JavaServer Pages(JSP)

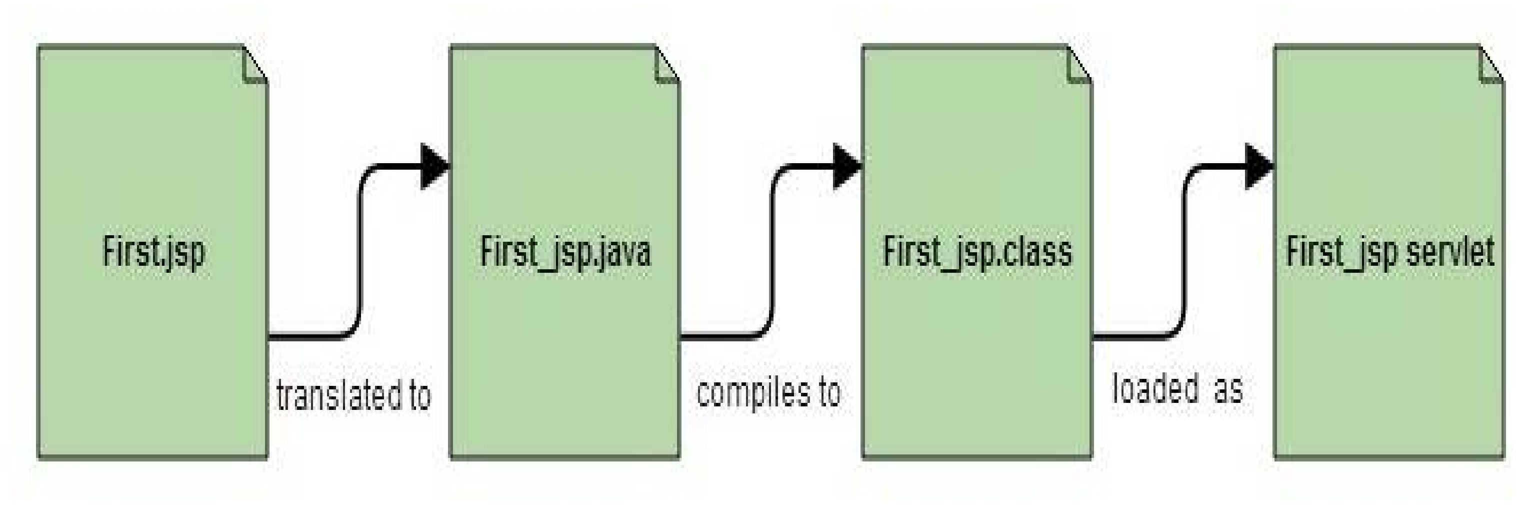
Definition

- JavaServer Pages is a scripting environment that allows you to combine Java code with HTML code to generate content dynamically.
- JSP pages are easier to maintain than a Servlet.
- JSP pages are opposite of Servlets as a servlet adds HTML code inside Java code, while JSP adds Java code inside HTML using JSP tags. Every JSP page is converted into a servlet.
- The conversion is done by the JSP engine.
- The JSP files should have extension .jsp

Why JSP is preferred over Servlets?

- JSP provides an easier way to code dynamic web pages.
- JSP does not require additional files like, java class files, web.xml etc.
- Any change in the JSP code is handled by Web Container(Application server like tomcat), and doesn't require re-compilation.
- JSP pages can be directly accessed, and web.xml mapping is not required like in servlets.

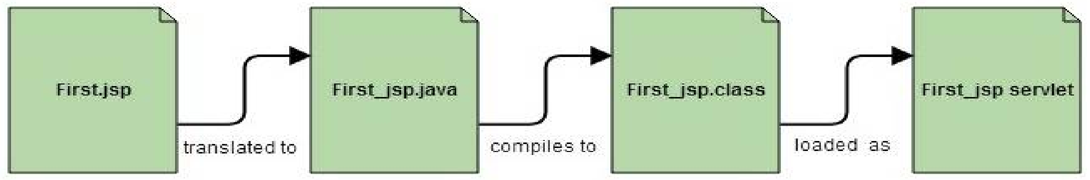
How JSP Works



JSP

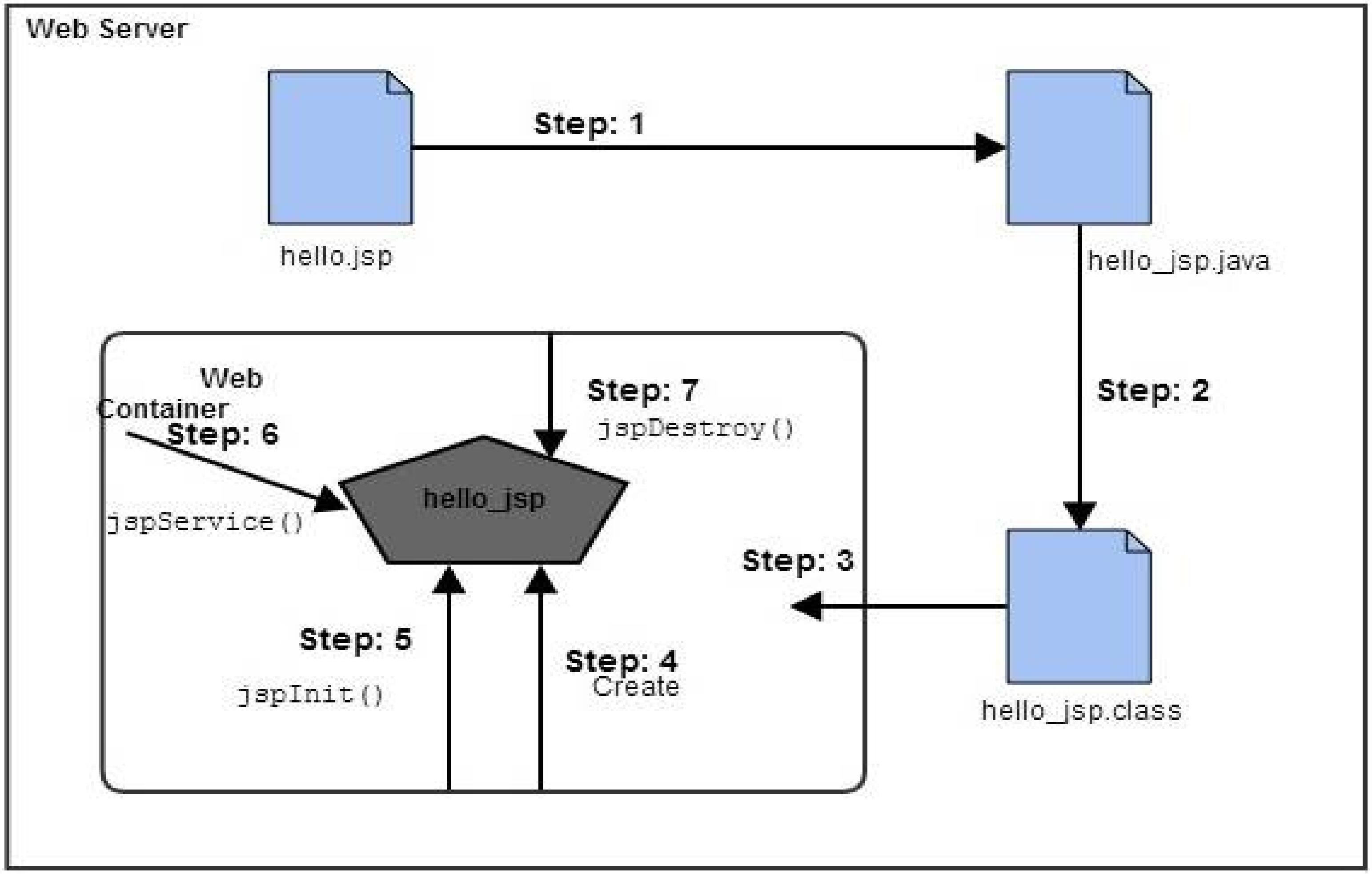
How does JSP work?

1) Browser requests for a JSP page.
2) The WebServer forwards the request to Servlet Engine.
3) Servlet Engine finds that the request is for a JSP page and forwards it to JSP engine.
4) JSP Engine parses the JSP page, converts it into a servlet (source code - file.jsp converted into file_jsp.java) and compiles it.
5) JSP Engine then forwards servlet .class file to servlet engine for execution.



Parsing, conversion into servlet source code and compilation take place only once when JSP page is requested for the first time, for subsequent requests the JSP page is ready for execution.

JSP Life Cycle



JSP life cycle

There are three events that occur in the entire life of a JSP page - initialization, request handling and shutdown. These three events correspond to three methods namely,

1) public void jspInit( )
2) public void **_jspService**(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
3) public void jspDestroy( )

JSP Code Specification

- Elements – that are processed by JSP engine on the web server
- Template data - elements that are ignored by the JSP engine

JSP Scripting Elements

- Written inside <% %> tags.
- The code written inside <% %> tags are processed by the JSP Engine.
- Other text in the JSP page is considered as HTML code or plain text.

Types of JSP Scripting Elements

Scripting Element	Example
• Comment	<%-- comment --%>
• Declaration	<%! declarations %>
• Scriptlet	<% scriptlets %>
• Expression	<%= expression %>
• Directive	<%@ directive %>

JSP Elements

JSP combine Java code with HTML using JSP elements.

1) JSP comments `<%--   comment here     --%>`

2) JSP scriptlets - allows you to write java code inside JSP page.

```
        <% Java statements %>
e.g. a)      <%
               if(total >90) ………….
        else if(total>80)…………
        else……………
         %>
    b)
  <% if (Math.random()>0.5) { %>
        HTML CODE
   <% } %>
  <% else %>
           HTML CODE
```

Example

index.html

```
 <form method="post" action="welcome.jsp">
    Name <input type="text" name="user" >
     <input type="submit" value="submit">
</form>
```

welcome.jsp

```
 <html>
   <%
      String user = request.getParameter("user");
   %>
  <body>
        Hello, <% out.println(user); %>
  </body>
</html>
```

JSP Elements

3) JSP declaration

<%! Declaration of field/method/class/interface %>

e.g.

```
a)    <%! public static final double PI = 3.14; %>

b)     <%!    public  int f(…) {
                      <%-- body of the method --%>
           }          %>
c)     <%! class X{
           <%-- body of the class --%>
       } %>
```

JSP Elements

4) JSP expression <%= Java expression %>

e.g. <%= new java.util.Date().toString()%>

<%= (2*5) %> → out.print((2*5));

JSP

Behind the scene a JSP page is converted into a servlet source file.
For conversion:
The JSP expressions go inside the _jspService( ) method replacing a **JSP expression** by
out.print(JSPExpression)

e.g. <%= new java.util.Date( ).toString( )%>
Is converted into
out.print(new java.util.Date( ) .toString( ));

The regular HTML becomes print statements with double quotes around the text. e.g. out.print(“ HTML code”);

JSP

The JSP scriptlets also go inside _jspService method after removing the JSP scriptlet tag.

e.g.
```
<%
           if(total >90) …………..
    else if(total>80)…………
    else……………
       %>
```

Will be written inside _jspService method as

```
           if(total >90) …………..
    else if(total>80)…………
    else……………
```

JSP

JSP declarations result in code that is placed **inside the generated servlet class** definition but **outside the _jspService method**.

Since fields, methods, classes and interfaces can be declared in any order, it does not matter whether the code from declarations goes at the top or bottom of the generated servlet.

The JSP comments are ignored by the JSP engine and so they are not found inside the generated servlet.

JSP

```
<H1>A Random Number</H1>
<%= Math.random() %>
```

<u>Inside the generated servlet</u>

```
public void _jspService(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException {
HttpSession session = request.getSession();
JspWriter out = response.getWriter();
//other implicit objects
response.setContentType("text/html");
out.print("<H1>A Random Number</H1>");
out.print(Math.random());
...
}
```

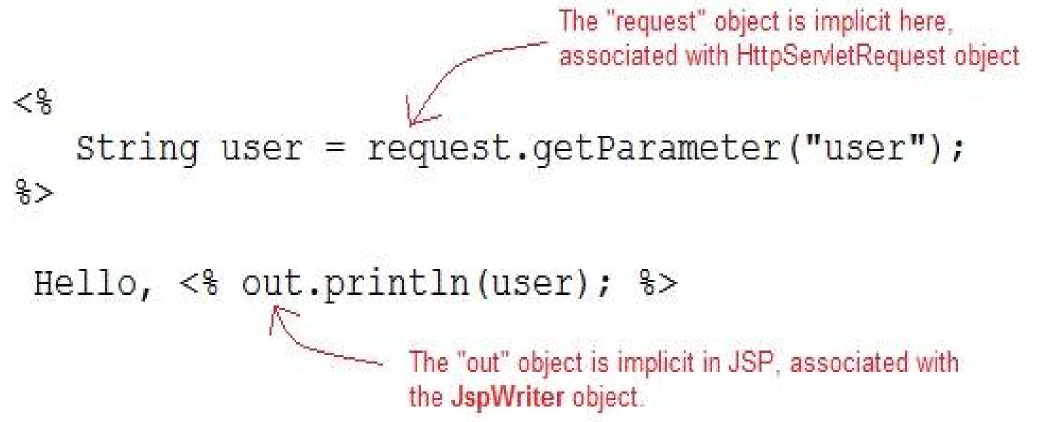
JSP

Implicit Objects

- **Implicit objects in jsp are the objects that are created by the container automatically and the container makes them available to the developers, the developer do not need to create them explicitly. Objects that are accessible to JSP writer without him/her declaring the object.**

Can be accessed only inside _jspService Method.

In a JSP page implicit objects are accessible only from JSP expression and JSP scriptlets.



```
<%
   String user = request.getParameter("user");
%>

Hello, <% out.println(user); %>
```

The "request" object is implicit here, associated with HttpServletRequest object
The "out" object is implicit in JSP, associated with the JspWriter object.

JSP

The Name and Type of Implicit Objects

Name	**Type**
session	**HttpSession**
e.g. <%= session.getId() %>	
request	**HttpServletRequest**
e.g. <%= request.getParameter(“item”) %>	
response	**HttpServletResponse**
out	**JspWriter**

JSP

application	**ServletContext**
config	**ServletConfig**
exception	**Throwable**
page	**Object**
pageContext	**PageContext**

JSP

If you are writing code that is **not part of the _jspService method**, these variables are not available.

Since **JSP declarations** result in **code that appears outside** the _jspService method, these **variables are not accessible inside JSP declarations**.

JSP

About pageContext object

pageContext provides a single point access to other Implicit Objects.

```
out=pageContext.getOut();
session=pageContext.getSession();
request=pageContext.getRequest();
response=pageContext.getResponse();
application=pageContext.getServletContext();
config=pageContext.getServletConfig();
```

JSP

5) <u>JSP directive</u>

- Directive Tag gives special instruction to Web Container at the time of page translation.
- There are three types of JSP directives:
  - Page
  - include
  - taglib

JSP

JSP page directive

- Defines page dependent properties such as language, session, errorPage etc
- A JSP page directive can be placed **anywhere** within the
- JSP page & there may be any number of page directives.

Syntax:

<%@ page

attribute_1 = “list of values separated by comma”

attribute_2 = “list of values separated by comma”

attribute_n = “list of values separated by comma”

%>

JSP

Importing packages

<%@ page **import** =” java.sql.*, java.io.* ” %>

OR

<%@ page import =” java.sql.*” %>

<%@ page import =”java.io.*” %>

JSP

<%@ page **session** = ”true|false” %>
If session attribute is not present in the page directive then its default value is true.

Whenever a JSP page is created a new session is created, The value of the session attribute controls whether the page participates in HTTP session.
A value of session = “false” means that no session will be automatically created and that attempts to access the variable session will result in errors at the time the JSP page is translated into a servlet.

JSP

<%@ page **content-type**=”type” %>
default is text/html
<%@ page **buffer**=”none|8kb|32kb” %>
8kb is default value or a multiple of 8kb is allocated but at least 8kb is allocated
<%@ page **autoFlush**=”true|false” %> //by default is true.
<%@ page **extends**=”pkg1.ClassName” %>
The generated Servlet extends the class (pkg1.ClassName)

JSP

<%@ page **language**=”java” %>

<%@ page **errorPage**=”url” %>

url is the path of the page the user is redirected whenever exception occurs.

<%@ page **isErrorPage**=”true|false” %> default is false

e.g.   If url= page2.jsp

then Inside the error page(page2.jsp)we must have this page directive.

JSP

<%@ page **info**=”text” %>
Info about the author of the JSP page, date it was modified and so on.

The info attribute defines a string that can be retrieved from the servlet by means of the **getServletInfo** method.

JSP

JSP include directive (static inclusion)

<%@ include file=”***relative-path-to-resource***” **%>**

JSP

JSP include directive (static inclusion)
You use the include directive to include a file in the main JSP document at the time the document is translated into a servlet (which is typically the first time it is accessed). The syntax is as follows:

<%@ include file=”***relative-path-to-resource***” **%>**

The content of the file (not its output) is incorporated into this JSP page at the time of translating it into servlet.

Think of the include directive as a preprocessor: the included file is inserted character by character into the main page, then the resultant page is treated as a single JSP page.

Taglib directive

- Collection of custom tags that can be used by the page.
- Syntax

  <%@ taglib prefix="prefixOfTag" uri="uriOfTagLibrary" %>

  - TagLibraryURI – URI of tag library descriptor
  - Prefix: defines the custom tag.

Example

```
<html>
    <head>
        <title>Welcome Page</title>
    </head>
   <%@ taglib prefix="mine" uri="myTags" %>
  <body>
        Welcome,  <mine:userName / >
  </body>
</html>
```

JSP Actions

- Action elements are predefined functions.
- <jsp:action_name attribute="value" />
- Actions available
  - jsp:include
  - jsp:forward
  - jsp:useBean
  - jsp:setProperty
  - jsp:getProperty

JSP

Runtime inclusion (Dynamic inclusion)

If you want to incorporate the output of a JSP page or a servlet or an HTML page into this JSP page then use JSP include action.

Syntax:

<jsp:include page="*relative-path-to-resource*" />

OR

<jsp:include page=”*relative-path-to-resource* >
<%-- JSP code --%> <%-- This is optional --%>
</jsp:include>

jsp:include

```
<jsp:include page="relativeURL | <%= expression %>">
        <jsp:param name="parametername"
                      value="parametervalue | <%=expression%>" />
</jsp:include>
```

JSP

The fundamental difference between jsp:include and the include directive is the time at which they are invoked: jsp:include is invoked at request time, whereas the include directive is invoked at page translation time. However, there are more implications of this difference than you might first think.

JSP

<u>Differences Between jsp:include and the include Directive</u>

What does basic syntax look like?	
<jsp:include page="..." />	<%@ include file="..." %>
When does inclusion occur?	
Request time	Page translation time
What is included?	
Output of page	Actual content of file
How many servlets result?	
Two	One

JSP

Request forwarding

<jsp:forward

page=” *relative-path-to-resource*” flush=”true”/>

OR

<jsp:forward page=” *relative-path-to-resource*” flush=”true”>

<%-- JSP code --%> <%-- This is optional --%>

</jsp:forward>

JSP

Example jsp:forward

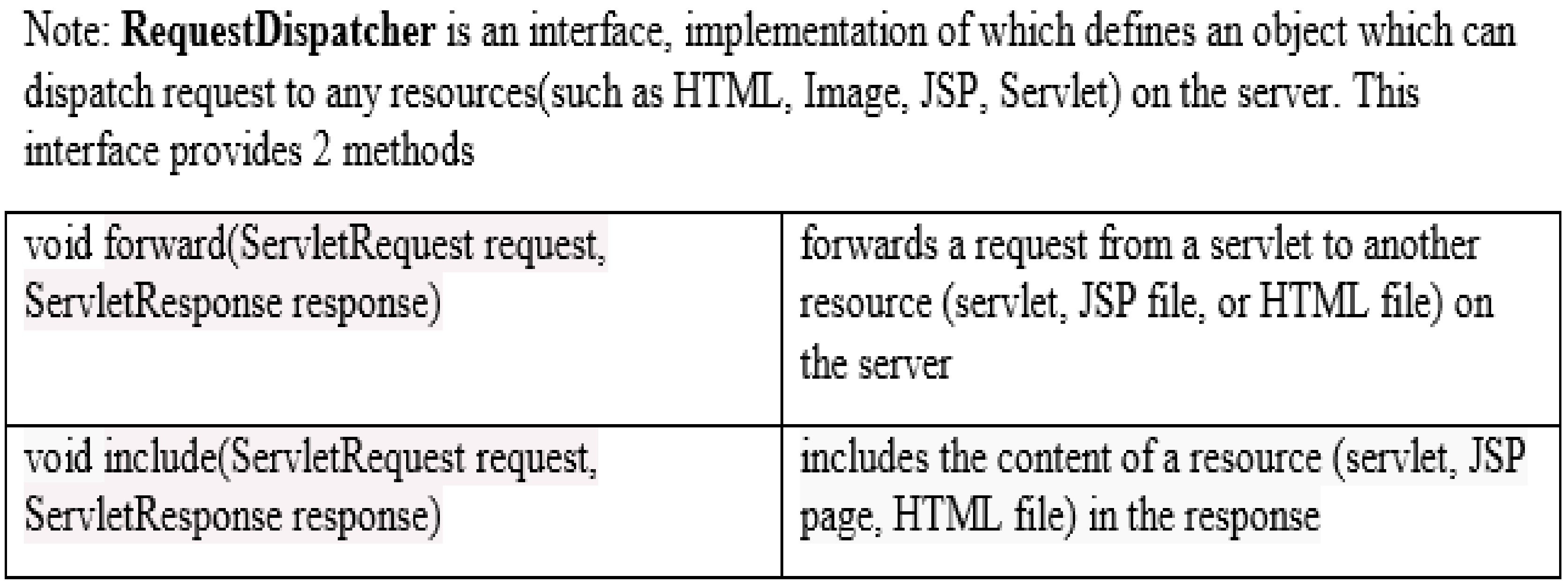
```
<% String destination;
if (Math.random() > 0.5) {
destination = "/examples/page1.jsp";
} else {
destination = "/examples/page2.jsp";
}
%>
<jsp:forward page="<%= destination %>" />
```

JSP

Passing Parameters

```
<jsp:forward page=”name” flush=”true”>
   <jsp:param name=”pname1” value=”value1” />
   <jsp:param name=”pname2” value=”value2” />
</jsp:forward>
```

RequestDispatcher



Note: **RequestDispatcher** is an interface, implementation of which defines an object which can dispatch request to any resources(such as HTML, Image, JSP, Servlet) on the server. This interface provides 2 methods

void forward(ServletRequest request, ServletResponse response)	forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server
void include(ServletRequest request, ServletResponse response)	includes the content of a resource (servlet, JSP page, HTML file) in the response

RequestDispatcher

To invoke these methods so as to navigate to various pages.

getRequestDispatcher() method of ServletRequest returns the object of RequestDispatcher.

```
RequestDispatcher rd =
                              request.getRequestDispatcher("Registrationform.jsp");
```

JavaBean

- A JavaBean is a specially constructed Java class written in the Java and coded according to the JavaBeans API specifications.

JavaBean

- Following are the unique characteristics that distinguish a JavaBean from other Java classes −
  - It provides a default, no-argument constructor.
  - It should be serializable and that which can implement the **Serializable** interface.
  - It may have a number of properties which can be read or written.
  - It may have a number of "**getter**" and "**setter**" methods for the properties.

JavaBeans Properties

- A JavaBean property is a named attribute that can be accessed by the user of the object. The attribute can be of any Java data type, including the classes that you define.
- A JavaBean property may be **read, write, read only**, or **write only**. JavaBean properties are accessed through two methods in the JavaBean's implementation class −

S.No.	**Method and Description**
1	get**PropertyName**() For example, if property name is *firstName*, your method name would be **getFirstName()** to read that property. This method is called accessor.
2	set**PropertyName**() For example, if property name is *firstName*, your method name would be **setFirstName()** to write that property. This method is called mutator.

- A read-only attribute will have only a **getPropertyName()** method, and a write-only attribute will have only a **setPropertyName()** method.

Example - Assume this class is available in package BeanPackage

```
public class StudentClass implements java.io.Serializable {
    private String firstName = null;    private String lastName = null;     int age=0;
   public StudentClass() {
   }
   public String getFirstName(){
      return firstName;
   }
   public String getLastName(){
      return lastName;
   }
   public int getAge(){
       return age;
   }
   public void setFirstName(String firstName){
      this.firstName = firstName;
   }
   public void setLastName(String lastName){
      this.lastName = lastName;
   }
   public void setAge(int age){
      this.age=age;
   }
}
```

Accessing JavaBeans

- The **useBean** action declares a JavaBean for use in a JSP. Once declared, the bean becomes a scripting variable that can be accessed by both scripting elements and other custom tags used in the JSP. The full syntax for the useBean tag is as follows −

```
<jsp:useBean id = "bean's name" scope = "bean's scope" class="className">
   . . .
</jsp:useBean>
```

- Here values for the
  - **scope** attribute can be a **page, request, session** or **application** based on your requirement.
  - **id** attribute may be any value as a long as it is a unique name among other **useBean declarations** in the same JSP.

Accessing JavaBeans Properties

- Along with **<jsp:useBean...>** action, you can use the
  - **<jsp:getProperty/>** action to access the get methods
  - **<jsp:setProperty/>** action to access the set methods.
- Syntax

```
<jsp:useBean id = "id" class = "bean's class" scope = "bean's scope">
   <jsp:setProperty name = "bean's id" property = "property name"
      value = "value"/>
   <jsp:getProperty name = "bean's id" property = "property name"/>
   ...........
</jsp:useBean>
```

The name attribute references the id of a JavaBean previously introduced to the JSP by the useBean action.

The property attribute is the name of the **get** or the **set** methods that should be invoked.

Example

```
<html>
   <body>
     <%
      String firstName = request.getParameter("fName");
      String lastName = request.getParameter("lName");
      int age = Integer.parseInt(request.getParameter("age"));
     %>
    <jsp:useBean id = "students" class = "BeanPackage.StudentClass">
      <jsp:setProperty name = "students" property = "firstName" value = "<%=firstName %>"/>
      <jsp:setProperty name = "students" property = "lastName" value = "<%=lastName %>"/>
      <jsp:setProperty name = "students" property = "age" value = "<%=age %>"/>
    </jsp:useBean>
    <p>Student First Name:
      <jsp:getProperty name = "students" property = "firstName"/>
    </p>
    <p>Student Last Name:
      <jsp:getProperty name = "students" property = "lastName"/>
    </p>
    <p>Student Age:
      <jsp:getProperty name = "students" property = "age"/>
    </p>
  </body>
</html>
```