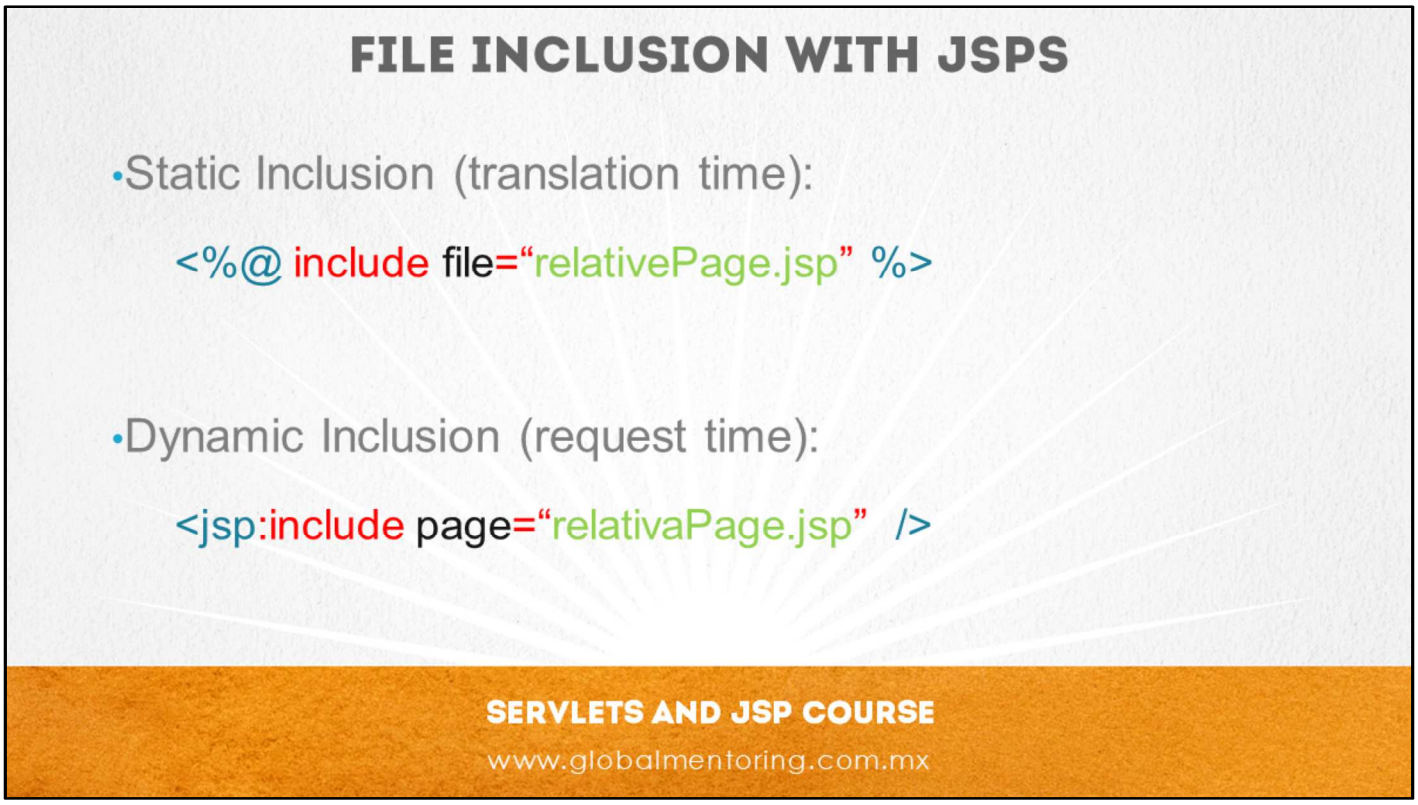
Hello, Ubaldo Acosta greets you. Welcome again. I hope you're ready to start with this lesson.

We are going to study the topic of file inclusion with JSP's.

Are you ready? OK let's go!



In this lesson we are going to review the topic of file inclusion using JSPs.

In simple terms the inclusion of files, as the name says, is to add a JSP inside another, however there are two ways to perform this action when using JSPs.

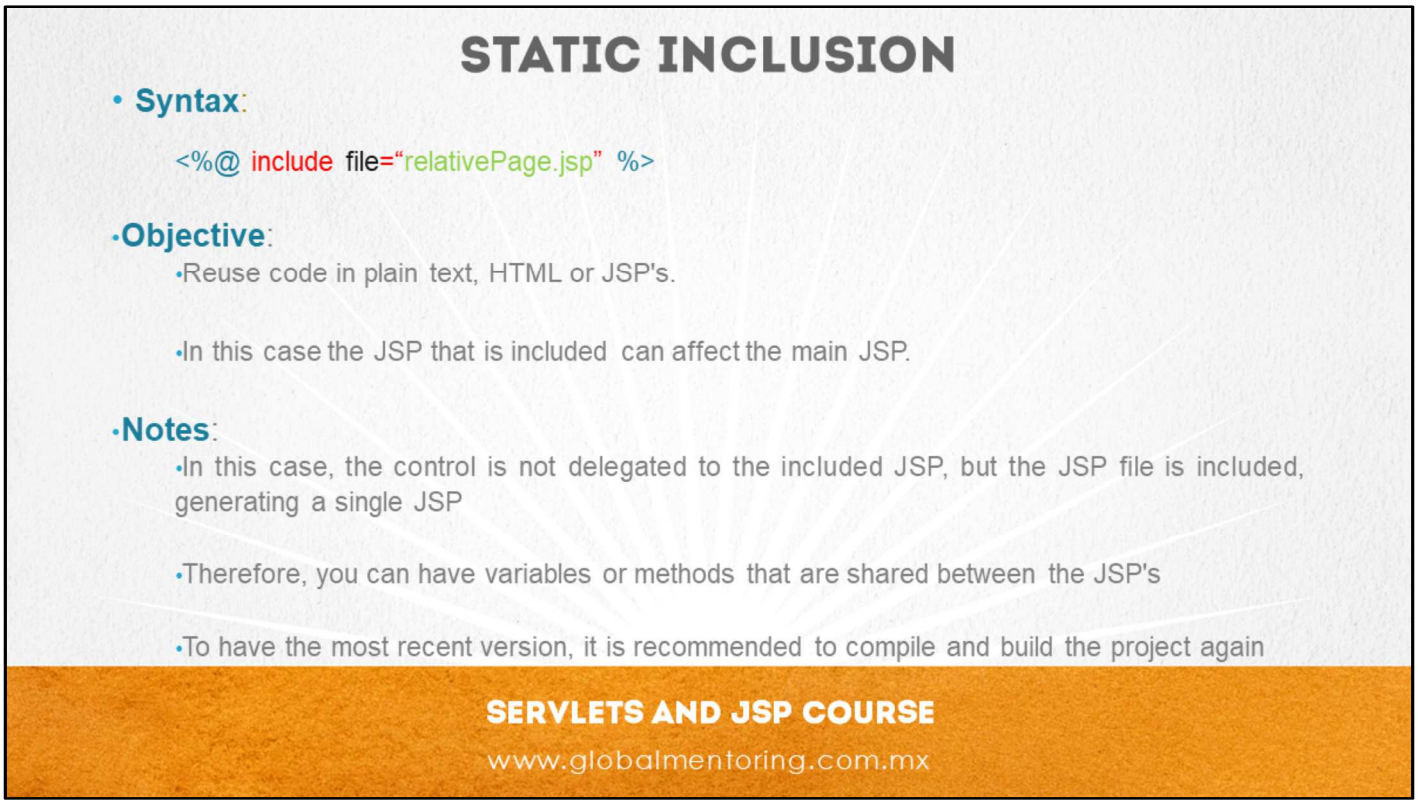
In the first place we have the static inclusion also known as inclusion in translation time (translation time), on the other hand we have the dynamic inclusion also known as inclusion in execution time (request time).

Let's review the differences between each type of file inclusion.

Let's start with static inclusion. This concept can be applied using the include directive and as we saw earlier in the topic of directives we can recognize this directive by means of the @ symbol. Later what we do is specify by means of the file attribute the name of the JSP that we are going to include in the main JSP.

In the case of dynamic inclusion, what happens is that we will use, instead of a directive, the concept of an action, then we will review more actions, especially to access the JavaBeans, but in this case an action will also be it will allow us to dynamically include another JSP. After the name of the action, we specify the page attribute and the value of the JSP attribute that we want to include in our main JSP.

We can see that part of the differences is that in our directive <% @ include file = "relativePage.jsp"%> we are handling the file attribute and in the dynamic inclusion <jsp: include page = "relativePage.jsp" /> using the page attribute, these are just some of the differences and we are going to review these two concepts in more detail.

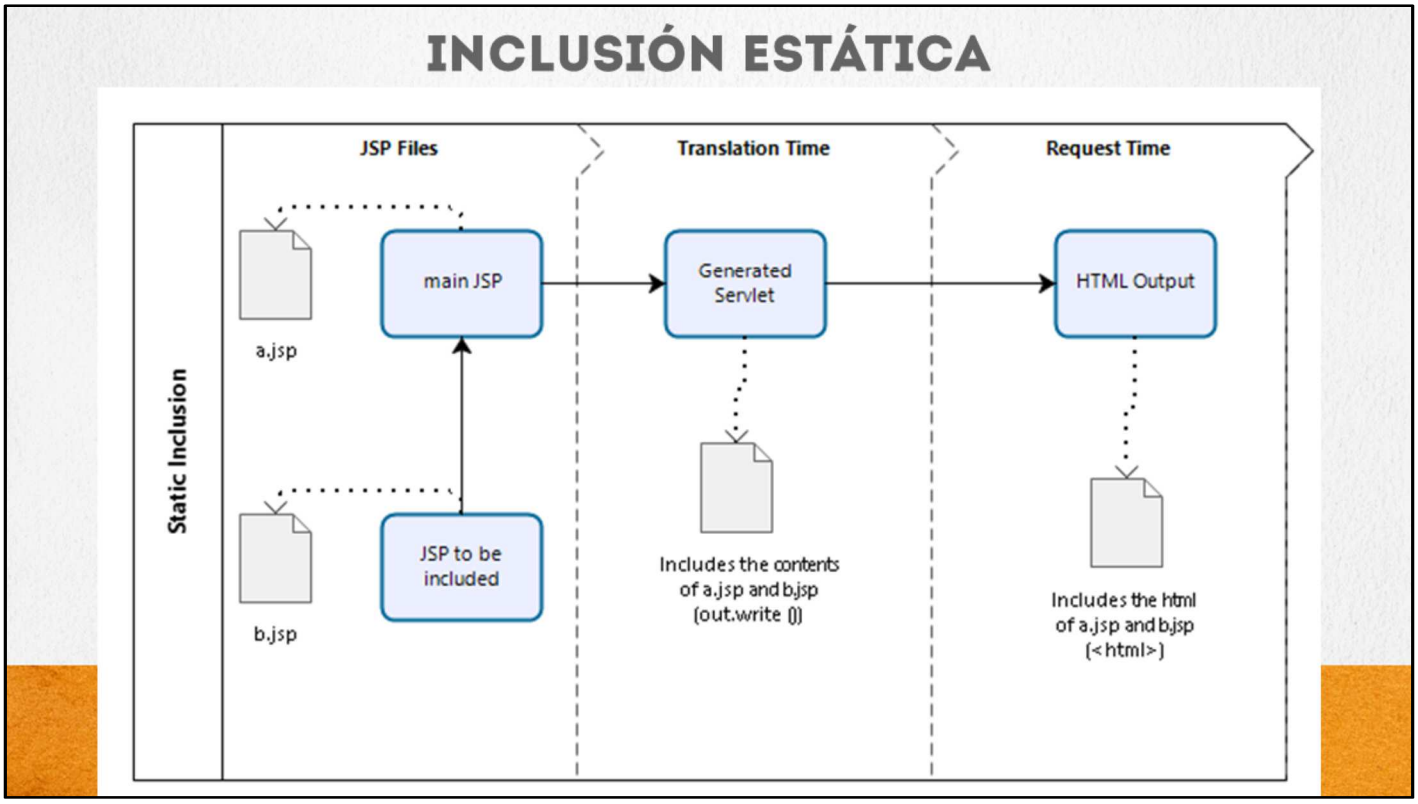


In static inclusion, as we have seen, we have the following syntax:

<% @ include file = "relativePage.jsp"%> We are going to use a directive, so we open an angle bracket(<), followed by a percent sign (%) and use the at sign (@), then use the include directive and finally we specify the name of the file that we are going to include in the main JSP.

The objective of using this static inclusion is to reuse code that can come from a plain text of an HTML or another JSP. In the case of static inclusion, the JSP that we are including within the main JSP may affect or share information with the main JSP. In the static inclusion, the control is not delegated to the JSP that we are going to include, but it is going to generate a single JSP that will be the combination of the main JSP and the JSP that we are including. Recall that a JSP is finally compiled into a Servlet, since the Java class is necessary to be able to execute the associated JSP. Later we will see an example to have this concept more clear.

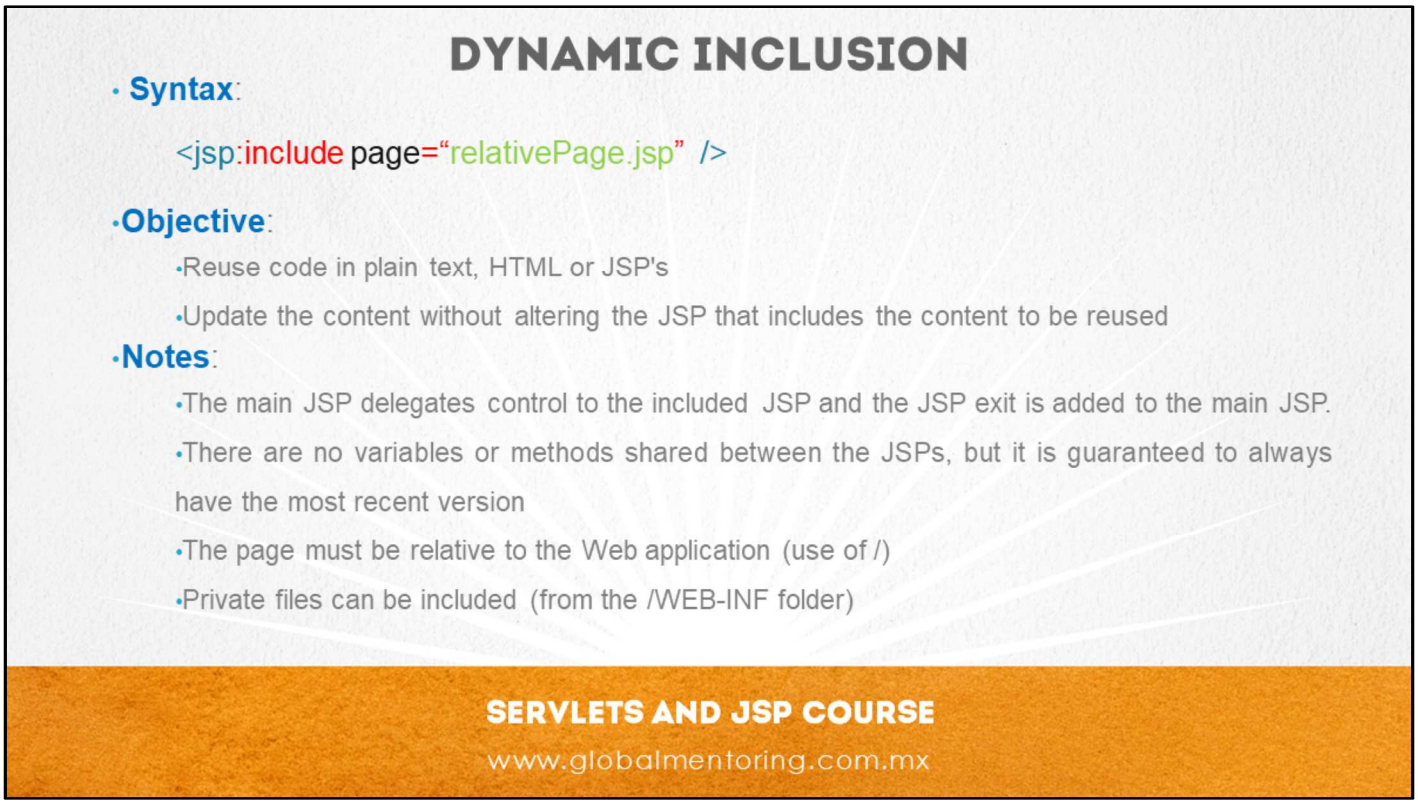
Once we already have a JSP or rather a Servlet in a unique way and that contains both JSPs, then we can share variables and methods among the different JSPs that we are using. One of the details is that, with the static inclusion, to have the most recent version of the JSP that we are including it is recommended to compile and reconstruct the project again because this static inclusion does not guarantee that we have the latest version of the included JSP. This is because the content of the included JSP is going to be flushed in the main JSP; this is going to be detailed later.



Let's review a diagram of a static inclusion. Suppose we have a main JSP called a.jsp and a JSP that we are going to include called b.jsp. In the first block of the diagram we are mentioning the jsp files with which we are going to work, in the second block we are working already in translation time of the JSP, that is, when a servlet is generated and in the third block we are specifying the time of execution, that is, when we send to call our JSP. Let's see in more detail each of the steps.

The main JSP (a.jsp) must add the directive as we have seen in the static inclusion syntax and it will add the contents of the included JSP (b.jsp). Once we request our resource a translation of the JSP is made, here the great difference of the static inclusion is that only one Servlet will be generated, and this Servlet will contain the code of the two JSPs (a.jsp and b. jsp) and the HTML code generated by each of the JSPs will be mixed in a single Servlet using the out.write () method. This html already includes the result of the two JSPs.

Later we will see an example of this type of inclusion.



Let's now review the dynamic inclusion.

The syntax as we have discussed is an action of the JSPs, an action has the following syntax:

<jsp: include page = "relativePage.jsp" /> open an angle bracket (<) later we put the prefix jsp and use two points (:) followed by the name include, this is the action that we are using the include action and later we specify by means of the page attribute the jsp page that we are going to include in our main jsp.

The objective of this dynamic inclusion is the same as the static inclusion, what we want to achieve is to reuse code whether it comes from plain HTML text or from some other source such as a JSP.

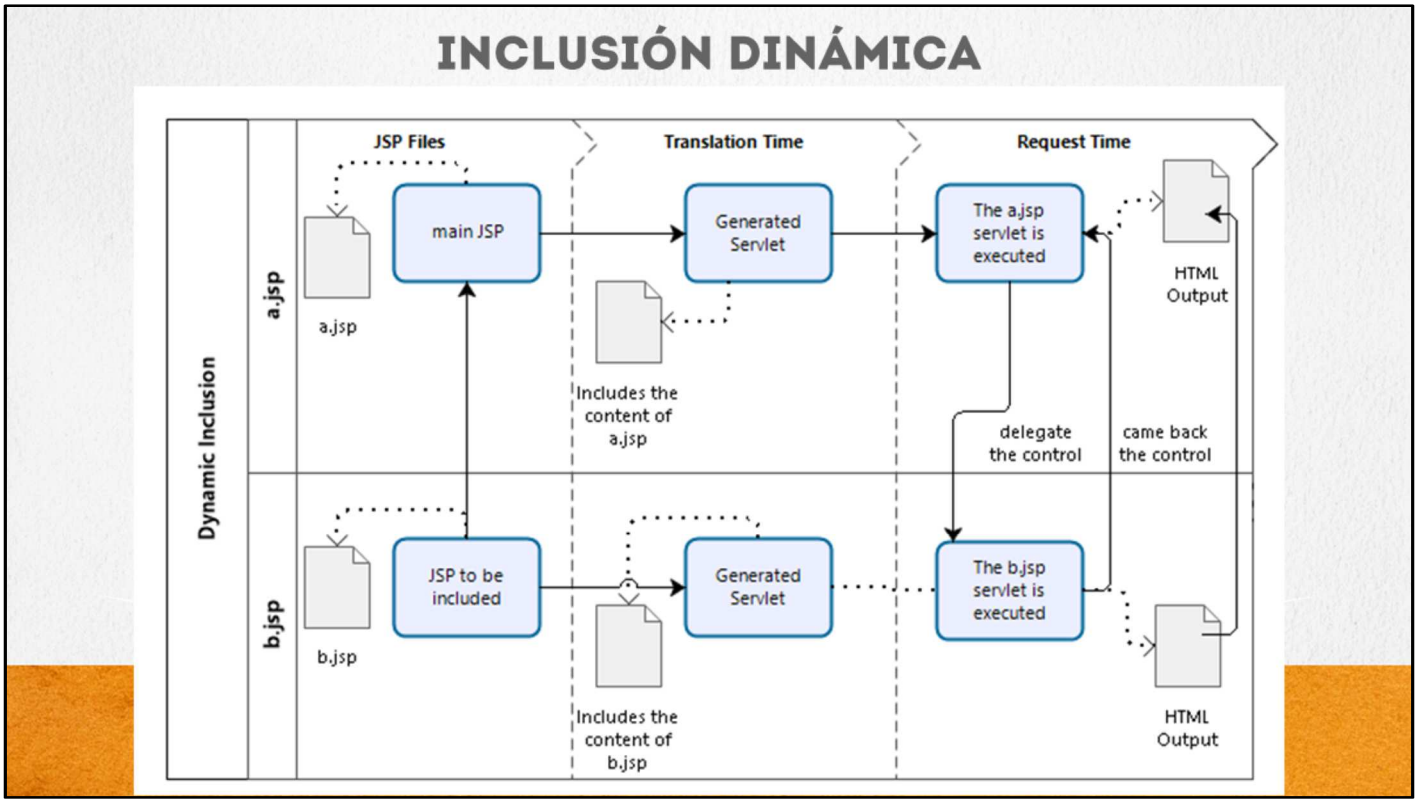
The advantage of using dynamic inclusion is that it will allow us to update the content of the main JSP without altering it. The main JSP instead of including the other JSP is going to delegate control to the included JSP and the output that is the result of running the included JSP will be added to the main JSP.

Due to this situation, 2 servlets are generated, one for each JSP, since each JSP will be executed independently. This implies that we will not be able to share variables or methods among the JSPs however this allows us to guarantee that we will always have the most recent version of the JSP that we are including, since every time we send a dynamic inclusion we will execute the JSP included independently because control is delegated to the included JSP. Once finished running that included JSP returns control to the main JSP.

Like static inclusion we must manage an inclusion relative to our web application, so it will be important to use a diagonal (/) and then we will indicate the web resource we are going to use.

Another advantage of dynamic inclusion is that we can also include private files, that is, files or JSPs that are inside the WEB-INF folder. Therefore we can access both public resources that are outside the WEB-INF folder or private resources, ie JSPs or resources that are within the WEB-INF folder.

Later we will carry out an exercise of this type of inclusion.



Let's now look at a dynamic inclusion diagram. We can see that now we have the same blocks but we are already separating the a.jsp and b.jsp files in more detail.

Also in the same way we observe the files that we are going to use (a.jsp and b.jsp), the translation time and we observe the execution time block.

The main JSP will use an action of the JSPs to include the b.jsp file inside the a.jsp file, later what happens in translation time is that two servlets will be generated, one corresponding to the main JSP (a. jsp) and the other corresponding to the JSP to include (b.jsp).

Then we can see that a JSP is generated with the contents of the a.jsp file and another servlet is generated with the contents of the b.jsp file, but here comes the interesting thing when we execute this JSP. What happens if we use a dynamic inclusion is that we are requesting the resource of a.jsp and as soon as this JSP detects the inclusion of the file we want to add then it deletes the control to the other servlet in this case it is the servlet associated with the jsp of b .jsp and as soon as it finishes executing this JSP will return the control to the main jsp and finally what it will do is generate an output, this output of the included servlet will be added within the main JSP, for that reason we can be sure that This JSP that is being included is always going to contain the latest version.

Finally, the output that is sent to the client will have both the output of the main JSP and the output of the JSP that was included.

COMPARISON BETWEEN TYPES OF INCLUSION

	Static Inclusion: <%@ include … %>	Dynamic Inclusion: <jsp:include …>
When does inclusion occur?	In translation time	At runtime
What is included?	The content of the file	The exit of the page
Number of generated Servlets	1	2
Can include headers that affect the main JSP	Yes	No
Can you include attributes or methods in common?	Yes	No
Should the main page be updated if the included page is updated?	Yes	No
Does the main JSP delegate control to the included JSP?	No	Yes

Finally we will review a comparison between the different types of inclusion.

The static inclusion in which we use a directive and the dynamic inclusion in which we use an action of the JSPs, we can observe when the inclusion occurs, in the case of a static inclusion occurs in translation time and in a dynamic inclusion occurs in time of execution.

We can also answer the question of What is included in the main JSP? In the case of static inclusion, the contents of the file will be included directly and in the case of dynamic inclusion, the output of the JSP page will be included.

The number of Servlets generated in a static inclusion is only 1 and in the case of a dynamic inclusion, two Servlets are generated, one for each JSP, the main one and the one to be included.

We can also answer the question, can we include headings that affect the main JSP? In the case of static inclusion, only one Servlet is generated, so we can modify the main JSP by means of headers from the JSP that we are including. In the case of dynamic inclusion this can not happen because the JSPs are executed independently, so there is no information shared between them, and no headers.

Regarding the question, can we include attributes or methods in common? In the case of static inclusion is true and is because it generates a single servlet therefore we will be able to share attributes or methods that are defined in one Servlet or another and in the case of dynamic inclusion this is not possible Because JSPs never mix, control is simply delegated and that is why it is false.

Regarding the question Should we update the main page if the included page is updated? In the case of static inclusion this is true to ensure that we have the latest version of the JSP included. In the case of dynamic inclusion this is not necessary because as we delegate the control the latest version of the included JSP is always running and in the case of the main JSP delegates control to the included JSP.

Next we will review some exercises to put into practice the use of static and dynamic inclusions using JSPs.