MODULE- 9
JSP

Course Topics

- → Module 1
  - » Introduction to Java
- → Module 2
  - » Data Handling and Functions
- → Module 3
  - » Object Oriented Programming in Java
- → Module 4
  - » Packages and Multi-threading
- → Module 5
  - » Collections
- → Module 6
  - » XML
- → Module 7
  - » JDBC
- → Module 8
  - » Servlets
- → Module 9
  - » **JSP**
- → Module 10
  - » Hibernate
- → Module 11
  - » Spring
- → Module 12
  - » Spring, Ajax and Design Patterns
- → Module 13
  - » SOA
- → Module 14
  - » Web Services and Project

Objectives

At the end of this module, you will be able to

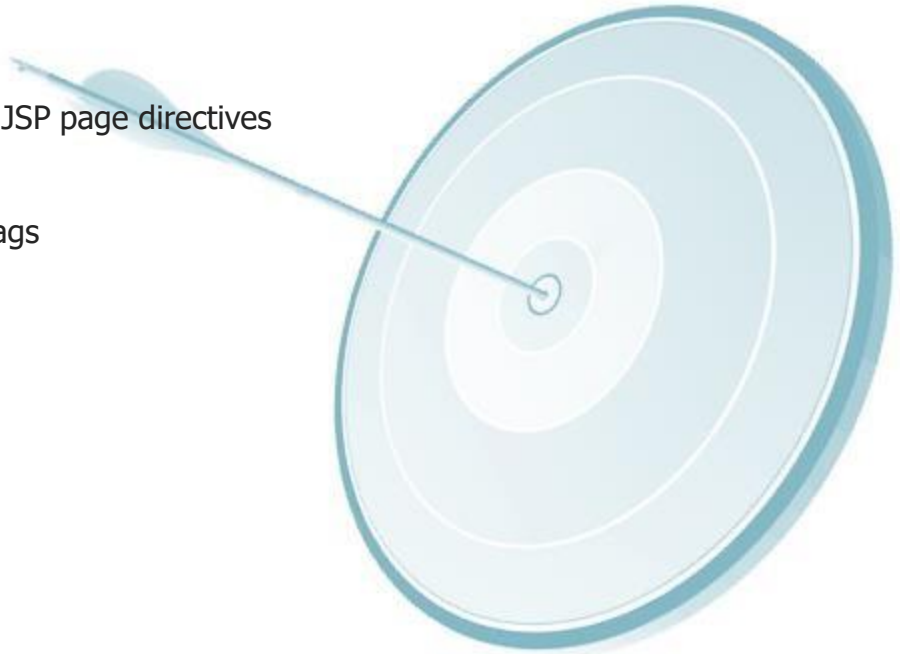
→ Understand JSP page

→ Learn to write a JSP page

→ Understand implicit objects available in JSP page and JSP page directives

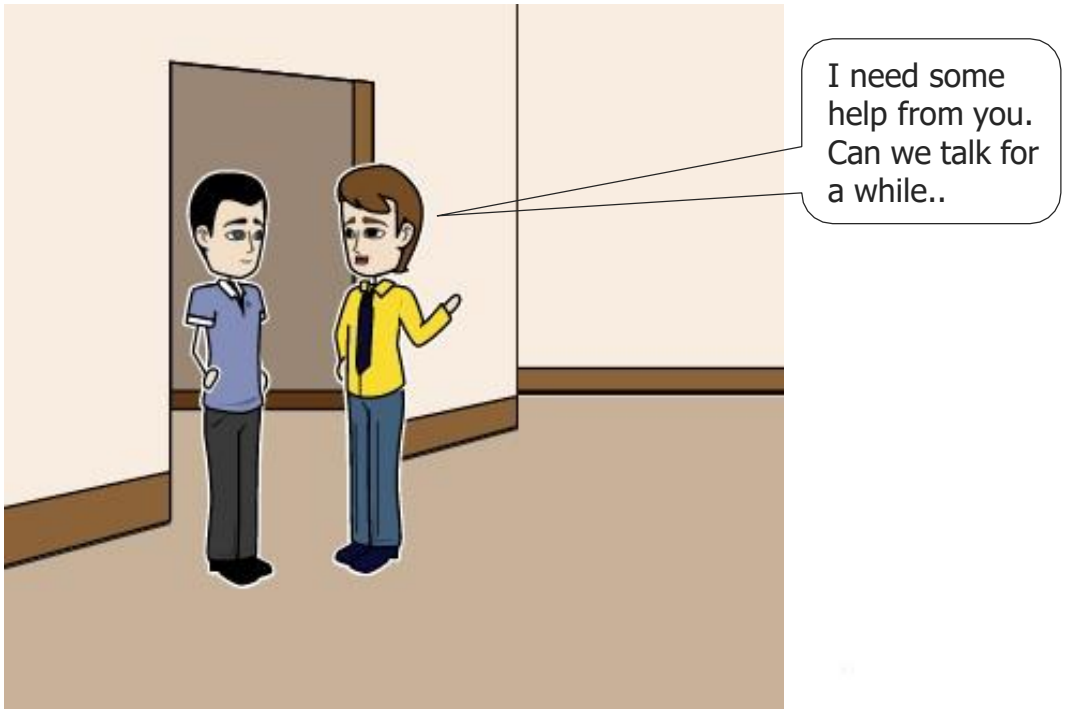
→ Learn about cookies and session handling

→ Learn about JSP standard actions, JSTL and custom tags

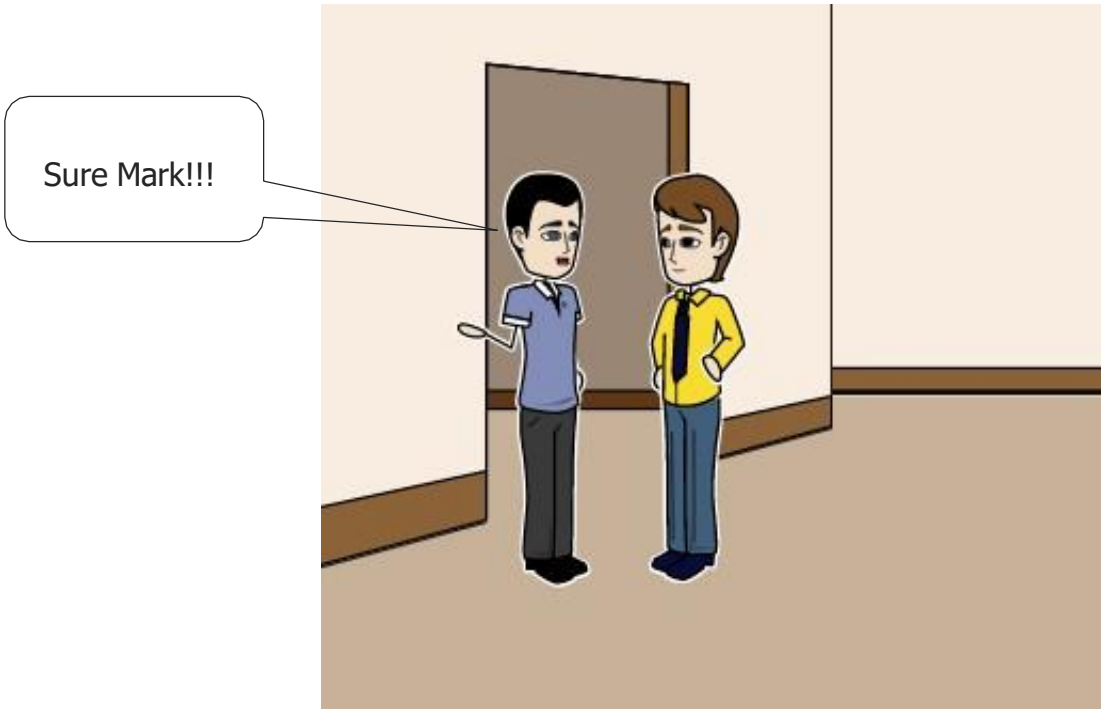


Mark Greets John

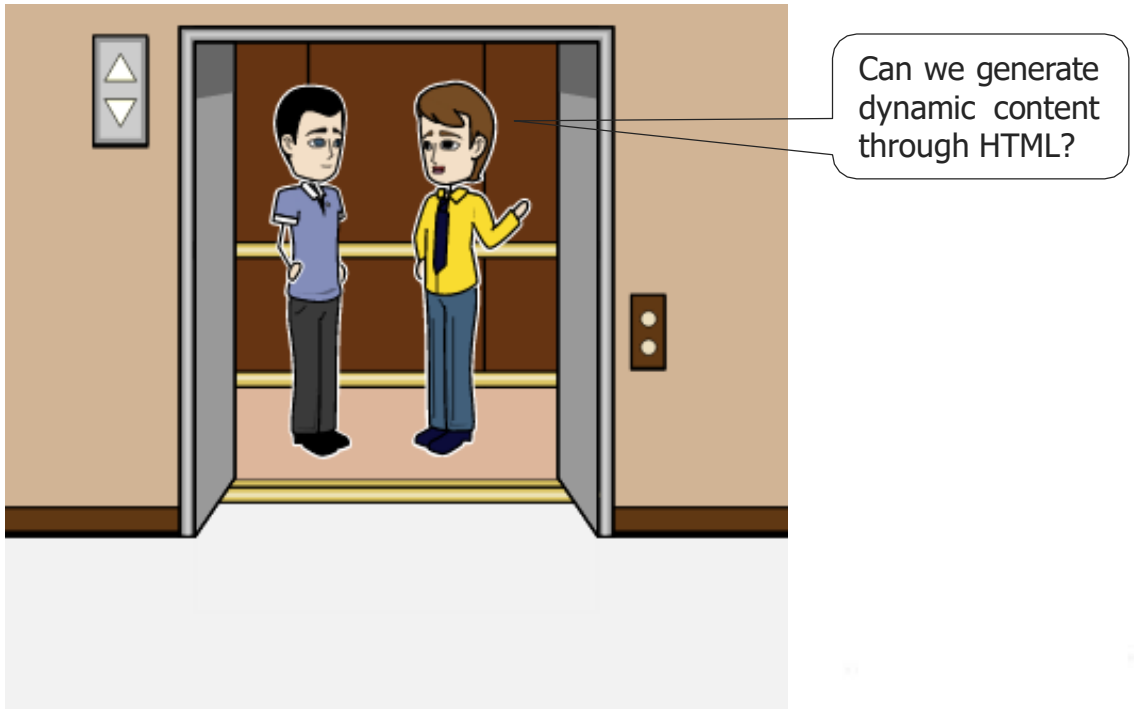
Mark has a doubt



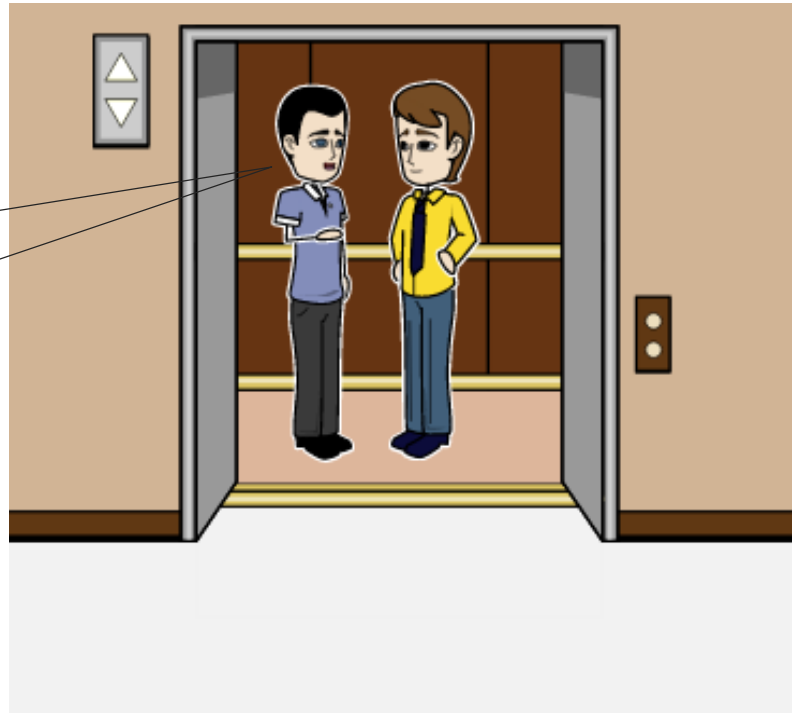
John helps Mark



John helps Mark



John helps Mark



Lets Learn More….

What is JSP?

→ JSP Stands for Java Server Pages.

→ JSP will have the extension of .jsp.

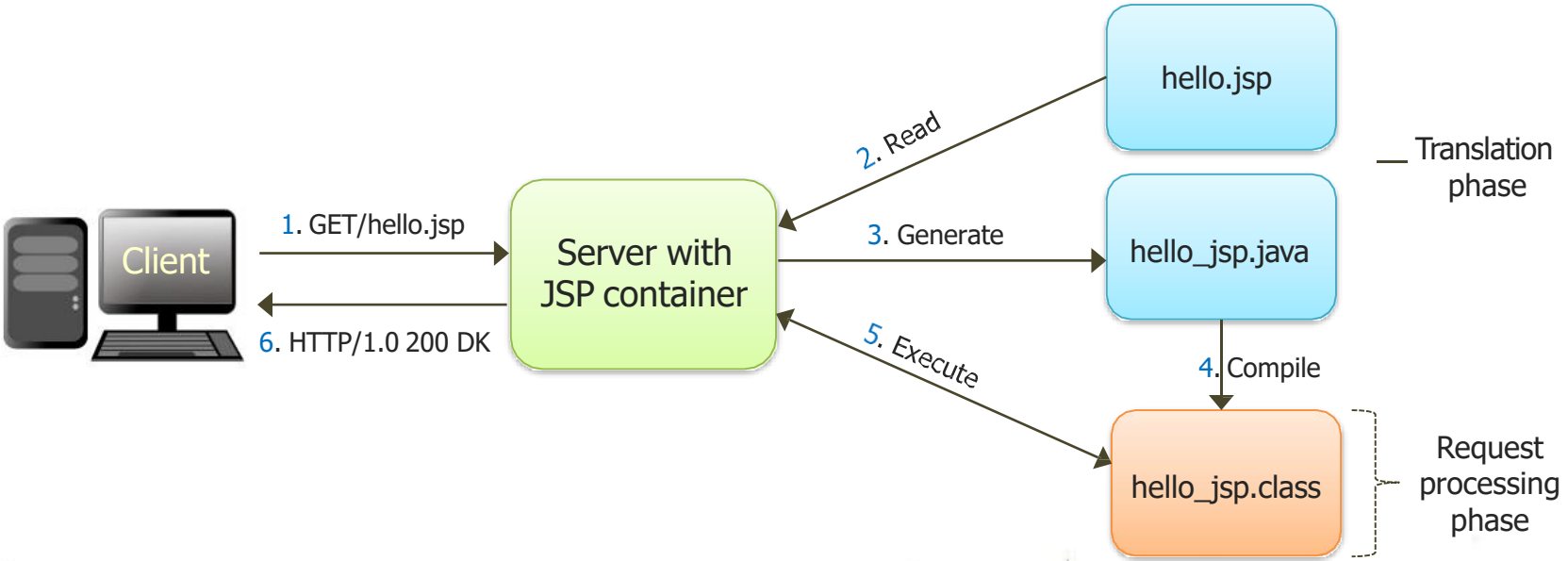
→ Like servlet, jsp file has to be executed on web server or app server.

→ HTML + Java is jsp. That means a JSP file will have both HTML and Java code.

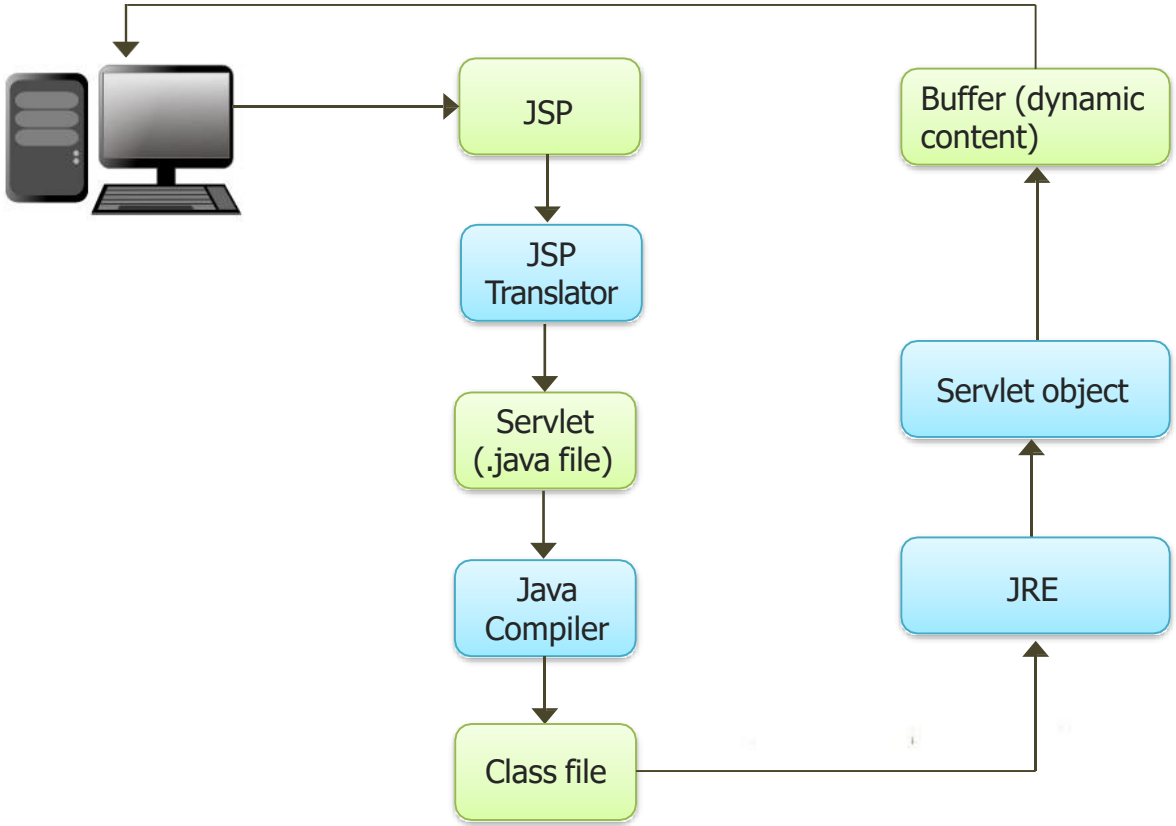
→ JSP is used for UI/front end/display purpose.

JSP Execution Procedure

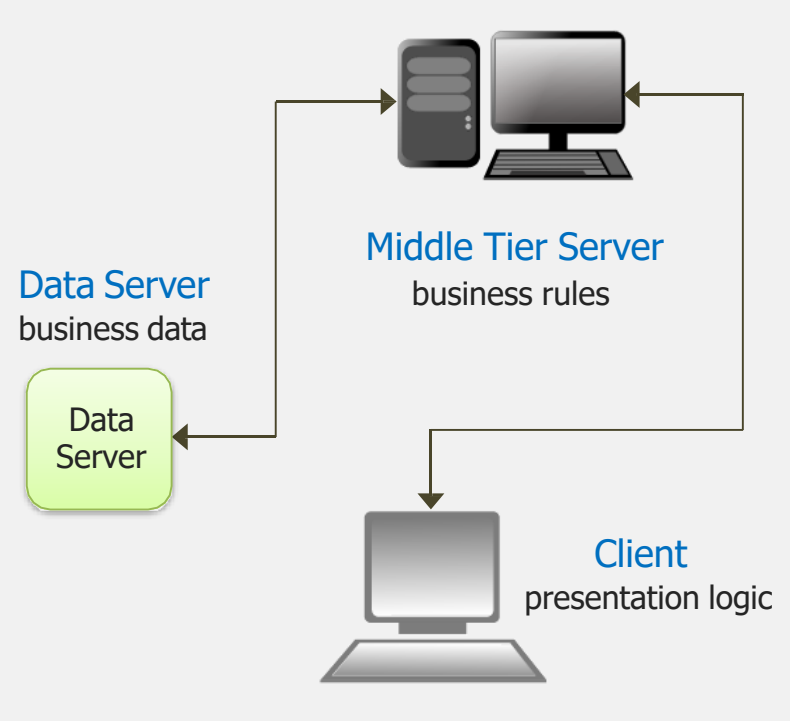
A JSP will be converted to a servlet file. The servlet file is compiled and generates a .class file. This .class file is loaded into memory and executed by JVM as usual.



JSP Execution Procedure (contd.)



Advantages of JSP over HTML



User need not write HTML and Java code separately.

JSP can be used for both front end and for writing business logic/middle later.

JSP is very easy to write when compared to Servlets.

JSP is dynamic compilation, which means when a JSP is modified, it need not be compiled and restarted in the web server. After the modification of JSP, refresh the browser, changes will be reflected.

In many applications only JSP is used for first two tiers of 3 tier architecture, which is also called 2 tier architecture. Going forward in frameworks like struts and spring, MVC is used which is same as 3 tier architecture. MVC stands for Model View controller.

Model is data. View is for display and controller – is a centralized controller which says what the application has to do.

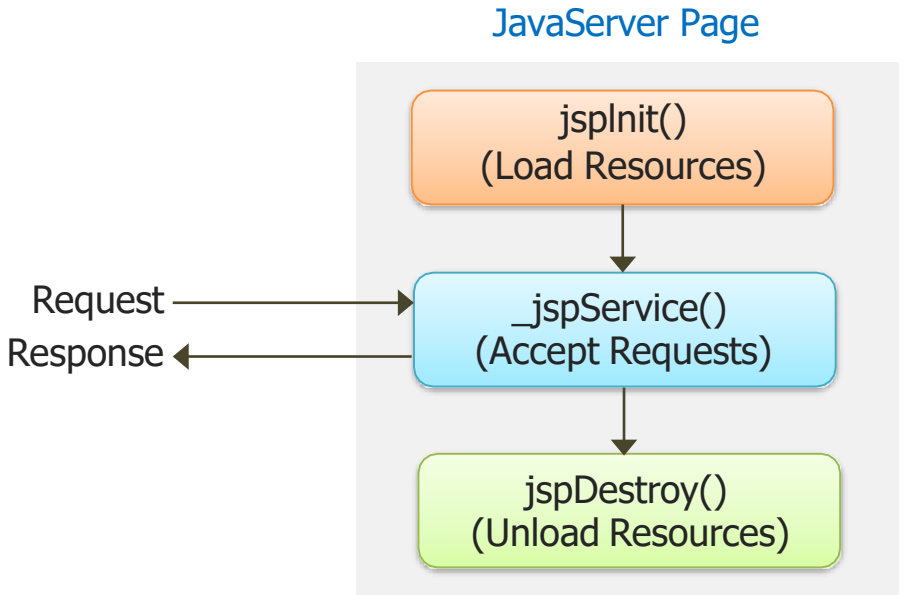
Annie’s Question

What is 3 tier architecture and why do we need it?

Annie’s Answer

In 3 tier architecture we have Presentation Tier which represents information to the user, Logic Tier here we make logical decisions and perform basic computation. It is the Logic tier which transfers data between Presentation Tier and Data Tier where information is stored. 3 tier architecture increases performance, scalability, flexibility and maintainability.

JSP Life-Cycle



Like in servlet init(), service() and destroy(), jsp also has jspinit(), _jspService(), jspdestroy().

Same as servlet, when jspInit() is called when jsp is loaded in memory. _jspService() is called when the client requests and jspDestroy() is invoked when jsp is removed from memory.

JSP Files Execution Procedure

All the jsp files will be written under WebContent directory of Eclipse.

→ In eclipse, for executing a jsp, select the jsp and right click on it and click on "Run As → Run on Server". Jsp file will be executed by opening the default browser.

→ User can select the internal browser for displaying the jsp by selecting "window menu → Web Browser → Internal Browser". The advantage of selecting internal browser is, all the time we need not shuffle between external browser and the program if there are any modifications in the program.

→ Once the complete code is developed, then for final testing external browser can be used to see the result.

JSP Tags

There are three tags in jsp. They are

1. Scriptlet → For writing Java code.
2. Declarative → For declaring attributes and methods.
3. Expression → For displaying data.

Scriptlet tag

Scriptlet tag is represented by <% %>

Java code should be embedded in this scriptlet tag.

```
<html>
<body>
<% int a=20,b=30,c;
      c=a+b;
      out.println( "Result is: "+c);
%>
</body>
</html>
```

Here Java code is embedded inside the scriptlet tag. When this code is executed 50 is displayed on the browser.

Declaration Tag

Declaration tag is used to define attributes and methods in a jsp file.

Declaration tag is represented by <%! %>

One exclamation symbol has to be added in scriptlet tag.

```
<html>
<body>
<%!
          int x = 20;
          public int  add (int a, int b)
            { return (a + b); }
%>
<%  out.println("Value of x is : "+x);
       out.println( "After adding : "+add(30,30));
%>
</body>
</html>
```

Example of declaration tag is:

This code will display value of x is 20 and after adding: 60 on browser.

Expression Tag

→ Expression tag is represented by <%= %>

→ Expression tag is used to display the values.

→ It is equivalent to <% out.println (“”); %>

```
<%! int x = 20; %>
        <%= ("Value of x is : " + x) %>
```

This will print Value of x. The value of x is 20.

Implicit Objects

Implicit objects are by default available in jsp file. There are 9 implicit objects:

out → Used to print.

request → Used to get the client request information.

response → Used to send the client response.

exception → Used for exception handling.

config → Used to read configuration param of web.xml

application → Used to read configuration param for the whole application.

session → Use to set and read values in session.

pageContext → Can set the values of variable in a scope of page, request, session and application.

page → This is equivalent to object and just replaced with this object.

Implicit Object – out

In the previous examples, we have used out object to print the data.

```
out.println ("Value of is x : " + x);
```

It is same as getting PrintWriter object in servlets from response object and writing using this object to the client.

out object is implicitly available to all jsp files. So we don't have to write response.getWriter() as we do in case of servlets.

Implicit Object – request

Request object is used to read the client's input.

```
<%
String name=request.getParameter("name");
out.print("Hello "+name);
%>
```

Above code will retrieve the name parameter from the request scope. If there is no parameter named "name" in request scope you will get null.

Implicit Object – response

It is same as request.
request will get the client input and response object is used to send the response back to the client.

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

This will redirect the response object to google.com. By executing the above code, browser will display google.com web page.

Implicit Object – pageContext

In web, there are 4 types of scope for a variable. They are

→ page – Variable is only available in that JSP page.

→ request – Variable is available in entire request.

→ session – Variable is available in the entire session.

→ application – Application variables are available across all the sessions .

Implicit Object – pageContext

```
<html>
<body>
<% String name=request.getParameter("name");
pageContext.setAttribute("userID",name,PageContext.SESSION_SCOPE);
%>

<% String username=(String)pageContext.getAttribute("userID",PageContext.SESSION_SCOPE);
out.print("Welcome "+username);
%>
</body>
</html>
```

→ In the first section we are retrieving the value of name parameter that was previously set in request scope. Then we are setting that value in the session scope with the attribute name user ID. If there is no parameter named "name" in request scope you will get null.

→ In the next section we are retrieving the value of attribute user ID that we set in first section.

→ Since the user ID attribute is set in session scope, the value of user ID attribute can be obtained by any jsp file/servlet file in the session.

Cookies

Cookies are used to track the user information. Small piece of information about the client is written on the client's browser which is sent to server on subsequent request.

```
<%
// Create cookies for first and last names.
   Cookie firstName = new Cookie("first_name",request.getParameter("first_name"));
   Cookie lastName = new Cookie("last_name",request.getParameter("last_name"));

// Set expiry date after 24 Hrs for both the cookies.
   firstName.setMaxAge(60*60*24);
   lastName.setMaxAge(60*60*24);

// Add both the cookies in the response header.
   response.addCookie(firstName );
   response.addCookie(lastName );
%>
```

In above code we are creating two cookies first_name and last_name which have been set to value as given by request parameters. Then we are setting the time limit for which cookies must be stored on client's machine and in last two lines we are adding cookies on client's machine.

Accessing Cookies

```
<%
   Cookie cookie = null;
   Cookie[] cookies = null;

  // Get an array of Cookies associated with this domain

  cookies = request.getCookies();
  if( cookies != null ){
   out.println("<h2> Found Cookies Name and Value</h2>");
     for (int i = 0; i<cookies.length; i++){
            cookie = cookies[i];
            out.print("Name : " + cookie.getName( ) + ",  ");
            out.print("Value: " + cookie.getValue( )+" <br/>");
            }
  }else{
   out.println("<h2>No cookies founds</h2>");
   }
%>
```

In above code, first we are retrieving cookies and saving it in an array.

Then if there is even a single cookie, we iterate over it and prints its name and value.

Handling Sessions

Http is a stateless protocol, it means it does not have any way to figure out that the second request is made by the same person who made the first request. So how do we do that?

Two ways of handling sessions:

→ Cookies
→ URL Rewriting

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

This is the first page where we are asking user's name.

Handling Sessions

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<body>
<%

String name=request.getParameter("uname");
out.print("Welcome "+name);

session.setAttribute("user",name);
%>
</br>
<a href="session_ImplicitObject_Second.jsp">second jsp page</a>


</body>
</html>
```

Here we are retrieving the value of uname parameter and setting it as a session attribute user.

Handling Sessions

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<body>
<%

String name=(String)session.getAttribute("user");
out.print("Hello "+name);

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

Now we are retrieving the value of user session attribute and showing a customized hello message.

Implicit Object – Session

Session object can be directly used incase if the attribute needs to be set in the session scope.

It can be used as:

```
session.setAttribute("username","edureka");
```

In the above example, username attribute has been explicitly set to edureka in the session.

username for the session in any other file can be obtained by using

session.getAttribute("username");

Implicit Object – exception

→ With this object, exceptions can be handled in jsp.

→ In Java try..catch blocks are used to handle the exception.

→ Specify which jsp file to be executed when there is an exception by using errorPage.

```
<%@ page errorPage="exception_ImplicitObject_error.jsp" %>
```

→ exception implicit object is used to print the stacktrace in the errorPage.

JSP - Directives

JSP Directives help the web container to convert the directive into its corresponding servlet.

There are 3 types of directives:

(1) Page directive

(2) include directive

(3) Tag lib directive

Page Directive

Page directive can set the attributes for the current JSP page.

Page directive has many attributes. They are as follows:

buffer → Can set the buffer size of the page.

import → Imports the Java or package file. This is similar to import statement in Java.

contentType → Set the response content Type. It could be HTML/text/DOC etc.

extends → Specifies the superclass for the servlet when generated.

info → Used to give the information about the jsp page and it will be written into getServletInfo() when converted to servlet.

Page import Directive

This code will import java.util.Date and displays today's date.

```
<%@ page import="java.util.Date" %>
<html>
<body>
<%
Date date = new Date();
%>

<%= ("Current date is : " + date) %>
</body>
</html>
```

JSP include Directive

JSP include directive will include the specified resource at conversion time.

Html/jsp file can be included.

Syntax is:

```
<%@ include file="footer.html" %>
```

All the lines of file footer.html will be included where @include is added.

Annie’s Question

What is the advantage of include directive?

Annie’s Answer

Instead of repeating the same set of code, one file can be written and it can be included in many places. This is called code re-usability.

JSP taglib Directive

This taglib directive will be discussed during JSTL (JSP Standard Tag Library) in coming slides.

Expression Language (EL)

Expression language(EL) is introduced in JSP 2.0. The main purpose of EL is to simplify the process of accessing data from bean properties and from implicit objects.

Without EL

```
<%
String name=request.getParameter("name");
out.print("Hello "+name);
%>
```

With EL

```
${param.name}
```

Other great feature El provides is that it handles null gracefully.
So if in the above code if there is no parameter named "name" have been set then first code snippet will show null while El shows nothing.

JSP – Action tags

There are JSP action tags which performs the specific tasks.

Some of the tags are:

jsp:include → Includes the resource at run time not at compile time.

jsp:forward → Forwards the control to another page.

jsp:useBean → Uses bean class.

jsp:setProperty → Sets the property of the bean class.

jsp:getProperty → Retrieves the value of the property from the bean class.

JSP action tag – include

Include action tag can be included as:

```
<jsp:include page="dynamicContent.jsp"/>
```

This will include dynamicContent.jsp at run time.

Annie's Question

What is the difference between jsp include directive and jsp include action tag?
When do you use directive or action tag?

Annie’s Answer

Directive is at compile time and action tag is at run time. When static code is required to be included like HTML, use directive and when dynamic code is required like JSP, use action tag.

JSP action tag – forward

```
<jsp:forward page="forwardAction.jsp"/>
```

When this tag is executed control moves to forwardAction.jsp

JSP action tag – useBean

Bean Class

```
package co.edureka;
public class Bean {
String str;
public String getStr() {
return str;
}
public void setStr(String str) {
this.str = str;}}
```

JSP Page

```
<jsp:useBean id="useBean" class="co.edureka.Bean">
<jsp:setProperty name="useBean" property="str" value="Charan" />
</jsp:useBean>
<p> <jsp:getProperty name="useBean" property="str" /></p>
```

UseBean

Java bean is a Java class which has getter and setter methods to retrieve and set the values of the attributes.

In the example, bean class has str as the string attribute.

In the jsp file, useBean tag is used to access this bean class and setProperty() method is used to set the attribute str and getProperty is used to get the attribute value.

JSP and JDBC

JSP also can interact with JDBC as in Java.

JSP needs to include jdbc driver from corresponding database vendor.

All the steps of the JDBC in JSP is same to interact with the database.

Steps are:

→ Load the driver
→ Get the connection to the database
→ Create a statement
→ Execute the statement
→ Close the connection

JSP and JDBC

```
<%@ page import="java.sql.*" %>
<html>
<body>
<%
Class.forName("oracle.jdbc.driver.OracleDriver");
System.out.println("Driver loaded...");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","databaseusername","password");
System.out.println("Connected to the database");
Statement st=con.createStatement();
st.executeUpdate("insert into student values('42','Pavas')");
System.out.println("  row inserted");
con.close();
System.out.println("Connection closed...");
%>
</body>
</html>
```

JSTL

JSTL Stands for JSP Standard Tag Library.

This is similar to

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:forEach var="number" begin="10"  end="20">
<c:out value="${number}"> </c:out>
</c:forEach>
```

Instead of writing in a scriptlet, tags can be used to get the same functionality.

```
<%
        for(int number = 10; number <= 20; ++number)
            out.print (number);
%>
```

Custom tags

There are cases when JSTL is not enough, we want the JSP tag to do specific things to achieve our requirement. We can create custom tags like <my:headerTag>

Three files are required to create custom tags.

Java file → What to do when a tag is called.

Tld file → Tag library descriptor file. This file maps the tag name and the Java file to be executed. It is like web.xml which maps URL pattern to servlet class.

Jsp file → File which uses the tag.

Page Directive (contd.)

language → The language used in the jsp. Java/Java Script etc.

isELIgnored → Specifies when the expression language is ignored or not.

isThreadSafe → Specifies the jsp page is thread safe or not.

autoFlush → Automatically output buffer will be flushed.

session → Specifies whether this page uses HTTPSession object or not.

errorPage and isErrorPage → Used in exception handling in jsp.

LAB

QUESTIONS

Assignment

→ Develop a login screen which has user id and password fields and a submit button.

On clicking the submit button, the user's information should be sent to the server and authenticate the user's credentials by checking with the database table user_pass. If the valid user id and password is given then display the message "valid user" else display the invalid user id/password and include the login screen again.

Agenda of the Next Class

In the next class, you will be able to

→ Understand Hibernate and how to use hibernate in eclipse.

→ Create an application using Hibernate

→ Map tables using Hibernate

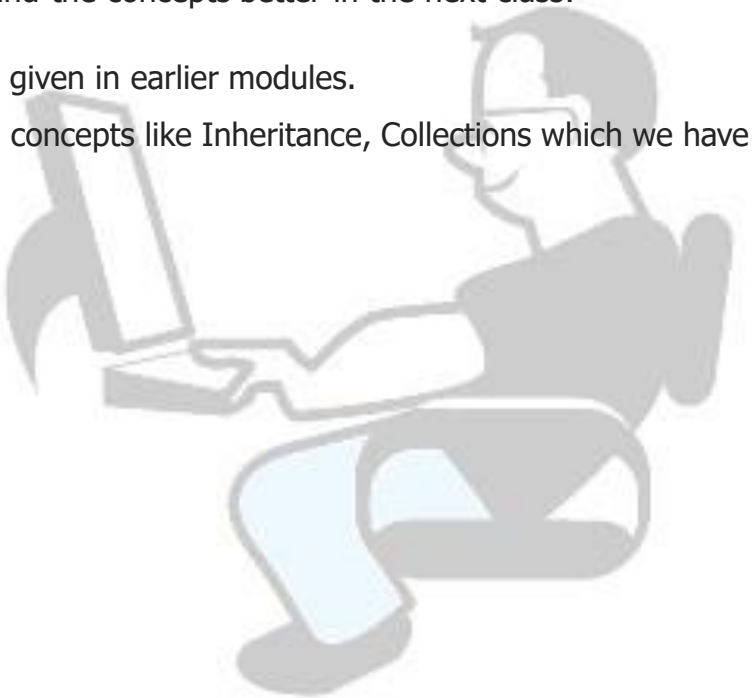
→ Understand inheritance in hibernate and hibernate cache mechanism.

Pre-work

Practice the following to understand the concepts better in the next class:

→ Practice all programs on JDBC given in earlier modules.

→ Repeat and revise all the Java concepts like Inheritance, Collections which we have covered in earlier classes.



Survey

Your feedback is important to us, be it a compliment, a suggestion or a complaint. It helps us to make the course better!

Please spare few minutes to take the survey after the webinar.

Thank you!