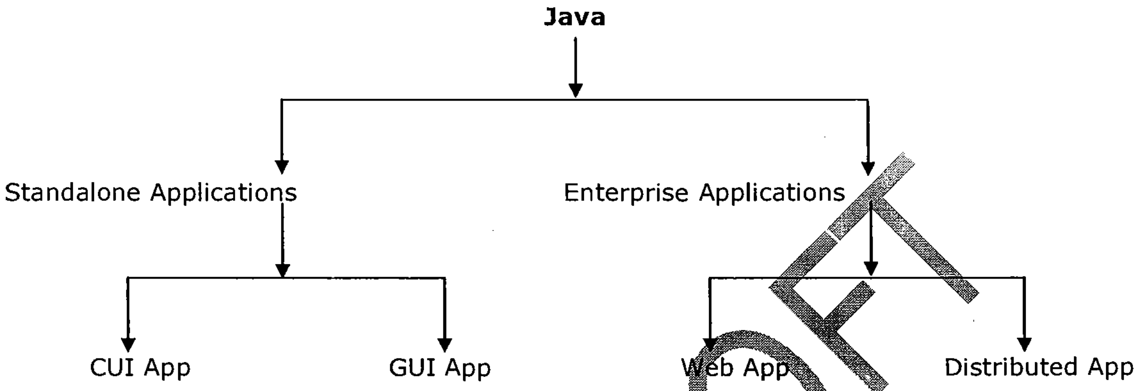
Introduction:

In general by using Java technology we are able to design following 2 types of applications.



1. Standalone Applications:

These are the java applications, which will be designed without using Client-Server Architecture.

There are 2 types of Standalone applications.

1. CUI Applications
2. GUI Applications

Q: What are the differences between CUI Applications and GUI Applications?

Ans: CUI Applications are the standalone applications, which will be designed in such a way to take input and to provide output by using Command prompt, where Command prompt is acting as user interface and it able to support character data. So that the java application which will design on the basis of command prompt i.e. character user interface is called as **CUI application**.

GUI Applications are the standalone applications, which will be designed in such a way to take input and to provide output by using a collection of Graphic component, where the collection of Graphic components is acting as an user interface and it able to support GUI component so that the java application which will be design on the basis of Graphical user interface is called as **GUI Application**.

2. Enterprise Applications:

These are the java applications, which will be designed on the basis of Client-Server Architecture. There are 2 types of Enterprise applications.

1. Web Applications
2. Distributed Applications

Q: What are the differences between Web Applications and Distributed Applications?

Ans: 1. Web Application is the server side application, it will be designed without disturbing its application logic over multiple number of JVM's.

Distributed application is the server side application it will be designed by disturbing application logic over multiple number of JVM's.

2. In general web applications will be designed by using a set of server side technologies called as Web technologies like CGI, Servlets, JSP's and so on.

Distributed applications will be designed by using a set of technologies called as Distributed technologies like Socket programming, RMZ, EJB's, Webservices and so on.

3. The main purpose of web applications is to generate dynamic response from server machine, but the main purpose of distributed applications is to establish distributed communication between local machine and remote machine in order to access the remote services.

4. In general web applications will provide services for web clients, but distributed applications will provide services for any type of clients.

5. In general web applications will be executed by using both web servers and application servers, but distributed applications will be executed by using only application servers.

1. Web Applications:

Web Application is a collection of web components, it will be executed by using web containers like Servlet Container to execute servlets, Jsp Container to execute JSP's and so on.

2. Distributed Applications:

Distributed Application is a collection of distributed components, it will be executed by using the distributed containers like EJB's Container to execute EJB's.

At the beginning stages of the computer we have Client-Server architecture, where the purpose of server is to hold up some resources and to share that resources to all the clients as per the client request.

In the above context, when we send a request from client to server for a particular resource then server will identify requested resource, pick up the content and send that content as response to client without performing any particular action at server. In this case the response which was generated by server is called as static response.

As per the application requirements we need to generate dynamic response from server, for this we need to execute the application at server called as Web Application. To design web applications at server side we need a set of server side technologies called as **Web Technologies.**

To design web applications we will use web technologies like CGI, Servlets, Jsp's, Perl, PHP and so on. Therefore, the main purpose of servlets and jsp's is to design web applications at server in order to generate dynamic response from server.

Q: To design web applications at server we have already CGI Technology then what is the requirement to go for servlets?

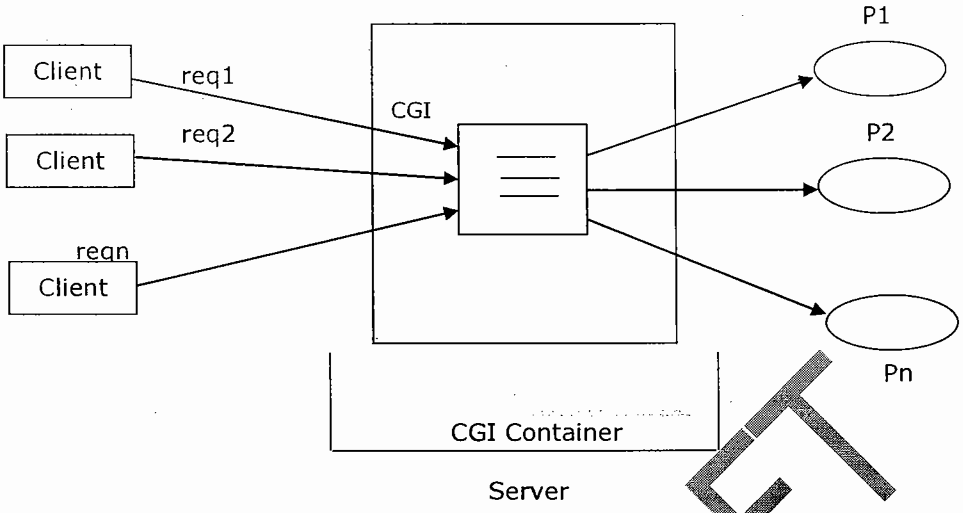
Ans: CGI is the server side technology designed on the basis of C technology and Scripting. C technology is the process based technology, it will make CGI technology as process based technology.

If we deploy any CGI application at server side then container will create a separate process for each and every request. If we increase number of requests automatically CGI container will generate number of processes at server.

Basically process is a heavy weight component, to handle single process system has to consume a lot of system memory and execution time.

Due to the above reason to handle the multiple number of processes at a time server machine may get burden, due to this reason server may generate delayed responses to the client request, it will reduce the performance of server side application.

In the above context, to increase the performance of the server side application we have to use an alternative server side technology i.e. Servlets.

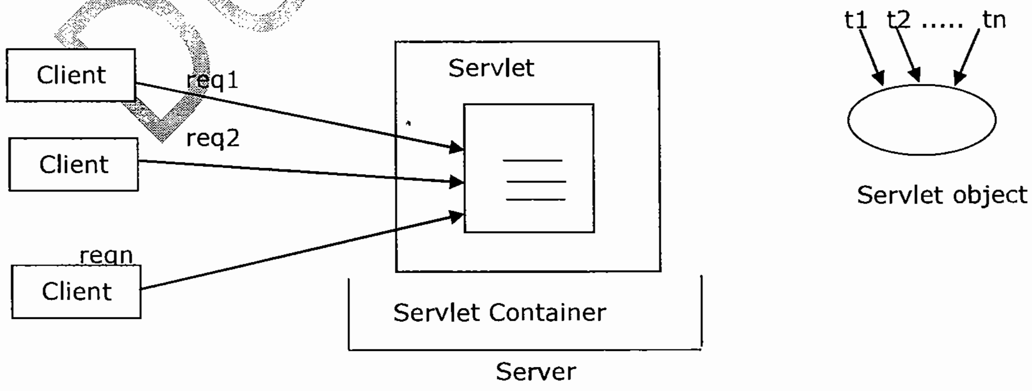


Servlet is the server side technology designed on the basis of java technology. Java technology is a Thread based technology, it will make servlets as Thread based technologies.

If we deploy any servlet application at server then for every client request servlet container will generate a separate thread on the respective servlet object.

In the above context, if we increase number of requests even container will create number of threads instead of processes.

When compared to process thread is light weight, to handle multiple number of requests i.e. thread server machine may not get burden, it may provide quick responses for the client request, it may increase the performance of server side application.



Q: What are the differences between servlets and jsps?

Ans: 1. If we want to design web applications by using Servlets we must require very good java stuff.

To design web applications by using Jsp technology it is not required to have java knowledge, it is sufficient to have presentation skills.

Note: The main intention to introduce Jsp technology is to reduce java code as much as possible in web applications.

2. In web applications, we will prefer to use Servlets to pick up the request and process the request.

But we will prefer to use Jsp's to generate dynamic response to the client with very good look and feel.

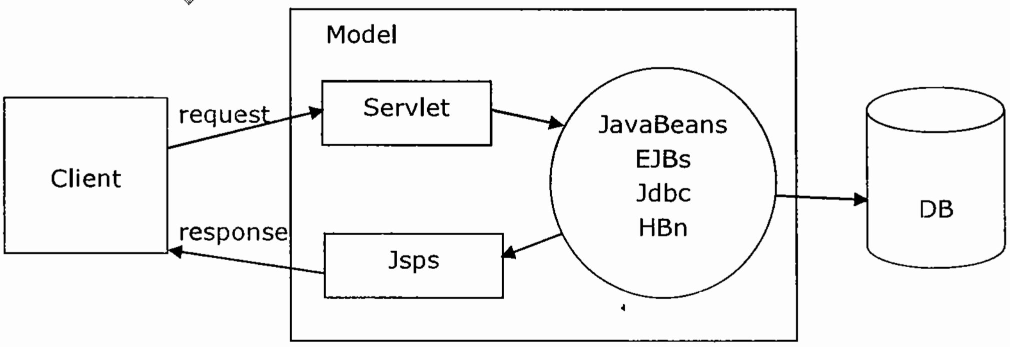
Note: In web applications, we will utilize Servlets to provide controller logic, integration logic, validation logic, implementing business logic and so on. But we will prefer to use Jsp technology only to provide presentation part.

3. In case of Servlets, we are unable to separate both presentation logic and business logic, but in case of Jsp's we are able to separate presentation logic and business logic because in Jsp pages we are able to use html tags to prepare presentation logic and we are able to use Jsp tags to prepare business logic.

4. If we perform any modifications on the existed Servlets then we have to perform recompilation and reloading on to the server explicitly.

If we perform any modifications on the existed Jsp's then it is not required to perform recompilation and reloading because Jsp pages are auto compiled and auto loaded.

5. If we want to design any web application on the basis of MVC architecture then we have to use a servlet as controller and a set of Jsp pages as view part.



Server

Note 1: Struts is a web application framework designed on the basis of MVC architecture, where we have to use ActionServlet as controller and we have to use a set of Jsp pages as view part.

Note 2: JSF(Java Server Faces) is a web application framework, where we have to use FacesServlet as a controller and a set of Jsp pages as view part.

Enterprise:

Enterprise is a business organization, a group of organizations running under a single label.

Enterprise Application:

It is a software application, which will be designed for a particular enterprise in order to simplify the internal business processing.

To design enterprise applications we have to use the following 3 layers.

1. User Interface Layer
2. Business Processing Layer
3. Data Storage and Access Layer

1. User Interface Layer:

User Interface Layer is the top most layer in enterprise application, it will provide starting point for the customers to interact with enterprise application.

The main purpose of User Interface Layer in enterprise applications is

1. To improve look and feel for the enterprise applications.
2. To accept user details in order to execute server side application.
3. To provide very good environment for client side validations with Javascript functions.
4. To specify different types of requests at client browser like get, post, head and so on.

To prepare User Interface Layer in enterprise application we will use a separate logic is called as **Presentation Logic.**

To prepare presentation logic in enterprise applications we have to use technologies like AWT, SWING, HTML, JSP, Velocity and so on.

2. Business Processing Layer:

Business Processing Layer is the heart of the enterprise application, it will provide very good environment to define and execute all the business rules and regulations which are required by the client.

To prepare Business Processing Layer in enterprise application we will use a separate logic is called as **Business Logic.**

In enterprise application development,to prepare Business logic we have to use technologies like Servlets, Jsp's, JavaBeans, EJBs and so on.

3. Data Storage and Access Layer:

This layer is the bottom most layer in enterprise applications.

The main purpose of this layer is to provide data persistency in enterprise application.

Data Storage and Access Layer will provide very good environment to provide the basic database operations(CRUD) as per the enterprise application requirement.

To prepare Data Storage and Access Layer we will use a separate logic called as **Persistence Logic.**

In enterprise application development,to prepare persistence logic we have to use technologies like Jdbc, ~~EJB-Entity Beans~~ Hibernate, JPA and so on.

To design enterprise application we need to define the degree of enterprise application, for this we have to use system architectures.

1-Tier Architecture

2-Tier Architecture

----

n-Tier Architecture

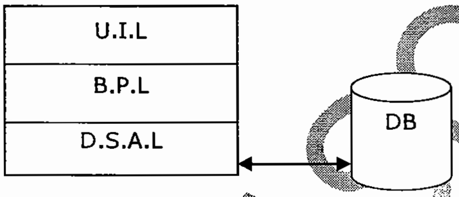
1-Tier Architecture:

To design enterprise application if we use 1-tier architecture then we have to provide User Interface Layer, Business Processing Layer and Data Storage and Access Layer within a single machine.

1-Tier architecture is highly recommended for standalone applications.

In 1-Tier architecture, a single machine we have to use to accommodate the complete application so that a single machine resources may not be sufficient to execute applications, it may reduce the performance of enterprise application.

1-Tier architecture will not provide any environment to handle multiple number requests, it is able to provide less sharability.



1-Tier Architecture

2-Tier Architecture:

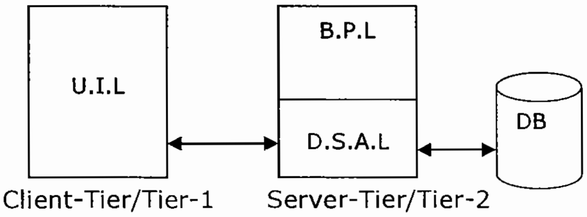
To design any enterprise application it is minimum to use 2-Tier architecture. The best example for 2-Tier architecture is Client-Server architecture.

2-Tier architecture will allow to design and execute the application in 2 layers of machines.

In case of 2-Tier architecture, tier-1 will manage User Interface Layer and tier-2 will manage Business Processing Layer and Data Storage and Access Layers.

2-Tier architecture will provide loosely coupled design when compared with 1-Tier architecture.

2-Tier architecture will provide very good environment to handle multiple number of requests and it is able to improve sharability.

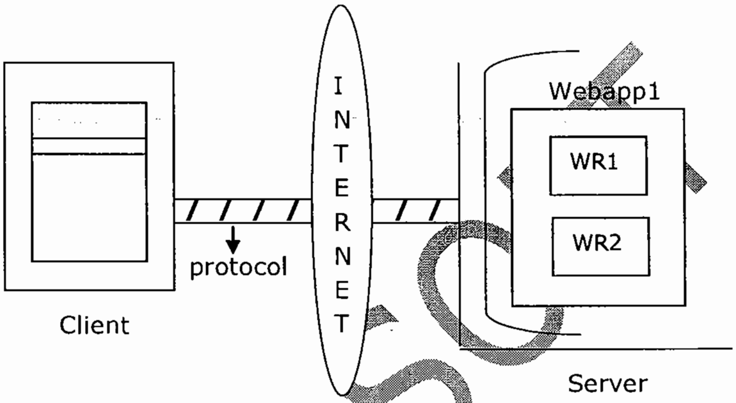


2-Tier Architecture

Note: As part of the enterprise application development always it is suggestible to use Tiered architectures because it will improve sharability, able to provide more loosely coupled design and so on, but we should not increase number of tiers in enterprise applications without having the requirement otherwise maintenance problems will be increased.

Client-Server Architecture:

BrowserContainer



From the above representation, there are three major components in client-server architecture.

1. Client
2. Protocol
3. Server

1. Client:

The main purpose of the Client in client-server architecture is to send request to the server and to set the responses from server.

To send request and to set response from server we need to use a tool at client machine called as **Browser.** In client-server application browser is acting as client.

To access a particular resource available at server from client browser we need to specify a particular string at browser address bar called **URI.**

There are two types of URI's

1. URL 2. URN

Q: What are the differences between URI, URL and URN?

Ans: URIis a string specification provided at client address bar, it can be used to refer a particular resource available at server machine.

URL is a string specification provided at client address bar, it can be used to refer a particular resource available at server machine through its locator.

URN is a string specification, it can be used to refer a particular resource available at server machine through its logical name.

Note 1: In case of servlets, locator is an URL pattern defined in web.xml file.

Note 2: In case of servlets, logical name is a name specified along with <servlet-name> tag in web.xml file.

Note 3: Almostall the servers are able to accept URL kind of request, but almost all the servers are not accept URN kind of request.

If we want to provide URL at client address bar then we have to use the following syntax.

Protocol_Name://Server_IP_Address:Server_Port_No/Application_Context/Resource_Name[Query_String]

Ex: http://121.120.92.98.8080/loginapp/logon?uname=abc&upwd=abc

Here Query_String, i.e. uname=abc&upwd=abc is optional.

Q: What is the difference between IP Address and Port Number?

Ans: IP Address is an unique identification for each and every machine over Network, which will be provided by Network Manager at the time of Network Configuration.

Port Number is an unique identification to each and every process being executed at the same machine and which could be provided by the local operating system.

Q: What is Query String and what is the purpose of Query String in web applications?

Ans: Query String is a collection of name-value pair appended to the URL, which can be used to pass input parameters to the respective server side resource in order to perform the required server side action.

2. Protocol:

The main job of the Protocol in client-server architecture is to carry the request data from client to server and to carry the response data from server to client.

Protocol is a set of rules and regulations, which can be used to carry the data from one machine to another machine over the Network.

Ex: TCP/IP, FTP, HTTP, SMTP, ARP, RARP........

In general in web applications, we will use http protocol to send request from client to server and to set response from server to client.

Q: What is the requirement of http protocol in web applications?

Ans: In web applications, to transfer the data between client and server we require a protocol, it should be

1. A Connectionless Protocol
2. A Stateless Protocol
3. A compatible Protocol to carry hypertext data.

Where Connectionless Protocol is protocol, it should not require a physical connection, but require a logical connection to carry the data.

Where Stateless Protocol is a protocol, which should not remember previous request data at the time of processing the later request.

In general in client server application, request data will be transferred from client to server in the form of hypertext data and the response data will be transferred from server to client in the form of hypertext data so that we require a Compatible Protocol to carry hypertext data between client and server.

Among all the protocols http protocol is able to satisfy all the above requirements so that we will use http protocol in web applications.

Q: How http protocol is able to manage stateless nature?

Ans: In client server applications, when we send a request from client to server protocol will pick up the request and perform the following actions.

1. Protocol will establish a virtual socket connection between client and server as per the server IP address and protocol which we provided in URL.
2. Protocol will prepare the request format with header part and body part, where header part will manage all the request headers(metadata about client) and body part will manage request parameters(the data which was provided by the user at client browser).
3. After preparing request format protocol will carry request format to server through the virtual socket connection.

Upon receiving request from protocol server will identify the request resource, execute generate dynamic response and dispatch that dynamic response to client. When server dispatch the dynamic response to the client protocol will pick up the response and perform the following actions.

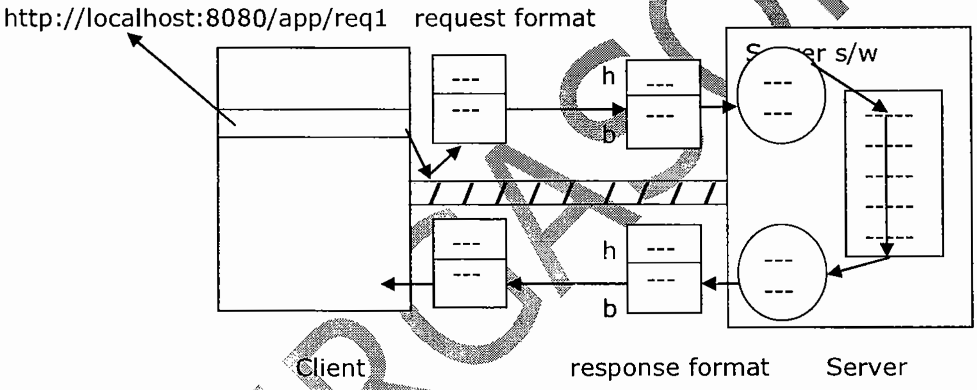
1. Protocol will prepare response format with header part and body part, where header part will manage response headers(metadata about the dynamic response) and body part will manage the actual dynamic response.

2. After setting response format protocol will carry response format to client.
3. When the dynamic response reached to client protocol will terminate the virtual socket connection, with this protocol will eliminate the present request data from its memory.

In the above context, the present request data will be managed by the protocol up to the connections existence, will protocol connection has terminated then protocol will not manage request data.

Due to the above reason http protocol is unable to manage clients previous request data at the time of processing later request. Therefore, http protocol is a stateless protocol.

Note: If we use http protocol, a stateless protocol in our web applications then we are unable to manage clients previous request data, but as per the application requirements we need to manage clients previous request data at the time of processing later request. In this context, to achieve the application requirement we have to use a set of explicit mechanism at server side called as **Session Tracking Mechanisms.**



In web applications, with http protocol we are able to specify different types of requests at client browser. The above flexibility is possible for the http protocol due to the availability of 7 number of http methods called as BIG 7 http methods.

Http protocol has provided the following http methods along with http1.0 version.

1. GET
2. POST
3. HEAD

Http protocol has provided the following http methods as per http1.1 version.

1. OPTIONS
2. PUT
3. TRACE

4. DELETE

Http1.1 version has provided a reserved http method i.e. CONNECT.

Q: What are the differences between GET and POST methods?

or

What are the differences between doGet(_,_) and doPost(_,_) methods?

Ans: 1. GET request type is default re quest type in web applications, but POST request type is not default request type.

2. GET request type should not have body part in the request format, but POST request type should have body part in the request format.

3. If we specify request parameters along with GET request then that request parameters will be transferred to server through request format header part due to the lack of body part.

In general request format header part will have memory limitation so that it is able to carry maximum 256 no. of characters. Therefore, GET request is able to carry less data from client to server

Due to the availability of body part in POST request all the request parameters will be transferred to server through request format body part, here there is no memory limitation in request format body part so that the POST request is able to carry large data from client to server.

4. If we specify request parameters along with GET request then GET request will display all the request parameters at client address bar as query string. Therefore, GET request is able to provide less security for the client data.

If we provide request parameters along with POST request then GET request will not display the request parameters at client address bar. Therefore, POST request is able to provide very good security for the client data.

5. In general in web applications, GET request can be used to get the data from server i.e. download the data from server.

In general in web applications, POST request can be used to post the data to server i.e. upload the data on to the server.

Note: Bookmarks are supported by GET request and which are not supported by POST request.

Q: What is the difference between GET request and HEAD request?

Ans: If we send Get request for a particular resource available at server machine then server will send only the requested resource as a response to client.

If we send HEAD request for a particular resource available at server then server will send requested resource as well as the metadata about the requested resource as response.

Note: Internally HEAD request uses GET request to get the requested resource from server.

OPTIONS Request:

The main purpose of the OPTIONS request type is to get the http methods which are supported by the present server.

Note: In general http protocol has provided by 7 http methods conceptually, supporting all the methods or some of the methods or none is completely depending on the server implementation provided by the server providers.

With this convention we are unable to credit how many number of http methods are supported by the present application server, where to credit the http methods which are supported by the present server we have to use OPTIONS request type.

Q: What is the difference between POST request and PUT request?

Ans: Both POST request and PUT request can be used to upload the data on the server machine. To upload the data on server machine if we use POST request then it is not mandatory to specify particular address location along with POST request.

To upload the data on server machine if we use PUT request then it is mandatory to specify server side location along with PUT request.

TRACE Request:

The main purpose of the TRACE request is to get the working status of a particular resource available at the server machine. TRACE request type is able to execute its functionality like echo server.

DELETE Request:

The main purpose of this request type is to delete a particular resource available at server machine.

Note: Almost all the servers may not support PUT, DELETE and request types as per their security constraints. In general almost all the servers are able to support GET and POST request types.

Status Codes:

In web applications, the main purpose of the status codes is to give the status of the request processing to the client.

Http1.1 version has provided all the status codes in the form of number representations. As per the web application requirement http1.1 version has provided the following status codes.

1xx --→ 100 to 199 --→ Inforamtional status codes

2xx --→ 200 to 299 --→ Success related status codes

3xx --→ 300 to 399 --→ Redirectional status codes

4xx --→ 400 to 499 --→ Client side error status codes

5xx --→ 500 to 599 --→ Server side error status codes

In general in web applications, when we send a request from client to server, server will identify the requested resource, execute it and generate dynamic response to client.

In the above context, when server dispatches the response protocol will pick up the response and prepare response format, where the dynamic response will be stored in the response format body part.

At the time of processing response format server will provide the respective status code value in the response format header part i.e. with status line field.

When protocol carry the response format to the client then client will pick up the status code value from status line field, prepare itself to get the response from response format body part.

3. Server:

The main purpose of the server in client server applications is to pick up the request from client, identify the requested resource, generate the dynamic response and dispatch dynamic response to client.

Note: Servlet is a program available at server machine, it is not capable to pick up the request and dispatch response to client, if server execute servlet program then some dynamic response will be generated.

Examples of Servers: Apache Tomcat, BEA Weblogic, IBM Websphere, Macromedia JRun, SUN Sunone, J2EE, GlassFish and so on.

There are 2 types of servers to execute enterprise applications.

1. Web Servers
2. Application Servers

Q: What are the differences between Web Servers and Application Servers?

Ans: 1.Web Server is a server, which will provide very good environment to execute web applications only. But Application servers will provide very good environment to execute any type of J2EE applications like web applications, distributed applications and so on.

2. In general web servers will not provide all the middle ware services. But application servers will provide all the middle ware services like JND, Jdbc and so on as in-built support.

Application server = Web server + Middleware services

Note: Initially the main intention of web servers is to execute static resources in order to generate static response and the main intention of application servers is to execute dynamic resources in order to generate dynamic response.

If we want to specify a particular machine has server machine then we have to install a particular server software, when we install server software on server machine automatically that server software will be available in the form of the following 2 modules.

1. Main Server
2. Container

Q: What is the difference between Main server and Container?

Ans: When we send a request from client to server then main server will pick up the request from protocol and check whether the request data is in well-formed formator not, if it is not in well-formed format then main server will stop request their itself and generate the respective response to client.

If the request data is in well-formed format then that request will by pass to container, wherecontainer will identify the requested resource, execute it, generate dynamic response and dispatch dynamic response to main server, where main server will bypass response to client through the protocol.

In general containers could be classified into the following 2 ways.

1. As per the technology which we used to design server side component. There are some containers

1. Servlet Container --→ To execute servlets

2. Jsp Container --→ To execute Jsp's

3. EJB Container --→ To execute EJB's

Note: All the above specified containers can be used to execute the respective components because the above containers have implemented the respective technology API.

2. As per the containers physical existency there are 3 types of containers.

1. Standalone Container:

It is an integration of main server and container as a single program.

2. Inprocess Container:

It is a container existed inside the main server.

3. Out of process Container:

It is a container existed outside of the main server.

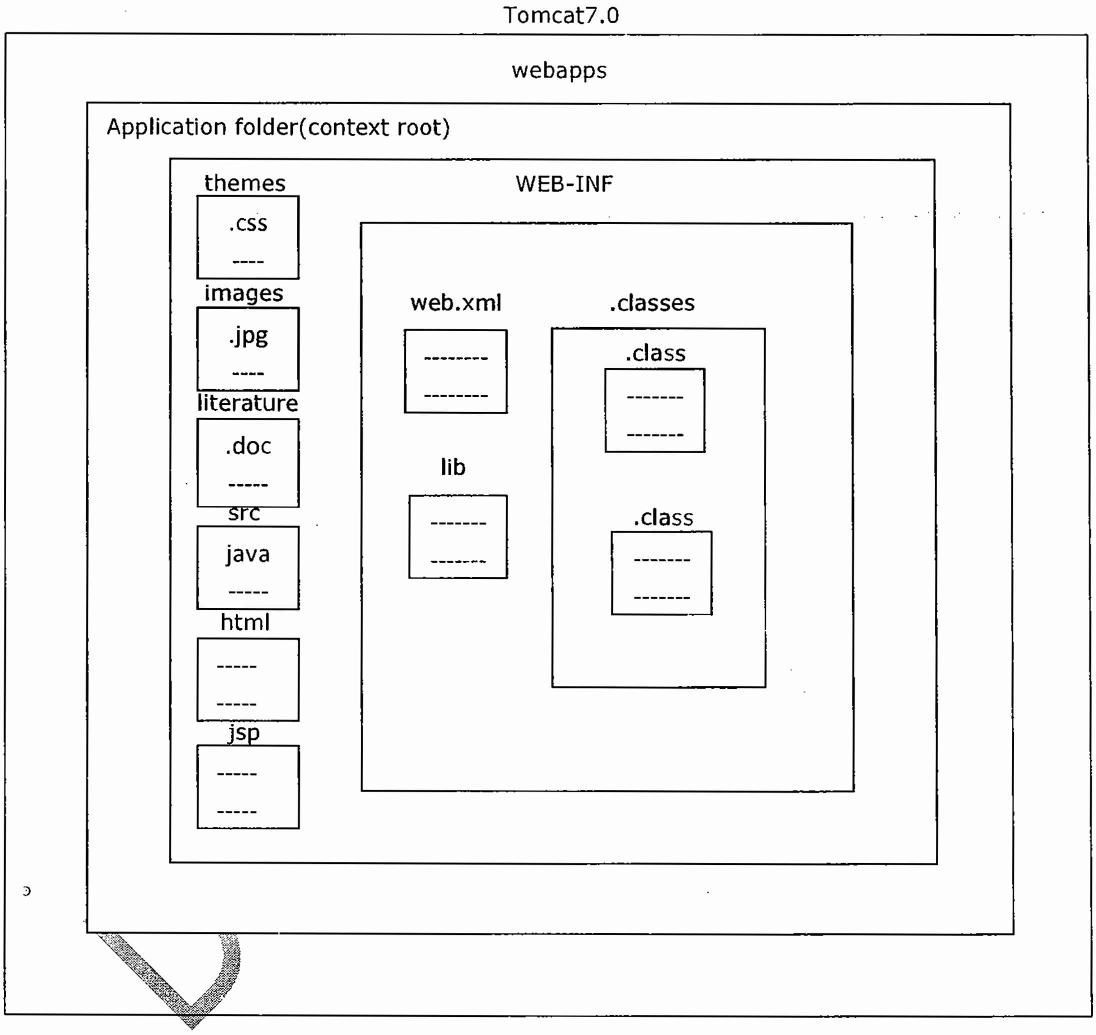
Steps to design first web application(Servlet application):

To design first web application we have to use the following steps.

1. Prepare web application directory structure
2. Prepare deployment descriptor
3. Prepare the required web resources like html, servlets, Jsp's and so on
4. Start the server
5. Access the web application

Step 1: Web Application Directory Structure:

If we want to design any web application then we have to prepare the following directory structure at server machine.



When we install Tomcat server automatically Tomcat7.0 folder will be created. In Tomcat7.0 folder, the main purpose of webapps folder is to accommodate all the web applications provided by the developer.

To represent our own web applications in server we have to prepare a separate folder under webapps folder i.e. application folder or context root. Inside the application folder,

1. Themes: To store .css files(cascade style sheet) to generate reports.

2. Images: To store logos of the organizations, background sceneries and so on in the form of .jpg, .jpeg, .gif files.

3. Literature: To store documentations in the form of .doc, .docex and so on.

4. src(Source code): It can be used to store all the source files like Java files.

5. Along with all these folders it is possible to provide some static resources directly like .html files and dynamic resources like .jsp files.

6. WEB-INF folder will include

1. web.xml: This xml file can be used to store the metadata about the web application required by the container to perform a particular action.

2. lib: It is able to manage jar files which are required by the server side components.

3. classes: It will include .class files of servlets, filters, listeners and so on.

The above web application directory structure was divided into the following

1. Private area:

In web application directory structure, WEB-INF folder and its internal is treated as private area. If we deploy any resource under private area then client is unable to access that resource by using its name directly.

Note: In general we will keep servlet .class files under classes folder i.e. private area so that we are unable to access that servlet by using its name directly from client.

To access any servlet we have to define an URL pattern in web.xml file w.r.t this servlet, with the defined URL pattern only client is able to access the respective servlets.

2. Public area:

The area which is in outside of WEB-INF folder and inside the application folder is treated as public area. If we deploy any resource under public area then client is able to access that resource by using its name directly.

Note: In general in web applications, we are able to keep html files under application folder i.e. public area so that we are able to access that resources by using its name directly from client.

Step 2: Deployment Descriptor or web.xml file:

Deployment Descriptor is web.xml file, it can be used to provide the metadata about the present web application required by the container in order to perform a particular server side action.

In web applications, web.xml file include the following configuration details w.r.t the web application

1. Welcome Files Configuration
2. Display Name Configuration
3. Servlets Configuration
4. Filters Configuration
5. Listeners Configuration
6. Context Parameters Configuration
7. Initialization Parameters Configuration
8. Session Time Out Configuration
9. Load On Startup Configuration
10. Error Page Configuration
11. Tag Library Configuration
12. Security Configuration

In general in web applications, we will deploy the servlets .class files under classes folder of the web application directory structure i.e. private area.

If we deploy any resource under private area then client is unable to access that resource through its name, client is able to access that resource through alias names or locators. In case of servlets, client is able to access servlet classes through the locators called as **URL Patterns.**

If we provide multiple number of servlets under classes folder and we provide a particular request to a particular servlet available under classes folder with an URL pattern then container should require mapping details between URL patterns and servlets class names as metadata in order to identify w.r.t servlet.

In the above context, to provide the required metadata to the container we have to provide servlet configuration in web.xml file. To provide servlet configuration in web.xml file we have to use the following xml tags.

```
<web-app>

    --------

<servlet>

<servlet-name>logical_name</servlet-name>

<servlet-class>fully qualified name of servlet</servlet-class>
```

```
</servlet>

<servlet-mapping>

<servlet-name>logical_name</servlet-name>

<url-pattern>urlpattern_name</url-pattern>

</servlet-mapping>

    --------

</web-app>
```

Note: In the above servlets configuration, <servlet-name> tag value under <servlet> tag and <servlet-mapping> tag must be same.

Ex:

```
<web-app>

<servlet>

<servlet-name>loginservlet</servlet-name>

<servlet-class>com.dss.login.LoginServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>loginservlet</servlet-name>

        <url-pattern>/login</url-pattern>

</servlet-mapping>

</web-app>
```

If we want to access the above servlet then we have to provide the following URL at client browser.

http://localhost:8080/loginapp/login

In servlet configuration, there are 3 ways to define URL patterns.

1. Exact Match Method
2. Directory Match Method
3. Extension Match Method

1. Exact Match Method:

In Exact Match Method, we have to define an URL pattern in web.xml file, it must be prefixed with forward slash("/") and pattern name may be anything.

Ex: <url-pattern>/abc/xyz</url-pattern>

If we define any URL pattern with exact match method then to access the respective resource we have to provide an URL pattern at client address bar along with URL, it must be matched with the URL pattern which we defined in web.xml file.

Ex:	http://localhost:8080/app1/abc/xyz	Valid
	http://localhost:8080/app1/xyz/abc	Invalid
	http://localhost:8080/app1/xyz	Invalid
	http://localhost:8080/app1/abc	Invalid

Note: In general in web applications, we will prefer to use exact match method to define an URL pattern for a particular servlet when we have a requirement to access respective servlet independently.

2. Directory Match Method:

In Directory Match Method, we have to define an URL pattern in web.xml file, it must be prefixed with forward slash("/") and it must be terminated with "*".

Ex: <url-pattern>/abc/*</url-pattern>

If we define any URL pattern with this method then to access the respective resource from client we have to specify an URL pattern at client address bar it should match its prefix value with the prefix value of the URL pattern defined in web.xml file.

Ex:	http://localhost:8080/app1/abc/xyz	Valid
	http://localhost:8080/app1/xyz/abc	Invalid
	http://localhost:8080/app1/abc	Valid
	http://localhost:8080/app1/abc/abc	Valid

Note 1: In general in web applications, we will prefer to use directory match method to define an URL pattern when we have a requirement to pass multiple number of requests to a particular server side resource.

Note 2: In web applications we will use Filters to provide preprocessing and post processing to one or more number of servlets. In this context, when we send a request to respective servlet then container will bypass all the requests to the respective Filter.

To achieve this type of requirement we have to use directory match method to define an URL pattern for the respective Filter.

3. Extension Match Method:

In Extension Match Method, we have to define an URL pattern in web.xml file, it must be prefixed with "*" and it must be terminated with a particular extension.

Ex: <url-pattern>*.do</url-pattern>

If we define an URL pattern with this method then to access the respective server side resource from client we have to specify an URL pattern at client address bar, it may start with anything, but must be terminated with an extension which was specified in web.xml file.

Ex:	http://localhost:8080/app1/login.do	Valid
	http://localhost:8080/app1/reg.do	Valid
	http://localhost:8080/app1/add.xyz	Invalid
	http://localhost:8080/app1/search.doo	Invalid

Note 1: In general in web applications, we will prefer to use extension match method to define an URL pattern when we have a requirement to trap all the requests to a particular server side resource and to perform the respective server side action on the basis of the URL pattern name if we provided at client browser.

Note 2: If we design any web application on the basis of MVC then we have to use a servlet as controller, where the controller servlet has to trap all the requests and it has to perform a particular action on the basis of URL pattern name. Here to define URL pattern for the controller servlet we have to use extension method.

In web applications, web.xml file is mandatory or optional is completely depending on the server which we used.

In Apache Tomcat Server, web.xml file is optional when we have not used servlets, filters and so on. In Weblogic Server, web.xml file is mandatory irrespective of using servlets, filters and so on.

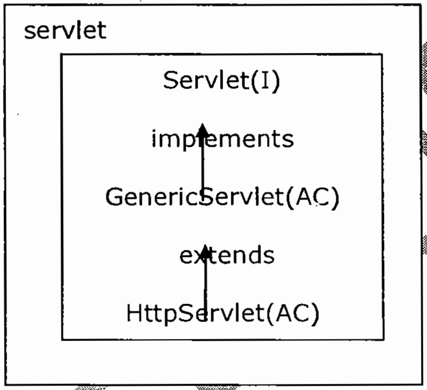
Up to servlets2.5 version[J2EE5.0] it is mandatory to provide web.xml file if we use servlets, listeners, filters and so on in our application. But in servlets3.0 version, there is a replacement for web.xml file i.e. Annotations, Annotations will make web.xml file is optional.

In web applications, it is not possible to change the name and location of the deployment descriptor because container will search for deployment descriptor with web.xml name under WEB-INF folder as per its predefined implementation.

Step 3: Prepare Web Resources:

If we want to prepare custom exceptions, threads and so on in Java applications then we have to use some predefined library provided by Java API.

Similarly if we want to prepare servlets in our web applications then we have to use some predefined library provided by Servlet API.



servlet

Servlet(I)

implements

GenericServlet(AC)

extends

HttpServlet(AC)

To design servlets Servlet API has provided the following predefined library as part of javax.servlet package and javax.servlet.http package.

Q: What is Servlet? and in how many ways we are able to prepare servlets?

Ans: Servlet is an object available at server machine which must implement either directly or indirectly Servlet interface.

As per the predefined library provided by Servlet API, there are 3 ways to prepare servlets.

1. Implementing Servlet interface:

In this approach, if we want to prepare servlet then we have to take an user defined class which must implement Servlet interface.

```
public class MyServlet implements Servlet
{ --------
  -------- }
```

2. Extending GenericServlet abstract class:

In this approach, if we want to prepare servlet then we have to take an user defined class as a subclass to GenericServlet abstract class.

```
public class MyServlet implements GenericServlet
{ --------
  -------- }
```

3. Extending HttpServlet abstract class:

In this approach, if we want to prepare servlet then we have to take an user defined class as a subclass to HttpServlet abstract class.

```
public class MyServlet implements HttpServlet
{ --------
  -------- }
```

Step 4: Start the Server:

There are 3 ways to start the server.

1. Execute either startup.bat file or Tomcat7 Service Runner available under bin folder of Tomcat server.
   C:\Tomcat 7.0\bin
2. Use system program monitor Tomcat
   Start ⟶ All Programs ⟶ Apache Tomcat 7.0 ⟶ Monitor Tomcat
3. Use Apache Tomcat System Service
   Start ⟶ Type services.MVC in search field ⟶ Select Apache Tomcat 7.0 ⟶ Select Start Service Icon.

Step 5: Access the Web Application:

There are 2 ways to access the web applications.

1. Open web browser and type the complete URL on address bar.
   http://localhost:8080/app1/servlet
2. Open web browser type the URL up to
   http://localhost:8080

If we do the above automatically the Tomcat Home Page will be opened, where we have to select Manager Applications, provide username and password in Security window and click on OK button.

If we do the above automatically list of applications will be opened, where we have to select the required application.

First Approach to Design Servlets (Implementing Servlet interface):

If we want to design a servlet with this approach then we have to take an user defined class it should be an implementation class to Servlet interface.

```
  public interface Servlet {
     public void init(ServletConfig config)throws ServletException;
     public void service(ServletRequest req, ServletResponse res)throws ServletExcepption;
     public ServletConfig getServletConfig();
     public String getServletInfo();
     public void destroy();
}
 public class MyServlet implements Servlet
  { -------
     ------- }
```

Where the purpose of init(_) method to provide servlets initialization.

Note: In servlets execution, to perform servlets initialization container has to access init(_) method. To access init(_) method container has to create ServletConfig object.

ServletConfig is an object, it will manage all the configuration details of a particular servlet like logical name of servlet, initialization parameters and so on.

In servlet, the main purpose of service(_,_) method is to accommodate the complete logic and to process the request by executing application logic.

Note: In servlet, service(_,_) method is almost all same as main(_) method in normal Java application.

When we send a request from client to server then container will access service(_,_) method automatically to process the request, where the purpose of getServletConfig() method is to get ServletConfig object at servlet initialization.

Where getServletInfo() method is used to return generalized description about the present servlet.

Where destroy() method can be used to perform servlet reinstantiation.

To prepare application logic in service(_,_) method we have to use the following conventions.

1. Specify response content type:

To specify response content type to the client we have to use the following method from ServletResponse.

public void setContentType(String MIME_TYPE)

where MIME_TYPE may be text/html, text/xml, image/jpeg, img/jpg and so on.

Ex: res.setContentType("text/html");

Note: The default content type in servlets is text/html.

When container encounters the above method then container will pick up the specified MIME_TYPE and container will set that value to content type response header in the response format.

When protocol dispatch the response format to client, before getting the response from response format body part first client will pick up content type response header value i.e. MiME_TYPE, client will prepare itself to hold the response as per the MIME_TYPE.

2. While executing the servlet we have to generate some dynamic response on response object, where to carry the dynamic response to the response object Servlet API has provide a predefined PrintWriter object internally.

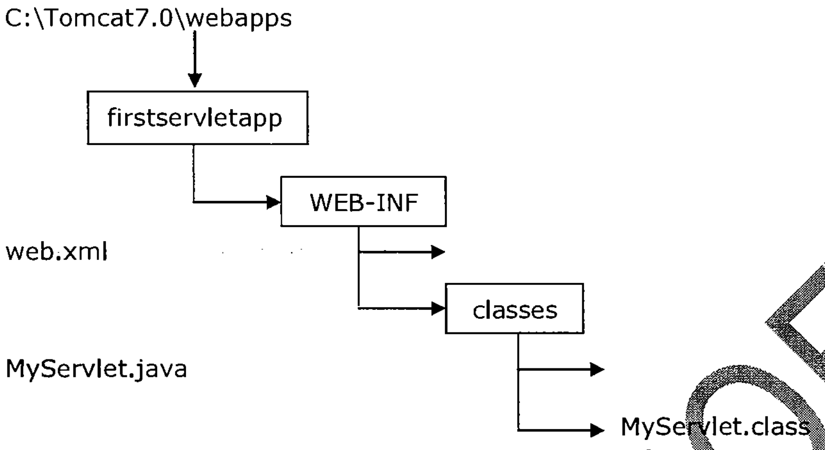
To get the predefined PrintWriter object we have to use the following method from ServletResponse.

public PrintWriter getWriter()

Ex: PrintWriter out=res.getWriter();

Steps to Design First Servlet Application:

1. Web application Directory Structure:



2. Prepare Deployment Descriptor(web.xml):

web.xml:-

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

3. Prepare Servlet:

MyServlet.java:-

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.Servlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;


import javax.servlet.ServletResponse;

public class MyServlet  implements Servlet {

   public MyServlet() { }

public void init(ServletConfig config) throws ServletException {}

public void destroy() {}

public ServletConfig getServletConfig() {
        return null;
}

public String getServletInfo() {
        return null;
}

public void service(ServletRequest request, ServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        out.println("<h1><center>hello</center></h1>");
}

}
```

To compile the above servlet we have to set classpath environment variable to servlet-api.jar provided by Tomcat server at C:\Tomcat7.0\lib\servlet-api.jar.

Ex: set classpath=%classpath%;C:\Tomcat7.0\lib\servlet-api.jar;

To access the above application we have to use the following URL on client address bar.

http://localhost:2020/firstservletapp/ms

Servlets Flow of Execution:

When we start the server the main job of container is to recognize each and every web application and to prepare ServletContext object to each and every web application.

While recognizing web application container will recognize web.xml file under WEB-INF folder then perform loading, parsing and reading the content of web.xml file.

While reading the content of web.xml file, if container identifies any context data in web.xml file then container will store context data in ServletContext object at the time of creation.

After the server startup when we send a request from client to server protocol will pick up the request tehn perform the following actions.

1. Protocol will establish virtual socket connection between client and server as part of the server IP address and port number which we specified in the URL.
2. Protocol will prepare a request format having request header part and body part, where header part will maintain request headers and body part will maintain request parameters provided by the user.
3. After getting the request format protocol will carry request format to the main server.

Upon receiving the request from protocol main server will check whether the request data is in well-formed format or not, if it is in well-formed then the main server will bypass request to container.

Upon receiving the request from main server container will pick up application name and resource name from request and check whether the resource is for any html page or Jsp page or an URL pattern for a servlet.

If the resource name is any html page or Jsp page then container will pick up them application folder and send them as a response to client.

If the resource name is an URL pattern for a particular servlet available under classes folder then container will go to web.xml file identifies the respective servlet class name on the basis of the URL pattern.

After identifying the servlet class name from web,xml file container will recognize the respective servlet .class file under classes folder then perform the following actions.

Step 1: Servlet Loading:

Here container will load the respective servlet class byte code to the memory.

Step 2: Servlet Instantiation:

Here container will create a object for the loaded servlet.

Step 3: Servlet Initialization:

Here container will create ServletConfig object and access init(_) method by passing ServletConfig object reference.

Step 4: Creating request and response objects(Request Processing):

After the servlet initialization container will create a thread to access service(_,_) method, for this container has to create request and response objects.

Step 5: Generating Dynamic response:

By passing request and response objects references as parameters to the service(_,_) method then container will access service(_,_) method, executes application logic and generate the required response on response object.

Step 6: Dispatching Dynamic response to Client:

When container generated thread reaching to the ending point of service(_,_) method then container will keep the thread in dead state, with this container will dispatch the dynamic response to main server from response object, where main server will bypass the response to the protocol.

When protocol receives the response from main server then protocol will prepare response format with header part and body part, where header part will manage all the response headers and body part will manage the actual dynamic response.

After getting response format protocol will carry that response format to the client.

Step 7: Destroying request and response objects:

When the response is reached to the client protocol will terminate the virtual socket connection between client and server, with this container destroy the request and response objects.

Step 8: Servlet Deinstantiation:

When container destroy request and response objects then container will go to the waiting state depends on the container implementation, if container identifies no further request for the same resource then container will destroy servlet object.

Note: In servlet execution, container will destroy ServletConfig object just before destroying the servlet object.

Step 9: Servlet Unloading:

After servlet deinstantiation container will eliminate the loaded servlet byte code from operational memory.

Step 10: Destroying ServletContextobject:

In the above servlet life cycle, all the objects like request, response and ServletConfig are destroyed before servlet deinstantiation, but still Servlet object is available in memory.

In general ServletContext object will be destroyed at the time of server shut down.

Drawbacks of First Approach:

To design servlets if we use this approach we have to provide implementation for each and every method declared in Servlet interface irrespective of the actual application requirement.

The above approach will increase burden to the developers and it will increase unnecessary methods in web applications.

To overcome the above problem we have to use an alternative i.e. GenericServlet.

Second Approach to Design Servlets (Extendiging GenericServlet abstract class):

If we want to design servlets by using this approach then we have to take an user defined class which must be a subclass to GenericServlet abstract class.

```
 public abstract class GenericServlet implements Servlet, ServletConfig, Serializable {
    private transient ServletConfig config;
    public void init(ServletConfig config)throws ServletException {
       this,config=config;
       init();
    }
    public void init()
    {  }
    public abstract void service(ServletRequest req, ServletResponse res)throws SE, IOE;
    public ServletConfig getServletConfig() {
      return config;
    }
    public String getServletInfo() {
       return null;
    }
    public void destroy()  {  }
 }
 public class MyServlet extends GenericServlet
{ -------
 ------- }
```

From the above predefined implementation of GenericServlet abstract class,

1. GenericServlet is an idea came from Adapter Design Pattern.
2. GenericServlet predefined abstract class has implemented Serializable interface so that all the GenericServlet objects(subclass objects of GenericServlet abstract class) are eligible for Serialization and Deserialization by default.
3. It is possible to serialize GenericServlet objects, but config predefined reference variable will not be participated in serialization and deserialization because config reference variable has declared as transient.
4. In GenericServlet abstract class, init(_) method is overloaded method.
5. In GenericServlet abstract class, still service(_,_) method is an abstract method.

Q: What is the requirement to override init(_,_) method in servlet applications?

Ans: In general as part of servlets design we will use service(_,_) method to provide application logic. In some cases, application logic may include the actual business logic and its prerequirement.

If we provide both prerequirement code and business logic within service(_,_) method then the performance of the application will be reduced because container will execute business logic and its prerequirement code for every request send by the client, but as per the convention it is sufficient to execute prerequirement code only one time.

In the above context, to improve the performance of servlet applicationwe have to provide the actual business logic in service(_,_) method and the prerequirement code in init(_) method because container will execute init(_) method only one time, but service(_,_) method for every request.

Note: In servlet applications, always it is suggestible to override init() method(second init() method) but if our prerequirement code may include any dat from ServletConfig object then it is suggestible to override init(ServletConfig config) method.

--------------Application by using GenericServlet---------------

Genericservletapp:-

web.xml:-

```
<web-app>
        <servlet>
                <servlet-name>GenericDemo</servlet-name>
                <servlet-class>GenericDemo</servlet-class>
        </servlet>
```

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

GenericDemo.java:-

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.GenericServlet;


import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class GenericDemo extends GenericServlet {

	public void service(ServletRequest request, ServletResponse response) throws
ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter out=response.getWriter();
		out.println("<h1><center>hello</center></h1>");
	}

}
```

The main difference between first approach and second approach of designing servlets in the point of flow of execution, in first approach container will execute only one init(_) method in servlet initialization, but in second approach container will execute two init() methods in servlet initialization.

Third Approach to Design Servlets (Extendiging HttpServlet abstract class):

Q: Whar are the differences between GenericServlet and HttpServlet?

Ans: 1. GenericServlet is protocol independent, but HttpServlet is http protocol dependent.

2. In case of GenericServlet, container will execute only service(_,_) method for any type of protocol request provided by the client, but In case of HttpServlet, container will execute a separate method on the basis request type which will be specified by the client.

Ex: If we specify Get request type at client browser then HttpServlet is able to execute doGet(_,_) method, for POST request type HttpServlet will execute doGet(_,_) method.

3. GenericServlet is not very good compatible with protocols, but HttpServlet is very good compatible with http protocol.

4. GenericServlet will not implement any specific protocol at server side, but HttpServlet has implemented http protocol at server side.

5. GenericServlet will not give any option to the developers to specify different types of requests at client browser, but HttpServlet will provide flexibility to the developers to specify different types of requests at client browser.

If we want to prepare servlets by using HttpServlet abstract class then we have to take an user defined class which must be a subclass to HttpServlet abstract class.

```
public abstract class HttpServlet extends GenericServlet {
  public void service(ServletRequest req, ServletResponse res)throws SE, IOException {
      HttpServletRequest hreq=(HttpServletRequest)req;
      HttpServletResponse hres=(HttpServletResponse)res;
      service(hreq, hres);
  }

  public void service(HttpServletRequest hreq, HttpServletResponse hres)throws SE, IOE {
      String method=hreq.getMethod();
      if(method.equals("GET")) {
          doGet(hreq, hres);
      }
      if(method.equals("POST")) {
          doPost(hreq, hres);
      }
  }
  public void doGet(HttpServletRequest hreq, HttpServletResponse hres)throws SE, IOE
  { }
```

```
    public void doPost(HttpServletRequest hreq, HttpServletResponse hres)throws SE, IOE
    { }
}
public class MyServlet extends HttpServlet
{ ------
  ------ }
```

Note: In case of HttpServlet, we have to override either doGet(_,_) method or doPodt(_,_) method and son on doXxx(_,_) method with our web application logic on the basis of request type which we provide at client browser.

--------------Application by using HttpServlet--------------

httpservletapp:-

web.xml:-

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

HttpDemo.java:-

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HttpDemo extends HttpServlet {

   public HttpDemo() {

   }
```

```
        protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out=response.getWriter();
                out.println("<h1>hello this is http</h1>");
        }

}
```

The main difference between GenericServlet flow of execution and HttpServlet flow of execution is in GenericServlet flow of execution container will execute only one service(_,_) method to process the request, but in case of HttpServlet container will execute service(ServletRequest, Servletresponse), service(HttpServletRequest, HttpServletresponse), doXxx(HttpServletRequest, HttpServletresponse) on the basis of request type in order to process the request.

Q: Is it possible to override service(_,_) method in HttpServlet?

Ans: In HttpServlet, it is possible to override any of the service(_,_) methods, doXxx(_,_) methods, but always it is suggestible to override doXxx(_,_) methods on the basis of request types, it is not at all suggestible to override service_,_) methods.

If we override service(_,_) method in HttpServlet then container will execute user provided service(_,_) method, but container is unable to execute predefined service(_,_) method so that it is not possible to reach doGet(_,_) method or doPost(_,_) method and so on.

Q: Is it possible provide both constructor and init() method in a single servlet?

Ans: Yes, it is possible provide both constructor and init() method in a single servlet.

If we provide a static block, a constructor, init(_) method, doXxx(_,_) method and destroy()method within a single servlet then container will execute static block at the time of servlet loading, constructor at the time of servlet instantiation, init() method at the time of performing servlet initialization, doXxx(_,_) method at the time of request processing and destroy() method at the time of servlet deinstantiation.

Ex: public class MyServlet extends HttpServlet {

```
        static {

           System.out.println("Servlet Loading");

        }

        public MyServlet() {

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

```
    }
    public void init() {
        System.out.println("Servlet Initialization");
    }



  public void doGet(HttpServletRequest hreq, HttpServletResponse hres)throws SE, IOE {
        System.out.println("Request Processing");
    }

    public void destroy() {
        System.out.println("Servlet Deinstantiation");
    }
}
```

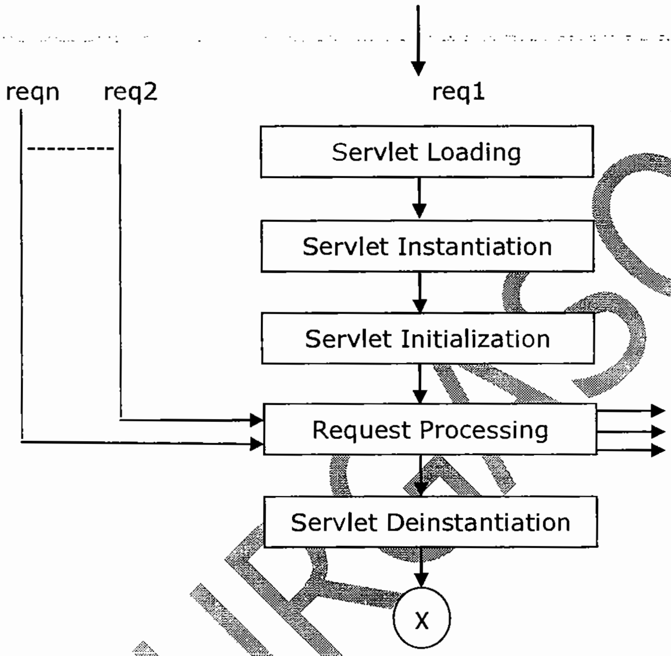
If we send a request to above servlet then are able to see the following output on Server prompt

```
Servlet Loading
Servlet Instantiation
Servlet Initialization
Request Processing
Servlet Deinstantiation (when we close the server).
```

If we want to provide any constructor in servlet class then that constructor should be public and zero argument because container will search and execute public and zero argument constructor as part of servlet instantiation to create servlet object.

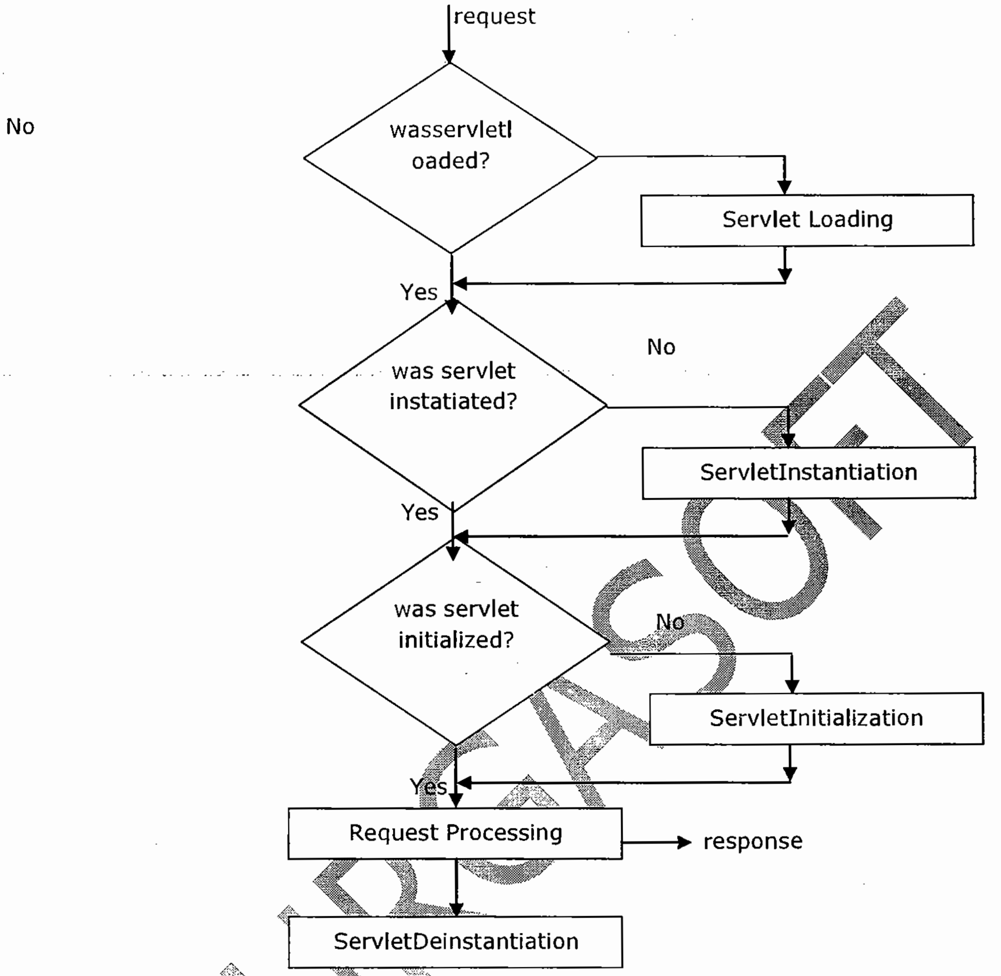
If we provide parameterized constructor without zero-argument constructor then container will raise an Exception like javax.servlet.ServletException : Error-Instantiating Servlet class MyServlet with the root case java.lang.InstantiationException:MyServlet.

Servlet Life Cycle:



From the above representation when we send multiple number of requests to a particular servlet at a time then container will perform Servlet Loading, Instantiation and Initialization only for first request, container will bring all the later request directly to Request processing phase by skipping Servlet Loading, Instantiation and Initialization phases.

If we send multiple number of requests to a particular servlet then container will perform the following work for each and every requests to execute life-cycle stages.



From the above representations all the servlets and Jsp's are able to allow multiple number of requests at a time i.e. multiple number of threads at a time and all the servlets are able to process multiple number of threads at a time without providing data inconsistency. Therefore all the servlets and Jsp's are by default Thread safe.

Note: If any resource is able to allow and process multiple number of threads at a time without having side effects then that resource is called as **Thread Safe Resource.**

In servlet applications, as per the application requirement if we want to allow only one request at a time to a particular servlet then Servlet API has provided javax.servlet.SingleThreadModel marker interface.

```
public class MyServlet extends HttpServlet implements SingleThreadModel

{ ------------------ }
```

Where SingleThreadModel interface is a deprecated interface provided by Servlet API at the initial versions of servlet, which was not supported by the latest versions of servers.

Still, if we want to achieve our requirement about to allow only one request at a time when we have to use synchronization in the respective servlets.

Among the synchronization it is suggestible to use synchronized blocks over synchronized methods in order to improve performance.

```
public class MyServlet extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res)throws SE, IOE

    { ---------

       ---------

       synchronized(this)

       { ---------

          --------- }

    }

}
```

Note : In web applications, always it is suggestible to avoid to use SingleThreadModel and Synchronization because they will decrease the performance of web application.

Q: As part of servlet design if we access destroy() method in init() method then what will be the response from servlet?

Ans: As part of servlet design if we access destroy() method in init() then container will not perform servlet deinstantiation as part of servlet initialization.

Container is a software which includes number of modules to perform Servlet life cycle. When we send a request to the respective servlet from a client then container will start Servlet life cycle by executing its internal modules.

While performing servlet initialization, container will access init() method, init() method is not responsible to perform servlet initialization completely.

Similarly container will perform servlet deinstantiation by executing its internal modules. As part of this, container will access destroy() method.

Due to the above reasons eventhough we access destroy() method in init() method then we are able to access our provided destroy() method just like a normal method, it will not perform servlet deinsatntiation.

Q: If we send GET request from user form but if we override doPost(_,_) in the corresponding servlet what will be response from servlet?

Ans: The main convention of HttpServlet is, if we specify xxx request type t

hen container will execute doXxx(_,_) method i.e. we will override doXxx(_,_) method.

With the above convention, if we specify GET request at user form then we must override doGet(_,_) method.

In the above content, if we override do Post(_,_) method for GET request then container will execute predefined doGet(_,_) method provided by HttpServlet as for the predefined implementation of service(_,_) method.

In HttpServlet, doGet(_,_) method was implemented in such a way to send an error message with the following error description.

HTTP Status 405-HTTP method GET is not supported by this URL

Note: In HttpServlet, by default all doXxx(_,_) methods are implemented with the above convention only.

Note: In case of Tomcat server, if we specify any request type except GET and POST at user form then container will treat that request type has default request type and it will execute doGet(_,_) method.

This type of implementation may not be available with all remaining servers. With the above implementation, Tomcat server has provided support for only GET and POST request types.

User Interface[Forms Design]:

In general, in web applications there are 2 ways to prepare user forms.

1. Static Form Generation
2. Dynamic Form Generation

In case of Static Form Generation, we will prepare user form in the form of html file separately under application folder at the time of designing the application.

In case of Dynamic Form Generation, we will define user form a servlet. If we require Dynamic form then we have to access required respective servlet.

Headersapp:

web.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">
<display-name>headersapp</display-name>
<welcome-file-list>
<welcome-file>index.html</welcome-file>
<welcome-file>index.htm</welcome-file>
<welcome-file>index.jsp</welcome-file>
<welcome-file>default.html</welcome-file>
<welcome-file>default.htm</welcome-file>
<welcome-file>default.jsp</welcome-file>
</welcome-file-list>
<servlet>
<description></description>
<display-name>HeadersServlet</display-name>
<servlet-name>HeadersServlet</servlet-name>
<servlet-class>com.durgasoft.HeadersServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>HeadersServlet</servlet-name>
<url-pattern>/headers</url-pattern>
```

```
</servlet-mapping>

</web-app>
```

HeadersServlet.java

```
package com.durgasoft;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.Enumeration;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;


public class HeadersServlet extends HttpServlet {

        private static final long serialVersionUID = 1L;

        protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

                response.setContentType("text/html");

                PrintWriter out=response.getWriter();

                Enumeration<String> e=request.getHeaderNames();

                out.println("<html>");

                out.println("<body><center><br><br>");

                out.println("<table border='1' bgcolor='lightblue'>");

                out.println("<tr><td align='center'><h3>Header Names</h3></td><td
align='center'><h3>Header Values</h3></td></tr>");

                while(e.hasMoreElements()){

                        String header_Name=(String)e.nextElement();

                        String header_Value=request.getHeader(header_Name);
out.println("<tr><td>"+header_Name+"</td><td>"+header_Value+"</td></tr>");
                }
                out.println("</table></center></body></html>");
        }

}
```

Parametersapp:

registrationform.java

```
----------------------
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<font color="red">
<h2>Durga Software Solutions</h2>
<h3>Student Registration Form</h3>
</font>
<form method="POST" action="./reg">
<table>
<tr>
        <td>Student Id</td>
        <td><input type="text" name="sid"/></td>
</tr>
<tr>
```

```
        <td>Student Name</td>
        <td><input type="text" name="sname"/></td>
</tr>
<tr>
        <td>Student Qualification</td>
        <td>
                <input type="checkbox" value="BSC" name="squal"/>BSC<br>
                <input type="checkbox" value="MCA" name="squal"/>MCA<br>
                <input type="checkbox" value="PHD" name="squal"/>PHD</br>
        </td>
</tr>
<tr>
        <td>Student Gender</td>
        <td>
                <input type="radio" value="Male" name="sgender"/>Male<br>
                <input type="radio" value="Female" name="sgender"/>Female<br>
        </td>
</tr>
<tr>
        <td>Student Technologies</td>
        <td>
                <select size="5" name="stech" multiple="multiple">
                        <option value="C">C</option>
                        <option value="C++">C++</option>
                        <option value="Java">JAVA</option>
                        <option value=".Net">.Net</option>
                        <option value="Oracle">Oracle</option>
```

```
                  <option value="Testing Tools">Testing Tools</option>
              </select>
          </td>
</tr>
<tr>
          <td>Branch</td>
          <td>
              <select name="branch">
                  <option value="Ameerpet">Ameerpet</option>
                  <option value="S R Nagar">S R Nagar</option>
                  <option value="Madapur">Madapur</option>
                  <option value="KPHB">KPHB</option>
              </select>
          </td>
</tr>
<tr>
          <td>Student Address</td>
          <td><textarea rows="10" cols="50" name="saddr"></textarea></td>
</tr>
<tr>
          <td><input type="submit" value="Registration"/></td>
</tr>
</table>
</form>
</body>
</html>
```

web.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">

<display-name>parametersapp</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

<welcome-file>index.jsp</welcome-file>

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

<welcome-file>default.jsp</welcome-file>

</welcome-file-list>

<servlet>

<description></description>

<display-name>RegistrationServlet</display-name>

<servlet-name>RegistrationServlet</servlet-name>

<servlet-class>com.durgasoft.RegistrationServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>RegistrationServlet</servlet-name>

<url-pattern>/reg</url-pattern>

</servlet-mapping>

</web-app>
```

RegistrationServlet.java

```
package com.durgasoft;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class RegistrationServlet extends HttpServlet {

        private static final long serialVersionUID = 1L;


        protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

                response.setContentType("text/html");

                PrintWriter out=response.getWriter();

                String sid=request.getParameter("sid");

                String sname=request.getParameter("sname");

                String[] squal=request.getParameterValues("squal");

                String sgender=request.getParameter("sgender");

                String[] stech=request.getParameterValues("stech");

                String branch=request.getParameter("branch");

                String saddr=request.getParameter("saddr");

                String qual="";

                for(int i=0;i<squal.length;i++){

                        qual=qual+squal[i]+"<br>";

                }
                String tech="";
                for(int j=0;j<stech.length;j++){
                        tech=tech+stech[j]+"<br>";
                }
                out.println("<html>");
                out.println("<body>");
                out.println("<font color='red'>");
                out.println("<h2>Durga Software Solutions</h2>");
                out.println("<h3>Student Registration Details</h3>");
                out.println("</font>");
                out.println("<table border='1'>");
                out.println("<tr><td>Student Id</td><td>"+sid+"</td></tr>");
                out.println("<tr><td>Student Name</td><td>"+sname+"</td></tr>");
                out.println("<tr><td>Student
Qualification</td><td>"+qual+"</td></tr>");
                out.println("<tr><td>Student Gender</td><td>"+sgender+"</td></tr>");
                out.println("<tr><td>Student
Technologies</td><td>"+tech+"</td></tr>");
                out.println("<tr><td>Branch</td><td>"+branch+"</td></tr>");
                out.println("<tr><td>Student Address</td><td>"+saddr+"</td></tr>");
                out.println("</table></body></html>");
        }

}
```

Loginapp:

layout.html

```
<!DOCTYPE html>
<frameset rows="20%,65%,15%">
        <frame src="header.html"/>
        <frameset cols="20%,80%">
                <frame src="menu.html"/>
                <frame src="welcome.html" name="body"/>
        </frameset>
        <frame src="footer.html"/>
</frameset>
```

Header.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body bgcolor="maroon">
<center>
<font color="white" size="7">
<b>
DURGA SOFTWARE SOLUTIONS
</b>
</font>
```

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

menu.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body bgcolor="lightyellow">
<br><br>
<h3>
<a href="loginform.html" target="body">Login</a>
<br><br>
<a href="registrationform.html" target="body">Registration</a>
</h3>
</body>
</html>
```

welcome.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
```

```
<title>Insert title here</title>
</head>
<body bgcolor="lightblue">
<center>
<font color="red" size="6">
<b><br><br><br>
<marquee>
Welcome To Durga Software Solutions
</marquee>
</b>
</font>
</center>

</body>
</html>
```

footer.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body bgcolor="blue">
<center>
<font color="white" size="5">
```

```
<b>
copyright reserved @ Durga Software Solutions, S R Nagar
</b>
</font>
</center>

</body>
</html>
```

loginform.html

```
----------------
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body bgcolor="lightblue">
<br><br>
<form method="POST" action="./login">
<center>
<table>
<tr>
        <td>User Name</td>
        <td><input type="text" name="uname"/></td>
</tr>
<tr>
```

```
        <td>Password</td>
        <td><input type="password" name="upwd"/></td>
</tr>
<tr>
<td><input type="submit" value="Login"/></td>
</tr>
</table>
</center>
</form>
</body>
</html>
registrationform.html
----------------------
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body bgcolor="lightblue">
<br><br>
<form method="POST" action="./reg">
<center>
<table>
<tr>
        <td>User Name</td>
        <td><input type="text" name="uname"/></td>
```

```
</tr>
<tr>
        <td>Password</td>
        <td><input type="password" name="upwd"/></td>
</tr>
<tr>
        <td>User Email</td>
        <td><input type="text" name="uemail"/></td>
</tr>
<tr>
        <td>User Mobile Num</td>
        <td><input type="text" name="umobile"/></td>
</tr>

<tr>
<td><input type="submit" value="Registration"/></td>
</tr>
</table>
</center>
</form>
</body>
</html>
```

web.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">

<display-name>loginapp</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

<welcome-file>index.jsp</welcome-file>

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

<welcome-file>default.jsp</welcome-file>

</welcome-file-list>

<servlet>

<description></description>

<display-name>LoginServlet</display-name>

<servlet-name>LoginServlet</servlet-name>

<servlet-class>com.durgasoft.LoginServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>LoginServlet</servlet-name>

<url-pattern>/login</url-pattern>

</servlet-mapping>

<servlet>

<description></description>
```

```
<display-name>RegistrationServlet</display-name>
<servlet-name>RegistrationServlet</servlet-name>
<servlet-class>com.durgasoft.RegistrationServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>RegistrationServlet</servlet-name>
<url-pattern>/reg</url-pattern>
</servlet-mapping>
</web-app>
```

LoginServlet.html

```
package com.durgasoft;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class LoginServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;

        protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out=response.getWriter();
                String uname=request.getParameter("uname");
```

```
		String upwd=request.getParameter("upwd");
		UserService us=new UserService();
		String status=us.checkLogin(uname,upwd);
		out.println("<html>");
		out.println("<body bgcolor='lightblue'>");
		out.println("<center><br><br>");
		out.println("<font color='red' size='7'>");
		if(status.equals("success")){
			out.println("Login Success");
		}
		if(status.equals("failure")){
			out.println("Login Failure");
		}
		out.println("</font></center></body></html>");

	}

}
```

RegistrationServlet.java

```
package com.durgasoft;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class RegistrationServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;


        protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out=response.getWriter();
                String uname=request.getParameter("uname");
                String upwd=request.getParameter("upwd");
                String uemail=request.getParameter("uemail");
                String umobile=request.getParameter("umobile");
                UserService us=new UserService();
                String status=us.registration(uname,upwd,uemail,umobile);
                out.println("<html>");
                out.println("<body bgcolor='lightblue'>");
                out.println("<center><br><br>");
                out.println("<font color='red' size='7'>");
                if(status.equals("success")){
                        out.println("Registration Success");
                }
                if(status.equals("failure")){
                        out.println("Registration Failure");
                }
                if(status.equals("existed")){
                        out.println("User Existed Already");
                }
                out.println("</font></center></body></html>");


        }


}
```

UserService.java

```
package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class UserService {
        Connection con;
        Statement st;
        ResultSet rs;
        String status="";
        public UserService() {
                try {
                        Class.forName("oracle.jdbc.OracleDriver");

        con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "system",
"durga");
                        st=con.createStatement();
                } catch (Exception e) {
```

```
                e.printStackTrace();
        }
    }
    public String checkLogin(String uname, String upwd){
        try {
                rs=st.executeQuery("select * from reg_Users where
    uname='"+uname+"' and upwd='"+upwd+"'");
                boolean b=rs.next();
                if(b==true){
                        status="success";
                }else{
                        status="failure";
                }
        } catch (Exception e) {

        }
        return status;
    }
    public String registration(String uname, String upwd, String uemail, String umobile){
        try {
                rs=st.executeQuery("select * from reg_Users where
        uname='"+uname+"'");
                boolean b=rs.next();
                if(b==true){
                        status="existed";
                }else{
                        st.executeUpdate("insert into reg_Users
        values('"+uname+"','"+upwd+"','"+uemail+"','"+umobile+"')");
```

```
                    status="success";
                }
            } catch (Exception e) {
                status="failure";
                e.printStackTrace();
            }
            return status;
        }

}
```

Dynamicformapp:

updateform.html

```
-------------------
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<font color="red">
<h2>Durga Software Solutions</h2>
<h3>Student Update Form</h3>
</font>
<form method="GET" action="./edit">
<table>
<tr>
```

```
        <td>Student Id</td>

        <td><input type="text" name="sid"/></td>

</tr>

<tr>

        <td><input type="submit" value="GetEditForm"/></td>

</tr>

</table>

</form>

</body>

</html>
```

web.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">

<display-name>dynamicformapp</display-name>

<welcome-file-list>

<welcome-file>updateform.html</welcome-file>

</welcome-file-list>

<servlet>

<description></description>

<display-name>EditFormServlet</display-name>

<servlet-name>EditFormServlet</servlet-name>

<servlet-class>com.durgasoft.EditFormServlet</servlet-class>

</servlet>
```

```
<servlet-mapping>
<servlet-name>EditFormServlet</servlet-name>
<url-pattern>/edit</url-pattern>
</servlet-mapping>
<servlet>
<description></description>
<display-name>UpdateServlet</display-name>
<servlet-name>UpdateServlet</servlet-name>
<servlet-class>com.durgasoft.UpdateServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>UpdateServlet</servlet-name>
<url-pattern>/update</url-pattern>
</servlet-mapping>
</web-app>
```

EditFormServlet.java

```
package com.durgasoft;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
public class EditFormServlet extends HttpServlet {

        private static final long serialVersionUID = 1L;


        protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

                response.setContentType("text/html");

                PrintWriter out=response.getWriter();

                String sid=request.getParameter("sid");

                StudentService ss=new StudentService();

                StudentTo sto=ss.getStudent(sid);

                if(sto==null){

                        out.println("<html>");

                        out.println("<body>");

                        out.println("<br><br>");

                        out.println("<font color='red' size='6'>");

                        out.println("Student Not Existed");

                        out.println("</font>");

                        out.println("<br><br>");

                       out.println("<h3><a href='./updateform.html'>|Update
                       Form|</a></h3>");

                        out.println("</body></html>");

                }else{

                        out.println("<html>");

                        out.println("<body>");

                        out.println("<font color='red'>");

                        out.println("<h2>Durga Software Solutions</h2>");

                        out.println("<h3>Student Edit Form</h3>");

                        out.println("</font>");
```

```
            out.println("<form method='GET' action='./update'>");
            out.println("<table>");
            out.println("<tr><td>Student Id</td><td>"+sid+"</td></tr>");
            out.println("<input type='hidden' name='sid' value='"+sid+"'/>");
            out.println("<tr><td>Student Name</td><td><input type='text' 
            name='sname' value='"+sto.getSname()+"'/></td></tr>");
            out.println("<tr><td>Student Address</td><td><input type='text' 
            name='saddr' value='"+sto.getSaddr()+"'/></td></tr>");
            out.println("<tr><td><input type='submit' 
            value='Update'/></td></tr>");
            out.println("</table></form></body></html>");
        }
    }
}
```

UpdateServlet.java

```
-------------------
package com.durgasoft;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class EditFormServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
```

```
        protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

                response.setContentType("text/html");

                PrintWriter out=response.getWriter();

                String sid=request.getParameter("sid");

                StudentService ss=new StudentService();

                StudentTo sto=ss.getStudent(sid);

                if(sto==null){

                        out.println("<html>");

                        out.println("<body>");

                        out.println("<br><br>");

                        out.println("<font color='red' size='6'>");

                        out.println("Student Not Existed");

                        out.println("</font>");

                        out.println("<br><br>");

                        out.println("<h3><a href='./updateform.html'>|Update
                        Form|</a></h3>");

                        out.println("</body></html>");

                        }else{

                        out.println("<html>");

                        out.println("<body>");

                        out.println("<font color='red'>");

                        out.println("<h2>Durga Software Solutions</h2>");

                        out.println("<h3>Student Edit Form</h3>");

                        out.println("</font>");

                        out.println("<form method='GET' action='./update'>");

                        out.println("<table>");

                        out.println("<tr><td>Student Id</td><td>"+sid+"</td></tr>");
```

```
                    out.println("<input type='hidden' name='sid' value='"+sid+"'/>");
                    out.println("<tr><td>Student Name</td><td><input type='text'
                    name='sname' value='"+sto.getSname()+"'/></td></tr>");
                    out.println("<tr><td>Student Address</td><td><input type='text'
                    name='saddr' value='"+sto.getSaddr()+"'/></td></tr>");
                    out.println("<tr><td><input type='submit'
                    value='Update'/></td></tr>");
                    out.println("</table></form></body></html>");
            }
        }
}
```

StudentService.java

```
-------------------
package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class StudentService {
        Connection con;
        Statement st;
        ResultSet rs;
        String status="";
        StudentTo sto;
        public StudentService() {
                try {
```

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

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

                st=con.createStatement();

        } catch (Exception e) {

                e.printStackTrace();

        }

}

public StudentTo getStudent(String sid){

        try {

                rs=st.executeQuery("select * from student where sid='"+sid+"'");

                boolean b=rs.next();

                if(b==true){

                        sto=new StudentTo();

                        sto.setSid(rs.getString(1));

                        sto.setSname(rs.getString(2));

                        sto.setSaddr(rs.getString(3));

                }else{

                        sto=null;

                }

        } catch (Exception e) {


        }

        return sto;

}

public String update(String sid, String sname, String saddr){

        try {
```

```
			st.executeUpdate("update student set 
		sname='"+sname+"',saddr='"+saddr+"' where sid='"+sid+"'");
			status="success";
		} catch (Exception e) {
			status="failure";
			e.printStackTrace();
		}
		return status;
	}

}
```

StudentTo.java

```
-----------------
package com.durgasoft;
public class StudentTo {
	private String sid;
	private String sname;
	private String saddr;
	public String getSid() {
		return sid;
	}
	public void setSid(String sid) {
		this.sid = sid;
	}
	public String getSname() {
		return sname;
```

```
        }
        public void setSname(String sname) {
                this.sname = sname;
        }
        public String getSaddr() {
                return saddr;
        }
        public void setSaddr(String saddr) {
                this.saddr = saddr;
        }

}
```

Welcome Files:

In general, in all the web applications some pages like login pages, index pages, home pages and so on are the first pages.

In the above context, to access the first pages we have to specify the respective html page name or jsp page name as resource name in URL eventhough they are common for each and every user.

To overcome the problem, we have to declare the respective html or jsp page as welcome file.

Welcome file is the first page of the web application, it must be executed by the container automatically when we access the respective application without specifying resource name in URL.

To declare welcome file in web.xml file, we have to use the following xml tags.

Ex: <web-app>

```
<welcome-file-list>

<welcome-file>file1</welcome-file>

<welcome-file>file1</welcome-file>
```

</welcome-file-list>

</web-app>

From the above tags representation, it is possible to provide more than one welcome file with in a single web application but w.r.t. multiple no. of modules.

If we provide more than one welcome file with in a single web application w.r.t. modules the container will search for the respective welcome file as per the order in which we configured web.xml file.

Smooth Deployment:

In general, we will prepare web applications with Tomcat server by creating the entire web application directly structure under webapps folder.

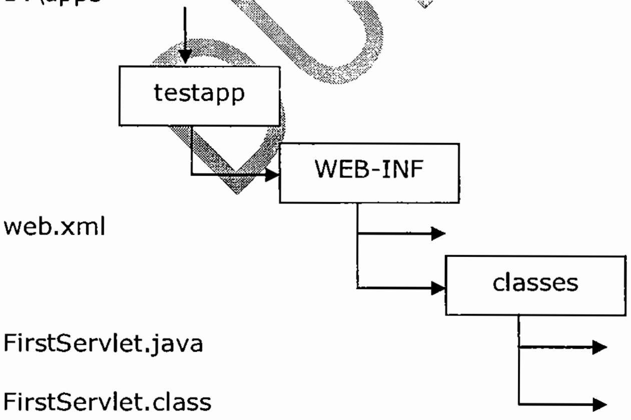
In this case, when we start the server then automatically the prepared web application will be deployed into the server.

The above approach to deploy the web applications is called **Hard Deployment**, it is not suggestible.

To perform Smooth Deployment for web applications we have to use the following steps.

Step 1: Prepare web application separately.

D:\apps



Step 2: Prepare war file for the web application by using the following command.

```
D:\apps\testapp>jar –cvf testapp.war *.*
```

Step 3: Start the Tomcat server and open Manager Applications.

http://localhost:1010/manager/html/

Step 4: Upload war file in order to deploy web application.

Go to war file to deploy section in Tomcat Web Application Manager, select war file by click on Browse button and click on Deploy button.

If we click on Deploy button then automatically uploaded war file will be deployed onto the Tomcat server with war file name as Application Context.

Step 5: Access the application.

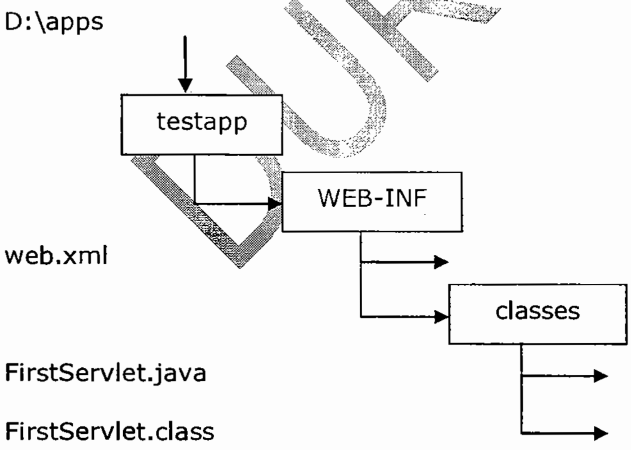
Weblogic Server:

Weblogic Server is an Application server introduced by Bea, it will provide very good middle ware services like JNDA, JTA, Security and so on.

Weblogic_10.3 version is compatible with jdk6 and it able to provide support for servlet2.5 and jsp2.1.

To deploy and execute web applications in Weblogic server we have to use the following steps.

Step 1: Prepare web application and its war file.



In the above application, we have to compile all the servlets under Weblogic environment.

To achieve this, we have to set CLASSPATH environment variable to weblogic.jar file provided by Weblogic server.

```
D:\apps\testapp\WEB-INF\classes>setclasspath=C:\bea\wlserver_10.3\

server\lib\weblogic.jar;

D:\apps\testapp\WEB-INF\classes>javac *.java
```

To prepare war file we have to use the following command on command prompt.

```
D:\apps\testapp>jar –cvf testapp1.war *.*
```

Step 2: Start Weblogic server and open Administration Console.

To start Weblogic server we have to use the following path.

Start
↓
All programs
↓
Oracle Weblogic
↓
Weblogic Server10gR3
↓
Weblogic Server

If we do the above then Weblogic server will start and it will open welcome page of Weblogic server.

To open Administration Console we have to use the following path.

Click on Start the Administration Console button

If we do the above then Weblogic server Administration Console will be open, where we have to provide browser name (weblogic) and password (weblogic) and finally click on login button.

If we click on login button then Weblogic server will open Home page.

Step 3: Deploy and access the web application.

To deploy web application we have to use the following path.

Click on Deployments in Home page
↓

Install
↓
Upload your files
↓
Select war file by click on browse button in Upload a deployment to the admin server
↓
Click on next
↓
Next
↓
Next
↓
Next
↓
Next
↓
Finish
↓
Click on Testing
↓
http://127.0.0.1:7001/testapp1

ServletConfig:

ServletConfig is an object, it will store all the configuration details of a particular servlet, where the configuration details include logical name of the servlet, initialization parameters, reference of ServletContext object and so on.

ServletConfig is an object, it will provide the complete view of a particular servlet.

In web application, container will prepare ServletConfig objects individual to each and every servlet.

In web application execution, container will prepare ServletConfig object immediately after servlet instantiation and just before calling init(_) method in servlet initialization.

Container will destroy the ServletConfig object just before servlet deinstantiation.

Due to the above reasons, the life cycle of ServletConfig object is upto a particular servlet.

If we declare any data in ServletConfig object then that data will be shared upto the respective servlet.

Due to the above reason, the scope of ServletConfig object is upto a particular servlet.

In web applications, ServletConfig object will allow only parameters data, it will not allow attributes data.

In web applications, there are 2 ways to get ServletConfig object .

1. Use getServletConfig() method from Servlet interface

Ex: ServletConfig config=getServletConfig();

2. Override init(_) method

```
 Ex: public class MyServlet extends HttpServlet {

                 ServletConfig config;

 public void init(ServletConfig config){

this.config=config;

}

-------------

}
```

To get logical name of the servlet from its ServletConfig object we have to use the following method.

public String getServletName()

Ex: String lname=config.getServletName();

If we want to provide initialization parameters inServletConfig object then first we have to declare them in web.xml file.

To declare initialization parameters in web.xml file we have to use the following xml tags.

```
<web-app>

<servlet>

<init-param>

<param-name>name</param-name>

<param-value>value</param-value>

</init-param>

            -------------

</servlet>

        -------------

</web-app>
```

If we declare initialization parameters with the above approach then container will read them and store onto ServletConfig object at the time of creation when it receives request from the client.

To get a particular initialization parameter from ServletConfig object we have to use the following method.

public String getInitParameter (String name)

Ex: String a=config.getInitParameter("a");

To get all the initialization parameters from ServletConfig object we have to use the following method.

public Enumeration getInitParameterNames()

Ex: Enumeration e=config.getInitParameterNames();

--------------Application by using ServletConfig---------------

configapp:-

web.xml:

```
<web-app>
<servlet>
<servlet-name>MyServlet</servlet-name>
<servlet-class>MyServlet</servlet-class>
<init-param>
<param-name>driver</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</init-param>
<init-param>
<param-name>url</param-name>
<param-value>jdbc:odbc:sri</param-value>
</init-param>
<init-param>
<param-name>user</param-name>
<param-value>system</param-value>
</init-param>

<init-param>
<param-name>password</param-name>
<param-value>durga</param-value>
</init-param>
</servlet>
<servlet-mapping>
<servlet-name>MyServlet</servlet-name>
<url-pattern>/config</url-pattern>
</servlet-mapping>
</web-app>
```

MyServlet.java:

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

publicclass MyServlet extends HttpServlet {

      protectedvoid doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
            response.setContentType("text/html");
            PrintWriter out=response.getWriter();
            ServletConfig config=getServletConfig();
            String logicalName=config.getServletName();
            String driver=config.getInitParameter("driver");
            String url=config.getInitParameter("url");
            String user=config.getInitParameter("user");
            String password=config.getInitParameter("password");
            out.println("<html><body><h1>");
            out.println("Logical Name: "+logicalName+"<br><br>");
            out.println("Driver      : "+driver+"<br><br>");
            out.println("Url         : "+url+"<br><br>");
            out.println("User        : "+user+"<br><br>");
            out.println("Password    : "+password+"<br><br>");
            out.println("</h1></body></html>");
      }
}
```

ServletContext:

Q: What are the differences between ServletConfig and ServletContext?

1. ServletConfig is an object, it will manage all the configuration details of a particular servlet, where the configuration details include logical name of the servlet, initialization parameters and so on.

ServletContext is an object, it will manage all the context details of a particular web application, where the context details include logical name of web application and context parameters and so on.

2. ServletConfig is an object, it will provide the complete view of a particular servlet.

ServletContext is an object, it will provide the complete view of particular web application.

3. ServletConfig object will be prepared by the container immediately after servlet instantiation and just before calling init(_) method in servlet initialization.

ServletContext object will be prepared by the container the moment when we start the server i.e. the time when we deploy the web application.

4. ServletConfig object will be destroyed by the container just before servlet deinstaniation.

ServletContext object will be destroyed by the container when we shutdown the server i.e. the time when we undeploy the web application.

5. Due to the above reasons, the life of ServletConfig object is almost all the life of the respective servlet object.

The life of ServletContext object is almost all the life of the respective web application.

6. If we declare any data in ServletConfig object then that data will be shared upto respective servlet.

If we declare any data inServletContext object then that data will be shared to all the no. of resources which are available in the present web application.

7. Due to the above reason, ServletConfig object will provide less sharability where as ServletContext object will provide more sharability.

8. In web applications, container will prepare ServletConfig object when it receives the request from client only except in load-on-startup case.

In web applications, container will prepare ServletContext object irrespective of getting request from client.

9. In web applications, ServletConfig object will allow only parameters data but ServletContext object will allow both parameters and attributes data.

10. Due to the above reason, ServletConfig object will allow only Static Inclusion of data where as ServletContext object will allow both Static Inclusion and Dynamic Inclusion of data.

To get the ServletContext object we have to use the following method from ServletConfig.

```
public ServletContext getServletContext();
```

Ex: ServletContext context=config.getServletContext();

Note: In servlet3.0 version, it is possible to get ServletContext object even from ServletRequest.

Ex: ServletContext context=req.getServletContext();

If we want to get the logical name of the web application from ServletContext object first of all we have to declare it in web.xml file.

To declare a logical name in web.xml file we have to use the following xml tag.

```
<web-app>

          -------------

<display-name>logical_name</display-name>

          -------------

</web-app>
```

To get the logical name of web application from ServletContext object we will use the following method.

Public String getServletContextName()

Ex: String lname=context.getServletContextName();

If we want to provide context parameters on ServletContext object first we have to declare them in web.xml file.

To declare a context parameter in web.xml file we have to use the following xml files.

```
<web-app>

         ------------

<context-param>

<param-name>name</param-name>

<param-value>value</param-value>

</context -param>

-------------

</web-app>
```

When we start the server or at the time of application deployment the main job of the container is to recognize the web application and to prepare ServletContext object.

At the time of recognizing the application container will recognize web.xml file then perform web.xml file loading, parsing and reading the content.

While reading the content container will read all its context parameters and store them in ServletContext object at the time of creation.

To get the particular context parameter value from ServletContext object we will use the following method.

public String getInitParameter(String name)

Ex: String value=context.getInitParameter("name");

To get all the context parameter names from ServletContext object we will use the following method.

public Enumeration getInitParameterNames()

Ex: Enumeration e=context.getInitParameterNames();

In web application, ServletContext object is able to allow attributes.

To set an attribute on ServletContext object we will use the following method.

public void setAttribute(String name, Object value)

Ex: context.setAttribute("location", "Hyd");

To get an attribute from ServletContext object we will use the following method.

public Object getAttribute(String name)

Ex: String location=(String)context.getAttribute("location");

To remove an attribute from ServletContext object we will use the following method.

public void removeAttribute(String name)

Ex: context.removeAttribute("location");

To get all the names of attributes from ServletContext object we will use the following method.

public Enumeration getAttributeNames()

Ex: Enumeration e=context.getAttributeNames();

Q: What is ForeignContext?

Ans: ForeignContext is a ServletContext object of another web application being executed in the same server.

To get ForeignContext object we have to use the following method.

public ServletContext getContext(String path)

--------------Application by using ServletContext---------------

contextapp:-

web.xml:

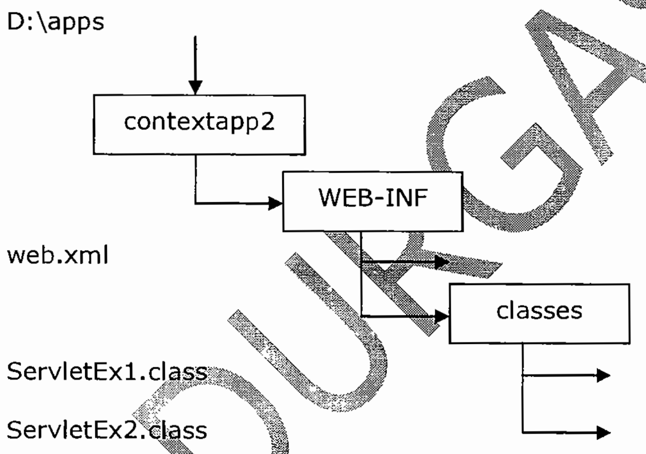
```
<web-app>
<display-name>Context Application</display-name>
<context-param>
        <param-name>a</param-name>
        <param-value>apple</param-value>
</context-param>
<context-param>
        <param-name>b</param-name>
        <param-value>bombay</param-value>
</context-param>
<servlet>
<servlet-name>MyServlet</servlet-name>
<servlet-class>MyServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>MyServlet</servlet-name>
<url-pattern>/context</url-pattern>
</servlet-mapping>
</web-app>
```

MyServlet.java:

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

publicclass MyServlet extends HttpServlet {

        protectedvoid doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out=response.getWriter();


                ServletContext context=getServletConfig().getServletContext();
                String logicalName=context.getServletContextName();
                String a=context.getInitParameter("a");
                String b=context.getInitParameter("b");
                Enumeration e=context.getInitParameterNames();
                context.setAttribute("c", "cat");
                context.setAttribute("d", "dog");
                out.println("<html><body><h1><br>");
```

```
            out.println("Logical Name : "+logicalName);
            out.println("<br>");
            out.println("a for ... "+a);
            out.println("<br>");
            out.println("b for ... "+b);
            out.println("<br>");
            while(e.hasMoreElements()){
                    out.println(e.nextElement()+"<br>");
            }
            out.println("c for ... "+context.getAttribute("c"));
            out.println("<br>");
            out.println("d for ... "+context.getAttribute("d")+"<br>");
            e=context.getAttributeNames();
            while(e.hasMoreElements()){
                    out.println(e.nextElement()+"<br>");
            }
            out.println("</h1></body></html>");
        }
}
```

Q: Consider the following web application



web.xml:

A short representation of web.xml

a → aaa

b → bbb Here a,b are context parameters

ServletEx1

c ccc →

d ddd → Here c,d are initialization parameters

```
/first
   ServletEx2
e       eee ——→
f       fff ——Here e,f are initialization parameters
/second
```

ServletEx1.java:

```
// import statements
public class ServletEx1 extends HttpServlet{
    public void deGet(HSR req, HSR res)throws SE, IOE {
ServletConfig config=getServletConfig();
 ServletContext context=req.getServletContext();
out.println(“a-→”+context.getInitParameter(“a”));
out.println(“b-→”+context.getInitParameter(“b”));
 out.println(“c-→”+config.getInitParameter(“c”));
  out.println(“d-→”+config.getInitParameter(“d”));
 out.println(“e-→”+config.getInitParameter(“e”));
 out.println(“f-→”+config.getInitParameter(“f”));
}
}
```

ServletEx2.java:

```
// import statements
public class ServletEx2 extends HttpServlet{
 public void deGet(HSR req, HSR res)throws SE, IOE{
ServletConfig config=getServletConfig();
 ServletContext context=req.getServletContext();
 out.println("a-→"+context.getInitParameter("a"));
out.println("b-→"+context.getInitParameter("b"));
 out.println("c-→"+config.getInitParameter("c"));
out.println("d-→"+config.getInitParameter("d"));
 out.println("e-→"+config.getInitParameter("e"));
out.println("f-→"+config.getInitParameter("f"));
}
}
```

Q1: http://localhost:8080/contextapp2/first?

```
Ans:  a-→aaa     c-→ccc     e-→null
       b-→bbb     d-→ddd     f-→null
```

Q2: http://localhost:8080/contextapp2/second?

```
Ans:  a-→aaa     c-→null     e-→eee
       b-→bbb     d-→ null    f-→fff
```

In above web application, we can differentiate the scope of ServletConfig and ServletContext objects i.e. the scope of ServletConfig object is upto the respective servlet where as the scope of ServletContext object is through out the web application.

JBoss Server:

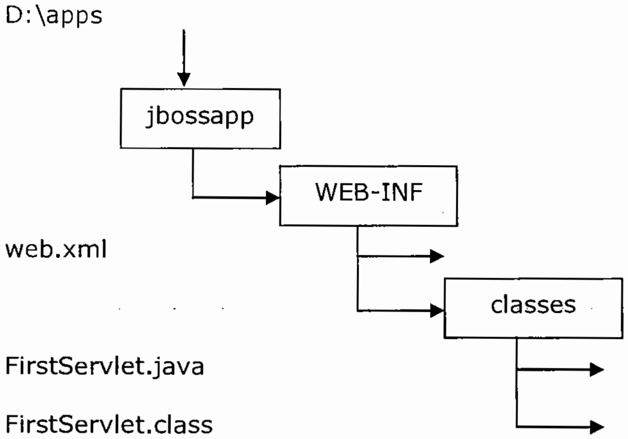
JBoss is an Application Server, it will provide almost all the middle ware services what application servers are provided in general.

JBoss Server is compatible with jdk7 and it able to provide support for servlet3.0 version, jsp2.1 version and so on.

JBoss server is not having its own web container, it was used Tomcat Container.

If we want to deploy and execute web applications with JBoss server we have to use the following steps.

Step 1: Prepare web application and its war file.



In case of JBoss7 to compile all the servlets we need to set CLASSPATH environment variable to jboss-Servlet API_3.0_spec-1.0.0.Final.jar, which has provided by JBoss server in the following location.

```
C:\jboss-as-7.1.0.Final\modules\javax\servlet\api\main\ jboss-servlet-api_3.0_spec-1.0.0.Final.jar
```

```
D:\apps\jbossapp\WEB-INF\classes>set classpath=C:\jboss-as-  7.1.0.Final
\modules\javax\servlet\api\main\ jboss-Servlet API_3.0_spec-1.0.0.Final.jar;
D:\apps\jbossapp\WEB-INF\classes>javac *.java
```

To prepare war file we have to use the following command on command prompt.

```
D:\apps\jbossapp>jar -cvf  jbossapp1.war *.*
```

Step 2: Start JBoss Application Server.

C:\jboss-as-7.1.0.Final\bin, where double click on standalone.bat file

Step 3: Open Administration Console.

To open Administration Console we have to use the following URL on browser.

http://localhost:8888

Actually JBoss port number is 8080, but it was changed to 8888.

If we use the above URL then we are able to get JBoss Server Welcome page, where click on Administration Console and provide username(admin) and password(durga) in security window.

Step 4: Deploy web application.

If we click on OK button in security window automatically JBoss Home page will between open, where click on Manage Deployments under Deployments section.

If we do the above list of deployed applications will be displayed, where to deploy a new web application we have to use following path.

Click on Add Content button
↓
Browse
↓
Select war file
↓
Next
↓
Save
↓
Enable respective
↓
Confirm

Step 5: Access the web application.

Open another window, where we have to provide the following URL.

http://localhost:8888/jbossapp1/first

To change JBoss port number we have to use the following path.

C:\jboss-as-7.1.0.Final\standalone\configuration\ standalone.xml,

where search for 8080 and replace our port number(8888).

<select-binding name="http" port="8888"/>

To create an account in JBoss server we have to use the following path.

C:\jboss-as-7.1.0.Final\bin, where double click on add-user.bat file.

If we do the above then a command prompt will be open, where we have to provide the required details.

1. User type : Application User, press enter
2. Management realm : press enter

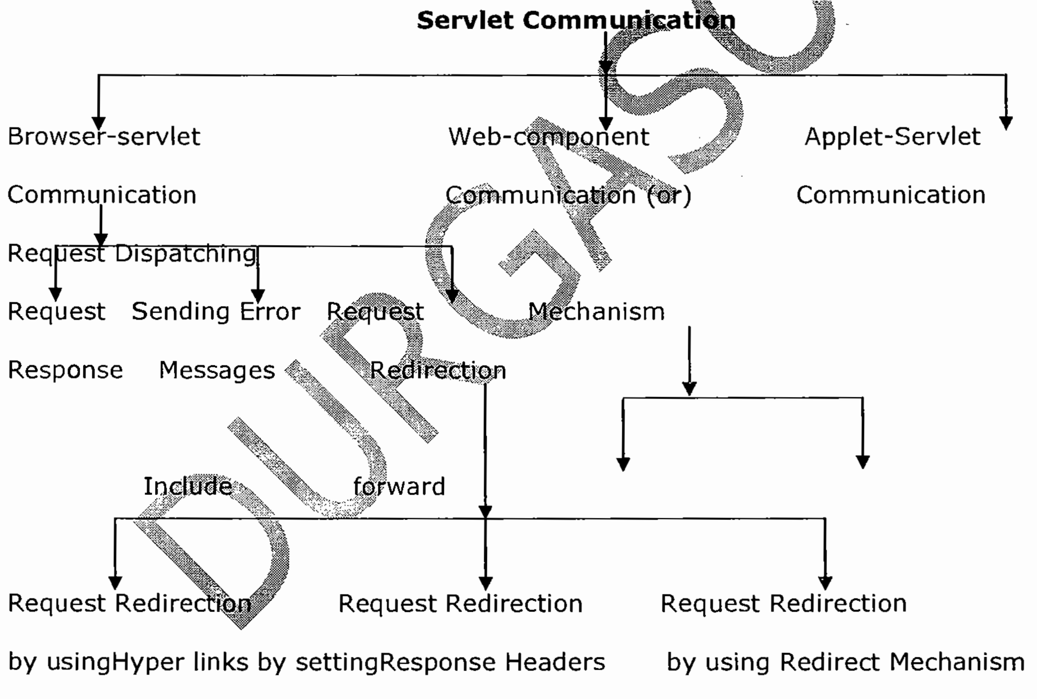
3. Username : durga, press enter
4. Password : durga, press enter
5. Re-enter password : durga, press enter
6. Is this correct yes/no ? yes press enter

Servlet Communication:

In general in web application deployment is not at all suggestible to provide the complete application logic with in a single web resource, it is suggestible to distribute the complete application logic over multiple web resources.

In the above context, to execute the application we must require communication between all the web resources, for this we have to use Servlet Communication.

In web application, we are able to provide servlet communication in the following 3 ways.


Servlet Communication
Browser-servlet Communication
Web-component Communication (or)
Applet-Servlet Communication
Request Dispatching
Request Response
Sending Error Messages
Request Redirection
Mechanism
Include
forward
Request Redirection by usingHyper links
by settingResponse Headers
Request Redirection
Request Redirection
by using Redirect Mechanism

1. Browser-Servlet Communication:

In general in web applications, we will use browser as a client at client machine, from the browser we will send a request to a servlet available at server, where the servlet will be executed and generate some response to the client browser.

In the above process, we have provided communication between client browser and servlet, so that sending a normal request from client to server and getting a normal response from server to client is an examplefor **Browser-Servlet Communication**.

2. Sending Error Messages:

As part of the web application execution, if the container identify any exception or error then container will send the respective error message to be client in its standalone template.

As part of our application, if we want to send our own messages to the client in the container defined template we have to use the following method from response.

```
public void sendError(int statuscode, String description)
```

where statuscode may be 5xx.

SendErrorApp:

Registrationform.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<font color='red'>
<h2>Durga Consultency Services</h2>
<h3>User Registration Form</h3>
</font>
<form method="POST" action="./reg">
```

```
<table>

<tr>

        <td>User Name</td>

        <td><input type="text" name="uname"/></td>

</tr>

<tr>

        <td>User Age</td>

        <td><input type="text" name="uage"/></td>

</tr>

<tr>

        <td>User Email</td>

        <td><input type="text" name="uemail"/></td>

</tr>

<tr>

        <td>User Mobile</td>

        <td><input type="text" name="umobile"/></td>

</tr>

<tr>

        <td><input type="submit" value="Registration"/></td>

</tr>



</table>

</form>

</body>

</html>
```

Web.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">

<display-name>senderrorapp</display-name>

<welcome-file-list>

<welcome-file>registrationform.html</welcome-file>

</welcome-file-list>

<servlet>

<description></description>

<display-name>RegistrationServlet</display-name>

<servlet-name>RegistrationServlet</servlet-name>

<servlet-class>com.durgasoft.RegistrationServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>RegistrationServlet</servlet-name>

<url-pattern>/reg</url-pattern>

</servlet-mapping>

</web-app>
```

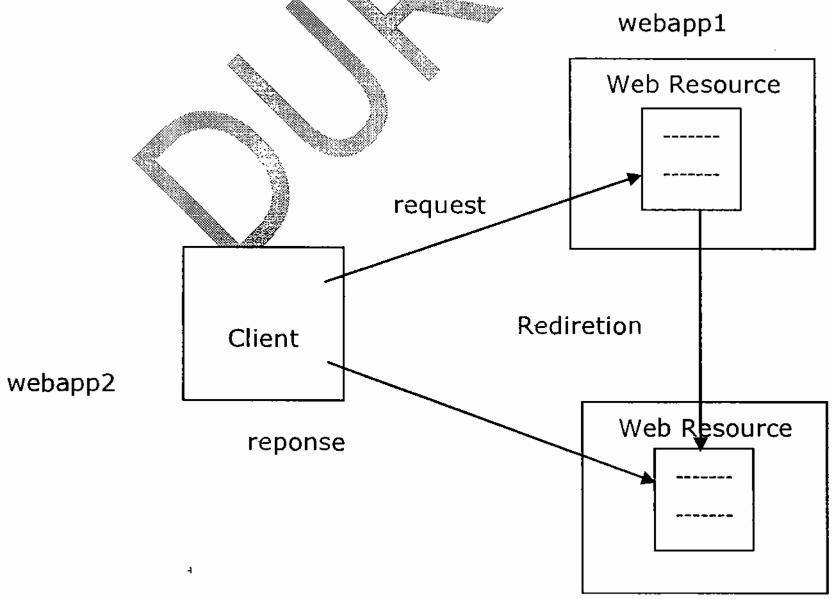
RegistrationServlet.java

```
package com.durgasoft;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class RegistrationServlet extends HttpServlet {

        private static final long serialVersionUID = 1L;

        protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

                try {

                        response.setContentType("text/html");

                        PrintWriter out=response.getWriter();

                        String uname=request.getParameter("uname");

                        int uage=Integer.parseInt(request.getParameter("uage"));

                        String uemail=request.getParameter("uemail");

                        String umobile=request.getParameter("umobile");


                        if(uage<18 || uage>30){

                        response.sendError(504, "User Age Is Not Sufficient for this
                        Recruitment");

                        }else{

                                out.println("<html>");

                                out.println("<body>");

                                out.println("<font color='red'>");

                                out.println("<h2>Durga Consultency Services</h2>");

                                out.println("<h3>User Registration Details</h3>");

                                out.println("</font>");

                                out.println("<table border='1'>");

                                out.println("<tr><td>User
                                Name</td><td>"+uname+"</td></tr>");

                                out.println("<tr><td>User
                                Age</td><td>"+uage+"</td></tr>");
```

```
                    out.println("<tr><td>User
                    Email</td><td>"+uemail+"</td></tr>");

                    out.println("<tr><td>User
                    Mobile</td><td>"+umobile+"</td></tr>");

                    out.println("</table></body></html>");

                }

            } catch (Exception e) {

                e.printStackTrace();

            }

        }


}
```

3. Request Redirection:

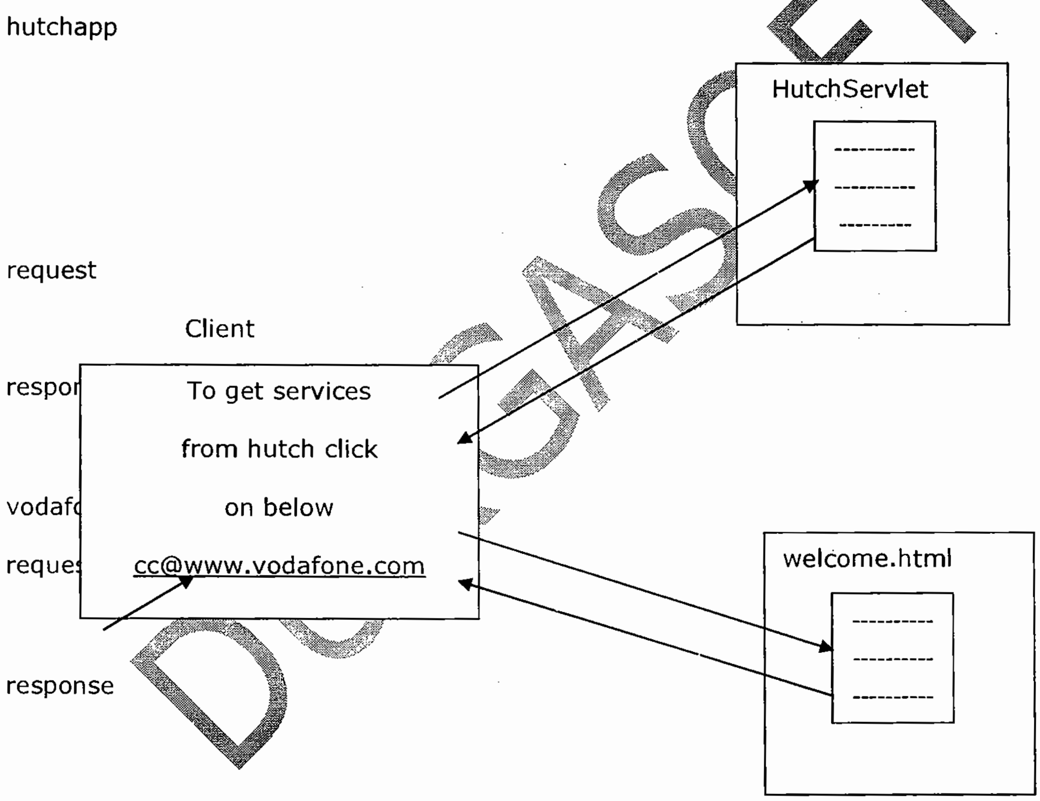
The process of bypassing the request from one web application to another web application is called as **Request Redirection.**



In web applications, we are able to achieve Request Redirection in the following 3 ways.

1. Request Redirection by using Hyper links
2. Request Redirection by setting Response Headers
3. Request Redirection by using Send Redirect Mechanism

1. Request Redirection by using Hyper links:



In this mechanism, when we send a request to first application some resources will be executed and generated an hyper link at client browser as a response.

By getting hyper link at client browser we have to click on it and we have to send another request to the new web application.

By executing some resources at new web application the required response will be generated to client browser.

Drawback:

In this Request Redirection mechanism, user may or may not click the generated hyper link at client browser after sending first request. So that this mechanism won't provide guarantee to achieve Request Redirection.

2. Request Redirection by setting Response Headers:

In this mechanism, first we will send a request to first web application, where first web application will set Redirectional Status Code to Status Line field and new web application URl to Location Response Header.

When the Response Format reached to the client then client will pick up Redirectional status code value from Status Line field, with this client browser will pick up Location Response Header value i.e. new web application URL then client browser will send a new request to new web application.

By executing some resources at new web application the required response will be generated at client machine.

To represent Request Redirection HttpServletResponse has introduced the following 2 constants.

1. public static final int SC_MOVED_TEMPORARILY;
2. public static final int SC_MOVED_PERMANENTLY;

To set a particular status code value to Response Header we will use the following method.

public void setStatus(int statuscode)

To set a particular Response Header value in Response Format we have to use the following method.

public void setHeader(String header_name, String value)

Drawback:

To perform Request Redirection, if we use this approach then every time we have to set Redirectional status code and new web application URL to Location Response Header.

3. Request Redirection by using Send Redirect Mechanism:

To perform Request Redirection, If we use Send Redirect Mechanism no need to use Hyper links, not required to set status code and Response Header values to the Response Format, but We need to use the following method.

public void sendRedirect(String url)

sendredirectapp:-

web.xml:

```
<web-app>
<display-name>sendredirectapp</display-name>
<servlet>
<servlet-name>HutchServlet</servlet-name>
<servlet-class>HutchServlet</servlet-class>
</servlet>


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

HutchServlet.java:

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

publicclass HutchServlet extends HttpServlet {

        protectedvoid doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                response.sendRedirect("http://localhost:2020/vodaphone/welcome.html");
        }

}
```

vodaphoneapp:-

welcome.html:

```
<html>
<bodybgcolor="red">
        <center><b><fontsize="7"color="black">
        <br><br>
        Welcome to Vodaphone
        </font></b></center>
</body>
</html>
```

2. Web-Component Communication:

The process of providing communication between more than one web component available at server machine is called as **Web-Component Communication.**

In general, web-component communication is available in between Servlet-Servlet, Servlet-Jsp, Servlet-HTML, Jsp-Jsp, Jsp-Servlet, Jsp-HTML and so on.

In web applications, we are able to achieve web-component communication in the following 2 ways.

1. Include Mechanism
2. Forward Mechanism

If we want to perform the above mechanisms internally we must use RequestDispatcher object. So that both include and forward mechanisms are commonly called as Request Dispatching Mechanisms.

To achieve web-component communication in web applications we have to use the following 2 steps

Step 1: Get RequestDispatcher object.

Step 2: Apply either include or forward mechanism by using the respective methods.

Step 1: RequestDispatcher object:

RequestDispatcher is an object, it will provide very good environment either to include the target resource response into the present resource response or to forward request from present resource to the target resource.

To get RequestDispatcher object we will use the following 2 ways.

1. ServletContext

    1. getRequestDispatcher(_) method

2. gteNamedDispatcher(_) method

2. ServletRequest

1. getRequestDispatcher(_) method

Q: What is the difference between getRequestDispatcher(_) method and getNamedDispatcher(_) method from ServletContext ?

Ans: Both the methods are used to get the RequestDispatcher object.

To get RequestDispatcher object, if we use getRequestDispatcher(_) method then we should pass the locator of target resource as parameter.

Note : In case of the servlet, url pattern is treated as locator.

public RequestDispatcher getRequestDispatcher(String path)

To get RequestDispatcher object, if we use getNamedDispatcher(_) method then we have to pass logical name of target resource as parameter.

Note : In caseof the servlet, logical name is a name specified along with <servlet-name> tag in web.xml file.

public RequestDispatcher getNamedDispatcher(String path)

Q: What is the difference between getRequestDispatcher(_) method ServletContext and ServletRequest?

Ans: Both the methods can be used to get the RequestDispatcher object.

To get RequestDispatcher object, if we use getRequestDispatcher(_) method from ServletContext then we must pass the relative path of target resource.

To get RequestDispatcher object, if we use getRequestDispatcher(_) method from ServletRequest then we have to pass either relative path or absolute path of target resource.

Note: In general, relative path should prefix with forward slash("/") and absolute path should not prefix with forward slash("/").

Step 2: Apply either Include mechanism or Forward mechanism to achieve Web-Component Communication:

To represent Include and Forward mechanisms RequestDispatcher has provided the following methods.

public void include(ServletRequest req, ServletResponse res)throws SE, IOE

public void forward(ServletRequest req, ServletResponse res)throws SE, IOE

Q: What are the differences between Include and Forward mechanisms?

Ans: In web applications, Include Request Dispatching mechanism can be used to include the target resource response into the present resource response.

In case of Include mechanism, when we send a request to first resource then container will prepare request and response objects, by executing some part in first resource container may generate some response in response object.

When container encounter include(_,_) method then container will bypass the request and response objects to target resource along with flow of execution without refreshing response object.

By executing the target resource some response will be generated in response object, at the end of target resource container will bypass request and response objects back to the first resource to continue its further execution.

In the above context, container will execute remaining content in first resource , some response will be added to response object, at the end of first resource container will dispatch ovaerall response to client.

Therefore, In case of Include mechanism, client is able to receive all resources which are participated in the present request processing.

In web applications, the main purpose of Forward mechanism is to forward request from present resource to target resource.

In case of Forward mechanism, when we send a request to first resource then container will create request and response objects, by executing some part in first resource container may generate some response in response object.

When container encounter forward(_,_) method then container will bypass the request and response objects to the target resource along with flow of execution by refreshing response object (by eliminating previous content in response object).

By executing the target resource some response will be generated in response object, at the end of target resource container will dispatch ovaerall response to client directly without moving back to the first resource.

Therefore, In case of Forward mechanism, client is able to receive only the target resource response which has included in the present request processing.

Q: What is Servlet Chaining?

Ans: The processofincluding more than one servlet in order to process a single request is called as **Servlet Chainingor** **Servlet Collaboration.**

Q:What are the differences between Forward mechanism and send Redirect mechanism?

Ans: 1. In web applications, Forward Request Dispatching mechanism can be used to provide the communication between two resources which must be available at same server.

In web applications, Send Redirect mechanism can be used to provide the communication between two resources which may be available at same server or at two different servers.

2. In case of Forward mechanism, one request is sufficient to establish the communication between two resources.

In case of Send Redirect mechanism, we need requests to establish the communication between two web resources.

includeapp:-

addform.html:

```
<html>
<bodybgcolor="lightgreen">
<center>
<formmethod="get"action="./add">
<br><br>
<tablebgcolor="lightyellow">
<tr><tdcolspan="2"><center><b><fontsize="6"color="red"><u>Product Add
Form</u></font></b></center></td></tr>
<tr><td>Product Id</td><td><inputtype="text"name="pid"/></td>
</tr>


<tr>
<td>Product Name</td>
<td><inputtype="text"name="pname"/></td>
</tr>
<tr>
<td>Product Cost</td>
<td><inputtype="text"name="pcost"/></td>
</tr>
<tr>
<td><inputtype="submit"value="ADD"/></td>
</tr>
</table></form></center></body></html>
```

web.xml:

```
<web-app>
<display-name>includeapp</display-name>
<welcome-file-list>
<welcome-file>addform.html</welcome-file>
</welcome-file-list>
<servlet>
<description></description>
<display-name>AddServlet</display-name>
```

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

Product.java:

```
publicclassProduct {
        private String pid;
        private String pname;
        privateintpcost;
        public String getPid() {
                returnpid;
        }
        publicvoid setPid(String pid) {
                this.pid = pid;
        }
        public String getPname() {
                returnpname;
        }
        publicvoid setPname(String pname) {
                this.pname = pname;
        }



publicint getPcost() {
                returnpcost;
        }
        publicvoid setPcost(int pcost) {
                this.pcost = pcost;
        }
}
```

ProductDao.java:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;

publicclass ProductDao {
        Connection con;
        Statement st;
        ResultSet rs;
        ArrayList<Product>al;
        ProductDao(){
                try {
```

```
            Class.forName("oracle.jdbc.driver.OracleDriver");
    con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "system",
"durga");
            st=con.createStatement();
            al=new ArrayList<Product>();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    publicvoid add(String pid, String pname, int pcost){
        try {
            st.executeUpdate("insert into product
values('"+pid+"','"+pname+"',"+pcost+")");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public ArrayList<Product> getProducts(){
        try {
            rs=st.executeQuery("select * from product");
            while(rs.next()){
                Product p=new Product();
                p.setPid(rs.getString(1));
                p.setPname(rs.getString(2));
                p.setPcost(rs.getInt(3));
                al.add(p);
            }
        } catch (Exception e) {


        e.printStackTrace();
        }
        returnal;
    }
}
```

AddServlet.java:

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

publicclass AddServlet extends HttpServlet {

    protectedvoid doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        try {
```

```
                        response.setContentType("text/html");
                        PrintWriter out=response.getWriter();
                        String pid=request.getParameter("pid");
                        String pname=request.getParameter("pname");
                        int pcost=Integer.parseInt(request.getParameter("pcost"));
                        ProductDao pd=new ProductDao();
                        pd.add(pid,pname,pcost);
                        ArrayList<Product> prds=pd.getProducts();
                        out.println("<html><body><center><br><br>");
                        out.println("<table border='1' bgcolor='lightyellow'>");
                out.println("<tr><td>PID</td><td>PNAME</td><td>PCOST</td></tr>");
                        for(Object o : prds){
                                Product p=(Product)o;
        out.println("<tr><td>"+p.getPid()+"</td><td>"+p.getPname()+"</td><td>"+p.g
etPcost()+"</td></tr>");
                        }
                        out.println("</table></center><br><hr><br></body></html>");

        RequestDispatcherrd=request.getRequestDispatcher("/addform.html");
                                rd.include(request, response);
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

forwardapp:-

layout.html:

```
<html>
<framesetrows="20%,65%,15%">
<framesrc="header.html"/>
<framesetcols="20%,80%">
<framesrc="menu.html"/>
<framesrc="welcome.html"name="body"/>
</frameset>
<framesrc="footer.html"/>
</frameset>
</html>
```

Header.html:

```
<html><bodybgcolor="red">
<center><b><fontsize="7"color="white">
    Durga Software Solutions
</font></b></center>
</body></html>
```

menu.html:

```
<html>
<bodybgcolor="cyan"><br><br><center><b><fontsize="6">
<ahref="./addform.html"target="body">Add</a><br><br>
<ahref="./searchform.html"target="body">Search</a> <br><br>
<ahref="./updateform.html"target="body">Update</a><br><br>
<ahref="./deleteform.html"target="body">Delete</a>
</font></b></center></body>
</html>
```

footer.html:

```
<html><bodybgcolor="blue">
<center><b><fontsize="6"color="white">
copyrights2010-2020@www.durgasoft.com
</font></b></center></body></html>
```

welcome.html:

```
<html><bodybgcolor="lightyellow">
<center><b><fontsize="7"color="red">
<br><br><marquee>
Welcome to Durga Software Solutions
</marquee>
</font></b></center></body></html>
```

addform.html:

```
<html>
<bodybgcolor="lightgreen">
<b><fontsize="7">
<br>
<formmethod="get"action="./add">
<pre>
            Student Id          <inputtype="text"name="sid"/>

            Student Name        <inputtype="text"name="sname"/>

            Student Marks       <inputtype="text"name="smarks"/>

            <inputtype="submit"value="Add"/>
</pre></form></font></b></body></html>
```

searchform.html:

```
<html>
<bodybgcolor="lightgreen">
<b><fontsize="7"><br><br>
```

```
<formmethod="get"action="./search">
<pre>
               Student Id              <inputtype="text"name="sid"/>

               <inputtype="submit"value="Search"/>
</pre></form></font></b></body></html>
```

updateform.html:

```
<html>
<bodybgcolor="lightgreen">
<b><fontsize="7"><br><br>
<formmethod="get"action="./edit">
<pre>
               Student Id              <inputtype="text"name="sid"/>

               <inputtype="submit"value="GetEditForm"/>
</pre></form></font></b></body></html>
```

deleteform.html:

```
<html>
<bodybgcolor="lightgreen">
<b><fontsize="7">
<br><br>
<formmethod="get"action="./delete">
<pre>

               Student Id              <inputtype="text"name="sid"/>


<inputtype="submit"value="Delete"/>
</pre></form></font></b></body></html>
```

success.html:

```
<html>
<bodybgcolor="lightyellow">
<center><b><fontsize="7"color="red">
<br><br>
Success
</font></b></center></body></html>
```

failure.html:

```
<html>
<bodybgcolor="lightyellow">
<center><b><fontsize="7"color="red">
<br><br>
Failure
</font></b></center></body></html>
```

existed.html:

```
<html>
<bodybgcolor="lightyellow">
<center><b><fontsize="7"color="red">
<br><br>
Student Existed Already
</font></b></center></body></html>
```

notexisted.html:

```
<html>
<bodybgcolor="lightyellow">
<center><b><fontsize="7"color="red">
<br><br>
Student Not Existed
</font></b></center></body></html>
```

StudentDao.java:

```
import java.sql.ResultSet;

publicinterface StudentDao {
        public String add(String sid, String sname, int smarks);
        public ResultSet search(String sid);
        public ResultSet getStudent(String sid);
        public String update(String sid, String sname, int smarks);
        public String delete(String sid);
}
```

StudentDaoImpl:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class StudentDaoImpl implements StudentDao {

        Connection con;
        Statement st;
        ResultSet rs;
        String status = "";

        public StudentDaoImpl() {
                try {
                        Class.forName("oracle.jdbc.driver.OracleDriver");
                        con = DriverManager.getConnection(
                                "jdbc:oracle:thin:@localhost:1521:xe", "system", "durga");
                        st = con.createStatement();
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }

        public ResultSet getStudent(String sid) {
                try {
                        rs = st.executeQuery("select * from student where sid='" + sid
                                        + "'");
                } catch (Exception e) {
                        e.printStackTrace();
                }
                return rs;
        }

        public String add(String sid, String sname, int smarks) {
                try {
                        rs = getStudent(sid);
                        boolean b = rs.next();
                        if (b == true) {
                                status = "existed";
                        } else {
                                int rowCount = st.executeUpdate("insert into student values('"
                                                + sid + "','" + sname + "'," + smarks + ")");
                                if (rowCount == 1) {
                                        status = "success";
                                } else {
                                        status = "failure";
                                }
                        }
                } catch (Exception e) {

e.printStackTrace();
                }
                return status;
        }

        public ResultSet search(String sid) {
                return getStudent(sid);
        }

        public String update(String sid, String sname, int smarks) {
                try {
                        int rowCount = st
                                        .executeUpdate("update student set sname='" + sname
                                                        + "',smarks=" + smarks + " where sid='"
+ sid + "'");
                        if (rowCount == 1) {
                                status = "success";
                        } else {
                                status = "failure";
                        }
                } catch (Exception e) {
```

```
                e.printStackTrace();
            }
            return status;
        }

        public String delete(String sid) {
            try {
                rs = getStudent(sid);
                boolean b = rs.next();
                if (b == true) {
                    int rowCount = st
                    .executeUpdate("delete from student where sid='" + sid+ "'");
                    if (rowCount == 1) {
                        status = "success";
                    } else {
                        status = "failure";
                    }
                } else {
                    status = "notexisted";
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
            return status;
        }
}
```

AddServlet.java:

```
import java.io.IOException;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class AddServlet extends HttpServlet
{
        public void doGet(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException
        {
            try
            {
                String sid=req.getParameter("sid");
                String sname=req.getParameter("sname");
                int smarks=Integer.parseInt(req.getParameter("smarks"));
                StudentDao sd=new StudentDaoImpl();
                String status=sd.add(sid,sname,smarks);
            if(status.equals("existed"))
            {
            RequestDispatcherrd1=req.getRequestDispatcher("/existed.html");
                rd1.forward(req,res);
            }
            if(status.equals("success"))
            {
                req.getRequestDispatcher("success.html").forward(req,res);
            }
            if(status.equals("failure"))
            {
                req.getRequestDispatcher("failure.html").forward(req,res);
            }
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

SearchServlet.java:

```
import java.io.IOException;
import java.io.PrintWriter;

import java.sql.ResultSet;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;


import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class SearchServlet extends HttpServlet
{
    public void doGet(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException
    {
        try
        {
            res.setContentType("text/html");
            PrintWriter out=res.getWriter();
            String sid=req.getParameter("sid");
            StudentDao sd=new StudentDaoImpl();
            ResultSet rs=sd.search(sid);
            boolean b=rs.next();
            if(b==true)
            {
                out.println("<html>");
                out.println("<body bgcolor='lightyellow'>");
                out.println("<b><font size='6'><br>");
                out.println("<pre>");
                out.println("          Student Id....."+rs.getString(1));
                out.println();
                out.println("          Student Name....."+rs.getString(2));
                out.println();
                out.println("          Student Marks....."+rs.getInt(3));
                out.println("</pre></font></b></body></html>");
            }
            else
            {
                req.getRequestDispatcher("notexisted.html").forward(req,res);
            }
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

EditFormServlet:

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.ResultSet;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class EditFormServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException {
        try {
            res.setContentType("text/html");
            PrintWriter out = res.getWriter();
            String sid = req.getParameter("sid");
            StudentDao sd = new StudentDaoImpl();
            ResultSet rs = sd.getStudent(sid);
            boolean b = rs.next();
            if (b == true) {
                out.println("<html>");
                out.println("<body bgcolor='lightgreen'>");
                out.println("<b><font size='7'>");
                out.println("<br>");
                out.println("<form method='get' action='./update'>");
                out.println("<pre>");
                out.println("          Student Id            " + rs.getString(1));
                out.println("<input type='hidden' name='sid' value='" + sid
                        + "'/>");
                out.println("          Student Name               <input
type='text' name='sname' value='"
                                        + rs.getString(2) + "'/>");
                                out.println();
                                out.println("          Student marks               <input
type='text' name='smarks' value='"
                                        + rs.getInt(3) + "'/>");
                                out.println();
                                out.println("          <input type='submit' value='Update'/>");
                                out.println("</pre></form></font></b></body></html>");
                        } else {
                                req.getRequestDispatcher("notexisted.html").forward(req, res);
                        }
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

UpdateServlet.java:

```
import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class UpdateServlet extends HttpServlet {
        public void doGet(HttpServletRequest req, HttpServletResponse res)
                        throws ServletException, IOException {


                try {
                        String sid = req.getParameter("sid");
                        String sname = req.getParameter("sname");
                        int smarks = Integer.parseInt(req.getParameter("smarks"));
                        StudentDao sd = new StudentDaoImpl();
                        String status = sd.update(sid, sname, smarks);
                        if (status.equals("success")) {
                                req.getRequestDispatcher("success.html").forward(req, res);
                        }
                        if (status.equals("failure")) {
                                req.getRequestDispatcher("failure.html").forward(req, res);
                        }
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

DeleteServlet.java:

```
import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class DeleteServlet extends HttpServlet {
        public void doGet(HttpServletRequest req, HttpServletResponse res)
                        throws ServletException, IOException {
                String sid = req.getParameter("sid");
                StudentDao sd = new StudentDaoImpl();
                String status = sd.delete(sid);
                if (status.equals("success")) {
                        req.getRequestDispatcher("success.html").forward(req, res);
                }
                if (status.equals("failure")) {
                        req.getRequestDispatcher("failure.html").forward(req, res);
                }
                if (status.equals("notexisted")) {
                        req.getRequestDispatcher("notexisted.html").forward(req, res);
                }
        }
}
```

3. Applet-Servlet Communication:

In general in web application, we will use a browser as client, we will send request from client browser to a servlet available at server , by executing the respective servlet some response will be send back to the client browser.

Similarly in case of Applet-Servlet Communication, we will use applet as client, from the applet only we will send request to the respective servlet available at server machine, where by executing the respective servlet the required response will be generated and send back to the applet.

In above situation, the communication which we provided between applet and servlet is called as **Applet-Servlet Communication.**

If we want to achieve Applet-Servlet Communication in web applications we have to use the following steps.

Step 1: Prepare URL object with the respective url.

```
URL u=new
        URL(“http://localhost:8080/loginapp/login?uname=abc&upwd=abc123”);
```

Step 2: Establish connection between applet and server by using URLConnection object.

```
URLConnection uc=u.openConnection();
```

Step 3: Send request from applet to servlet.

```
uc.setDoInput(true);
uc.setDoOutpput(true);
```

Note: If we do the above step a request will be send to servlet from applet where container will execute the respective servlet, generate the response and send that response to applet client. But, the response is available on URLConnection object.

Step 4: Get InputStream from URLConnection.

```
InputSream is=uc.getInputStream();
```

Step 5: Read the response from InputStream.

```
BufferedReader br=new BufferedReader(new InputStreamReader(is));
String res=br.readLine();
```

loginapp1:-

web.xml:-

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

LoginServlet.java:-

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class LoginServlet extends HttpServlet {

        protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                PrintWriter out=response.getWriter();
                String uname=request.getParameter("uname");
                String upwd=request.getParameter("upwd");
                if(uname.equals("durga") && upwd.equals("durga")){
                        out.println("Login Success");
                }
                else{
                        out.println("Login Failure");
                }
        }

}
```

LoginApplet.java:-

```
import java.applet.Applet;
import java.awt.Button;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Label;


import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URLConnection;

public class LoginApplet extends Applet implements ActionListener {
        Label l1,l2;
        TextField tf1,tf2;
        Button b;
        String res="";
        public void init(){
                this.setBackground(Color.pink);
                this.setLayout(new FlowLayout());
                l1=new Label("User Name");
                l2=new Label("Password");
                tf1=new TextField(20);
                tf2=new TextField(20);
                tf2.setEchoChar('*');
                b=new Button("Login");
                b.addActionListener(this);
                this.add(l1);
                this.add(tf1);
                this.add(l2);
                this.add(tf2);
                this.add(b);
        }
        public void actionPerformed(ActionEvent ae) {
                try{
                        URL u=new
URL("http://localhost:2011/loginapp1/login?uname="+tf1.getText()+"&upwd="+tf2.getTex
t());
                        URLConnection uc=u.openConnection();
                        uc.setDoInput(true);
                        InputStream is=uc.getInputStream();
                        BufferedReader br=new BufferedReader(new InputStreamReader(is));
                        res=br.readLine();
                        repaint();
                }catch (Exception e) {
                        e.printStackTrace();
                }
        }
        public void paint(Graphics g){
                Font f=new Font("arial", Font.BOLD, 30);
                g.setFont(f);
                g.drawString("Status :"+res, 50, 250);
        }
}
```

LoginApplet.html:-

```
<applet code="LoginApplet" width="500" height="500"></applet>
```

GlassFish Server:

GlassFish Server is an Application Server, provided by Sun Micro Systems.

GlassFish server will provide almost all the middle ware services what application servers are provided in general like JNDI, JMS, Java Mail, Security, JTA and so on.

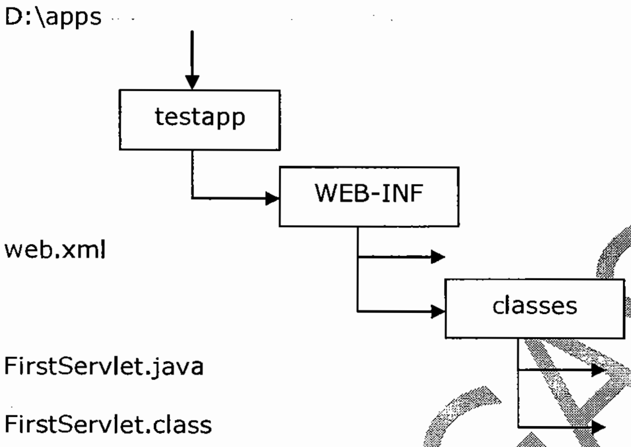
GlassFish version3 is compatible with java6 and above, it is able to provide support support for servlet3.0, jsp2.1 and so on.

To design and execute servlets in GlassFish server we are able to use Tomcat provided servlet API implementation i.e. Servlet API.jar.

If we want to design and execute web applications in GlassFish server

we have to use the following steps.

Step 1: Prepare web application and its war file separately.



To compile all the servlet files GlassFish server has provided Servlet API implementation in the form of javax.servlet.jar file.

GlassFish server has provided javax.servlet.jar file at the following location.

```
C:\glassfishv3\glassfish\modules

D:\apps\testapp\WEB-INF\classes>set classpath=C:\glassfishv3\
    glassfish\modules\ javax.servlet.jar;

D:\apps\testapp\WEB-INF\classes>javac *.java

D:\apps\testapp>jar –cvf testapp.war *.*
```

Step 2: Start the Application server.

To start Glassfish server we have execute startserv.bat file provided by Glassfish server at the following location.

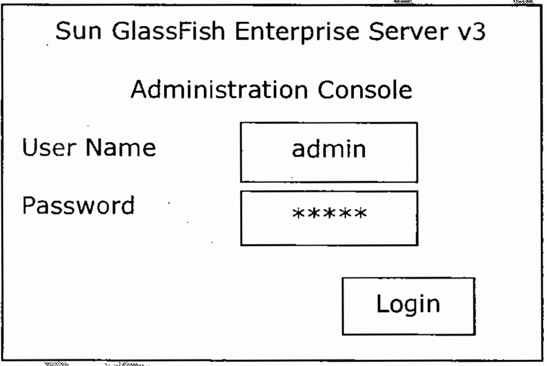
C:\glassfishv3\glassfish\bin

Step 3: Open Administration Console.

To open Administration Console in GlassFish server we have to use the following url at client browser.

http://localhost:4848

If we do the we above then Administration Console will be open, where we have to provide username and password.



Sun GlassFish Enterprise Server v3

Administration Console

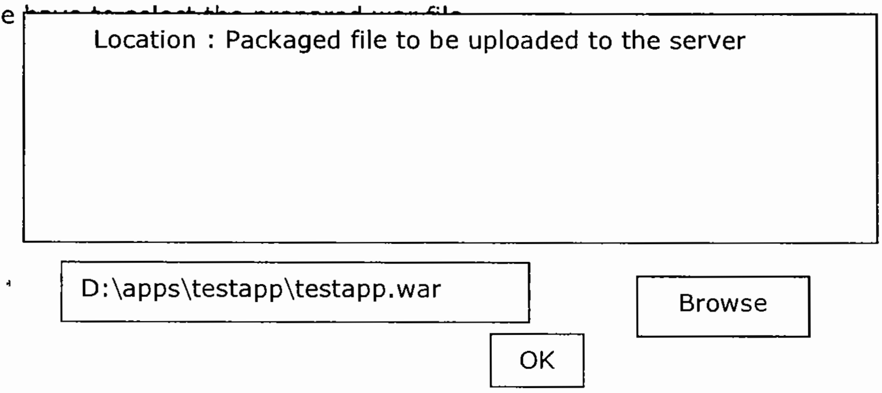
User Name: admin

Password: *****

Step 4: Deploy web application on GlassFish server.

If we click on Login button in Administration Console then automatically GlassFish server Home page will be open, where to deploy wed application go for Deployment section, where click on either List Deployed Application or Deploy an application.

If we click on Deploy an application automatically a browsing window will be open, where we have to select the prepared war file.



Location : Packaged file to be uploaded to the server

D:\apps\testapp\testapp.war

If we click on OK button, the specified war file will be uploaded to the server and the application context name will be displayed in the list of deployed applications.

NameEnabledEnginesAction

testapp ✓ web [Launch] [Redeploy] [Restart]

To access the web application we have to click on Launch hyper link under Action part.

If we click on Launch hyper link automatically a window will be open with the following url.

http://localhost:1111/testapp

If we write url pattern of the particular servlet then access the application

By using the following url.

http://localhost:1111/testapp/first

Session Tracking Mechanisms:

As part of the web application development it is essential to manage the clients previous request data at the time of processing later request.

To achieve the above requirement if we use request object then container will create request object when it receives request from client and container will destroy request object when it dispatch response to client.

Due to this reason request object is not sufficient to manage clients previous request data at the time of processing later request.

To achieve the above requirement we able to use ServletContext object, but ServletContext object will share its data to all the components which are used in the present applications and to all the users of the present web application.

Due to this reason ServletContext object is unable to provide clear cut separation between multiple users.

In web applications, to manage clients previous request data at the time of processing later request and to provide separation between multiple users we need a set of mechanisms explicitly at server side called as **Session Tracking Mechanisms.**

Session:

Session is a time duration, it will start from the starting point of client conversation with server and will terminate at the ending point of client conversation with the server.

The data which we transferred from client to server through multiple number of requests during a particular session then that data is called **State of the Session.**

In general in web applications, container will prepare a request object similarly to represent a particular user we have to prepare a separate session.

If we allow multiple number of users on a single web application then we have to prepare multiple number of session objects.

In this context, to keep track of all the session objects at server machine we need a set of explicit mechanisms called as **Session Tracking Mechanisms.**

In web applications, there are 4 types of Session Tracking Mechanisms.

1. HttpSession Session Tracking Mechanism
2. Coockies Session Tracking Mechanism
3. URL-Rewriting Session Tracking Mechanism
4. Hidden Form Fields Session Tracking Mechanism

From the above Session Tracking Mechanisms Servlet API has provided the first 3 Session Tracking Mechanisms as official mechanisms, Hidden Form Fields Session Tracking Mechanism is purely developers creation.

1. HttpSession Session Tracking Mechanism:

In **HttpSession Session Tracking Mechanism**, we will create a separate HttpSession object for each and every user, at each and every request we will pick up the request parameters from request object and we will store them in the respective HttpSession object for the sake of future reusability.

After keeping the request parameters data in HttpSession object we have to generate the next form at client browser by forwarding the request to particular static page or by generating dynamic form.

In HttpSession Session Tracking Mechanism, to create HttpSession object we will use either of the following methods.

1. req.getSession();
2. req.getSession(false);

Q: What is the difference between getSession() method and getSession(false) method?

Ans: Both the methods can be used to return HttpSession object.

To get HttpSession object if we getSession() method then container will check whether any HttpSession object existed for the respective user or not, if any HttpSession object is existed then container will return the existed HttpSession object reference.

If no HttpSession object is existed for the respective user then container will create a new HttpSession object and return its reference.

public HttpSession getSession()

Ex: HttpSession hs=req.getSession();

To get HttpSession object if we getSession(false) method then container will check whether any HttpSession object existed w.r.t. user or not, if any HttpSession object is existed then container will return that HttpSession object reference.

If no HttpSession object is existed then container will return null value without creating new HttpSession object.

public HttpSession getSession(boolean b)

Ex: HttpSession hs=req.getSession(false);

Note: getSession(true) method functionality is almost all same as getSession() method.

Q: If we allow multiple number of users to access present web application then automatically container will create multiple number of HttpSession objects. In this case how container will identify user specific HttpSession object in order to put user specific attributes and to get attributes?

Ans: In HttpSession Session Tracking Mechanism, when we create HttpSession object automatically container will create an unique identification number in the form of hexadecimal number called as **Session Id.**Container will prepare session id in the form of Cookie with the name **JSESSIONID.**

In general the basic nature of Cookie is to transfer from server to client automatically along with response and it will be transferred from client to server automatically along with request.

Due to the above nature of Cookies session id Cookie will be transferred from server to client and from client to server automatically.

In the above context, if we use getSession() method or getSession(false) method first container will pick up session id value from request and it will identify the user specific HttpSession object on the basis of session id value.

To destroy HttpSession object explicitly we will use the following method from HttpSession.

public void invalidate()

If we want to destroy HttpSession object after a particular ideal time duration then we have to use the following method.

public void setMaxInactiveInterval(int time)

In web applications, HttpSession object will allow only attributes data, it will not allow parameters data.

To set an attribute on to the HttpSession object we have to use the following method.

public void setAttribute(String name, Object value)

To get a particular attribute value from HttpSession object we have to use the following method.

public Object getAttribute(String name)

To get all attribute names from HttpSession object we have to use the following method.

public Enumeration getAttributeNames()

To remove an attribute from HttpSession object we have to use the following method.

public void removeAttribute(String name)

Drawbacks:

1. In HttpSession Session Tracking Mechanism, we will create a separate HttpSession object

for each and every user, where if we increase number of users then automatically number of HttpSession object will be created at server machine, it will reduce the overall performance of the web application.

2. In case of HttpSession Session Tracking Mechanism, we are able to identify user specific HttpSession object among multiple number of HttpSession objects by carrying Session Id value from client to server and from server to client.

In the above context, if the client browser disable Cookies then HttpSession Session Tracking Mechanism will not execute its functionality.

httpsessionapp:-

form1.html:

```
<html>
<bodybgcolor="pink">
<center>
        <formmethod="get"action="first">
                Name : <inputtype="text"name="uname"/><br><br>
                Age : <inputtype="text"name="uage"/><br><br>
                <inputtype="submit"value="Next"/>
        </form>
</center>
</body>
</html>
```

form2.html:

```
<html>
<bodybgcolor="pink">
<center>
        <formmethod="get"action="second">
                Qualification : <inputtype="text"name="uqual"/><br><br>
                Designation : <inputtype="text"name="udesig"/><br><br>
                <inputtype="submit"value="Next"/>
        </form>
</center>
</body>
</html>
```

form3.html:

```
<html>
<bodybgcolor="pink">
<center>
        <formmethod="get"action="display">
                Email : <inputtype="text"name="email"/><br><br>
                Mobile : <inputtype="text"name="mobile"/><br><br>
                <inputtype="submit"value="Display"/>
        </form>
</center>
</body>
</html>
```

web.xml:

```
<web-app>
<display-name>httpsessionapp</display-name>
<welcome-file-list>
<welcome-file>form1.html</welcome-file>
</welcome-file-list>
<servlet>
<servlet-name>FirstServlet</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>FirstServlet</servlet-name>
<url-pattern>/first</url-pattern>
</servlet-mapping>
<servlet>
<servlet-name>SecondServlet</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>SecondServlet</servlet-name>
<url-pattern>/second</url-pattern>
</servlet-mapping>


<servlet>
<servlet-name>DisplayServlet</servlet-name>


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

FirstServlet.java:

```
import java.io.IOException;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

publicclass FirstServlet extends HttpServlet {

	protectedvoid doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		String uname=request.getParameter("uname");
```

```
                String uage=request.getParameter("uage");
                HttpSession hs=request.getSession();
                hs.setAttribute("uname", uname);
                hs.setAttribute("uage", uage);
                RequestDispatcher rd=request.getRequestDispatcher("form2.html");
                rd.forward(request, response);
        }
}
```

SecondServlet.java:

```
import java.io.IOException;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

publicclass SecondServlet extends HttpServlet {

        protectedvoid doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

                String uqual=request.getParameter("uqual");
                String udesig=request.getParameter("udesig");
                HttpSession hs=request.getSession();
                hs.setAttribute("uqual", uqual);
                hs.setAttribute("udesig", udesig);
                RequestDispatcher rd=request.getRequestDispatcher("form3.html");
                rd.forward(request, response);
        }
}
```

DisplayServlet.java:

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

publicclass DisplayServlet extends HttpServlet {

        protectedvoid doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out=response.getWriter();
                String email=request.getParameter("email");
                String mobile=request.getParameter("mobile");
                HttpSession hs=request.getSession();
                String uname=(String)hs.getAttribute("uname");
                String uage=(String)hs.getAttribute("uage");
                String uqual=(String)hs.getAttribute("uqual");
                String udesig=(String)hs.getAttribute("udesig");
                out.println("<html>");
                out.println("<body bgcolor='lightgreen'>");
                out.println("<center><br><br>");
                out.println("<table bgcolor='lightyellow'>");
                out.println("<tr><td colspan='2'><center><b><font size='5'
color='red'><u>Registration Details</u></font></b></center></td></tr>");
                out.println("<tr><td>User Name</td><td>"+uname+"</td></tr>");
                out.println("<tr><td>User Age</td><td>"+uage+"</td></tr>");
                out.println("<tr><td>Qualification</td><td>"+uqual+"</td></tr>");
                out.println("<tr><td>Designation</td><td>"+udesig+"</td></tr>");
                out.println("<tr><td>Email</td><td>"+email+"</td></tr>");
                out.println("<tr><td>Mobile</td><td>"+mobile+"</td></tr>");
                out.println("<tr><td>Status</td><td>Success</td></tr>");
                out.println("</table></center>");
                out.println("</body>");
                out.println("</html>");
        }
}
```

2. Cookies Session Tracking Mechanism:

Cookie is a small object, it can be used to represent a single name value pair and which will be maintained permanently at client machine.

In HttpSession Session Tracking Mechanism, all the clients conversations will be maintained at server machine in the form of HttpSession objects.

In HttpSession Session Tracking Mechanism, if we increase number of users then automatically number of HttpSession objects will be created at

server. So that HttpSession Session Tracking Mechanism will increase burden to server machine.

To overcome the above problem we have to use an alternative mechanism, where we have to manage all the clients conversations at the respective client machines only.

To achieve the above requirement we have to use **Cookies Session Tracking Mechanism.**

In Cookies Session Tracking Mechanism, at each and every client we will pick up all the request parameters, prepare a separate Cookie for each and every request parameter, add all the Cookies to response object.

In the above context, when container dispatch response to client automatically all the added Cookies will be transferred to client and maintain at client machine permanently.

In the above context, when we send further request from the same client automatically all the Cookies will be transferred to server along with request.

By repeating the above process at each and every request we are able to manage clients previous data at the time of processing later request.

Drawbacks:

1. If we disable the Cookies at client browser then Cookies Session Tracking Mechanism will not execute its functionality.
2. In case of Cookies Session Tracking Mechanism, all the clients data will be maintain at the respective client machine only, which is open to every user of that machine. So that Cookies Session Tracking Mechanism will not provide security for the application data.

cookiesapp:-

form1.html:

```
<html>
<bodybgcolor="lightgreen">
<center>
<formaction="./first">
<br><br><br>
<h1>Product Registration Form</h1><br><br>
<h2>
        Product Id <inputtype="text"name="pid"/><br><br>
        Product Name <inputtype="text"name="pname"/><br><br>
        <inputtype="submit"value="Next"/>
</h2>
</form></center></body>
</html>
```

form2.html:

```
<html>
<bodybgcolor="lightgreen">
<center>
<formaction="./second">
<br><br><br>
<h1>Product Registration Form(Cont..)</h1><br><br>
<h2>
        Product Cost <inputtype="text"name="pcost"/><br><br>
        Product Quantity <inputtype="text"name="pquantity"/><br><br>
        <inputtype="submit"value="Next"/>
</h2>
```

```
</form></center></body>
</html>
```

form3.html:

```
<html>
<bodybgcolor="lightgreen"><center>
<formaction="./reg">
<br><br><br>
<h1>Product Registration Form(Cont..)</h1><br><br>
<h2>
        Manufactured Date <inputtype="text"name="man_Date"/><br><br>
        Expiry Date <inputtype="text"name="exp_Date"/><br><br>
        <inputtype="submit"value="Registration"/>
</h2>
</form></center></body>
</html>
```

web.xml:

```
<web-app>
<display-name>cookiesapp</display-name>
<welcome-file-list>
<welcome-file>form1.html</welcome-file>
</welcome-file-list>
<servlet>
<servlet-name>FirstServlet</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>FirstServlet</servlet-name>
<url-pattern>/first</url-pattern>
</servlet-mapping>
<servlet>
<servlet-name>SecondServlet</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>SecondServlet</servlet-name>
<url-pattern>/second</url-pattern>
</servlet-mapping>
<servlet>
<servlet-name>RegistrationServlet</servlet-name>
<servlet-class>RegistrationServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>RegistrationServlet</servlet-name>
<url-pattern>/reg</url-pattern>
</servlet-mapping>
```

```
</web-app>
```

FirstServlet.java:

```
        import java.io.IOException;

        import javax.servlet.RequestDispatcher;
        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 FirstServlet extends HttpServlet {

                protected void doGet(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException {
                        String pid=request.getParameter("pid");
                        String pname=request.getParameter("pname");


                        Cookie c1=new Cookie("pid", pid);
                        Cookie c2=new Cookie("pname", pname);
                        response.addCookie(c1);
                        response.addCookie(c2);
                        RequestDispatcher rd=request.getRequestDispatcher("form2.html");
                        rd.forward(request, response);
                }
        }
```

SecondServlet.java:

```
        import java.io.IOException;

        import javax.servlet.RequestDispatcher;
        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 SecondServlet extends HttpServlet {

                protected void doGet(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException {
                        try{
                                int pcost=Integer.parseInt(request.getParameter("pcost"));
                                int
        pquantity=Integer.parseInt(request.getParameter("pquantity"));
                                Cookie c3=new Cookie("pcost", ""+pcost);
                                Cookie c4=new Cookie("pquantity", ""+pquantity);
                        response.addCookie(c3);
                        response.addCookie(c4);
                        RequestDispatcher
        rd=request.getRequestDispatcher("form3.html");
                        rd.forward(request, response);
                }
                catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

RegistrationServlet.java:

```
        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 RegistrationServlet extends HttpServlet {

                protected void doGet(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException {
                        response.setContentType("text/html");
                        PrintWriter out=response.getWriter();
                        String man_Date=request.getParameter("man_Date");
                        String exp_Date=request.getParameter("exp_Date");
                        Cookie[] c=request.getCookies();
                        String pid=c[0].getValue();
                        String pname=c[1].getValue();
                        int pcost=Integer.parseInt(c[2].getValue());
                        int pquantity=Integer.parseInt(c[3].getValue());
                        ProductBean pb=new ProductBean();
                        String
        status=pb.register(pid,pname,pcost,pquantity,man_Date,exp_Date);
                        out.println("<html>");
                        out.println("<body bgcolor='pink'>");
                        out.println("<center><br><br>");
                        out.println("<u>Product Registration Details</u><br><br>");
                        out.println("Product Id....."+pid+"<br><br>");
                        out.println("Product Name....."+pname+"<br><br>");
                        out.println("Product Cost....."+pcost+"<br><br>");
                        out.println("Product Quantity....."+pquantity+"<br><br>");
                        out.println("Product Manufactured
        Date....."+man_Date+"<br><br>");
                        out.println("Product Expiry Date....."+exp_Date+"<br><br>");
                        out.println("Status....."+status);
                        out.println("</center>");
                        out.println("</body>");
                        out.println("</html>");
                }
}
```

ProductBean.java:

```
        import java.sql.Connection;
        import java.sql.DriverManager;
        import java.sql.ResultSet;
        import java.sql.Statement;

        public class ProductBean {
                Connection con;
                Statement st;
                ResultSet rs;
                String status="";
                public ProductBean(){
                        try{
                                Class.forName("oracle.jdbc.driver.OracleDriver");


                con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","sy
        stem","durga");
                                st=con.createStatement();
                        }
                        catch (Exception e) {
                                e.printStackTrace();
                        }
                }
                public String register(String pid,String pname,int pcost,int pquantity,String
        man_Date,String exp_Date){
                        try{
                                rs=st.executeQuery("select * from product where
        pid='"+pid+"'");
                                boolean b=rs.next();
                                if(b==true){
                                        status="Product Existed Already";
                                }
                                else{
                                        int rowCount=st.executeUpdate("insert into product
        values('"+pid+"','"+pname+"',"+pcost+","+pquantity+",'"+man_Date+"','"+exp_Da
        te+"')");
                                        if(rowCount==1){
                                                status="SUCCESS";
                                        }
                                        else{
                                                status="FAILURE";
                                        }
                                }
                        }
                        catch (Exception e) {
```

```
                                        status="FAILURE";
                                        e.printStackTrace();
                              }
                              return status;
                    }
}
```

To represent a Cookie in web application Servlet API has provided a predefined class in the form of javax.servlet.http.Cookie.

To create Cookie object with a particular name-value pair we have to use the following Cookie class constructor.

    public Cookie(String name, String value)

To add a Cookie to the response object we have to use the following method from HttpServletResponse.

    public void addCookie(Cookie c)

To get all the Cookies from response object we need to use the following method.

    public Cookie[] getCookies()

To get name and value of a Cookie we have to use the following methods,

    public String getName()

    public String getValue()

In web applications, it is possible to provide comments to the Cookies. So that to set the comment to Cookie and get the comment from Cookie we need to use the following methods.

    public void setComment(String comment)

    public String getComment()

In web applications, it is possible to provide version numbers to the Cookies. So that to set a version number to Cookie and get a version number from Cookie we need to use the following methods.

    public void setVersion(int version_no)

    public int getVersion()

In web applications, it is possible to specify life time to the Cookies. So that to set max age to Cookie and get max age from Cookie we need to use the following methods.

    public void setMaxAge(int age)

public int getMaxAge()

In web applications, it is possible to provide domain names to the Cookies. So that to set domain name to Cookie and get domain name from Cookie we need to use the following methods.

public void setDomain(String domain)

public String getDomain()

In web applications, it is possible to provide a particular path to the Cookies to store. So that to set a path to Cookie and get a path from Cookie we need to use the following methods.

public void setPath(String path)

public String getPath()

3. URL-Rewriting Session Tracking Mechanism:

In case of HttpSession Session Tracking Mechanism, when we create HttpSession object automatically Session Id will be created in the form of the Cookie, where Session Id Cookie will be transferred from server to client and from client to server along with response and request automatically.

In HttpSession Session Tracking Mechanism, we are able to identify user specific HttpSession object on the basis of Session Id only.

In this context, if we disable Cookies at client browser then HttpSession Session Tracking Mechanism will not execute its functionality.

In case of Cookies Session Tracking Mechanism, the complete client conversation will be stored at the respective client machine only in the form of Cookies, here the Cookies data will be opened to every user of that machine. So that Cookies Session Tracking Mechanism will not provide security for the application data.

To overcome the above problem, we have to use URL-Rewriting Session Tracking Mechanism.

In case of URL-Rewriting Session Tracking Mechanism, we will not maintain the clients conversation at the respective client machine, we will maintain the clients conversationin the form of HttpSession object at server machine. So that URL-Rewriting Session Tracking Mechanism is able to provide very good security for the application data.

URL-Rewriting Session Tracking Mechanism is almost all same as HttpSession Session Tracking Mechanism, in URL-Rewriting Session Tracking Mechanism we will not depending on a Cookie to maintain Session Id value, we will manage Session Id value as an appender to URL in the next generated form.

In this context, if we send a request from the next generated form automatically the appended Session Id value will be transferred to server along with the request.

In this case, eventhough if we disable Cookies at client browser, but still we are able to get Session Id value at server machine and we are able to manage clients previous request data at the time of processing the later request.

In URL-Rewriting Session Tracking Mechanism, every time we need to rewrite the URL with Session Id value in the next generated form. So that this mechanism is called as **URL-Rewriting Session Tracking Mechanism.**

In URL-Rewriting Session Tracking Mechanism, it is mandatory to append Session Id value to the URL by getting Session Id value explicitly.

To perform this work HttpServletResponse has provided a separate method like,

public String encodeURL(String url)

Ex: out.println("<form method='get'

action='"+res.encodeURL("./second")+"'>");

Drawback:

In URL-Rewriting Session Tracking Mechanism, every time we need to rewrite the URL with Session Id value in the generated form, for this we must execute encodeURL() method. So that URL-Rewriting Session Tracking Mechanism should require dynamically generated forms, it will not execute its functionality with static forms.

urlrewritingapp:-

form1.html:

```
<html>
<bodybgcolor="lightgreen">
        <br><br><center><h1>Deposit Form</h1></center>
        <br><br><hr><br><br>
        <formaction="./first">
                <pre>
                        Account Number <inputtype="text"name="accNo"/>

                        Account Name   <inputtype="text"name="accName">

                        <inputtype="submit"value="Next">
                </pre>
        </form></body>
</html>
```

web.xml:

```
<web-app>
<display-name>urlrewritingapp</display-name>
<welcome-file-list>
<welcome-file>form1.html</welcome-file>
</welcome-file-list>
<servlet>
<servlet-name>FirstServlet</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>FirstServlet</servlet-name>
<url-pattern>/first</url-pattern>
</servlet-mapping>
<servlet>
<servlet-name>SecondServlet</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>SecondServlet</servlet-name>
<url-pattern>/second</url-pattern>
</servlet-mapping>

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

FirstServlet.java:

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class FirstServlet extends HttpServlet {

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		PrintWriter out=response.getWriter();
		String accNo=request.getParameter("accNo");
		String accName=request.getParameter("accName");
		HttpSession session=request.getSession();
		session.setAttribute("accNo", accNo);
```

```
                session.setAttribute("accName", accName);
                out.println("<html><body bgcolor='cyan'><br><br>");
                out.println("<center><h1>Deposit
Form(Cont..)</h1></center><br><br><hr><br><br>");
                out.println("<form method='get'
action='"+response.encodeUrl("./second")+"'>");
                out.println("<pre>");
                out.println("Account Type    <input type='text' name='accType'/>");
                out.println();
                out.println("Account Branch <input type='text' name='accBranch'/>");
                out.println();
                out.println("  <input type='submit' value='Next'/>");
                out.println("</pre></form></body></html>");
        }
}
```

SecondServlet.java:

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class SecondServlet extends HttpServlet {

        protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                PrintWriter out=response.getWriter();
                String accType=request.getParameter("accType");
                String accBranch=request.getParameter("accBranch");
                HttpSession session=request.getSession();
                session.setAttribute("accType", accType);
                session.setAttribute("accBranch", accBranch);
                out.println("<html><body bgcolor='cyan'><br><br>");
                out.println("<center><h1>Deposit
Form(Cont..)</h1></center><br><br><hr><br><br>");
                out.println("<form method='get'
action='"+response.encodeUrl("./deposit")+"'>");
                out.println("<pre>");
                out.println("Depositor Name <input type='text' name='depName'/>");
                out.println();
                out.println("Deposit Amount <input type='text' name='depAmount'/>");
                out.println();
                out.println("  <input type='submit' value='Deposit'/>");
                out.println("</pre></form></body></html>");

        }
}
```

DepositServlet.java:

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class DepositServlet extends HttpServlet {


        protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out=response.getWriter();
                String depName=request.getParameter("depName");
                int depAmount=Integer.parseInt(request.getParameter("depAmount"));
                HttpSession session=request.getSession();
                String accNo=(String)session.getAttribute("accNo");
                String accName=(String)session.getAttribute("accName");
                String accType=(String)session.getAttribute("accType");
                String accBranch=(String)session.getAttribute("accBranch");
                TransactionBean tb=new TransactionBean();
                String status=tb.deposit(accNo,depAmount);
                out.println("<html><body bgcolor='lightblue'><br><br>");
                out.println("<center><table>");
                out.println("<tr><td colospan='2'><center><b><font size='6'
color='red'>");
                out.println("<u>Transaction
Details</u><font></b></center></td></tr>");
                out.println("<tr><td>Transaction Id</td><td>:t1</td></tr>");
                out.println("<tr><td>Transaction Name</td><td>:Deposit</td></tr>");
                out.println("<tr><td>Account Number</td><td>:"+accNo+"</td></tr>");
                out.println("<tr><td>Account Name</td><td>:"+accName+"</td></tr>");
                out.println("<tr><td>Account Type</td><td>:"+accType+"</td></tr>");
                out.println("<tr><td>Account
Branch</td><td>:"+accBranch+"</td></tr>");
                out.println("<tr><td>Depositor
Name</td><td>:"+depName+"</td></tr>");
                out.println("<tr><td>Deposit
Amount</td><td>:"+depAmount+"</td></tr>");
                int totalAvailableAmount=tb.getTotalAvailableAmount();
                out.println("<tr><td>Total Available
Amount</td><td>:"+totalAvailableAmount+"</td></tr>");
                out.println("<tr><td>Transaction
Status</td><td>:"+status+"</td></tr>");
                out.println("<tr><td colspan='2'><b><font size='5'>...Visit
Again...</td></tr>");
                out.println("</table></center></body></html>");
        }
}
```

TransactionBean.java:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class TransactionBean {

        Connection connection;


        Statement statement;
        ResultSet resultSet;
        String status="";
        int totalAvailableAmount;

        TransactionBean(){
                try{
                        Class.forName("oracle.jdbc.driver.OracleDriver");

        connection=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",
"system", "durga");
                        statement=connection.createStatement();
                }
                catch (Exception e) {
                        e.printStackTrace();
                }
        }
        public String deposit(String accNo,int depAmount){
                try{
                        int rowCount=statement.executeUpdate("update acc_details set
balance=balance+"+depAmount+"where accNo='"+accNo+"'");
                        if(rowCount==1)
                                status="SUCCESS";
                        else
                                status="FAILURE";
                }
                catch (Exception e) {
                        status="FAILURE";
                        e.printStackTrace();
                }
                return status;
        }
        public int getTotalAvailableAmount() {
                try{
                        resultSet=statement.executeQuery("select * from acc_details");
                        resultSet.next();
                        totalAvailableAmount=resultSet.getInt(4);
                }
                catch (Exception e) {
                        e.printStackTrace();
                }
                return totalAvailableAmount;
        }
}
```

4. Hidden Form Field Session Tracking Mechanism:

Hidden Form Field Session Tracking Mechanism is not official Session Tracking Mechanism from Servlet API, it was purely developers creation.

In Hidden Form Field Session Tracking Mechanism, at each and every request we will pick up all the request parameters, generate dynamic form, in dynamic form generation we have to maintain the present request parameters data in the form of hidden fields.

In the above context, if we dispatch the response to client then we are able to get a dynamic form with visible fields and with invisible fields.

If we send a request from dynamic form then automatically all the visible fields data and invisible fields data will be send to server as request parameters.

By repeating above process at each and every request we are able to manage the clients previous request data at the time of processing the later request.

hiddenformfieldsapp:-

studentform.html:-

```
<html>
<head>
<bodybgcolor="lightgreen">
        <formmethod="get"action="./first">
        <center><b><br><br>
        Student Name:<inputtype="text"name="sname"/><br><br>
        Student Id:<inputtype="text"name="sid"/><br><br>
        Student Address:<inputtype="text"name="saddr"/><br><br>
        <inputtype="submit"value="Submit">
        </b></center>
        </form>
</body>
</html>
```

web.xml:-

```
<web-app>
<display-name>hiddenformfieldsapp</display-name>
<welcome-file-list>
<welcome-file>studentform.html</welcome-file>
</welcome-file-list>
<servlet>
<servlet-name>FirstServlet</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>FirstServlet</servlet-name>
<url-pattern>/first</url-pattern>
</servlet-mapping>
<servlet>
<servlet-name>SecondServlet</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>


<servlet-mapping>
<servlet-name>SecondServlet</servlet-name>
<url-pattern>/second</url-pattern>
</servlet-mapping>
<servlet>
<servlet-name>ThirdServlet</servlet-name>
<servlet-class>ThirdServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>ThirdServlet</servlet-name>
<url-pattern>/third</url-pattern>
</servlet-mapping>
</web-app>
```

FirstServlet.java:-

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class FirstServlet extends HttpServlet {

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		PrintWriter out=response.getWriter();
		String sname=request.getParameter("sname");
		String sid=request.getParameter("sid");
		String saddr=request.getParameter("saddr");
		out.println("<html><body bgcolor='lightyellow'>");
```

```
                out.println("<center><b><br><br>");
                out.println("Welcome to Student Application");
                out.println("<br><br>");
                out.println("<form method='get' action='/hiddenformfieldsapp/second'>");
                out.println("<input type='hidden' name=sname value='"+sname+"'>");
                out.println("<input type='hidden' name=sid value='"+sid+"'>");
                out.println("<input type='hidden' name=saddr value='"+saddr+"'>");
                out.println("<br><br>");
                out.println("Student Age:");
                out.println("<input type='text' name='sage'>");
                out.println("<br><br>");
                out.println("<input type='submit' value='Submit'>");
                out.println("</form></b></center></body></html>");

        }

}
```

SecondServlet.java:-

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class SecondServlet extends HttpServlet {

        protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                PrintWriter out=response.getWriter();
                String sname=request.getParameter("sname");
                String sid=request.getParameter("sid");
                String saddr=request.getParameter("saddr");
                String sage=request.getParameter("sage");
                out.println("<html><body bgcolor='lightyellow'>");
                out.println("<center><b><br><br>");
                out.println("Student Details Are...");
                out.println("<br><br>");
                out.println("Student Name....."+sname+"<br><br>");
                out.println("Student Id....."+sid+"<br><br>");
                out.println("Student Address....."+saddr+"<br><br>");
                out.println("<a href='/hiddenformfieldsapp/third?sage="+sage+"'>
                                                        SHOW STUDENT AGE</a>");
}
}
```

ThirdServlet.java:-

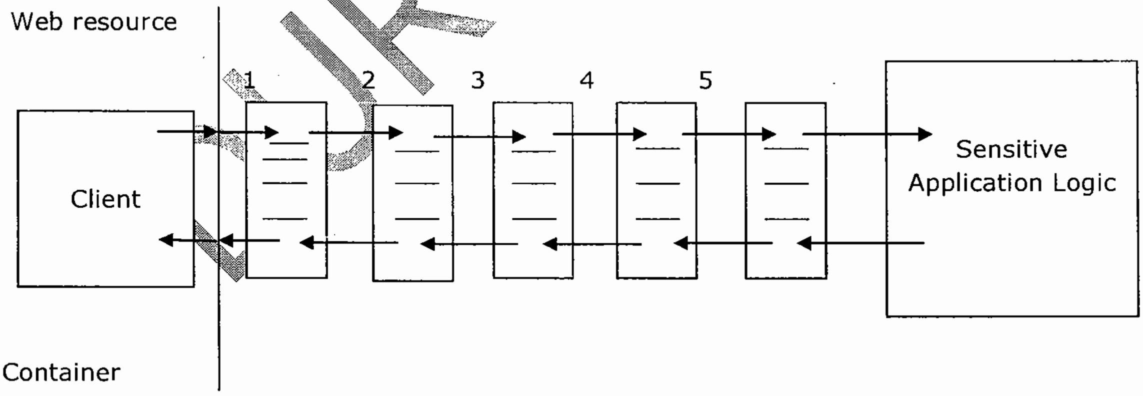
```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ThirdServlet extends HttpServlet {

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
			PrintWriter out=response.getWriter();
			String sage=request.getParameter("sage");
			out.println("<html><body bgcolor='lightpink'>");
			out.println("<center><b><br><br>");
			out.println("Student Age is....."+sage);
			out.println("</b></center></body></html>");
	}
}
```

Filters:

In general in web application development, we will provide the complete application logic in the form of the web resources like servlets, jsps and so on.

As part of the web application development some web resources may require the services like Authentication, Authorization, Security, Data compression and decompression and so on as preprocessing and post processing.

In the above context, to implement all the above preprocessing and post processing services Servlet API has provided a separate component called **Filter.**



1. Authentication Filter 2. Authorization Filter 3. Security Filter

4. Compression and Decompression Filter 5. Encoding and Decoding Filter

→ Preprocessing ← Post processing

From the above representation when we send a request from client to server for a particular web resource then container will pick up that request, container will check whether any Filter is associated with the respective web resource, if container identify any Filter or Filters then container will execute that Filters first.

While executing a Filter if the present request is satisfied all the Filter constraints then only container will bypass that request to next Filter or next web resource.

If the present request is not satisfied the Filter constraints then container will generate the respective response to client.

Filter is a server side component, it will be executed by the container automatically when it receives request from client for a particular web resource.

If we want to use Filters in our web applications, we have to use the following steps.

Step 1: Prepare Filter class.

Step 2: Configure Filter class in web.xml file.

Step 1: Prepare Filter class:

Filter is an object available at server machine, it must implement Filter interface either directly or indirectly.

```
public interface Filter  {
public void init(FilterConfig config)throws ServletException;
public void doFilter(ServletRequest req, ServletResponse res, FilterChain fc)throws SE, IOE
   public void destroy(); }
 public class MyFilter implements Filter { -----}
```

Where init(_) method can be used to perform Filter Initialization.

Where doFilter(_,_,_) method is same as service(_,_) method in servlets it is able to accommodate actual Filter logic.

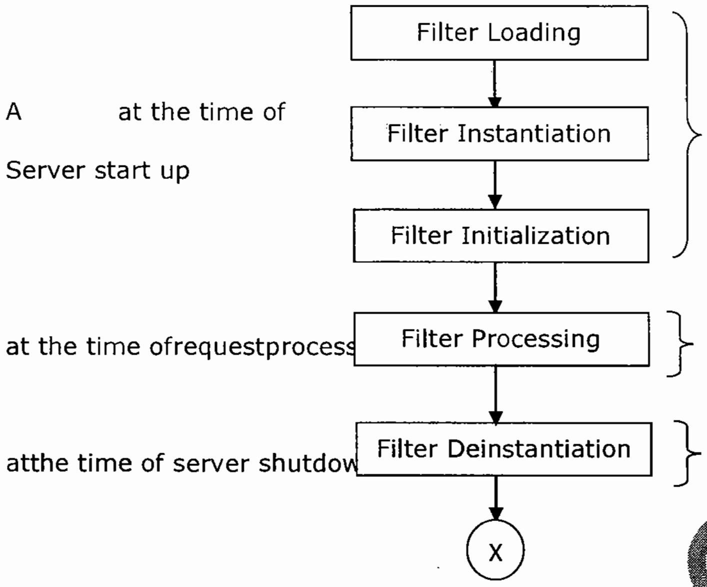
Where destroy() method can be used to perform Filter Deinstantiation.

While executing a particular Filter in web applications, if we satisfy all the Filter constraints then we need to bypass the request from present Filter to the next Filter web resource, for this we need to use the following method from FilterChain.

```
public void doFilter(ServletRequest req, ServletResponse res)throws SE, IOE
```

While executing a particular web application, when container identify a particular Filter to execute then container will execute that Filter by following the following Filter life cycle.

↓



In web applications, by default all the Filters are auto-loaded, auto-instantiated, auto-initialized at the time of server start up. So that Filters should not require load-on-startup configuration in web.xml file.

Step 2: Filter class Configuration:

To configure a Filter in web.xml file we have to use the following xml tags.

```
<web-app>

          --------

<filter>

<filter-name>logical_name</filter-name>

<filter-class>Fully Qualified name of Filter</filter-class>

</filter>

<filter-mapping>

<filter-name>logical_name</filter-name>

<url-pattern>pattern_name</url-pattern>

                                  or

<servlet-name>logical name of servlet</servlet-name>
```

```
</filter-mapping>

--------

</web-app>
```

If we want to provide mapping between a Filter and Servlet then we have to provide the same url-pattern for both Filter and Servlet or provide the respective servlet logical name along with <servlet-name> tag in place of <url-pattern> tag under <filter-mapping>.

In web applications, it is possible to use a single Filter for multiple number of web resources.

To achieve this we have to use " **/*** " (Directory match) as url-pattern to the respective Filter.

In web applications, it is possible to provide multiple number of Filters for a single web resource, in this case container will execute all the Filters as per the order in which we provided <filter-mapping> tags in web.xml file.

filterapp:-

studentform.html:-

```
<html><bodybgcolor="lightgreen">
<center><br><br>
        <formmethod="get"action="./filter">
        <tablebgcolor="pink"border="2">
        <tr>
                <td>Student Name</td>
                <td><inputtype="text"name="sname"/></td>
        </tr>
        <tr>
                <td>Student Age</td>
                <td><inputtype="text"name="sage"/></td>
        </tr>
        <tr>
                <td>Student Address</td>
                <td><inputtype="text"name="saddr"/></td>
        </tr>
        <tr>
                <td><inputtype="submit"value=""/></td>
                <td><inputtype="submit"value="SUBMIT"/></td>
        </tr>
        </table></form></center>
</body></html>
```

web.xml:-

```
<web-app>
<display-name>filterapp</display-name>
<welcome-file-list>
<welcome-file>studentform.html</welcome-file>
</welcome-file-list>
<servlet>
<servlet-name>MyServlet</servlet-name>
<servlet-class>MyServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>MyServlet</servlet-name>
<url-pattern>/filter</url-pattern>
</servlet-mapping>
<filter>
<filter-name>MyFilter</filter-name>
<filter-class>MyFilter</filter-class>
</filter>
<filter-mapping>
<filter-name>MyFilter</filter-name>
<url-pattern>/filter</url-pattern>
</filter-mapping>
</web-app>
```

MyFilter.java:-

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class MyFilter implements Filter {

    public MyFilter() {

    }

        public void destroy() {

        }

        public void doFilter(ServletRequest request, ServletResponse response, FilterChain
chain) throws IOException, ServletException {
                PrintWriter out=response.getWriter();
                out.println("<body bgcolor='yellow'>");
                out.println("<center><b><br><br>");
                int age=Integer.parseInt(request.getParameter("sage"));
                if(age<18)
                {
                        out.println("U R not allowed for this Fashion Show");
                        out.println("<br><br>");
                        out.println("Bcoz, U R minor");
                        out.println("<br><br>");
                        out.println("Try after few years");
                        out.println("<br><br>");
                        out.println("<a href='/filterapp/studentform.html'>New Student
Form</a>");

                }
                else{
                        chain.doFilter(request, response);
                }
        }

        public void init(FilterConfig fConfig) throws ServletException {

        }

}
```

MyServlet.java:-

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MyServlet extends HttpServlet {

        protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

                PrintWriter out=response.getWriter();
                String sname=request.getParameter("sname");
                int sage=Integer.parseInt(request.getParameter("sage"));
                String saddr=request.getParameter("saddr");
                out.println("<br><br>");
                out.println("Hello Student..... Ur details are....");
                out.println("<br><br>");
                out.println("Name....."+sname);
                out.println("<br><br>");
                out.println("Name....."+sage);
                out.println("<br><br>");
                out.println("Name....."+saddr);
                out.println("<br><br>");
                out.println("Enjoy the Fashion show");
        }

}
```

ValidationsApp

Registrationform.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<font color="red">
<h2>Durga Software Solutions</h2>
<h3>Employee Registration Form</h3>
</font>
<form method="post" action="./reg">
<table>
<tr>
        <td>Employee Id</td>
        <td><input type="text" name="eid"/></td>
</tr>
<tr>
        <td>Employee Name</td>
        <td><input type="text" name="ename"/></td>
</tr>
<tr>
        <td>Employee Age</td>
        <td><input type="text" name="eage"/></td>
</tr>
<tr>
        <td>Employee Email</td>
        <td><input type="text" name="eemail"/></td>
</tr>
<tr>
        <td>Employee Mobile</td>
        <td><input type="text" name="emobile"/></td>
</tr>
<tr>
        <td><input type="submit" value="Registration"/></td>
</tr>
</table>
</form>
</body>
</html>
```

Web.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">
<display-name>filtersapp</display-name>
<welcome-file-list>
<welcome-file>registrationform.html</welcome-file>
</welcome-file-list>
<servlet>
<description></description>
<display-name>RegistrationServlet</display-name>
<servlet-name>RegistrationServlet</servlet-name>
<servlet-class>com.durgasoft.RegistrationServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>RegistrationServlet</servlet-name>
<url-pattern>/reg</url-pattern>
</servlet-mapping>
<filter>
<display-name>RegistrationFilter</display-name>
<filter-name>RegistrationFilter</filter-name>
<filter-class>com.durgasoft.RegistrationFilter</filter-class>
</filter>
<filter-mapping>
<filter-name>RegistrationFilter</filter-name>
<url-pattern>/reg</url-pattern>
</filter-mapping>
</web-app>
```

RegistrationFilter.java

```
package com.durgasoft;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
public class RegistrationFilter implements Filter {
String eid_err_msg="",
ename_err_msg="",
eage_err_msg="",
eemail_err_msg="",
emobile_err_msg="";
        boolean b=true;
        public void destroy() {
        }

        public void doFilter(ServletRequest request, ServletResponse response, FilterChain
chain) throws IOException, ServletException {
                try {
```

```
response.setContentType("text/html");
PrintWriter out=response.getWriter();
String eid=request.getParameter("eid");
String ename=request.getParameter("ename");
String eage=request.getParameter("eage");
String eemail=request.getParameter("eemail");
String emobile=request.getParameter("emobile");
if(eid == null || eid.equals("")){
        eid_err_msg="Employee Id is Required";
        b=false;
}else{
        if(!eid.startsWith("DSS-")){
        eid_err_msg="Employee Id Must be in DSS-ddd Format";
                b=false;
        }else{
                b=true;
        }
}
if(ename == null || ename.equals("")){
        ename_err_msg="Employee Name is Required";
        b=false;
}else{
        b=true;
}
if(eage == null || eage.equals("")){
        eage_err_msg="Employee Age is Required.";
        b=false;
}else{
        int age=Integer.parseInt(eage);
        if(age<20 || age>30){
eage_err_msg="Employee Age Must be in between 20 to 30 Years.";
                b=false;
        }else{
                b=true;
        }
}
if(eemail == null || eemail.equals("")){
        eemail_err_msg="Employee Email is Required";
        b=false;
}else{
        if(!eemail.endsWith("@durgasoft.com")){
                eemail_err_msg="Employee Email is Invalid";
                b=false;
        }else{
                b=true;
        }
}

if(emobile == null || emobile.equals("")){
        emobile_err_msg="Employee Mobile is Required";
        b=false;
}else{
        if(!emobile.startsWith("91-")){
```

```
                    emobile_err_msg="Employee Mobile Num. is Invalid";
                    b=false;
                }else{
                    b=true;
                }
            }
            if(b==false){
            out.println("<html>");
            out.println("<body>");
            out.println("<font color='red'>");
            out.println("<h2>Durga Software Solutions</h2>");
            out.println("<h3>Employee Registration Form</h3>");
            out.println("</font>");
            out.println("<font color='blue' size='5'>");
            out.println(eid_err_msg+"<br>");
            out.println(ename_err_msg+"<br>");
            out.println(eage_err_msg+"<br>");
            out.println(eemail_err_msg+"<br>");
            out.println(emobile_err_msg+"<br>");
            out.println("</font>");
            out.println("<form method='post' action='./reg'>");
            out.println("<table>");
            out.println("<tr><td>Employee Id</td><td><input type='text'
name='eid' value='"+eid+"'/></td><td>"+eid_err_msg+"</td></tr>");
            out.println("<tr><td>Employee Name</td><td><input type='text'
name='ename' value='"+ename+"'/></td><td>"+ename_err_msg+"</td></tr>");
            out.println("<tr><td>Employee Age</td><td><input type='text'
name='eage' value='"+eage+"'/></td><td>"+eage_err_msg+"</td></tr>");
            out.println("<tr><td>Employee Email</td><td><input type='text'
name='eemail' value='"+eemail+"'/></td><td>"+eemail_err_msg+"</td></tr>");
            out.println("<tr><td>Employee Mobile</td><td><input type='text'
name='emobile' value='"+emobile+"'/></td><td>"+emobile_err_msg+"</td></tr>");
            out.println("<tr><td><input type='submit'
value='Registration'/></td></tr>");
            out.println("</table></form></body></html>");
            eid_err_msg="";
            ename_err_msg="";
            eage_err_msg="";
            eemail_err_msg="";
            emobile_err_msg="";
            b=true;
            }else{
                chain.doFilter(request, response);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
    public void init(FilterConfig fConfig) throws ServletException {
    }

}
```

RegistrationServlet.java

```
package com.durgasoft;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class RegistrationServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;
        protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out=response.getWriter();
                try {
                        String eid=request.getParameter("eid");
                        String ename=request.getParameter("ename");
                        int eage=Integer.parseInt(request.getParameter("eage"));
                        String eemail=request.getParameter("eemail");
                        String emobile=request.getParameter("emobile");
                        out.println("<html>");
                        out.println("<body>");
                        out.println("<font color='red'>");
                        out.println("<h2>Durga Software Solutions</h2>");
                        out.println("<h3>Employee Registration Details</h3>");
                        out.println("</font>");
                        out.println("<table border='1'>");
                        out.println("<tr><td>Employee Id</td><td>"+eid+"</td></tr>");
                        out.println("<tr><td>Employee
Name</td><td>"+ename+"</td></tr>");
                        out.println("<tr><td>Employee
Age</td><td>"+eage+"</td></tr>");
                        out.println("<tr><td>Employee
Email</td><td>"+eemail+"</td></tr>");
                        out.println("<tr><td>Employee
Mobile</td><td>"+emobile+"</td></tr>");
                        out.println("</table></body></html>");
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }

}
```

Servlet Wrappers:

The main purpose of Servlet Wrappers in web applications is to simplify the customization of request and response objects.

In general in web application execution when we send a request from client to server for a particular servlet then container will pick up the request, identify the requested servlet, perform servlet life cycle stages like servlet loading, servlet instantiation, servlet initialization, request processing and servlet deinstantiation.

As part of request processing container has to access service(_,_) method, for this container has to prepare servlet request and servlet response objects.

As part of application requirement we need to use our own request and response objects in the servlets instead of using container provided request and response objects.

To achieve this requirement we have to perform the customization of request and response objects.

In general Servlet is an abstraction i.e. collection of interfaces provided by Sun Micro Systems and whose implementations could be provided by all the server vendors.

Similarly if we want to customize request and response objects we have to implement ServletRequest and ServletResponse interfaces by taking implementation classes.

```
    public class MyRequest implements ServletRequest
{ ---------
   --------- }
public class MyResponse implements ServletResponse
   { ---------
      --------- }
```

To perform request and response objects coustomization if we use the above approach then we must implement directly ServletRequest and ServletResponse interfaces i.e. we must provide implementation for each and every method declared in ServletRequest and ServletResponse interfaces. This approach will increase burden to the developers.

To overcome the above problem Servlet API has provided an alternative in the form of a set of predefined classes called **Wrapper classes.**

All the Servlet Wrapper classes are direct implementation classes to the respective ServletRequest and ServletResponse and so on.

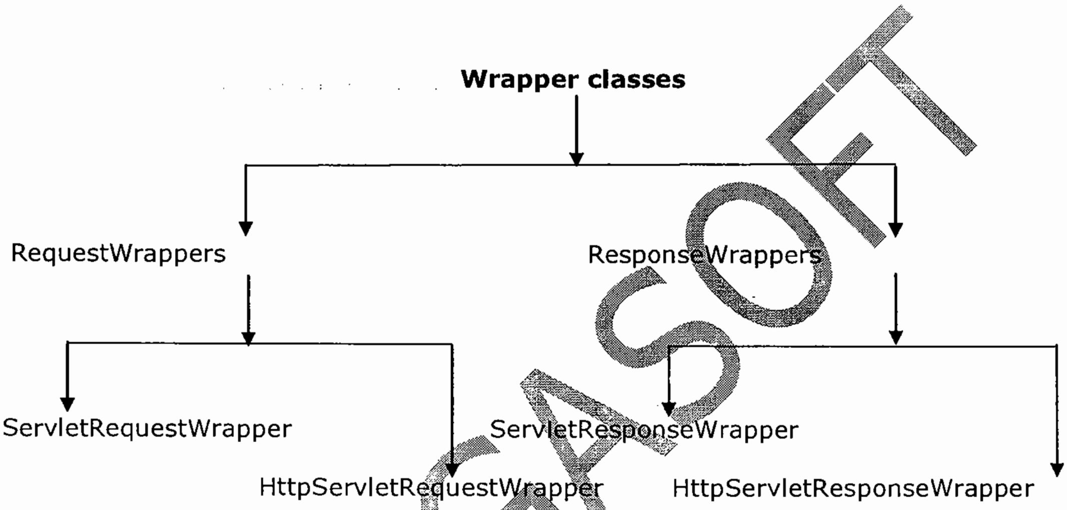
Servlet Wrapper classes is an idea provided by a design pattern called as **Adapter Design Pattern.**

If we want to prepare our own request and response classes we have to extend the respective Servlet Wrapper class and override required method.

```
public class MyClass extends Xxxxx
{ --------
  -------- }
```

Where Xxxxx may be a particular Wrapper class.

There are 2 types of Wrapper classes in Servlet API.



To customize the request and response objects we have to use a Filter in order to prepare our own request and response objects and to pass our own request and response objects to the service(_,_) method.

requestwrapperapp:-

registrationform.html:-

```
<html>
<bodybgcolor="lightgreen">
        <center><h1><u>Registration Form</u></h1></center>
        <b><fontsize="7">
        <formmethod="get"action="./reg">
        <pre>
                Name      <inputtype="text"name="uname"/>

                Password  <inputtype="password"name="upwd"/>

                Email     <inputtype="text"name="email"/>@dss.com

                Mobile    <inputtype="text"name="mobile"/>
                <inputtype="submit"value="REGISTRATION"/>
        </pre>
        </form>
        </font></b>
</body>
</html>
```

web.xml:-

```
<web-app>
<display-name>requestwrapperapp</display-name>
<welcome-file-list>
<welcome-file>registrationform.html</welcome-file>
</welcome-file-list>

<filter>
<filter-name>MyFilter</filter-name>
<filter-class>MyFilter</filter-class>
</filter>
<filter-mapping>
<filter-name>MyFilter</filter-name>
<url-pattern>/reg</url-pattern>
</filter-mapping>
<servlet>
<servlet-name>MyServlet</servlet-name>
<servlet-class>MyServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>MyServlet</servlet-name>
<url-pattern>/reg</url-pattern>
</servlet-mapping>
</web-app>
```

MyFilter.java:-

```
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class MyFilter implements Filter {

   public void destroy() {

        }
        public void doFilter(ServletRequest request, ServletResponse response, FilterChain
chain) throws IOException, ServletException {

                MyRequest myRequest=new MyRequest(request);
                chain.doFilter(myRequest, response);
        }
        public void init(FilterConfig fConfig) throws ServletException {

        }

}
```

MyRequest.java:-

```
import javax.servlet.ServletRequest;
import javax.servlet.ServletRequestWrapper;

public class MyRequest extends ServletRequestWrapper {


        ServletRequest request;

        public MyRequest(ServletRequest request){
                super(request);
                this.request=request;
        }
        public String getParameter(String name){
                String value=request.getParameter(name);
                if(name.equals("email")){
                        if(!value.endsWith("@dss.com")){
                                value=value+"@dss.com";
                        }
                }
                return value;
        }
}
```

MyServlet.java:-

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.GenericServlet;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class MyServlet extends GenericServlet {

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

                response.setContentType("text/html");
                PrintWriter out=response.getWriter();
                String uname=request.getParameter("uname");
                String upwd=request.getParameter("upwd");
                String email=request.getParameter("email");
                String mobile=request.getParameter("mobile");
                out.println("<html><body bgcolor='lightyellow'>");
                out.println("<b><font size='7'><br><br>");
                out.println("Name......"+uname+"<br><br>");
                out.println("Password......"+upwd+"<br><br>");
                out.println("Email......"+email+"<br><br>");
                out.println("Mobile......"+mobile);
                out.println("</font></b></body></html>");

        }
}
```

responsewrapperapp1:-

web.xml:-

```
<web-app>
<display-name>responsewrapperapp1</display-name>
<filter>
<filter-name>MyFilter</filter-name>
<filter-class>MyFilter</filter-class>
</filter>
<filter-mapping>
<filter-name>MyFilter</filter-name>
<url-pattern>/wrapper</url-pattern>
</filter-mapping>
<servlet>
<servlet-name>MyServlet</servlet-name>
<servlet-class>MyServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>MyServlet</servlet-name>
<url-pattern>/wrapper</url-pattern>
</servlet-mapping>
</web-app>
```

MyFilter.java:-

```
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MyFilter implements Filter {
```

```
public void init(FilterConfig fConfig) throws ServletException {}

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
throws IOException, ServletException {

                HttpServletRequest httpRequest=(HttpServletRequest)request;
                HttpServletResponse httpResponse=(HttpServletResponse)response;
                MyResponse myResponse=new MyResponse(httpResponse);
                chain.doFilter(httpRequest, myResponse);
        }
        public void destroy() { }

}
```

MyResponse.java:-

```
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServletResponseWrapper;

publicclass MyResponse extends HttpServletResponseWrapper {

        HttpServletResponse httpResponse;

        public MyResponse(HttpServletResponse httpResponse){
                super(httpResponse);
                this.httpResponse=httpResponse;
        }
        publicvoid setContentType(String type){
                if(!type.equals("text/html")){
                        httpResponse.setContentType("text/html");
                }
        }
}
```

MyServlet.java:-

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MyServlet extends HttpServlet {

        protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                response.setContentType("img/jpg");
                PrintWriter out=response.getWriter();
                out.println("<h1>Hello....... from MyServlet</h1>");
        }
}
```

responsewrapperapp2:-

reverseform.html:-

```
<html>
<bodybgcolor="lightgreen">
        <formmethod="get"action="./wrapper">
                <center><b><fontsize="6"><br><br><br>
                        Enter Text : <inputtype="text"name="text"/>
                        <br><br>
                        <inputtype="submit"value="Reverse"/>
                </font></b></center>
        </form>
</body>
</html>
```

web.xml:-

```
<web-app>
<display-name>responsewrapperapp2</display-name>
<welcome-file-list>
<welcome-file>reverseform.html</welcome-file>
</welcome-file-list>
<filter>
<filter-name>MyFilter</filter-name>
<filter-class>MyFilter</filter-class>
</filter>
<filter-mapping>
<filter-name>MyFilter</filter-name>
<url-pattern>/wrapper</url-pattern>
</filter-mapping>
<servlet>
<servlet-name>MyServlet</servlet-name>
<servlet-class>MyServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>MyServlet</servlet-name>
<url-pattern>/wrapper</url-pattern>
</servlet-mapping>
</web-app>
```

MyFilter.java:-

```
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;


import javax.servlet.http.HttpServletResponse;

public class MyFilter implements Filter {

   public MyFilter() {

   }
   public void init(FilterConfig fConfig) throws ServletException {

        }
        public void destroy() {

        }
        public void doFilter(ServletRequest request, ServletResponse response, FilterChain
chain) throws IOException, ServletException {
                HttpServletRequest httpRequest=(HttpServletRequest)request;
                HttpServletResponse httpResponse=(HttpServletResponse)response;
                MyResponse myResponse=new MyResponse(httpResponse);
                chain.doFilter(httpRequest, myResponse);
        }

}
```

MyResponse.java:-

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServletResponseWrapper;


publicclass MyResponse extendsHttpServletResponseWrapper{
        HttpServletResponse httpServletResponse;
        public MyResponse(HttpServletResponse httpServletResponse){
                super(httpServletResponse);
                this.httpServletResponse=httpServletResponse;
        }
        public PrintWriter getWriter() throws IOException{
                PrintWriter out=httpServletResponse.getWriter();
                MyWriter myWriter=new MyWriter(out);
                return myWriter;
        }
}
```

MyWriter.java:-

```
import java.io.PrintWriter;

publicclass MyWriter extends PrintWriter {
        PrintWriter out;
        public MyWriter(PrintWriter out){
                super(out);
                this.out=out;
        }
        publicvoid println(String data){
                if(!data.startsWith("<")){
                        StringBuffer sb=new StringBuffer(data);
                        out.println(sb.reverse());
                }else{
                        out.println(data);
                }
        }
}
```

MyServlet.java:-

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MyServlet extends HttpServlet {

        protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out=response.getWriter();
                String data=request.getParameter("text");
                out.println("<html><body bgcolor='lightblue'>");
                out.println("<center><b><font size='7' color='red'>");
                out.println("<br><br>");
                out.println(data);
                out.println("</font></b></center></body></html>");
        }

}
```

Servlet Listeners:

In GUI applications, when we click on a button, when we select an item in checkboxes, when we select an item in the list, choice boxes automatically the respective GUI components may raise the respective events.

In GUI applications, all the GUI components are capable of raising the events only, they are not capable of handling the events.

In the above context, to handle the events all the GUI components will bypass the generated events to a separate implicit component called as **Listener.**

The main role of Listener in GUI applications is to listen an event from the respective GUI

component, handle it by executing Listener methods and send back the response to GUI application.

The above process in GUI applications is treated as **Event Handling** or **Event Delegation Model.**

In GUI applications, all the events are represented by some predefined classes and all the Listeners are represented in the form of interfaces.

Similarly in web application execution, container may generate events at the time of creating request object, adding an attribute, replacing an attribute, removing an attribute and destroying request object.

In this context, to listen the events generated by the container and to handle that events Servlet API has provided a set of interfaces called as Servlet Listeners.

Therefore, the main purpose of **Servlet Listeners** is to read all the life cycle stages of request object, ServletContext object and HttpSession objects.

In web applications, to perform event handling Servlet API has provided the following predefined library.

Where ServletRequestListener, ServletContextListener and HttpSessionListener can be used to read the life cycle stages like creation, destruction of ServletRequest object, ServletContext object and HttpSession object.

Where ServletRequestAttributeListener, ServletContextAttributeListener and HttpSessionAttributeListener can be used to read the life cycle stages of request object, context object and session object like adding a attribute, replacing an attribute and removing an attribute.

Where HttpSessionBindingListener can be used to read the life cycle stages of HttpSession object like adding HttpSessionBindingListener implementation class object reference to HttpSession object and eliminating HttpSessionBindingListener implementation class object reference from HttpSession object.

Where HttpSessionActivationListener can be used to read the life cycle stages of HttpSession object like participating HttpSession object in Serialization process of Marshalling and participating HttpSession object in Deserialization process of Unmarshalling in RMI applications i.e. Distributed applications.

If we want to use Listeners in our web applications then we have to use the following steps.

Step 1: Prepare Listener class.

Here take one user defined class, it must be an implementation class to Listener interface.

Ex: public class MyListener implements Listener {-----}

Step 2: Configure Listener classin web.xml file.

To configure Listener class in web.xml file we have to use the following xml tags.

```
<web-app>

              -------------

<listener>

<listener-class>Fully Qualified name of Listener class</listener-class>

</listener>

              -------------

</web-app>
```

The above Listener configuration is required for all Listeners except HttpSessionBindingListener and HttpSessionActivationListener.

listenerapp:-

web.xml:-

```
<web-app>
<display-name>listenerapp</display-name>
<listener>
<listener-class>HitCountListener</listener-class>
```

```
</listener>
<servlet>
<servlet-name>MyServlet</servlet-name>
<servlet-class>MyServlet</servlet-class>
</servlet>



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

HitCountListener.java:-

```
import javax.servlet.ServletContext;
import javax.servlet.ServletRequestEvent;
import javax.servlet.ServletRequestListener;

public class HitCountListener implements ServletRequestListener {
        int count=0;
         public void requestInitialized(ServletRequestEvent e) {
               System.out.println("Request Object Created");
        }
public void requestDestroyed(ServletRequestEvent e) {
count=count+1;
ServletContext context=e.getServletContext();
context.setAttribute("count", count);
System.out.println("Request Object Destroyed");
}

}
```

MyServlet.java:-

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MyServlet extends HttpServlet {

        protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out=response.getWriter();
                ServletContext context=getServletConfig().getServletContext();
                out.println("<center><h1>Hit Count is....."
+context.getAttribute("count")+"</h1></center>");
      }

}
```

SERVLETS INTERVIEW QUESTIONS:

1.In How many ways we can develop a servlet?

2.What is the difference between CGI & Servlet?

3.What is the difference between ServletConfig & ServletContext?

4.How to define a Servlet?

5.Specify parallel technologies to the servlet?

6.What is the difference between web server & web container?

7.What is the difference between web server & application server?

8.What are the various types of web containers?

9.By using which 2 Packages we can implement Servlet?

10.What is the purpose of RequestDispatcher?

11.What methods we can call on RequesrDispatcher Object?

12.Explain about SingleThreadModeal?

13.By using which interfaces we can implements Filter concept?

14.What are the various listeners are available in Servlet specification?

15.What are the various Event classes present in Servlet Specification?

16.By using which object we can send text data as response from the Servlet?

17.By using which object we can read binary data send by the client?

18.By using which object we can send binary data send as response from the Servlet?

19.What are the (various) lifecycle methods of the Servlet?

20.Explain the purpose of the init() method & How many times it will be excuted? When it will be executed?

21.If init() method throws any Exception i.e; if init() method fails to execute then what will be happen?

22.Is it possible to write constructor with in the Servlet?

23.Why init() is required to perform initialization activities instead of constructor?

24.Is it possible to perform Initialization activities with in the constructor?

25.What is the purpose of service() & How Many times it will be Executed?

26.Explain about destroy?

27.Is it Possible to call destroy() method explicitly from service?

28. With in the <servlet-mapping> how many url pattrerns taken at a time?

29.Explain LifeCycle of the Servlet?

30.What is the purpose of <load-on-startup>?

31.What is the significance of the number for <load-on-startup>?

32.If two servlets having same <load-on-startup>value then which wii be loaded first?

33.Explain about GenericServlet?

34.Which interfaces are implemented by GenericServlet?

35.What is the necessity of having 2 init() methods with in the Servlet?

36.Explain best way of overriding init()?

37.What are various possible status code of response?

38.Explain the difference between GET&POST?

39.What are various HttpRequest methods?

40.What is the difference between HEAD&GET?

41.What is the difference between PUT&GPOST?

42.Which Http methods are non-idempotent?

43.Which Http methods are idempotent?

44.What is the default method for the form?

45.How many service() methods available in HttpServlet?

46.Is it recommended to override service() in Http based Servlet?

47.If you are sending Get request but our Servlet contains doGet() & service() Methods then which method will be executed?

48.If you are sending Get request but our Servlet doesn't contain doGet() what happen?

49.Even though HttpServlet doesn't contain any abstract method why it is declared as abstract class?

50.What are the various methods to retrieve from parameters?

51.What is the purpose of request Headers?

52.How we can retrieve headers associated with the ServletRequest?

53. To what value is a variable of the String type automatically initialized?How we can retrieve cookies from the request?

54.By using which method we can get client &server information from the request?

55.How we can add response headers to the ServletResponse?

56.How we can set ContentType to the response?

57.What is the MIME type?

58.Is it possible to send multiple content type as response from the same servlet?

59.Write Servlet code to send movie file as response?

60.Is it possible to get PrintWriter & ServletOutputStream objects simultaneously?

61.How we can implement Redirection mechanism?

62.Explain, difference between sendRedirect&forword?

63.How many times we can call sendRedirect() method with in the same Servlet?

64.How we can add cookies to the response?

65.Explain the directory structure of a web application?

66.In which location we have to place static content?

67.Is it possible to access resources present in the context root directly?

68.Is WEB-INF folder mandatory for web application?

69.Explain about web.xml?

70.Where we have to place 3rd party jar files?

71.If the required class file available in classes folder and lib folder jar file, then which one will get preference?

72.Is there any alternate location to place servlet .class file Other than classes folder?

73.Is it possible to access web.xml directory?

74.Where we have to place tag libraries inside web application?

75.Is it important the order of tags in the web.xml?

76.Can you specify any 10 tags of web.xml?

77.With in the <web-app> which tags are mandatory?

78.What is the purpose of <servlet-name>

79.How many names are possible for a servlet in web-app?

80.Is it possible to configure jsp's in web.xml?

81.When we have to configure jsp's in web.xml?

82.What is the purpose of Servlet initialization parameters and explain how to configure in web.xml?

83.With in the servlet how we can access logical name of the servlet?

84.With in the servlet how we can access Servlet initialization parameter?

85.What is the ServletConfig object and explain the methods available in ServletConfig interface?

86.What is the purpose of <load-on-startup>explain its advantages & disadvantages?

87.If two Servlets having same<load-on-startup> values what will be happen?

88.How many types of url patterns are possible according to Servlet specification?

89.With in the <servlet-mapping>how many url pattrern tags we can take?

90.What is the difference between url, uri & urn?

91.How we can get Contextpath and quereyString directly with in the Servlet?

92.What is the necessity of welcome-file and explain How to configure welcome files in web.xml?

93.What is the default welcome-file?

94. Is it possible to configure welcome file in folder wise?

95.What is the necessity of error page and explain the process of Configuration in web.xml?

96.How to send ErrorCode programmatically?

97.What is the purpose of <mime-mapping>?

98.Explain about of war file & the process of creation?

99.What is the purpose of META-INF folder?

100.Explain about MANIFEST.MF?

101.Explain about <context-param>tag &<init-param>?

102.What are the differences between Servlet initialization parameters& Context initialization parameters?

103.How we can access context parameters and servlet parameters with in the Servlet?

104.How we can declare context & Servlet <init-parameters>in web.xml?

105.What is the scope of <context-param>&<init-param>?

106.What are the difference between parameters & attributes?

107.What is the purpose of an attribute?

108.What are various scopes available for Servlets?

109.Explain the cases where we should go for each scope?

110.explain the methods to perform following activities?

a) Adding an attribute?

b) Get the name of an attribute?

c) Remove an attribute?

d) Modify the value of an attribute?

e) To display all attribute names present in a specified scope?

111.If we store information in application scope by default it is available everywhere with in the web application, then what is the need of session and Request scopes?

112.What is the purpose of RequestDispatcher?

113.How many possible ways we can get the RequestDispatcher?

114.What is the difference between obtaining RequestDispatcher from ServletRequest and ServletContext object?

115.What is the difference between forward() &include()?

116.What is the difference between forward() &sendRedirect()?

117.What is the foreign RequestDispatcher & explain the process how we can get it?

118.What are various attributes added by web container while forwarding & including? What is the purpose of these attributes?

119.What is the purpose of filter? Explain the cases where exactly required?

120.By using which interfaces we can implement filter concepts?

121. What is the purpose of FilterChain?

122.How we can configure filter in web.xml?

123.In how many ways we can map a filter?

124.In filter chain in which order the filters will be executed?

125.What is the difference between Filter interface doFilter() & FilterChain doFilter()?

126.What is the purpose of wrapper?

127.Explain the types of wrappers?

128.Explain the cases where you used filters & wrappers in your previous project?

129.What is the need of Session Management? Explain cases where Session Management is required with example?

130.What are the various Session Management techniques are available?

131.Explain the process of creating Session object?

132.What is the difference between getSession() & getSession(boolean b)?

133.Explain with an example where getSession(false) is required?

134.How we can invalidate a Session?

135.Define Session management?

136.How we can configure SessionTimeout in web.xml?

137.What is the difference between <Session-timeout>and <SetMaxInactiveInterval>?

138.How to know SessionCreation time & lastaccesed time?

139.Explain the Session Management Mechanism by SessionAPI?

140.What is the default session timeout?

141.Explain Session Management by using Cookies?

142.How the SessionId is exchanging between Client & Server?

143.What is the difference between Session API & Cookie which approach is recommended?

144.What is the difference between persistent and non persistent cookies?

145.What is URL Rewriting?

146.By using which Methods we can implement URLRewriting?

147.BY using which methods we can identify under laying Session Management technique?

148.Explain advantages & disadvantages of

cookies

URL Writing

Session API

149.Explain the purpose of Listener?

150.What are the various listeners are available according to Servlet specification?

151.What is the difference between ServletRequestListener and ServletRequestAttributeListener?

152.How to configure listener in web.xml?

153.To print hit count of the web application which listener is required to use?

154.To print the no of active session objects at server side which listener is responsible?

155.What is the difference between HSAL & HSBL?

156.What are various methods present in binding listeners?

157.Explain about HttpSessionActivationListener?

158.At the time of application deployment to perform certain activities which listener is responsible?

159.What are various listeners which are not required to configure in web.xml?

160.What are various event classes available in ServletSpecification?

161.Is it possible to configure more than one listener of same type?

162.If you are configures more than one listener of the same type then in which order listener will be executed?

163.Who is responsible to perform instantiation of listener?

164.At what time listener classes will be instantiated?

165.What is the difference between declarative Security & programmatic security?

166.By using which methods we can implement programmatic Security?

167.What is the purpose of <security-role-ref>?

168.How we can configure users in Tomcat?

169.In your previous project how you implement security?

170.In your previous project what type of authentication used?