

Name of the School: School of Computer Science and Engineering

Course Code: R1UC602C

Course Name: Web Technology

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Subject Name: Web Technology

Day: 29

Topics Covered: Server Side Scripts-JSP

Prerequisites, Objectives and Outcomes

Prerequisite of topic: Basic concepts related to web programming

Objective: To make students aware about the server side programming using JSP.

Outcome : 1. Students will be able to use JSP as a server side technology.

2. Students will be able to use web server along with deployment of application

3. Students will be able to implement in practical applications.

Java Server Pages

Servlet

- JAVA APPLETs run on client side (in web browsers) or writing Internet applications.
- JAVA SERVLETs run on server side for web-based application. (No GUI is required)
- JAVA have built-in support for multithread.
- Servlet API is Standard Java Extension API, (NOT part of core Java) and available as add-on package.

Security

- Rely on HTTP-specific authentication
- Secure Socket Layer (SSL)
- Java advantage: no memory access violations, strong typing violations. (Servlet will not crash servers.)
- Security Manager. Only trusted servlets will be allow to access network services or local files.
- Support fine grained access control (more secure than MS. ActiveX.)

JavaServer Pages VS. Servlets

- Dynamic Page require for.
 - Working on any web or application server
 - Separating the application logic from the appearance of the page
 - Allowing fast development and testing
 - Simplifying the process of developing interactive web-based applications
- JSP is a new approach to fit this need.
- Servlet ,to turn page, have to edit and recompile.

JavaServer Pages Approach

- Separating content generation from presentation
- Emphasizing reusable components
- Simplifying page development with tags
- Java Technology benefits (memory management and security).
- Scalability (integrated with J2EE)

JSP

- JSP enables us to write HTML pages containing tags, inside which we can include powerful Java programs.
- **Using JSP, one can easily separate Presentation and Business logic** as a web designer can design and update JSP pages creating the presentation layer and java developer can write server side complex computational code without concerning the web design. And both the layers can easily interact over HTTP requests.

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.

Advantage of JSP

- Easy to maintain and code.
- High Performance and Scalability.
- JSP is built on Java technology, so it is platform independent.

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

Advantages of JSP over Servlet

- There are many advantages of JSP over the Servlet. They are as follows:

- **1) Extension to Servlet**

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

- **2) Easy to maintain**

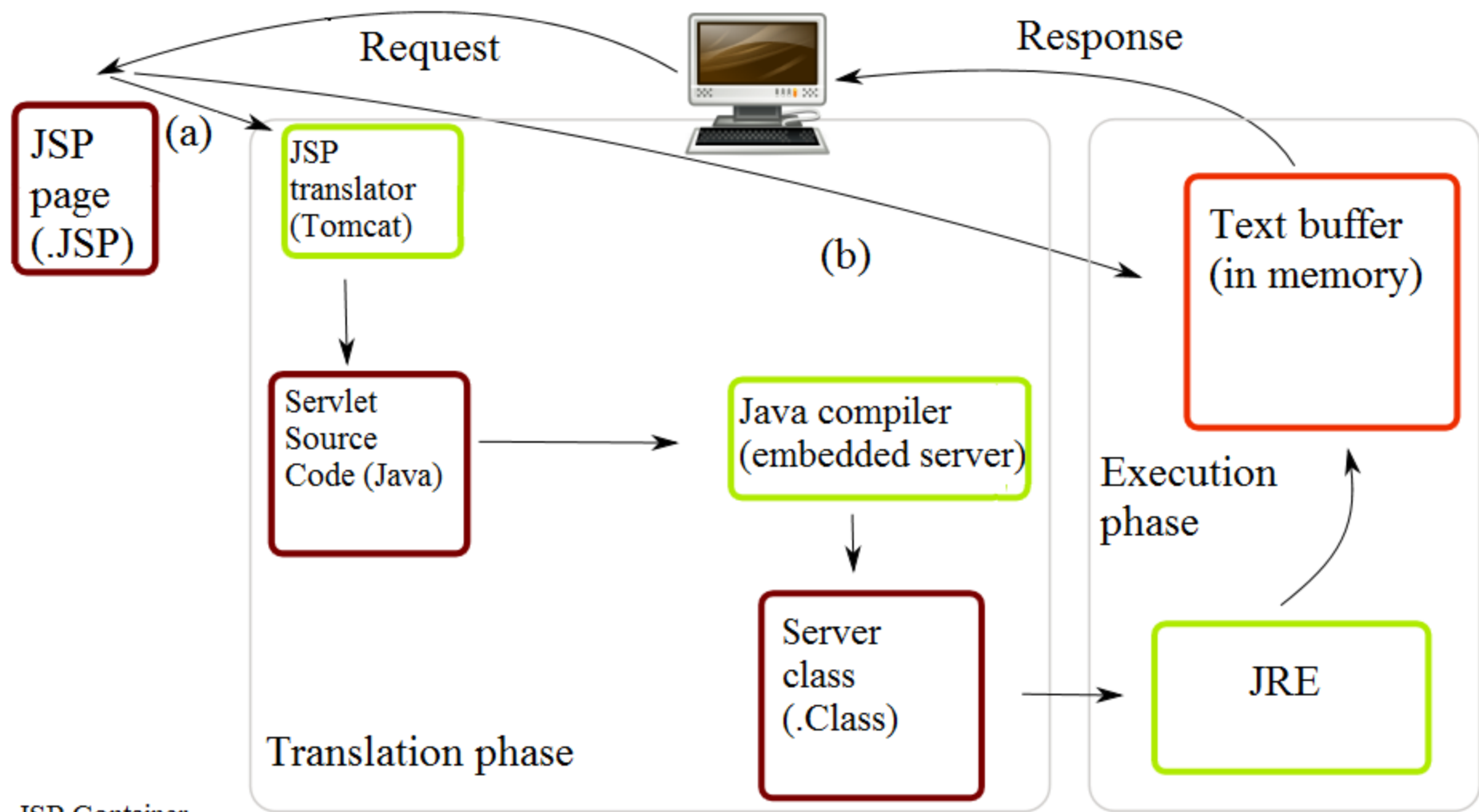
JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

- **3) Fast Development:** No need to recompile and redeploy

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

- **4) Less code than Servlet**

In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.



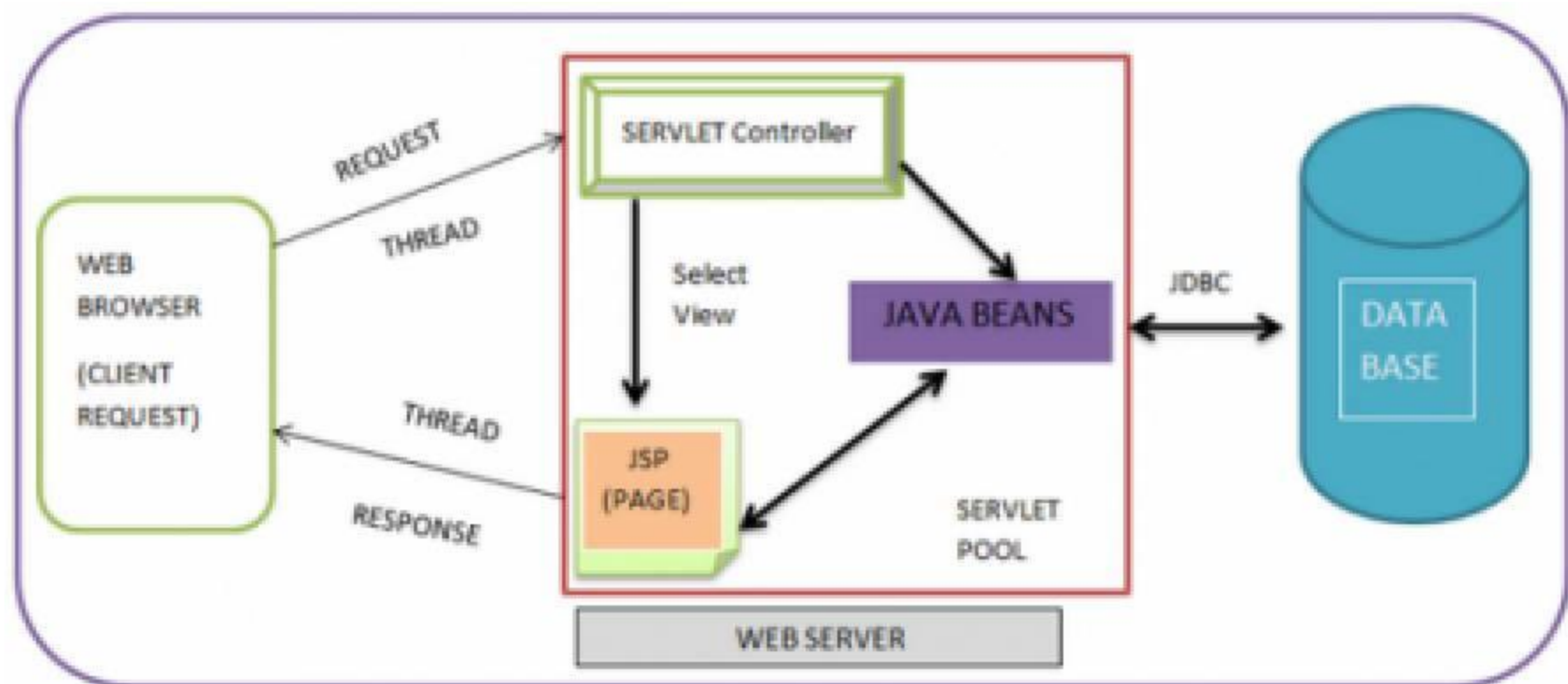
JSP Container

(a) Translation occurs at this point, if JSP has been changed or is new.

(b) If not, translation is skipped.

Lifecycle of JSP

- A JSP page is converted into Servlet in order to service requests.
- The translation of a JSP page to a Servlet is called Lifecycle of JSP. JSP Lifecycle is exactly same as the Servlet Lifecycle, with one additional first step, which is, translation of JSP code to Servlet code.
- **Following are the JSP Lifecycle steps:**
 - Translation of JSP to Servlet code.
 - Compilation of Servlet to bytecode.
 - Loading Servlet class.
 - Creating servlet instance.
 - Initialization by calling `jspInit()` method
 - Request Processing by calling `_jspService()` method
 - Destroying by calling `jspDestroy()` method
 -



Hello World

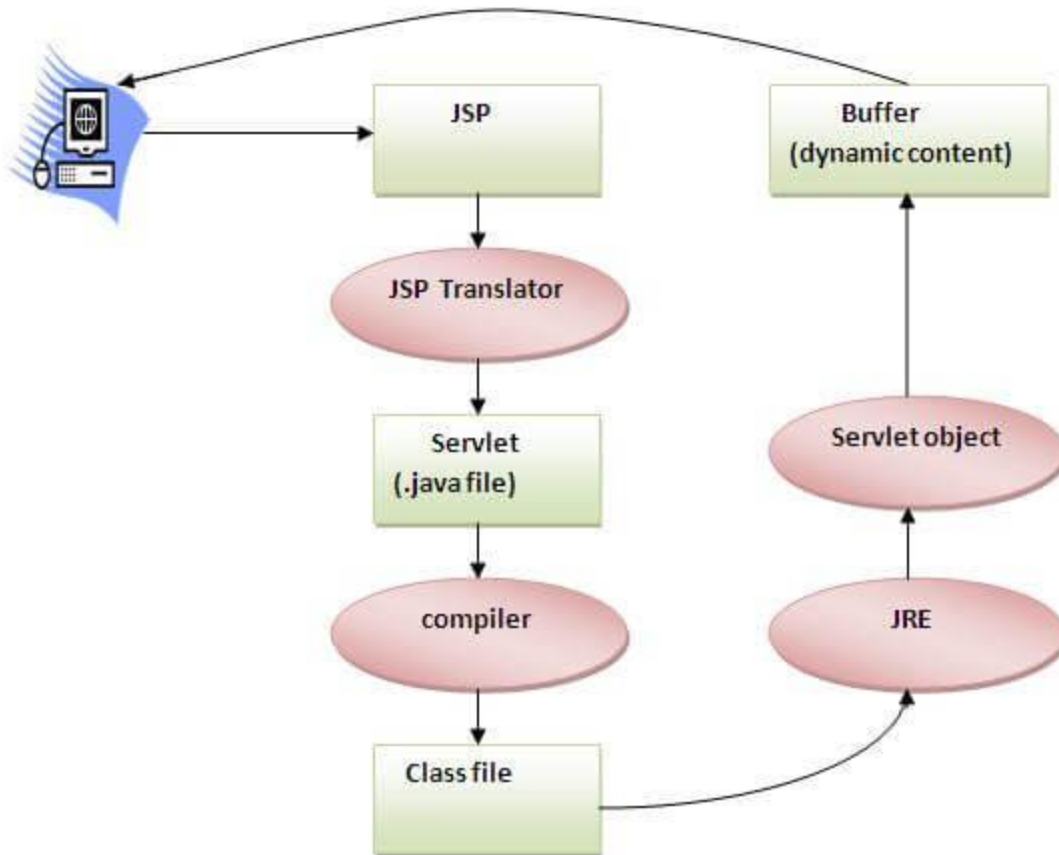
- `<%@ page language="java" contentType="text/html; "%>`
- `<html>`
- `<head>`
- `<title>Hello World - JSP tutorial</title>`
- `</head>`
- `<body>`
- `<%= "Hello World!" %>`
- `</body>`
- `</html>`

web.xml

- `<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">`
- `<web-app>`
- `<servlet>`
- `<servlet-name>start</servlet-name>`
- `<jsp-file>/start.jsp</jsp-file>`
- `</servlet>`
- `<servlet-mapping>`
- `<servlet-name>start</servlet-name>`
- `<url-pattern>/start/*</url-pattern>`
- `</servlet-mapping>`
- `</web-app>`

The Lifecycle of a JSP Page

- The JSP pages follow these phases:
- Translation of JSP Page
- Compilation of JSP Page
- Classloading (the classloader loads class file)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (the container invokes `jspInit()` method).
- Request processing (the container invokes `_jspService()` method).
- Destroy (the container invokes `jspDestroy()` method).



Creating a simple JSP Page

- Index.jsp

```
<html>
```

```
<body>
```

```
<% out.print(2*5); %>
```

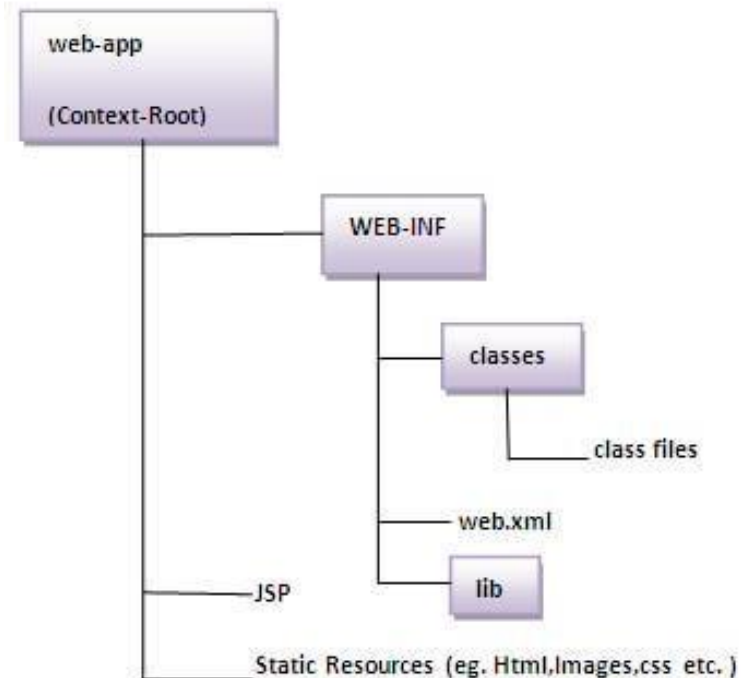
```
</body>
```

```
</html>
```

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

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

- No, there is no need of directory structure if you don't have class files or TLD files. For example, put JSP files in a folder directly and deploy that folder. It will be running fine. However, if you are using Bean class, Servlet or TLD file, the directory structure is required.
- The Directory structure of JSP



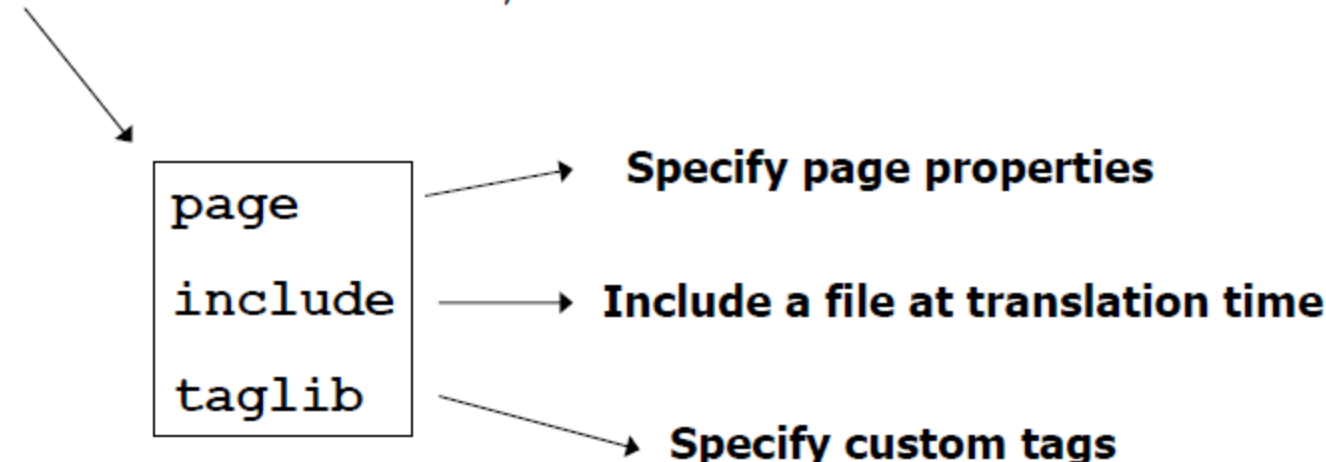
JSP elements (overview)

- Writing a program in JSP is nothing but making use of various tags which are available in JSP.
- In JSP we have three categories of tags; they are
 - 1. **Directives** of the form `<%@ ... %>`
 - 2. **Scripting elements**
 - [?] Expressions of the form `<%= expr %>`
 - [?] Scriptlets of the form `<% code %>`
 - [?] Declarations of the form `<%! code %>`
 - [?] JSP Comments `<%-- ... --%>`
 - 3. **Standard actions**
 - [?] Example: `<jsp:useBean> ... </jsp:useBean>`
 - [?] Implicit variables like request, response, out

JSP Directives

- They have the form

```
<%@ name attribute1="...", attribute2="..." ... %>
```



page
include
taglib

Specify page properties

Include a file at translation time

Specify custom tags

JSP Directive Examples

- Import java packages

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

- Multiple import statements

-  **<%@ page import="java.util.*" %>**

<%@ page import="java.sql.*" %>

-  Including file at *translation time*

<%@ include file="header.html" %>

Important JSP tags

- **1. Directives**
- **2. Declarations**
- **3. Scriptlets**
- **4. Expressions**

1. Directives

- These types of tags are used primarily to import packages. Alternatively you can also use these tags to define error handling pages and for session information of JSP page.
- Code: <%@page language="java" %>
- Code: <%@page language="java" session="true" %>
- Code: <%@page language="java" session="true" errorPage="error.jsp" %>
- Code: <%@page language="java" import="java.sql.*, java.util.*" %>
- Code: <%@ include file="/title.jsp"%>

2. Declarations

- JSP declarations starts with '<%! ' and ends with '%>'. In this you can make declarations such as `int i = 1`, `double pi = 3.1415` etc. If needed, you can also write Java code inside declarations.
- ```
<%! int radius = 7;
 double pi = 3.1415; %>
<%! double radius = 7;
 double pi = 3.1415;
 double area()
 {
 return pi*radius*radius;
 }
%>
```

# Scriptlet

- JSP Scriptlets starts with '<% ' and ends with '%>'. Scriptlets are basically used to write a pure java code. Whatever the java code we write as a part of scriplet, that code will be available as a part of service () method of servlet.

## Example

Code:

- <% String id, name, dob, email, address;
- id = request.getParameter("id");
- name = request.getParameter("name");
- dob = request.getParameter("dob");
- email = request.getParameter("email");
- address = request.getParameter("address");
- sessionEJB.addClient(id, name, dob, email, address); %>
- request, response, session and out are the variables available in scriptlets.

# JSP Scriptlet Examples

- Check a request parameter
- **<% String name = request.getParameter("name");**
- **if (name == null)**
- **{ %>**
- **<h3>Please supply a name</h3>**
- **<% }**
- **else**
- **{ %>**
- **<h3>Hello <%= name %></h3>**
- **<% } %>**

# Expressions

- JSP expressions starts with '<%= ' and ends with '%>'. If you want to show some value, you need to put it in between these tags.

**Example:**

Code:

- <%! double radius = 7;
- double pi = 22/7;
- double area()
- {
- return pi\*radius\*radius;
- } %>
- <html> <body> Area of circle is <%= area() %> </body> </html>
- Output: Area of circle is 147.0
- **Note:** Expressions in the expression tag should not be terminated by semi-colon (;) .

# JSP Expression Examples

- Displaying request parameters (request is an implicit object available in a JSP)
- Doing calculations

Your name is `<%= request.getParameter("name") %>`  
and your age is `<%= request.getParameter("age") %>`

- The value of pi is `<%= Math.PI %>` and the square root
- of two is `<%= Math.sqrt(2.0) %>` and
- today's date is `<%= new java.util.Date() %>`.



# A simple JSP

- `<html>`
  - `<head><title>JSP Test</title></head>`
  - `<body>`
  - `<h1>JSP Test</h1>`
  - Time: `<%= new java.util.Date() %>`
  - `</body>`
  - `</html>`
- 
- The expression scripting element `<%= ... %>` is equivalent to the scriptlet `<% out.print(...); %>`

# Example of JSP expression tag

- In this example of jsp expression tag, we are simply displaying a welcome message.
- **<html>**
- **<body>**
- `<%= "welcome to jsp" %>`
- Current Time: `<%= java.util.Calendar.getInstance().getTime() %>`
- **</body>**
- **</html>**
- Note: Do not end your statement with semicolon in case of expression tag

## Example of JSP expression tag that prints the user name

- Index.jsp

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

*File: welcome.jsp*

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

</body>
</html>
```

# JSP Implicit Objects

- There are **9 jsp implicit objects**. These objects are *created by the web container* that are available to all the jsp pages.
- The available implicit objects are out, request, config, session, application etc.
- A list of the 9 implicit objects is given below:

# 9 implicit objects

These objects are *created by the web container* that are available to all the jsp pages.

Object	Type
out	JspWriter
request	HttpServletRequest
response	HttpServletResponse
config	ServletConfig
application	ServletContext
session	HttpSession
pageContext	PageContext
page	Object
exception	Throwable

# 1) JSP out implicit object

- For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWriter. In case of servlet you need to write:
- `PrintWriter out=response.getWriter();`

# JSP response implicit object

- In JSP, response is an implicit object of type `HttpServletResponse`. The instance of `HttpServletResponse` is created by the web container for each jsp request.
- It can be used to add or manipulate response such as redirect response to another resource, send error etc.

**index.html**

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

</form>
```

**welcome.jsp**

```
<%
response.sendRedirect("http://www.google.com");
%>
```

# errorPage isErrorPage exception JSP

- "errorPage" and "isErrorPage" are the two of [14 attributes](#) supported by page directive.
- "errorPage"
- Any JSP file declared with **errorPage**, when generates exceptions, can send the exceptions to another JSP file that is declared with **isErrorPage**. errorPage attribute is used to specify the address of another JSP file where isErrorPage is set to true. **errorPage** and **isErrorPage** go together.
- **Syntax:** <%@ page errorPage="URL of other JSP file" %>



# isErrorPage

- Any JSP file declared with **isErrorPage** (and set to a value of true), is capable to receive exceptions from other JSP pages. Implicit object exception is available to these pages only (set with true). Default value is false.
- **Syntax:** `<%@ page isErrorPage="true" %>`
- It should be noted that **errorPage** and **isErrorPage** are used in combination with "**exception**" object.

- "exception" is one of the [9 implicit objects](#) supported by Java. "exception" is an object of [java.lang.Throwable](#), available to the programmer to be used directly in the code without creating it.
- The **exception object** is used to print the exception message at runtime in JSP file, or to say, exception object represents errors and exceptions.

# program\*(Index.html)

- `<html>`
- `<body>`
- `<h2>Find Quotient</h2>`
- `<form method="get" action="start.jsp"> <b>`
- `Enter First Number <input type="text" name="t1"> <br>`
- `Enter Second Number <input type="text" name="t2"> <br>`
- `<input type="submit" value="Please Send"> </b>`
- `</form>`
- `</body>`
- `</html>`

# Start.jsp

- `<html>`
- `<head><title>Sum of two number</title></head>`
- `<body>`
- `<%@ page errorPage="Receive.jsp" %>`
- `<%`
- `int fn = Integer.parseInt(request.getParameter("t1"));`
- `int sn = Integer.parseInt(request.getParameter("t2"));`
- `%>`
- `<h3>Your first number is <%= fn %> and second number is <%= sn %>.`  
`<br>`
- `Quotient is <%= fn/sn %> </h3>.`
- `</body></html>`

# Receive.jsp

- `<body>`
- `<%@ page isErrorPage="true" %>`
- `<h3> <font color="red">`
- `Sorry, Quotient cannot be printed. <br>`
- `Cause of problem: <%= exception %>`  
`</font> </h3>`
- `</body>`

# Config Implicit Object in JSP

- It is an instance of **javax.servlet.ServletConfig**. Config Implicit object is used for getting configuration information for a particular JSP page. Using application implicit object we can get application-wide initialization parameters, however using Config we can get initialization parameters of an individual servlet mapping.

# Methods of Config Implicit Object

- **String getInitParameter(String paramname)** – Same what we discussed in application implicit object tutorial.
- **Enumeration getInitParameterNames()** – Returns enumeration of Initialization parameters.
- **ServletContext getServletContext()** – This method returns a reference to the Servlet context.
- **String getServletName()** – It returns the name of the servlet which we define in the web.xml file inside `<servlet-name>` tag.

# To know servlet name

- `<html>`
- `<head> <title> Config Implicit Object</title>`
- `</head>`
- `<body>`
- `<%`
- `String sname=config.getServletName();`
- `out.print("Servlet Name is: "+sname);`
- `%>`
- `</body>`
- `</html>`



# context-param

- The “context-param” tag is define in “web.xml” file and it provides parameters to the entire web application.
- For example, store administrator’s email address in “context-param” parameter to send errors notification from our web application.

# ShoppingCart.html

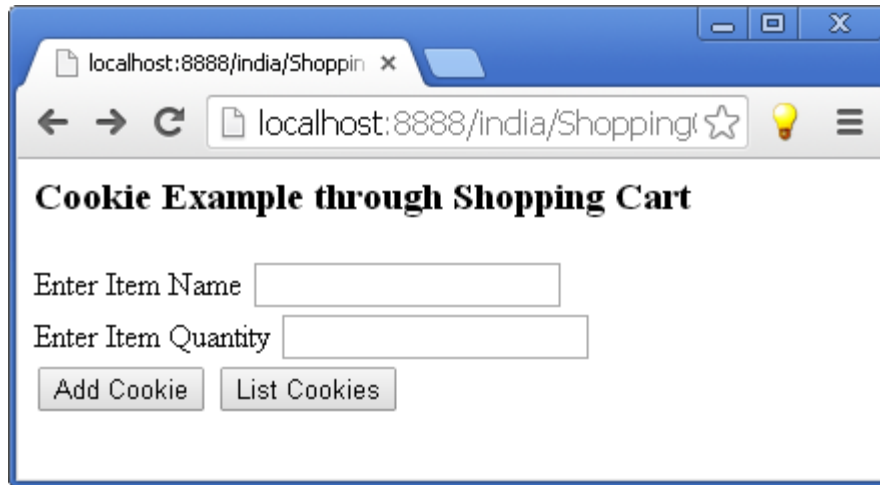
- <Html>
  - <body>
- ```
<form method="get" action="http://localhost:8888/india/ShoppingCart.jsp">
```
- - Enter Item Name <input type="text" name="item">

 - Enter Item Quantity <input type="text" name="qty">

 - <input type="submit" value="Add Cookie" name="add">
 - <input type="submit" value="List Cookies" name="list">
 -
 - </form>
 - </body></html>

- `<body>`
- `<% String str1 = request.getParameter("item"); // item name`
- `String str2 = request.getParameter("qty"); // item quantity`
- `String str3 = request.getParameter("add"); // submit button by name add`
- `String str4 = request.getParameter("list"); // submit button by name list`
- `if(str3 != null)`
- `{`
- `Cookie c1 = new Cookie(str1, str2);`
- `response.addCookie(c1);`
- `response.sendRedirect("ShoppingCart.html");`
- `}`
- `else if(str4 != null)`
- `{`
- `Cookie clientCookies[] = request.getCookies();`
- `for(int i = 0; i < clientCookies.length; i++)`
- `{ out.print("" + clientCookies[i].getName() + " : " +`
- `clientCookies[i].getValue() + "
"); } } %>`
- `</body>`

- Empty ShoppingCart:



A screenshot of a web browser window with the address bar showing 'localhost:8888/india/Shoppin'. The page title is 'Cookie Example through Shopping Cart'. The form contains two input fields: 'Enter Item Name' and 'Enter Item Quantity'. Below the fields are two buttons: 'Add Cookie' and 'List Cookies'.



A screenshot of a web browser window with the address bar showing 'localhost:8888/india/Shoppin'. The page title is 'Cookie Example through Shopping Cart'. The form contains two input fields: 'Enter Item Name' with the value 'REXONA' and 'Enter Item Quantity' with the value '3'. Below the fields are two buttons: 'Add Cookie' and 'List Cookies'.

Output screen when
List Cookies submit
button is clicked (after
adding few items).



A screenshot of a web browser window with the address bar showing 'localhost:8888/india/SC?item='. The page displays the following output:

```
JSESSIONID : 68CC7353C7606CA75C5D634204C425A3  
LUX : 10  
SANTOOR : 5  
REXONA : 3
```