jsp programs use same request response objects.

→ Request attributes are visible through out request cycle.

→ The session attribute created in A.jsp by getting request from browser window b1 is visible and accessible in all jsp programs of web application irrespective their request, response objects but all these web resource programs must get request from same browser window b1.

→ **Note:** session attributes are visible in all web resource programs of web application but they are specific to a browser window.

→ Application attribute created in A.jsp is visible and accessible in all other web resource programs of web application irrespective of the browser window from which they are getting request and irrespective of request and response objects they are using.

→ attributes are logical names holding values the servlet programming, jsp programing uses attributes ato pass data from one web resource program to another web resource program.

→ setAttribute(-,-) → to create new attribute or to modify existing attribute values.

→ getAttribute(-,-) → to read existing attribute values.

program:

A.jsp

```
<% //create attributes
        request.setAttribute("attr1","val1");
        request.setAttribute("attr2","val2");
        application.setAttribute("attr3","valu3");
        <!-- forward the request to B.jsp -->
                <jsp:forward page="B.jsp"/>
```

B.jsp

```
<!-- read attribute values --> <b> from B.jsp</b><br>
attr1(req) attribute value is=<%=request.getAttribute("attr1")%><br>
attr1(session) attribute value is=<%=session.getAttribute("attr2")%><br>
attr1(application) attribute value is=<%=application.getAttribute("attr3")%><br>
<!-- forward the request to C.jsp-->
<jsp:forward page="C.jsp"/>
```

C.jsp

```
<!-- read attribute values --> <b> from C.jsp</b><br>
attr1(req) attribute value is=<%=request.getAttribute("attr1")%><br>
attr1(session) attribute value is=<%=session.getAttribute("attr2")%><br>
attr1(application) attribute value is=<%=application.getAttribute("attr3")%><br>
```

D.jsp

```
<!-- read attribute values --> <b> from D.jsp</b><br>
attr1(req) attribute value is=<%=request.getAttribute("attr1")%><br>
attr1(session) attribute value is=<%=session.getAttribute("attr2")%><br>
attr1(application) attribute value is=<%=application.getAttribute("attr3")%><br>
```

→ **URL to test the application:**

→ give first request to A.jsp and give other request to other jsp programs either from same browser window or from different browser windows.
→ Request object can create only request scope attributes.
→ Session object can create only session scope attributes.
→ Application object can create only application scope attributes where as pageContext object can create page scope or request scope or session scope or application scope attributes.

→ To create pageContext attribute:

pageContext.setAttribute("attr1","val1"); → create pageContext attribute having page scope

pageContext.setAttribute("attr2","val2",pageContext.SESSION_SCOPE); → create pageContext attribute having session scope.

→Other possible scopes:

pageContext.PAGE_SCOPE
pageContext.SESSION_SCOPE
pageContext.APPLICATION_SCOPE
pageContext.REQUEST_SCOPE

All these scopes constants of javax.servet.sp.PageContext class.

→To modify pageContext attribute values:

pageContext.setAttribute("attr1","val11");==> modifies attr1 value of page scope.

pageContext.setAttribute("attr2","val22",pageContext.SESSION_SCOPE);==> modifies attr2 attribute value of session scope.

→ To read pageContext attribute values:

String s1=(String)pageContext.getAttribute("attr1"); → reads attr1 attribute value from page scope.

String s2=(String)pageContext.getAttribute("attr2",pageContext.SESSION_SCOPE); → reads attr2 attribute value from session scope.

→ To remove pageContext attributes:

pageContext.removeAttribute("attr1");→ remove attr1 attribute from page

pageContext.removeAttribute("attr2",pageContext.SESSION_SCOPE); → removes attr2 attribute from session scope

→ To find attribute:

String s1=(String)pageContext.findAttribute("attr2"); → searches and reads attr2 attribute in multiple scopes in the following order...

a) page Scope b) request scope c) application scope d) session scope

→ Difference between getAttribute() and findAttribute() invoked on the **pageContext object:**

getAttribute() searches the given attribute only in the specified scope(if no scope specified, in pageScope). Whereas findAttribute() searches the given attribute if the multiple scopes in a particular order like page scope, request scope, session scope and application scope.

Program with the support of pageContext attribute:

A.jsp

```
<% //create attributes
        pageContext.setAttribute("attr1","val1",pageContext.REQUEST_SCOPE);
        pageContext.setAttribute("attr2","val2",pageContext.SESSION_SCOPE);

pageContext.setAttribute("attr3","valu3",pageContext.APPLICATION_SCOPE);
```

```
			<!-- forward the request to B.jsp -->
				<jsp:forward page="B.jsp"/>
```
B.jsp
```
		<!-- read attribute values -->
		<b> from B.jsp</b><br>
		attr1(req) attribute value is =
<%=pageContext.getAttribute("attr1",pageContext.REQUEST_SCOPE)%><br>
		attr1(session) attribute value is=<%=pageContext.findAttribute("attr2")%><br>
		attr1(application) attribute value
is=<%=pageContext.findtAttribute("attr3")%><br>
		<!-- forward the request to C.jsp-->
		<jsp:forward page="C.jsp"/>
```
C.jsp
```
		<!-- read attribute values -->
		<b> from C.jsp</b><br>
		attr1(req) attribute value is =
<%=pageContext.getAttribute("attr1",pageContext.REQUEST_SCOPE)%><br>
		attr1(session) attribute value is=<%=pageContext.findAttribute("attr2")%><br>
		attr1(application) attribute value
is=<%=pageContext.findtAttribute("attr3")%><br>
```
D.jsp
```
		<!-- read attribute values --> <b> from D.jsp</b><br>
		attr1(req) attribute value is =
<%=pageContext.getAttribute("attr1",pageContext.REQUEST_SCOPE)%><br>
		attr1(session) attribute value is=<%=pageContext.findAttribute("attr2")%><br>
		attr1(application) attribute value
is=<%=pageContext.findtAttribute("attr3")%><br>
```
→ **<jsp:plugin>** tag is given to display applets on the browser window. This tag internally uses <applet> tag or <object> tag of html to display applets.

→ **<jsp:callback>** tag should be used as sub tag of <jap:plugin> tag. This tag exevutes only when underlaying browser window does not support applets,

→ The tomcat web server internally uses <embed>,<noembed> tags for <jsp:plugin>,<jsp:fallback> tags.

Example:

→ request url to test the application:
http://localhost:8080/jsp_plug/plug.jsp

TestApp.java
```
import java.applet.Applet;
import java.awt.*;
public class TestApp extends Applet
{
public void paint(Graphics g)
{
setBackground(Color.red);
g.drawString("This is from applet to test",50,50);
}
}
```
plug.jsp
```
<html>
```

```
                              <body bgcolor="pink">
                              <jsp:plugin type="applet" code="TestApp.class" codebase="/jsp_plug" 
width="300" height="300">
                              <jsp:fallback> browser does not support applets</jsp:fallback>
                              </jsp:plugin>
                              </body>
                    </html>
```

→ /jsp_plug → in this specify the directory where given applet class(TestApp.class) is available

→ **web.xml**

<web-app/>

→ **JavaBean:**

→ A java bean is a java class that contains getter and setter methods. there is no special keyword to develop the java class as java bean.

→ The member variables of java bean class are technically called as bean properties and every bean property contains one getter and one setter method.

→Getter methods are useful to get or read the data from bean properties. Where as setter methods are useful to write the data to bean properties.

Ex:

```
public class stuBean
{
//bean properties
private int sno;
private String name;
getters() and setters()
}
```

→ if java programmers follow java bean standards while developing their java classes then it becomes easy for the programmers to exchange the java classes.

→ java bean is a standard to develope java classes having getXxx() and setXxx(-) methods.

→ the best way to develop re-usable simple java class is developing that class as java bean class.

→ **The difference between java bean and EJB:**

→**Java bean**

→ an ordinary java class coantaining getter and setter methods
→ just needs JVM for execution
→ no life cycle methods
→ userful as a reusable helper class in projects.
→ allows only local clients.

→ **EJB**

→ Distributed technology develop distributed applications.
→ needs JVM and EJB container for execution.
→ life cycle methods are there
→ useful as technology to develop business components of the project.
→ allows both local and remote clients.

→ In one computer multiple JVMs can be there to run multiple java applications simultaneously or parallelly. If application and its client resides in the same JVM then that client is called local client tp application. If application and its client resides on two different JVMs of same computer or different computers then that client is called remote client to application.

→ A java bean class can have 3 types of bean properties:

1) Simple bean properties
2) Boolean bean properties
3) Indexed bean properties

1) **simple bean properties** → allows to store only one value at a time.

Ex: int sno;

```
public void setSno(int sno){this.sno=sno;}
public int getSno(){return sno;}
```

2) **boolean bean properties** → allows to store only true or false value.

Ex: boolean married;

```
public void setMarried(boolean married)
{this.married=married;}
public boolean getMarried()/isMarried() { return married;}
```

3) **Indexed bean properties** → allows multiple values to store. This bean property type is array.

Ex: String[] colors;

```
public void setColors(String[] colors){this.colors=colors;}
public String[] getColors(){return colors;}
```

→ for servlet program to java-bean communication the servlet program should be an object of java-bean class and should call methods on that object.

→ For jsp program to java bean communication we can use the following tags:

<jsp:useBean>
<jsp:setProperty>
<jsp:getProperty>

→ **java based web application development models**

1) **model1 architecture** → use only servlets or only jsp programs as server side web resource programs of web application.

2) **model2 architecture(MVC architecture):**

2.1) MVC1 architecture
2.2) MVC2 architecture

→ In MVC1 and MVC2 architectures multiple technologies support will be taken to develop web applications.

→ Logics that can be there in our java web applications:

1) request data gathering logic
2) form validation logic
3) business logic
4) session management logic
5) persistence logic
6) middleware services logic
7) presentation logic and etc..

→ **Model-1 architecture:** In this model either servlet programs or jsp programs will be used as server side web resource programs. moreover if servlet programs are used jsp programs will not be used and vice verse.

→ In every server side program multiple logics will be mixed up

Advanages:

→ knowledge on either servlet or jsp is sufficient to develop web resource programs.

→ since parallel development is not possible multiple programmers are not required to develop the web applications.

Disadvantages:

→ multiple logics will be mixed up in every server side web resource program. This indicates there is no clean separation of logics and this also indicates modification done in one logic may disturb other logic.

→ parallel development is not possible so the productivity is very poor.

→ middleware services(some) must be implemented by the programmer manually. this will make burden on the programmer.

→ **MVC-2 Architecture:**

→ **Model** → represents BL+Persistence logic

→ **View** → represents presentation logic

→ **Controller** → represents Integration logic/connectivity logic

→ use servlet or servlet filter program to develop this

→ Integration logic is responsible to take the request from browser window, to pass that request to model layer resource, to gather result from model layer resource and to pass the result to view layer resource.

→ integration logic is responsible to control and monitor every operation of the web applications execution.

→ in model-2 MVC architecture based web application development multiple technologies support will be taken by programmers.

→ in **MVC1** architecture the same servlet/jsp programs will act as view layer, controller layer resources representing presentation, business logics and separate resources will taken for model layer.

→ in **mvc2** 3 different resources will be taken in 3 different layers:

JSP programs → in view layer
Servlet program → in controller layer
Java class/javaBean/ejb component/ → in model layer

MVC-2 Advantages:

1) there are multiple layers in web application development so we can achieve clean separation of logics.
2) modifications done in logics of one layer does not effect logics of another layer.
3) parallel development is possible so productivity will be good.
4) provides easiness in enhancement and maintenance of the web application.
5) MVC-2 is a industry standard to develop the web applications.
6) when spring, hibernate, EJB technologies are used in the model layer we can get the benefit from middleware services(built-in). this reduce burden on the programmer.
7) MVC-2 is industry standard to develop the web applications.

Disadvantages:

1) for parallel development multiple programmers are required.
2) knowledge in multiple technologies is required to develop the web applications.

→ while developing MVC2 architecture based web applications along with taking support from multiple technologies we should also implement set of rules which are called MVC-2 rules or principles.

1) Every layer is given to have certain logics. Just place only those logics and don't place any excess logics.
2)It is recommended to use jsps in view layer, servlets in controller layer. use java bean or java class in model layer when business logic is less.
3) use EJB or spring with hibernate in model layer when business logic is huge.
4) every operation in the web application must take place under the controller of controller servlet.
5) There can be multiple resources in the view layer and multiple resources in model layer but there must be only one controller sevlet or controller server filter program in controller layer.
6) The view layer resources must not talk with model layer resources directly and vice verse. That means they must communicate with each other through controller servlet.
7) Two resources of view layer(jsp programs) must not talk with each other directly. That mean they must communicate with each other through controller layer.

→ **<jsp:useBean>** :
syn: <jsp:useBean attributes/> →used for creating/locating an object of java bean class
attributes:
id="" //instance name(Object name)
class="" //a fully qualified java bean class name
scope="page/session/request/application"(default scope is "page")
scope specifies the scope of the bean class object
page(default): bean class obj is available throughout the page.
session: bean class obj is available throughout the session.
request: bean class obj is available throughout the request.
application: available for all pages within the context(web application)
beanName="logical name of java bean"
type="reference class type of java bean class object"

Ex: <jsp:useBean id="st" class="p1.stuBean" scope="session"/>
creates p1.studentBean class obj having name st and keeps in session scope.
→ if this object is already available in session scope then it locates that object from session scope. For this it internally uses pageContext.setAttribute(-,-), pageContext.getAttribute(-) methods.

→ **The above statement creates object like this:**
p1.StuBean st=new p1.StuBean();
"st" object type is p1.stuBean
"st" reference type is p1.stuBean

Ex: <jsp:useBean id="st" type="ABC" class="p1.StuBean" scope="session">
creates or locates p1.StuBean class object(st) in/from request scope.
Bean class object creation statement:
ABC st=nre p1.StuBean();
"st" ==> reference type is ABC class
"st" ==> object type is p1.StuBean class
==> Here p1.StuBEan class extends from ABC class
→ <jsp:useBean>, <jsp:setProperty>,<jsp:getProperty> are 3 independent tags.

<jsp:setProperty> → calls setXxx() on java class object to set given values to bean properties.

syn: <jsp:setProperty attributes/>

attributes:

property=""//bean property name
name=""// bean class obj name id attribute value given in te <jsp:useBean>
value=""//value to be set for bean property
param=""//input(request) parameter name

Ex: <jsp:setProperty name="st" property="sno" value="567"/>

This internally calls setSno() method on st obj and assigns value 567 to the bean property "sno".

<jsp:getProperty> → calls getXxx() methods to read values from bean properties.

syn: <jsp:getProperty attributes/>

attributes:

name=" " //(bean obj name) id attribute value given in <jsp:useBean>
property: "bean property name"

Ex: <jsp:getPropertu name="st" property="sno"/>

gives "sno" bean property value by calling getSno() method on bean class object("st") and also displays that value on browser window.

→ **example application on jsp to jave bean communication(mvc1)**

StudentBean.java

```
package p1;
public class StudentBean
{
 //bean properties
 private int sno;
 private String name;
 //constructor
 public StudentBean()
 {
 syso("StudentBean 0-param constuctor");
 }
 public int getSno()
 {
 return sno;
 }
 public void setSno(int sno)
 {
  syso("StudentBean:setSno(-) method...")
  this.sno=sno;
 }
 public String getName()
 {
 return name;
 }
 public void setName(String name)
 {
  syso("StudentBean:setName(-) method...")
  this.name = name;
 }
}
```

→ **javac -d StudentBean.java**

SetValues.jsp

```
<%@page import="p1.StudentBean"%>
<!-- create or locates java bean class obj -->
<jsp:useBean id="st" class="p1.StudentBean" scope="session"/>
<!-- set values to bean properties -->
<jsp:setProperty name="st" property="sno" value="101"/>
<jsp:setProperty name="st" property="name" value="GURU"/><br>
values are set to bean properties.
```

GetValues.jsp

```
        <%@page import="p1.StudentBean"%>
        <!-- create or locates java bean class obj -->
        <jsp:useBean id="st" class="p1.StudentBean" scope="session"/>
        <!-- get values from bean properties -->
        Sno value: <jsp:getProperty name="st" property="sno"/><br>
        Name Value:<jsp:getProperty name="st" property="name"/><br>
        values are set to bean properties.
```

web.xml

```
<web-app/>
```

→ irrespective of whether Thread safety is enabled or not enabled (isThreadSafe="true/false") the jsp equalet servlet manages session scoped, application scoped javabean class objects having thread safety with the support of synchronized blocks.

URL to test the application

http://localhost:8080/JspApp/SetVAlues.jsp
http://localhost:8080/JspApp/GetVAlues.jsp

→ we can use the param attribute of <jsp:setProperty> tag to set the received request parameter value as bean property value.

```
<jsp:setProperty name="st" property="sno" param="stno" value="101"/>
<jsp:setProperty name="st" property="name" param="stname" value="GURU"/>
```

→ if request parameter names are matching with bean property names then we can use property="*" in <jsp:setProperty> tag to set request parameter values as bean property values.

SetValues.jsp

```
<%@page import="p1.StudentBean"%>
<!-- create or locates java bean class obj -->
<jsp:useBean id="st" class="p1.StudentBean" scope="application"/>
<!-- set values to bean properties -->
<jsp:setProperty name="st" property="*"/>
```

→ The application that can display and rotate advertisements/adds in web pages is called as AddRotator.

→ where exactly we have utilized the collection frame work data structures in the project?

→java.util.properties is required while maintaining JDBC driver and db details

outsideteh JDBC application.
→ArrayList or Vector support will be taken to send ResultSet object data over the network.
→To maintain JNDI properties, mail properties of java mail api map data structures are required.
→ to maintain application data(huge amount) within the application collection framework data structures are required.
→ To maintain same form page data across the multiple requests as a single value during the session collection frame work data structures are required.
***→NOTE: to send collection framework data structure over the network the objects that added to data structures as elements must be serializable objects.

→ **Custom JSP tag Library:**
→Jsp tag lib is a library that contains set of JSP tags.
→programmers can work with three types of tag libraries in their jsp programs.
1) built-in tag libraries of jsp technologies(all standard Action tags of jsp)
2) Third party supplied jsp tag libraries
Ex:struts tag libraries, JSF tag libraries, JSTL and etc...
3) custom jsp tag libraries (Developed by programmers)

JSTL :
→ given by sun micro systems
→ is not part of jsp technology
→ should configure and should be used with jsp programs separately
→ Gives the following 04 number of jsp tag libraries:
1) Core jsp tag library
2) Sql jsp tag library
3) Xml jsp tag library
4) Formatting jsp tag library

Tag lib		tld file name		recommended prefix	jar files
core	→	c.tld	→	c	→ jstl.jar, standard, saxpath,jaxen-full
sql	→	sql.tld	→	sql	→ jstl.jar, standard, saxpath,jaxen-full
xml	→	x.tld	→	x	→ jstl.jar, standard, saxpath,jaxen-full
fmt	→	fmt.tld	→	fmt	→ jstl.jar, standard, saxpath,jaxen-full

→ The difference between encodeURL(-), encodeRedirectURL(-) methods of response object:
encodeURL(-) → is useful to append session id to the given URL. This is useful while working with HttpSession with URL rewriting technique.

encodeRedirectURL(-) → is useful to send session id from one web application to another application then they are interact with each other with the help of sendRedirect(-). This method is useful to make session object of one web application to another web application.

Ex: in servlet program: srv1 prog of WA1 webappl
```
String enurl=res.enCodeRedirectURL("http://localhost:8080/WA2/srv2");
res.sendRedirect(enurl);
```

this code sends session id from WA1 app to WA2 web app.

Annotations :

Important Annotations in Servlet 3.x:
@WebServlet → to configure servlet program with logical name and url pattern
@InitParam → to configure init parameters of servlet program
@ServletFilter → to configure servlet filter program
@WebServletContextListener → to configure ServletContextListeners.

→ **The Servlet api 3.0 having 03 packages:**
javax.servlet, javax.servlet.http and javax.servlet.annotation

→ **Example application using servlet annotations:**
input.html
```
<form action="test1">
        <input type="text" name="userName"><br>
        <input type="subnmit" value="check">
</form>
```

FormSrv.java
```
                import java.io.*;
                improt javax.servlet.*;
                import javax.servlet.http.*;
                import javax.servlet.annotation.WebServlet;
                @WebServlet(name="abc" urlPatterns={"/test1"})
                public class FormSrv extends HttpServlet
                {
                public void doGet(HttpServletRequest request,HttpServletResponse response)
throws ServletException, IOException
                {
                 String username=request.getParameter("userName");
                 response.setContentType("text/html");
                 PrintWriter pw=response.getWriter();
                 pw.println("Hello!Mr."+username+" Today's Date:: "+new java.util.Date());
                 pw.close();
                }
                public void doPost(HttpServletRequest request,HttpServletResponse response)
throws ServletException, IOException
                {
                doGet(request,response);
                }
                }
```
→ Request URL → http://localhost:8080/FormAnnoApp/input.html

→ To develop and use our own annotations we need to develop one simulator class called container on the top of JRE then only we can provide functionality to that annotation.

→ procedure to develop user defined annotation having the following functionality:
@sum(num1=10, num2=20)
int result → should assign 20 to result variable

Steps:
step1: make sure that jdk1.5+ s/w is available
step2: design your annotation

```
package  p3;
import java.lang.annotation.*;
@extention(RetentionPolicy.RUNTIME) → makes annotation as runtime annotation
@Target(ElementType.FIELD) → this annotation is applicable only on fields
public interface Sum {
int num1() default 0;
int num2() default 0;
}
```

step3: develop your own container class.

```
pacakage p3;
improt java.lang.reflect.Field;
public class Container
{
public static void addThis(Object theObject)
{
 try{
 //get access to all the member variables of current object(Test class obj)
 Field[] fileds = Class.forName(theObject.getClass().getName()).getDeclatedFields();
 for(int i=0;i<fields.length;i++)
 {
 //get access to @Sum annotation
 Sum obj=fileds[i].getAnnotation(Sum.class);
 if(obj!=null)
 {
 //getting values from annotations parameter
 int val1=obj.num1();
 int val2=obj.num2();
 int tot=val1+val2;
 //setting it to our class variable(like result)
 fields[i].setInt(theObject,tot);
 }//if
 }//for
}//try
 catch(Exception e){e.printStackTrace();}//catch
}//addThis
}//class
```

step 4: use the annotation in your application

```
pacakge p3;
public class Test
{
@Sum(num1=15, num2=25)
staic int result;
public static void main(String args[])
{
Test t1=new Test();
```

```
        Container.addThis(t1);//activate user-defined container class
        System.out.println("The sum is :: "+result);
        }
        }
```

step5: compile and execute the application:

→ javac -d . *.java

→ java p3.Test

Tecnicalmumbo.wordpress.com

→Reflection:

→ Reflection means mirror image.

→ Reflection api application is capable of gathering given java class or interface details programatically and dynamically.

→ Being from one application if you want to gather certain java class details like member variables, methods and etc., details programatically then develop the code of that application by using reflection api.

→ Reflection api based java applications can gather the internal details about given java class oar interface programmatically and dynamically and these details can be used in the application as input values for other requirements.

→ each reflection api based java application acts as mirror image having the ability to gather details about given class or interface dynamically.

→ Reflection api means working with classes and interfaces of java.lang.reflect package.

→ java.lang is default package in java applications but its sub packages are not default packages.

```
public final class Test
{
float static int a;
private float b;
Test()
{
 ------;
}
Test(int x)
{
----;
}
public int sum(int x, int y)
{
---;
}
}
```

→ In java environment member variables are technically called as fields and constructors are called as creation methods.

→ java **TestApp.java**

→ java **TestApp java.lang.System** → command line arg value

→ prints all the details of java.lang.System class as output

→ java **TestApp java.lang.Runnable**

→ prints all the details of java.lang.Runnable(I) as output

→ **Reflection api** is popularly used in the development of java based software products like debuggers, classBrowsers, GUI builders and etc...

→ we can gather the basic details about given class or interface by using various methods of java.lang.Class, but use classes and interfaces of java.lang.reflection api to gather advanced details.

→ **java.lang.reflect package**
→ classes → Field, Constructor, Method, Modifier
→ Interfaces → InvocationHandler, Member.

→ **program to get super class name of given class**

```
public class App1
{
public static void main(String args[]) throws Exception
{
//load given java class into App
 Class c=Class.forName(args[0]);
 //get super class of given class
 Class sc=c.getSuperClass();
 //print given class, super class name
 System.out.println("Given class name::"+c.getName());
 System.out.println("Super class name::"+sc.getName());
}//main
}//class
```

→ javac **App1.java**
→ java **App1 java.lang.String**
→ java **App1 java.lang.System**

→ when the object of java.lang.Class points to class or interface and to get that classname or interface name as String value use getName() method call on java.lang.Class object.

→ **Program to get all the classes of inheritance hierarchy for a given java class:**

```
public class App2
{
public static void main(String args[]) throws Exception
{
//load given java class into App
 Class c=Class.forName(args[0]);
 //get super class of given class
 Class sc=c.getSuperClass();
 //print all the classes of inheritance hirarchy
 System.out.println("Classes in the inheritance hierarchy for::"+arg[0]+" class are...: ");
 while (sc!=null)
 {
 System.out.println("\t\t"+sc.getName());
 c=sc;
 sc=c.getSuperClass();
 }//while
}//main
}//class
```

→ **javac App2.java**
→ java **App2 java.lang.String**
→ java **App2 java.lang.Integer**
→ java **App2 java.applet.Applet**
→ object of java.lang.class can represent a concrete class or interface or abstract class or data type.

→ **program to get list of java interfaces implemented by the given java class:**

```
public class App3
{
public static void main(String args[]) throws Exception
{
//load given java class into App
 Class c=Class.forName(args[0]);
 //call getInterfaces() of java.lang.Class
 Class inter[]=c.getInterfaces();
 //print interfaces name
 System.out.println("Given class name::"+args[0]+" related interfaces are...: ");
 for(int i=0;i<inter.length;i++)
 {
 System.out.println("\t\t"+inter[i].getName());
 }//for
}//main
}//class
```

→ javac **App3.java**
→ java **App3 java.lang.String**
→ java **App3 java.lang.Thread**

→int[] means all elements in that array are int values.
→ java.lang.class object[] means all elements in that array are the objects of java.lang.class.
→ public, final, abstract are the modifiers of java class. But we can't apply final and abstract modifiers at a time on a java class because final clas sdoes not support inheritance and abstract class needs the support of inheritance(we can not write subclass to final class but we need to write subclass to abstract class to implement abstract methods).

→ **program to get modifiers of given class**

```
public class App4
{
public static void main(String args[]) throws Exception
{
//load given java class into App
 Class c=Class.forName(args[0]);
 //call getModifiers() of java.lang.Class
 int x = c.getModifiers();
 //print modifiers
 if(Modifier.isPublic(x))
 System.out.println("\tpublic");
  if(Modifier.isAbstract(x))
 System.out.println("\tabstract");
  if(Modifier.isFinal(x))
 System.out.println("\tfinal");
}//main
}//class
```

→ javac **App4.java**
→ java **App4 java.lang.String**
→ java **App4 java.lang.Thread**

→ **program to get the details of member variables/fields available in java class**

```
public class App5
{
public static void main(String args[]) throws Exception
{
//load given java class into App
 Class c=Class.forName(args[0]);
 //call getDeclareFields() of java.lang.Class
 Filed f[] = c.getDeclareFields();
 //print Field details
 for(int i=0;i<f.length;i++)
 {
 int x=f[i].getModifiers();//given modifiers
 String type=f[i].getType().getName();//gives datatype of each field
 String name=f[i].getName();//gives name of each field
           if(Modifier.isPublic(x))
 System.out.println("\tpublic");
  if(Modifier.isAbstract(x))
 System.out.println("\tabstract");
  if(Modifier.isFinal(x))
 System.out.println("\tfinal");
  System.out.println(type+"  "+name);
 }//for
}//main
}//class
```

→ javac **App5.java**
→ java **App5 java.lang.String**
→ java **App5 java.lang.Thread**

→ **program to gather all the method details of given java class**

```
public class App6
{
public static void main(String args[]) throws Exception
{
//load given java class into App
 Class c=Class.forName(args[0]);
 //call getDeclareMethods() of java.lang.Class
 Method m[] = c.getDeclareMethods();
 //print Method details
 for(int i=0;i<m.length;++i)
```

```
          {
          int x = m[i].getModifiers();//given modifiers of each method
          String rtype = m[i].getReturnType().getName();//gives return type of each method
          String mname = m[i].getName();//gives name of each method
          Class p[]=m[i].getParameterTypes(); //gives parameter types
                    if(Modifier.isPublic(x))
          System.out.println("\tpublic");
           if(Modifier.isAbstract(x))
          System.out.println("\tabstract");
           if(Modifier.isFinal(x))
          System.out.println("\tfinal");
           System.out.println(rtype+"  "+mname+"(");
           for(int k=0;k<p.length;++k)
           {
            System.out.println(p[k].getName()+"  ");
           }//for
           System.out.println(")");
          }//for
         }//main
         }//class
```

→ javac **App6.java**
→ java **App6 java.lang.String**
→ java **App6 java.lang.Thread**

MVC1 VS MVC2

MVC is a design pattern. It contains two models:-
1)MVC Model 1
2) MVC Model 2

MVC1	**MVC2**
HTML or JSP files are used to code the presentation. JSP files use java beans to retrieve data if required.	This architecture removes the page-centric property of MVC1 architecture by separating Presentation, Control logic and Application state.
MVC1 architecture is page-centric design all the business and processing logic means any JSP page can either present in the JSP or may be called directly from the JSP page.	In MVC2 architecture there is one Controller which receive all request for the application and is responsible for taking appropriate action in response to each request.
MVC1 both view and controller implemented in servlets .	MVC2 view implemented in JSP and controller implemented in servlets.

EL Implicit Objects:-

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]} //requestScope is used for set and get attribute it is not an request object	request.getAttribute("phones");
cookie	${cookie.userName.value}	cookie[i].getValue();
initParam	${initParam.mainEmail}	config.getInitParameter("mainEmail");
sessionScope	${sessionScope.phones[0]);	session.getAttribute("phones");

JSTL

- JSTL stands for JavaServer Pages Standard Tab Library.
- It is introduced in JSTL 1.2.
- It is a set of Java tag libraries that use for simplify coding on JSP.
- JSTL includes a wide variety of tags that fit into seperate functional areas.

<u>Libraries are as follows:-</u>

Core:- http://java.sun.com/jsp/jstl/core

XML:- http://java.sun.com/jsp/jstl/xml

Internationalization:- http://java.sun.com/jsp/jstl/fmt

SQL:- http://java.sun.com/jsp/jstl/sql

Functions:- http://java.sun.com/jsp/jstl/functions

- You have required to download two jar files, jstl.jar and standard.jar from the JSTL jars athttp://jakarta.apache.org/site/downloads/downloads_taglibs-standard.cgi.

- Add jstl.jar and standard.jar to the WEB-INF/lib directory of your web application project.

When You want to create a jsp :

You need to specify which JSTL 1.0 core library tags used,which can be declared on your jsp as follows:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
```

Classic Tag

- A classic tag is a Java class.
- It implements the Tag,IterationTag, or BodyTag interface.
- It is the runtime representation of a custom action
- Tag interface,javax.servlet.jsp.tagext.Tag,was the top-level interface untill JSP 1.2 and other interfaces were sub-interfaces of the Tag interface.
- However,JSP 2.0 is new top-level interface,javax.servlet.jsp.tagext. JspTag,has been introduced.
- The Tag and SimpleTag interfaces both extend JspTag.

Tag interface:-

- Tag interface is the base interface for all classic tag.
- All classic tag handlers directly or indirectly implement Tag interface.
- Tag interface used for writing basic tags that don't involve any iterations or processing of the tag's body content.

Methods	Description
void setPageContext(PageContext)	Calls this method to set the context for current page by Container.
void setParent(Tag)	Sets the parent tag,by Container.
int doStartTag()	Invokes this method on encountering the start of the tag by Container.

Methods	Description
int doEndTag()	Invokes this method on encountering the end of the tag by Container.
Tag getParent()	Gets the parent of the current tag by this method.
void release()	Calls to release any resources held.

Iteration interface:-

- IterationTag interface used primary which the tag content is to be repeatedly generated.
- IterationTag interface extends the Tag interface and provide single new method to enable reevaluation of the body of the tag.

Methods	Description
Int doAfterBody()	Calls after the body contents of the tag are evaluated by this method.

BodyTag interface:-

- The BodyTag interface extends the IterationTag interface.
- It introduces two new methods and one new constant.
- The purpose of this interface is enable buffering of the body content of a tag.

Methods	Description
void setBodyContent(BodyContent)	Calls this method just before calling doInitBodyand this method is called only if the tag has a body by Container.
void doInitBody ()	Calls this method just before body-tag processing begins by Container.

IterationTag and BodyTag,both interfaces are closely related,and BodyTag providing a content buffering capability that builds on the IterationTag capability, IterationTag does not provide. Use BodyTag to solve the problem that we face when using IterationTag.

Custom Tags

- JSP 1.1 support creation of custom tags that enables seperation 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. <br> No Body , No Attribute
2. <hr size="3" > No Body with Attribute
3. <b>THIS IS TEXT </b> With body No Attribute
4. <Font face="Arial" size="10"> THIS IS TEXT </font> With body with Attribute.

custom.tld

```
<taglib>
 <tlib-version>1.0</tlib-version>
 <jsp-version>2.0</jsp-version>
 <short-name>Example </short-name>
 <tag>
   <name>Hello</name>
   <tag-class>com.learn.JSPExample</tag-class>
   <body-content>scriptless</body-content>
   <attribute>
      <name>message</name>
```

```
    </attribute>
  </tag>
</taglib>
```

JSPExample.java

```
package com.learn;
import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.*;

public class JSPExample extends SimpleTagSupport {

   private String message;

   public void setMessage(String msg) {
      this.message = msg;
   }

   StringWriter sw = new StringWriter();

   public void doTag()
      throws JspException, IOException
    {
       if (message != null) {
        /* Use message from attribute */
        JspWriter out = getJspContext().getOut();
        out.println( message );
      }
      else {
        /* use message from the body */
        getJspBody().invoke(sw);
        getJspContext().getOut().println(sw.toString());
      }
   }

}
```

msg.jsp

```
<%@ taglib prefix="ex" uri="/WEB-INF/custom.tld"%>
<html>
  <head>
    <title>A sample custom tag</title>
  </head>
  <body>
    <Ex:Hello message="This is Simple custom tag" />
  </body>
```