Java Server Pages (JSP)

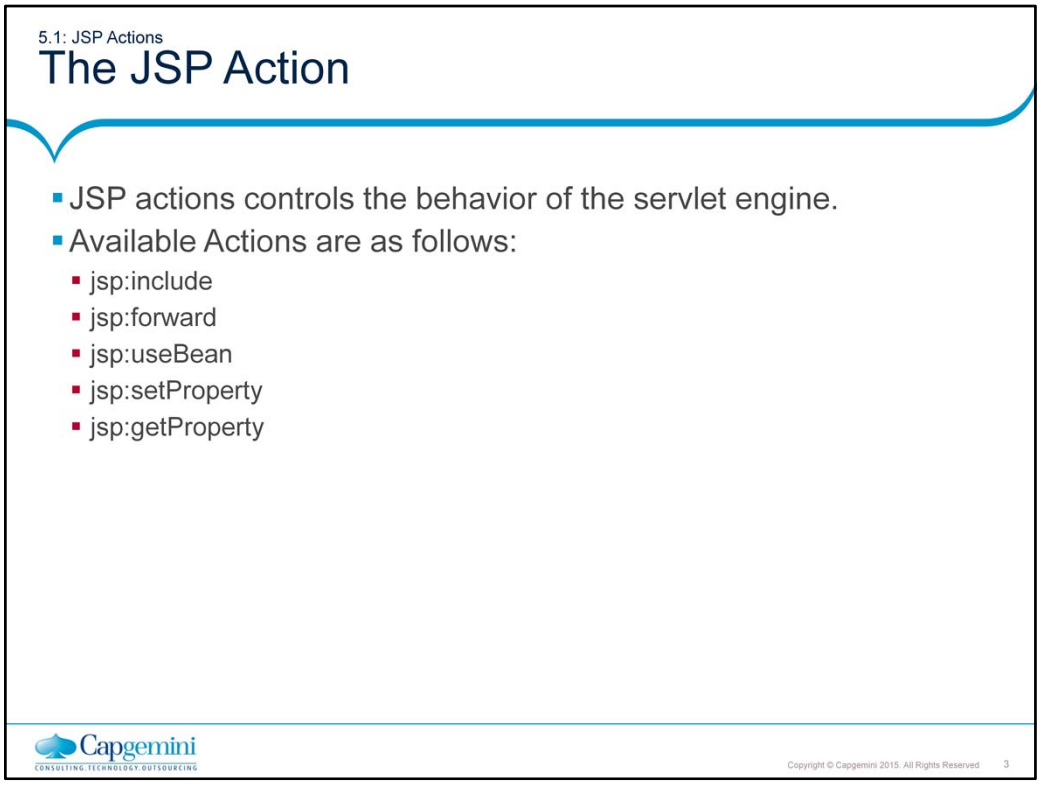
Lesson 05 : JSP Actions

Lesson Objectives

- In this lesson, you will learn the following concepts:
  - JSP Actions
  - jsp:include Action
    - include Action Vs Directive
  - jsp:forward Action
    - Integrating Servlets & JSP
    - jsp:forward Vs response.sendRedirect
  - Java Bean
  - Bean Related JSP Actions

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2



JSP Actions:

What is a JSP Action?

- **JSP actions** use constructs in XML syntax to control the behavior of the servlet engine. With JSP actions, following actions can be performed:
  - Dynamically insert a file
  - Reuse JavaBeans components
  - Forward the user to another page
  - Generate HTML for the Java plugin
- Available actions include:
  - **jsp:include** : It includes a file at the time the page is requested.
  - **jsp:forward :** It forwards a client request to an HTML file, JSP file, or servlet for processing.
  - **jsp:useBean** : It finds or instantiates a JavaBean.
  - **jsp:setProperty** : It sets the property of a JavaBean.
  - **jsp:getProperty** : It inserts the property of a JavaBean into the output.

- The above actions are described in more detail further in this lesson.

5.2: jsp:include Action

The Include Action

- The include action allows to insert files into the page being generated.
- Syntax:

```
<jsp:include page="{relativeURL | <%= expression%>}"   flush="true"/>
```

or

```
<jsp:include page="{relativeURL | <%= expression %>}"   flush="true" >
<jsp:param name="parameterName"
value="{parameterValue | <%= expression %>}" />+
 </jsp:include>
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

 4

The jsp:include Action:

- The **<jsp:include>** element allows to include either a **static** or **dynamic** file in a JSP file. The results of including static and dynamic files are quite different.
  - If the file is static, its content is included in the calling JSP file.
  - If the file is dynamic, it acts on a request and sends back a result that is included in the JSP page.
- When the include action is finished, the JSP container continues processing the remainder of the JSP file.
- We cannot always determine from a pathname if a file is static or dynamic. For example: http://server:8080/index.html might map to a dynamic servlet through a Web server alias. The **<jsp:include>** element handles both types of files, so it is convenient to use when we do not know whether the file is static or dynamic.
- If the included file is dynamic, we can use a **<jsp:param>** clause to pass the name and value of a parameter to the dynamic file. As an example, we can pass the string username and a user's name to a login form that is coded in a JSP file.
- The above slide lists the syntax of **jsp:include** action. The value for the page attribute is a **relative URL** that locates the file to be included, or an **expression** that evaluates to a String equivalent to the relative URL. The relative URL looks like a pathname - it cannot contain a protocol name, port number, or domain name. The URL can be absolute or relative to the current JSP file. If it is absolute (beginning with a /), then the pathname is resolved by Web or application server.
- We must include the attribute **flush="true"**, as it is not a default value. We cannot use a value of false. Use the **flush** attribute exactly as shown in the slide.

5.2: jsp:include Action

Examples for jsp:include

- Example 1:

```
<jsp:include page="scripts/login.jsp" />
    <jsp:include page="copyright.html" />
    <jsp:include page="/index.html" />
```

- Example 2:

```
<jsp:include page="scripts/login.jsp">
    <jsp:param name="username" value="jsmith" />
  </jsp:include>
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 5

<u>The jsp:include Action</u>:

Examples for jsp:include:

- The above slide lists some examples of **jsp:include** action with the page attribute value as a relative URL.
- In the second example on the slide, the **<jsp:param>** clause is used which allows to pass one or more name / value pairs as parameters to an included file. The included file should be dynamic, that is a JSP file, servlet, or other file that can process the parameter.
- We can use more than one **<jsp:param>** clause if we want to send more than one parameter to the included file. The name attribute specifies the parameter name and takes a case-sensitive literal string. The value attribute specifies the parameter value and takes either a case-sensitive literal string or an expression that is evaluated at request time.

5.2: jsp:include Action

Demo

- Demo on:
  - jspIncludeAction.jsp
  - header.jsp
  - footer.jsp

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 6

Deploy web application **Lesson5-JSPActions** and show demo by executing each of the above JSP pages.

Note: Here is an example, where a JSP page uses jsp:include action to inserts header and footer into jspIncludeAction.jsp

5.2.1: include Action Vs Directive

include Action versus include Directive

- The include directive includes files at the time the JSP page is translated into a servlet. Hence it is called static include.
  - The include action includes files when the page is requested. Hence it is called dynamic include.
  - In case of include directive as the include process is static, a change in the included file will not be reflected in the JSP file in which it is included (this behavior is dependent on JSP container).

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

 7

The jsp:include Action:

The JSP Include Action versus JSP Include Directive:

- Unlike the **include directive**, which inserts the file at the time the JSP page is "translated" into a servlet, the **include action** inserts the file at the time the page is "requested". This pays a small penalty in efficiency, and precludes the included page from containing general JSP code (for example, it cannot set HTTP headers), but it gains significantly in flexibility.
- A JSP container can include a mechanism for being notified if an included file changes (in case of include directive), so the container can recompile the JSP page. However, the JSP 2.2 specification does not have a way of directing the JSP container that included files that have changed.
- In the recent versions of the **Server containers** (such as JBOSS, Tomcat, Websphere Application server), this mechanism has been implemented. Hence there is no change in the behavior of **include directive** and **include action**. Both include the files dynamically, hence change in the included file is reflected in the original page in both the cases. However, it is recommended that we should not rely on the container's behavior and always use **include action** whenever a **dynamic include** is required.

5.3: jsp:forward Action

The forward Action

- The <jsp:forward> element forwards the request object containing the client request information from one JSP file to another file.
- The target file can be an HTML file, another JSP file, or a servlet.
- A jsp:forward effectively terminates the execution of the current page.
- If the page output is buffered, then the buffer is cleared prior to forwarding.

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

 8

The jsp:forward Action:

- The **<jsp:forward>** element forwards the **request object** containing the client request information from one JSP file to another file. The target file can be an HTML file, another JSP file, or a servlet, as long as it is in the same application context as the forwarding JSP file. The lines in the source JSP file after the **<jsp:forward>** element are not processed.
- We can pass parameter names and values to the target file by using a **<jsp:param>** clause. An example of this would be passing the parameter name ***username*** (with name="username") and the value ***scott*** (with value="scott") to a jsp login file as part of the request. If we use **<jsp:param>**, then the target file should be a dynamic file that can handle the parameters.
- Be careful while using **<jsp:forward>** with unbuffered output. If the **<%@ page %>** directive has used with **buffer=none** to specify that the output of the JSP file should not be buffered, and if the JSP file has any data in the out object, using **<jsp:forward>** will cause an IllegalStateException.

5.3: jsp:forward Action

Syntax and Examples for jsp:forward

- Syntax:

```
<jsp: forward page="{ relativeURL | <%= expression %> }"
    {/>|> [ <jsp:param name=" parameterName " value="{
parameterValue | <%= expression %>}" /> ]+
    </jsp: forward>
```

- Example 1:

```
<jsp:forward page="/pages/login.jsp" />
```

- Example 2:

```
<jsp:forward page="/pages/login.jsp">
    <jsp:param name="username" value="jsmith" />
</jsp:forward>
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

 9

<u>The jsp:forward Action</u>:

The above slide lists the syntax for **jsp:forward** action:

- Like **jsp:include**, the value for the page attribute in case of jsp:forward is a **relative URL** or an **expression** representing the file to which we are forwarding the request. The file can be another JSP file, a servlet, or any other dynamic file that can handle a request object.
- The **jsp:param** element is similar to the one that we saw for jsp:include action.

Examples:

- The above slide shows examples of **jsp:forward** action.
    - The first example depicts how a request can be forwarded to another jsp page.
    - In the second example, the request is forwarded to another jsp and a request parameter “username” is also passed to the forwarded page. In the **login.jsp** page, the username request parameter can be obtained by using the following method:
        - request .getParameter (“username”)

5.3: jsp:forward Action

Demo

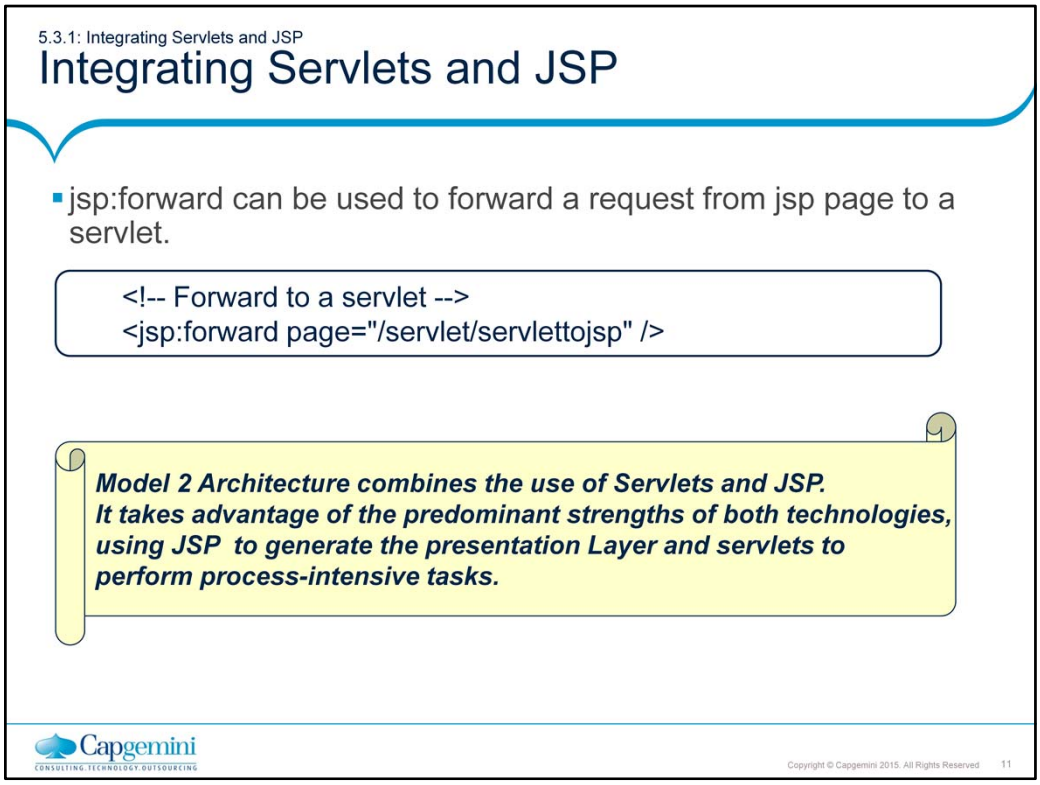
- Demo on:
  - first.jsp
  - second.jsp

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10

Deploy web application **Lesson5-JSPActions a**nd show demo by executing each of the above JSP pages in **forward** folder.

```
<html>
 <body bgcolor="white">
 <font color="red"> VM Memory usage > 50%.
 </html>
```



<u>The jsp:forward Action</u>:

Integrating Servlets and JSP:

- The Model 2 architecture, discussed in lesson 1, is a hybrid approach for serving dynamic content, since it combines the use of both servlets and JSP. It takes advantage of the predominant strengths of both technologies, using JSP to generate the presentation layer and servlets to perform **process-intensive tasks**. On a more primitive level, we have seen how to integrate JSPs and servlets together using the **jsp:forward** tag.
- **Example:** A request to **jsptoservlet.jsp** forwards the request to the **servletToJsp** servlet which in turn sets the attribute in the request to contain the servlet name and obtains the **RequestDispatcher** to **hello.jsp** page and forwards the request using **RequestDispatcher**. Upon receiving the request, the **hello.jsp** shows the servlet name.
- The above slide shows the code fragment from the **jsptoservlet.jsp** file. The next slide shows the code listing for the **servlet** and **hello.jsp** files.

5.3.1: Integrating Servlets and JSP

Demo: Integrating Servlets and JSP

- Demo on:
  - servletToJsp.java (servlet)

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 12

Note: Following is the code fragment from servletToJsp servlet which forwards the request and response to hello.jsp page.

```
public void doGet (HttpServletRequest request,
                   HttpServletResponse response) {
try {
        // Set the attribute and Forward to hello.jsp
      request.setAttribute ("servletName", "servletToJsp");
      getServletConfig().getServletContext().getRequestDispatcher(
                "/pages/hello.jsp").forward(request, response);
     } catch (Exception ex) {
                ex.printStackTrace ();
     } }
```

The following example shows a code fragment from **hello.jsp**, which displays the servlet name passed by **servletToJsp** servlet.

```
<body bgcolor="white">
<h1> I have been invoked by
<% out.print (request.getAttribute("servletName").toString()); %>
Servlet. </h1>
```

5.3.1: jsp:forward versus response.sendRedirect

jsp:forward versus response.sendRedirect

jsp:forward	response.sendRedirect
Server-side redirect, hence no network traffic	Client-side redirect, hence additional network round trip
The address of the destination page hidden from user	The address of the destination page visible to user in the address bar
Allows forwarding of the request to another page in the same context as the current page.	Allows re-direction of the request to another page in same or different context as the current page

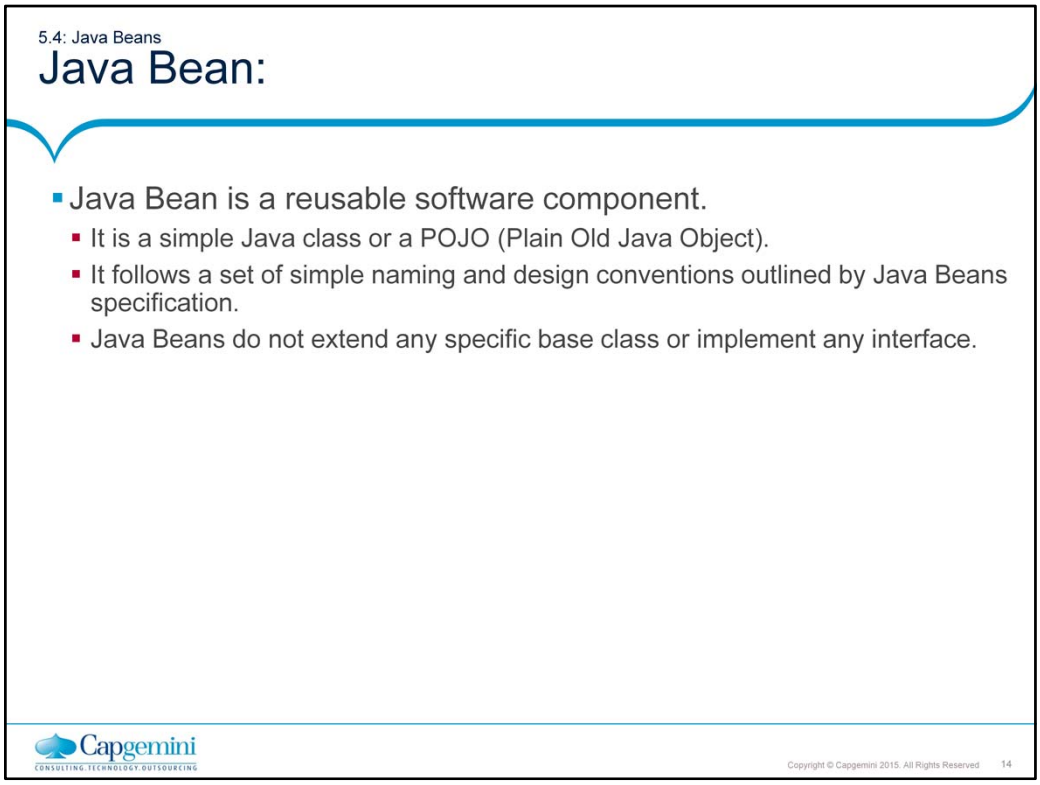
Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

 13

The jsp:forward Action:

jsp:forward versus response.sendRedirect:

- **forward** is server side redirect and **sendRedirect** is client side redirect. When we invoke a forward request, the request is sent to another resource on the server, without the client being informed that a different resource is going to process the request. This process occurs completely within the web container, and then returns to the calling method. When a **sendRedirect** method is invoked, it causes the web container to return to the browser indicating that a new URL should be requested. Since the browser issues a completely new request, any objects that are stored as **request attributes** before the redirect occurs will be lost. As a result of this extra round trip, a redirect is slower than forward. Client can disable sendRedirect.
- In case of **sendRedirect**, the user sees only the address of the destination page and can bookmark it and access it independently.
- In case of forward, the request can be forwarded to another page in the same context/domain as the current page whereas in case of **sendRedirect** we can redirect the request to a page in another context or domain as well. for e.g. if **sendRedirect** was called at www.mydomain.com then it can also be used to redirect a call to a resource on www.yourdomain.com.



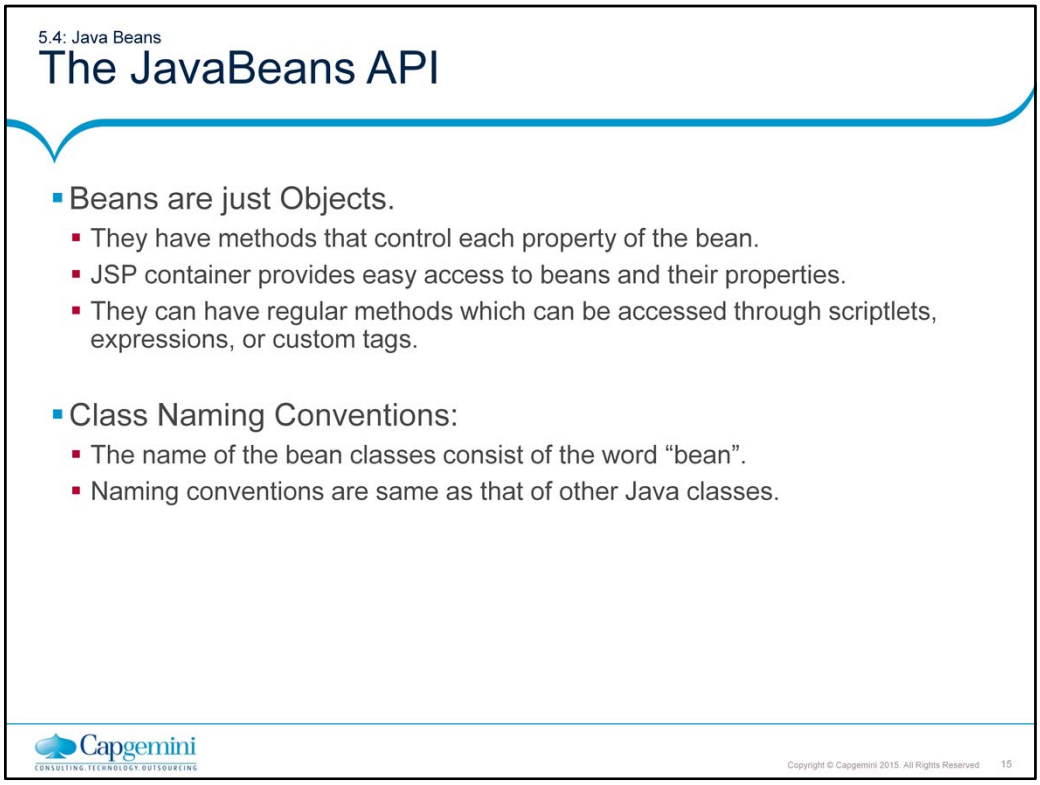
<u>Java Beans</u>:

Building own JSP components:

- The **JavaServer Pages (JSP)** component model is centered on Java software components called **Beans**, which must adhere to specifications outlined in the **JavaBeans API**. The JavaBeans API, created by Sun Microsystems with industry cooperation, dictates the rules that software developers must follow to create stand-alone, reusable Java software components. By using JSP's collection of Bean tags, content developers can use the power of Java to add dynamic elements to their pages without writing a single line of code.

So, what makes a Bean so special?

- A Bean is simply a **Java class** or a **POJO** that follows a set of simple naming and design conventions outlined by the JavaBeans specification. Beans are not required to extend a specific base class or implement a particular interface. If a class follows the Bean conventions, and we treat it like a Bean, then it is a Bean. A particular, good feature of the Bean conventions is that they are rooted in sound programming practices. In the next section, we will discuss these conventions and show how to create Beans.



<u>Java Beans</u>:

The JavaBeans API:

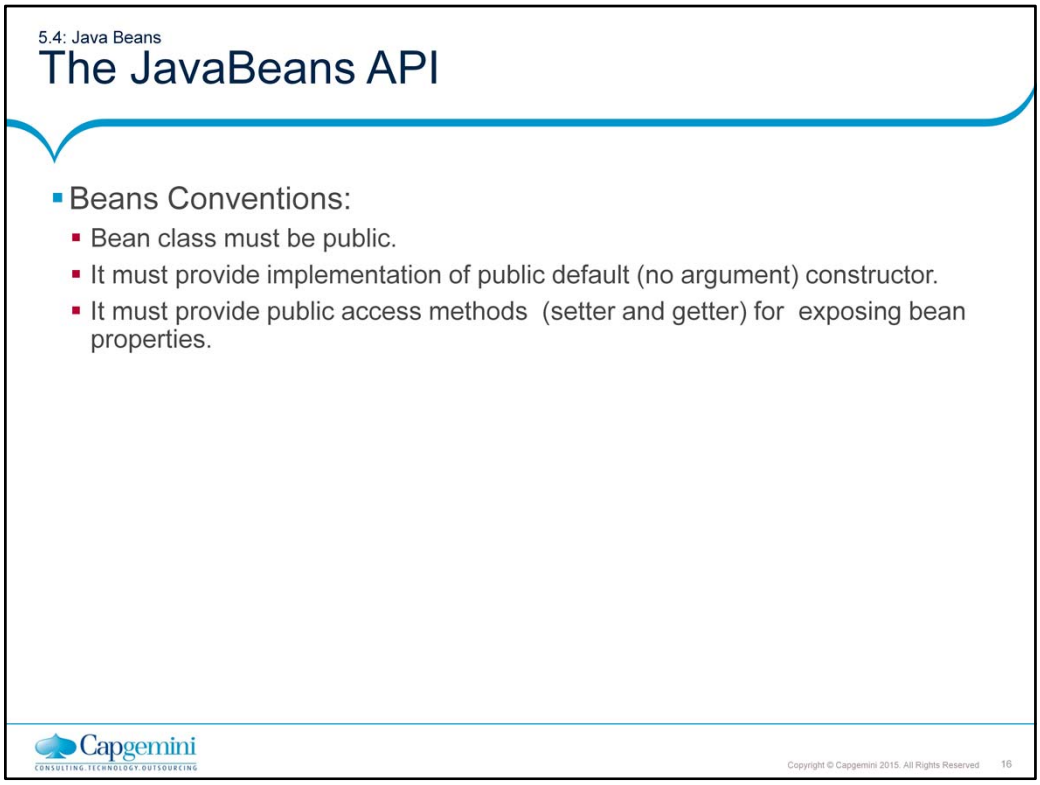
- If we follow the conventions specified by the **JavaBeans API**, then the JSP container can interact with **Beans** at a programmatic level, even though the container has no real understanding of what the Bean does or how it works.

Bean Are Just Objects:

- As with any other Java class, instances of Bean classes are simply Java objects. Hence, we can reference Beans and their methods directly through Java code in other classes or through JSP scripting elements. Since they follow the Bean conventions, we can work with Beans without having to write Java code. Bean containers, such as a JSP container, can provide easy access to Beans and their properties. Following the JavaBeans API coding conventions, means creating methods that control access to each property we wish to define for our Bean. Beans can also have regular methods like any other Java object which can be accessed through scriptlets, expressions, or custom tags.

Class Naming Conventions:

- Note that in most examples Bean classes often include the word Bean in their name, such as UserBean, ClockBean, DataAccessBean. This is not a rule hence not mandatory. Beans follow the same class naming rules as other Java classes. They must start with an alphabetic character, contain only alphanumeric and underscore characters, and be case-sensitive. Additionally, like other Java classes, it is common (but not required) to start the name of a Bean class with a capital letter.



<u>Java Beans</u>:

Bean Conventions:

- The **Bean conventions** enable us to develop Beans, because they allow a Bean container to analyze a Java class file and interpret its methods as properties, designating the class as a Bean. The conventions dictate rules for defining a Bean's constructor and the methods that will define its properties.

The Bean Constructor:

- The first rule of JSP Bean building is that we must implement a constructor that takes no arguments. This constructor is used by the JSP container to instantiate the Bean through the **<jsp:useBean>** tag. If a class does not explicitly specify any constructors, then a default zero-argument constructor is assumed. As a result of this default constructor rule, the following Java class is perfectly valid and technically satisfies the Bean conventions:
  - *public class DoNothingBean { }*

The Magic of Introspection:

- The **Bean container** can interact with the bean through introspection. This introspection process happens at run-time and allows a class to expose its properties on request. One way in which introspection can occur is through "Reflection". The Bean container determines the properties that a Bean supports by analyzing its public methods for the presence of **property access methods** that meet criteria defined by the JavaBeans API.
- For a property to exist, its Bean class must define an **access method** to return the value of the property, change the value of the property, or both. It is the presence alone of the specially named **access methods** that determines a Bean class' properties.

5.4: Java Beans

Example of JavaBeans

- Let us see an example on JavaBeans:

```
public class CurrentTimeBean {
    private int hours, minutes;
    public CurrentTimeBean( ) {
        Calendar now = Calendar.getInstance();
        this.hours = now.get (Calendar.HOUR_OF_DAY );
        this.minutes = now.get (Calendar.MINUTE );   }
    public int getHours() {  return hours;   }
    public int getMinutes() {  return minutes;   }
  }
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

 17

<u>Java Beans</u>:

Example of Java Bean:

- The example on the slide shows how a **CurrentTimeBean** is coded. This bean follows all the conventions. It consists of a public **no-argument constructor** that initializes the value for the private instance variables hours and minutes. It has the public access methods for two properties, namely **hours** and **minutes**. Since it has only the **getter** methods for these properties, they will be exposed as read-only properties.
- The two methods simply return the appropriate values as stored in the instance variables. Since these methods meet the Bean conventions for naming access methods, we have just defined two properties that we can access through JSP Bean tags.
- Properties should not be confused with instance variables, even though instance variables are often mapped directly to property names. Properties of a Bean are not required to correspond directly with instance variables. A Bean's properties are defined by the method names themselves, not the variables or implementation behind them. This leaves the Bean designer free to alter the inner workings of the Bean without altering the interface and collection of properties exposed to users of the Bean.

5.5: Bean Related Actions

Introducing useBean and getProperty Action

- Consider the following bean:

```
<HTML>
<BODY>
 <jsp:useBean id="time" class="beans.CurrentTimeBean"/>
```

- It is now:

```
<jsp:getProperty name="time" property="minutes"/>  minutes
past the hour.
</BODY>
</HTML>
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 18

<u>Bean Related Actions</u>:

Accessing the Java Bean and its property in the JSP page:

- The code shown in the above slide depicts how we can use the **jsp:useBean** action to create an instance of the **CurrentTimeBean** Java bean.
- It uses **jsp:getProperty** action to access the minutes properties. Internally the bean container will invoke the **getMinutes()** method of the **CurrentTimeBean** to retrieve the property value as a string.

5.5: Bean Related Actions

Actions Performed by jsp:useBean Action

- The jsp:useBean action attempts to locate a Bean instance within the scope and name specified.
  - It defines a variable with the name specified in id.
  - If the instance is found, then its reference is stored in the variable. If type specified, then it casts the object to the type given.
  - If it does not find the instance, then it creates an instance of the class specified in class, storing a reference to it in new variable.
  - If it instantiates (but does not locate) a JavaBean and has body tags, then it executes the body tags.

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

 19

Bean Related Actions:

The <jsp:useBean> Action

- A **<jsp:useBean>** action associates an instance of a Java object defined within a given scope available with a given id via a newly declared scripting variable of the same id.
- The action tries to find an existing object using id and scope. If it is not found, then it will attempt to create the object using the other attributes.

The actions performed are:

- It attempts to locate an object based on the attribute values (id, scope).
- It defines a scripting language variable with the given id in the current **lexical scope** of the scripting language of the specified type (if given) or class (if type is not given).
- If the object is found, then the variable's value is initialized with a reference to the located object, cast to the specified type. If the cast fails, then a **java.lang.ClassCastException** occurs. This completes the processing of this **useBean** action.
- If the **jsp:useBean** element has a non-empty body, then it is ignored. This completes the processing of this **useBean** action.
- If the object is not found in the specified scope and neither class nor beanName are given, then a **java.lang.InstantiationException** occurs. This completes the processing of this **useBean** action.

5.5: Bean Related Actions

Syntax for jsp:useBean Action

- Given below is the syntax for jsp:useBean Action:

```
<jsp:useBean     id="beanInstanceName" scope="page |
   request | session | application"

      class="package.class"| type="package.class" |
      class="package.class" type="package.class" |
      beanName="{package.class | <%= expression   %>}"
     type="package.class"
     /> |
        > other elements
  </jsp:useBean>
```

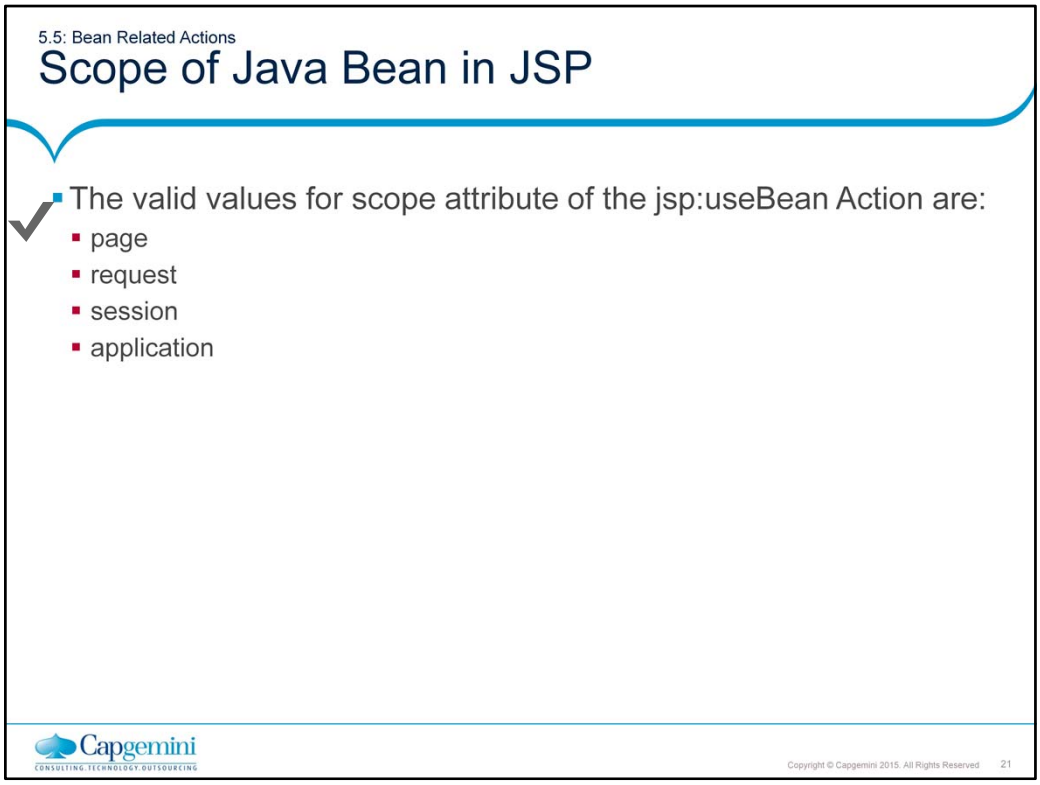
Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

 20

Bean Related Actions:

The Jsp:useBean Action - Attributes and Usage:

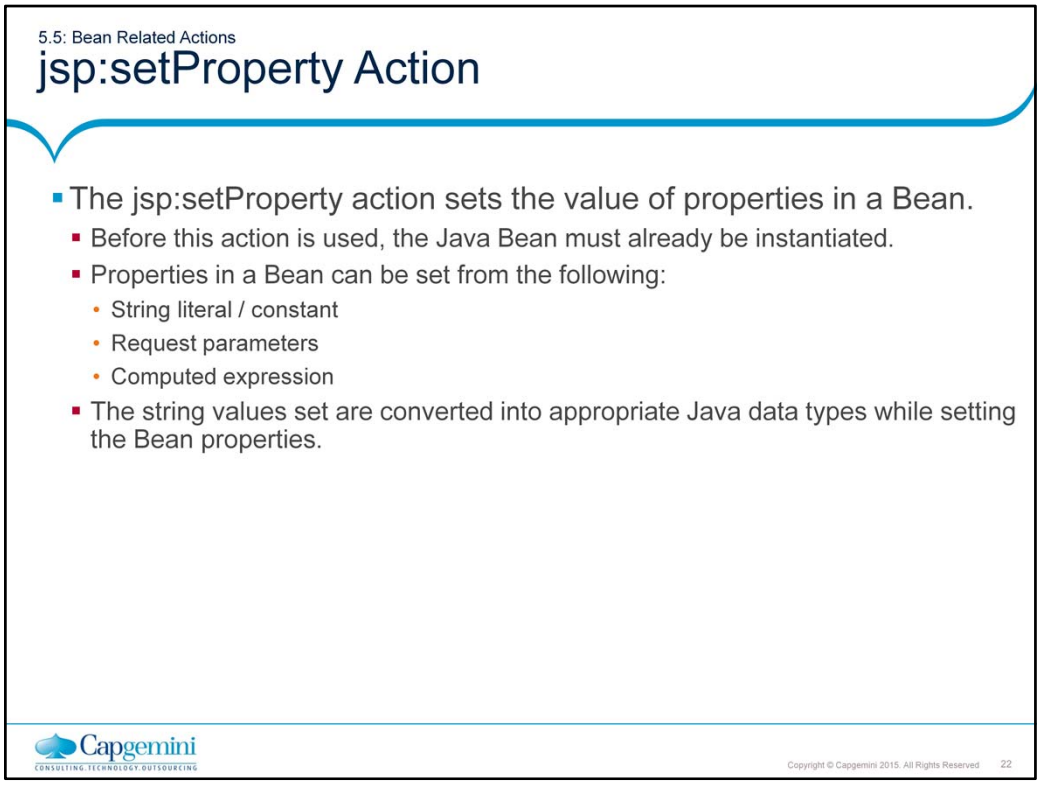
- **id="beanInstanceName"** : It is a variable that identifies the Bean in the scope specified. The variable name can be used in expressions or scriptlets in the JSP file. The name is case sensitive and must conform to the naming conventions of the scripting language used in the JSP page. If the Bean has already been created by another <jsp:useBean> element, then the value of id must match the value of id used in the original <jsp:useBean> element.
- **class="package.class“:** It instantiates a Bean from a class, using the “**new**” keyword and the **class constructor**. The class must not be abstract and must have a public, no-argument constructor.
- **type="package.class"** : If the Bean already exists in the scope, then it gives the Bean a data type other than the class from which it was instantiated. If we use type without class or beanName, then no Bean is instantiated.
- **class="package.class" type="package.class"** : It instantiates a Bean from the class named in class, and assigns the Bean the data type specified in type. The value of type can be the same as class, a superclass of class, or an interface implemented by class.
- **beanName="{package.class | <%= expression %>}" type="package.class"** : It instantiates a Bean from either a class or a serialized template, using the **java.beans.Beans.instantiate** method, and gives the Bean the type specified in type.
- **scope**: This attribute is explained in the next slide.



<u>Bean Related Actions</u>:

The Jsp:useBean Action - Attributes and Usage:

- scope="**page** | request | session | application" : It indicates the scope in which the Bean exists and the variable named in id is available. The default value is page. The meanings of the different scopes are as follows:
    - **page:** In this scope the Bean can be used within the JSP page with the **<jsp:useBean>** element or any of the page's static include files, until the page sends a response back to the client or forwards a request to another file.
    - **request:** In this scope the Bean can be used from any JSP page processing the same request, until a JSP page sends a response to the client or forwards the request to another file. We can use the request object to access the Bean.
      For example: request.getAttribute(*beanInstanceName*)
    - **session:** In this scope the Bean can be used from any JSP page in the same session as the JSP page that created the Bean. The Bean exists across the entire session, and any page that participates in the session can use it. The page in which the Bean is created must have a **<%@ page %>** directive with **session=true**.
    - **application:** In this scope the Bean can be used from any JSP page in the same application as the JSP page that created the Bean. The Bean exists across an entire JSP application, and any page in the application can use the Bean.



<u>Bean Related Actions</u>:

The jsp:setProperty Action:

- The **jsp:setProperty action** sets the value of properties in a Bean. The **name** attribute denotes an object that must be defined before this action appears.
- There are two variants of the **jsp:setProperty action**. Both variants set the values of one or more properties in the Bean based on the type of the properties. The usual Bean introspection is done to discover what properties are present, and for each property, its name, whether they are simple or indexed, their type, and setter and getter methods are determined.
- Properties in a Bean can be set from either of the following:
  - one or more parameters in the request object
  - a String constant
  - a computed request-time expression
- Simple and indexed properties can be set using **setProperty**.
- The string values set are converted into appropriate Java data type while setting the Bean properties using its setter method. The conversion applied is listed in subsequent slide.
- While assigning values to **indexed properties**, the value must be an **array**.

5.5: Bean Related Actions

Syntax for jsp:setProperty Action

- Given below is the syntax for jsp:setProperty Action:

```
<jsp:setProperty name="beanInstanceName"
        property= "*"   |
        property="propertyName" [
                        param="parameterName"
  ]  |
        property="propertyName" value="{string | <%=
                expression %>}"
  />
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

 23

Bean Related Actions:

The jsp:setProperty Action - Attributes and Usage

- **name="beanInstanceName"**: It indicates the name of an instance of a Bean that has already been created or located with a <jsp:useBean> element.
- **property="*"** : It stores all the values that the user enters in the viewable JSP page (called **request parameters**) in matching Bean properties. The names of the properties in the Bean must match the names of the **request parameters**, which are usually the elements of an HTML form. If a request parameter has an empty or null value, then the corresponding Bean property is not set. Likewise, if the Bean has a property that does not have a matching request parameter, then the property value is not set.
- **property="propertyName" [param="parameterName"]** : It sets one Bean property to the value of one request parameter. In the syntax, **property** specifies the name of the Bean property and **param** specifies the name of the request parameter by which data is being sent from the client to the server. If the Bean property and the request parameter have different names, then we must specify both property and param. If they have the same name, then we can specify **property** and omit **param**. If a parameter has an empty or null value, then the corresponding Bean property is not set.
- **property="propertyName" value="{string | <%= expression %>}"** : It sets one Bean property to a specific value. The value can be a String or an expression that is evaluated at runtime. We cannot use both the param and value attributes in a <jsp:setProperty> element.

5.5: Bean Related Actions

Type Conversion for setProperty Action

Property Type	Conversion from String
boolean or Boolean	java.lang.Boolean.valueOf (String)
byte or Byte	java.lang.Byte.valueOf (String)
char or Character	java.lang.Character.valueOf (String)
double or Double	java.lang.Double.valueOf (String)
int or Integer	java.lang.Integer.valueOf (String)
float or Float	java.lang.Float.valueOf (String)
long or Long	java.lang.Long.valueOf (String)

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

 24

Bean Related Actions:

Type Conversion for <jsp:setProperty> Action:

- All property setter methods accessed with a **<jsp:setProperty>** tag will be automatically converted from a String to the appropriate native type by the JSP container. This is accomplished via methods of Java's wrapper classes, as shown in the table in the above slide.
- A conversion failure leads to an error. The error may be at **translation** or at **request-time**.

5.5: Bean Related Actions

jsp:getProperty Action

- The jsp:getProperty element retrieves the value of a bean property, converts it to a string, and inserts it into the output .
- Syntax:

```
<jsp:getProperty name="beanInstanceName" property="
propertyName"/>
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 25

Bean Related Actions:
The jsp:getProperty Action:
The jsp:getProperty element retrieves the value of a bean property, converts it to a string, and inserts it into the output. The two required attributes are:

name: the name of a bean previously referenced via jsp:useBean
property: the property whose value should be inserted

The syntax is very simple and is listed in the above slide. We will need to specify the Bean instance name (should match with the id specified in the jsp:useBean action) and the property name.
Internally this action invokes the getter method of the specified property and converts the return value into string.

5.5: Bean Related Actions

Type Conversion for getProperty Action

Property Type	Conversion to String
boolean	java.lang.Boolean.toString (boolean)
byte	java.lang.Byte.toString (byte)
char	java.lang.Character.toString (char)
double	java.lang.Double.toString (double)
int	java.lang.Integer.toString (int)
float	java.lang.Float.toString (float)
long	java.lang.Long.toString (long)

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

 26

Bean Related Actions:

Type Conversion for jsp:getProperty Action

- A JSP component's properties are not limited to string values, but it is important to understand that all property values accessed through the **<jsp:getProperty>** tag will be converted to strings. A **getter** method need not return a String explicitly. However, the JSP container will automatically convert the return value to a String as needed.
- For the Java primitive types, conversion is handled by the methods shown in Table in the above slide.
- Properties are not restricted to primitive types either. For objects, the JSP container will invoke the object's **toString()** method, which unless overloaded it in the class, will probably not be very representative of the data stored in the object.
- We can also overload **getter** and **setter** methods to accept the appropriate object type, although custom tags or JSP scripting elements will be required to access the overloaded methods, since the <jsp:setProperty> and <jsp:getProperty> tags work exclusively with String values.

5.5: Bean Related Actions

Configuring Beans

- Following are the approaches for configuring Beans:
  - Use jsp:setProperty Action
  - Use Scriptlets

```
<%@ page import="beans.Thermostat" %>
<jsp:useBean id="t" class="beans.Thermostat"/>
<%
  Thermostat t2=new Thermostat(50);
  t.setTemp(78);
%>
The thermostat was set at a temperature of <jsp:getProperty
name="t" property="temp"/> degrees.<P>
<%-- <jsp:getProperty name="t2" property="temp"/>  will not work --
%>
The thermostat was set at a temperature of <%=t2.getTemp()%>
degrees.
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

 27

<u>Bean Related Actions</u>:

Configuring Beans:

- Many times a Bean requires runtime configuration by the page that is initializing it before it can properly perform its tasks. Since we cannot pass information into the Bean's **constructor**, we have to use the **Bean's properties** to hold configuration information. We do this by setting the appropriate **property values** immediately after the container instantiates the Bean in the body of the **<jsp:useBean>** tag or anywhere in the page before the **Bean's properties** are accessed. It can be useful to set a flag in class to indicate whether an instance is in a useful state, toggling the flag when all of the necessary propearties have been set.
- Even though the **Bean tags** do not allow to pass any arguments into a **Bean's constructor**, we can still define constructors that take arguments. We will not, however, be able to call them through **Bean tags**. The only way to instantiate an object requiring arguments in its constructor within a JSP page is through a **scriptlet**.
- One technique that has been found useful is to provide a single method that handles all the configuration steps. This method can be called by the **constructors** that take arguments, for use outside the Bean tags, as well as by the **property access methods** once all the necessary properties have been configured.

Demo: jsp:useBean Action

- Demo on:
  - input.jsp
  - output.jsp
  - output2.jsp
  - output3.jsp
  - Employee.java

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 28

Refer Demo on **Lesson5-JSPActions, useBean** folder and show demo by executing each of the above JSP pages.
Note: The listing for **Employee.java (Bean)** is as follows:

```
public class Employee {
private int eid;
private String enm;
private double esl;

public int getEid() {
return eid;
}
public void setEid(int eid) {
this.eid = eid;
}
public String getEnm() {
return enm;
}
public void setEnm(String enm) {
this.enm = enm;
}
public double getEsl() {
return esl;
}
public void setEsl(double esl) {
this.esl = esl;
}
}
```

Lab

- Lab 2.1
- Lab 2.2

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 29

Summary

- In this lesson, you have learnt the following concepts:
  - JSP Actions: include, forward, and plugin
  - Java Beans
  - Bean Related Actions

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 30

Review Question

- Question 1: The include action is a ___ include.
  - Option 1: dynamic
  - Option 2: static
  - Option 3: both the above
- Question 2: The ___ attribute of include action is mandatory to be set.
- Question 3: ___ sub-element can be used to pass information to included or forwarded JSP page.
- Question 4: ___ action terminates the execution of current page before moving on to the next page.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 31

Review Question

- Question 5: ___ is a client-side redirect.
- Question 6: ___ action generates browser dependent construct, namely object or embed.
- Question 7: ___ tag is used to display message for the user if the plugin cannot be started.
- Question 8: Java Bean does not extend any specific base class or implement any interface. True/False
- Question 9: JSP uses ___ to interact and access Java Bean's and its properties.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 32

Review Question

- Question 10: The useBean action creates a new instance of a bean only if an existing instance of the same bean within the scope is not available. True/False
- Question 11: The conversion from string value to appropriate data type is done by ___ action.
- Question 12: The multiple argument constructor of a Java Bean can be called through JSP actions in a JSP page.
  - True/False

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 33