Web Component Development Using Java

Session: 11

JSP Expression Language

Objectives

- Explain how to use script expressions in JSP
- Describe the implicit objects used in EL
- Describe the various operators used in EL
- Explain how to create static method and tag library descriptor using EL
- Explain how to modify deployment descriptor using EL
- Explain how to access EL functions within JSP
- Explain the concept of boxing and unboxing
- Explain how to coerce a value to string or number type

Expression Language 1-3

- Expression Language (EL) is a primary feature of the JSP technology.
- EL:
  - Is simple and robust and can handle both expressions and literals, which are constants and are assigned some memory location.
  - Provides cleaner syntax and is specifically designed for JSP.
  - Makes possible to easily access application data stored in JavaBeans components.
  - Is a great help to the page authors in accessing and manipulating the application data without mastering the complexities of the programming language, such as Java and JavaScript.
  - Can be used to display the generated dynamic content in a table on a Web page. In addition, EL can also be used in HTML tags.
- **Syntax**:

```
$ {EL expression}
```

where,

- $ indicates the beginning of an expression in EL.
- { is the opening delimiter.
- EL expression specifies the expression.
- } is the closing delimiter.

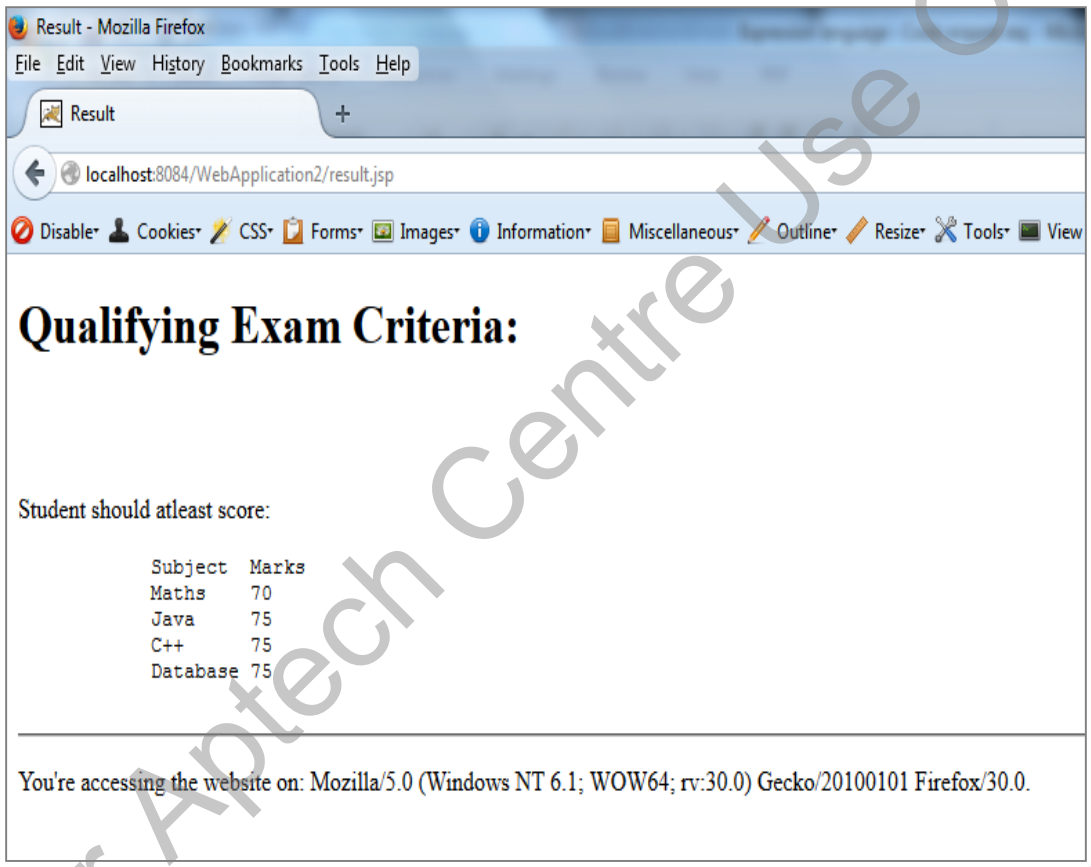
Expression Language 2-3

- The code snippet demonstrates the use of the expression language on JSP.

```
<!-- result.jsp -->

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>

<html>
. . .
<body>
        <h1>Qualifying Exam Criteria:</h1>
        <br />
        <br />
        <p>Student should atleast score:</p>
        <pre>
            Subject     Marks
            Maths       ${40+30}
            Java        ${40+35}
            C++         ${40+35}
            Database    ${40+35}
        </pre>
       <hr />
        <p>You're accessing the Website on: ${header["user-
agent"]} p>

    </body>
</html>
```

Expression Language 3-3

❖ **Output:**



Request Headers and Parameters 1-2

❖ Several implicit objects are available for the easy access through EL .

param

- Returns a value that maps a request parameter name to a single string value.
- The code snippet demonstrates the use of EL to read parameters.

```
<!-- If the request parameter name is null or an empty string,
this snippet returns true. -->

${empty param.Name}
```

paramvalues

- Returns an array of values, which is mapped to the request parameters from client.
- The code snippet demonstrates the use of EL to read parameter values from an array.

```
<!-- Takes the address line as the input -->

<br>Address Line 1: ${paramValues.address[0]}
<br>Address Line 2: ${paramValues.address[1]}
```

Request Headers and Parameters 2-2

header

- Returns a request header name and maps the value to single string value.
- The code snippet demonstrates the use of header.

```
<!--returns the host as a header name -->
${header["host"]}
```

headerValues

- Returns an array of values that is mapped to the request header.
- **Example:**

```
<!--Returns the multiple cookies with the same name -->
${headerValues.name}
```

cookie

- Returns the cookie name mapped to a single cookie object.
- **Example:**

```
<!--Returns the value of the cookie -->
${cookie.name.value}
```

initParam

- Returns a context initialization parameter name, which is mapped to a single value.

Scoped Variables 1-2

❖ The four scopes for implicit objects in JSP are as follows:

pageScope

- It returns page-scoped variable names, which are mapped to their values.
- It is accessible from the JSP page that creates the object.
- **Example:**

```
<!-- Accesses the page-scoped attribute, book-->
${pageScope.book}
```

requestScope

- It provides access to the attributes of request object.
- It returns `requestscoped` variable names, which are mapped to their values.
- It is accessible from Web components handling a request that belongs to the session.
- **Example:**

```
<!-- Returns the value of the request-scoped attribute, name.-->
${requestScope.student.name}
```

Scoped Variables 2-2

sessionScope

- It returns session-scoped variable names, which are mapped to their values.
- It is accessible from Web components handling a request that belongs to the session.
- **Example:**

```
<! -- Returns the value of the numberOfPages property of the session-scoped attribute named book. -->
${sessionScope.book.numberOfPages}
```

applicationScope

- It returns application-scoped variable and maps the variable name to their values.
- **Example:**

```
<!-- Checks the value of application-scoped attribute, booklist -->
${applicationScope.booklist == null}
```

Page Context

- The `pageContext` implicit object defines the context for the JSP page.
- It provides access to page attributes.
- It provides Web page information using the following objects:
  - **`servletContext`**
  - **`session`**
  - **`request`**
  - **`response`**

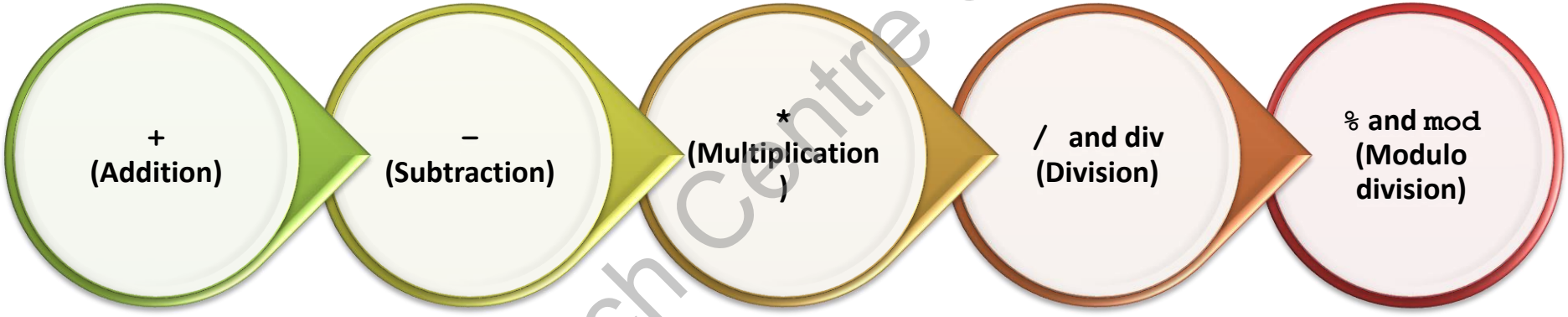
EL Operators

- EL helps in easy access of application data stored in JavaBeans components.
- Following table shows all operators used by the JSP EL.

Category	Operators
Variable	. and [ ]
Arithmetic	+, - (binary), *, / and div, % and mod, - (unary)
Conditional	A ? B : C
Relational	==, eq, !=, ne, <, lt, >, gt, <=, le, >=, ge
Logical	and, &&, or, ||, not, !
Empty/Null checking	empty

EL Arithmetic Operators 1-2

- Operators are used to perform different arithmetic, relational, and logical operations.
- Dot operator (.) or [ ] is used to access value of a variable.
- The EL supports the following arithmetic operators:



- An arithmetic statement written in JSP EL may contain more than one operator.
- The EL arithmetic operators accept strings that can be converted into numbers as parameters. Thus, the expression ${"2"+"2"} will evaluate to output 4.

EL Arithmetic Operators 2-2

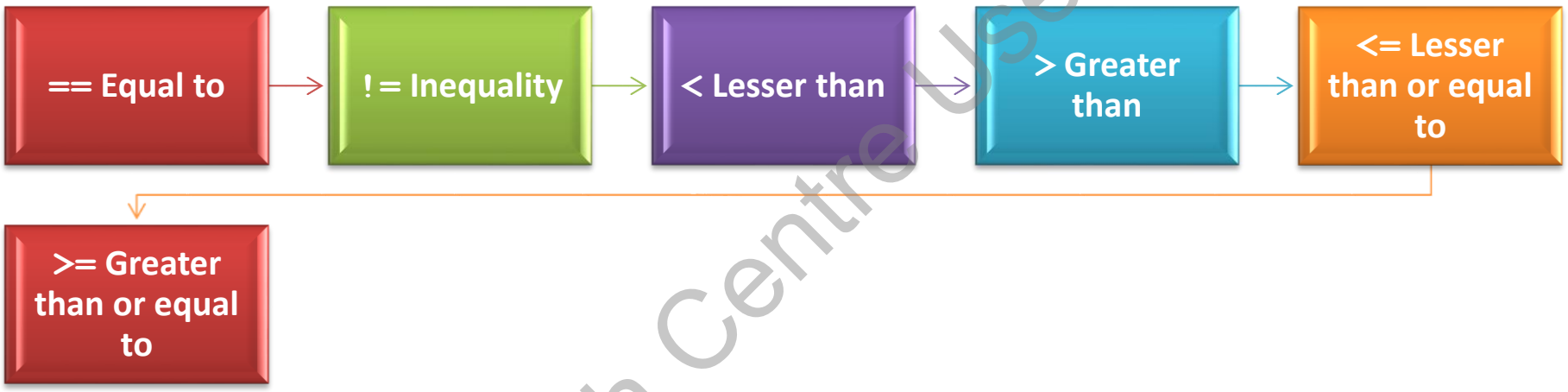
- The code snippet demonstrates the use of EL with arithmetic operators.

```
<%--Following code illustrates how you can use
the math operators of the EL --%>

<html>

${2+2}
${3-1}
${2 * 2}
${10/2}
${10 div 2}
${10 % 9}
${10 mod 9}

</html>
```

- The output of this code is : 4 2 4 5.0 5.0 1 1

EL Relational Operators

- Relational operators are used to make comparisons against other values, such as boolean, string, integer, or floating point literals.
- Various relational operators used in Expression Language are as follows:



- The code snippet demonstrates the EL with relational operators.

```
<!—comparing numbers-->
4 > '3'    ${4 > '3'}<br/>
'4' > 3    ${'4' > 3}<br/>
'4' > '3'  ${'4' > '3'} <br/>
4 >= 3     ${4 >= 3}<br/>
4 <= 3     ${4 < 3}<br/>
4 == '4'   ${4 == 4}<br/>
```

EL Logical Operators

- The logical operators supported by EL are as follows:

and, && (Logical AND)	or, || (Logical OR)	! or not (Boolean complement)

- The code snippet demonstrates the use of EL with logical operators.

```
<!--Test the condition-->

<c:if test="${(guess >= 10)  && (guess <= 20)}">
    <b>You're in the range!</b><br/>
</c:if>
<c:if test="${(guess < 10)  || (guess > 20)}">
    <b>Try again!</b><br/>
</c:if>
```

EL Empty Operators

- It is a prefix operation that can be used to determine whether the value is null or empty.
- It returns true if the string is empty.
- The string is said to be empty if it contains no character.
- If the string is not empty, then empty operator returns false.

- The code snippet demonstrates the use of empty operators.

```
<% -- this code returns true if the string is empty
and string "sometext" --%>

<b>
empty "" ${empty ""}<br/>
empty "sometext" ${empty "sometext"}<br/>
</b>
```

EL Dot Operators 1-4

- It is used to access attribute values of JavaBean and map values within the EL.
- The code to the left of the operator must specify a JavaBean or a map.
- The code to the right of the operator must specify a JavaBean or a map key.
- The property name follows the conventions of Java identifiers.
- **Syntax:**

```
${mapObject.keyName}
```

- The EL allows to replace dot notation with array notation (square brackets).
- **Example:** `${param.header}` can be replaced with `${param["header"]}`
- The advantages of array notation are as follows:
  - Can be used to access elements of an array or a list by specifying an index.
  - Allows to use values as property names.

EL Dot Operators 2-4

- The code snippet demonstrates the use of dot operator.

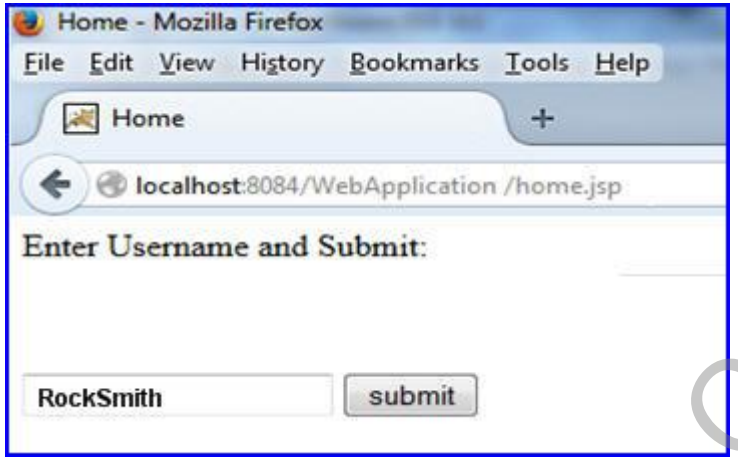
```
<!-- home.jsp -->
<html>
    <head>  . . .   </head>
    <body>
        <p>Enter Username and Submit:</p><br /><br />

          <form action="welcomeuser.jsp" method="post">

          <input name="txtval" type="text" size="20"
maxlength="60" placeholder="Enter Username..." required/>

          <input name="sbtName" type="submit" value="submit" />
       </form>

       <%
// setAttribute(name,object) binds an object to a given attribute

       application.setAttribute("PHP", "Contact Form in PHP");
       application.setAttribute("HTML5", "HTML5 new tags");
       application.setAttribute("Review", "Complete Review of all
the documents");
        %>

    </body>
</html>
```

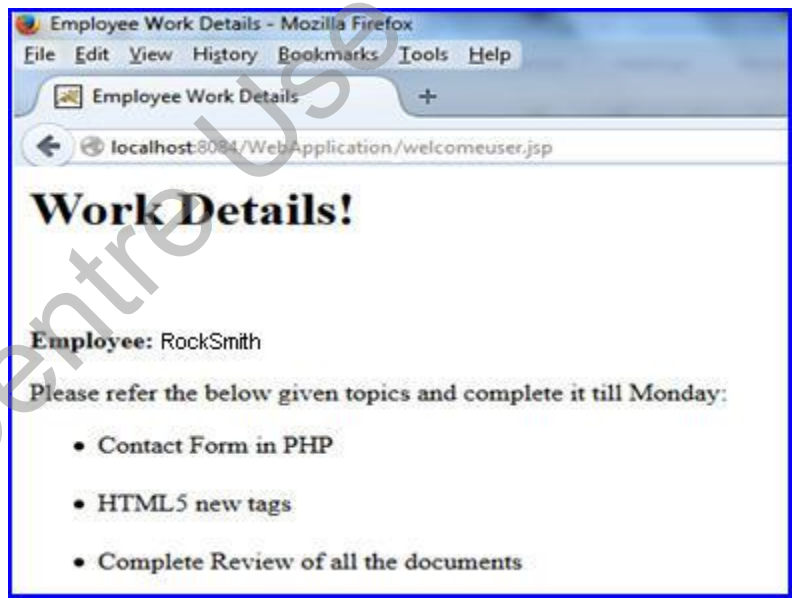
EL Dot Operators 3-4

```
<!- welcomeuser.jsp -->
<%@page contentType="text/html"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
. . .
<body>
    <h1>Work Details!</h1> <br /><br />
    <b>Employee: ${param.txtval}</b> <br />
    <p>Please refer to the given topics and
complete it till Monday:</p>

  <ul>
    <li>${applicationScope.PHP}</li> <br />
    <li>${applicationScope.HTML5}</li> <br />
    <li>${applicationScope.Review}</li> <br />
  </ul>

   </body>
</html>
```

EL Dot Operators 4-4

- Figure shows the username entered as `RockSmith` during submitting the form and the output after submitting the form successfully.





Creating 'static' Methods 1-2

- The `static` java methods can be called within the EL expression.
- To access the function using EL, the function must be implemented as a `static` function in a java class.
- A user can define many functions in a single class.
- After defining the functions, a user need to map the function name with EL using a Tag Library Descriptor (TLD) file.
- **Syntax**:

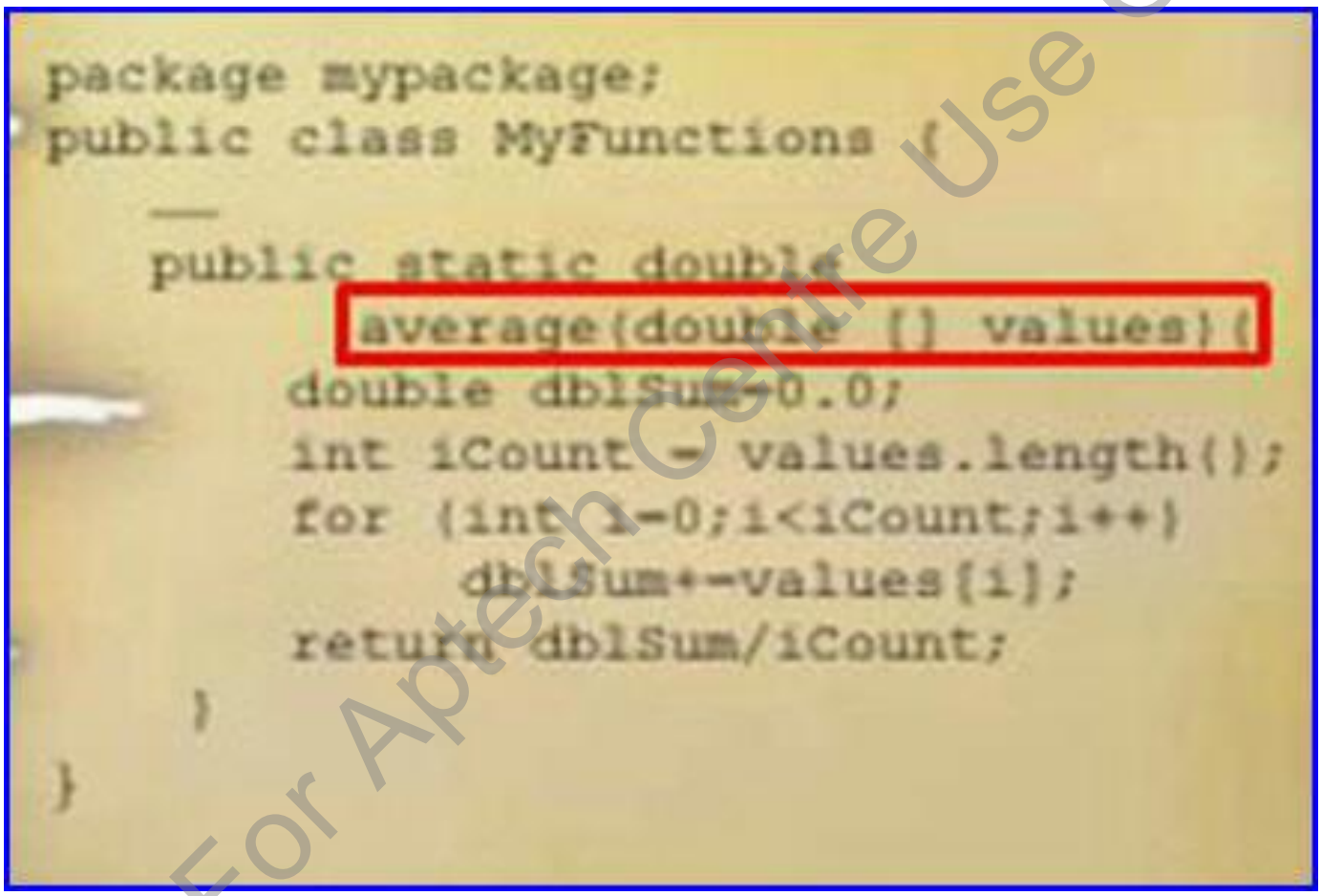
```
ns:funcName(arg1, arg2, . . .)
```

where,

- `ns` refers to name space and function name. Name space is generally a class or a tag library or a function name to access a static method in some class.

Creating 'static' Methods 2-2

- Following figure demonstrates the `static` function.



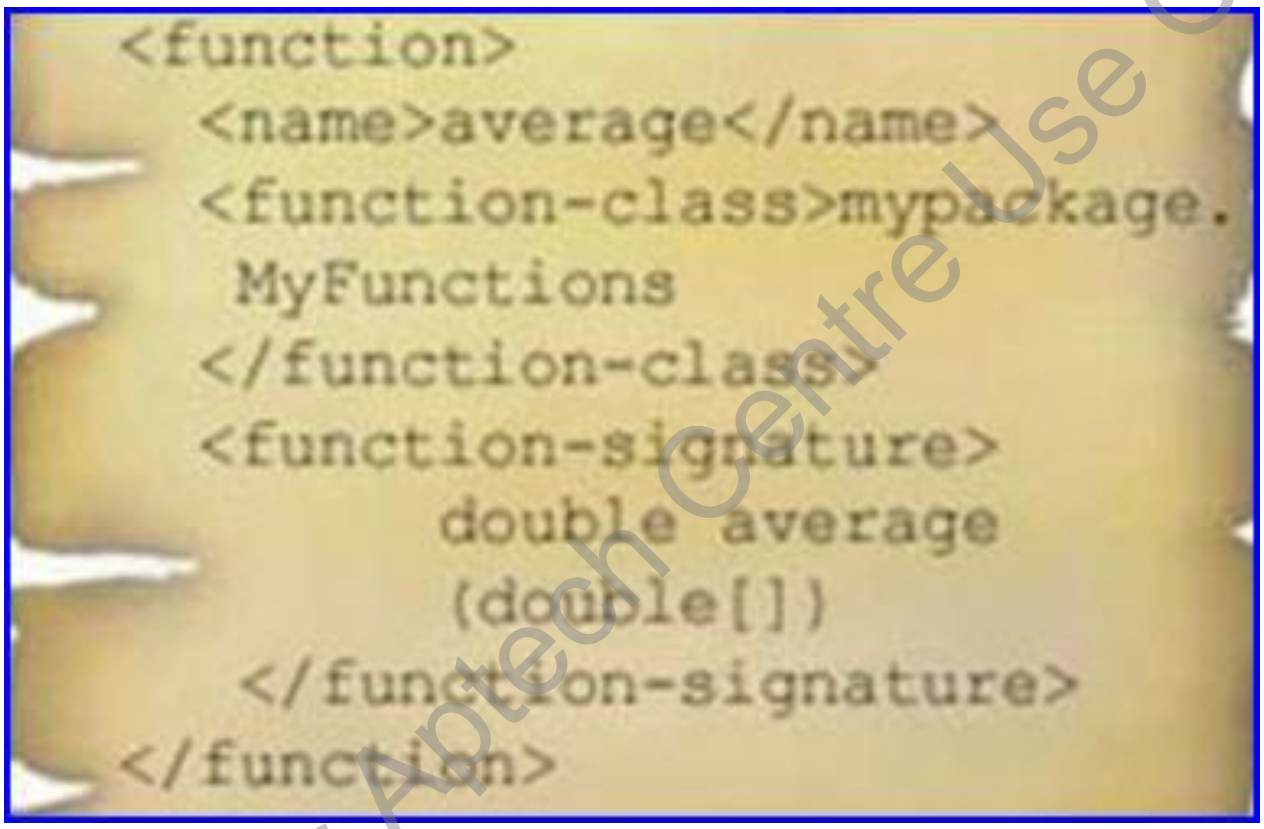
```
package mypackage;
public class MyFunctions {
    ___
    public static double
           average(double [] values){
        double dblSum=0.0;
        int iCount = values.length();
        for (int i=0;i<iCount;i++)
            dblSum+=values[i];
        return dblSum/iCount;
    }
}
```

Creating Tag Library Descriptor 1-2

- A TLD file uses XML syntax to map the name of functions defined in a class with EL.
- In the `<function>` tag, in a tag library descriptor file, user need to mention the name of the function using element `<name>`.
- A user also need to mention the class in which the function is defined using `<function-class>` element and the signature of the function in the TLD file using `<function-signature>` element.
- Then save this TLD file in the `/WEB-INF/tlds` folder, where `tlds` is a user-created folder.

Creating Tag Library Descriptor 2-2

- Following figure depicts EL function configuration in the TLD.



Modifying Deployment Descriptor 1-2

- The default mode for JSP version 1.2 technology or before is to ignore EL expressions.
- The default mode for JSP pages delivered with JSP version 2.0 technology is to evaluate EL expressions.
- Setting the value of the `<el-ignored>` element in the deployment descriptor can explicitly change the default mode.
- The `<el-ignored>` element is a sub element of `<jsp-property-group>`.
- It has no sub elements.
- Its valid values are true and false.
- In deployment descriptor, **web.xml** file, declare as follows:

Syntax:

```
<el-ignored>false</el-ignored>
```

where,

- `true` indicates that EL expressions will be ignored.
- `false` indicates that EL expressions will be enabled for interpretation by servlet container.

Modifying Deployment Descriptor 2-2

- The code snippet shows the deployment descriptor, **web.xml** which has to be modified in the following ways:

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
        . . .
    <jsp-config>
        <jsp-property-group>
            <url-pattern>*.jsp</url-pattern>
             <el-ignored>false</el-ignored>
        </jsp-property-group>
    </jsp-config>
</web-app>
```

- `<jsp-config>`: Includes JSP configuration, such as interpretation of tag library and property information.
- `<jsp-property-group>`: Defines a set of properties that applies to a set of files representing the JSP pages.
- `<url-pattern>`: Specifies that JSP properties defined in `<jsp-property-group>` to specific JSP pages, `*.jsp` indicates that these apply to all JSP pages.
- `<el-ignored>`: Enables interpretation of JSP EL in JSP pages.
- `<scripting-enabled>`: Allows JSP scripting.

Accessing EL Functions within JSP

- To access the function created in a TLD file using a JSP file, you need to import the TLD file using the `taglib` directive.
- In the directive statement, you need to mention a prefix for the tags and the location of the TLD file.
- After importing the TLD file, you can access the function using an EL expression.
- For the taglib directive, syntax declares as follows:

Syntax:

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

where,

- `prefix` is the prefix to be used for the tags defined in the TLD file.
- `path` is the location of the TLD file.

- For accessing the function:

```
<%=${prefix:functionName(arguments)}%>
```

- The code snippet shows how to access EL function in JSP.

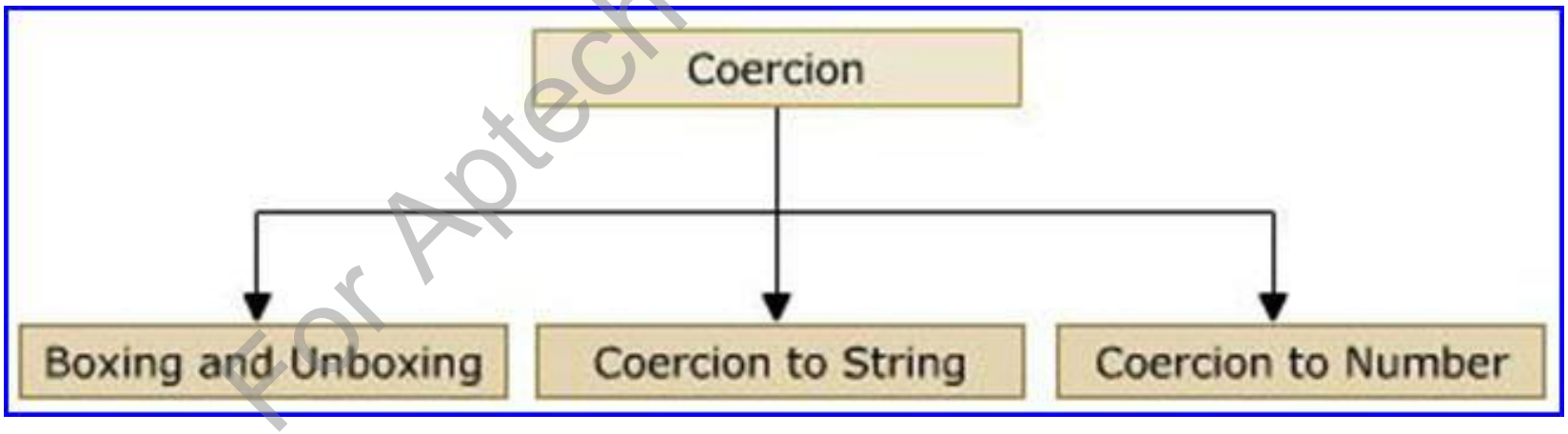
```
<%@ taglib prefix="fn" uri="/WEB-INF/tlds/functions"%>
……
Average of the values is : <%=${fn:average(values)}%>
……
```

Coercion

- Automatic conversion of a data from one data type to another data type within an expression is called Coercion.
- It occurs when the datum is stored as one data type, but its context requires a different data type.

Coercion Concept in EL

- Coercion means that the parameters are converted to the appropriate objects or primitives automatically.
- The JSTL defines appropriate conversions and default values.
- Following figure depicts types of coercion.



Boxing and Unboxing

- Boxing converts values of primitive type to corresponding values of reference type.
- Unboxing converts values of reference type to corresponding values of primitive type.

The precise rules for boxing

A) If `i` is a `boolean` value, then boxing conversion converts `i` into a reference `r` of class and type `Boolean`, such that `r.value() == i`.

B) If `i` is a `byte` value, then boxing conversion converts `i` into a reference `r` of class and type `Byte`, such that `r.value() == i`.

The precise rules for unboxing

A) If `r` is a `Boolean` reference, then unboxing conversion converts `r` into a value `v` of type `boolean`, such that `r.value() == v`.

B) If `r` is a `Byte` reference, then unboxing conversion converts `r` into a value `v` of type `byte`, such that `r.value() == v`.

C) If `r` is a `Character` reference, then unboxing conversion converts `r` into a value `v` of type `char`, such that `r.value() == v`.

Coercion to String

- The rule to coerce a value to `String` type is as follows:
  - `A` is `String`, return `A`.
  - `A` is `null`, return "" .
  - `A.toString()` throws exception, return `error`. Otherwise, return `A.toString()`.

Coercion to Number 1-2

- The rule to coerce a value to number type is as follows:
  - If `A` is `null` or "", return `0`.
- A is character and is converted to short, apply the following rules:
  - If `A` is `Boolean`, return `error`.
  - If `A` is number type, return `A`.

Coercion to Number 2-2

- A is number, coerce occurs quietly to type N using the following algorithms:
  - If `N` is `BigInteger`
  - If `A` is `BigDecimal`, return `A.toBigInteger()`
  - Otherwise, return `BigInteger.valueOf(A.longValue())`
  - If `N` is `BigDecimal`
  - If `A` is a `BigInteger`, return `new BigDecimal(A)`
  - Otherwise, return `new BigDecimal(A.doubleValue())`
  - If `N` is `Byte`, return `new Byte(A.byteValue())`
  - If `N` is `Short`, return `new Short(A.shortValue())`
  - If `N` is `Integer`, return `new Integer(A.integerValue())`
  - If `N` is `Long`, return `new Long(A.longValue())`
  - If `N` is `Float`, return `new Float(A.floatValue())`
  - If `N` is `Double`, return `new Double(A.doubleValue())`
  - Otherwise, return `error`

Summary

- EL is simple and robust. It can handle both expressions and literals, which are constants and are assigned some memory location.
- EL is a great help to the page authors in accessing and manipulating the application data without mastering the complexities of the programming language such as Java and JavaScript.
- JSP implicit objects are a standard set of classes. The user creates an instance of an implicit object to use available methods and variables.
- Operators are used to perform different arithmetic, relational, and logical operations. Dot operator (.) or [] is used to access value of a variable. Various operators used in Expression Language are arithmetic operators, relational operators, logical operators, empty operators, and dot operators.
- In Expression language, the static java methods can be called within the EL expression. To access the function using EL, the function must be implemented as a static function in a java class.
- The TLD file uses XML syntax to map the name of functions defined in a class with EL. Setting the value of the <el-ignored> element in the deployment descriptor can explicitly change the default mode.
- The accessing of the function created in a TLD file using a JSP file is possible by importing the TLD file using the taglib directive.
- Coercion means that the parameters are converted to the appropriate objects or primitives automatically. Coercion is an implicit type conversion.