UNIT-III

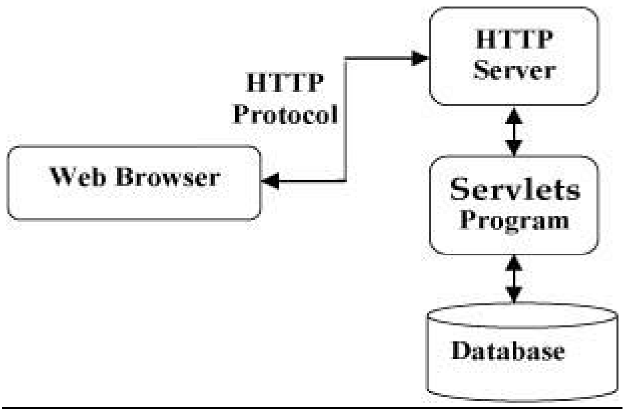
Introduction to Servlets: Common Gateway Interface (CGI), Lifecycle of a Servlet, deploying a servlet, The Servlet API, Reading Servlet parameters, Reading Initialization parameters, Handling Http Request & Responses, Using Cookies and Sessions, connecting to a database using JDBC.

What are Servlets:

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server. Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically. Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI. Performance is significantly better. Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request. Servlets are platform-independent because they are written in Java. Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted. The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

Servlets Architecture

The following diagram shows the position of Servlets in a Web Application



Servlets – Life Cycle

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet

 The servlet is initialized by calling the **init ()** method.

 The servlet calls **service()** method to process a client's request.

 The servlet is terminated by calling the **destroy()** method.

 Finally, servlet is garbage collected by the garbage collector of the JVM.

Now let us discuss the life cycle methods in detail.

The init() Method

The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations, just as with the init method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this:

```
public void init() throws ServletException {
// Initialization code...
}
```

The service() Method

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method:

```
public void service(ServletRequest request,
ServletResponse response)
throws ServletException, IOException{
}
```

The service () method is called by the container and service method invokes doGe, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.

The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
// Servlet code
}
```

The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
// Servlet code
}
```

The destroy() Method

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this:

```
public void destroy() {
// Finalization code...
}
```

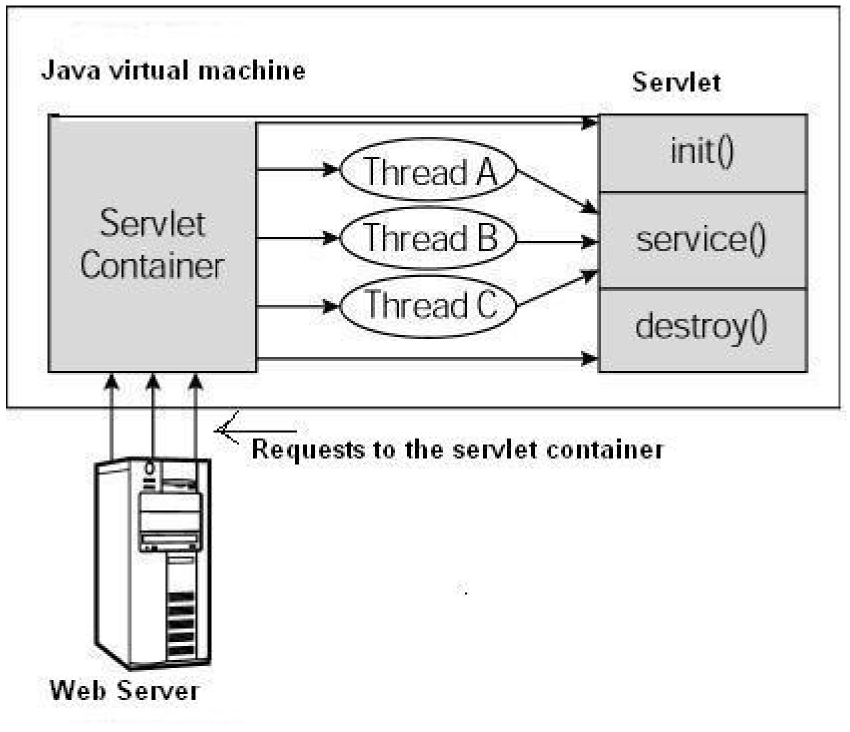
Architecture Diagram

The following figure depicts a typical servlet life-cycle scenario.

 First the HTTP requests coming to the server are delegated to the servlet container.

 The servlet container loads the servlet before invoking the service() method.

 Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet.



SERVLET DEPLOYEMENT:

Steps to create a servlet example

1.Steps to create the servlet using Tomcat server

1. Create a directory structure

2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the application

There are given 6 steps to create a **servlet example**. These steps are required for all the servers.

The servlet example can be created by three ways:

1. By implementing Servlet interface,
2. By inheriting GenericServlet class, (or)
3. By inheriting HttpServlet class

The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.

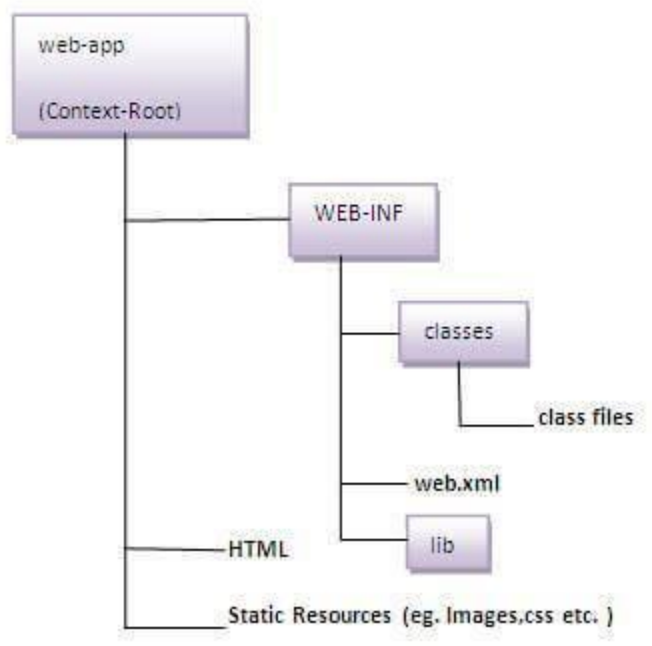
Here, we are going to use **apache tomcat server** in this example. The steps are as follows:

1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the project
6. Access the servlet

1)Create a directory structures

The **directory structure** defines that where to put the different types of files so that web container may get the information and respond to the client.

The Sun Microsystem defines a unique standard to be followed by all the server vendors. Let's see the directory structure that must be followed to create the servlet.



As you can see that the servlet class file must be in the classes folder. The web.xml file must be under the WEB-INF folder.

2)Create a Servlet

There are three ways to create the servlet.

1. By implementing the Servlet interface
2. By inheriting the GenericServlet class

3. By inheriting the HttpServlet class

The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests Such as doGet(), doPost, doHead() etc.

In this example we are going to create a servlet that extends the HttpServlet class.

In this example, we are inheriting the HttpServlet class and providing the implementation of the doGet() method.

Notice that get request is the default request.

DemoServlet.java

```
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();//get the 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();//closing the stream
}
}
```

3)Compile the servlet

For compiling the Servlet, jar file is required to be loaded. Different Servers provide different jar files:

Jar file	Server
1) servlet-api.jar	Apache Tomcat
2) weblogic.jar	Weblogic
3) javaee.jar	Glassfish
4) javaee.jar	JBoss

Two ways to load the jar file

1. set classpath
2. paste the jar file in JRE/lib/ext folder

Put the java file in any folder. After compiling the java file, paste the class file of servlet in **WEB-INF/classes** directory.

4)Create the deployment descriptor (web.xml file)

The **deployment descriptor** is an xml file, from which Web Container gets the information about the servet to be invoked.

The web container uses the Parser to get the information from the web.xml file. There are many xml parsers such as SAX, DOM and Pull.

There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

web.xml file

```
<web-app>
<servlet>
<servlet-name>sonoojaiswal</servlet-name>
<servlet-class>DemoServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

</web-app>
```

Description of the elements of web.xml file

There are too many elements in the web.xml file. Here is the illustration of some elements that is used in the above web.xml file. The elements are as follows:

<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.

5)Start the Server and deploy the project

To start Apache Tomcat server, double click on the startup.bat file under apache-tomcat/bin directory.

One Time Configuration for Apache Tomcat Server

You need to perform 2 tasks:

1. set JAVA_HOME or JRE_HOME in environment variable (It is required to start server).
2. Change the port number of tomcat (optional). It is required if another server is running on same port (8080).

6) How to access the servlet

Open broser and write http://hostname:portno/contextroot/urlpatternofservlet. For example:

http://localhost:9999/demo/welcome

Servlet API

Servlet API consists of two important packages that encapsulates all the important classes and interface, namely :

- **javax.servlet**
- **javax.servlet.http**
- 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.
- Let's see what are the interfaces of javax.servlet package

Servlet Interface

Servlet interface provides common behaviour to all the servlets.

Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

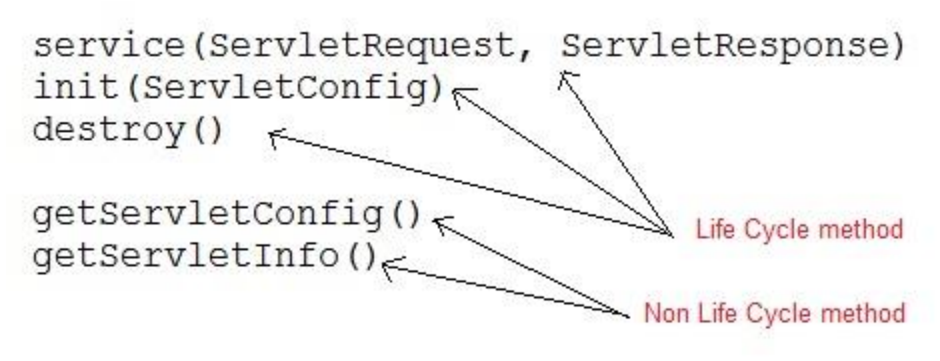
Methods of Servlet interface

There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.

Method	**Description**

public void init(ServletConfig config)	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
public void service(ServletRequest request,ServletResponse response)	provides response for the incoming request. It is invoked at each request by the web container.
public void destroy()	is invoked only once and indicates that servlet is being destroyed.
public ServletConfig getServletConfig()	returns the object of ServletConfig.
public String getServletInfo()	returns information about servlet such as writer, copyright, version etc.

Servlet Interface provides five methods. Out of these five methods, three methods are **Servlet life cycle** methods and rest two are non life cycle methods.



GenericServlet Class

GenericServlet is an abstract class that provides implementation of most of the basic servlet methods. This is a very important class.

Methods of GenericServlet class

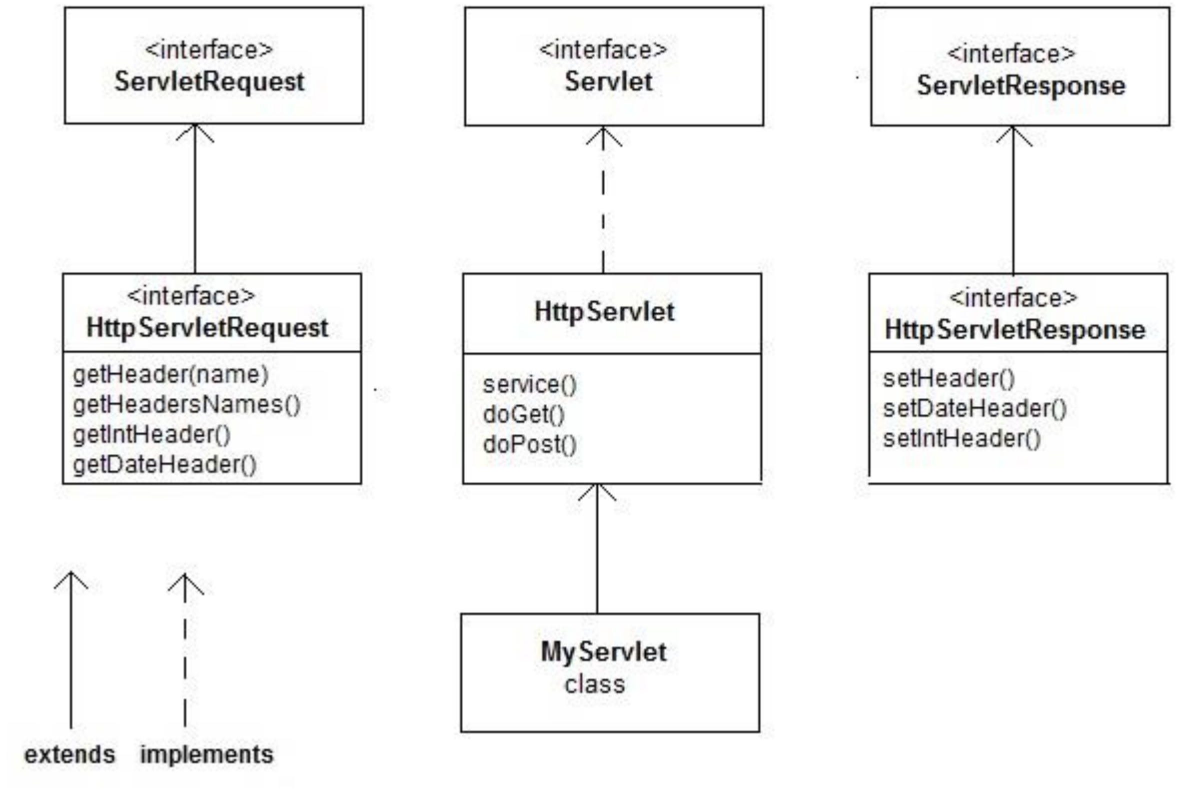
- public void init(ServletConfig)
- public abstract void service(ServletRequest request,ServletResposne response)
- public void destroy()
- public ServletConfig getServletConfig()
- public String getServletInfo()
- public ServletContext getServletContext()
- public String getInitParameter(String name)
- public Enumeration getInitParameterNames()
- public String getServletName()
- public void log(String msg)
- public void log(String msg, Throwable t)

HttpServlet class

HttpServlet is also an abstract class. This class gives implementation of various service() methods of **Servlet** interface.

To create a servlet, we should create a class that extends **HttpServlet** abstract class. The Servlet class that we will create, must not override service() method. Our servlet class will override only the doGet() and/or doPost() methods.

The service() method of **HttpServlet** class listens to the Http methods (GET, POST etc) from request stream and invokes doGet() or doPost() methods based on Http Method type.



Reading initialization parameters

Most of the time, data (Ex: admin email, database username and password, user roles etc…) need to be provided in the production mode (client choice). Initialization parameters can reduce the complexity and maintenance. Initialization parameters are specified in the web.xml file as follows:

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()

Following example demonstrates initialization parameters:

login.html

```
<!DOCTYPE html>
<html>
<head>
<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>
```

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>
```

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;
public class ValidServ extends HttpServlet {
private static final long serialVersionUID = 1L;
ServletConfig cfg;
public ValidServ() {
super();
// TODO Auto-generated constructor stub
}
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.

Reading Servlet Parameters :

The ServletRequest class includes methods that allow you to read the names and values of parameters that are included in a client request. We will develop a servlet that illustrates their use.

The example contains two files.

A Web page is defined in **sum.html** and a servlet is defined in **Add.java**

sum.html:

```
<html>

<body>

<center>

<form name="Form1" method="post"

action="Add">

<table>

<tr>

<td><B>Enter First Number</td>

<td><input type=textbox name="Enter First Number" size="25" value=""></td>

</tr>

<tr>

<td><B>Enter Second Number</td>

<td><input type=textbox name="Enter Second Number" size="25"
value=""></td>

</tr>

</table>

<input type=submit value="Submit">

</body>

</html>
```

The HTML source code for sum.html defines a table that contains two labels and two text fields. One of the labels is Enter First Number,and the other is Enter Second Number. 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.

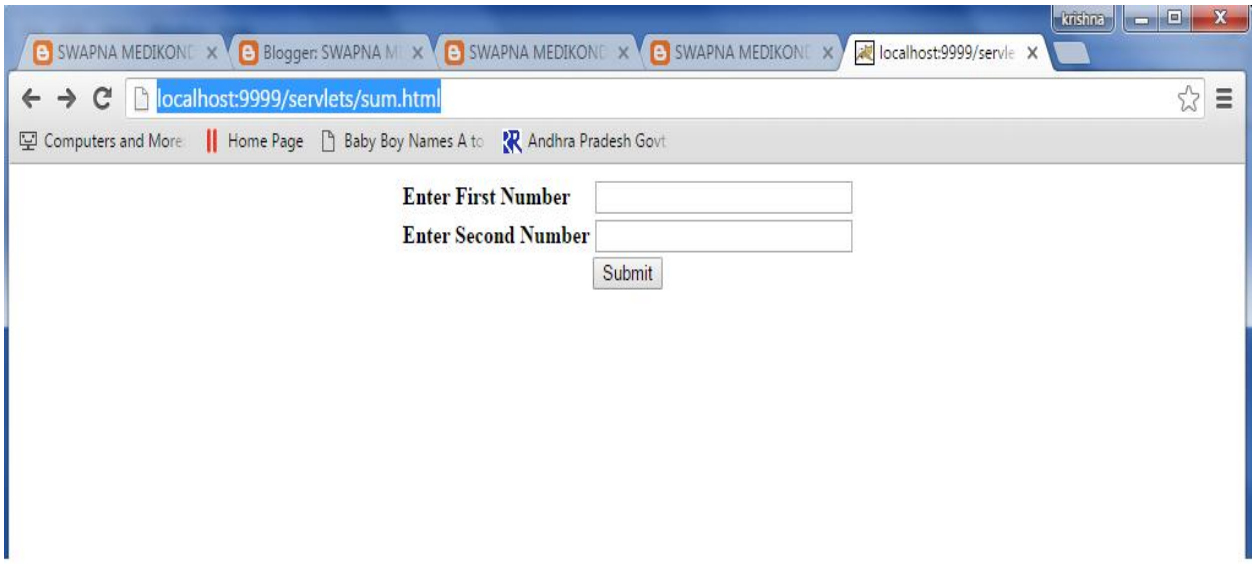
Add.java

```
import java.io.*;

import java.util.*;

import javax.servlet.*;

import javax.servlet.http.*;

public class Add

extends HttpServlet

{

public void doPost(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException

{

// Get print writer.

response.getContentType("text/html");

PrintWriter pw = response.getWriter();

// Get enumeration of parameter names.

Enumeration e = request.getParameterNames();

// Display parameter names and values.

int sum=0;

while(e.hasMoreElements())

{
```

```
String pname = (String)e.nextElement();

pw.print(pname + " = ");

String pvalue = request.getParameter(pname);

sum+=Integer.parseInt(pvalue);

pw.println(pvalue);

}

pw.println("Sum = "+sum);

pw.close();

}

}
```

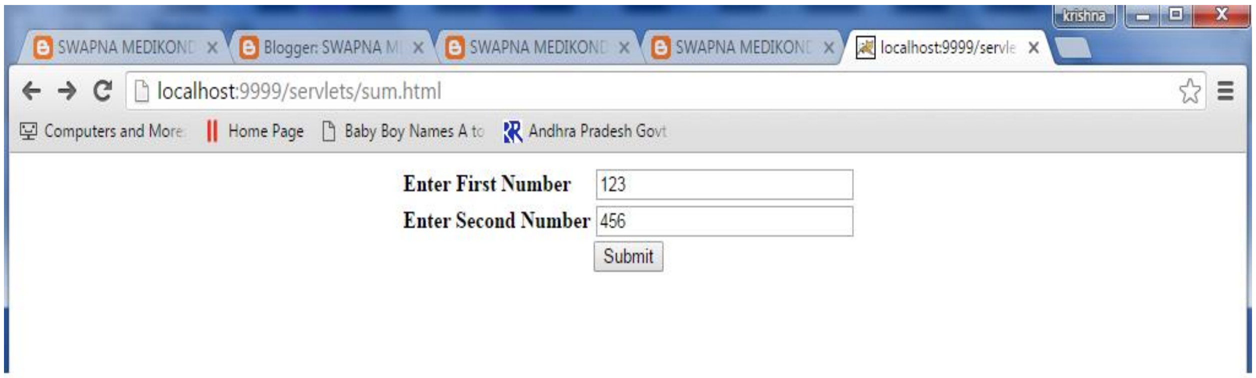
The source code for Add.java contains doPost( ) method is overridden to process client requests. The getParameterNames( ) method returns an enumeration of the parameter names. These are processed in a loop.we can see that the parameter name and value are output to the client. The parameter value is obtained via the getParameter( ) method.

after typing URL : http://localhost:9999/servlets/sum.html


localhost:9999/servlets/sum.html

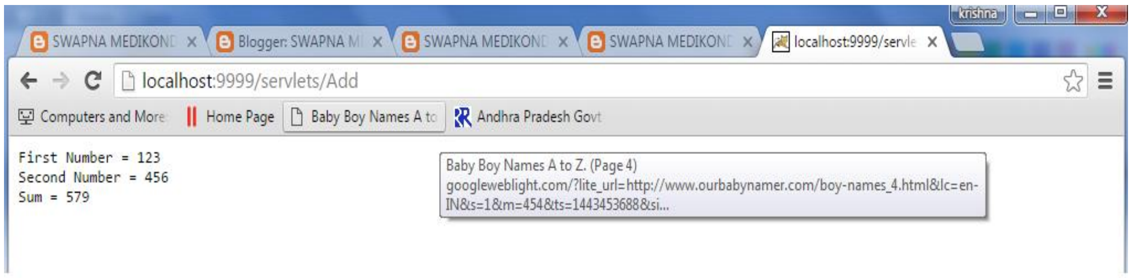
Enter First Number	
Enter Second Number	

Submit


localhost:9999/servlets/sum.html

Enter First Number	123
Enter Second Number	456

Submit



```
First Number = 123
Second Number = 456
Sum = 579
```

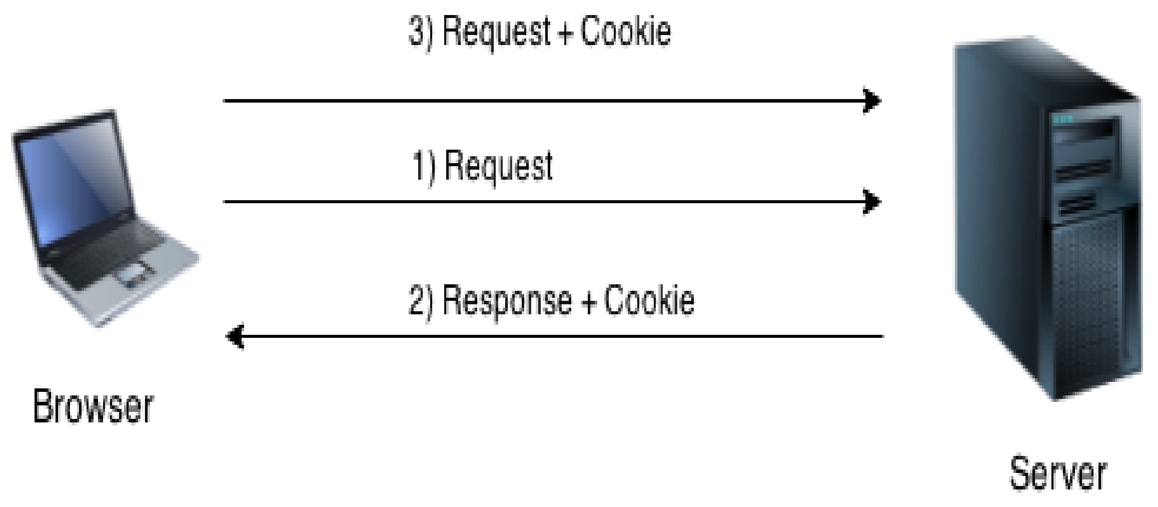
Cookies in Servlet

A **cookie** is a small piece of information that is persisted between the multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

How Cookie works

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.



Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

Non-persistent cookie

It is **valid for single session** only. It is removed each time when user closes the browser.

Persistent cookie

It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

Advantage of Cookies

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.

2. Only textual information can be set in Cookie object.

Note: Gmail uses cookie technique for login. If you disable the cookie, gmail won't work.

Cookie class

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

Constructor of Cookie class

Constructor	Description
Cookie()	constructs a cookie.
Cookie(String name, String value)	constructs a cookie with a specified name and value.

Useful Methods of Cookie class

There are given some commonly used methods of the Cookie class.

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

Other methods required for using Cookies

For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces.

They are:

1. **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add cookie in response object.
2. **public Cookie[] getCookies():**method of HttpServletRequest interface is used to return all the cookies from the browser.

How to create Cookie?

Let's see the simple code to create cookie.

```
Cookie ck=new Cookie("user","sonoo jaiswal");//creating cookie object

response.addCookie(ck);//adding cookie in the response
```

How to delete Cookie?

Let's see the simple code to delete cookie. It is mainly used to logout or signout the user.

```
Cookie ck=new Cookie("user","");//deleting value of cookie

ck.setMaxAge(0);//changing the maximum age to 0 seconds

response.addCookie(ck);//adding cookie in the response
```

How to get Cookies?

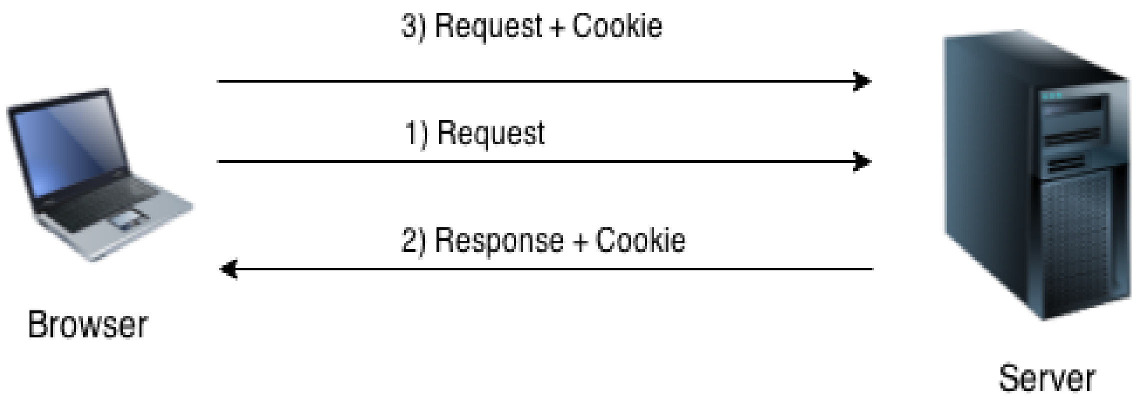
Let's see the simple code to get all the cookies.

```
Cookie ck[]=request.getCookies();
for(int i=0;i<ck.length;i++){
```

```
out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cookie
}
```

Simple example of Servlet Cookies

In this example, we are storing the name of the user in the cookie object and accessing it in another servlet. As we know well that session corresponds to the particular user. So if you access it from too many browsers with different values, you will get the different value.



index.html

```
<form action="servlet1" method="post">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet {
  public void doPost(HttpServletRequest request, HttpServletResponse response){

    try{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String n=request.getParameter("userName");
    out.print("Welcome "+n);
    Cookie ck=new Cookie("uname",n);//creating cookie object
    response.addCookie(ck);//adding cookie in the response
    //creating submit button
    out.print("<form action='servlet2'>");
    out.print("<input type='submit' value='go'>");
    out.print("</form>");
    out.close();
        }catch(Exception e){System.out.println(e);}
  }
}
```

SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SecondServlet extends HttpServlet {
public void doPost(HttpServletRequest request, HttpServletResponse response){
    try{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    Cookie ck[]=request.getCookies();
    out.print("Hello "+ck[0].getValue());
    out.close();
        }catch(Exception e){System.out.println(e);}
    }
}
```

web.xml

```
<web-app>
```

```
<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>
<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>
</web-app>
```

Servlet Login and Logout Example using Cookies ((Real Login App))

In this application, we have created following files.

1. index.html
2. link.html
3. login.html
4. LoginServlet.java
5. LogoutServlet.java
6. ProfileServlet.java
7. web.xml

File: index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Servlet Login Example</title>
</head>
```

```
<body>

<h1>Welcome to Login App by Cookie</h1>
<a href="login.html">Login</a>|
<a href="LogoutServlet">Logout</a>|
<a href="ProfileServlet">Profile</a>
</body>
</html>
```

File: link.html

```
<a href="login.html">Login</a> |
<a href="LogoutServlet">Logout</a> |
<a href="ProfileServlet">Profile</a>
<hr>
```

File: login.html

```
<form action="LoginServlet" method="post">
Name:<input type="text" name="name"><br>
Password:<input type="password" name="password"><br>
<input type="submit" value="login">
</form>
```

File: LoginServlet.java

```
package com.javatpoint;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class LoginServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse respon
se)
```

```java
                    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out=response.getWriter();
    request.getRequestDispatcher("link.html").include(request, response);
    String name=request.getParameter("name");
    String password=request.getParameter("password");
    if(password.equals("admin123")){
      out.print("You are successfully logged in!");
      out.print("<br>Welcome, "+name);
      Cookie ck=new Cookie("name",name);
      response.addCookie(ck);
    }else{
      out.print("sorry, username or password error!");
      request.getRequestDispatcher("login.html").include(request, response);
    }
    out.close();
  }
}
```

File: LogoutServlet.java

```java
package com.javatpoint;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class LogoutServlet extends HttpServlet {
  protected void doGet(HttpServletRequest request, HttpServletResponse respons
e)
                    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out=response.getWriter();
    request.getRequestDispatcher("link.html").include(request, response);
    Cookie ck=new Cookie("name","");
    ck.setMaxAge(0);
```

```
        response.addCookie(ck);
        out.print("you are successfully logged out!");
    }
}
```

File: ProfileServlet.java

```
package com.javatpoint;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class ProfileServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse respons
e)
                          throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();

        request.getRequestDispatcher("link.html").include(request, response);

        Cookie ck[]=request.getCookies();
        if(ck!=null){
         String name=ck[0].getValue();
        if(!name.equals("")||name!=null){
            out.print("<b>Welcome to Profile</b>");
            out.print("<br>Welcome, "+name);
        }
        }else{
            out.print("Please login first");
            request.getRequestDispatcher("login.html").include(request, response);
        }
        out.close();
    }
}
```

File: web.xml

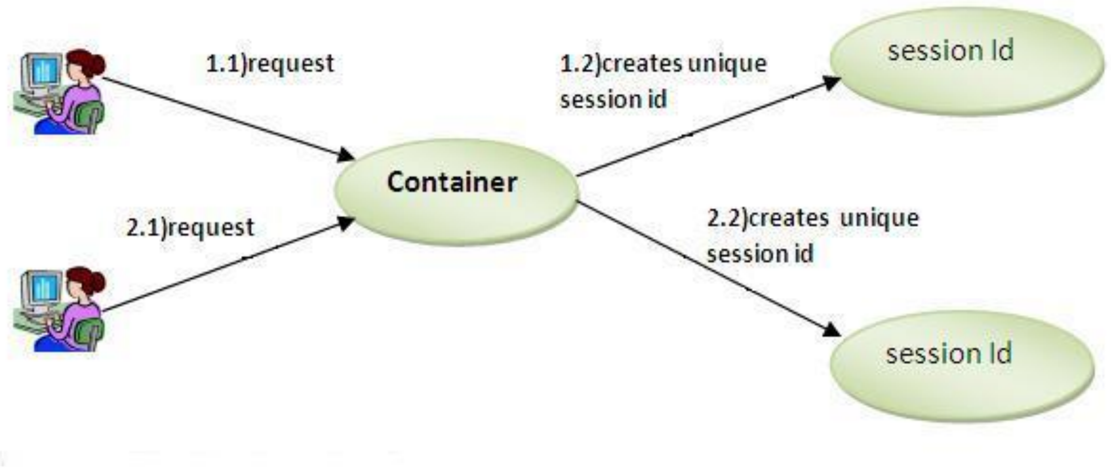
```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">

  <servlet>
    <description></description>
    <display-name>LoginServlet</display-name>
    <servlet-name>LoginServlet</servlet-name>
    <servlet-class>com.javatpoint.LoginServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>/LoginServlet</url-pattern>
  </servlet-mapping>
  <servlet>
    <description></description>
    <display-name>ProfileServlet</display-name>
    <servlet-name>ProfileServlet</servlet-name>
    <servlet-class>com.javatpoint.ProfileServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ProfileServlet</servlet-name>
    <url-pattern>/ProfileServlet</url-pattern>
  </servlet-mapping>
  <servlet>
    <description></description>
    <display-name>LogoutServlet</display-name>
    <servlet-name>LogoutServlet</servlet-name>
    <servlet-class>com.javatpoint.LogoutServlet</servlet-class>
  </servlet>
```

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

HttpSession interface

container creates a session id for each user.The container uses this id to identify the particular user.An object of HttpSession can be used to perform two tasks:

1. bind objects
2. view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



How to get the HttpSession object ?

The HttpServletRequest interface provides two methods to get the object of HttpSession:

1. **public HttpSession getSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **public HttpSession getSession(boolean create):**Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

Commonly used methods of HttpSession interface

1. **public String getId():**Returns a string containing the unique identifier value.

2. **public long getCreationTime():**Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. **public long getLastAccessedTime():**Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
4. **public void invalidate():**Invalidates this session then unbinds any objects bound to it.

Example of using HttpSession

In this example, we are setting the *attribute in the session scope in one servlet and getting that value from the session scope in another servlet.*

To set the attribute in the session scope, we have used the **setAttribute() method of HttpSession interface** and to **get** the attribute, we have used the **getAttribute method.**

index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

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

```

```

public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){
        try{
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String n=request.getParameter("userName");
        out.print("Welcome "+n);
        HttpSession session=request.getSession();
        session.setAttribute("uname",n);
        out.print("<a href='servlet2'>visit</a>");
        out.close();

                }catch(Exception e){System.out.println(e);}
    }

}
```

SecondServlet.java

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

public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
        try{

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        HttpSession session=request.getSession(false);
        String n=(String)session.getAttribute("uname");
        out.print("Hello "+n);

        out.close();
        }catch(Exception e){System.out.println(e);}
    }


}
```

web.xml

```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```

CONNECT DATABASE USING JDBC:

5 Steps to connect to the database in java

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

- Register the driver class
- Creating connection
- Creating statement
- Executing queries
- Closing 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.

Syntax of forName() method

1. **public static void** forName(String className)**throws** ClassNotFoundException

Example to register the OracleDriver class

1. 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.

Syntax of getConnection() method

1. **public static** Connection getConnection(String url)**throws** SQLException

2. **public static** Connection getConnection(String url,String name,String password) **throws** SQLException

Example to establish connection with the Oracle database

1. 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.

Syntax of createStatement() method

1. **public** Statement createStatement()**throws** SQLException

Example to create the statement object

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.

Syntax of executeQuery() method

1. **public** ResultSet executeQuery(String sql)**throws** SQLException

Example to execute query

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

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

5. }

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.

Syntax of close() method

1. **public void** close()**throws** SQLException

Example to close connection

con.close();

Handling Http Request & Responses :

Client HTTP Request:

When a browser requests for a web page, it sends lot of information to the web server which cannot be read directly because this information travel as a part of header of HTTP request. You can check **HTTP Protocol** for more information on this.

Methods to read HTTP Header
There are following methods which can be used to read HTTP header in your servlet program. These methods are available with *HttpServletRequest* object.

S.N.	Method & Description
1	**Cookie[] getCookies()** Returns an array containing all of the Cookie objects the client sent with this request.
2	**Enumeration getAttributeNames()** Returns an Enumeration containing the names of the attributes available to this request.
3	**Enumeration getHeaderNames()** Returns an enumeration of all the header names this request contains.

4	**Enumeration getParameterNames()** Returns an Enumeration of String objects containing the names of the parameters contained in this request.
5	**HttpSession getSession()** Returns the current session associated with this request, or if the request does not have a session, creates one.
6	**HttpSession getSession(boolean create)** Returns the current HttpSession associated with this request or, if if there is no current session and value of create is true, returns a new session.
7	**Locale getLocale()** Returns the preferred Locale that the client will accept content in, based on the Accept-Language header.

9	**ServletInputStream getInputStream()** Retrieves the body of the request as binary data using a ServletInputStream.
10	**String getAuthType()** Returns the name of the authentication scheme used to protect the servlet, for example, "BASIC" or "SSL," or null if the JSP was not protected.
11	**String getCharacterEncoding()** Returns the name of the character encoding used in the body of this request.
12	**String getContentType()** Returns the MIME type of the body of the request, or null if the type is not known.
13	**String getContextPath()** Returns the portion of the request URI that indicates the context of the request.
14	**String getHeader(String name)** Returns the value of the specified request header as a String.
15	**String getMethod()** Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.

Server HTTP Response :

when a Web server responds to an HTTP request, the response typically consists of a status line, some response headers, a blank line, and the document.

Methods to Set HTTP Response Header
There are following methods which can be used to set HTTP response header in your servlet program. These methods are available with *HttpServletResponse* object.

Method & Description	
1	**String encodeRedirectURL(String url)** Encodes the specified URL for use in the sendRedirect method or, if encoding is not needed, returns the URL unchanged.
2	**String encodeURL(String url)** Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged.
3	**boolean containsHeader(String name)** Returns a Boolean indicating whether the named response header has already been set.
4	**boolean isCommitted()** Returns a Boolean indicating if the response has been committed.
5	**void addCookie(Cookie cookie)** Adds the specified cookie to the response.
6	**void addDateHeader(String name, long date)** Adds a response header with the given name and date-value.
7	**void addHeader(String name, String value)** Adds a response header with the given name and value.
8	**void addIntHeader(String name, int value)** Adds a response header with the given name and integer value.
9	**void flushBuffer()** Forces any content in the buffer to be written to the client.
10	**void reset()** Clears any data that exists in the buffer as well as the

	status code and headers.
11	**void resetBuffer()** Clears the content of the underlying buffer in the response without clearing headers or status code.
12	**void sendError(int sc)** Sends an error response to the client using the specified status code and clearing the buffer.
13	**void sendError(int sc, String msg)** Sends an error response to the client using the specified status.
14	**void sendRedirect(String location)** Sends a temporary redirect response to the client using the specified redirect location URL.
15	**void setBufferSize(int size)** Sets the preferred buffer size for the body of the response.
16	**void setCharacterEncoding(String charset)**