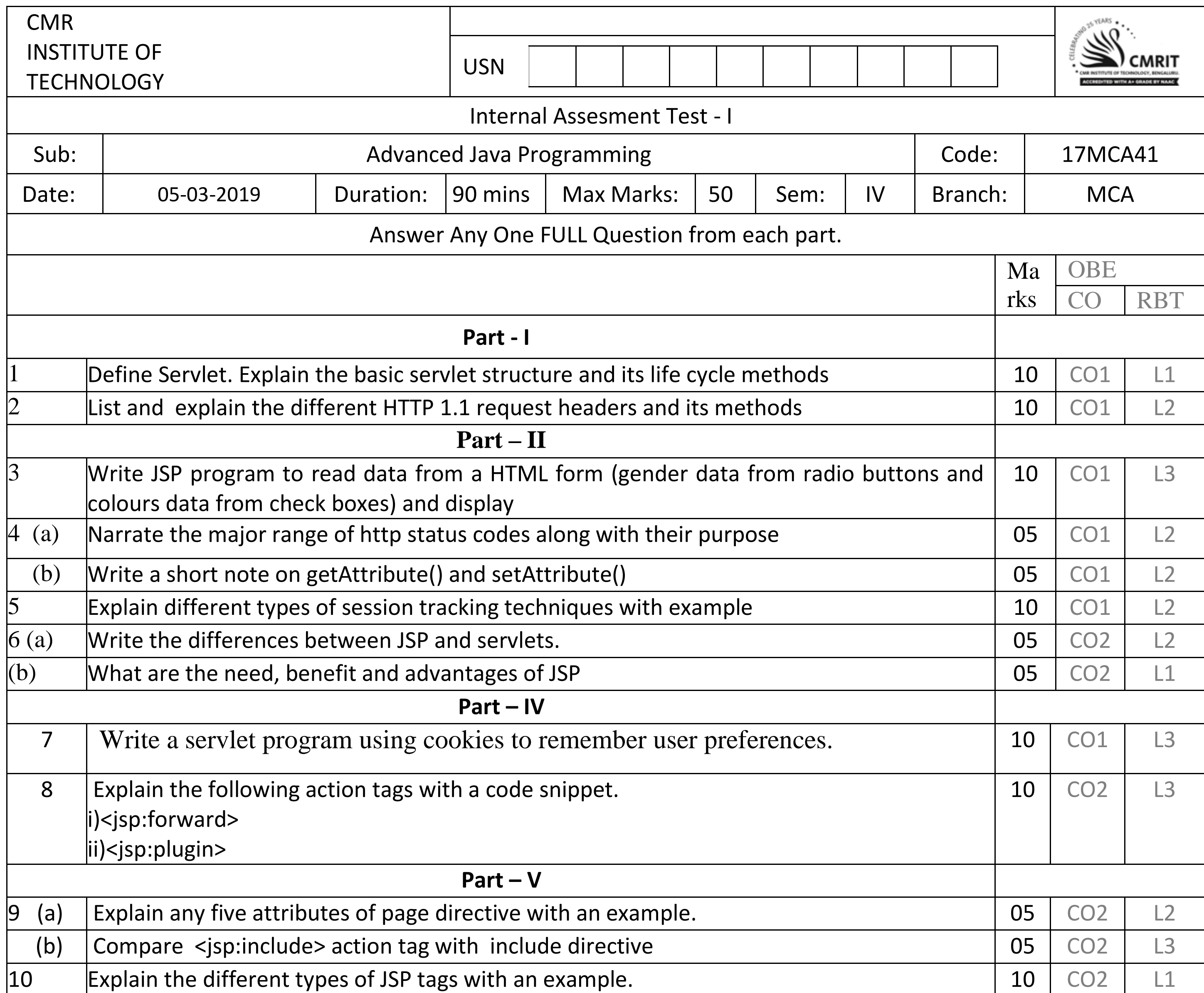


CMR INSTITUTE OF TECHNOLOGY	USN									

Internal Assesment Test - I

Sub:	Advanced Java Programming							Code:	17MCA41
Date:	05-03-2019	Duration:	90 mins	Max Marks:	50	Sem:	IV	Branch:	MCA

Answer Any One FULL Question from each part.

		Marks	OBE	
			CO	RBT
	Part - I			
1	Define Servlet. Explain the basic servlet structure and its life cycle methods	10	CO1	L1
2	List and explain the different HTTP 1.1 request headers and its methods	10	CO1	L2
	Part – II			
3	Write JSP program to read data from a HTML form (gender data from radio buttons and colours data from check boxes) and display	10	CO1	L3
4 (a)	Narrate the major range of http status codes along with their purpose	05	CO1	L2
(b)	Write a short note on getAttribute() and setAttribute()	05	CO1	L2
5	Explain different types of session tracking techniques with example	10	CO1	L2
6 (a)	Write the differences between JSP and servlets.	05	CO2	L2
(b)	What are the need, benefit and advantages of JSP	05	CO2	L1
	Part – IV			
7	Write a servlet program using cookies to remember user preferences.	10	CO1	L3
8	Explain the following action tags with a code snippet. i)<jsp:forward> ii)<jsp:plugin>	10	CO2	L3
	Part – V			
9 (a)	Explain any five attributes of page directive with an example.	05	CO2	L2
(b)	Compare <jsp:include> action tag with include directive	05	CO2	L3
10	Explain the different types of JSP tags with an example.	10	CO2	L1

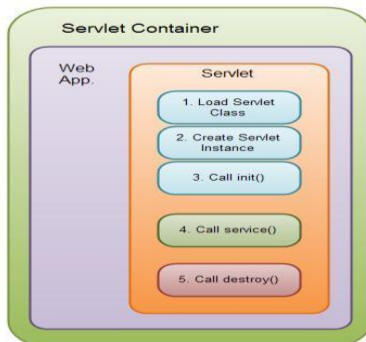
Internal Assesment Test – I

Advanced Java Programming – Answer Key

Date: 05.03.2019 **Code: 17MCA41**

1. Define Servlet. Explain the basic servlet structure and its life cycle methods

Java Servlets are programs that run on a Web or Application server
 Act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.
 Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.
 Servlets are server side components that provide a powerful mechanism for developing web applications.
A servlet life cycle can be defined as the entire process from its creation till the destruction. The Following are the paths followed by a servlet



 The servlet is initialized by calling the init () method.
 The servlet calls service() method to process a client's request.
 The servlet is terminated by calling the destroy() method.
 Finally, servlet is garbage collected by the garbage collector of the JVM.
Now let us discuss the life cycle methods in details.

The init() method :
 The init method is designed to be called only once.
 It is called when the servlet is first created, and not called again for each user request. So, it is used for one-time initializations, just as with the init method of applets.
 The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.
 The init() method simply creates or loads some data that will be used throughout the life of the servlet.
The init method definition looks like this:

```
public void init() throws ServletException {
// Initialization code...
}
```

The service() method :

 The service() method is the main method to perform the actual task.

 The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.
 Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.
Signature of service method:

```
public void service(ServletRequest request, ServletResponse response)
throws ServletException, IOException
{
}
```

 The service () method is called by the container and service method invokes doGe, doPost, doPut, doDelete, etc.methods as appropriate.
 So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.
 The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
// Servlet code
}
```

The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
// Servlet code
}
```

The destroy() method :

 The destroy() method is called only once at the end of the life cycle of a servlet.
 This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.
 After the destroy() method is called, the servlet object is marked for garbage collection.
The destroy method definition looks like this:

```
public void destroy() {
// Finalization code...
}
```

2. List and explain the different HTTP 1.1 request headers and its methods

When a browser requests for a web page, it sends lot of information to the web server which can not be read directly because this information travel as a part of header of HTTP request. You can check HTTP Protocol for more information on this.

Header	Description
Accept	This header specifies the MIME types that the browser or other clients can handle. Values of **image/png** or**image/jpeg** are the two most common possibilities.
Accept-Charset	This header specifies the character sets the browser can use to display the information. For example ISO-8859-1.
Accept-Encoding	This header specifies the types of encodings that the browser knows how to handle. Values of **gzip** or**compress** are the two most common possibilities.
Accept-Language	This header specifies the client's preferred languages in case the servlet can produce results in more than one language. For example en, en-us, ru, etc.
Authorization	This header is used by clients to identify themselves when accessing password-protected Web pages.
Connection	This header indicates whether the client can handle persistent HTTP connections. Persistent connections permit the client or other browser to retrieve multiple files with a single request. A value of **Keep-Alive** means that persistent connections should be used
Content-Length	This header is applicable only to POST requests and gives the size of the POST data in bytes.
Cookie	This header returns cookies to servers that previously sent them to the browser.
Host	This header specifies the host and port as given in the original URL.
If-Modified-Since	This header indicates that the client wants the page only if it has been
	changed after the specified date. The server sends a code, 304 which
	means Not Modifiedheader if no newer result is available.
If-Unmodified-	This header is the reverse of If-Modified-Since; it specifies that the
Since	operation should succeed only if the document is older than the

	specified date.
Referer	This header indicates the URL of the referring Web page. For example, if you are at Web page 1 and click on a link to Web page 2, the URL of Web page 1 is included in the Referer header when the browser requests Web page 2.
User-Agent	This header identifies the browser or other client making the request and can be used to return different content to different types of browsers.

List the methods to Read the request Headers

Reading headers is straightforward; just call the getHeader method of HttpServletRequest, which returns a String if the specified header was supplied on this request, null otherwise. Header names are not case sensitive. So, for example, request.getHeader("Connection") and Request.getHeader("connection") are interchangeable.

Although getHeader is the general-purpose way to read incoming headers, there are a couple of headers that are so commonly used that they have special access methods in HttpServletRequest.

getCookies : The getCookies method returns the contents of the Cookie header, parsed and stored in an array of Cookie objects. This method is discussed more in Chapter 8 (Handling Cookies).

getAuthType and getRemoteUser : The getAuthType and getRemoteUser methods break the Authorization header into its component pieces.

getContentLength The getContentLength method returns the value of the Content-Length header (Int)

getContentType The getContentType method returns the value of the Content-Type header (as a String).

getDateHeader and getIntHeader The getDateHeader and getIntHeader methods read the specified header and then convert them to Date and int values, respectively.

getHeaderNames Rather than looking up one particular header, you can use the getHeaderNames method to get an Enumeration of all header names received on this particular request.

getHeaders In most cases, each header name appears only once in the request. Occasionally, however, a header can appear multiple times, with each occurrence listing a separate value. Accept-Language is one such example.

getMethod The getMethod method returns the main request method (normally GET or POST, but things like HEAD, PUT, and DELETE are possible).

getRequestURI The getRequestURI method returns the part of the URL that comes after the host and port but before the form data. For example, for a URL of

http://randomhost.com/servlet/search.BookSearch, getRequestURI would return /servlet/search.BookSearch.

getProtocol Lastly, the getProtocol method returns the third part of the request line, which is generally HTTP/1.0 or HTTP/1.1. Servlets should usually check getProtocol before specifying response headers (Chapter 7) that are specific to HTTP 1.1.

3. Write JSP program to read data from a HTML form (gender data from radio buttons and colours data from check boxes) and display

```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" %>
<html>
<body>
<form name="frm" method="get" action="radioInput.jsp">
<table width="100%" border="0" cellspacing="0" cellpadding="0">
  <tr>
    <td width="22%"> </td>
    <td width="78%"> </td>
    </tr>
  <tr>
    <td>Male<input type="radio" name="gender" value="Male"></td>
    <td>Female <input type="radio" name="gender" value="Female"></td>
  </tr>

  <tr>
    <td>RED <input type="checkbox" name="colour" value="red"></td>
    <td>GREEN <input type="checkbox" name="colour" value="green"></td>
    <td>BLUE <input type="checkbox" name="colour" value="blue"></td>

  </tr>
  <tr>
 <tr>
    <td> </td>
    <td><input type="submit" name="submit" value="Submit"></td>
    </tr>
  <tr>
    <td> </td>
    <td> </td>
    </tr>
</table>
</form>
</body>
```

```
</html>
```

radioInput.jsp

```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" %>
<%
String radioInJSP=request.getParameter("gender");
%>
%>
<html>
<body>
Value of radio button box in JSP : <%=radioInJSP%>
<%
String colours[]= request.getParameterValues("colour");
if(colours != null)
{
%>
<h4>I likes colour/s mostly</h4>
<ul>
<%
for(int i=0; i<colours.length; i++)
{
%>
<li><%=colours[i]%></li>
<%
}
%>

</body>
</html>
```

4.a) Narrate the major range of http status codes along with their purpose

The status line consists of the

- HTTP version (HTTP/1.1 in the example above),
- a status code (an integer; 200 in the above example),
- a very short message corresponding to the status code (OK in the example).

- **100-199**
  Codes in the 100s are informational, indicating that the client should respond with some other action.
- **200-299**
  Values in the 200s signify that the request was successful.
- **300-399**
  Values in the 300s are used for files that have moved and usually include a `Location` header indicating the new address.
- **400-499**
  Values in the 400s indicate an error by the client.
- **500-599**
  Codes in the 500s signify an error by the server.

200 (OK) – Everything is fine; document follows. – Default for servlets.
204 (No Content) - – Browser should keep displaying previous document.
301 (Moved Permanently)
– Requested document permanently moved elsewhere (indicated in Location header).
– Browsers go to new location automatically.
401 (Unauthorized)
– Browser tried to access password-protected page without proper Authorization header.
404 (Not Found)
– No such page. Servlets should use sendError to set this.
– Problem: Internet Explorer and small (< 512 bytes) error pages. IE ignores small error page by default.

4.b) Write a short note on getAttribute() and setAttribute()

a) **getAttribute() is the method of HTTPSession which** Returns the String object specified in the parameter, from the session object. If no object is found for the specified attribute, then the getAttribute() method returns null.

public Object getAttribute(String name):

TO get the value from session we use the getAttribute() method of HttpSession interface. Here we are fetching the attribute values using attribute names.

```
String userName = (String) session.getAttribute("uName");
String userEmailId = (String) session.getAttribute("uemailId");
String userAge = (String) session.getAttribute("uAge");
```

b) setAttribute() is the method of HTTPSession which binds the object with a name and stores the name/value pair as an attribute of the HttpSession object. If an attribute already exists, then this method replaces the existing attributes.

public void setAttribute(String name, Object value)

example

```
session.setAttribute("uName", "ChaitanyaSingh");
session.setAttribute("uemailId", "myemailid@gmail.com");
session.setAttribute("uAge", "30");
```

This First parameter is the attribute name and second is the attribute value. For e.g. uName is the attribute name and ChaitanyaSingh is the attribute value in the code above.

5.Explain different types of session tracking techniques with example

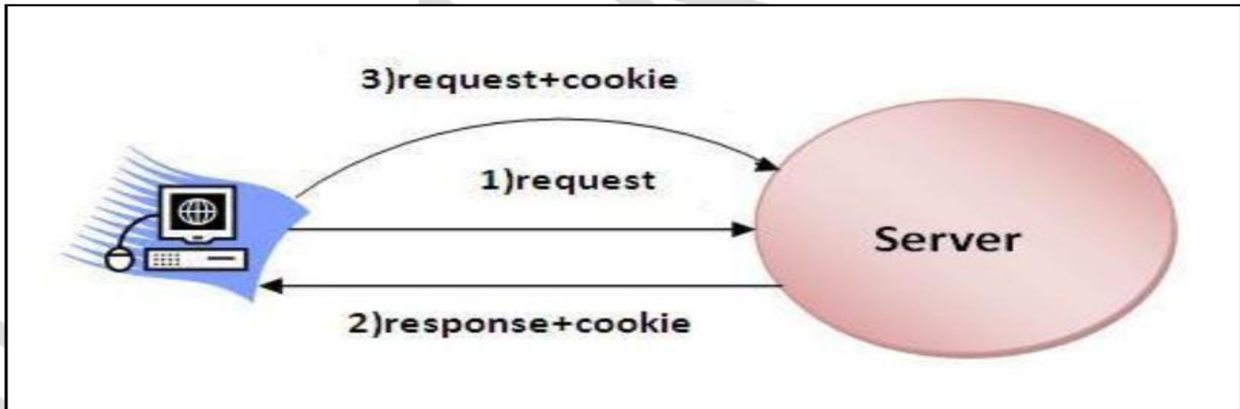
• Session tracking is the capability of a server to maintain the current state of a single client's sequential requests.
• Session simply means a particular interval of time.
• Session Tracking is a way to maintain state of a user.
• The HTTP protocol used by Web servers is stateless.
• Each time user requests to the server, server treats the request as the new request.
• So we need to maintain the state of a user to recognize to particular user.
• This type of stateless transaction is not a problem unless you need to know the sequence of actions a client has performed while at your site.
• For example, an online video store must be able to determine each visitor's sequence of actions.

Session Tracking Techniques

There are four techniques used in Session tracking:
• Cookies
• Hidden Form Field
• URL Rewriting
• HttpSession

1. Cookies

• A cookie is a small piece of information that is persisted between the multiple client requests.
• A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.



javax.servlet.http.Cookie → class provides the functionality of using cookies.

Constructor of Cookie class

Cookie(String name, String value): Constructs a cookie with a specified name and value.

Commonly used methods of Cookie class

There are given some commonly used methods of the Cookie class.

1. **public void setMaxAge(int expiry):** Sets the maximum age of the cookie in seconds.
2. **public String getName():** Returns the name of the cookie. The name cannot be changed after creation.
3. **public String getValue():** Returns the value of the cookie.

Other methods required for using Cookies

For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:

1. **public void addCookie(Cookie ck):** method of HttpServletResponse interface is used to add cookie in response object.
2. **public Cookie[] getCookies():** method of HttpServletRequest interface is used to return all the cookies from the browser.

Advantage of Cookies:

1. Identifying a User During an E-commerce Session
2. Avoiding Username and Password
3. Customizing a Site
4. Focusing Advertising
5. Simplest technique of maintaining the state.
6. Cookies are maintained at client side.

Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

2) Hidden Form Field

A hidden (invisible) textfield is used for maintaining the state of an user.

In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.

Let's see the code to store value in hidden field.

```
1. <input type="hidden" name="uname" value="Vimal Jaiswal">
```

Here, uname is the hidden field name and Vimal Jaiswal is the hidden field value.

Real application of hidden form field

It is widely used in comment form of a website. In such case, we store page id or page name in the hidden field so that each page can be uniquely identified.

Advantage of Hidden Form Field

1. It will always work whether cookie is disabled or not.

Disadvantage of Hidden Form Field:

1. It is maintained at server side.
2. Extra form submission is required on each pages.
3. Only textual information can be used.

3.URL Rewriting

The client appends some extra data on the end of each URL that identifies the session, and the server associates that identifier with data it has stored about that session.

For example, with `http://host/path/file.html;jsessionid=1234`, the session information is attached as `jsessionid=1234`.

url?name1=value1&name2=value2&??

This is also an excellent solution, and even has the advantage that it works when browsers don't support cookies or when the user has disabled them.

Advantage of URL Rewriting

1. It will always work whether cookie is disabled or not (browser independent).
2. Extra form submission is not required on each pages.

Disadvantage of URL Rewriting

1. It will work only with links.
2. It can send Only textual information.

4) HTTP Session

In HTTP Session, container creates a session id for each user.The container uses this id to identify the particular user.An object of HttpSession can be used to perform two tasks:

1. bind objects
2. view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.

How to get the HttpSession object ?

The HttpServletRequest interface provides two methods to get the object of HttpSession:

1. **public HttpSession getSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **public HttpSession getSession(boolean create):**Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

Commonly used methods of HttpSession interface

1. **public String getId():**Returns a string containing the unique identifier value.
2. **public long getCreationTime():**Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. **public long getLastAccessedTime():**Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
4. **public void invalidate():**Invalidates this session then unbinds any objects bound to it.

6.a) Write the differences between JSP and servlets.

JSP	Servlets
JSP is a webpage scripting language that can generate dynamic content.	Servlets are Java programs that are already compiled which also creates dynamic web content.
JSP run slower compared to Servlet as it takes compilation time to convert into Java Servlets.	Servlets run faster compared to JSP.

It's easier to code in JSP than in Java Servlets.	Its little much code to write here.
In MVC, jsp act as a view.	In MVC, servlet act as a controller.
JSP are generally preferred when there is not much processing of data required.	servlets are best for use when there is more processing and manipulation involved.
The advantage of JSP programming over servlets is that we can build custom tags which can directly call Java beans.	There is no such custom tag facility in servlets.
We can achieve functionality of JSP at client side by running JavaScript at client side.	There are no such methods for servlets.

6.b) What are the need, benefit and advantages of JSP

Need:

• **It is hard to write and maintain the HTML**. Using print statements to generate HTML? Hardly convenient: you have to use parentheses and semicolons, have to insert backslashes in front of embedded double quotes, and have to use string concatenation to put the content together. Besides, it simply does not look like HTML, so it is harder to visualize. Compare this servlet style with Listing 10.1 where you hardly even notice that the page is not an ordinary HTML document.

• **You cannot use standard HTML tools.** All those great Web-site development tools you have are of little use when you are writing Java code.

• **The HTML is inaccessible to non-Java developers**. If the HTML is embedded within Java code, a Web development expert who does not know the Java programming language will have trouble reviewing and changing the HTML.

Benefits:

JSP provides the following benefits over servlets alone:

• **It is easier to write and maintain the HTML.** Your static code is ordinary HTML: no extra backslashes, no double quotes, and no lurking Java syntax.

• **You can use standard Web-site development tools**. For example, we use Macromedia Dreamweaver for most of the JSP pages in the book. Even HTML tools that know nothing about JSP can be used because they simply ignore the JSP tags.

• **You can divide up your development team.** The Java programmers can work on the dynamic code. The Web developers can concentrate on the presentation layer. On large projects, this division is very important. Depending on the size of your team and the complexity of your project, you can enforce a weaker or stronger separation between the static HTML and the dynamic content.

Advantages:

Following table lists out the other advantages of using JSP over other technologies −
vs. Active Server Pages (ASP)
The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.
vs. Pure Servlets
It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.
vs. Server-Side Includes (SSI)
SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.
vs. JavaScript
JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc.
vs. Static HTML
Regular HTML, of course, cannot contain dynamic information.

7.Write a servlet program using cookies to remember user preferences.

```
import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/store")

public class store extends HttpServlet {
	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

		response.setContentType("text/html;charset=UTF-8");
		PrintWriter out=response.getWriter();
		try
		{
			String s1=request.getParameter("color");
			if (s1.equals("RED")||s1.equals("BLUE")||s1.equals("GREEN"))
			{
				Cookie ck1=new Cookie("color",s1);
				response.addCookie(ck1);
				out.println("<html>");
				out.println("<body>");
				out.println("You selected: "+s1);
				out.println("<form action='retrieve' method='post'>");
				out.println("<input type='Submit' value='submit'/>");
				out.println("</form>");
				out.println("</body>");
				out.println("</html>");
					}
```

```
        }
        finally
        {
            out.close();
        }
  }
}
```

retrieve.java

```
package j2ee.prg4;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/retrieve")
public class retrieve extends HttpServlet {
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

                response.setContentType("text/html;charset=UTF-8");
                PrintWriter out=response.getWriter();
                try
                {
                                Cookie ck[]=request.getCookies();
                                out.println("<html>");
                                out.println("<head>");
                                out.println("<title>servlet</title>");
                                out.println("</head>");
                                out.println("<body bgcolor="+ck[0].getValue()+">");
                                out.println("You selected color is: "+ck[0].getValue()+"</h1>");
                                out.println("</body>");
                                out.println("</html>");
                }
                finally
                {
                        out.close();
                }
        }

}
```

Index.jsp

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
```

```
<title>Insert title here</title>
</head>
<body>
<!-- send the form data to the url store and the post method is used  -->
<form action="store" method="post">
<!-- Display the Radio button with three option  -->
RED:<input type="radio" name="color" value="RED"/><br>
GREEN:<input type="radio" name="color" value="GREEN"/><br>
BLUE:<input type="radio" name="color" value="BLUE"/><br>
<input type="submit" value="submit"/>
</form>
</body>
</html>
```

8. Explain the following action tags with a code snippet.
i)<jsp:forward>
ii)<jsp:plugin>

i)<jsp:forward>

The jsp:forward action tag is used to forward the request to another resource it may be jsp, html or another resource.

Syn:

```
<jsp:forward  page="name of resource" />
```

Ex:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<!-- send the form data to login.jsp and the get method is used  -->
<form method="get" action="login.jsp">
UserName :  <input type="text" name ="name"><br>
Password :  <input type="password" name ="pass"><br>
<input type="Submit" value ="Submit"/><br>
</form>
</body>
</html>
```

login.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
//Getting the input name from the html form and storing in String 'uname'
String uname = request.getParameter("name");
//Getting the input pass from the html form and storing in String 'upass'
String upass = request.getParameter("pass");
if(uname.equals("admin") && upass.equals("admin"))
{
 %>
        <jsp:forward page="main.jsp"></jsp:forward>

<%
}
%>
</body>
</html>
```

(ii) <jsp:plugin> Action

 The plugin action is used to insert Java components into a JSP page. It determines the type of browser and inserts the <object> or <embed> tags as needed.
 If the needed plugin is not present, it downloads the plugin and then executes the Java component. The Java component can be either an Applet or a JavaBean.
 The plugin action has several attributes that correspond to common HTML tags used to
format Java components.
 The <param> element can also be used to send parameters to the Applet or Bean.

Following is the typical syntax of using plugin action:

The simplest way to use jsp:plugin is to supply four attributes: type, code, width, and height.

```
<APPLET CODE="MyApplet.class" WIDTH=475 HEIGHT=350>
</APPLET>

<jsp:plugin type="applet" code="MyApplet.class" width="475" height="350">

</jsp:plugin>
```

- Type - For applets, this attribute should have a value of applet. The Java Plug-In also permits

embed JavaBeans elements in Web pages. Use a value of bean in such a case.

- Code - This attribute is used identically to the CODE attribute of APPLET, specifying the toplevel

applet class file that extends Applet or JApplet.

- Width - This attribute is used identically to the WIDTH attribute of APPLET, specifying the

width in pixels to be reserved for the applet .

- Height - This attribute is used identically to the HEIGHT attribute of APPLET, specifying the

height in pixels to be reserved for the applet

• codebase - This attribute is used identically to the CODEBASE attribute of APPLET, specifying the

base directory for the applets. The code attribute is interpreted relative to this directory. As with the APPLET element, if you omit this attribute, the directory of the current page is used as the default. In the case of JSP, this default location is the directory where the original JSP file resided, not the system-specific location of the servlet that results from the JSP file.

9. a) Explain any five attributes of page directive with an example.

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

Following is the basic syntax of page directive:

```
<%@ page attribute="value" %>
```

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

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

1. The import Attribute:

The import attribute serves the same function as, and behaves like, the Java import statement. The value for the import option is the name of the package you want to import.

To import java.sql.*, use the following page directive:

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

To import multiple packages you can specify them separated by comma as follows:

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

By default, a container automatically imports java.lang.*, javax.servlet.*, javax.servlet.jsp.*, and javax.servlet.http.*.

Example code

```
<html>
<body>

<%@ page import="java.util.Date" %>
Today is: <%= new Date() %>

</body>
</html>
```

2. The session Attribute:

The session attribute indicates whether or not the JSP page uses HTTP sessions. A value of true means that the JSP page has access to a builtin **session** object and a value of false means that the JSP page cannot access the builtin session object.

Following directive allows the JSP page to use any of the builtin object session methods such as session.getCreationTime() or session.getLastAccessTime():

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

Example

```
<body>

<%@ page session="true" %>
 <%                                                          // a scriptlet
  String empName = request.getParameter("nameField");        // reading from client
  String empSalary = request.getParameter("salaryField");

  session.setAttribute("en", empName);              // setting with session object
  session.setAttribute("es", empSalary);
                                        // just a message
  out.println("<b>Employee values are set with session object. <br>");
  out.println("Can be accessed from a different program.</b>");
 %>

</body>
```

3. The buffer Attribute:

The **buffer** attribute specifies buffering characteristics for the server output response object.

You may code a value of "none" to specify no buffering so that all servlet output is immediately directed to the response object or you may code a maximum buffer size in kilobytes, which directs the servlet to write to the buffer before writing to the response object.

To direct the servlet to write output directly to the response output object, use the following:

```
<%@ page buffer="none" %>
```

Use the following to direct the servlet to write output to a buffer of size not less than 8 kilobytes:

```
<%@ page buffer="8kb" %>
```

4. The autoFlush Attribute:

The **autoFlush** attribute specifies whether buffered output should be flushed automatically when the buffer is filled, or whether an exception should be raised to indicate buffer overflow.
A value of true (default) indicates automatic buffer flushing and a value of false throws an exception.
The following directive causes the servlet to throw an exception when the servlet's output buffer is full:

```
<%@ page autoFlush="false" %>
```

This directive causes the servlet to flush the output buffer when full:

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

Usually, the buffer and autoFlush attributes are coded on a single page directive as follows:

```
<%@ page buffer="16kb" autoflush="true" %>
```

5. The contentType Attribute:

The contentType attribute sets the character encoding for the JSP page and for the generated response page. The default content type is text/html, which is the standard content type for HTML pages.
If you want to write out XML from your JSP, use the following page directive:

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

The following statement directs the browser to render the generated page as HTML:

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

The following directive sets the content type as a Microsoft Word document:

```
<%@ page contentType="application/msword" %>
```

9. b) Compare <jsp:include> action tag with include directive

Table 13.1 Differences Between `jsp:include` and the `include` Directive

	`jsp:include` Action	**`include` Directive**
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 (main page and included page each become a separate servlet)	One (included file is inserted into main page, then that page is translated into a servlet)

Table 13.1 Differences Between `jsp:include` and the `include` Directive *(continued)*

	`jsp:include` Action	`include` Directive
Can included page set response headers that affect the main page?	No	Yes
Can included page define fields or methods that main page uses?	No	Yes
Does main page need to be updated when included page changes?	No	Yes
What is the equivalent servlet code?	`include` method of `RequestDispatcher`	None

10. Explain the different types of JSP tags with an example.

1) Scriptlet Tag (<% … %>)

- A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.
- Embeds Java code in the JSP document that will be executed each time the JSP page is processed.
- Code is inserted in the service() method of the generated Servlet

Syntax two forms:

- **<% any java code %>**
- **<jsp:scriptlet> ... </jsp:scriptlet>. (XML form)**

- **Example**

```
<% if (Math.random() < 0.5)
{
%>
    Have a <B>nice</B> day!
<%
}
Else
{ %>
Have a <B>lousy</B> day!
<%
 }
%>
```

- **Representative result**

```
– if (Math.random() < 0.5) {
out.println("Have a <B>nice</B> day!");
} else {
out.println("Have a <B>lousy</B> day!");
}
```

2) Expression Tag: (<%= … %>)

 A JSP expression element contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file.

 Because the value of an expression is converted to a String, you can use an expression within a line of text, whether or not it is tagged with HTML, in a JSP file.

 The expression element can contain any expression that is valid according to the Java Language Specification but you cannot use a semicolon to end an expression.

Syntax two forms:
• **<%= expr %>**

• **<jsp:expression> expr </jsp:expression> (XML form)**

Example: 1

```
<html>
  <body>
   <%=  Integer.toString( 5 * 5 )  %>
</body>
</html>
```

Example: 2

```
<html> <body>
<p>
Today's date:
<%= (new java.util.Date()).toLocaleString()%>
</p>
</body> </html>
```

3) Declaration Tag (<%! … %>)

 A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.

 Code is inserted in the body of the servlet class, outside the service method.
o May declare instance variables.

o May declare (private) member functions.

Syntax two forms:
• **<%! declaration %>**

• **<jsp:declaration> declaration(s)</jsp:declaration>**

Example for declaration of Instance Variable:

```
<html>
<body>
<%! private int accessCount = 0; %>
<p> Accesses to page since server reboot:
<%= ++accessCount %> </p>
</body>
</html>
```

4) Directive Tag (<%@ … %>)

 Directives are used to convey special processing information about the page to the JSP container.

 The Directive tag commands the JSP virtual engine to perform a specific task, such as importing a Java package required by objects and methods.

Directive	**Description**
<%@ page ... %>	Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
<%@ include ... %>	Includes a file during the translation phase.
<%@ taglib ... %>	Declares a tag library, containing custom actions, used in the page

5) JSP Action

• **Description:**
Action that takes place when the page is requested
• **Example:**

<jsp:*blah*>...</jsp:*blah*>

6) JSP Expression Language Element
• **Description:**
Shorthand JSP expression
• **Example:**

${ *EL Expression* }

7) Custom Tag (Custom Action)
• **Description:**
Invocation of custom tag
• **Example:**
<prefix:name>
Body

</prefix:name>

8)HTML Text
– <H1>Blah</H1>
– Passed through to client. Really turned into servlet code that looks like
out.print("<H1>Blah</H1>");
9) HTML Comments
– <!-- Comment -->
– Same as other HTML: passed through to client
10) JSP Comments
– <%-- Comment --%>
– Not sent to client