Web Technology (KCS-602)
CSE-3rd Year

By
Mr.Ganesh Kr. Yadav
Assistant Professor
Department of Computer Science & Engineering,
ABESIT College of Engineering, Ghaziabad.

Web Technology (KCS-602)

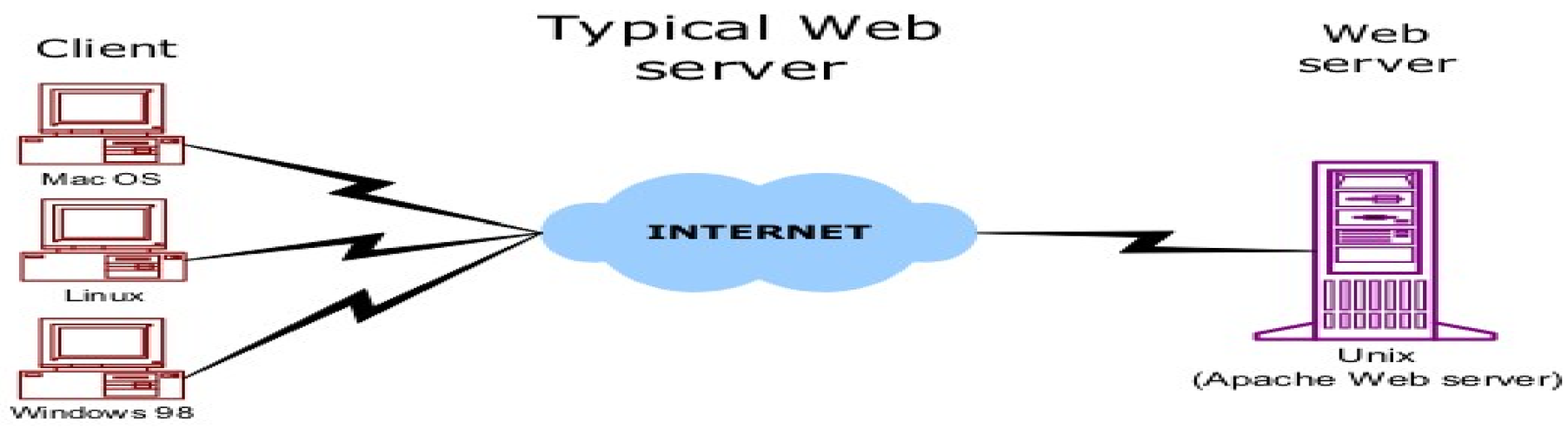
UNIT-5 (Syllabus)

Introduction of JSP

- **Introduction, Java Server Pages Overview,**
- **A First Java Server Page Example,**
- **Implicit Objects,**
- **Scripting,**
- **Standard Actions,**
- **Directives,**
- **Custom Tag Libraries.**

JSP (Java Server Pages)

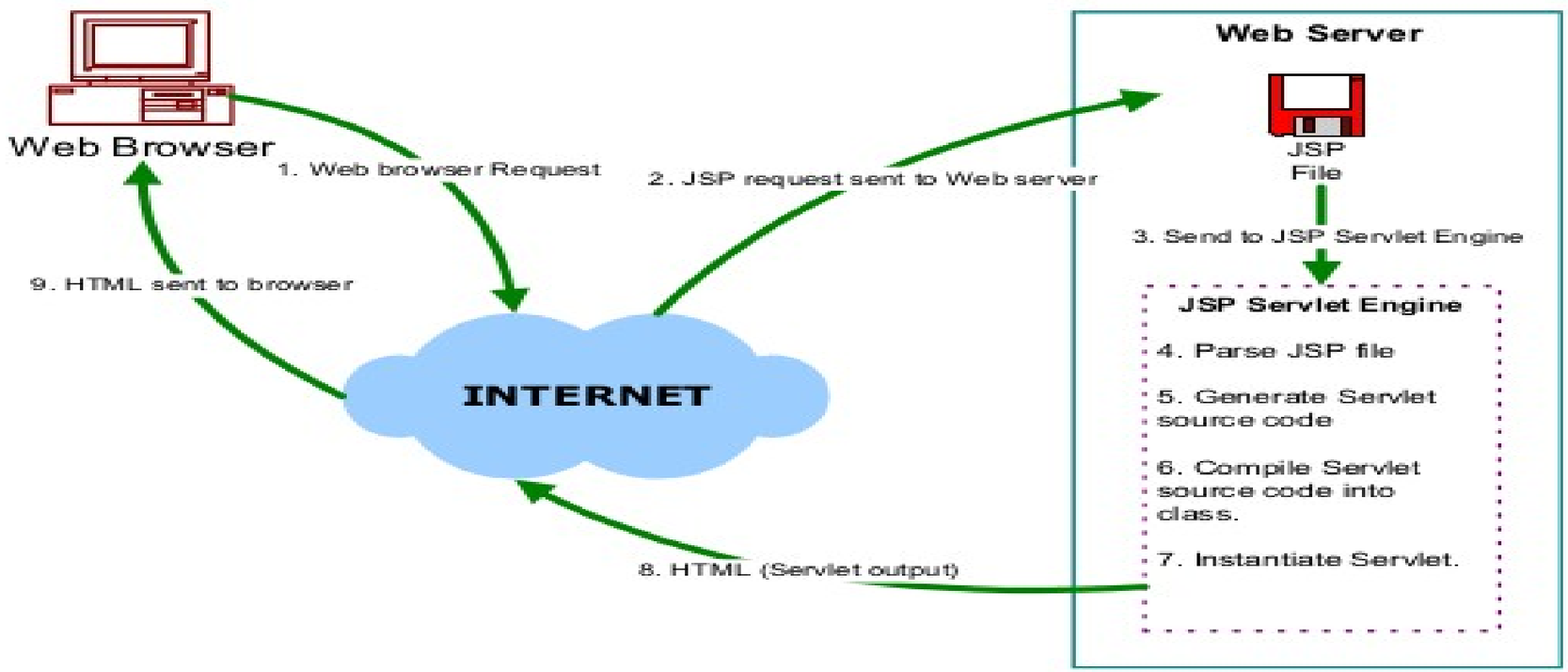
- Java Server Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications.
- JSP have access to the entire family of Java APIs, including the JDBC API to access enterprise databases.
- JSP technology is used to create web application just like Servlet technology.
- It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.
- A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development.
- It provides some additional features such as Expression Language, Custom Tags, etc.



JSP Architecture

Steps required for a JSP request:

1. The user goes to a web site made using JSP. The user goes to a JSP page (ending with .jsp). The web browser makes the request via the Internet.
2. The JSP request gets sent to the Web server.
3. The Web server recognises that the file required is special (.jsp), therefore passes the JSP file to the JSP Servlet Engine.
4. If the JSP file has been called the first time, the JSP file is parsed, otherwise go to step 7.
5. The next step is to generate a special Servlet from the JSP file. All the HTML required is converted to println statements.
6. The Servlet source code is compiled into a class.
7. The Servlet is instantiated, calling the *init* and *service* methods.
8. HTML from the Servlet output is sent via the Internet.
9. HTML results are displayed on the user's web browser.



JSP Processing

As shown in figure, the client sends the request to server, the server have container in that the container logic reads the sample.JSP file and converts into sample.java file. This is called Translation phase.

And then, the sample.java file converted into sampl class file to request processing, then the container gener the response and that finally send to client.

⇒ Translation phase :-

Generating the .java file from the .jsp file.

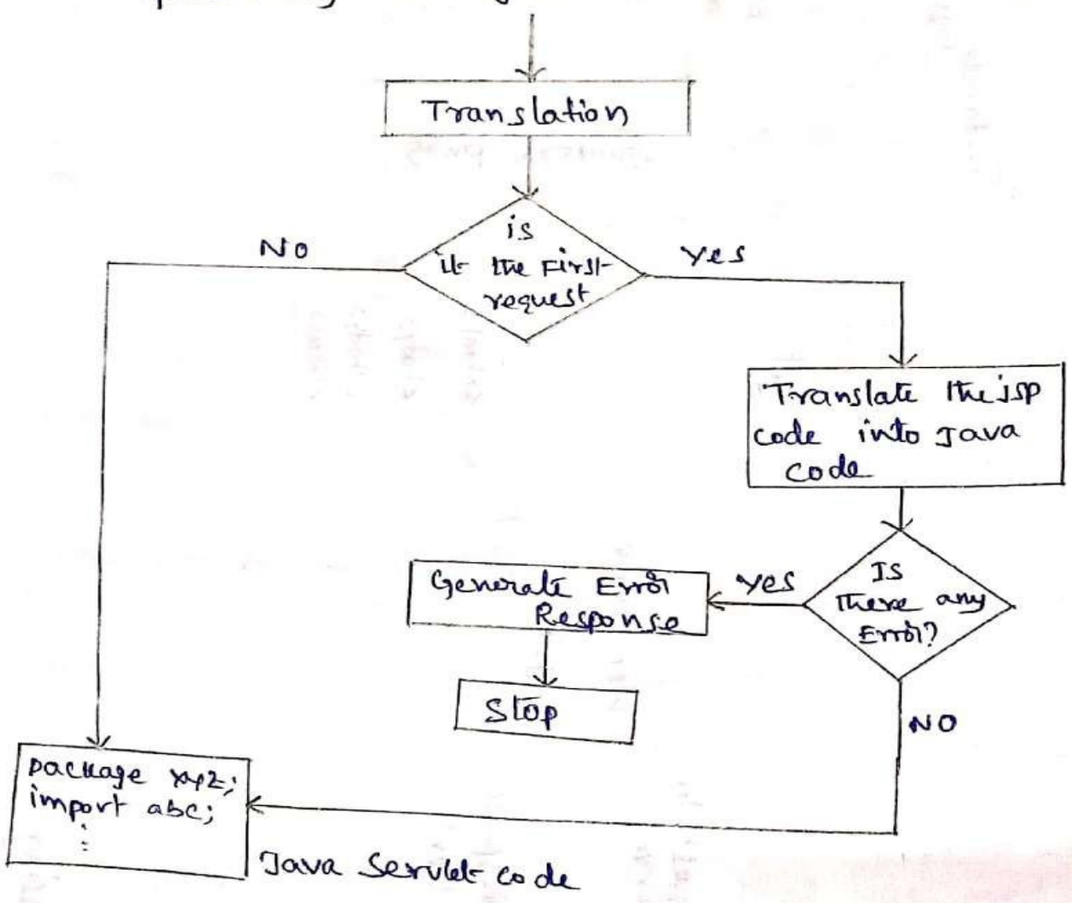


Fig:- Translation phase.

⇒ Request processing phase :-

Generating the .class file from the .java file and handling the request.

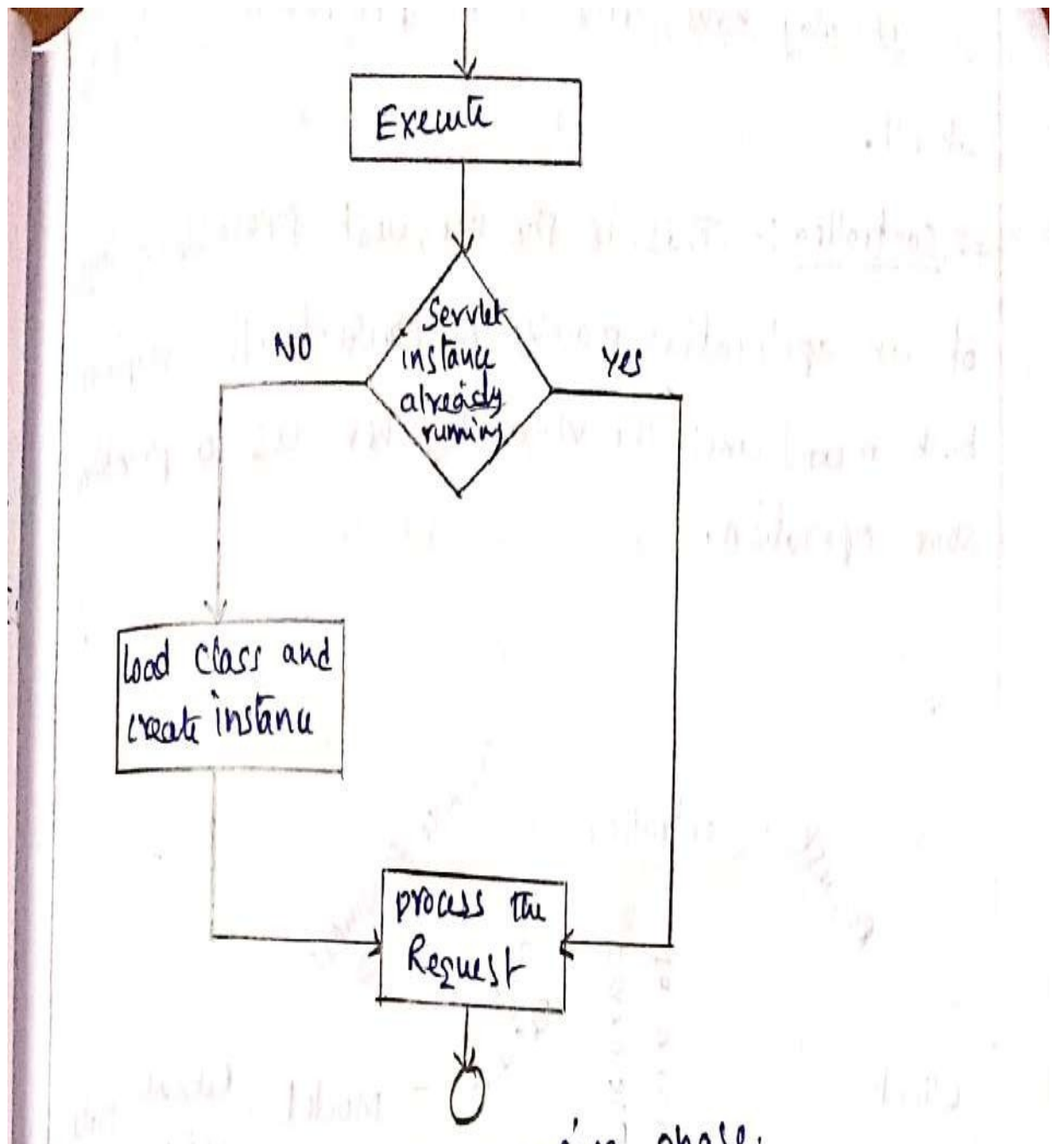


Fig:- Request processing phase.

The Lifecycle of a JSP Page

The JSP pages follow these phases:
o Translation of JSP Page

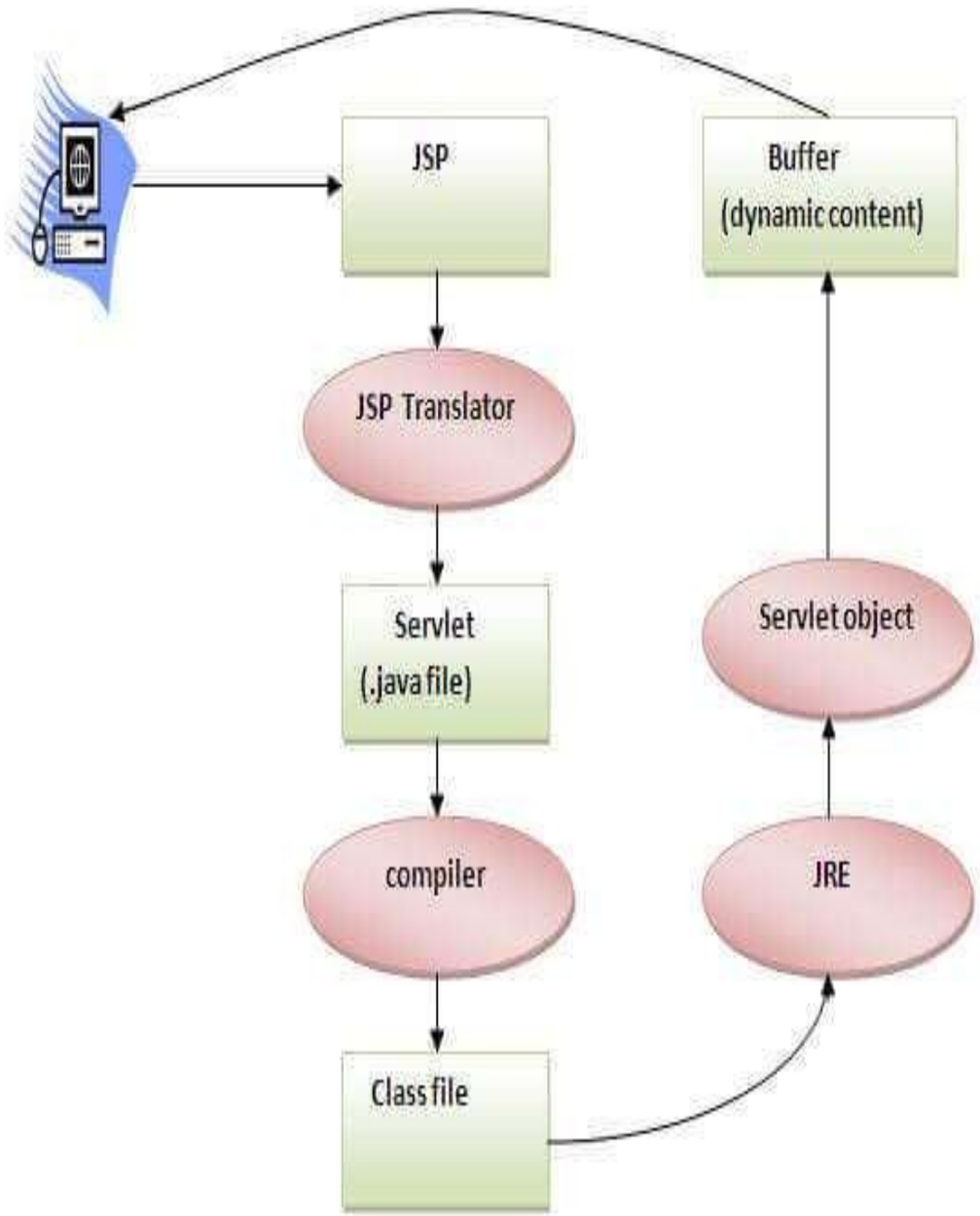
o Compilation of JSP Pageo Classloading (the classloader loads class file)

o Instantiation (Object of the Generated Servlet is created).

o Initialization (the container invokes jspInit() method).

o Request processing (the container invokes _jspService() method).

o Destroy (the container invokes jspDestroy() method).



Index.jsp

```
<html>
<body>
<% System.out.println(“Web Tech”); %>
</body>
</html>
```

JSP Implicit Objects

- These objects are created by the web container that are available to all the jsp pages. A list of the 9 implicit objects is given below:

Object	**Type**
out	JspWriter
request	HttpServletRequest
response	HttpServletResponse
config	ServletConfig
application	ServletContext
session	HttpSession
pageContext	PageContext
page	Object
exception	Throwable

Example of out implicit object

- For writing any data to the buffer, JSP provides an implicit object named out.
- It is the object of JspWriter. In case of servlet you need to write:1. PrintWriter out=response.getWriter();
- But in JSP, you don't need to write this code.

index.jsp

```
<html>
<body>
 <% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
</body>
</html>
```

JSP Element

- **Directives:** The jsp directives are messages that tells the web container how to translate a JSP page into the corresponding servlet.

  There are three types of directives**:** **page , include, taglib**

  <%@ directive attribute="value" %>

- **Scriptlet:** The scripting elements provides the ability to insert java code inside the jsp.

  <% java source code %>

- **Expression**: 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.

  <%= statement %>

- **Declaration:** The JSP declaration tag is used to declare fields and methods.

  <%! field or method declaration %>

- **Action: <% jsp: action type attribute %>**

Example:- "Directive.JSP"

```
<%@ page import = "java.util.Date" %>
<html>
<body>
<% Date date = new Date ();
    System.out.println ("server time is now:");
    System.out.println (date);
%>
Going to include welcome.html file <br>
<%@ include file = "welcome.html" %>
</body>
</html>
```

welcome.html

```
<html>
<body>
<h2> Welcome to JSP programming </h2>
<h3> Thank you </h3>
</body>
</html>
```

scriptlet

- File: **index.html**

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

- File: **welcome.jsp**

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
</form>
</body>
</html>
```

- **JSP expression tag:**

```
<html>
 <body>
<%= "welcome to jsp" %>
 </body>
 </html>
```

JSP Declaration tag:

```
<%!

    private int counter = 0 ;
    private String get Account ( int accountNo) ;
%>
```

Action tag

There are three main roles of action tags :

1)enable the use of server side Javabeans
2)transfer control between pages
3)browser independent support for applets.

Javabeans

A Javabean is a special type of class that has a number of methods. The JSP page can call these methods so can leave most of the code in these Javabeans. For example, if you wanted to make a feedback form that automatically sent out an email. By having a JSP page with a form, when the visitor presses the submit button this sends the details to a Javabean that sends out the email. This way there would be no code in the JSP page dealing with sending emails (JavaMail API) and your Javabean could be used in another page (promoting reuse).

The following is a list of Javabean scopes:

page – valid until page completes.
request – bean instance lasts for the client request session – bean lasts for the client session.
application – bean instance created and lasts until application ends.

Dynamic JSP Include

You have seen how a file can be included into a JSP using an Include Directive:

```
<%@ include file = “include/privacy.html %>
```

This is useful for including common pages that are shared and is included at compile time.

To include a page at run time you should use dynamic JSP includes.

```
<jsp:include page="URL" flush="true" />
```

⇒ 3) Action Elements :-

Specific functionality is encapsulated by action elements in predefined tags, to be used by programmer in JSP page.

At translation time these actions elements are replaced by Java code which corresponds to specify functionality.

The action elements can dynamically generate HTML.

The action Elements should be represented using strictly XML syntax as shown below

```
<prefix: action    [attribute, = "value; ---] >
         ↓                    ↓
      Tag Name             optional
  // body
</prefix : action>
```

There are standard actions all are have 'jsp' as their prefix. Hence, they have the following syntax.

```
<jsp: tagname [attribute, = "value,"-- ]>
  // body
</jsp: tagname>
```

The Jsp container supports the following standard action Elements

- <jsp: include> — Dynamically includes the content other resources at request process
- <jsp: forward> — Terminates the current Jsp page execution, and forwards the Http request to another Jsp for processing
- <jsp: useBean> — specifies that a JavaBean instance is used by the Jsp page.
- <jsp: Element> — Dynamically generates an XML elements.
- <jsp: body> — Specifies the body of a tag.
- <jsp: text> — Encloses template data.

Example:-

```
<%@ page import = "java.util.*" %>
<html>
<body>
Here including the action
<jsp: include page = "abc.html" />
Here forwarding the page
<jsp: forward page = "ay2.jsp" />
</body>
</html>
```

<%--WAP to count the no. Of visit by visitor on a website --%>

```
<HTML>
<HEAD>
<TITLE> JSP Example 2</TITLE>
</HEAD>
<BODY> JSP Example 2
<BR>
<%!
String sitename = "Programming Distributed Applications";
int counter = 0;
private void incrementCounter()
{
counter ++;
}
%>

Website of the day is
<%= sitename %>
<BR>
page accessed
<%= counter %>
</BODY>
</HTML>
```

Implicit Objects

The developer can create Javabeans and interact with Java objects.

- There are several objects that are automatically available in JSP called implicit objects.
- There is no need to call the object of each inbulild libraries.
- The implicit objects are:

Variable	Of type
request	javax.servlet.http.httpservletrequest
response	javax.servlet.http. httpservletresponse
out	javax.servlet.jsp.jspwriter
session	javax.servlet.http.httpsession
pagecontent	javax.servlet.jsp.pagecontext
application	javax.servlet.http.servletcontext
config	javax.servlet.http.servletconfig
page	java.lang.object
exception	java.lang.throwable

The following are the main steps involved:

get the value of the session variable - visitcounter
if the session variable (visitcounter) is null
set the session variable to 0 and welcome the visitor.
if the session variable is not null (after step 2), increment the session varaible and display the number of visits.

```
<!-- session.jsp  checks to see if you have visited a page and keeps a counter.
-->   <html> <head></head> <body>
<%   // get the value of the session variable - visitcounter
Integer totalvisits = (Integer)session.getValue("visitcounter");
// if the session variable (visitcounter) is null
      if (totalvisits == null)
{
// set session variable to 0
totalvisits = new Integer(0);
session.putValue("visitcounter", totalvisits);
// print a message to out visitor
out.println("Welcome, visitor");   }
else
{
// if you have visited the page before then add 1 to the visitcounter
totalvisits = new Integer(totalvisits.intValue() + 1);
session.putValue("visitcounter", totalvisits);
out.println("You have visited this page " + totalvisits + " time(s)!");
}   %>   </body>    </html>
```

* Database programming using JDBC :-

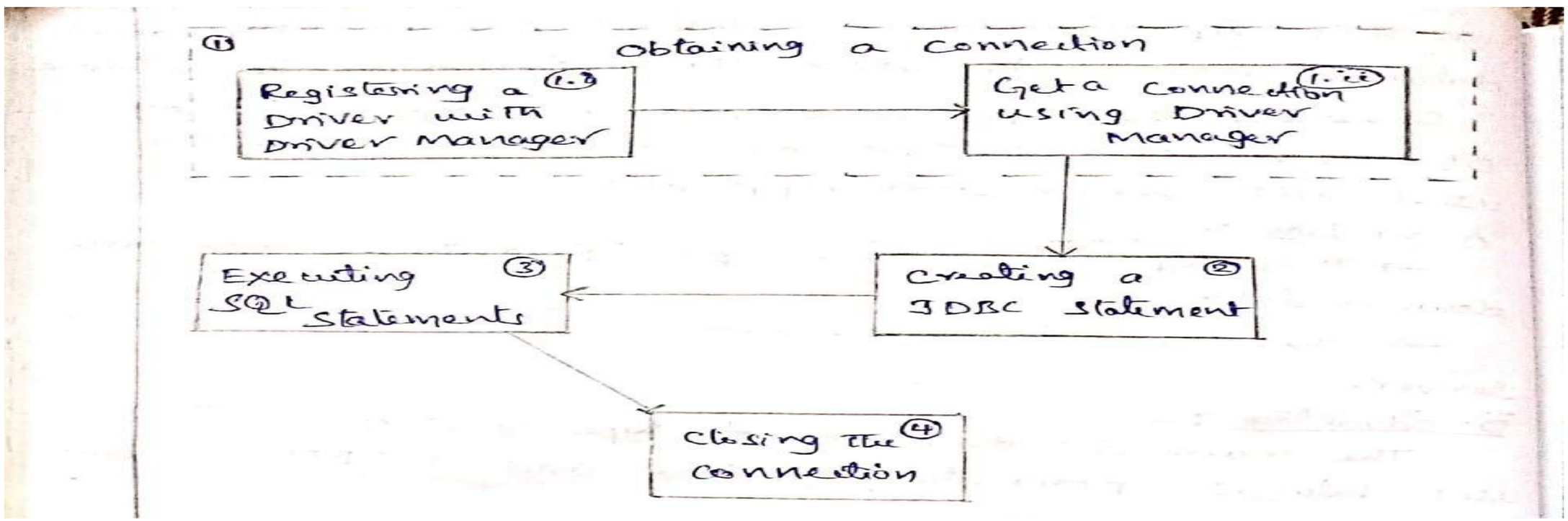
JDBC APIs are used by a Java application to communicate with a database.

In otherwords, we use JDBC to communicate with a database, and the communicate implemented by the JDBC API.

The JDBC application-specific code should be written with in an application that has to communicate with the database.

There are some basic steps in JDBC connectivity. those are

Step 1 → Obtaining a connection
Step 2 → Creating a JDBC statement
Step 3 → Executing SQL statements
Step 4 → Closing the connection.



* open connectivity code :

```
<%-- 'import the java.sql package --%>
<%@ page language="Java" import="java.sql.*"%>
<%
 try
 {
 // Load driver class file
 DriverManager.registerDriver(new <driver class>);
 Connection conn = DriverManager.getConnection(url,
                                               uname, pwd)
 }
 %>
```

* Statement Code :

```
<% // create the Statement
 Statement stmt = conn.createStatement();
 String query = "SELECT * FROM student";
 stmt.ExecuteQuery(query);
 %>
```

* close connectivity code :

```
<%  // close the Statement
     stmt.close();
     // close the connection
     conn.close();
 %>
```

By using presentation code we can presenting the result with in the browser.

The presentation code can be written by using HTML.

* open connectivity code:

```
<!-- import the java.sql package -- %>
<%@ page language="Java" import="java.sql.*" %>
<%
 try
 {
 // Load driver class file
 DriverManager.registerDriver (new <driver class>);
 Connection conn = DriverManager.getConnection(url,
                                        uname, pwd)
 }
 %>
```

* Statement Code:

```
<% // create the Statement
 Statement stmt = conn.createStatement ();
 String query = "SELECT * FROM student";
 Stmt. ExecuteQuery (query);
 %>
```

* close connectivity code:

```
<%. // close the Statement
      stmt.close();
      // close the connection
      conn.close();
    %>.
```

By using presentation code we can presenting the result with in the browser.

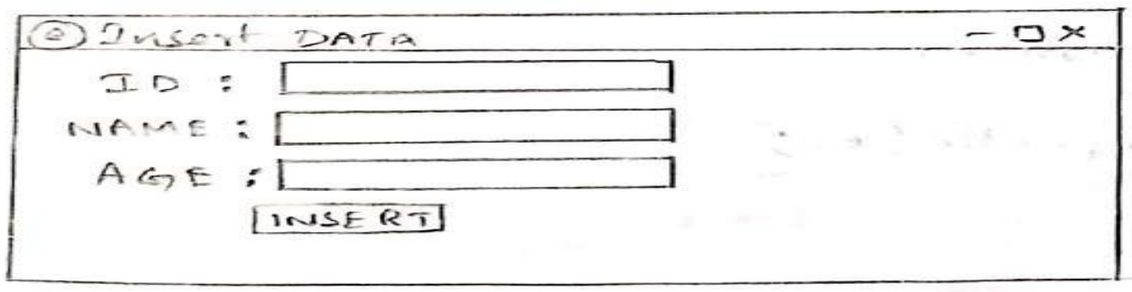
The presentation code can be writtern by using HTML.

Example :- program for inserting the records into database by using JSP.

insertdb.html

```
<html>
<head>
<title> Insert DATA </title>
</head>
<body>
<form action = "insert.jsp">
ID : <input type = "text" name = "ID" /> <br/>
NAME : <input type = "text" name = "NAME" /> <br/>
AGE : <input type = "text" name = "AGE" /> <br/>
<input type = "submit" value = "INSERT" />
</form>
</body>
</html>
```

O/p:-



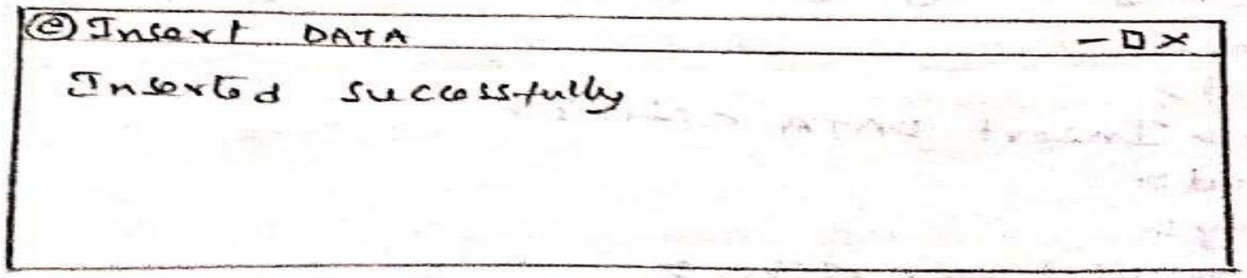
insert.jsp :

```
<%@ page language = "java" import = "java.sql.*" %>
<html>
<head>
<title> Insert DATA </title>
</head>
<body>
<h2> Welcome </h2>
```

```
<%
    int id = Integer.parseInt(request.getparameter("ID"));
    int age = Integer.parseInt(request.getparameter("AGE"));
    int name = Integer.parseInt(request.getparameter("NAME"
try
{
 DriverManager.registerDriver(new sun.jdbc.odbc.
                                    jdbcodbcDriver());

 Connection con = DriverManager.getConnection
                        ("jdbc:odbc:students");
 Statement s = con.createStatement();
 String query = "insert into students(SID,SNAME,AGE
                        values(id, name, age)";
 s.executeUpdate(query);
 out.println("Inserted successfully");
 s.close();
 con.close();
}
catch(Exception e)
{
 System.out.println(e);
}
%>
</body>
</html>.
```

O/p:-



* Deploying JavaBeans in a JSP page :-

JavaBeans are reusable software components that separates the business logic from the presentation logic.

In general JavaBeans are simple Java classes that follow certain specifications to develop dynamic content.

JavaBeans are easier to write, compile, test, debug and reuse. JavaBeans uses getter and setter methods to invoke various functionality with JSP pages.

The JSP:useBean action lets you load a bean to used into JSP page.

The simplest syntax for specting a bean should be used is

<JSP:useBean id = "name" class = "package.class" />

This usually means "instantiable an object of the class specified by class and binds it to a variable".

Example :-

Deploying JavaBean in a Jsp page.

SimpleBean.java

```
package msgbeans
public class SimpleBean
{
 private string message = " message not yet set" ;
 public string getmessage()
 }
   return (message);
 }
  public void setmessage(string message).
  {
    this . message = message;
  {
```

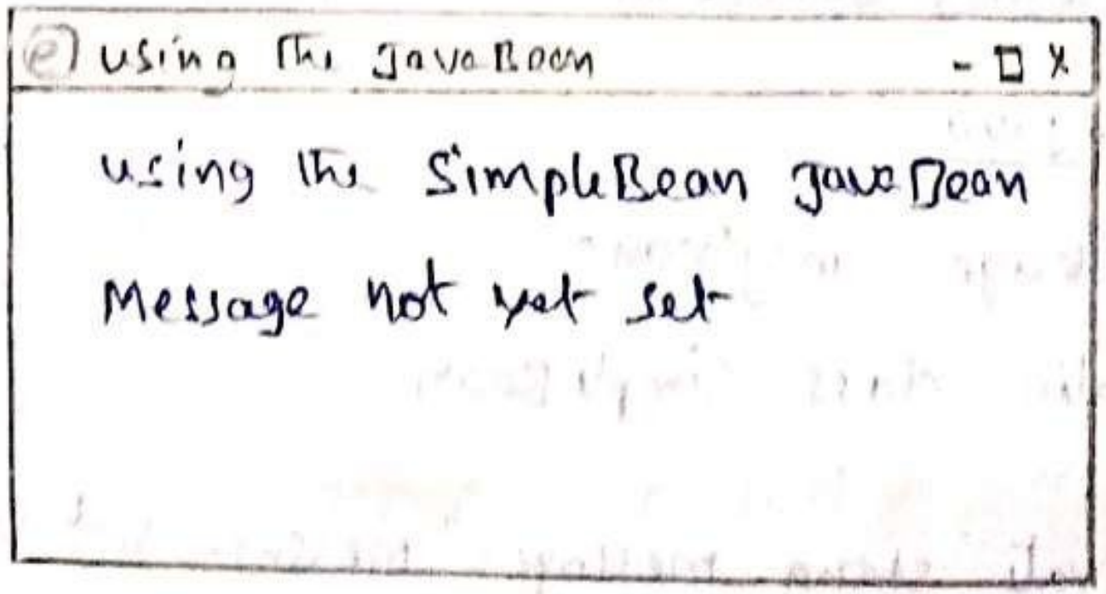
UseBeans.jsp

```
<html>
<head>
<title> using the JavaBean </title>
</head>
<body>
<h2>
 using the SimpleBean JavaBean
 </h2>
 <jsp: useBean id = "Simple Bean" class = "msgbeans.
                                          simple Bean" />

 <jsp : get property  name = "messageBean"
                                property = " Message"/
 <jsp : Set property  name = " messageBean"
                property = " message" value = " hello" />

 </body>
 </html>
```

O/P :-


using the JavaBean
- □ X
using the SimpleBean JavaBean
Message not yet set

Thank You