What Is JSP?

JSP is a collection of technologies developed by Sun Microsystems. It is used to develop web pages by inserting Java code into the HTML pages by making special JSP tags. It can consist of either HTML or XML (combination of both is also possible) with JSP actions and commands. The full form of JSP is Java Server Pages.

Why use JSP?

Here, are reasons of using JSP:

- In Java server pages JSP, the execution is much faster compared to other dynamic languages.
- It is much better than Common Gateway Interface (CGI).
- Java server pages (JSP)are always compiled before its processed by the server as it reduces the effort of the server to create process.
- Java server pages are built over Servlets API. Hence, it has access to all Java APIs, JNDI, JDBC EJB, and other components of java.
- JSP is an important part of Java EE (Enterprise Edition), which is a platform for enterprise-level applications.

Disadvantage of JSP

- It is hard to trace JSP pages error because JSP pages are translated to servlet.
- As JSP output is HTML, it is not rich in features.
- It is very hard to debug or trace errors because JSP pages are first translated into servlets before the compilation process.
- JSP pages require more disk space and time to hold JSP pages as they are compiled on the server.

Servlet Vs JSP

Servlet	JSP
Servlets run faster than JSP.	JSP runs slower than servlet as it takes time to compile the program and convert into servlets.

It is hard to write code in servlet.	It's easier to code in JSP compared to servlets.
In MVC architecture, servlet works as a controller.	In MVC architecture, JSP works as a view for displaying output.
It should be use when there is more data processing involved.	JSP is generally used when there is no involvement of much data processing.
There is no custom tag writing facility in servlets.	You can easily build custom tags that can directly call Java beans.
Servlet is a java code.	JSP is a HTML-based code.
It can accept all protocol requests, including HTTP.	It can only accept HTTP requests.
You can override the service() method.	In JSP, you can't override the service() method.
In Servlet, by default, session management is not enabled, user has to enable it explicitly.	In JSP, session management is automatically enabled.
In Servlet, you have to implement both business logic and presentation logic in the single file.	In JSP, business logic is split from presentation logic using JavaBeans.
Modification in Servlet file is a time consuming due to reloading, recompiling, and restarting the server.	JSP modification is fast, as you just need to click one refresh button.

Life cycle of JSP :

A JSP life cycle is defined as the process from its creation till the destruction. This is similar to a servlet life cycle with an additional step which is required to compile a JSP

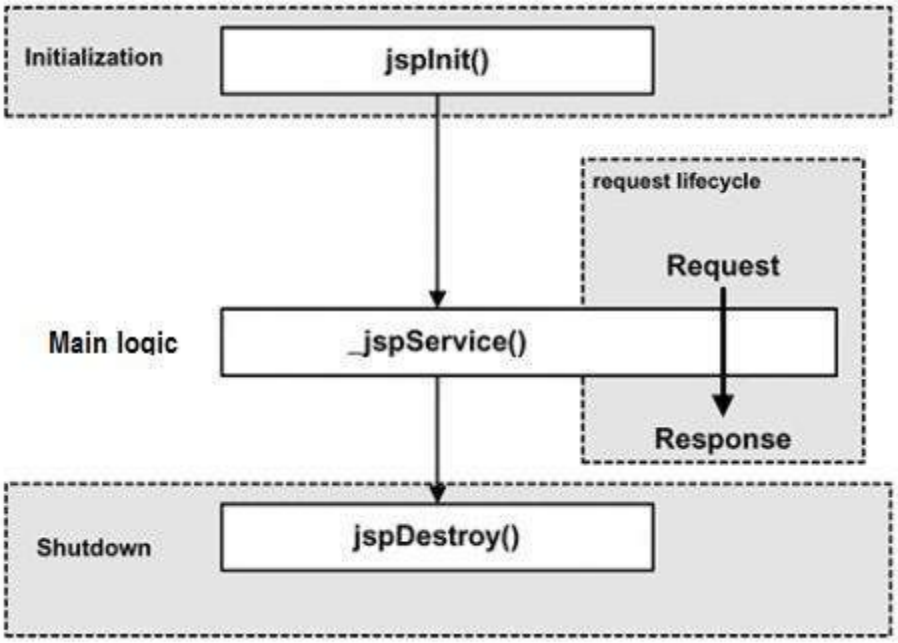
into servlet.

Paths Followed By JSP

The following are the paths followed by a JSP −

- Compilation
- Initialization
- Execution
- Cleanup

The four major phases of a JSP life cycle are very similar to the Servlet Life Cycle. The four phases have been described below −



JSP Compilation

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.

The compilation process involves three steps −

- Parsing the JSP.
- Turning the JSP into a servlet.

- Compiling the servlet.

JSP Initialization

When a container loads a JSP it invokes the **jspInit()** method before servicing any requests. If you need to perform JSP-specific initialization, override the **jspInit()** method −

```
public void jspInit(){
  // Initialization code...
}
```

Typically, initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the jspInit method.

JSP Execution

This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.

Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the **_jspService()** method in the JSP.

The _jspService() method takes an **HttpServletRequest** and an **HttpServletResponse** as its parameters as follows −

```
void _jspService(HttpServletRequest request, HttpServletResponse response) {
  // Service handling code...
}
```

The **_jspService()** method of a JSP is invoked on request basis. This is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods, i.e, **GET, POST, DELETE**, etc.

JSP Cleanup

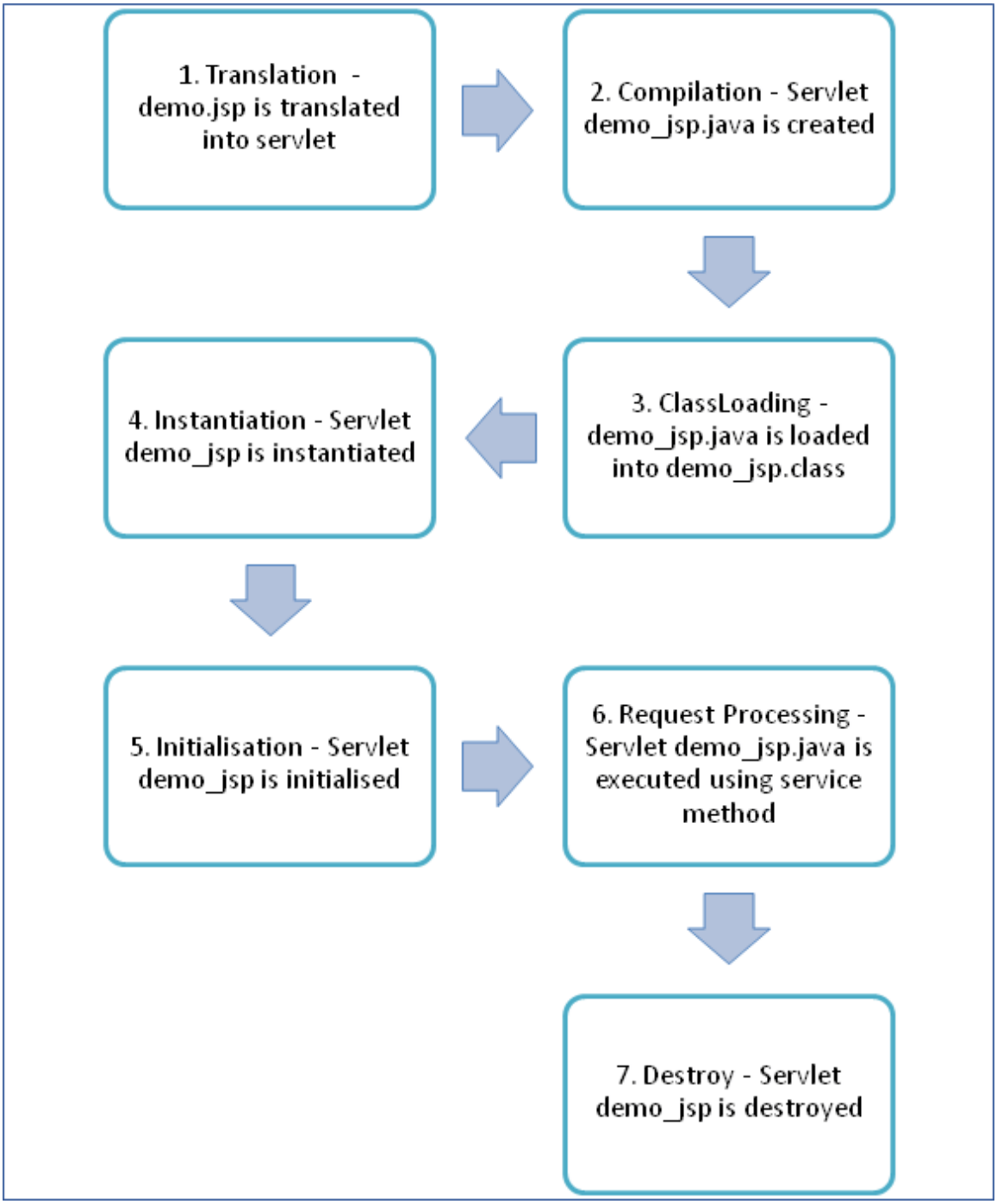
The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.

The **jspDestroy()** method is the JSP equivalent of the destroy method for servlets. Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.

The jspDestroy() method has the following form −

```
public void jspDestroy() {
  // Your cleanup code goes here.
}
```

How does a JSP function



Scripting Elements

JSP scripting elements allows you to insert Java code into Java Servlet generated from JSP page. These are the scripting elements:
scriptlet, declaration, expression & comment.

1) **Scriptlet**

- A scriptlet can contain any number of JAVA language statements, variable & expressions that are valid in the page scripting language.
- It can't contain method defination but you can call method within scriptlet

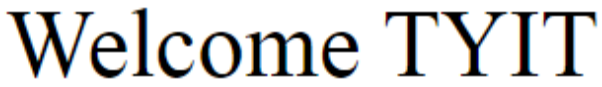
Following is the syntax of Scriptlet –
<% code fragment %>

Example of Scriptlet <% %>

1) Design a simple jsp page to display following

Input :-

Output :-



Client File name : - Index.html

```
<html>
   <head>
      <title>Scriptlet Example</title>

   </head>
   <body>
      <form action="server.jsp" method="post">
         Enter Name :<input type="text" name="t1"><br>

         <input type="submit" name="b1" value="send">
      </form>

   </body>
</html>
```

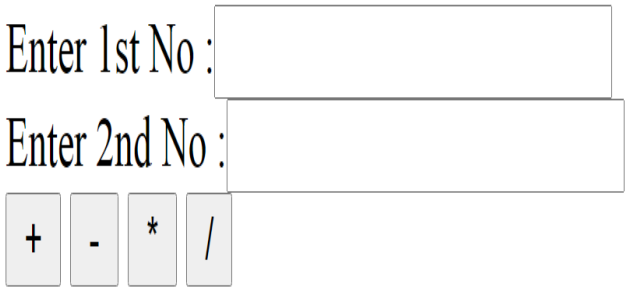
Server File name :- server.jsp

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

      <title>Scriptlet Example</title>
   </head>
   <body>
      <%

        String n=request.getParameter("t1");
        out.println("Welcome "+n);
      %>
   </body>
</html>
```

Example

2) Design a simple calculator

Input: -



Output: -

Addition=15

Client File Name:- index.html

```
<html>
   <head>
      <title>Simple Calculator</title>
   </head>
   <body>

      <form action="calculator.jsp" method="get">
         Enter 1st No :<input type="text" name="t1"><br>
         Enter 2nd No :<input type="text" name="t2"><br>
         <input type="submit" name="add" value="+">
```

```
        <input type="submit" name="sub" value="-">
        <input type="submit" name="mult" value="*">
        <input type="submit" name="div" value="/">
      </form>

  </body>
</html>
```

Server File name :- calculator.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
  <head>
     <title>Calculator</title>
  </head>
  <body>
    <%
    int c=Integer.parseInt(request.getParameter("t1"));
    int d=Integer.parseInt(request.getParameter("t2"));
   if(request.getParameter("add")!=null)
   {
    int e=c+d;
    out.println("Addition="+e);
   }
   if(request.getParameter("sub")!=null)
   {
    int f=c-d;
    out.println("Subtraction="+f);
   }
   if(request.getParameter("mult")!=null)
   {
    int g=c*d;
    out.println("Multiplication="+g);
   }
   if(request.getParameter("div")!=null)
   {
    int h=c/d;
    out.println("Division="+h);
   }
    %>
```

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

2) Declaration tag

- If you want to define methods or fields, you can use JSP declaration. The JSP declaration is surrounded by the sign <%! and %>. For example, if you want to declare a variable x, you can use JSP declaration as follows:

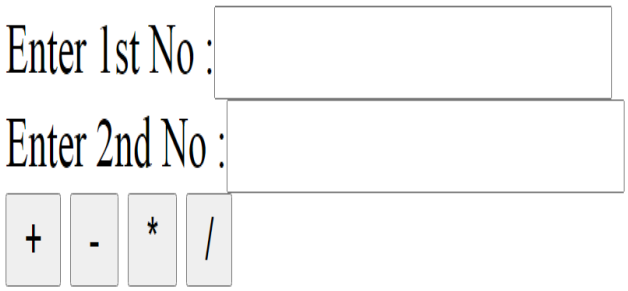
```
<%!  int x = 10;  %>
```

- The final semicolon is required.
- The difference between a variable using declaration and a variable is declared using scriptlet is that a variable declared using declaration tag is accessible by all methods, while a variable declared using scriptlet is only accessible to the *_jspservice()* method of the generated servlet from the JSP page.

Example

1) Design a simple calculator by using Declaration tag

Input: -



Output: -

Addition=15

Client File name :- index.html

```
<html>
  <head>
    <title>Calculator using Declaration tag</title>
  </head>
```

```
        <body>
            <form action="declaration_server.jsp" method="get">
                Enter 1st No :<input type="text" name="t1"><br>
                Enter 2nd No :<input type="text" name="t2"><br>
                <input type="submit" name="add" value="+">
                <input type="submit" name="sub" value="-">
                <input type="submit" name="mult" value="*">
                <input type="submit" name="div" value="/">
            </form>

        </body>
    </html>
```

Server File name :- declaration_server.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <title>Simple Calculator using Declaration</title>
    </head>
    <body>
        <%!
        int i=0,j=0;
        %>
        <%
         i=Integer.parseInt(request.getParameter("t1"));
         j=Integer.parseInt(request.getParameter("t2"));
         if(request.getParameter("add")!=null)
         {
            int s=addition();
            out.print("Addition="+s);
         }
         if(request.getParameter("sub")!=null)
         {
            int s=subtraction();
            out.print("subtraction="+s);
         }
         if(request.getParameter("mult")!=null)
         {
```

```
        int s=multiplication();
        out.print("Multiplication="+s);
      }
      if(request.getParameter("div")!=null)
      {
        int s=division();
        out.print("Division="+s);
      }
      %>
      <%!
      public int addition()
      {
        int k=i+j;
        return k;
      }
      public int subtraction()
      {
        int k=i-j;
        return k;
      }
      public int multiplication()
      {
        int k=i*j;
        return k;
      }
      public int division()
      {
        int k=i/j;
        return k;
      }
      %>
  </body>
</html>
```

3. JSP expression tag

The code placed within **JSP expression tag** is *written to the output stream of the response*. So you need not write out.print() to write data. It is mainly used to print the

values of variable or method.

This tag allow the developer to embed any java expression and is short for out.println().

A semicolon (;) does not appear at the end of the code inside the tag.

Syntax:-

<%= statement %**>**

Example :

File : index.html

```
<html>
<body>
<form action="welcome.jsp" method=”post”>
<input type="text" name="uname"><br/>
<input type="submit" value="go">
</form>
</body>
</html>
```

File : welcome.jsp

```
<%@page contentType=”text/html”%>
<html>
<body>
<%
 String s = request.getParameter("uname") ;
%>
<%= "Welcome "+s %>
</body>
```

</html>

4. JSP comment tag

JSP comments are simillar to HTML comments <!--Html comment --> except JSP comments are never sent to the user's browser.

HTML Comments are visible in the page sourse.

Syntax:- <%-- JSP Comment --%>

Example:-

```
<html>
<head>
    <title>HTML and JSP Comments</title>
</head>
<body>
<!--This is HTML comment; visible in the page source-->
<%-- This is JSP comment; invisible in the page source --%>

</body>
</html>
```