



## JSP - Java Server Pages

USI CBO CR LAUNCHPAD TRAINING PROGRAM

# JSP – Java Server Pages

## Context, Objectives, Agenda

### Context

- *Java Server Pages (JSP) is a Server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications.*
- *JSP have access to the entire family of Java APIs, including the JDBC API to access enterprise databases.*

### Objectives

- To learn
  - What is JSP and its notable features
  - How to use various Elements and Tags in JSP
  - How to use Java Server Pages to develop Web applications in simple and easy steps

### Agenda

Topic	Content	
JSP	<ul style="list-style-type: none"><li>• Introduction to JSP</li><li>• Life cycle of JSP</li><li>• JSP Scripting Elements/Tags</li></ul>	<ul style="list-style-type: none"><li>• Implicit Objects in JSP</li></ul>
JSP Directives	<ul style="list-style-type: none"><li>• page</li><li>• include</li></ul>	<ul style="list-style-type: none"><li>• taglib</li></ul>
JSP Action Elements	<ul style="list-style-type: none"><li>• jsp:forward</li><li>• jsp:include</li><li>• jsp:useBean</li><li>• include</li></ul>	<ul style="list-style-type: none"><li>• jsp:plugin</li><li>• Java Bean Class</li><li>• Hands On</li></ul>
Development in JSP	<ul style="list-style-type: none"><li>• Registration Form – Demo</li><li>• Login Form - Demo</li></ul>	<ul style="list-style-type: none"><li>• Dev Approach &amp; Source Code</li></ul>
JSP - Expression Language (EL)	<ul style="list-style-type: none"><li>• What is EL?</li><li>• Overview &amp; Types</li><li>• Examples</li></ul>	<ul style="list-style-type: none"><li>• Knowledge Check</li><li>• Hands on</li></ul>
JSTL	<ul style="list-style-type: none"><li>• What is JSTL?</li><li>• Advantages</li><li>• JSTL - Deep dive</li></ul>	<ul style="list-style-type: none"><li>• Illustrations</li></ul>

# JSP(Java Server Pages)

# JSP

## Objectives

01

Introduction to JSP  
JSP Architecture

03

JSP Scripting  
Elements / Tags

02

Life cycle of JSP

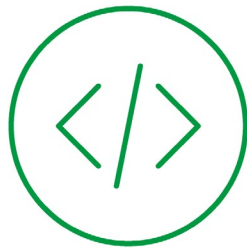
04

JSP Implicit objects

# JSP

## Introduction to JSP

- Java Server Pages (JSP) is a **Server side** technology to develop **dynamic web pages** by inserting Java code into the HTML pages by making special JSP tags.
- These JSP tags which allow java code to be included into it look something like these `<% ----java code----%>`.
- It can consist of either HTML or XML (combination of both is also possible) with JSP actions and commands.
- It can be used as HTML page, which can be used in forms and registration pages with the dynamic content into it.
- Dynamic content includes some fields like dropdown, checkboxes, etc. whose value will be fetched from the database.



✓ Internally JSPs are compiled into servlets

✓ JSPs can also be used to access JavaBeans objects

✓ JavaBeans are classes which encapsulate several objects into a single object.

It helps in accessing these object from multiple places

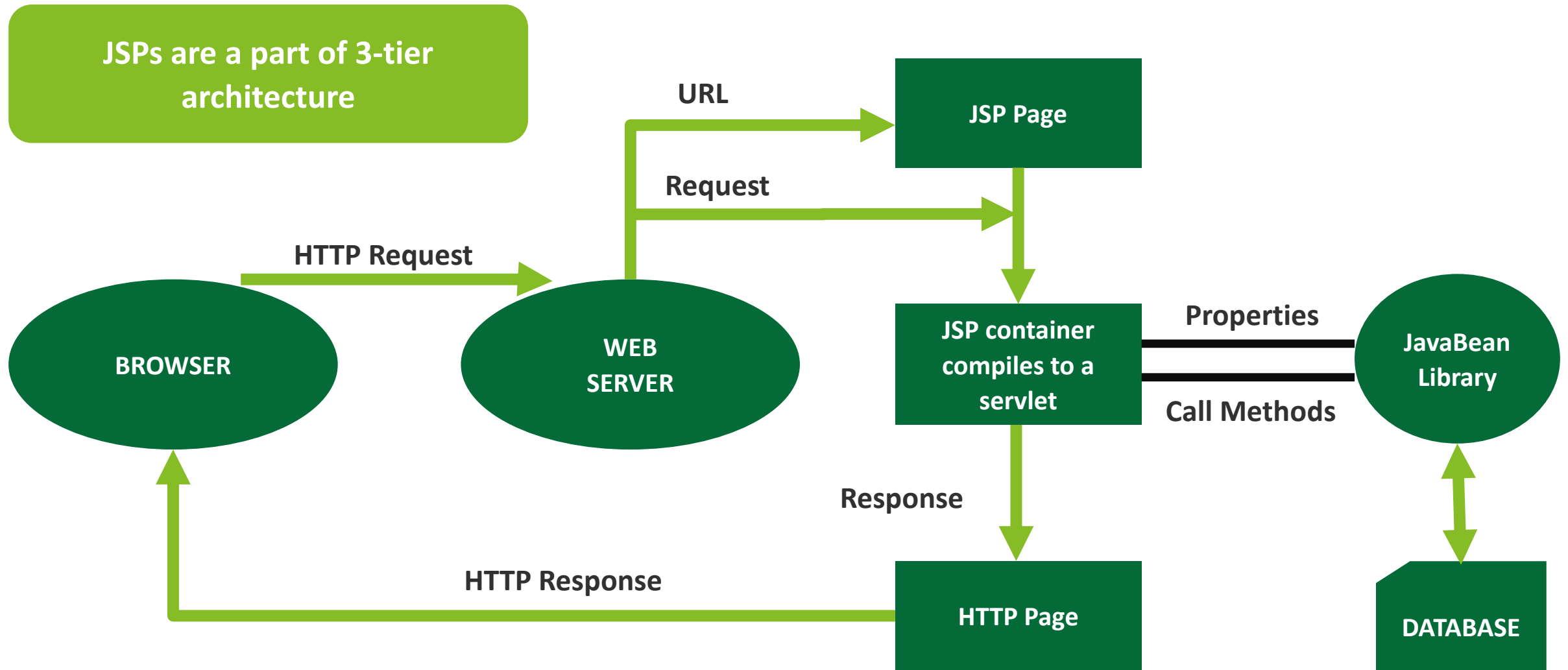
✓ JavaBeans contains several elements like Constructors, Getter/Setter Methods and much more

✓ We can share information across pages using *request* and *response* objects

✓ *Separation of Presentation layer from the business logic* in the web application

# JSP

## Architecture



# JSP

## Architecture - Explained

1

- As with a normal page, your browser sends an HTTP request to the web server.

2

- The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine.
- This is done by using the URL or JSP page which ends with .jsp instead of .html.

3

- The JSP engine loads the JSP page from disk and converts it into a servlet content.
- This conversion is very simple in which all template text is converted to `println( )` statements and all JSP elements are converted to Java code that implements the corresponding dynamic behavior of the page.

4

- The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
- A part of the Web server called the *Servlet engine* loads the Servlet class and executes it.

5

- During execution, the Servlet produces an Output in HTML format, which the servlet engine passes to the web server inside an HTTP response.
- The web server forwards the HTTP response to your browser in terms of static HTML content.

6

- Finally web browser handles the dynamically generated HTML page inside the HTTP response exactly as if it were a static page.

# JSP

## Servlet Vs JSP

### Servlet

---

1. Servlet is a Java code
2. Writing code for servlet is harder than JSP as it is html in java
3. Servlet plays a Controller role in MVC approach
4. Servlet is faster than JSP
5. Servlet can accept all protocol requests.
6. In Servlet, we can override service() method
7. In Servlet by default session management is not enabled, user have to enable it explicitly
8. In Servlet we have to implement everything like business logic and presentation logic in just one servlet file
9. Modification in Servlet is a time consuming task because it includes reloading, recompiling and restarting the server

### JSP

---

1. JSP is a Html based code
2. JSP is easy to code as it is java in html
3. JSP is the View in MVC approach for showing output
4. JSP is slower than Servlet because the first step in JSP lifecycle is the translation of JSP to Java code and then compile
5. JSP only accept http requests
6. In JSP, we cannot override its service() method
7. In JSP session management is automatically enabled
8. In JSP business logic is separated from presentation logic by using JavaBeans
9. JSP modification is fast, just need to click the refresh button

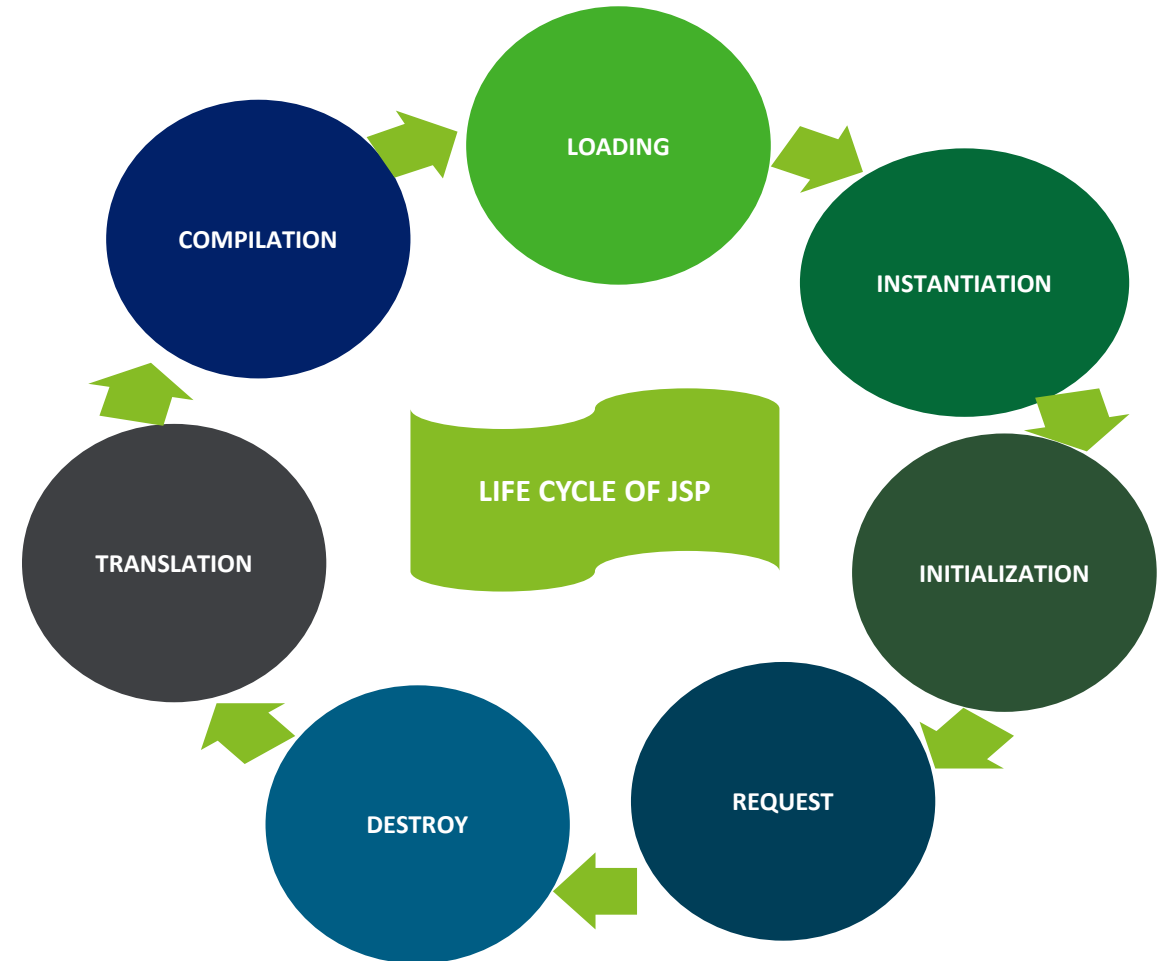


# JSP

## Life Cycle of JSP

- ✓ JSP Life Cycle is defined as translation of JSP Page into servlet as a JSP Page needs to be converted into servlet first in order to process the service requests.
- ✓ The Life Cycle starts with the creation of JSP and ends with the disintegration of that.
- ✓ When the browser asks for a JSP, JSP engine first checks whether it needs to compile the page. If the JSP is last compiled or the recent modification is done in JSP, then the JSP engine compiles the page.

- Compilation process of JSP page involves three steps:



# JSP

## Life Cycle of JSP

### 01 Translation

- ✓ JSP container parse the JSP pages and translate them to generate corresponding servlet source code. If JSP file name is hello.jsp, usually its named as hello\_jsp.java.

### 02 Compilation

- ✓ If the translation is successful, then container compiles the generated servlet source file to generate class file. Translation will be done only when container finds the servlet class is older than the JSP page.

### 03 Class Loading

- ✓ Once JSP is compiled as servlet class, its lifecycle is similar to servlet and it gets loaded into memory.

### 04 Instance Creation

- ✓ After JSP class is loaded into memory, its object is instantiated by the container.

### 05 Initialization

- ✓ The JSP class is then initialized and it transforms from a normal class to servlet. The container invokes `_jspInit()` method.

### 06 Request Processing

- ✓ The container invokes the `_jspService()` method, passing a `request(HttpServletRequest)` and `response(HttpServletResponse)` object and generate the HTML response.

### 07 Destroy

- ✓ The container invokes the `_jspDestroy()` method if the container needs to remove the instance of the servlet class.

# JSP

## JSP Scripting Elements / Tags

---

- **SCRIPTLET TAG :**

- enclosed in `<%` and `%>` markers:
- A Scriptlet can contain any number of Java language statements, variable or method declarations, or expressions that are valid in the page scripting language.

```
<%  
String message = "Hello World";  
out.println(message);  
%>
```

---

- **EXPRESSION TAG :**

- enclosed in `<%=` and `%>` markers
- A expression is used to insert the result of a Java expression directly into the output.

```
The time is : <%= new java.util.Date() %>
```

---

- **DECLARATION TAG :**

- DECLARATION enclosed in `<%!` and `%>` markers
- This tag allows the developer to declare variables or methods.
- Code placed in this must end in a semicolon(;).
- Declarations do not generate output, so are used with JSP expressions or scriptlets.

```
<%! private int counter = 0;  
private String getAccount (int accountNo); %>
```



## Knowledge Check

1. The difference between Servlets and JSP is the .....

- ☐ Translation
- ☐ Compilation
- ☐ Syntax
- ☐ Both A & B

2. What is the first step in the lifecycle of JSP?

- ☐ Compilation
- ☐ Translation
- ☐ Class Loading
- ☐ None of the above

3. Java Server Pages often serve the same purpose as programs implemented using the Common Gateway Interface (CGI).

- ☐ True
- ☐ False

# JSP

## Implicit Objects

- These Objects are the Java objects that the JSP Container makes available to the developers in each page and the developer can call them directly without being explicitly declared.
- JSP Implicit Objects are also called pre-defined variables.
- ***There are 9 Implicit Objects that JSP supports –***

1

**request** : This is the HttpServletRequest object associated with the request.

6

**config** : This is the ServletConfig object associated with the page.

2

**response** : This is the HttpServletResponse object associated with the response to the client.

7

**pageContext** : This encapsulates use of server-specific features like higher performance JspWriters.

3

**out** : This is the PrintWriter object used to send output to the client.

8

**page** : This is simply a synonym for this, and is used to call the methods defined by the translated servlet class.

4

**session** : This is the HttpSession object associated with the request.

9

**exception** : The Exception object allows the exception data to be accessed by designated JSP.

5

**application** : This is the ServletContext object associated with the application context.



## Knowledge Check

1. How many jsp implicit objects are there and these objects are created by the web container that are available to all the jsp pages?

- ☐ 8
- ☐ 9
- ☐ 10
- ☐ 7

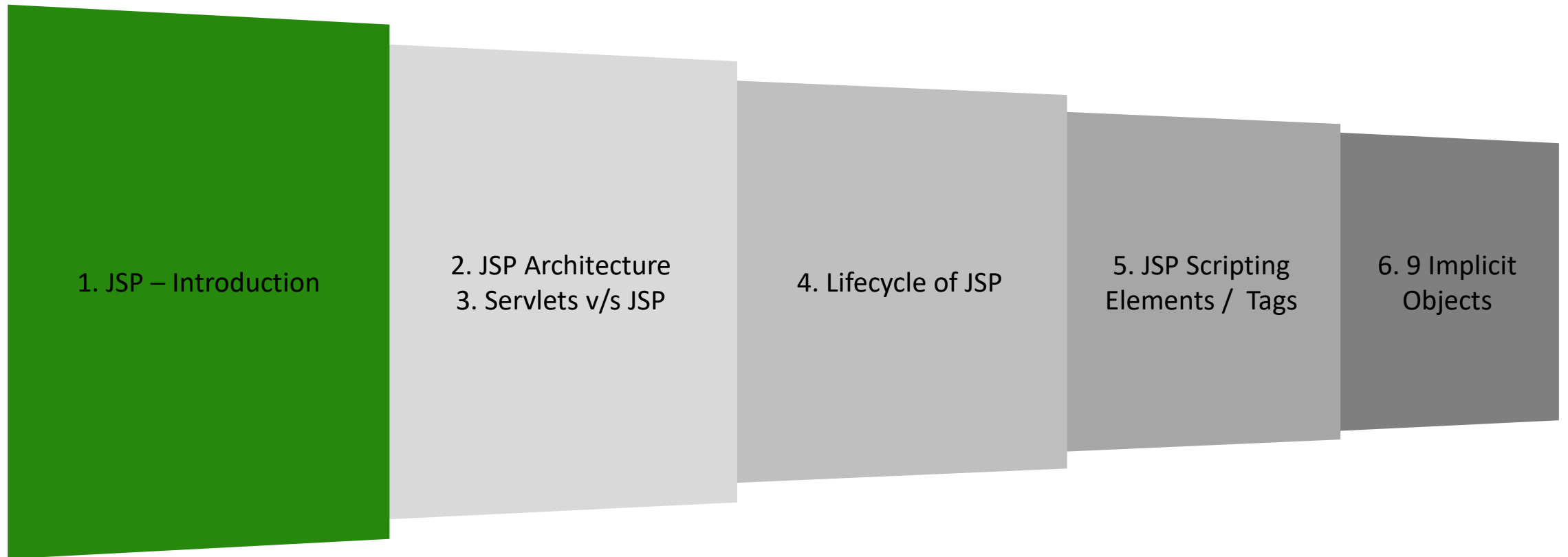
2. Which of the following encapsulates use of server-specific features like higher performance JspWriters.

- ☐ page
- ☐ session
- ☐ pageContext
- ☐ None of the above

3. This object allows the JSP programmer access to the Servlet or JSP engine initialization parameters such as the paths or file locations etc.

- ☐ config
- ☐ response
- ☐ page
- ☐ session

## Glimpse of Important points



**Break – 15 min.**



# JSP Directives

# JSP Directives

## Objectives

01 page

02 include

03 taglib

04 Knowledge Check

# JSP Directives

## Introduction

- ❖ JSP directives are the messages to JSP container. They provide global information about an entire JSP page.
- ❖ JSP directives are used to give special instruction to a container for translation of JSP to servlet code.
- ❖ In JSP life cycle phase, JSP has to be converted to a servlet which is the translation phase.
- ❖ They give instructions to the container on how to handle certain aspects of JSP processing
- ❖ Directives can have many attributes by comma separated as key-value pairs.
- ❖ In JSP, directive is described in `<%@ %>` tags.

### Syntax of Directive :

```
<%@ directive attribute=" " %>
```

- ***There are 3 types of JSP Directives:***

✓ Page Directive

✓ Include Directive

✓ Taglib Directive

# JSP Directives

## page

- ❖ It provides attributes that get applied to entire JSP page.
- ❖ It defines page dependent attributes, such as scripting language, error page, and buffering requirements.
- ❖ It is used to provide instructions to a container that pertains to current JSP page.

### Syntax of page Directive :

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

- Following are its list of attributes associated with page directive:

• language

• autoFlush

• extends

• buffer

• import

• isErrorPage

• contentType

• pageEncoding

• info

• errorPage

• session

• isElIgnored

• isThreadSafe

# JSP Directives

## page

- **language:** It defines the programming language (underlying language) being used in the page.

- **Syntax :**

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

- Here value is the programming language (underlying language)

- **Example :**

```
<%@ page language="java"  
contentType="text/html; charset=ISO-8859-1"  
pageEncoding="ISO-8859-1"%>
```

- **Explanation:** In the above example, attribute language value is Java which is the underlying language in this case. Hence, the code in expression tags would be compiled using java compiler.

- **extends:** This attribute is used to extend (inherit) the class like JAVA does

- **Syntax :**

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

- Here the value represents class from which it has to be inherited.

- **Example :**

```
<%@ page language="java"  
contentType="text/html; charset=ISO-8859-1"  
pageEncoding="ISO-8859-1"%>
```

```
<%@ page extends="demotest.DemoClass" %>
```

- **Explanation:** In the above code, JSP is extending DemoClass which is within demotest package, and it will extend all class features.

# JSP Directives

## page

- **import:** This attribute is most used attribute in page directive attributes. It is used to tell the container to import other java classes, interfaces, enums, etc. while generating servlet code. It is similar to import statements in java classes, interfaces.

- **Syntax :**

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

- Here value indicates the classes which have to be imported.

- **Example :**

```
<%@ page language="java"  
contentType="text/html; charset=ISO-8859-1"  
import="java.util.Date" pageEncoding="ISO-8859-1"%>
```

- **Explanation:** In the above code, we are importing Date class from java.util package (all utility classes), and it can use all methods of the following class.

- **contentType:** It defines the character encoding scheme i.e. it is used to set the content type and the character set of the response

- The default type of contentType is "text/html; charset=ISO-8859-1".

- **Syntax :**

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

- **Example :**

```
<%@ page language="java"  
contentType="text/html; charset=ISO-8859-1"  
pageEncoding="ISO-8859-1"%>
```

- **Explanation:** In the above code, the content type is set as text/html, it sets character encoding for JSP and for generated response page.

# JSP Directives

## page

- **info** : It defines a string which can be accessed by `getServletInfo()` method.
- This attribute is used to set the servlet description.

- **Syntax :**

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

- Here, the value represents the servlet information.

- **Example :**

```
<%@ page language="java"
contentType="text/html; charset=ISO-8859-1"
info="My Directive JSP"
pageEncoding="ISO-8859-1"%>
```

- **Explanation:** In the above code, string "My Directive JSP" can be retrieved by the servlet interface using `getServletInfo()`

- **session:** JSP page creates session by default.
- Sometimes we don't need a session to be created in JSP, and hence, we can set this attribute to false in that case. The default value of the session attribute is true, and the session is created. When it is set to false, then we can indicate the compiler to not create the session by default.

- **Syntax :**

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

- Here in this case session attribute can be set to true or false

- **Example :**

```
<%@ page language="java"
contentType="text/html; charset=ISO-8859-1"
session="false"%>
```

- **Explanation:** In the above example, session attribute is set to "false" hence we are indicating that we don't want to create any session in this JSP

# JSP Directives

## page

- **isThreadSafe** : It defines the threading model for the generated servlet. It indicates the level of thread safety implemented in the page.
- Its default value is true so simultaneous.
- We can use this attribute to implement SingleThreadModel interface in generated servlet. If we set it to false, then it will implement SingleThreadModel and can access any shared objects and can yield inconsistency.

- **Syntax :**

```
<% @ page isThreadSafe="true/false" %>
```

- Here true or false represents if synchronization is there then set as true and set it as false

- **Example :**

```
<%@ page language="java"  
contentType="text/html; charset=ISO-8859-1"  
isThreadSafe="true"%>
```

- **Explanation:** In the above code, isThreadSafe is set to "true" hence synchronization will be done, and multiple threads can be used.

- **autoFlush** : This attribute specifies that the buffered output should be flushed automatically or not and default value of that attribute is true.
- If the value is set to false the buffer will not be flushed automatically and if its full, we will get an exception.
- When the buffer is none then the false is illegitimate, and there is no buffering, so it will be flushed automatically.

- **Syntax :**

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

- Here true/false represents whether buffering has to be done or not

- **Example :**

```
<%@ page language="java"  
contentType="text/html; charset=ISO-8859-1"  
autoFlush="false"%>
```

- **Explanation:** In the above code, the autoFlush is set to false and hence buffering won't be done and it has manually flush the output.



# JSP Directives

## page

- **buffer** : Using this attribute the output response object may be buffered.
- We can define the size of buffering to be done using this attribute and default size is 8KB.
- It directs the servlet to write the buffer before writing to the response object.

- **Syntax :**

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

- Here the value represents the size of the buffer which has to be defined. If there is no buffer, then we can write as none, and if we don't mention any value then the default is 8KB

- **Example :**

```
<%@ page language="java"
contentType="text/html; charset=ISO-8859-1"
buffer="16KB"%>
```

- **Explanation:** In the above code, buffer size is mentioned as 16KB wherein the buffer would be of that size

- **isErrorPage** : It indicates that JSP Page that has an errorPage will be checked in another JSP page
- Any JSP file declared with "isErrorPage" attribute is then capable to receive exceptions from other JSP pages which have error pages.
- Exceptions are available to these pages only.
- The default value is false.

- **Syntax :**

```
<%@ page isErrorPage="true/false"%>
```

- **Example :**

```
<%@ page language="java"
contentType="text/html; charset=ISO-8859-1"
isErrorPage="true"%>
```

- **Explanation:** In the above code, isErrorPage is set as true. Hence, it will check any other JSPs has errorPage (described in the next attribute) attribute set and it can handle exceptions.

# JSP Directives

## page

- **pageEncoding** : The "pageEncoding" attribute defines the character encoding for JSP page.
- The default is specified as "ISO-8859-1" if any other is not specified. This attribute is used to set the servlet description.

- **Syntax :**

```
<%@ page pageEncoding="vaue" %>
```

- Here value specifies the charset value for JSP

- **Example :**

```
<%@ page language="java"  
contentType="text/html;"  
pageEncoding="ISO-8859-1"  
isErrorPage="true"%>
```

- **Explanation:** In the above code "pageEncoding" has been set to default charset ISO-8859-1

- **errorPage**: This attribute is used to set the error page for the JSP page if JSP throws an exception and then it redirects to the exception page.

- **Syntax :**

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

- Here value represents the error JSP page value

- **Example :**

```
<%@ page language="java"  
contentType="text/html;"  
pageEncoding="ISO-8859-1"  
errorPage="errorHandler.jsp"%>
```

- **Explanation:** In the above code, to handle exceptions we have errorHandler.jsp

# JSP Directives

## page

- **isELIgnored** : IsELIgnored is a flag attribute where we have to decide whether to ignore EL tags or not.
- Its datatype is java enum, and the default value is false hence EL is enabled by default.

- **Syntax :**

```
<%@ page isELIgnored="true/false" %>
```

- Here, true/false represents the value of EL whether it should be ignored or not.

- **Example :**

```
<%@ page language="java"
contentType="text/html;"
pageEncoding="ISO-8859-1"
isELIgnored="true"%>
```

- **Explanation:** In the above code, isELIgnored is true and hence Expression Language (EL) is ignored here.

- **Example with four attributes :**

```
<%@ page language="java" contentType="text/html;"
pageEncoding="ISO-8859-1"
isELIgnored="false"%>
<%@page import="java.util.Date" %>
```

```
<!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=ISO-8859-1">
<title>Directive MY JSP1</title>
</head>
<body>
<a>Date is:</a>
<%= new java.util.Date() %>
</body>
</html>
```

- **Output :** Date is: Current date using the date method of the date class

Date is: Sat Jul 11 14:06:18 UTC 2020

# JSP Directives

## include

- JSP "include directive" is used to include one file to the another file
- This included file can be HTML, JSP, text files, etc.
- It is also useful in creating templates with the user views and break the pages into header & footer and sidebar actions.
- It includes file during translation phase

### Syntax of include Directive :

```
<%@ include...%>
```

### Example

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ include file="directive_header_jsp3.jsp" %>
<!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=ISO-8859-1">
<title>My Directive JSP2</title>
</head>
<body>
<a>This is the main file</a>
</body>
</html>
```

*Directive\_jsp2.jsp (Main file)*

```
<%@ 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>
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=ISO-8859-1">
</head>
<body>
<a>Header file : </a>
<%int count =1; count++;
out.println(count);%> :
</body>
</html>
```

*Directive\_header\_jsp3.jsp (which is included in the main file)*

# JSP Directives

## include

---

- **Explanation of the code:**

- **Directive\_jsp2.jsp:**
  - **Code Line 3:** In this code, we use include tags where we are including the file directive\_header\_jsp3.jsp into the main file(\_jsp2.jsp)and gets the output of both main file and included file.
- **Directive\_header\_jsp3.jsp:**
  - **Code Line 11-12:** We have taken a variable count initialized to 1 and then incremented it. This will give the output in the main file as shown below.

---

- **Output :**

- The output is Header file: 2 : This is the main file
- The output is executed from the directive\_jsp2.jsp file while the directive\_header\_jsp3.jsp included file will be compiled first.
- After the included file is done, the main file is executed, and the output will be from the main file "This is the main file". So you will get the output as "Header file: 2" from \_jsp3.jsp and "This is main file" from \_jsp2.jsp.

# JSP Directives

## taglib

- JSP taglib directive is used to define the tag library with "taglib" as the prefix, which we can use in JSP.
- JSP taglib directive is used in the JSP pages using the JSP standard tag libraries
- It uses a set of custom tags, identifies the location of the library and provides means of identifying custom tags in JSP page.

### Syntax of taglib Directive :

```
<%@ taglib uri="uri" prefix="value"%>
```

- Here "uri" attribute is a unique identifier in tag library descriptor and "prefix" attribute is a tag name.
- 
- **Explanation:**
  - Code Line 4 : Here "taglib" is defined with attributes uri and prefix.
  - Code Line 13 : "testtag" is the custom tag defined and it can be used anywhere

## Example

```
<%@ page language="java"
contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="testtag"
uri="http://java.sun.com/jsp/jstl/core" %>
<!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=ISO-8859-1">
<title>My Directive JSP</title>
<testtag:hello/>
</head>
<body>
</body>
</html>
```



## Knowledge Check

1. Which directive is used to include one file to the another file ?

- ☐ page
- ☐ include
- ☐ taglib

2. Which of the following defines the threading model for the generated servlet & indicates the level of thread safety implemented in the page?

- ☐ session
- ☐ isThreadSafe
- ☐ autoflush
- ☐ buffer

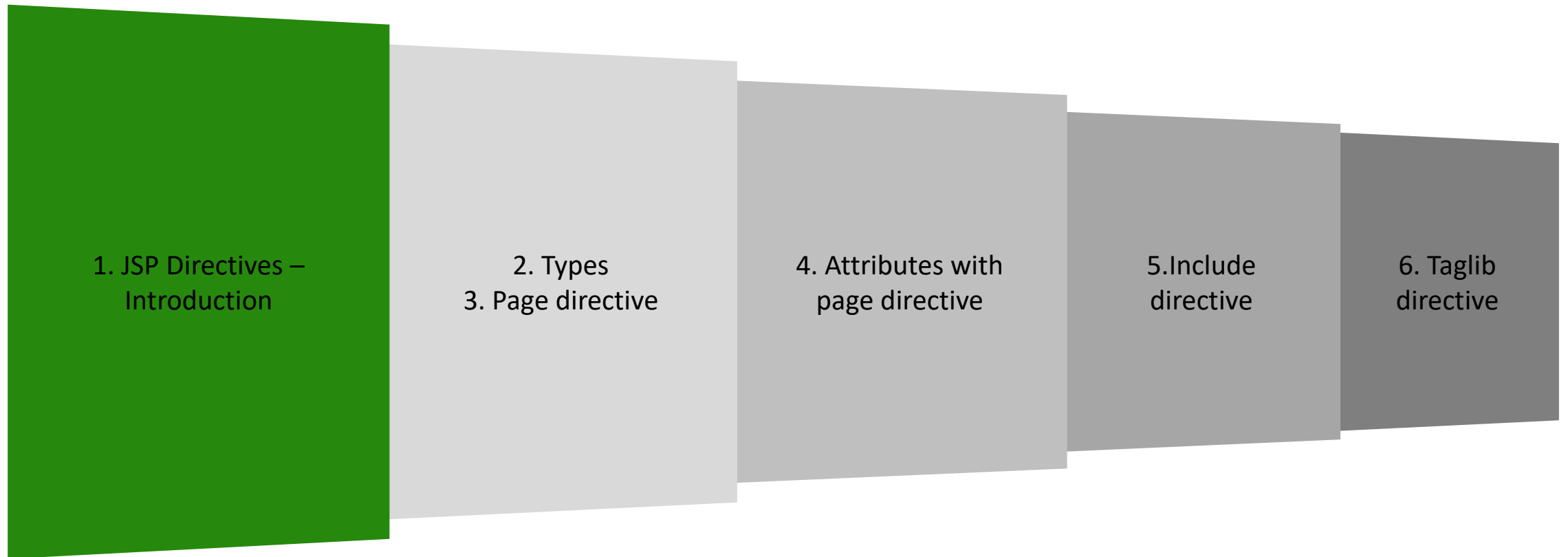
3. Which directive provides attributes that get applied to entire JSP page.

- ☐ page
- ☐ include
- ☐ taglib

# JSP Directives

## Recap

### Glimpse of Important points





**Lunch Break – 45 min.**

# JSP Action Elements

# JSP Action Elements

## Objectives

01

Introduction

02

Types

03

Example

04

Knowledge Check

# JSP Action Elements

## Introduction

- JSP actions use the construct in XML syntax to control the behavior of the servlet engine.
- We can dynamically insert a file, reuse the beans components, forward user to another page, etc. through JSP Actions like include and forward.
- Unlike directives, actions are re-evaluated each time the page is accessed.

### Syntax :

```
<jsp:action_name attribute="value" />
```

- **Following are the types of Standard Action Tags :**

jsp:include

jsp:plugin

jsp:usebean

jsp:element

jsp:setproperty

jsp:attribute

jsp:getproperty

jsp:body

jsp:forward

jsp:text

# JSP Action Elements

## Types

Sr No	Type	Purpose
1	<b>jsp:include</b>	<ul style="list-style-type: none"><li>Includes a file at the time the page is requested.</li></ul>
2	<b>jsp:useBean</b>	<ul style="list-style-type: none"><li>Finds or instantiates a JavaBean.</li></ul>
3	<b>jsp:setProperty</b>	<ul style="list-style-type: none"><li>Sets the property of a JavaBean.</li></ul>
4	<b>jsp:getProperty</b>	<ul style="list-style-type: none"><li>Inserts the property of a JavaBean into the output.</li></ul>
5	<b>jsp:forward</b>	<ul style="list-style-type: none"><li>Forwards the requester to a new page.</li></ul>
6	<b>jsp:plugin</b>	<ul style="list-style-type: none"><li>Generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin.</li></ul>
7	<b>jsp:element</b>	<ul style="list-style-type: none"><li>Defines XML elements dynamically.</li></ul>
8	<b>jsp:attribute</b>	<ul style="list-style-type: none"><li>Defines dynamically-defined XML element's attribute.</li></ul>
9	<b>jsp:body</b>	<ul style="list-style-type: none"><li>Defines dynamically-defined XML element's body.</li></ul>
10	<b>jsp:text</b>	<ul style="list-style-type: none"><li>Used to write template text in JSP pages and documents.</li></ul>

# JSP Action Elements

## Example

- Let us define a test bean that will further be used in our example –

```
/* File: TestBean.java */
package action;
public class TestBean {
    private String message = "No message specified";
    public String getMessage() {
        return(message);
    }
    public void setMessage(String message) {
        this.message = message;
    }
}
```

- Now use the following code in main.jsp file. This loads the bean and sets/gets a simple String parameter –

```
<html>
<head>
<title>Using JavaBeans in JSP</title>
</head>
<body>
<center>
<h2>Using JavaBeans in JSP</h2>
<jsp:useBean id = "test" class = "action.TestBean" />
<jsp:setProperty name = "test" property = "message"
value = "Hello JSP..." />
<p>Got message....</p>
<jsp:getProperty name = "test" property = "message" />
</center>
</body>
</html>
```

- Output –

### Using JavaBeans in JSP

```
Got message....
Hello JSP...
```



## Knowledge Check

1. Which of the following action inserts the file at the time the page is requested?

- ☐ usebean
- ☐ include
- ☐ Forward
- ☐ plugin

2. Which of the following action is used to retrieve the value of a given property and converts it to a string, and finally inserts it into the output?

- ☐ usebean
- ☐ getProperty
- ☐ setProperty
- ☐ plugin

3. Which of the following action is used to insert Java components into a JSP page?

- ☐ usebean
- ☐ include
- ☐ text
- ☐ plugin

# JSP Action Elements

## Recap

### Glimpse of Important points



1. JSP Action Elements –  
Introduction

2. Types

3. Example



# Development in JSP

# Development in JSP

## Objectives

01

Registration Form -  
Demo

03

Dev Approach &  
Source Code

02

Login Form - Demo

04

Knowledge Check

# Development in JSP

## Registration Form Demo

### ❑ Approach

- *For creating registration form, you must have a table in the database.*
- *You can write the database logic in JSP file, but separating it from the JSP page is better approach.*
- *Here, we are going to use DAO, Factory Method, DTO and Singleton design patterns.*

### ❑ Files involved

- index.jsp for getting the values from the user
- User.java, a bean class that have properties and setter and getter methods.
- process.jsp, a jsp file that processes the request and calls the methods
- Provider.java, an interface that contains many constants like DRIVER\_CLASS, CONNECTION\_URL, USERNAME and PASSWORD
- ConnectionProvider.java, a class that returns an object of Connection. It uses the Singleton and factory method design pattern.
- RegisterDao.java, a DAO class that is responsible to get access to the database

### ❑ Example of Registration Form in JSP :

- Let us first create the table in the Oracle database:

```
CREATE TABLE "USER432"  
(  "NAME" VARCHAR2(4000),  
  "EMAIL" VARCHAR2(4000),  
  "PASS" VARCHAR2(4000)  
)  
/
```

# Development in JSP

## Registration Form Demo

### ❑ index.jsp :

- We are having three fields here :

```
<form action="process.jsp">
<input type="text" name="uname" value="Name..."
onclick="this.value=''" /><br/>
<input type="text" name="uemail" value="Email ID..."
onclick="this.value=''" /><br/>
<input type="password" name="upass" value="Password..."
onclick="this.value=''" /><br/>
<input type="submit" value="register" />
</form>
```

### ❑ process.jsp :

- This jsp file contains all the incoming values to an object of bean class which is passed as an argument in the register method of the RegisterDao class.

```
<%@page import="bean.RegisterDao"%>
<jsp:useBean id="obj" class="bean.User"/>
<jsp:setProperty property="*" name="obj"/>
<%
int status=RegisterDao.register(obj);
if(status>0)
out.print("You are successfully registered");
%>
```

### ❑ User.java :

- It is the bean class that have 3 properties uname, uemail and upass with its setter and getter methods.

```
package bean;
public class User {
private String uname,upass,uemail;
public String getUname() {
return uname;
}
public void setUname(String uname) {
this.uname = uname;
}
public String getUpass() {
return upass;
}
public void setUpass(String upass) {
this.upass = upass;
}
public String getUemail() {
return uemail;
}
public void setUemail(String uemail) {
this.uemail = uemail;
}
}
```

# Development in JSP

## Registration Form Demo

### ❑ Provider.java :

- This interface contains four constants that can vary from database to database.

```
package bean;
public interface Provider {
String DRIVER="oracle.jdbc.driver.OracleDriver";
String CONNECTION_URL="jdbc:oracle:thin:@localhost:1521:xe";
String USERNAME="system";
String PASSWORD="oracle";
}
```

### ❑ connectionProvider.java :

- This class is responsible to return the object of Connection. Here, driver class is loaded only once and connection object gets memory only once.

```
package bean;
import java.sql.*;
import static bean.Provider.*;
public class ConnectionProvider {
private static Connection con=null;
static{
try{
Class.forName(DRIVER);
con=DriverManager.getConnection(CONNECTION_URL,USERNAME,PASSWORD);
}catch(Exception e){}}
public static Connection getCon(){
return con;}}
```

### ❑ RegisterDao.java :

- This class inserts the values of the bean component into the database.

```
package bean;
import java.sql.*;
public class RegisterDao {
public static int register(User u){
int status=0;
try{
Connection con=ConnectionProvider.getCon();
PreparedStatement ps=con.prepareStatement
("insert into user432 values(?,?,?)");
ps.setString(1,u.getUname());
ps.setString(2,u.getUemail());
ps.setString(3,u.getUpass());
status=ps.executeUpdate();
}catch(Exception e){}
return status;
}
}
```

# Development in JSP

## Login Form Demo

- In this example of creating login form, we have used the DAO (Data Access Object), Factory method and DTO (Data Transfer Object) design patterns. There are many files:
  - index.jsp it provides three links for login, logout and profile
  - login.jsp for getting the values from the user
  - loginprocess.jsp, a jsp file that processes the request and calls the methods.
  - LoginBean.java, a bean class that have properties and setter and getter methods.
  - Provider.java, an interface that contains many constants like DRIVER\_CLASS, CONNECTION\_URL, USERNAME and PASSWORD
  - ConnectionProvider.java, a class that is responsible to return the object of Connection. It uses the Singleton and factory method design pattern.
  - LoginDao.java, a DAO class that verifies the emailId and password from the database.
  - logout.jsp it invalidates the session.
  - profile.jsp it provides simple message if user is logged in, otherwise forwards the request to the login.jsp page.

## • Example of Registration Form in JSP :

- In this example, we are using the Oracle10g database to connect with the database. Let's first create the table in the Oracle database:

```
CREATE TABLE "USER432"  
(  "NAME" VARCHAR2(4000),  
  "EMAIL" VARCHAR2(4000),  
  "PASS" VARCHAR2(4000)  
)  
/
```

# Development in JSP

## Login Form Demo

- **index.jsp :**

- It simply provides three links for login, logout and profile.

```
<a href="login.jsp">login</a>|  
<a href="logout.jsp">logout</a>|  
<a href="profile.jsp">profile</a>
```

- **login.jsp :**

- This file creates a login form for two input fields name and password. It is the simple login form, you can change it for better look and feel. We are focusing on the concept only.

```
<%@ include file="index.jsp" %>  
<hr/>  
<h3>Login Form</h3>  
<%  
String profile_msg=(String)request.getAttribute("profile_msg");  
if(profile_msg!=null){  
out.print(profile_msg); }  
String login_msg=(String)request.getAttribute("login_msg");  
if(login_msg!=null){  
out.print(login_msg); }  
%>  
<br/>  
<form action="loginprocess.jsp" method="post">  
Email:<input type="text" name="email"/><br/><br/>  
Password:<input type="password" name="password"/><br/><br/>  
<input type="submit" value="login"/>  
</form>
```

- **loginprocess.jsp :**

- This jsp file contains all the incoming values to an object of bean class which is passed as an argument in the validate method of the LoginDao class. If emailid and password is correct, it displays a message you are successfully logged in! and maintains the session so that we may recognize the user.

```
<%@page import="bean.LoginDao"%>  
<jsp:useBean id="obj" class="bean.LoginBean"/>  
<jsp:setProperty property="*" name="obj"/>  
<%  
boolean status=LoginDao.validate(obj);  
if(status){  
out.println("You r successfully logged in");  
session.setAttribute("session","TRUE");  
}  
else  
{  
out.print("Sorry, email or password error");  
%>  
<jsp:include page="index.jsp"></jsp:include>  
<%  
}  
%>
```

# Development in JSP

## Login Form Demo

### LoginBean.java :

- It is the bean class that have 2 properties email and pass with its setter and getter methods.

```
package bean;
public class LoginBean {
    private String email,pass;
    public String getEmail() {
        return email; }
    public void setEmail(String email) {
        this.email = email; }
    public String getPass() {
        return pass; }
    public void setPass(String pass) {
        this.pass = pass; } }
```

### Provider.java

- This interface contains four constants that may differ from database to database.

```
package bean;
public interface Provider {
    String DRIVER="oracle.jdbc.driver.OracleDriver";
    String CONNECTION_URL="jdbc:oracle:thin:@localhost:1521:xe";
    String USERNAME="system";
    String PASSWORD="oracle"; }
```

### ConnectionProvider.java

- This class provides a factory method that returns the object of Connection. Here, driver class is loaded only once and connection object gets memory only once because it is static.

```
package bean;
import java.sql.*;
import static bean.Provider.*;
public class ConnectionProvider {
    private static Connection con=null;
    static{
        try{
            Class.forName(DRIVER);
            con=DriverManager.getConnection(CONNECTION_URL,USERNAME
                ,PASSWORD);
        }catch(Exception e){}
    }
    public static Connection getCon(){
        return con;
    }
}
```



# Development in JSP

## Login Form Demo

---

### loginBean.java :

- This class verifies the email id and password.

```
package bean;
import java.sql.*;
public class LoginDao {
    public static boolean validate(LoginBean bean){
        boolean status=false;
        try{
            Connection con=ConnectionProvider.getCon();
            PreparedStatement ps=con.prepareStatement(
                "select * from user432 where email=? and pass=?");
            ps.setString(1,bean.getEmail());
            ps.setString(2, bean.getPass());
            ResultSet rs=ps.executeQuery();
            status=rs.next();
        }catch(Exception e){}
        return status;
    }
}
```

# JSP - Expression Language (EL)

# JSP - Expression Language (EL)

## Objectives

01

What is EL?

03

Examples

02

Overview & Types

04

Knowledge Check

# JSP - Expression Language (EL)

## Introduction

- The Expression Language (EL) simplifies the accessibility of data stored in the Java Bean component, and other objects like request, session, application etc.
- There are many implicit objects, operators and reserve words in EL.
- It is the newly added feature in JSP technology version 2.0.

- **Example** - In this example, we will see how EL is used as an operator to add two numbers (1+2) and get the output respectively.

```
<%@ 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>
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=ISO-8859-1">
<title>My JSP1</title>
</head>
<body>
<a>Expression is:</a>
${1+2};
</body>
</html>
```

## Syntax for EL:

`${ expression }`

- **Explanation of Code - Code Line 14:**
  - Expression Language (EL) is set where we are adding two numbers 1+2, hence it will give output as 3.
  - When you execute the above code, you will have the following Output.
  - Expression is: 3
  - As numbers 1+2 will be added and serve as an output

# JSP - Expression Language (EL)

## Implicit Objects in Expression Language (EL)

Sr No	Implicit Objects	Usage
1	pageScope	<ul style="list-style-type: none"><li>it maps the given attribute name with the value set in the page scope</li></ul>
2	requestScope	<ul style="list-style-type: none"><li>it maps the given attribute name with the value set in the request scope</li></ul>
3	sessionScope	<ul style="list-style-type: none"><li>it maps the given attribute name with the value set in the session scope</li></ul>
4	applicationScope	<ul style="list-style-type: none"><li>it maps the given attribute name with the value set in the application scope</li></ul>
5	param	<ul style="list-style-type: none"><li>it maps the request parameter to the single value</li></ul>
6	paramValues	<ul style="list-style-type: none"><li>it maps the request parameter to an array of values</li></ul>
7	header	<ul style="list-style-type: none"><li>it maps the request header name to the single value</li></ul>
8	headerValues	<ul style="list-style-type: none"><li>it maps the request header name to an array of values</li></ul>
9	cookie	<ul style="list-style-type: none"><li>it maps the given cookie name to the cookie value</li></ul>
10	initParam	<ul style="list-style-type: none"><li>it maps the initialization parameter</li></ul>
11	pageContext	<ul style="list-style-type: none"><li>it provides access to many objects request, session etc.</li></ul>

# JSP - Expression Language (EL)

## Operators & Reserve Words

- Precedence of Operators in EL :**

- There are many operators that have been provided in the Expression Language. Their precedence are as follows:

[]

()

-(unary) not ! empty

\* / div % mod

+ - (binary)

< <= > >= lt le gt ge

== != eq ne

&& and

|| or

?:

- Reserve words in EL**

- There are many reserve words in the Expression Language. They are as follows:

lt

le

and

gt

not

or

ge

eq

Instanceof

div

ne

mod

true

empty

null

false



## Knowledge Check

1. Which of the following implicit object maps the request parameter to an array of values ?

- ☐ Param
- ☐ ParamValues
- ☐ Header
- ☐ headerValues

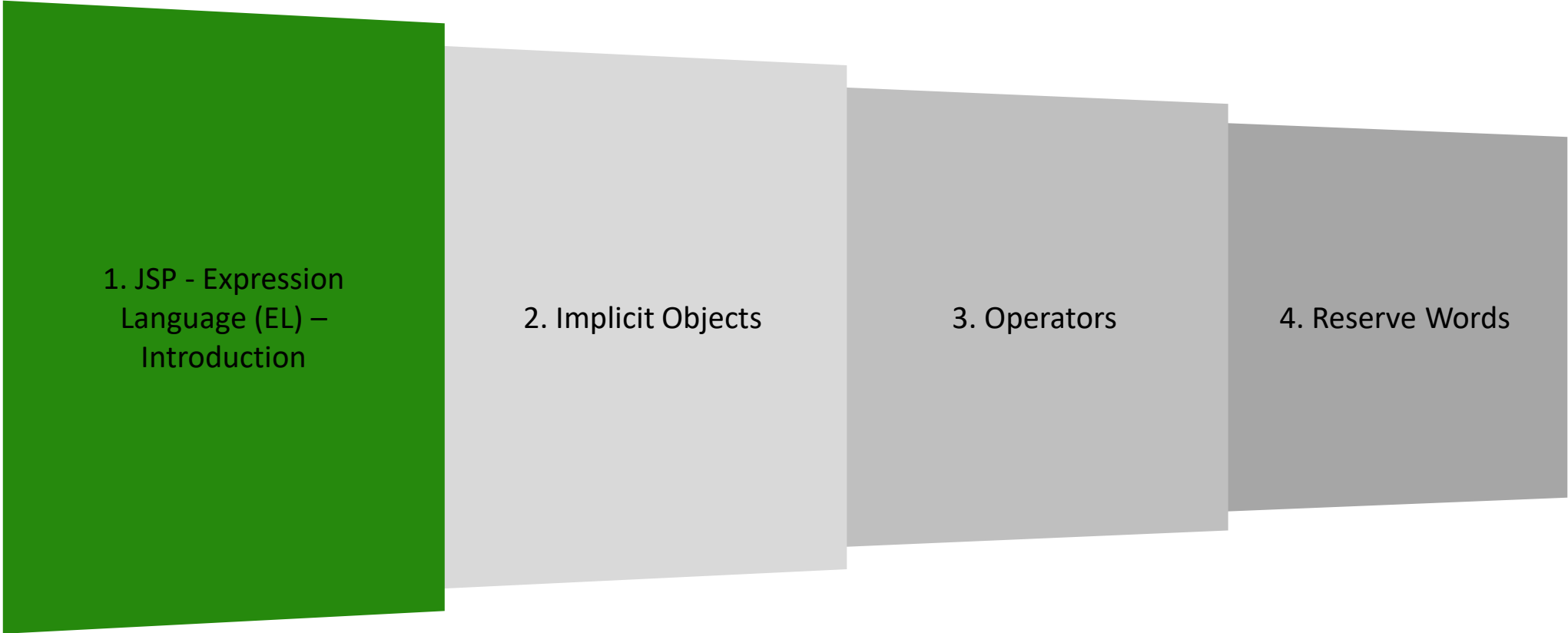
2. Which of the following is a reserve word in EL?

- ☐ gt
- ☐ gr
- ☐ gs
- ☐ gl

# Recap

## JSP - Expression Language (EL)

### Glimpse of Important points



1. JSP - Expression  
Language (EL) –  
Introduction

2. Implicit Objects

3. Operators

4. Reserve Words



**Break – 15 min.**

# **JSTL(JSP Standard Tag Library)**

01

What is JSTL?

02

Advantages

03

JSTL Deep Dive

04

Examples

# JSTL

## Introduction

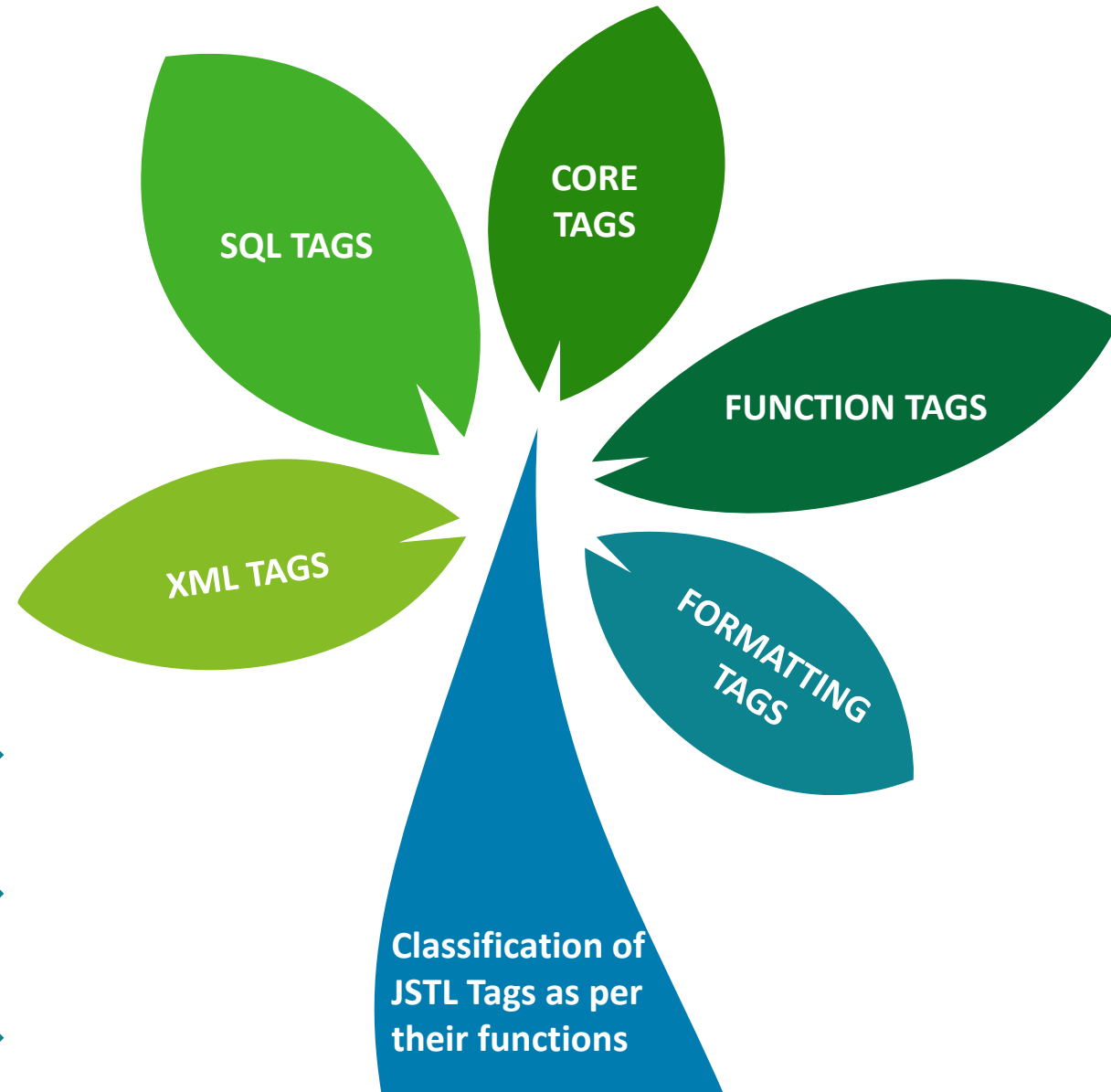
- The JavaServer Pages Standard Tag Library (JSTL) is a collection of useful JSP tags which encapsulates the core functionality common to many JSP applications.
- JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags.
- It also provides a framework for integrating the existing custom tags with the JSTL tags.

- **Advantages of JSTL:**

Fast Development - JSTL provides many tags that simplify the JSP.

Code Reusability - We can use the JSTL tags on various pages.

No need to use scriptlet tag - It avoids the use of scriptlet tag.



# JSTL

## JSTL Tags

Sr No	Tag Name	Description
1	Core tags	❖ The JSTL core tag provide variable support, URL management, flow control, etc. The URL for the core tag is <a href="http://java.sun.com/jsp/jstl/core">http://java.sun.com/jsp/jstl/core</a> . The prefix of core tag is <b>c</b> .
2	Function tags	❖ The functions tags provide support for string manipulation and string length. The URL for the functions tags is <a href="http://java.sun.com/jsp/jstl/functions">http://java.sun.com/jsp/jstl/functions</a> and prefix is <b>fn</b> .
3	Formatting tags	❖ The Formatting tags provide support for message formatting, number and date formatting, The URL for the Formatting tags is <a href="http://java.sun.com/jsp/jstl/fmt">http://java.sun.com/jsp/jstl/fmt</a> and prefix is <b>fmt</b> .
4	XML tags	❖ The XML tags provide flow control, transformation, etc. The URL for the XML tags is <a href="http://java.sun.com/jsp/jstl/xml">http://java.sun.com/jsp/jstl/xml</a> and prefix is <b>x</b> .
5	SQL tags	❖ The JSTL SQL tags provide SQL support to access SQL databases. The URL for the SQL tags is <a href="http://java.sun.com/jsp/jstl/sql">http://java.sun.com/jsp/jstl/sql</a> and prefix is <b>sql</b> .

# JSTL

## JSTL Core Tags

Sr No	Core Tags	Description
1	c:out	▪ It display the result of an expression, similar to the way <%=...%> tag work.
2	c:import	▪ It retrives relative or an absolute URL and display the contents to either a String in 'var',a Reader in 'varReader' or the page.
3	c:set	▪ It sets the result of an expression under evaluation in a 'scope' variable.
4	c:remove	▪ It is used for removing the specified scoped variable from a particular scope.
5	c:catch	▪ It is used for Catches any Throwable exceptions that occurs in the body.
6	c:if	▪ It is conditional tag used for testing the condition and display the body content only if the expression evaluates is true.
7	c:choose, c:when, c:otherwise	▪ It is the simple conditional tag that includes its body content if the evaluated condition is true.
8	c:forEach	▪ It is the basic iteration tag. It repeats the nested body content for fixed number of times or over collection.
9	c:forTokens	▪ It iterates over tokens which is separated by the supplied delimiters.
10	c:param	▪ It adds a parameter in a containing 'import' tag's URL.
11	c:redirect	▪ It redirects the browser to a new URL and supports the context-relative URLs.
12	c:url	▪ It creates a URL with optional query parameters.

# JSTL

## JSTL Function Tags

Sr No	Function Tags	Description
1	fn:contains()	▪ It is used to test if an input string containing the specified substring in a program.
2	fn:containsIgnoreCase()	▪ It is used to test if an input string contains the specified substring as a case insensitive way.
3	fn:endsWith()	▪ It is used to test if an input string ends with the specified suffix.
4	fn:escapeXml()	▪ It escapes the characters that would be interpreted as XML markup.
5	fn:indexOf()	▪ It returns an index within a string of first occurrence of a specified substring.
6	fn:trim()	▪ It removes the blank spaces from both the ends of a string.
7	fn:startsWith()	▪ It is used for checking whether the given string is started with a particular string value.
8	fn:split()	▪ It splits the string into an array of substrings.
9	fn:toLowerCase()	▪ It converts all the characters of a string to lower case.
10	fn:toUpperCase()	▪ It converts all the characters of a string to upper case.
11	fn:substring()	▪ It returns the subset of a string according to the given start and end position.
12	fn:substringAfter()	▪ It returns the subset of string after a specific substring.
13	fn:substringBefore()	▪ It returns the subset of string before a specific substring.
14	fn:length()	▪ It returns the number of characters inside a string, or the number of items in a collection.
15	fn:replace()	▪ It replaces all the occurrence of a string with another string sequence.

# JSTL

## JSTL Formatting Tags

Sr No	Formatting Tags	Description
1	fmt:parseNumber	▪ It is used to Parses the string representation of a currency, percentage or number.
2	fmt:timeZone	▪ It specifies a parsing action nested in its body or the time zone for any time formatting.
3	fmt:formatNumber	▪ It is used to format the numerical value with specific format or precision.
4	fmt:parseDate	▪ It parses the string representation of a time and date.
5	fmt:bundle	▪ It is used for creating the ResourceBundle objects which will be used by their tag body.
6	fmt:setTimeZone	▪ It stores the time zone inside a time zone configuration variable.
7	fmt:setBundle	▪ It loads the resource bundle and stores it in a bundle configuration variable or the named scoped variable.
8	fmt:message	▪ It display an internationalized message.
9	fmt:formatDate	▪ It formats the time and/or date using the supplied pattern and styles.



# JSTL

## JSTL XML Tags

Sr No	XML Tags	Description
1	x:out	▪ Similar to <%= ... > tag, but for XPath expressions.
2	x:parse	▪ It is used for parse the XML data specified either in the tag body or an attribute.
3	x:set	▪ It is used to sets a variable to the value of an XPath expression.
4	x:choose	▪ It is a conditional tag that establish a context for mutually exclusive conditional operations.
5	x:when	▪ It is a subtag of that will include its body if the condition evaluated be 'true'.
6	x:otherwise	▪ It is subtag of that follows tags and runs only if all the prior conditions evaluated be 'false'.
7	x:if	▪ It is used for evaluating the test XPath expression and if it is true, it will processes its body content.
8	x:transform	▪ It is used in a XML document for providing the XSL(Extensible Stylesheet Language) transformation.
9	x:param	▪ It is used along with the transform tag for setting the parameter in the XSLT style sheet.

# JSTL

## JSTL SQL Tags

Sr No	SQL Tags	Description
1	sql:setDataSource	▪ It is used for creating a simple data source suitable only for prototyping.
2	sql:query	▪ It is used for executing the SQL query defined in its sql attribute or the body.
3	sql:update	▪ It is used for executing the SQL update defined in its sql attribute or in the tag body.
4	sql:param	▪ It is used for sets the parameter in an SQL statement to the specified value.
5	sql:dateParam	▪ It is used for sets the parameter in an SQL statement to a specified java.util.Date value.
6	sql:transaction	▪ It is used to provide the nested database action with a common connection.



## Knowledge Check

1. Which of the following tags is used for executing the SQL update defined in its sql attribute or in the tag body.

- ☐ x:out
- ☐ fmt:bundle
- ☐ sql:update
- ☐ c:param

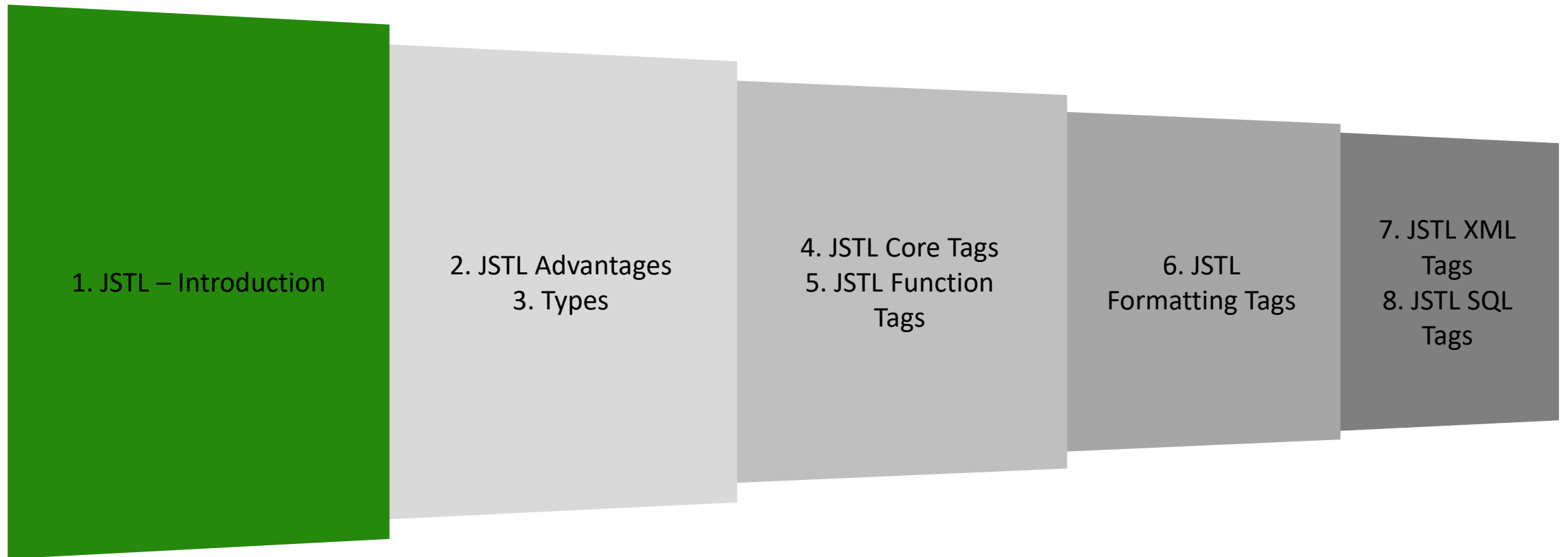
2. Which of the following tags escapes the characters that would be interpreted as XML markup.

- ☐ fn:escapeXml()
- ☐ x:transform
- ☐ sql:transaction
- ☐ fn:indexOf()

3. Which of the following tags is subtag of that follows tags and runs only if all the prior conditions evaluated be 'false'.

- ☐ x:choose
- ☐ x:when
- ☐ x:otherwise
- ☐ x:if

## Glimpse of Important points



**Any Questions?**

**Thank you**