

Custom Tags

Custom Tags

- Apart from the standard tags, JSP also allows creation of tags by the developer or 3rd party.
- This can then be used by the developers in JSP like standard tags are used.
- A tag library can be defined which contains a collection of custom tags.
- The idea behind having custom tag is to have scriptless JSP.
- With scriptless JSP, embedding too much code in the page can be avoided thus resulting in a neat JSP page.

Steps to create a custom tag library

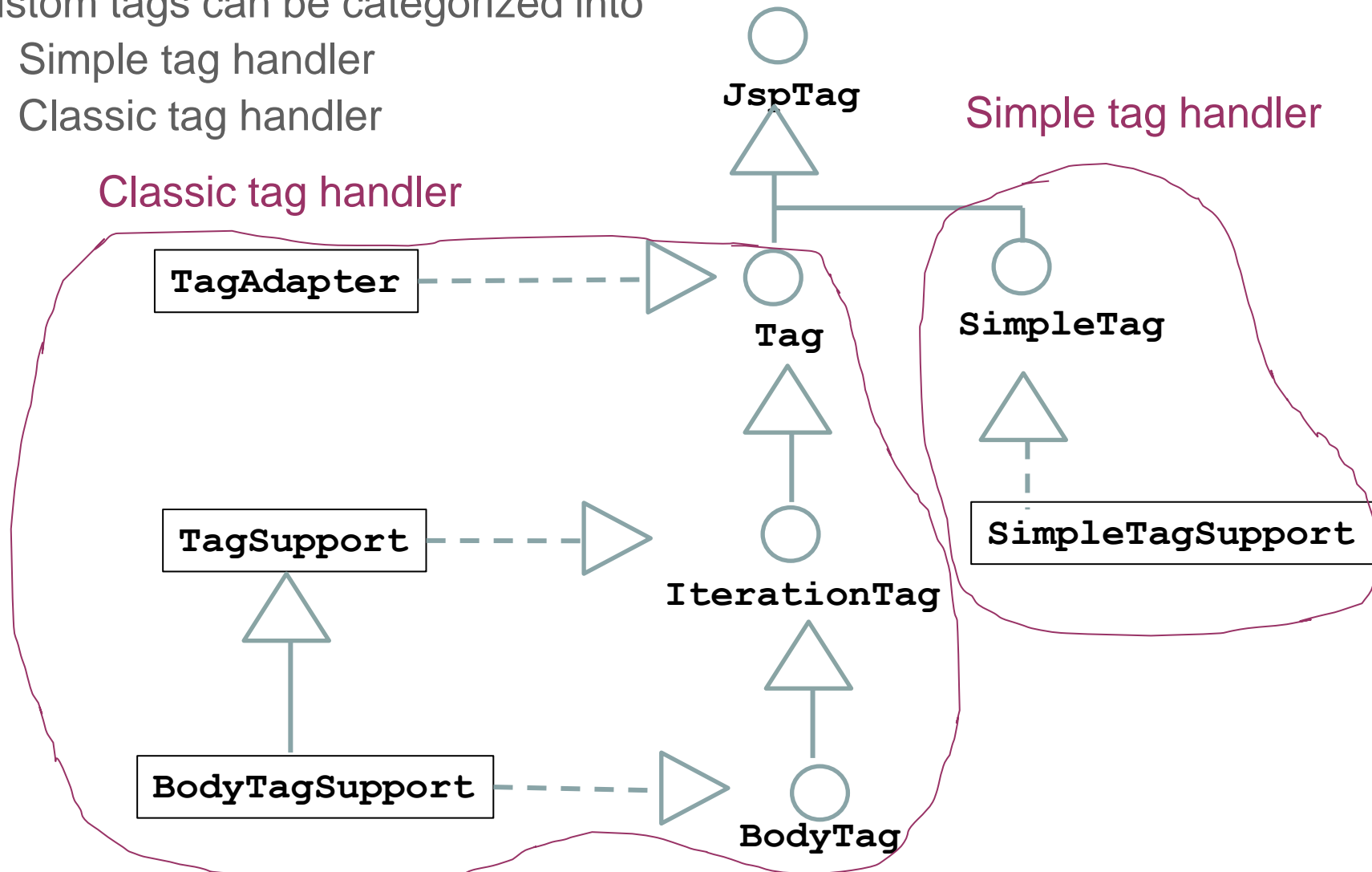
1. A tag handler class to be created:
 - This class specifies the action of a custom tag and is an instance of a Java class that implements either the **Tag**, **IterationTag**, or **BodyTag** interface in the standard. Which interface to implement depends on whether the tag has a body and whether the tag handler requires access to the body content. Location is WEB-INF/classes
2. A tag library descriptor (TLD) file to be created in WEB-INF/tlds directory
 - This XML document contains information about a tag library and about individual tags of the library. The file name of a TLD has the .tld extension.
3. In web.xml taglib element and two subelements: **taglib-uri** and **taglib-location** is specified. [optional]
4. Reference the tag library in your JSP source using the JSP **<taglib>** directive.

Test your understanding

- What are 2 fundamental types of tags that you can create?

Custom Tag Hierarchy

- Custom tags can be categorized into
 - Simple tag handler
 - Classic tag handler



Simple tag handler

- `JspTag` is a marker interface.
- The simple tag handler class implements the `javax.servlet.jsp.tagext.SimpleTag` interface
- `SimpleTag` interface provides a simple `doTag()` method where all tag logic like when the tag is opened , closed, iteration, body evaluations, etc. are performed. This method is called only once for each tag invocation.
- Tag can be empty or have body. The `setJspBody()` method can be used to work with the body of the tag.

Methods

<<interface>>

JspTag



<<interface>>

SimpleTag

```
void doTag()  
JspTag getParent()  
void setParent(JspTag p)  
void  
setJspBody(JspFragment f)  
void  
setJspContext(JspContext  
c)
```



SimpleTagSupport

```
static final JspTag  
findAncestorWithClass(Js  
pTag from, Class<?> k)  
  
JspFragment getJspBody()  
  
JspContext  
getJspContext()
```

The **SimpleTagSupport** class
implements the **SimpleTag**

JspContext

- `JspContext` is the base class for the `PageContext` class.
- It is an abstract class.
- This class can be used to get many information like
 - Attributes in various scope
 - `JspWriter` for output

*Do you recall **PageContext** ?*

JspContext Methods

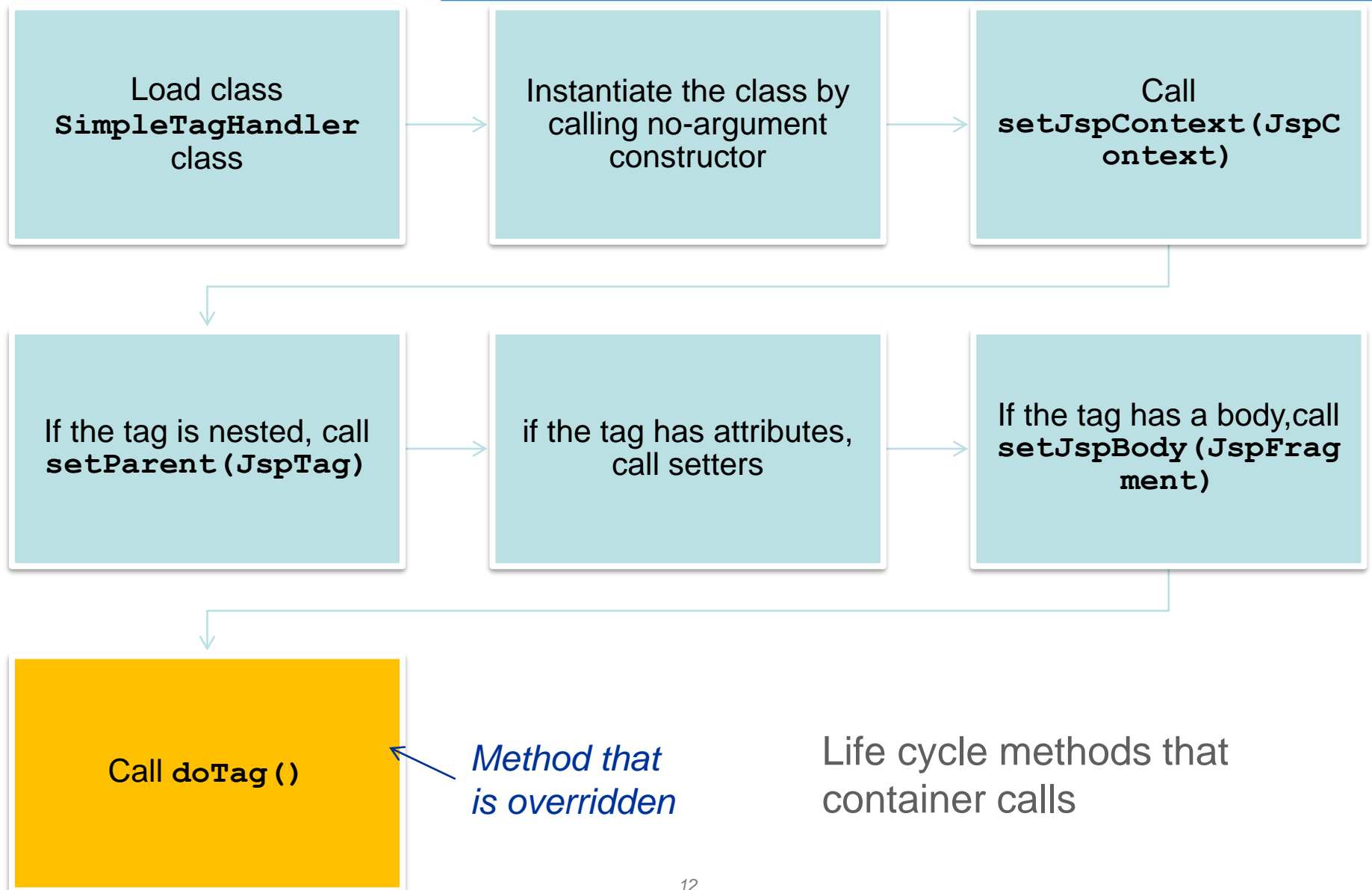
- `Object getAttribute(String name) ,`
`void setAttribute(String name, Object value)`
 - Get and sets attributes in the page scope.
- `Object getAttribute(String name, int scope)`
`void setAttribute(String name, Object value, int scope)`
 - Get and sets attributes in the specified scope.
 - static constant for scope is defined in `PageContext` class as
 - `APPLICATION_SCOPE`, `SESSION_SCOPE` , `REQUEST_SCOPE` and `PAGE_SCOPE`
 - `IllegalArgumentException` is thrown by if the scope is invalid
 - `IllegalStateException` is thrown if the scope is `PageContext.SESSION_SCOPE` but the page that was requested does not participate in a session or the session has been invalidated.

- `Object findAttribute(String name) :`
 - Searches for the named attribute in page, request, session (if valid), and application scope(s) in order and returns the value associated or null.
- `void removeAttribute(java.lang.String name)`
 - Remove the object reference associated with the given name from all scopes
- `void removeAttribute(java.lang.String name, int scope)`
 - Remove the object reference associated with the specified name in the given scope
- `JspWriter getOut()`
 - Returns out object using which output can be written

JspFragment, Methods

- The object of this class is used to encapsulate a portion of JSP code that can be invoked as many times as needed.
- JSP fragment must only contain template text and JSP action, custom actions and EL expressions. It must not contain scriptlets or scriptlet expressions.
- `SimpleTag`'s `setJspBody()` method has an argument `JspFragment` object that encapsulating the body of the tag.
- The tag handler implementation can call `invoke()` on that fragment to evaluate the body as many times as it needs.
- Methods:
 - `void invoke(java.io.Writer out) throws JspException, java.io.IOException`: Executes the fragment and directs all output to the given `Writer`, or the `JspWriter` returned by the `getOut()` method of the `JspContext` associated with the fragment if `out` is `null`.
 - `JspContext getJspContext()`

Simple Tag Life Cycle



Steps to write a Simple Tag

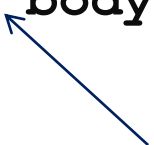
1. Write the class that extends `SimpleTagSupport` and implement the `doTag()` method
2. Create a TLD for the tag
3. Update web.xml [optional]
4. Write JSP that uses this tag. Add a reference the tag using the JSP `<taglib>` directive.
5. Deploy and test

1. Simple tag handler without body

```
package simple;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class SimpleTagEx extends SimpleTagSupport {
    public void doTag() throws JspException, IOException
    {
        getJspContext().getOut().print("Hello! Simple Custom
                                     Tag without body");
    }
}
```



Print this when the tag is invoked

2. TLD

- A *tag library description* (TLD) file is an XML document that contains information about a tag library and about individual tags of the library. The name of a TLD file has the .tld extension.
- A JSP container uses the TLD file in determining what action to take when it encounters a tag from the library.
- TLD files are placed inside **WEB-INF/tlds** directory.
- This path is provided in **web.xml**'s **<taglib-location>**
- If web.xml does not provide **<taglib>** tags, then this path must be provided in **uri** attribute of **<%@ taglib>** directive in individual pages.
- A TLD file can contain any number of **<tag>**s

```
<?xml version="1.0" encoding="UTF-8"?>

<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-
jsptaglibrary_2_0.xsd" version="2.0">

    <tlib-version> 1.2</tlib-version>

    <uri>SimpleTag</uri>

    <tag>

        <description>Simple tag</description>

        <name>Simple</name> ← Tag name that will be used in the jsp

        <tag-class>simple.SimpleTagEx</tag-class>
        ← Fully qualified name of the class

        <body-content>empty</body-content>
        ↑
        Says that the tag cannot have any body

    </tag>

</taglib>
```


3. Update web.xml

- This is an optional step. If this is not done then the `uri` attribute of the `taglib` directive in JSP page must specify the full path.
- This is a recommended because it delinks the JSP pages from changes in the location of tld files.
- For J2EE 1.4 and later projects, due to a change in the format of the `web.xml` file all `<taglib>` entries in your `web.xml` file must be enclosed in a top-level `<jsp-config></jsp-config>` tag.

```
<web-app>
```

```
<jsp-config>
```

```
<taglib>
```

```
<taglib-uri>/SimpleTag</taglib-uri>
```

```
<taglib-location>
```

```
    /WEB-INF/tlds/simple.tld
```

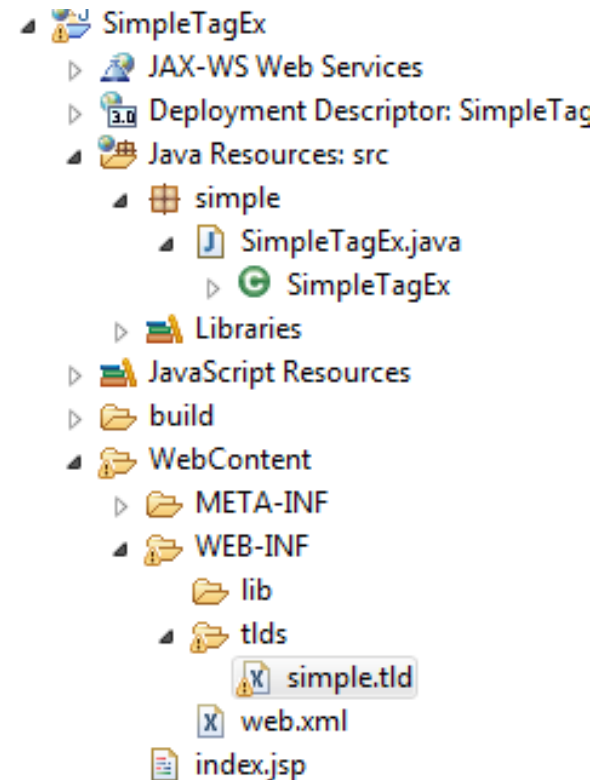
```
</taglib-location>
```

```
</taglib>
```

```
</jsp-config>
```

```
</web-app>
```

Same what was specified in the
tld file's uri attribute but with /



4. JSP page

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

```
<%@ taglib prefix="s" uri="/SimpleTag"%>
```

```
<html>
```

```
  <head>
```

```
    <title>Simple Tag</title>
```

```
  </head>
```

```
  <body>
```

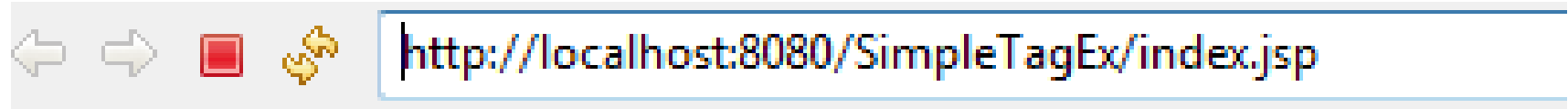
```
    <s:Simple/>
```

```
  </body>
```

```
</html>
```

This should be same as the
<taglib-uri> of
web.xml. If the web.xml
does not have <taglib>
entry then this should be fully
qualified path of tld → /WEB-
INF/tlds/simple.tld

Deploy and Execution



Hello! Simple Custom Tag without body

A simple tag with body

- To include body with tag two things have to done
 1. In TLD `<body-content>` must not be “**empty**”. It can be either
 1. **scriptless** (indicates that only scriptless JSP will be processed. Hence tag body cannot have scripting elements.)
 2. **tagdependent** (indicates that the tag body should not be translated. Any text in the body is treated as static text.)
 2. Include `getJspBody().invoke(null)` in `doTag()` method of Tag handler class
 - `getJspBody().invoke(null)` → process and print the body of the contents. The `null` argument indicates that response stream will be used for output.

Example: A simple tag with body

```
package simple;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.SimpleTagSupport;

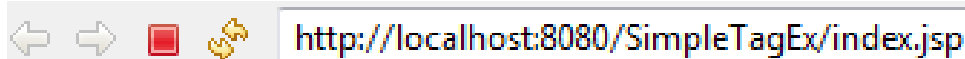
public class SimpleTagWithBody extends SimpleTagSupport
{
    public void doTag() throws
        JspException, IOException
    {
        getJspBody().invoke(null);
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web
-jsptaglibrary_2_0.xsd" version="2.0">
<tlib-version> 1.2</tlib-version>
<uri>SimpleTag</uri>
```

```
<tag>
    <description>Simple tag with body</description>
    <name>SimpleBody</name>
    <tag-class>simple.SimpleTagWithBody</tag-class>
    <body-content>scriptless</body-content>
</tag>
</taglib>
```

JSP page and Execution

```
<%@ page language="java" contentType="text/html;  
charset=ISO-8859-1"  
    pageEncoding="ISO-8859-1"%>  
    <%@ taglib prefix="s" uri="/SimpleTag"%>  
<html><head><title>Simple Tag</title> </head>  
<body>  
    <s:SimpleBody>  
        This is a simple tag with body. Scripless execution  
        test result: ${5*100}  
    </s:SimpleBody>  
</body></html>
```

A screenshot of a web browser's address bar. It contains the URL 'http://localhost:8080/SimpleTagEx/index.jsp'. To the left of the address bar are four small icons: a left arrow, a right arrow, a red square, and a yellow dollar sign.

This is a simple tag with body. Scripless execution test result: 500

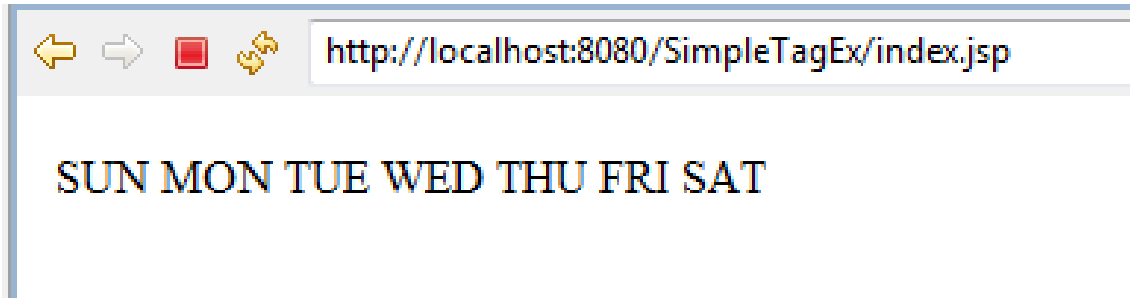
“invoke” ing multiple times

- A situation where we will need to invoke a body content multiple times is when we need to display a list of values using just a pair of custom tags.
- In such case multiple calls to invoke() is made in an iteration.

```
package simple;
import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.SimpleTagSupport;
public class SimpleTagWithBody extends
SimpleTagSupport {
String[]
week={"SUN", "MON", "TUE", "WED", "THU", "FRI", "SAT"};
public void doTag() throws JspException, IOException{
    for(String w:week){
        getJspContext().setAttribute("week", w);
        getJspBody().invoke(null);}}}
}
```

JSP and execution

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/SimpleTag"%>
<html>
  <body>
    <table><tr>
      <s:SimpleBody>
        <td> ${week}</td>
      </s:SimpleBody>
    </tr></table>
  <body>
</html>
```



Simple tag with attributes

- A tag with attribute maps to the instance variable of the tag handler class.
- When the value of the attribute is assigned, the setter method of the tag handle class is called.
- So to have a tag with attribute 2 things need to e done
 1. Tag handler class must have a instance field with the same name as attribute name and have setter and getter methods for it.
 2. In TLD, the <attribute> tag must be specified which describes the attribute.


Example: Simple tag with attributes

```
package simple;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class SimpleTagWithAttributes extends
SimpleTagSupport {
    private String title;
    public String getTitle() {return title;}
    public void setTitle(String title) {this.title = title;}

    public void doTag() throws JspException, IOException{
        getJspContext().getOut().print(title);
        getJspBody().invoke(null);
    }
}
```



Value that is fetched from the attribute

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-
jsptaglibrary_2_0.xsd" version="2.0">
<tlib-version> 1.2</tlib-version>
<uri>SimpleTag</uri>
<tag>
  <description>Simple tag with attributes</description>
  <name>SimpleAttr</name>
  <tag-class>simple.SimpleTagWithAttributes</tag-class>
  <body-content>scriptless</body-content>
  <attribute>
    <name>title</name>
    <required>true</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
</tag>
</taglib>
```

The default is false

Whether the attribute's value can be dynamically calculated at runtime by an EL expression. The default is false

JSP and execution

```
<%@ page language="java" contentType="text/html;  
charset=ISO-8859-1"  
    pageEncoding="ISO-8859-1"%>  
    <%@ taglib prefix="s" uri="/SimpleTag"%>  
<html>  
<body>  
    <s:SimpleAttr title="Simple tag with attribute and  
body">-->This is title attribute that was  
printed</s:SimpleAttr>  
</body>  
</html>
```

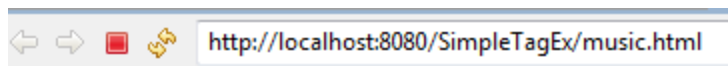


<http://localhost:8080/SimpleTagEx/index.jsp>

Simple tag with attribute and body-->This is title attribute that was printed

Example: Iteration using simple tag

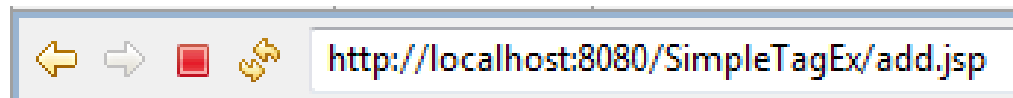
- In this example, an html pages show a list of items from which uses can select.
- On submit, a jsp page is called where the attribute of a simple tag is set using a el expression which returns a list of values that user selected in the previous page.
- In the `doTag()` method, iteration on each value in the list is performed and the request attribute containing individual value of list is set and `invoke()` method is called.



Buy Music

1. ☐ Dangerous
2. ☒ Arrival
3. ☒ Best Hits of Richard Clayderman
4. ☐ Hotel California

add to cart



Arrival

Best Hits of Richard Clayderman

```
<html><head><title>Books</title></head>
<body><h1>Buy Music</h1>
<form method="post" action="add.jsp">
<ol><li><input type="checkbox" name="music"
value="Dangerous">Dangerous
<li><input type="checkbox" name="music"
value="Arrival">Arrival
<li><input type="checkbox" name="music" value="Best
Hits of Richard Clayderman">Best Hits of Richard
Clayderman
<li><input type="checkbox" name="music" value="Hotel
California">Hotel California</ol>
<input type="submit" value="add to cart">
</form>
</body></html>
```



```

<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
    <%@ taglib prefix="s" uri="/SimpleTag"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <title>Insert title here</title>
</head>
<body>
    <table>
        <s:SimpleIter music="${paramValues.music}">
            <tr><td> ${mus}</td></tr>
        </s:SimpleIter>
    </tr>
</table>
</body></html>

```

Gets the music that has been ticked in music.html

Request attribute which is set in the doTag()

```

package simple;
import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class SimpleIteration extends SimpleTagSupport
{
    private String[] music;
    public String[] getMusic() {return music;}
    public void setMusic(String[] music) {
        System.out.println(music[0]);
        this.music = music;
    }

    public void doTag() throws JspException, IOException{
        for(String m: music){
            getJspContext().setAttribute("mus", m);
            getJspBody().invoke(null);
        }
    }
}

```

SkipPageException

- This exception can be thrown from the `doTag()` method.
- Everything in the page up to the point in the `doTag()` method where the exception is thrown is printed on the page.
- After the exception is caught, the rest of the page is not processed.

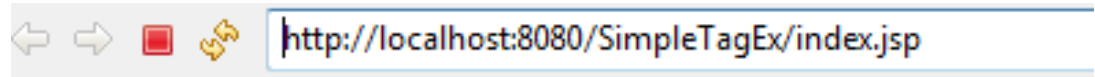
```
public void doTag() throws JspException, IOException{  
    getJspContext().getOut().print("Hello! Simple Custom Tag  
    without body");  
  
    throw new SkipPageException();  
}
```

```
<html><body>
```

```
<s:Simple/>
```

```
This will not be seen
```

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



Hello! Simple Custom Tag without body

This does not get printed

Classic tag

- A classic tag provide the old way (Pre-JSP 2.0) of writing custom tags.

This class is used to allow collaboration between classic Tag handlers and SimpleTag handlers.

We will not be dealing with this in this session

TagAdapter



Simple tag handler

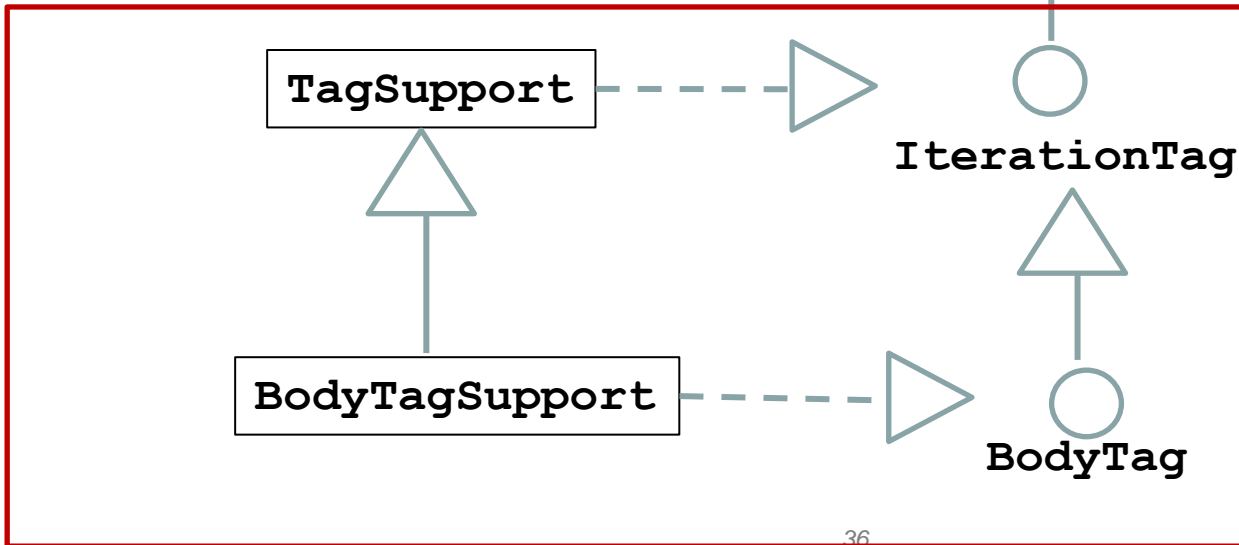
TagSupport

IterationTag

BodyTagSupport

BodyTag

SimpleTagSupport



Tag interface

- Used for the classic tag handler that does not want to manipulate its body.
- Members
 - Static constants: `EVAL_BODY_INCLUDE`, `SKIP_PAGE`, `EVAL_PAGE`, `SKIP_BODY`
 - `int doStartTag() throws JspException`
 - `int doEndTag() throws JspException`
 - `Tag getParent()`
 - `void release()`
 - `void setPageContext(PageContext pc)`
 - `void setParent(Tag t)`

IterationTag interface

- `IterationTag` inherits from `Tag` and adds `doAfterBody()`
- The `doAfterBody()` method is invoked after every body evaluation to control whether the body will be reevaluated or not. If `doAfterBody()` returns `IterationTag.EVAL_BODY_AGAIN`, then the body will be reevaluated. If `doAfterBody()` returns `Tag.SKIP_BODY`, then the body will be skipped and `doEndTag()` will be evaluated instead.
- `static final int EVAL_BODY_AGAIN`
- `int doAfterBody() throws JspException`
- `TagSupport` class implements `IterationTag`

Life cycle IterationTag is invoked

Load and
Instantiate
the class by
calling no-
argument
constructor

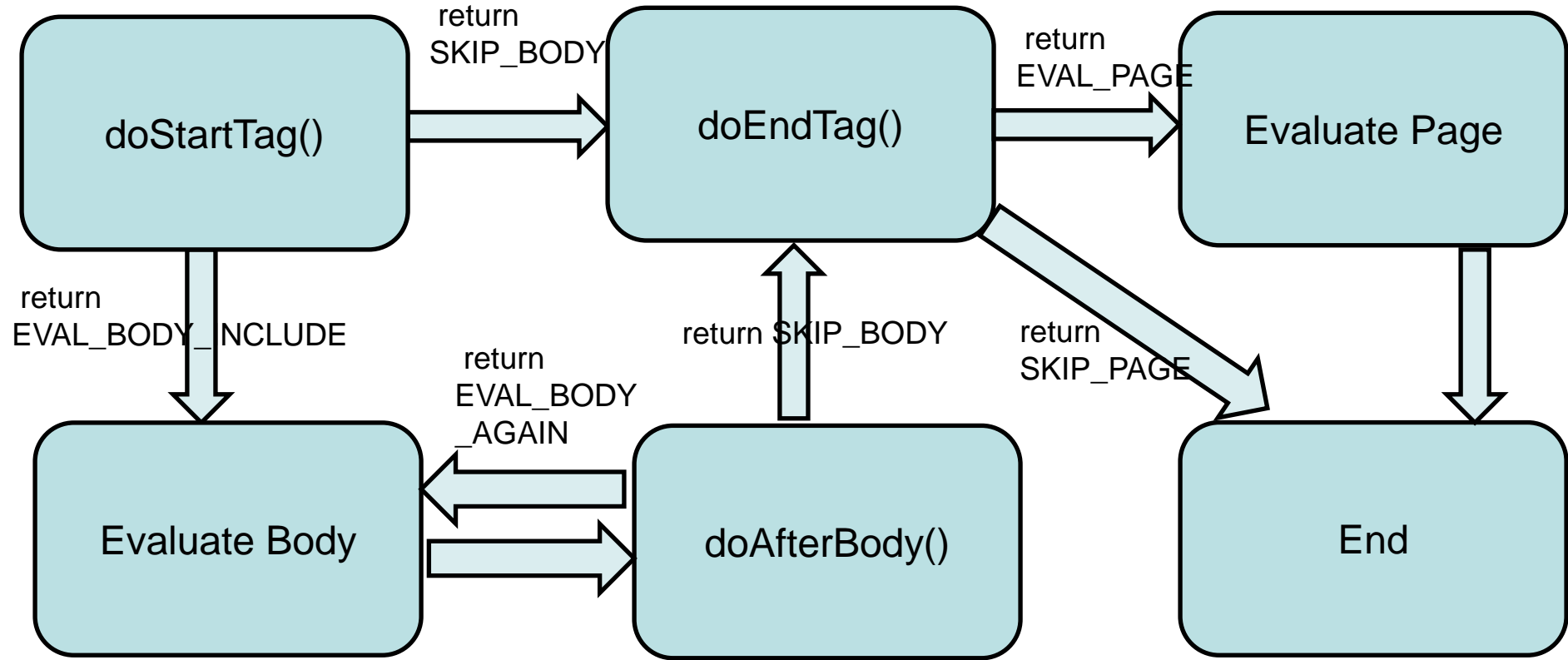
Call
setPageCon
text(PageCo
ncontext)

If the tag is
nested, call
setParent(Ta
g)

If tag has
attributes
call setters

Call
doStartTag()

Container calls....



Method that are overridden

Example: classic tag with no body

```
package classic;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.*;

public class ClassicWithoutBody extends TagSupport{
    public int doStartTag() throws JspException {
        JspWriter out=pageContext.getOut();
        try{
            out.println("Hello!Classic Tag without Body");
        }catch(IOException ie){ }
        return SKIP_BODY;}
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-
jsptaglibrary_2_0.xsd" version="2.0">
<tlib-version> 1.2</tlib-version>
<uri>ClassicTag</uri>
<tag>
<description>Classic tag without body</description>
<name>ClassicNoBody</name>
<tag-class>classic.ClassicWithoutBody</tag-class>
<body-content>empty</body-content>
</tag>
</taglib>
```

Created in the same way as Simple tags

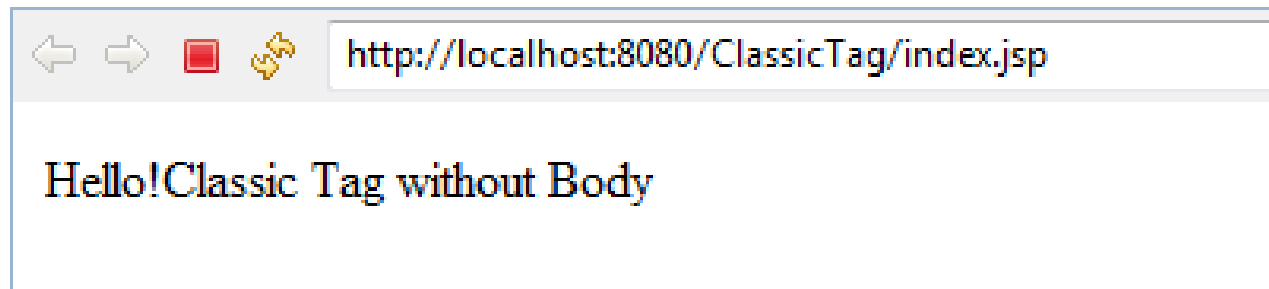
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
id="WebApp_ID" version="3.0">
  <display-name>SimpleTagEx</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>

  <jsp-config>
    <taglib>
      <taglib-uri>/ClassicTag</taglib-uri>
      <taglib-location>/WEB-INF/tlds/classic.tld</taglib-
location>
    </taglib>
  </jsp-config>
</web-app>
```

Same entries as that of a simple tag

JSP and execution

```
<%@ page language="java" contentType="text/html;  
charset=ISO-8859-1"  
    pageEncoding="ISO-8859-1"%>  
    <%@ taglib prefix="c" uri="/ClassicTag"%>  
<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html;  
charset=ISO-8859-1">  
<title>Insert title here</title>  
</head>  
<body>  
<c:ClassicNoBody/>  
</body>  
</html>
```

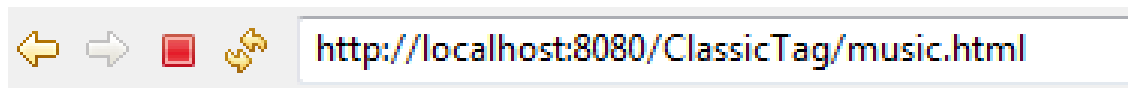


TLD `<body-content>`

- For classic tags `<body-content>` can be
 - **empty**
 - Indicates that tag should have no body
 - **scriptless**
 - indicates that only scriptless JSP will be processed. Hence tag body cannot have scripting elements.
 - **tagdependent**
 - indicates that the tag body should not be translated. Any text in the body is treated as static text.
 - **JSP**
 - indicates that Tag body should be processed as JSP source and translated
- The default value for `<body-content>` is **scriptless**.

Example: classic tag with body

- This example achieves the same that the Iteration using simple tag example did. The attributes setter methods are called in the same way!



Buy Music

1. ☒ Dangerous
2. ☒ Arrival
3. ☒ Best Hits of Richard Clayderman
4. ☐ Hotel California

add to cart

```

package classic;
import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;

public class ClassicTagWithBody extends TagSupport{
    private String[] music;
    int index=0;
    public String[] getMusic() {return music;}
    public void setMusic(String[] music) {
        this.music = music;
    }
    public int doStartTag() throws JspException {
        index=0;
        JspWriter out=pageContext.getOut();
        try {
            out.print("<br>Beginiing of the List <br><<<br>");
        } catch (IOException e) {e.printStackTrace();}
        pageContext.setAttribute("mus",music[index]);
        return EVAL_BODY_INCLUDE;
    }
}

```

```

public int doAfterBody() throws JspException{
    ++index;
    if(index<music.length) {
        pageContext.setAttribute("mus",music[index]);
        return EVAL_BODY_AGAIN;
    }
    else{
        JspWriter out=pageContext.getOut();
        try {
            out.print(">><br>End of the List");
        } catch (IOException e) {e.printStackTrace();}
        return SKIP_BODY;
    }
}

public int doEndTag() throws JspException{
    return EVAL_PAGE;
}

}

```


<tag>

<description>Classic tag with body</description>

<name>ClassicBody</name>

<tag-class>classic.ClassicTagWithBody</tag-class>

<body-content>JSP</body-content>

<attribute>

<name>music</name>

<required>true</required>

<rtexprvalue>true</rtexprvalue>

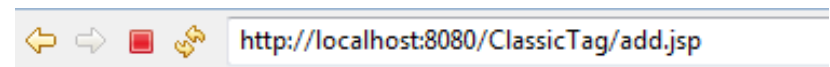
</attribute>

</tag>

JSP and execution

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="c" uri="/ClassicTag"%>
<html>
<body>
    <% int i=1; %>
    <c:ClassicBody music="${paramValues.music}">
        (<%=i++%>) ${mus}<br>
    </c:ClassicBody>
</body>
</html>
```

Since body content is
JSP, script elements
are allowed here



Beginiing of the List

<<

(1) Dangerous

(2) Arrival

(3) Best Hits of Richard Clayderman

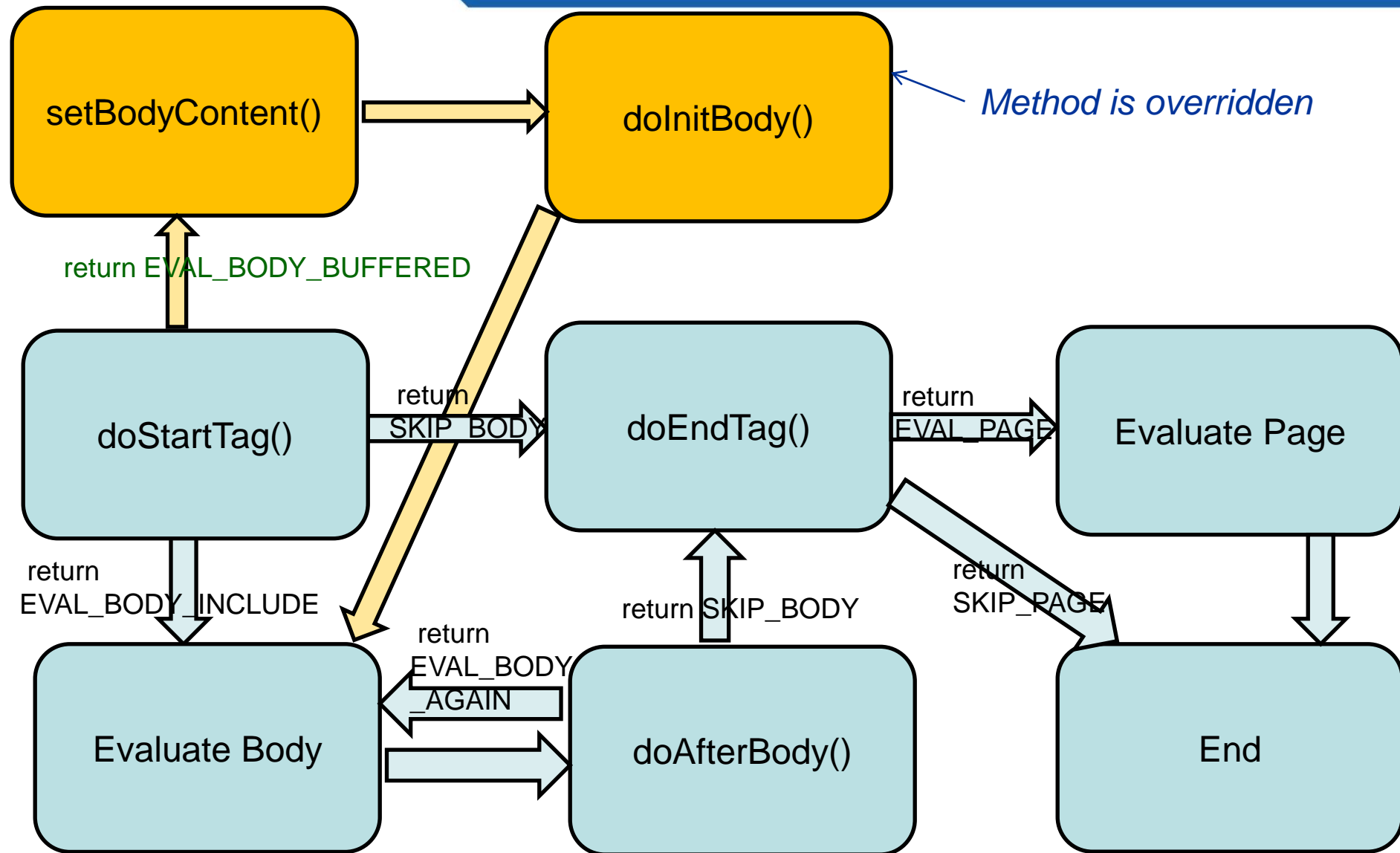
>>

End of the List

BodyTag interface

- This interface extends from **IterationTag**. In cases where the body content needs to be accessed and changed this class is useful.
- It adds 2 methods:
 - **void doInitBody() throws JspException**
 - **void setBodyContent(BodyContent b)**
 - These methods are called only once. Also these method will not be invoked for empty tags or for non-empty tags whose **doStartTag()** method returns **SKIP_BODY** or **EVAL_BODY_INCLUDE**.
 - The **doInitBody()** is the one overridden and is called after the body content is set but before it is evaluated. Generally this is used to perform any initialization that depends on the body content.
- **doStartTag()** returns **EVAL_BODY_BUFFERED** also here apart from **SKIP_BODY** and **EVAL_BODY_INCLUDE**.
- **BodyTagSupport** class implements this interface.

Addition to life cycle method calls



BodyContent

- Subclass of `JspWriter`
- `BodyContent` has methods to convert its contents into a `String`, to read its contents, and to clear the contents.
- `BodyContent` is made available to a `BodyTag` through a `setBodyContent()` call. The tag handler can use the object until after the call to `doEndTag()`.
- `public void clearBody()`
- `java.lang.String getString()`
- `JspWriter getEnclosingWriter()`

Example using BodyTag interface

- JSP contains the HTML code has to be displayed in the page.
- But since HTML used angular brackets they will be considered as template text.
- Therefore a custom tag is created that allows writing the html code inside it.
- When the custom tag processes the body content, the special characters are replaced so that when it is finally written to the page, the HTML code is displayed instead of being interpreted.

```
package classic;
import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.BodyTagSupport;
public class ClassicTagWithInitBody extends
BodyTagSupport{

public int doStartTag() throws JspException {
return EVAL_BODY_BUFFERED;
}

public void doInitBody() throws JspException {
try {
bodyContent.print("This is an example for HTML :");
} catch(IOException ioe) {
throw new JspException(ioe.getMessage());
}
}

public int doAfterBody() throws JspException {
return SKIP_BODY;
}
```

```
public int doEndTag() throws JspException {
String s=bodyContent.getString();
s=s.replace("<", "&lt;");
s=s.replace(">", "&gt;");
try {
pageContext.getOut().print("<BR><PRE>" + s + "</PRE>");
} catch (IOException e) {}
return EVAL_PAGE;
}
}
```

TLD:

<tag>

<description>Classic tag with body and
init</description>

<name>ClassicBodyInit</name>

<tag-class>classic.ClassicTagWithInitBody</tag-class>

<body-content>JSP</body-content>

</tag>

JSP and Execution

```
<%@ page language="java" contentType="text/html;  
charset=ISO-8859-1"  
    pageEncoding="ISO-8859-1"%>  
<%@ taglib prefix="c" uri="/ClassicTag"%>  
<html>  
<body>  
<c:ClassicBodyInit>  
  
<HTML>  
    <BODY>  
        Hello  
    </BODY>  
</HTML>  
</c:ClassicBodyInit>  
</body>  
</html>
```

