* Introduction to JSP :-

JSP - Java Server page.

→ JSP is a Technology based on the java language and enables the development of dynamic web sites.

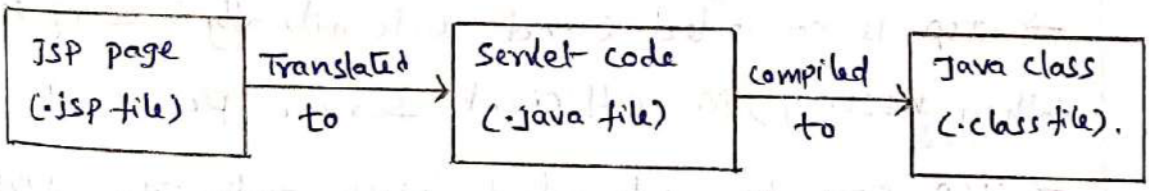
→ JSP was developed by sun micro systems to allow us to server side development.

→ A JSP page is a textual document that describes how to create a response object from request object for given protocol.

→ The main goal of JSP technology is to simplify the Certain and maintenace of server side html pages.

→ A JSP page contains standard markup elements such as HTML tags just like a regular web pages. And it contains special JSP elements that allows the server to insert dynamic content in the page.

The phases of a JSP page



* The problem With Servlet :-

programming with servlets is a very boring and continuing for long time, as a servlets not only handle the request processing and bussiness logic but also the presentation task.

The servlet has some problems as described below

→ It is a common fact that servlets incorporate both the processing logic and presentation logic.

→ In the servlet, every single change in the presentation logic of an application, requires the servlet programmer

to update the servlet code and recompile it.

→ Web page development tools do not directly support servlets when designing application.

Due to these problems, sun microsystems developed JSP Technology.

→ With JSP Technology the presentation logic may contain both static and dynamic content thereby making the web development very easy and flexible.

→ In JSP Technology, the business logic and request process logic are separated from presentation logic.

→ JSP provides a very powerful and flexible mechanism to provide dynamic web pages.

→ JSP doesn't require any special setup at the client-side.

→ JSP provides built in supports for HTTP session management.

→ JSP is compiled (and automatically recompiled when nessary) for efficient server processing.

→ JSP can be integrated with enterprise Java API.

- → JDBC API
- → Thread API
- → EJB API

→ Web development tools gives full support to JSP because JSP pages are like HTML pages.

---

* In linux (ubuntu), JSP files will be saved in

  /var - lib - tomcat7 - webapps

* To run localhost:8080/foldername/filename(JSP).

  ↓ port no. of Tomcat to servelet, # JSP file

* The Anatomy of Jsp page :-

The Jsp page is almost like a regular web page with the inclusion of dynamic behaviour through Jsp elements.

Hence we can say that a Jsp page has two components.

⇒ **Jsp Elements:-** These are the Things that the Jsp container understands and translates. Hence we can say that Jsp elements instructs the Jsp container, what code to generate and how it should operate.

⇒ **Template Data:-** Everything else in the page that the Jsp container doesnot understand is called template data.

Actually these are the static portion of the Jsp page which are passed through the Jsp container unprocessed to the browser.

Example:-

```
<%@ page import = "java.util.Date" session = "false" %>
<html>
<head>
<title> simple Jsp page </title>
</head>
<body>
<h3> current time is : </h3>
<%= new Date() %>
</body>
</html>
```

Note:- The Jsp container after processing the Jsp page, merges both, the static data (template data) and dynamic data, generated by processing the Jsp elements. And finally the resultant content is sent to the browser as a response.

↳ Jsp Elements :-

All the Jsp elements may be written in one of two forms.

→ The xml form

→ The <%...%> form

These are four types of Jsp elements, They are

1) Directive Elements 2) scripting Elements

3) Action Elements 4) Expression language expressions.

⇒ 1) Directive Elements :-

Directives gives message to Jsp Container, during Translation to generate the corresponding servlet and to control some characteristics of the Jsp page.

Directives have the following general form

<%@ directive [attribute1 = value1 ...] %>

↓ Name of directive

↓ optional

These are three standard directives they are

(a) page (b) include (c) taglib.

- (a) page :- It is used to specify current page settings as a whole.

Syntax :- <%@page [attribute1 = "value1 ...] %>

It defines following attributes

(i) import (ii) contentType (iii) session (iv) buffer.

- (b) include :- It is used to include content from other resources during the translation time.

Syntax :- <%@include file = "url" %>

(c) taglib :- functionality by making use of custom actions defined in the tag library.

Syntax :- `<%@ taglib uri = "liburi" prefix = "name" %>`

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>
```

⇒ 2) Scripting Elements :-

Scripting elements enable programmers to embed code in another programming language (Java) within a JSP page.

They are actually executed at request processing time and are used for a variety of purposes.

Such as

→ To manipulate objects

→ To perform calculation on runtime values.
→ To perform computation on an object.
→ To declare methods & variables.

Scripting elements are 3 types. They are

→ (a) Scriptlets :-

Scriptlets are the code fragments of a language which are executed at run time to process an Http reque

Scriptlets have following form:

<% Java code %> (or) <% statement, [statement2 ...] %>

→ (b) Declarations :-

Declaration enable a programmer to declare variables and methods of a language, which can be us from any point in the Jsp page.

unlike scriptlets they cannot access Jsp implict objects

Declarations have following form:

<%! statement, [statement2 ...] %>

→ (c) Expressions :-

Expressions enables a programmer to write expres in a language which gets evaluated at request processi time. These are used to insert values directly into the output.

It has following form:

<%= expression %>

Example :- "script.Jsp"

```
<%@ page import = "java.util.Date" %>
<html>
<body>
<%
    system.out.println ("Evaluating date now");
```

```
    Date date = new Date();
%>
Hello! The time is now
<% out.println(date); %>
<%! Date d = new Date();
    Date getDate()
    {
     return d;
    }
%>
Hello! The time is now <%= getDate()%>
</body>
</html>
```

⇒ 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

```
<pretin: action [attribute, = "value" ...] >
          ↓              ↓
       TagName        optional

 //body
</pretin : action>
```

These are standard actions all are have 'jsp' as their pretin. Hence, they have the following Syntax.

```
<isp:tagname [attribute,= "value," .. ]>
  // body
<//isp:tagname>
```

The Isp container supports the following standard action Elements

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

Example:-

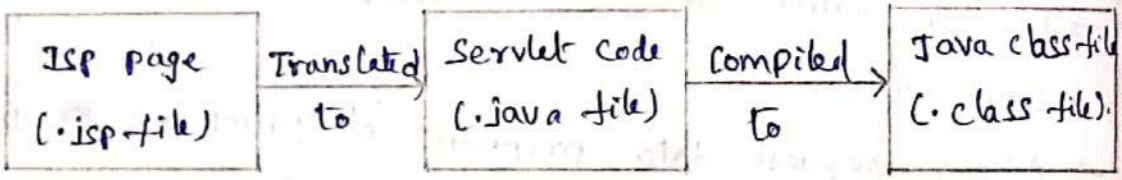
```
<%@page import = "Java.util.*" %>
<html>
<body>
Here including the action
<isp:include page="abc.html" />
Here forwarding the page
<Jsp:forward page="ay2.Jsp" />
</body>
</html>
```

* Jsp processing :-

Any webserver needs a container to run any web component (JSP) to provide interface to run web components.

A Jsp page needs a Jsp container to be processed.

Generally speaking a Jsp container acts as a mediator between the Jsp page and the server by taking all Jsp requests and to produce the response



⇒ Converting Jsp page to a servlet (.java file) and compiling servlet is called "translation phase".

⇒ The Jsp container is also responsible for invoking Jsp page implementation to process each request and generate the response. i.e converting the servlet code to .class file is called as "Request processing phase".

Note :-

All Jsp files converted to Java, that are stored in below path :

"C:\Program Files\Apache Software Foundation\Java\work\Catalina\Localhost\Jsbean\org\apache\jsp"

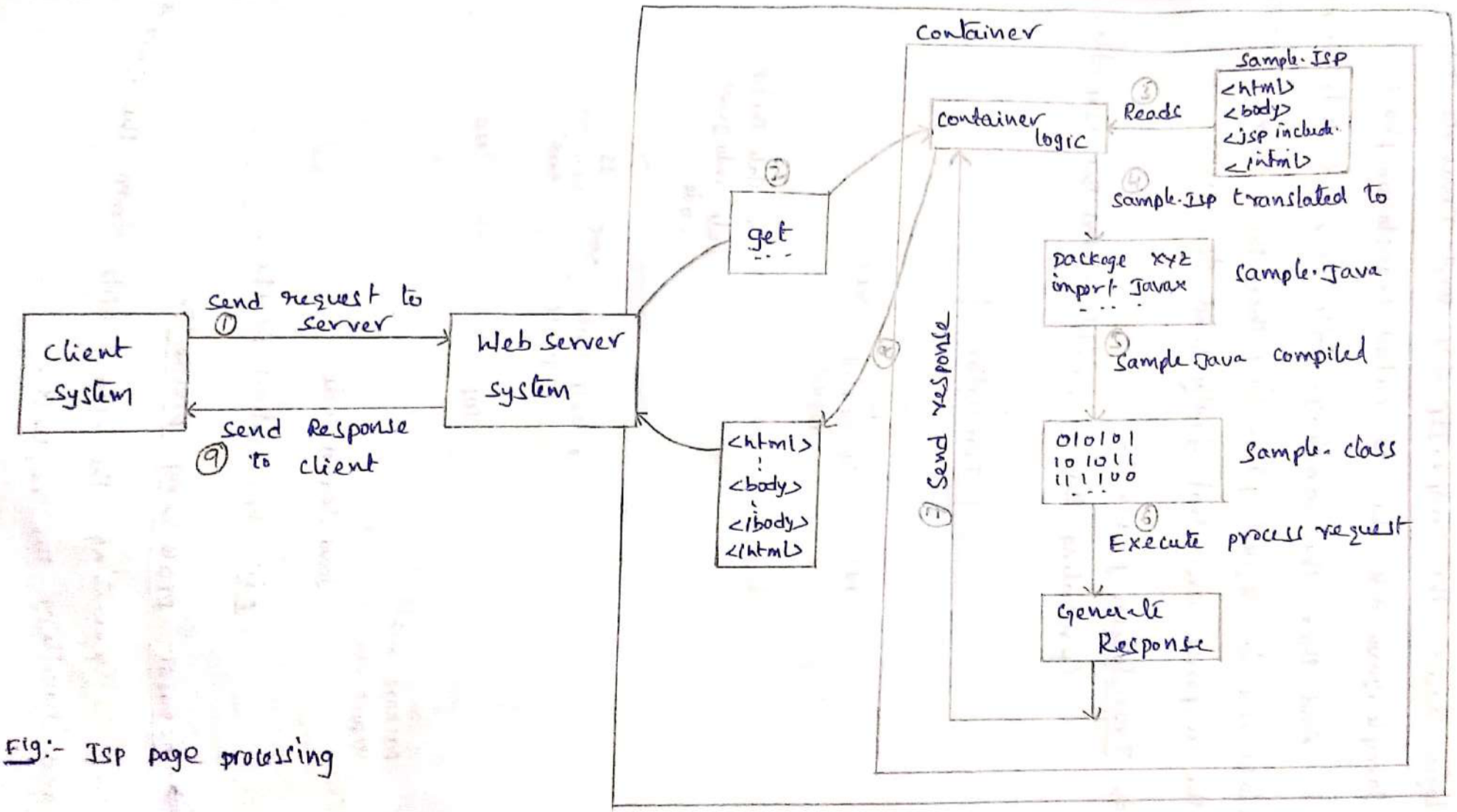


Fig:- Isp page processing

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.

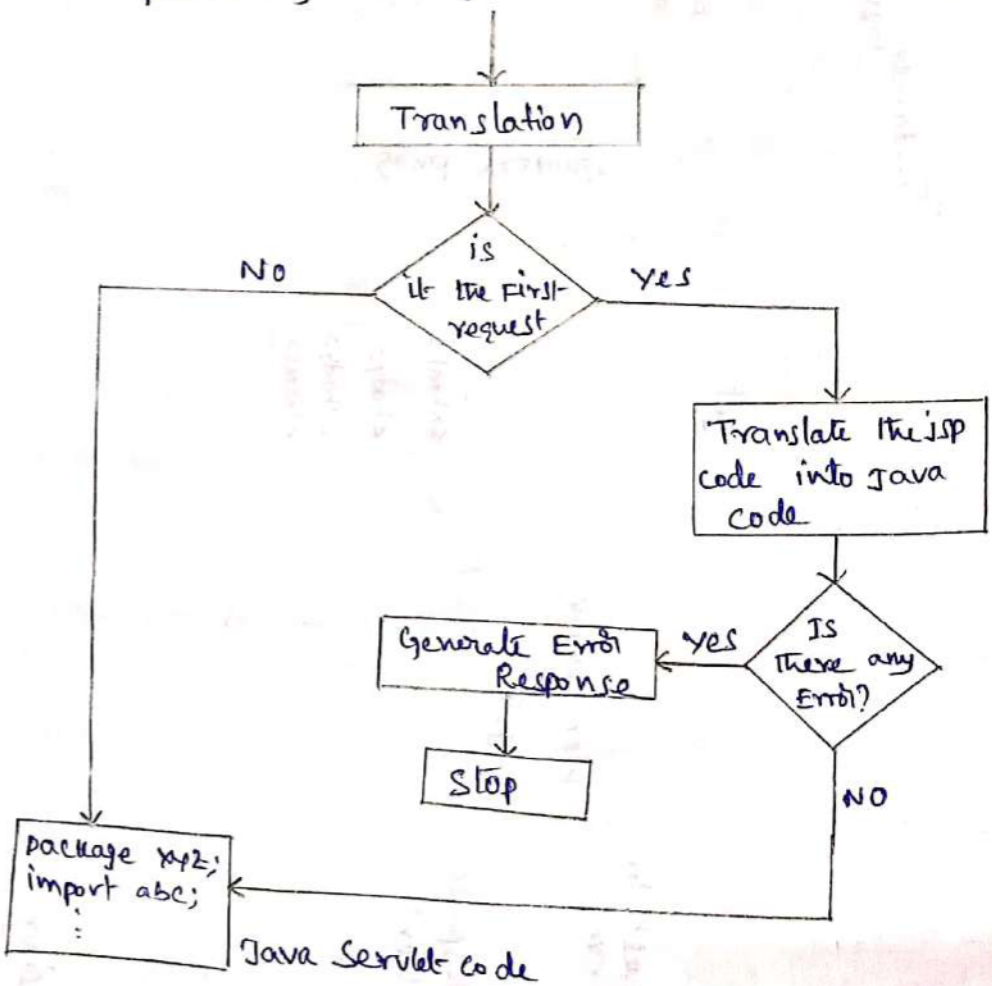


Fig:- Translation phase.

⇒ Request processing phase :-

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

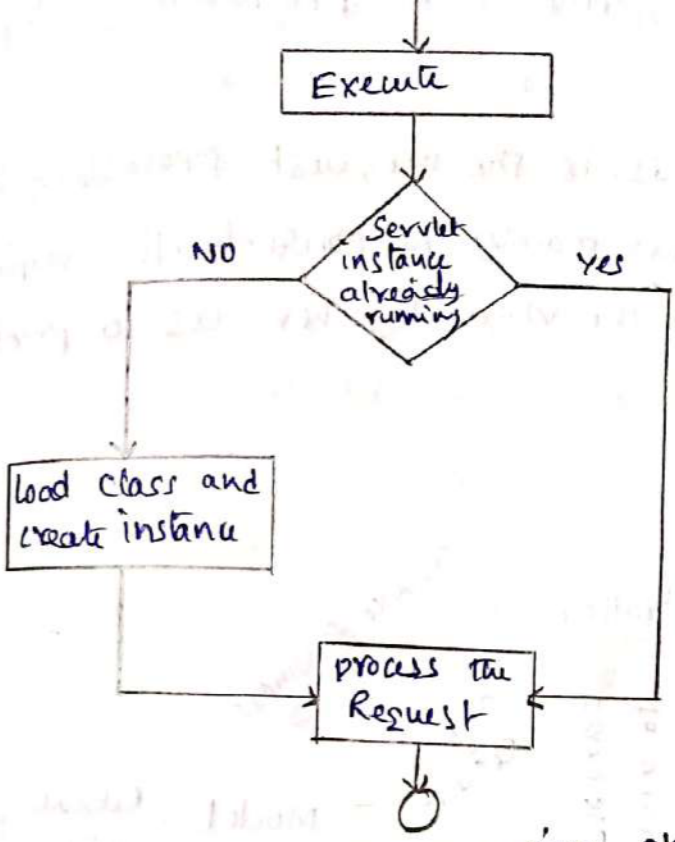


Fig:- Request processing phase.

---

* JSP Application Design with MVC :-

Model-view-controller (MVC) is a design pattern introduced by Xerox PARC (Palo Alto Research Center) in the 1980's.

MVC is a model used in swing components. The main aspect of MVC is to divide the applications into three distinct but interrelated units.

Those are

⇒ Model:- This is the bussiness logic of applications responsible for performing the actual work conducted by the application.

Hence we can say that this unit deals with the modeling of real-world problem and doesn't have any idea about how it is being displayed to the user.

⇒ View:- This is the presenation logic of an application and it represents visual appearance of the information

It may have little & no programming logic at all.

⇒ Controller :- This is the request processing logic of an application, mainly responsible for coupling both model and the view together as to perform some operation.

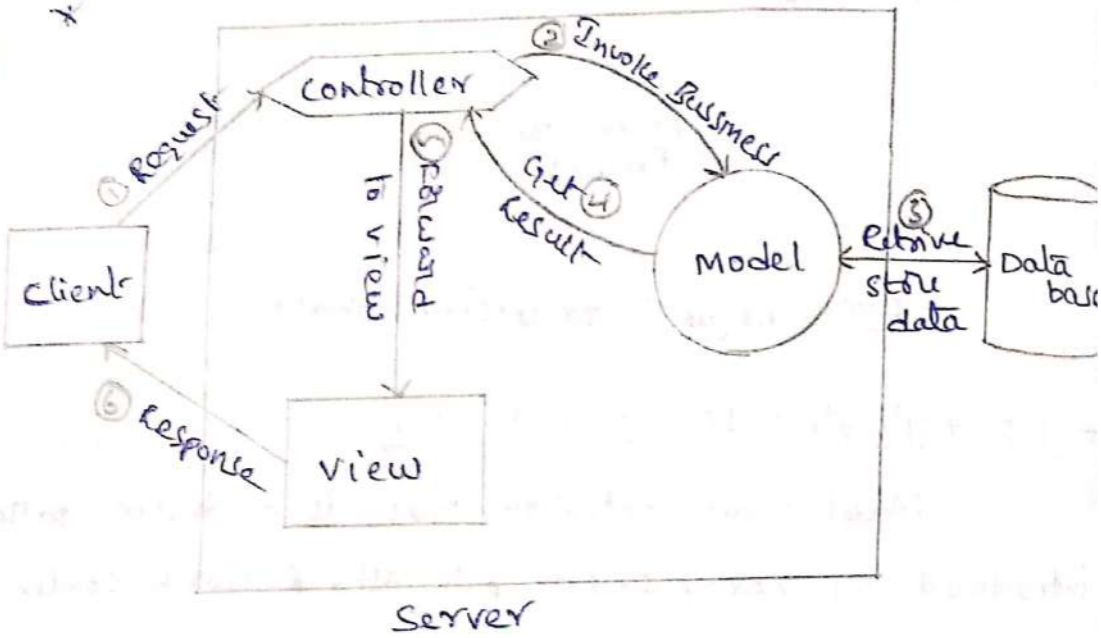


Fig:- MVC Architecture

In the above architecture, the client can sends requ to server, In the server controller receives the reques and invoke the business logic then the model unit can perform actions by storing / retriving the data from database.

And then the controller gets the result and forw to view then view presents the data and then fi send that response to client.

The result of application design with MVC is the separation of presentation logic and bussiness logic.

database libraries to retrive the records from the database.

Generally, this type of drivers are implemented by DBMS vendors, And then are recommended to be used with server-side applications.

Advantages:-

→ This type of drivers pure Java drivers and auto downloadable.

→ This type of driver doesn't require a middleware server.

Disadvantages:-

The main disadvantages of type-4 Driver is that uses database proprietary protocol and is DBMS vendors.

---

* 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.

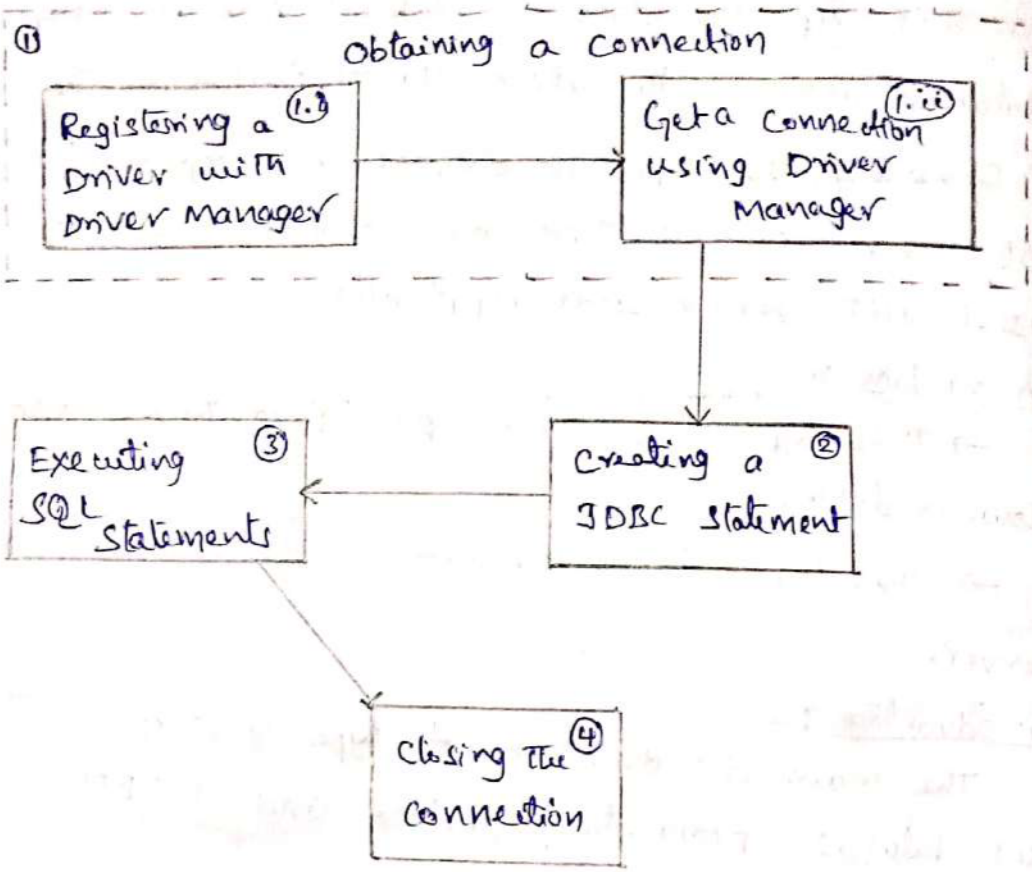
There are some basic steps in JDBC connectivity. these are

Step1 → Obtaining a connection

Step2 → Creating a JDBC statement

Step3 → Executing SQL statements

Step4 → Closing the connection.



↳ **Step1** :- (Obtaining a connection)

In this step, we can obtain a connection with the database server by registering a driver with driver manager and establishing a connection using Driver Manager.

→ Invoke '**Class.forName(<driver classname>)**' method to load & registering a Driver class.

Example:-

Class.forName("oracle.jdbc.driver.oracleDriver")

→ Invoke '**getConnection(String url, String username, String password)**' to create/establish connection with the database, where url is a JDBC url, which represents a unique name used to identify the driver and obtain connection.

↳ **Step 2** :- (Creating a JDBC statement)

After the connection made, we need to create the JDBC statement object to execute the SQL

statements.

To create a statement object, invoke the createStatement method, on the current connection object "connection"

The following code shows how to create a statement object.

```
Statement stmt = connection.createStatement();
```

↳ **Step-3** :- (Executing SQL statements).

After the statement object is created, it can be used to execute the SQL statements by using the executeUpdate() or executeQuery() method.

The executeQuery() method is only used in the SELECT statement.

For other database operations, such as insert, update and delete, the executeUpdate() method is used to execute the statements.

Ex:-

```
// using executeQuery()
String query = "select * from Tablename";
ResultSet results = stmt.executeQuery(query);
// using executeUpdate()
String query = "insert into table_name values (value1, value2";
int count = stmt.executeUpdate(query);
```

↳ **Step-4** :- (closing the connection)

Now, after executing all the required SQL statements and obtaining the results, we need to close the connection and release the session.

This can be done by calling the close() method. The following code shows how to close a connection

```
connection.close();
```

* Accessing a database from a JSP page :-

Now-a-days every website is becoming database-driven so that it can get the data for its user dynamically from a database efficiently.

JSP (Java-server-page)'s are Text-based document typically HTML pages, that contain Java code. The embedded Java code allows the page to contain dynamically generated content.

JSP it makes it possible to integrate content from the variety of data sources, such as databases, files and other respositories.

By using scriptlet, we can include database programming using JDBC (four steps) in JSP.

The Architecture of accessing a database by using JSP.

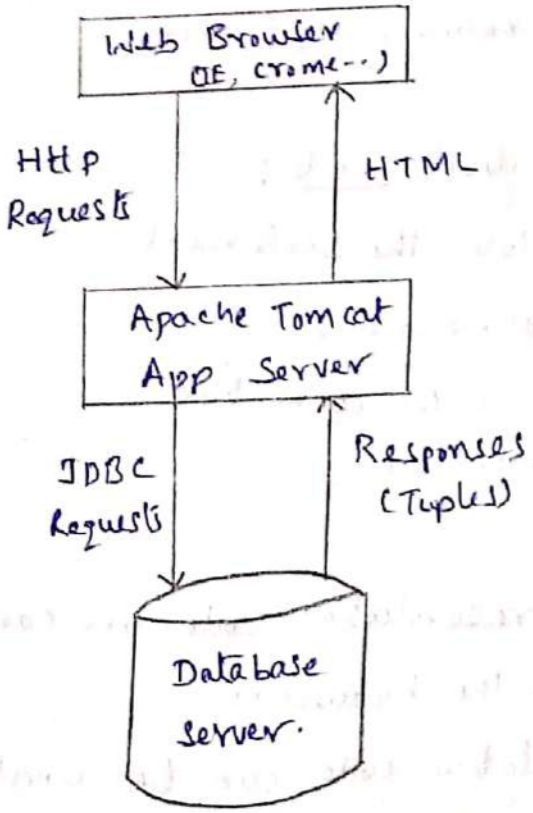


Fig:- 3-tier Architecture.

For this purpose we are writting following code.

* 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.

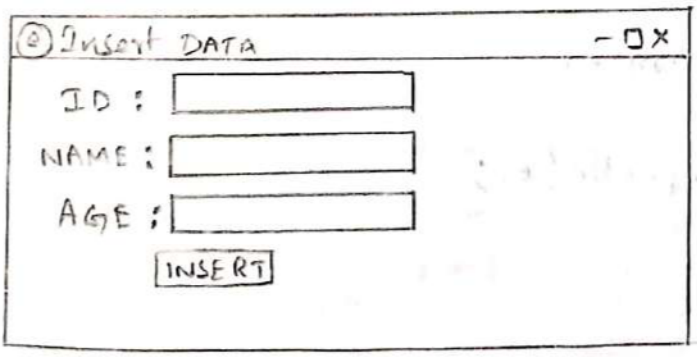
The presentation code can be written by using HTML.

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 :

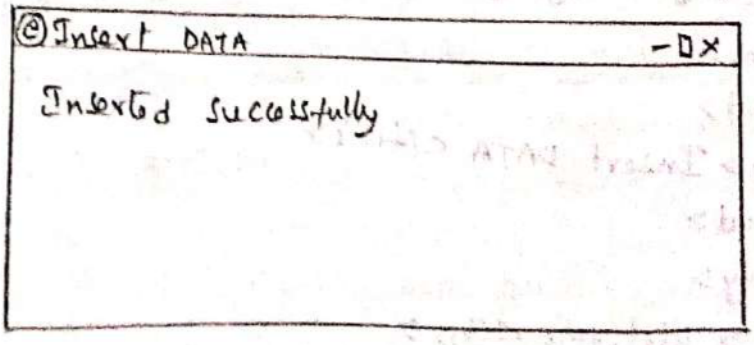
```
<%@ page language = "java" import = "java.sql.*" %>
<html>
<head>
<title> Insert DATA </title>
</head>
<body>
<h2> Welcome </h2>
```

```
<%
   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 bussiness 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 specifing 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 = "SimpleBean" class = "msgbeans.
                                        simpleBean" />

 <jsp:getproperty name = "messageBean"
                                property = "Message"/>

 <jsp:Setproperty name = "messageBean"
            property = "message" value = "Hello" />

 </Body>
 </html>
```

O/P:-