Unit III(Introduction to Servlets)

****

1. Common Gateway Interface (CGI) :

Common Gateway Interface (CGI) is an interface specification that enables the web servers to execute user requests programs.

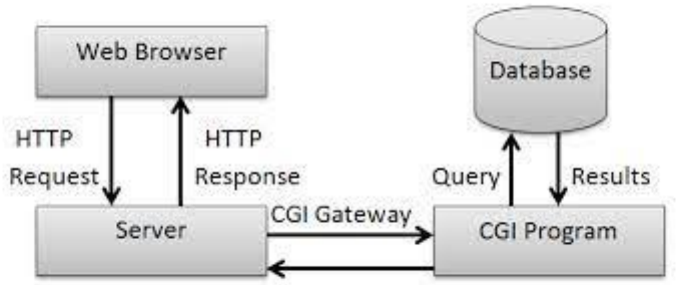
Such programs are often written in a scripting language and are commonly referred to as *CGI scripts*.

When a Web user submits a Web formon a web page that uses CGI. The form's data is sent to the Web server as an HTTP requestwith a URLdenoting a CGI script.

The Web server then processes the CGI script and responses to the browser's request.

i.e., The Web server typically passes the form information to a small application program that processes the data and may send back a confirmation message.

The process passing data back and forth between the server and the application is done by common gateway interface (CGI).



Features of CGI:

- It is a very well defined and supported standard.
- CGI scripts are generally written in either Perl, C, or maybe just a simple shell script.
- CGI is a technology that interfaces with HTML.
- CGI standard is generally the most compatible with today’s browsers

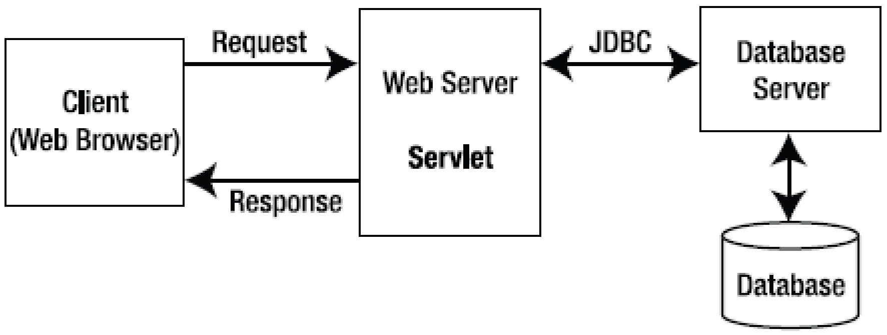
•CGI specifies that the programs can be written in any language, and on any platform, as long as they conform to the specification.

****

2. Life cycle of a Servlet

Servlet :-

- Servlets are server side programs that run on a Web or Application server and act as a middle layer between a requests coming from a Web browser and databases or applications on theserver.
- Using Servlets, you can collect input from users through web page forms or databases.
- Servlets don‘t fork new process for each request, instead a new thread is created.
- The Servlet can handle many requests simultaneously.



Servlet Life Cycle :

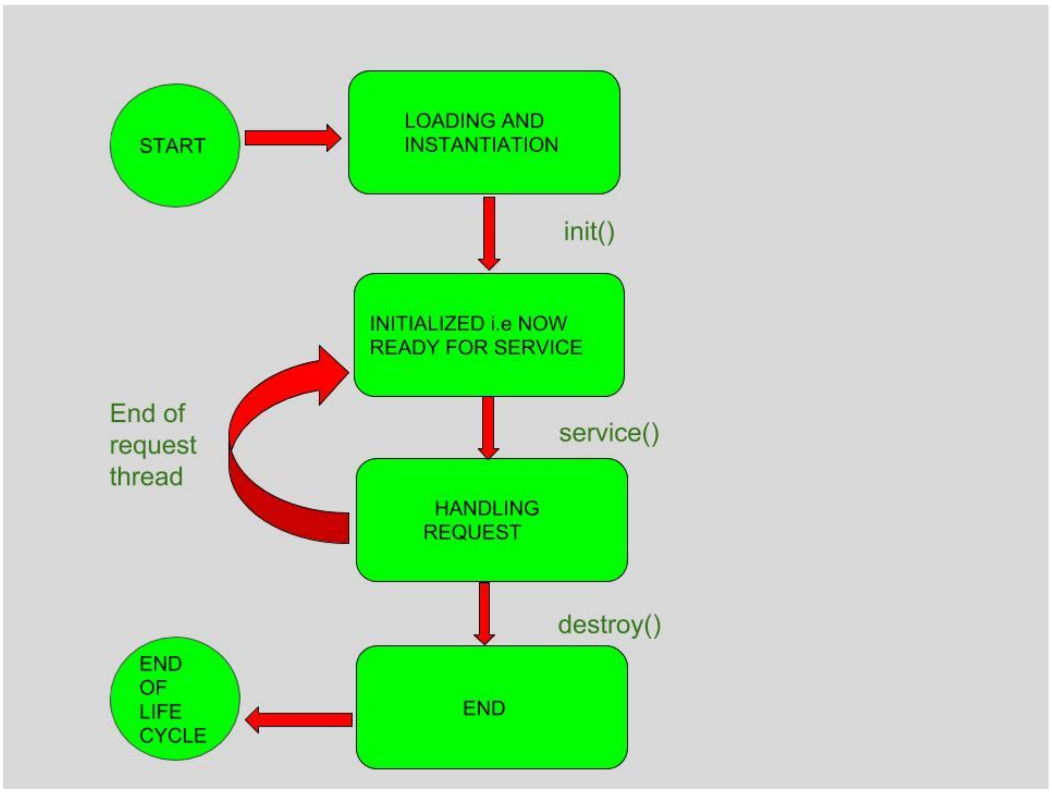
The Servlet life cycle mainly goes through four stages, they are,

•Loading a Servlet.

•Initializing the Servlet.

•Request handling.

•Destroying the Servlet.



1. Loading a Servlet:

The Servlet container performs two operations in this stage :

- Loading : Loads the Servlet class.
- Instantiation : Creates an instance of the Servlet. To create a new instance of the Servlet, the container uses the no-argument constructor.

2. Initializing a Servlet:

After the Servlet is instantiated successfully, the Servlet container initializes the instantiated Servlet object.

The method used is init().

The container initializes the Servlet object by invoking the Servlet.init(ServletConfig) method which accepts ServletConfig object reference as parameter.

The Servlet container invokes the Servlet.init(ServletConfig) method only once, immediately after the Servlet.init(ServletConfig) object is instantiated successfully. This method is used to initialize the resources, such as JDBC datasource.

Now, if the Servlet fails to initialize, then it informs the Servlet container by throwing the ServletException or UnavailableException.

3. Handling request: After initialization, the Servlet instance is ready to serve the client requests.

The method used is service().

The Servlet container performs the following operations when the Servlet instance is located to service a request :

- It creates the ServletRequest and ServletResponse objects. In this case, if this is a HTTP request, then the Web container creates HttpServletRequest and HttpServletResponse objects which are subtypes of the ServletRequest and ServletResponse objects respectively.
- After creating the request and response objects it invokes the Servlet.service(ServletRequest, ServletResponse) method by passing the request and response objects.

The service() method while processing the request may throw the ServletException or UnavailableException or IOException.

The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc.

The doGet() method processes client request, which is sent by the client. To handle client requests, we need to override the
doGet() method in the servletclass.
In the doGet() method,we can retrieve the client information of the HttpServletRequest object and use the HttpServletResponse object to send the response back to the client.

The doPost() method handles requests in a servlet, which is sent by the client.
Unlike the GET method, the POST request sends the data as part of the HTTP request body. As a result, the data sent does not appear as a part ofURL.
To handle requests in a servlet that is sent using the POST method, we need to override the doPost() method. In the doPost() method, we can process the request and send the response back to theclient.

4. Destroying a Servlet: When a Servlet container decides to destroy the Servlet,

The method used is destroy().

it performs the following operations,

- It allows all the threads currently running in the service method of the Servlet instance to complete their jobs and get released.
- After currently running threads have completed their jobs, the Servlet container calls the destroy() method on the Servlet instance.

After the destroy() method is executed, the Servlet container releases all the references of this Servlet instance so that it becomes eligible for garbage collection.

Program :

```
import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;


    public class  Serv extends HttpServlet

    {

    private String output;


            public void init() throws ServletException

            {

                    output = "Servlet Program";

            }


            public void doGet(HttpServletRequest req, HttpServletResponse resp) throws
                                    ServletException, IOException

            {

                    resp.setContentType("text/html");

                    PrintWriter out = resp.getWriter();

                    out.println(output);

            }


            public void destroy()

            {

                    System.out.println("Servlet Over");
```

```
        }
    }
```
****

3. Deploying a Servlet

1. Download and install the Java Software Development kit(JDK).
2. Download and install Webserver(Apache Tomcat).
3. Configure Webserver
   - After installation Tomcat folder will contain ―Start Tomcat and ―Stop Tomcat‖ shortcuts.

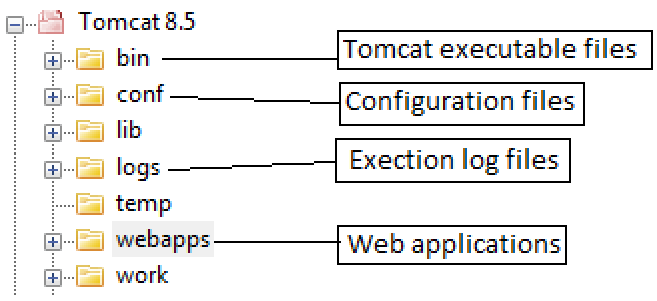
a) Set Environment Variable :
   - The JAVA_HOME environment variable should be set so that Tomcat can find JDK
   JAVA_HOME =c:\jdk1.9
   (Go to My Computer properties -> Click on advanced tab then environment variables -> Click on the new tab of user variable -> Write JAVA_HOME in variable name and paste the path of jdk folder in variable value -> ok -> ok -> ok.)

b) How to change default port number(8080) of apache tomcat

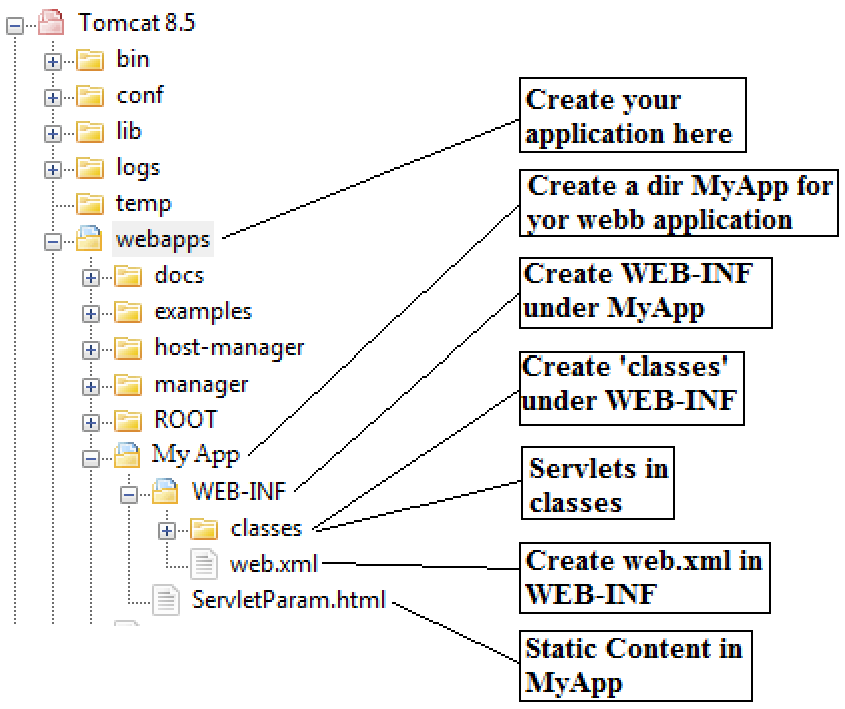
Open **server.xml file** in notepad. It is located inside the **apache- tomcat/conf** directory . Change the Connector port = 8080 and replace 8080 by any four digit number instead of 8080. Let us replace it by 9999 and save this file.

4. Setup deployment environment

Tomcat Diectory



- To set up a new application, add a directory under the “webapps” directory and create a subdirectory called “WEB-INF”.
- WEB-INF needs to contain web.xml (servlet configuration file)
- After WEB-INF directory is created, create a subdirectory classes under it. J ava classes will go under this directory(must Paste the class file here).



5. Creating ServletDemo Servlet
There are three different ways to create a servlet.

a. By extending HttpServlet class
b. By extending GenericServlet class
c. By implementing Servlet interface

Program :

```
import javax.servlet.http.*;

import javax.servlet.*;
import java.io.*;


public class DemoServlet extends HttpServlet
{
public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException
{
res.setContentType("text/html");  //setting the content type
PrintWriter pw=res.getWriter();   //getthe stream to write the data
//writing html in the stream
pw.println("<html><body>");
pw.println("Welcome to servlet");
pw.println("</body></html>");
pw.close();                              //closingthe stream
}
}
```

6. Compile Servlet and save the class file in classes folder.
7. Create a Deployment Descriptor
   - The deployment descriptor is an xml file, from which Web Container gets the information about the servlet to be invoked.
   - The web container uses the Parser to get the information from the web.xml file.
   - Add a <servlet> and a <servlet-mapping> for each servlet for Tomcat to run.
   Add entries after <web-app> tag inside web.xml

```
<web-app>
        <servlet>
                <servlet-name>Demo</servlet-name>
                <servlet-class>DemoServlet</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>Demo</servlet-name>
                <url-pattern>/welcome</url-pattern>
        </servlet-mapping>
</web-app>
```

<web-app>represents the whole application.

<servlet>is sub element of <web-app> and represents the servlet.

<servlet-name>is sub element of <servlet> represents the name of the servlet.

<servlet-class>is sub element of <servlet> represents the class of the servlet.

<servlet-mapping>is sub element of <web-app>. It is used to map the servlet.

<url-pattern>is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet.

8. Start Tomcatserver
To start Apache Tomcat server, double click on the startup.bat file under apache-tomcat/bin directory.
9. Open browser and type   "http://localhost/MyApp/DemoServlet"

****

4. The Servlet API

The two Servlet API are ,
javax.servlet and

javax.servlet.http packages.

The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.

The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

Classes in javax.servlet package

1.GenericServlet

2.ServletInputStream

3.ServletOutputStream

4.ServletRequestWrapper

5.ServletResponseWrapper

6.ServletRequestEvent

7.ServletContextEvent

8.ServletRequestAttributeEvent

9.ServletContextAttributeEvent

10.ServletException

11.UnavailableException

Interfaces in javax.servlet package

1.Servlet

2.ServletRequest

3.ServletResponse

4.RequestDispatcher

5.ServletConfig

6.ServletContext

7.SingleThreadModel

8.Filter

9.FilterConfig

10.FilterChain

11.ServletRequestListener

12.ServletRequestAttributeListener

13.ServletContextListener

14.ServletContextAttributeListener

Classes in javax.servlet.http package

1.HttpServlet

2.Cookie

3.HttpServletRequestWrapper

4.HttpServletResponseWrapper

5.HttpSessionEvent

6.HttpSessionBindingEvent

Interfaces in javax.servlet.http package

1.HttpServletRequest

2.HttpServletResponse

3.HttpSession

4.HttpSessionListener

5.HttpSessionAttributeListener

6.HttpSessionBindingListener

7.HttpSessionActivationListener

Example : HelloForm.java

```
import java.io.*;

import java.util.*;

import javax.servlet.http.*;
```

```
public class HelloForm extends HTTPServlet
{
public void doPost(HttpServletRequest request,HttpServletResponse
response) throws IOException,ServletException
{
PrintWriter pw = response.getWriter();
pw.print("<html><body>");
pw.print("Name: "+request.getParameter("first_name")+  “ ”
+request.getParameter("last_name"));
pw.print("</body></html>");
pw.close();
}
}
```

Execution Steps :

Compile HelloForm.java as follows: $javacHelloForm.java
Compilation would produce HelloForm.classfile.

Next you would have to copy this class file in<Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes

Create following entries in web.xml file locatedin
<Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

```
        <servlet>
                <servlet-name>HelloForm</servlet-name>
                <servlet-class>HelloForm</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>HelloForm</servlet-name>
                <url-pattern>/HelloForm</url-pattern>
        </servlet-mapping>
```

Now create a HTML page Hello.htmland put it in
<Tomcat-installation-directory>/webapps/ROOT directory

```
<html>
<body>
<form action="HelloForm"method="GET">
First Name: <inputtype="text"name="first_name"><br/>
Last Name: <inputtype="text"name="last_name"/></br>
<inputtype="submit"value="Submit"/>
</form>
</body>
</html>
```

Start Tomcat Server and openbrowser and type “http://localhost:8080/Hello.html”

Now enter firstname and lastname, ClickSubmit□
****
****

5. Reading Servlet parameters

The ServletRequest interface includes the methods that allow you to read the names and values of parameters that are included in a client request.

Eg:- contains two files.

(A web page is defined in PostParameters.html, and a servlet is defined in PostParametersServlet.java.)

The HTML source code for **PostParameters.html** is shown in the following listing. It defines a table that contains two labels and two text fields. One of the labels is Employee and the other is Phone. There is also a submit button. Notice that the action parameter of the form tag specifies a URL.

The URL identifies the servlet to process the HTTP POST request.

1. PostParameters.html

```
<html>
<body>
<center>
<form name="Form1" method="post"
action="http://localhost:8080/examples/servlets/
servlet/PostParametersServlet">
<table>
<tr>
<td><B>Employee</td>
<td><input type=textbox name="e" size="25" value=""></td></tr>
<tr>
<td><B>Phone</td>
<td><input type=textbox name="p" size="25" value=""></td></tr>
</table>
<input type=submit value="Submit"></body>
</html>
```

In PostParametersServlet.java , the service( ) method is overridden to process client requests.

The getParameterNames( ) method returns an enumeration of the parameter names which are processed in a loop.

The parameter value is obtained via the getParameter( ) method.

2. PostParametersServlet.java

```
import java.io.*;

import java.util.*;

import javax.servlet.*;

public class PostParametersServlet extends GenericServlet

{

public void service(ServletRequest request, ServletResponse

response)throws ServletException, IOException

{

        PrintWriter pw = response.getWriter();

Enumeration e = request.getParameterNames();

//Display parameter names and values.
while(e.hasMoreElements())
{
String pname = (String)e.nextElement();

pw.print(pname + " = ");

String pvalue = request.getParameter(pname);

pw.println(pvalue);

}

pw.close();

}
```

Compile the servlet. Next, copy it to the appropriate directory, and update the **web.xml** file, as previously described. Then, perform these steps to test this example:

•Start Tomcat (if it is not already running).
•Display the web page in a browser.
•Enter an employee name and phone number in the text fields and click Submit.

****

6. Reading Initialization parameters

Initialization parameters are stored as key value pairs. They are included in *web.xml*file inside *init-param*tags. The key is specified using the *param-name*tags and value is specified using the *param-value*tags.

Eg:-

```
<servlet>
        <servlet-name>Name of servlet</servlet-name>
        <servlet-class>Servlet class</servlet-class>
        <init-param>
                <param-name>Mail</param-name>
                <param-value>admin@company.com</param-value>
        </init-param>
</servlet>
```

Initialization parameters are stored as key value pairs. They are included in web.xml file inside init-param tags. The key is specified using the param-name tags and value is specified using the param-value tags.

Servlet initialization parameters are retrieved by using the ServletConfig object. Following methods in ServletConfig interface can be used to retrieve the initialization parameters:

- String getInitParameter(String parameter_name)
- Enumeration getInitParameterNames()

Eg:-

1. login.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Login</title>
</head>
<body>
        <form action="ValidServ" method="post">
```

```
                Username: <input type="text" name="txtuser" /><br/>
                Password: <input type="password" name="txtpass" /><br/>
                <input type="submit" value="Submit" />
                <input type="reset" value="clear" />
        </form>
</body>
</html>
```

2. web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
<servlet>
<servlet-name>ValidServ</servlet-name>
<servlet-class>ValidServ</servlet-class>

<init-param>
<param-name>user1</param-name>
<param-value>pass1</param-value>
</init-param>

<init-param>
<param-name>user2</param-name>
<param-value>pass2</param-value>
</init-param>

<init-param>
<param-name>user3</param-name>
<param-value>pass3</param-value>
</init-param>

<init-param>
<param-name>user4</param-name>
<param-value>pass4</param-value>
</init-param>
</servlet>

<servlet-mapping>
<servlet-name>ValidServ</servlet-name>
<url-pattern>/ValidServ</url-pattern>
</servlet-mapping>
</web-app>
```

3. ValidServ.java (Servlet file)

```
import java.io.IOException;
import java.util.Enumeration;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/* Servlet implementation class ValidServ  */
public class ValidServ extends HttpServlet
{
        private static final long serialVersionUID = 1L;

        ServletConfig cfg;

/* @see HttpServlet#HttpServlet()     */

public ValidServ()
{
super();
// TODO Auto-generated constructor stub
}

        /*@see Servlet#init(ServletConfig)  */
public void init(ServletConfig config) throws ServletException
{
        cfg = config;
}

public void doPost(HttpServletRequest request, HttpServletResponse
response)     throws ServletException, IOException
{
        String un = request.getParameter("txtuser");
        String pw = request.getParameter("txtpass");
        boolean flag = false;
        Enumeration<String> initparams = cfg.getInitParameterNames();
        while(initparams.hasMoreElements())
                {
                        String name = initparams.nextElement();
                        String pass = cfg.getInitParameter(name);
                        if(un.equals(name) && pw.equals(pass))
                        {
                                flag = true;
                }
            }
            if(flag)
            {
                response.getWriter().print("Valid user!");
            }
            else
            {
                response.getWriter().print("Invalid user!");
            }
        }
}
```

In the above example four pairs of initialization parameters are stored in *web.xml*file. In the servlet file we are validating the username and password entered in the login page against the initialization parameters by retrieving them from *web.xml*file.

7. Handling Http Request & Responses

HTTP Requests :

The message that a client sends is known as an HTTP request.

An HTTP request is an action performed on a resource identified by an URL.

These requests are sent with the help of various methods known as HTTP request methods.

These methods indicate specific action that has to be performed on a given resource.

There are some standard features shared by the various HTTP request methods.

an HTTP request is sent by the Client and is submitted to the server. The request is processed and then the server sends a response which contains the status information of the request.

It is clear that HTTP requests work as an intermediary between a client or an application and a server.

Various types of HTTP Request Methods are,

- GET Request: To retrieve and request data
- POST Request: It is used to send data to a server in order to create or update a resource.
- PUT Request :To update data
- DELETE Request: For deleting data
- HEAD method : It is commonly used for testing hypertext links for accessibility, validity or recent modification.

Eg:-

```
<html>
<body>
<center>
<form name="Form1" method="post"
action="http://localhost:8080/examples/servlets/servlet/ColorPostServlet">
<B>Color:</B>
```

```
<select name="color" size="1"><option value="Red">Red</option><option
value="Green">Green</option><option value="Blue">Blue</option></select>
<br><br>
<input type=submit value="Submit"></form>
</body>
</html>
```

HTTP Response :

HTTP Response is where the information is sent by the server to the client in response to the request made by the client.

HTTP Response basically has the information requested by the Client.

A response status line consists of HTTP protocol version, status code and a reason phrase.

Eg:-

```
import   java.io.*;
import   javax.servlet.*;
import javax.servlet.http.*;
public class ColorPostServlet extends HttpServlet
{
public void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
String color = request.getParameter("color");
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
pw.println("<B>The selected color is: ");
pw.println(color);
pw.close();
}
}
```

8. Using Cookies and Sessions

Cookies :
Cookies are text files stored on the client computer and they are kept for various information trackingpurpose.

There are three steps involved in identifying users:

- Server script sends a set of cookies to the browser in response header.
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server in request header and server uses that information to identify the user.

Cookies are created using Cookie class present in ServletAPI.
For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces,They are,

a. getCookies(): method of HttpServletRequest interface is used to return all the cookies from the browser.

b. addCookie(): method of HttpServletResponse interface is used to add cookie in response object.

c. getName() : Returns the name of the cookie. The name cannot
be changed after creation.

d. getValue() : Returns the value of the cookie.

e. setName(String name) : changes the name of the cookie.

f. setValue() : changes the value of the cookie.

g. setMaxAge() : Sets the maximum age of the cookie in seconds.

Disadvantage of Cookies
• It will not work if cookie is disabled from thebrowser.
• Only textual information can be set in Cookieobject.

Ex: List and AddCookie.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ListandAddCookie extends HttpServlet
```

```
{
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
response.setContentType("text/html");
PrintWriter out = response.getWriter();
Cookie cookie = null;
out.println("<html><body>"+
"<form method='get' action='/Aditya/CookieLab'>"+
"Name:<input type='text' name='user' /><br/>"+
"Password:<input type='text' name='pass'><br/>"+
"<input type='submit' value='submit'>"+
"</form>");

String name = request.getParameter("user");
String pass = request.getParameter("pass");
if(!pass.equals("")||!name.equals(""))
{
Cookie ck = newCookie(name,pass);
response.addCookie(ck);
}

Cookie[ ] cookies = request.getCookies();

if( cookies != null )
{
out.println("<h2> Found Cookies Name and Value</h2>");
for (inti = 0; i<cookies.length; i++)
{
cookie = cookies[i];
out.print("Cookie Name : " + cookie.getName() + ", ");
out.print("Cookie Value: " + cookie.getValue()+"<br/>");
}
}
out.println("</body></html>");
}
}
```

web.xml :

```
<web-app>
<servlet>
<servlet-name>ListandAddCookie</servlet-name>
<servlet-class>ListandAddCookie</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>ListandAddCookie</servlet-name>
<url-pattern>/ListandAddCookie</url-pattern>
</servlet-mapping>
</web-app>
```

Session Tracking

Session simply means a particular interval of time.
Session Tracking is a way to maintain state (data) of an user.
Http protocol is a stateless, each request is considered as the new request, so we need to maintain state using session tracking and recognize particular user.
The servlet container uses this interface to create a session between an HTTP client and an HTTPserver.
The session persists for a specified time period, across more than one connection or page request from the user.

Methods :

1. getId(): Returns a string containing the unique identifiervalue.
2. getCreationTime(): Returns the time when this session was created.
3. getLastAccessedTime():Returns the last time the client sent a request associated with this session, as the number of milliseconds.
4. invalidate(): Invalidates this session then unbinds any objects bound to it.

Ex: SessionTrack.java

```
import java.io.*;
importjavax.servlet.*;
importjavax.servlet.http.*;

public class SessionTrack extends HttpServlet
{
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{

// Create a session object if it is already notcreated.
HttpSession session = request.getSession(true);
String title = "Welcome to my website";
String userID = "";
Integer visitCount = new Integer(0);
if (session.isNew())
{
userID = "Deepu";
session.setAttribute("UserId", "Deepu");
}
else
{
visitCount = (Integer)session.getAttribute("visitCount");
visitCount = visitCount + 1;
userID = (String)session.getAttribute("UserId");
}

session.setAttribute("visitCount",visitCount);
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<html>" + "<body>" +"<h1>Session Infomation</h1>" +
"<table border='1'>" +
                "<tr><th>Session info</th><th>value</th></tr>" +
                "<tr><td>id</td><td>" + session.getId() + "</td></tr>" +
                "<tr><td>User ID</td<td>" + userID + —</td></tr>" +
                "<tr><td>Number of visits</td><td>" + visitCount + "</td></tr>" +
                "</table></body></html>");
}
}
```

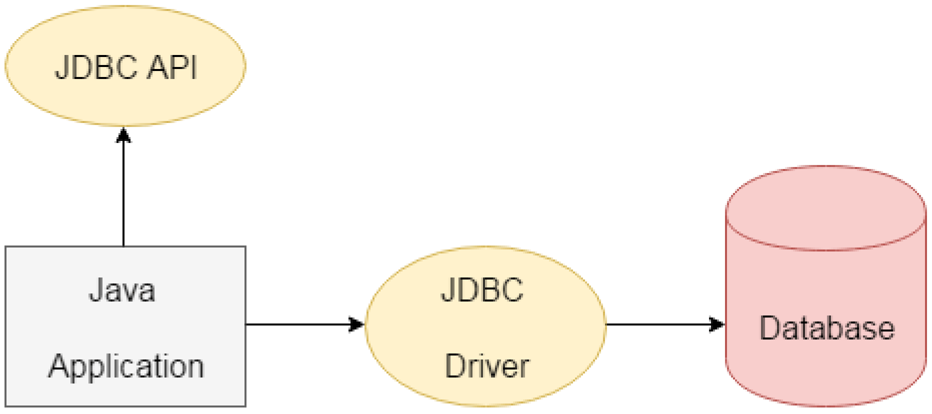
web.xml

```
<web-app>
<servlet>
        <servlet-name>SessionTrack</servlet-name>
        <servlet-class>SessionTrack</servlet-class>
</servlet>
<servlet-mapping>
        <servlet-name>SessionTrack</servlet-name>
        <url-pattern>/SessionTrack</url-pattern>
</servlet-mapping>
</web-app>
```

9. Connecting to a database using JDBC

JDBC stands for Java Database Connectivity.

JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition).



JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

1. JDBC-ODBC Bridge Driver,
2. Native Driver,
3. Network Protocol Driver, and
4. Thin Driver

1) JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

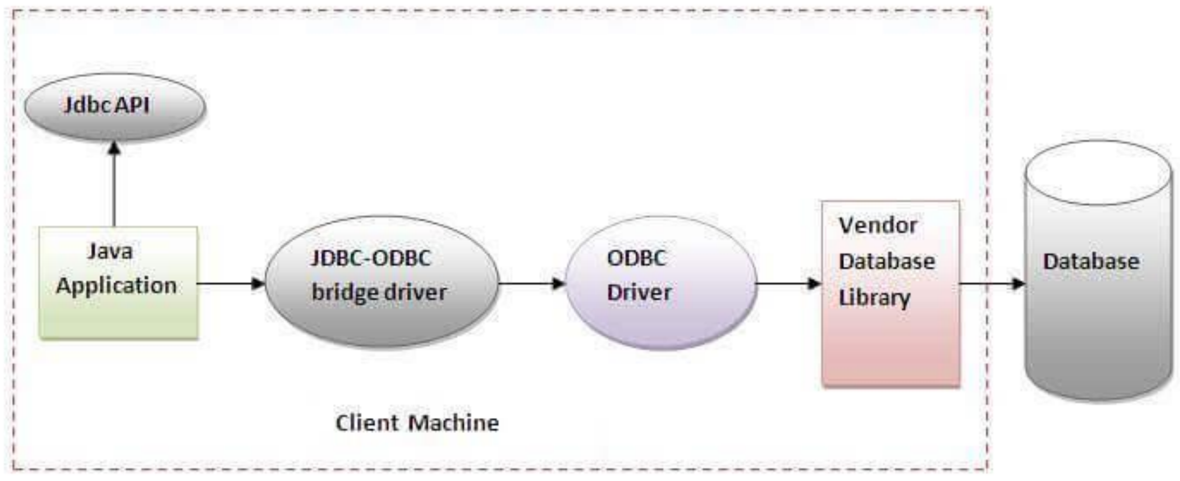


Figure- JDBC-ODBC Bridge Driver

2) Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

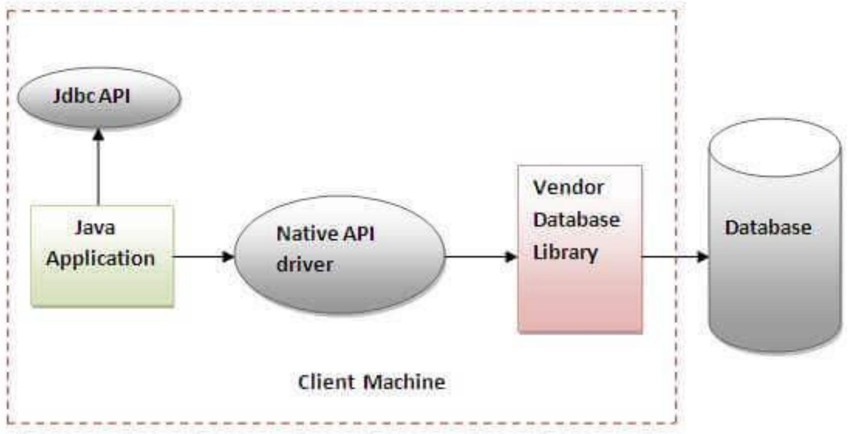


Figure- Native API Driver

3) Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

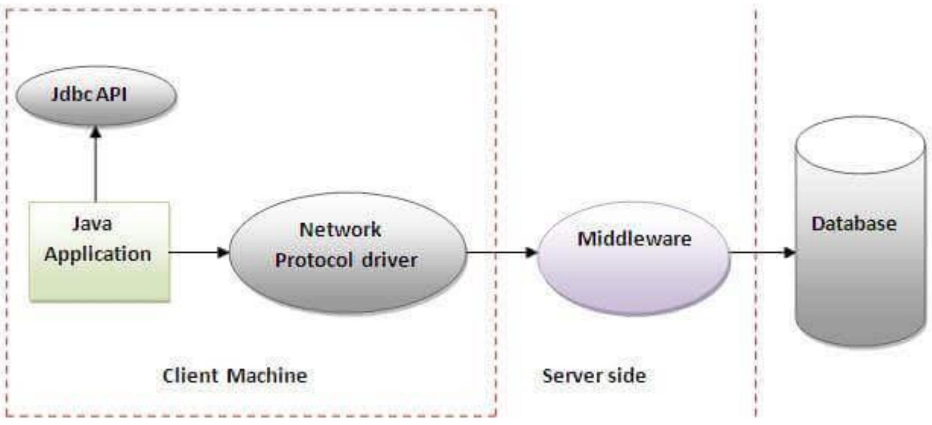


Figure- Network Protocol Driver

4) Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.

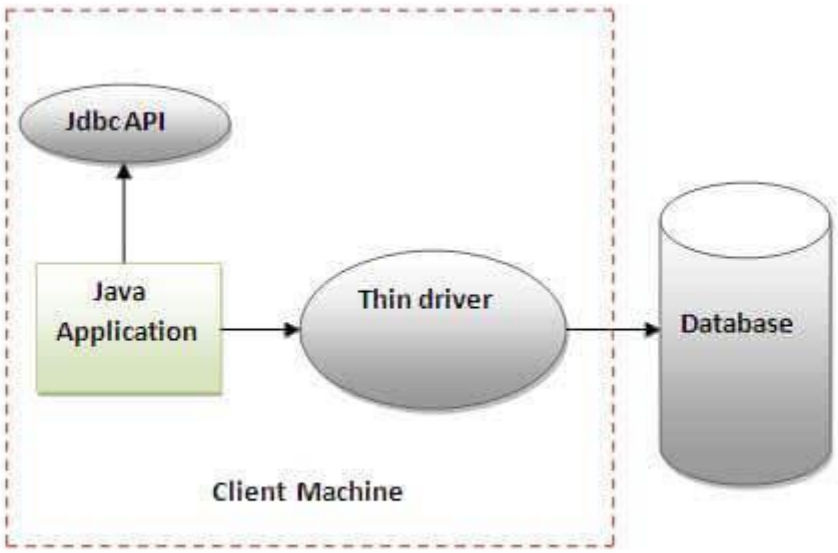


Figure- Thin Driver

Connectivity :

The java.sqlpackage contains classes and interfaces for JDBC API.

The various interfaces of JDBC API are,

- Driver interface
- Connection interface
- Statement interface
- PreparedStatement interface
- CallableStatement interface
- ResultSet interface
- ResultSetMetaData interface
- DatabaseMetaData interface
- RowSet interface

There are 5 steps to connect any java application with the database using JDBC. They are

- Register the Driver class
- Create connection
- Create statement
- Execute queries
- Close connection

1) Register the driver class

The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Eg:-

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

2) Create the connection object

The **getConnection()** method of DriverManager class is used to establish connection with the database.

Eg:-

```
Connection con = DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:xe", "system",
"password");
```

3) Create the Statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Eg:-

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

4) Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Eg:-

```
ResultSet  rs=stmt.executeQuery("select * from emp");


while(rs.next())
{
System.out.println(rs.getInt(1)+""+rs.getString(2));
}
```

5) Close the connection object

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Eg:-

```
con.close();
```

Connect Java Application with Oracle database

```
import java.sql.*;
class OracleCon{
public static void main(String args[])
{
try
{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection
("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+""+rs.getString(2)+""+rs.getString(3));
con.close();
}
catch(Exception e)
{
System.out.println(e);
}

}
}
```

Insert Records to a Table using JDBC Connection

```
import java.io.*;

import java.sql.*;

public class Database {

        static final String url = "jdbc:mysql://localhost:3306/db";

        public static void main(String[] args) throws ClassNotFoundException

        {

                try {

                        Class.forName("com.mysql.jdbc.Driver");

                        Connection conn = DriverManager.getConnection(

                                                url, "root", "1234");

                        Statement stmt = conn.createStatement();

                        int result = stmt.executeUpdate(

                                "insert into student(Id,name,number)
values('1','Pradeep','43')");

                        if (result > 0)

                                System.out.println("successfully inserted");

                        else

                                System.out.println("unsucessful insertion ");

                        conn.close();

                }


                catch (SQLException e) {

                        System.out.println(e);

                }

        }

}
```

}

Java Program to Retrieve Contents of a Table Using JDBC connection

```
import java.sql.*;
public class GFG {
        public static void main(String[] args)
        {
                Connection con = null;
                PreparedStatement p = null;
                ResultSet rs = null;
                con = connection.connectDB();

                try {
                        String sql = "select * from student";
                        p = con.prepareStatement(sql);
                        rs = p.executeQuery();
                        System.out.println("id \t\t name \t\t number");
                        while (rs.next()) {
                                int id = rs.getInt("id");
                                String name = rs.getString("name");
                                String email = rs.getString("");
                                System.out.println(id + "\t\t" + name      + "\t\t" + number);
                        }
                }
                catch (SQLException e)
{
                        System.out.println(e);
                }
        }
}
```