Servlets

Server :-

Server is a Software which manages all the resources along with which proces the client request and sends the client request.

The different types of servers are

1) Database Server
2) Application Server
3) web server.

1) Data base Server :-

It deals with data.

Eg :- Oracle, MySql, Duby, IBMdb2, MongoDB, Sybase, Informix etc.

2) Application Server :-

JBoss, IBM websphere, Oracle web logic.

It is used to execute Dynamic or real time application.

3) web Server :-

Apache Tomcat, Oracle Glayfish.

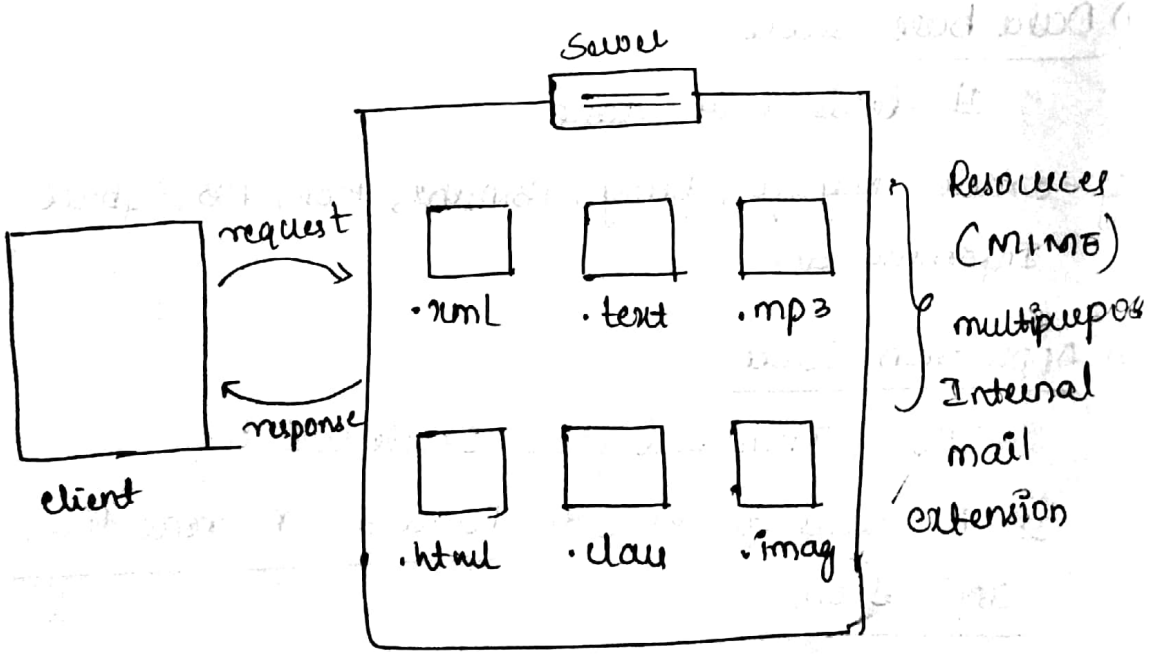
web server :- web server is a software which store and manages all the resources along with which proces the client request & sends the client request.

Resource formats

1) audio → mp3
2) Video → quick time
3) html → text
4) xml → text
5) application | zip.

web server

~~web2~~ → Apache tomcat
→ Oracle Glassfish.



Deployment :-

Making all the resources available to the server is known as deployment.

There are 2 different types of deployment

1) Manual
2) Automatic.

1) Manual Deployment :-

In case of manual deployment all the resources are made available to the server manually.

2) Automated Deployment :-

In case of automated deployment all the resources are made available to the server automatically with the help of automated tools such as ANT, MAVEN etc

→ whenever a web application is compressed, then we display it on to the server in the form of war file.

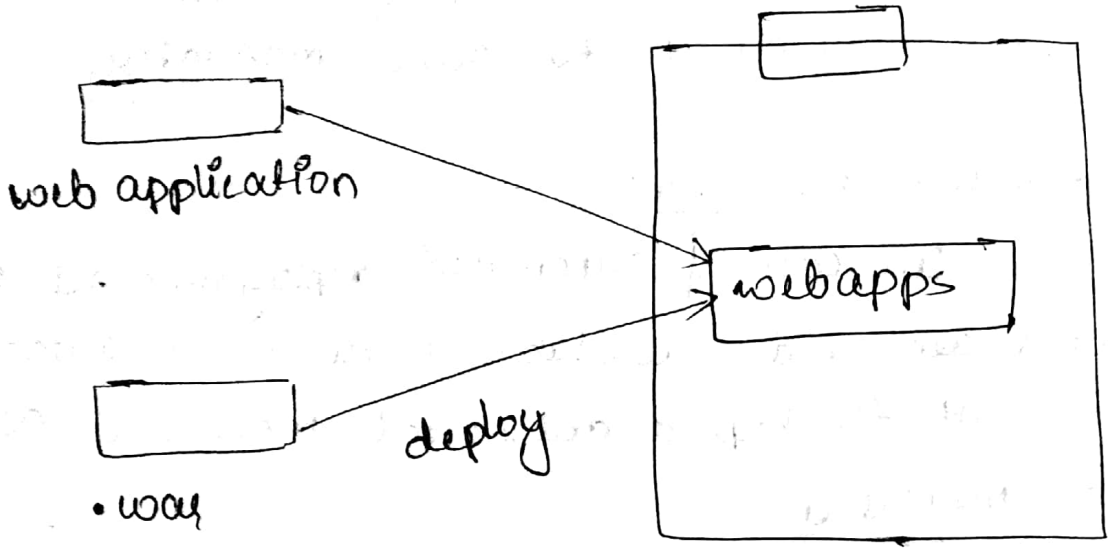
→ war refers to web archive that means it is a compressed version of web application.

** * All the web applications are deployed in to the webApps folder of apache tomcat server

Note:- we can directly deployed a web application or create a war file and deploy it on to web.apps folder of apache tomcat server.

* Apache tomcat server

web server.



→ Apache Tomcat server comes in to two different varients namely.

1) exe varient
2) zip varient

** Note:-

By default, the port number for Apache tomcat server in 8080.

Folders of Apache tomcat server:-

bin
conf
lib
logs
webapps
work.

1. bin :- It contains a set of start-up and shut-down batch files which is used to start & stop the Apache-tomcat server.

2) conf :-
It contains a set of configuration with respect to Apache-tomcat server.

3) lib :-
It contains a set of libraries in the form of jar file which is used to perform some additional functionalities.

4) logs :-
It is used to store all the log message which are displayed on the server console. since server-console has limited memory.

5) webapps :-
It is used to deploy all the web Applications on to Apache-tomcat server.

6) work :-
It is used to store the data with respect to translated servlets (conversion of JSP in to a servlet)

pre-requisites for Apache Tomcat Server :-
There are 3 different pre-requisites with respect to Apache tomcat server namely.

1) JAVA_HOME
2) CATALINA_HOME
3) Path
4) JRE_HOME [optional]

1) JAVA_HOME :-

Open C-drives, program files and open java and JDK folder and copy the entire path which includes all the folders with respect to JDK.

2) CATALINA_HOME :-

Open the Apache tomcat server folder from the respective directory and copy the entire path which includes all the folders with respect to Apache tomcat server.

3) Path :-

put a semicolon next to the existing path of system variables and copy paste the entire path which includes bin folder of Apache tomcat server

4) JRE_HOME :-

Open C-drive, program files and open java and JRE folder and copy the entire path which includes all the folder with respect to JRE

Note :-

It is mandatory to deploy all

home.html

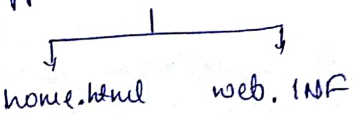
```
<html>
<body bgcolor = "yellow">
<h1> All are sleeping </h1>
</Body>
</html>
```

web.xml

```
<?xml version = "1.0" encoding = "UTF-8"?>
<web-app>
<display-name> First_proj </display-name>
<welcome-file-list>
<welcome-file> home.html </welcome-file>
</welcome-file-list>
</web-app>
```

Steps :-

1) Create a new Dynamic web project
2) Generate a web.xml → within WEB-INF
   a) Create Resource → home.html → web content (outside WEB-INF)
   b) Configure Resource → web.xml
   c) Deploy Resource → webapps → New folder



http://localhost:8080/first

New → DP →
Project → Java ∈ Roots → Des
html → web content → crtl n → html → Nav

home.html

```
<html>
<body bgcolor = "yellow">
<h1> All are awake since trainee is on time !! </h1>
</body>
</html>
```

web.xml

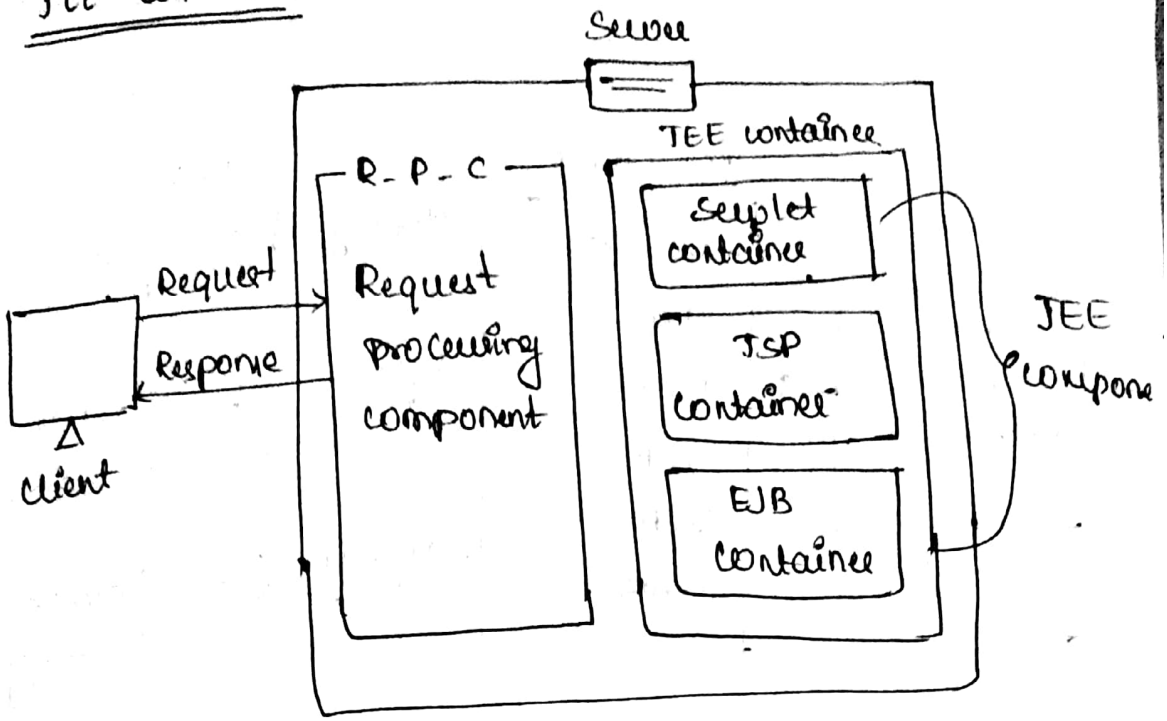
```
<?xml version = "1.0" encoding = "UTF-8" ?>
<web-app>
<display-name> New-proj </display-name>
<welcome-file-list>
<welcome-file> home.html </welcome-file>
</welcome-file-list>
</web-app>
```

Note:

(1) we cannot run a JEE application on all the servers

(2) To run a JEE application on a particular server, the server must contain a JEE container init.

JEE container



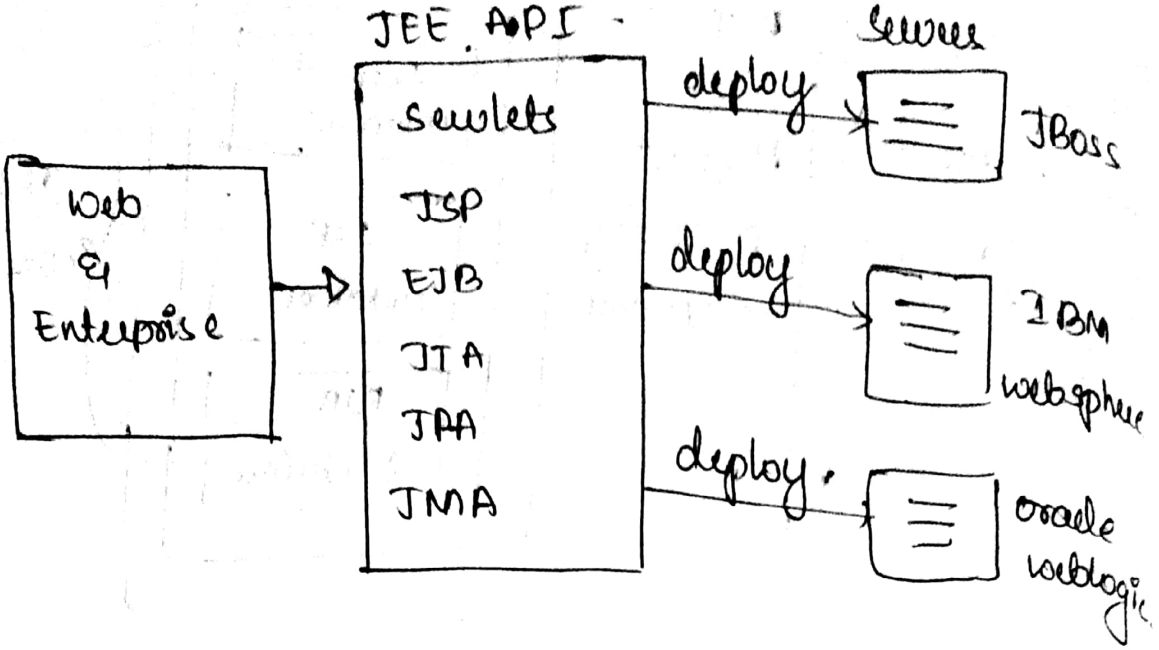
JEE container is an engine which is used to manage all the JEE components such as servlets, JSP, EJB etc...

→ Server basically perform 2 important task namely
  a) manage all the resources
  b) provides the runtym environment.

→ web server is used to execute only web appl
→ Application server is used to execute both web and enterprise application (JEE Application)

JEE application

It is a specification for developing web & enterprise appl which is given in the form of abstraction API to achieve loose coupling b/w Appl & Server.



→ few of the JEE API's are Servlets JSP, EJB, JTA, JPA, JMA.

JSP → Java Server Page

EJB → Enterprise Java Bean

JTA → Java transaction API

JPA → Java persistance API

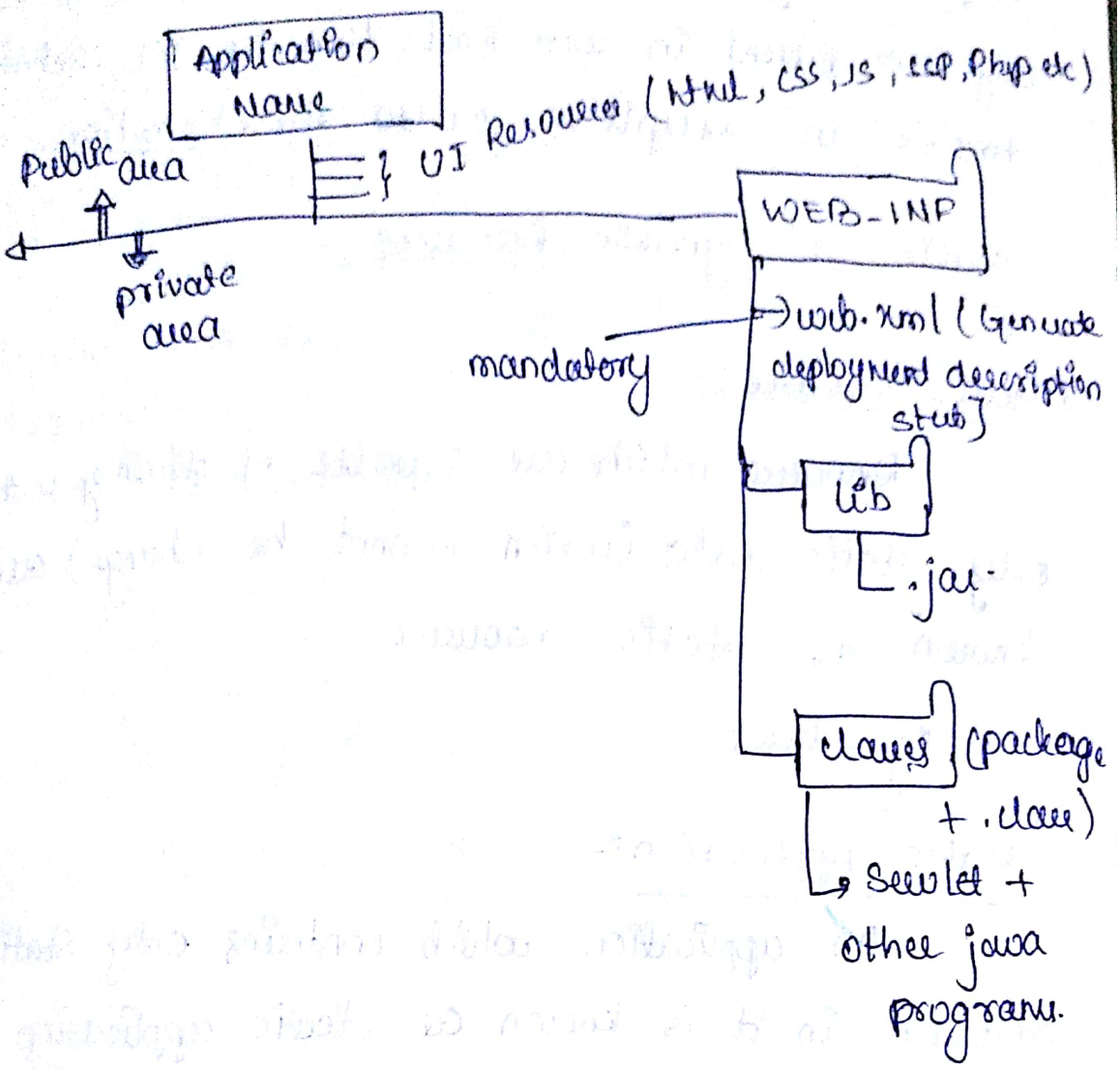
JMA → Java mail API

Note:-

Whenever we deploy any resource onto, the Server, the resource must be of type .class but not .java 1. Server is a machine which can understand ML

2. Servlet is a Independent technology

Structure of JEE Application:-



① Each & every application must have minimum & maximum of only one web.xml without which the JEE container fails to load an application. where it throws http 404 error (resources are not available)

⇒ we can deploy multiple applications onto one-single server but in this case, each & every application must have a unique name (or) different name.

⇒ whenever we start the server, all the applications are loaded sequentially one after the other by the JEE container.

⇒ At the time of application loading the web.xml is parsed (compile) by the JEE container where if there is any error present in web.xml, then the JEE container throws an exception called ParseException.

Static Vs Dynamic Resource :-

① Static resource :-

Resource which are capable of dealing with only static data (which cannot be change) are known as static resource.

Eg :- html

Static Application :-

An application which contains only static resource in it is known as static application

Eg :- stand alone application.

② Dynamic resource :-

Resource which are capable of dealing with only dynamic data (which can be changed) is known as Dynamic resource.

Eg :- Servlets, JSP, php etc

** Dynamic resource generally deals with the data base server & can perform 3 different types of logic namely

1) presentation logic
2) persistance logic
3) Business logic

Dynamic Application :-

An application which contains only dynamic resource in it is known as dynamic application.

eg:- Any real time applications

1. Presentation logic :-

It is used to present the contents on to the application.

technologies involved :- html, css, javascript, JSP, php, etc - - -

2) persistance logic :-

persist means to store. persistant logic is used to persist the data into the persistant system (database)

technologies involved :- JDBC, SQL, hypernet etc.

3) Business logic :-

It performs the core functionality that is some set of calculation and validation operations.

technologies involved :- servlets and other framework components.

Note:-

A dynamic application @ real time application is an integration of all the three different types of logic

welcome file or landing page:-

A file or page which is automatically displayed whenever a client uses an application is known as welcome file or landing page.

JOB ⇒

⇒ Index is considered as the default welcome file or landing page by JEE container.

⇒ we can explicity make a file to be welcome file or landing page by renaming it as Index.

⇒ Multiple files can be configured as welcome file or landing page but in this case the JEE container gives the priority based on the occurance

⇒ Note:-

If the configured files are not available then the JEE container fails to load an application where it throws http 404 error

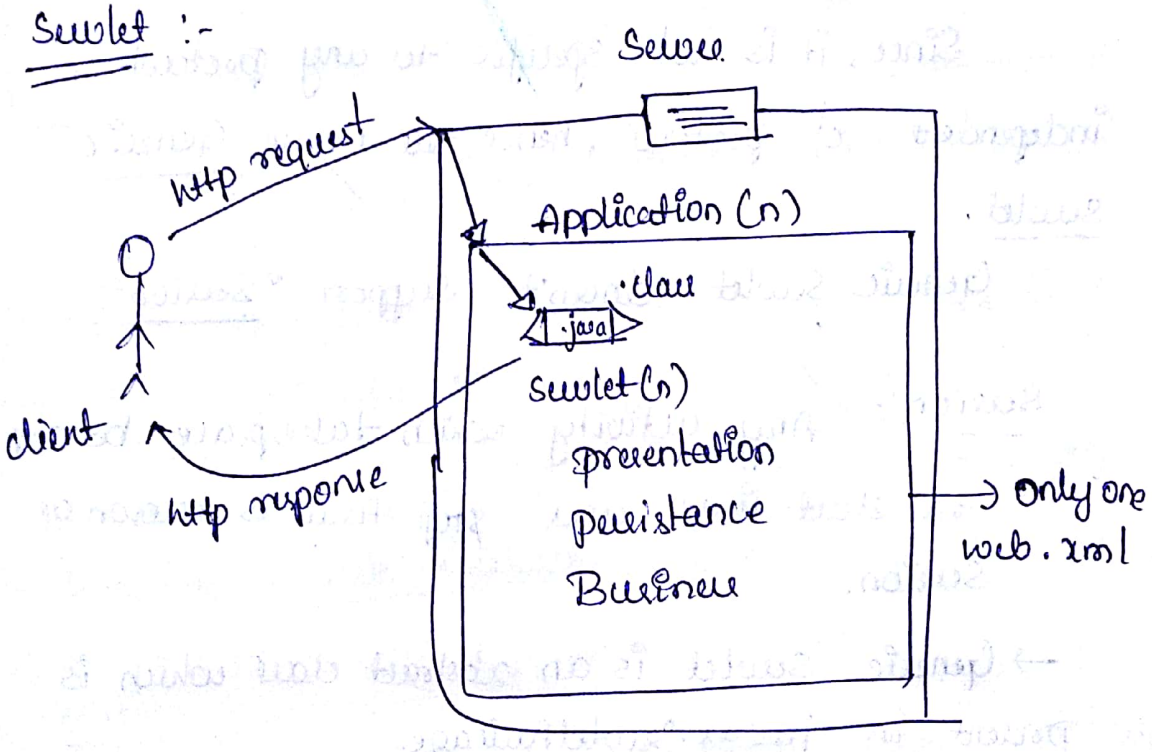
JOB How are all the configuration made in web.xml understood by the JEE container?

→ All the custom tags @ user defined tags are transformated @ converted into an 8-bit unicode

format based on which all the configurations made in web.xml are understood by the JEE container.

```
<html>
<meta charset = "ISO-8859-1">
<body bgcolor = "yellow">
<h1>
  Happy Balarid and vasavambalakshmi test !!
</h1>
</body>
</html>
```

Servlet :-



Servlet is a server side java program which can perform all the three different types of logic namely presentation logic, persistance logic and Business logic along with which process the http client request and get back some http response.

Note:-

→ All the applications within the server are managed by JEE container.

→ All the servlets within an application are managed by servlet container.

There are two different types of servlets namely.

1) Generic Servlet
2) Http Servlet

① Generic Servlet :-

Since, it is not specific to any protocol or independent of protocol, hence the name Generic servlet.

Generic servlet doesn't support "session".

Session:- Any activity which takesplace between the start time and stop time is known as session.

→ Generic servlet is an abstract class which is present in javax.servletPackage.

→ Generic servlet contains 3 different methods out of which one is an abstract method & other two are concrete methods.

→ The abstract method which is present in generic servlet is named as Service method which has to be mandatory overridden for

two important reason namely

1) since, service method is an abstract method

2) since, service method is the only method which takes two parameters called ServletRequest and ServletResponse which is responsible for processing the client request.

whenever we override service method it throws an exception called ServletException and IOException.

```
+ void service( ServletRequest req, ServletResponse resp)
                throws ServletException, IO Exception.
```

The other two methods present in GenericServlets are

1) init()
2) destroy()

where overriding this methods are optional since these are concrete methods

Writing a GenericServlet:-

write a servlet class which extends an abstract class called Generic Servlet as follows

Writing a

+ class OurServlet extends javax.servlet.GenericServlet
  ↳ servlet class ↳ abstract class
{
  @ Override
  + void service(ServletRequest req, ServletResponse res)
      throws ServletException, IO Exception
  {

  Impl logic

  }
}

2) Http Servlet :-

Since it is specific to http protocol, hence the name HttpServlet.

HttpServlet supports session

→ HttpServlet is an abstract class which is present in javax.servlet.http package

→ HttpServlets contains only concrete method without any abstract methods.

In case of HttpServlet, we have to override a respective concrete method called doXXX() for a particular type of HttpRequest.

There are 8 different types of HttpRequest present namely

1) Post
2) Get
3) Put
4) Delete
5) Trace
6) Option
7) Head
8) Connect.

Whenever we override doXxx(), it probably throws an Exception called ServletException and IOException.

void doXxx(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException.
↑
protected

writing a HttpServlet :-

write a servlet class which extends an abstract class called HttpServlet as follows.

+ class OurServlet extends javax.servlet.http.HttpServlet
  ↳ servlet class ↳ abstract class

{
  @ override
  # void doXxx (HttpServletRequest req, HttpServletRespons resp) throws ServletException, IO Exception
  {
    Impl logic
  }
}

Note:- Job:-

Q Is there a service method present in HttpServlet?

=> yes, service method in case of HttpServlet is present as a concrete method which is not a good practice to be overriden since, HttpServlet depends upon the type of HttpRequest.

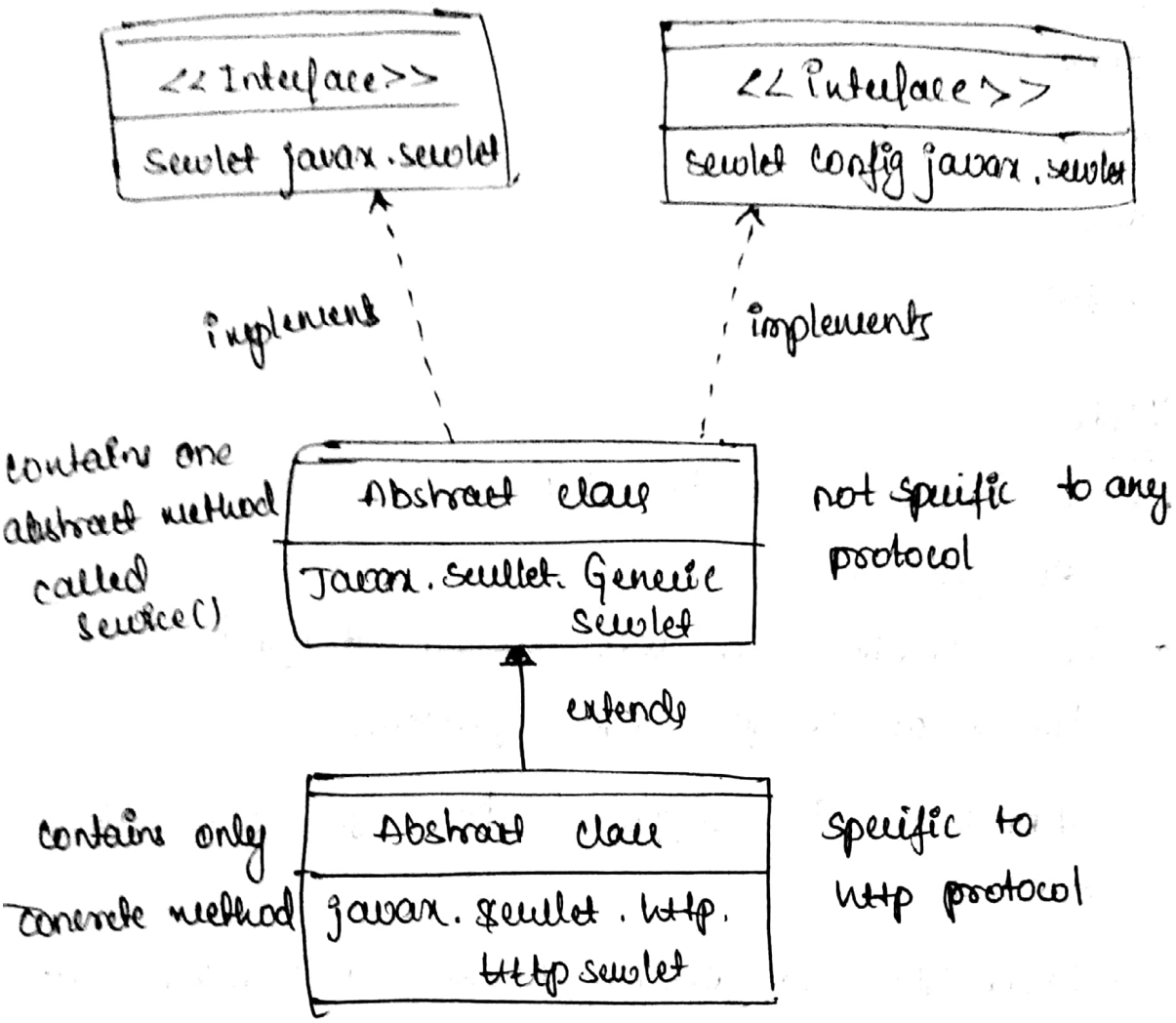
" Signature in case of Httpservlet

void service (HttpServletRequest req, HttpServlet Response resp) throws ServletException, IOException

JOB :- why is a class is made abstract inspite of having only concrete methods in it?

=> So that the further enhancement & modification made doesn't affect the user.

Servlet Hierarchy



Servlet :- methods are

init (servlet config)

service (servletRequest req, ServletResponse Resp)

destroy ()

getServletConfig ()

getServletInfo()

GenericServlet :- init()

ServletConfig :- getParameterNames()

getInitParameterNames()

getServletContext()

HttpServlet :-

service(HttpServletRequest req, HttpServletResponse resp).
doXxx(
doPost(
doGet(
doPut(

Note :-

→ web resource can be accessed based on unique URL pattern.

→ Since Servlet is a web resource it can be accessed based on unique URL pattern.

→ Annotation is supported from JEE 3.x version onwards

→ From JEE 3.x version onwards it is not mandatory to configure a resource in web.xml. Instead annotation can be used.

→ We use 3.1 version → JEE using current version.

Life cycle methods of Servlet :-

JOB There are 3 different life cycle methods of servlet present namely :-

1) init (ServletConf)
2) service (ServletRequest req, ServletResponse resp)
3) destroy ()

Criteria for a Servlet :-

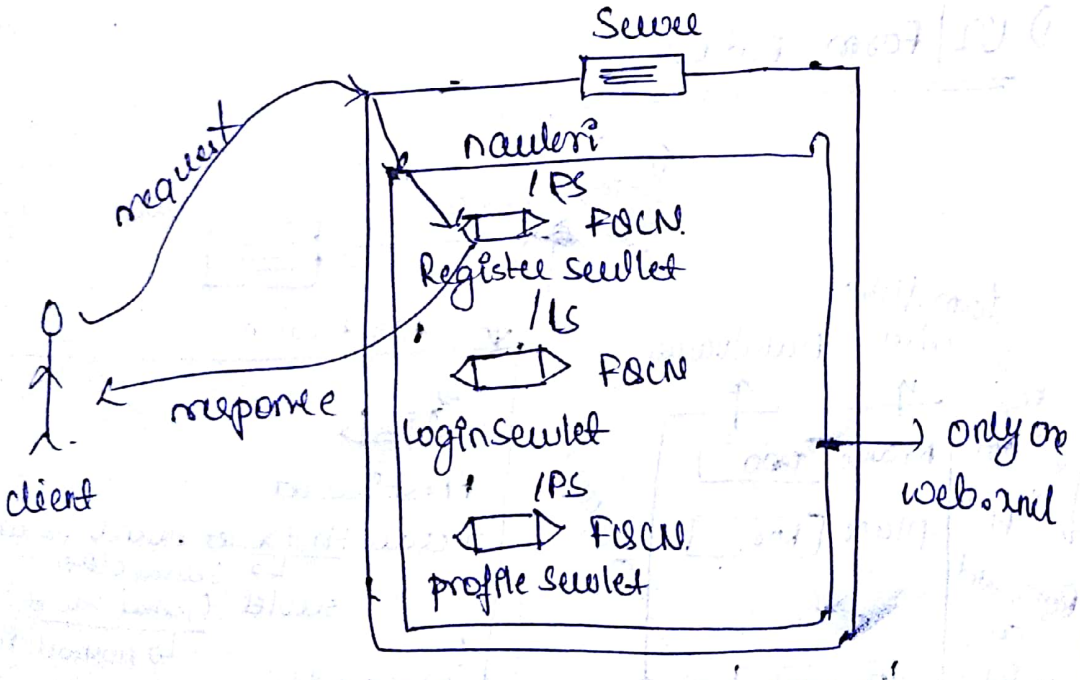
There are 3 different criteria

1) Create servlet
2) configure servlet
3) Deploy servlet.

Configuration of a Servlet in web.xml :-

Each and every servlet must be configure with three different properties in web.xml namely.

1) Servletname (unique)
2) URLpattern (unique)
3) Fullqualified classname (FQCN)



Syntax :-

```
<?xml version = "1.0" encoding = "UTF-8"?>
<web-app>
<servlet-mapping>
<servlet-name> UniqueName </servlet_name>
< URL-pattern > /URL </URL-pattern>
```

</servlet-mapping>

<servlet>

<servlet-name> unique Name </servlet-name>

<servlet-class> FQCN </servlet-class>

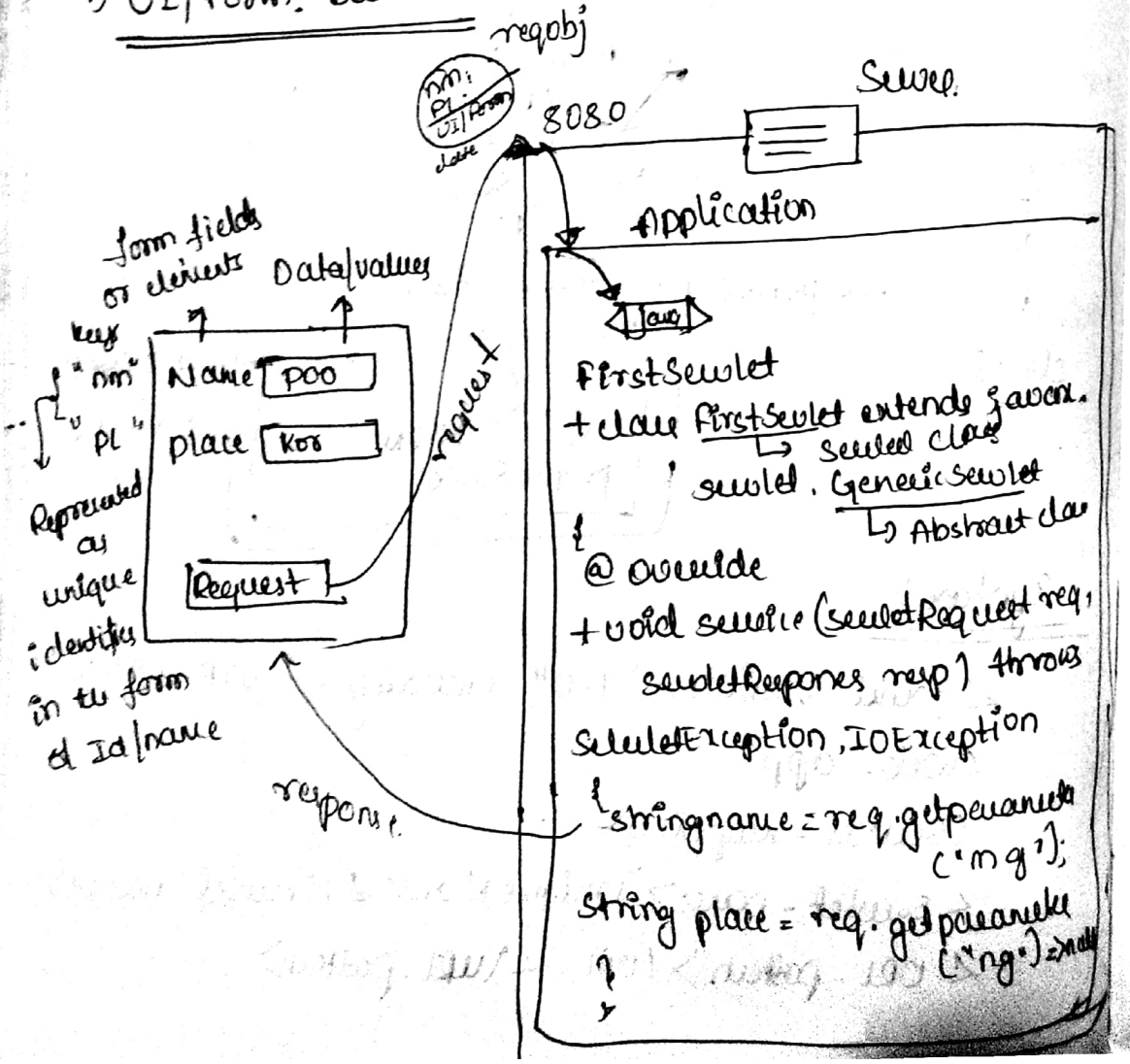
</servlet>

</web-app>

There are three different types of data present namely

1) UI / Form Data
2) Programmatic Data
3) Declarative Data.

1) UI / Form Data :-



→ The data which is entered by the enduser (client) on a form page and they submitted to the server in the form of key and value pair is known as UI/Form data.

→ All the keys associated with respect to the Form page are represented as unique identified in the form of Id/name

→ The UI/form data is always associated with a request and can be accessed only within the service()

→ Since service() is the only method which takes a parameter called ServletRequest and ServletResponse which is responsible for processing the client request

***
→ UI or form data can be fetched by using a method called getParameter()

+String getParameter(String key)

→ In this method, the key is taken as an argument

→ If the key is present then the method returns associated values

→ If the key is not present then the method returns null but not any Exception or any

JOB:- Note :-

The data's present in request object are UI/Form data

Steps

1) Create a new Dynamic Web Project
   a. Generate a web.xml [within WEB.INF]
   b. Add Servlet-api.jar into lib

2) Create Resource → form.html → webContent
   ↘ First Servlet.java → src

3) Config Resources → web.xml
   ↘ Annotation

4) Deploy Resource → webapps → New folder
   1) WEB.INF → lib, web.xml, class
   2) form.html

---

1) form.html

```
<html>
<body bgcolor = "cyan">
<form action = "fs">
Name: <input type = "text" name = "nm">
place: <input type = "text" name = "pl">
<br><br>
```

```
<input type = "submit" value = "request">
</form>
</body>
</html>
```

FirstServlet.java

```
package org.raj.UiApp
import java.io.*;
import javax.servlet.*;
public class FirstServlet extends GenericServlet
{
@Override
public void service(ServletRequest req, ServletResponse resp)
        throws ServletException, IOException
{
String name = req.getParameter("nm");
String place = req.getParameter("pl");
PrintWriter out = resp.getWriter();
out.println("<html><body bgcolour = "yellow">"
    + "<h1> Dabba fellow "+ name + " "+place+
    " </h1>" + "</body></html>");
out.flush();
out.close();
}
}
```

select:-java.io

factor | helpermethod

web.xml

```
<web-app>
<display-name> vi-proj </display-name>
<welcome-file-list>
<welcome-file> form.html </welcome-file>
</welcome-file-list>
<servlet-mapping>
<servlet-name> FirstServe </servlet-name>
<Url-pattern> /fs </url-pattern>
</servlet-mapping>
<servlet>
<servlet-name> Firstsew </servlet-name>
<Servlet-class> org.raj.viApp.FirstServlet
</servlet-class>
</servlet>
</web-app>
```

Annotation

Delete web.xml

1) form.html → index.html

2) before servlet class in FirstServlet
@ we ctrl+space
web servlet / * multiple)

code for a UI / Form data using Annotation configuration

index.html

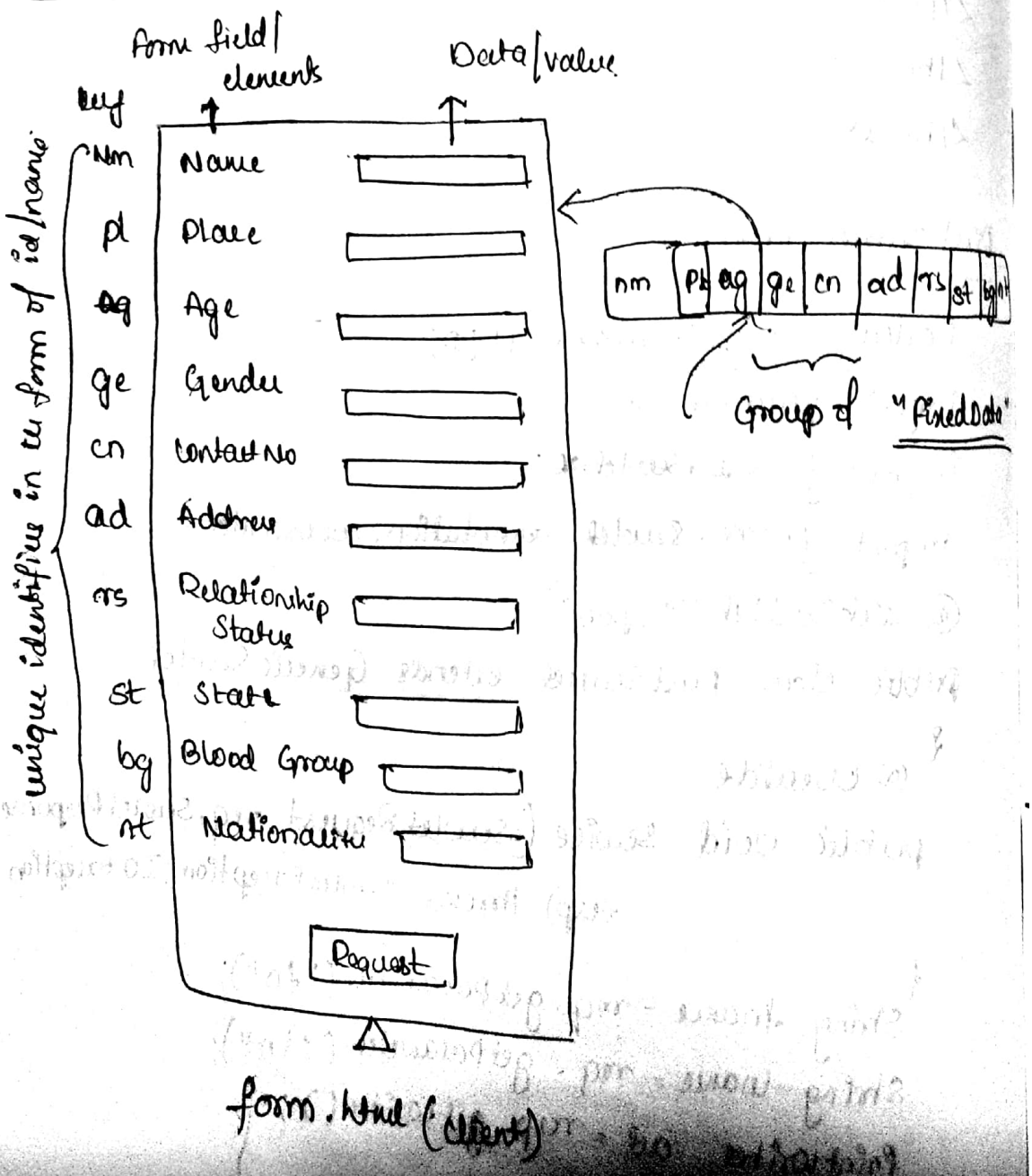
```
<html>
<body bgcolor = "cyan">
<form action = "poo">
First name : <input type = "text" name = "fn">
Last name : <input type = "text" name = "ln">
<br><br>
<input type = "submit" value = "request">
</form>
</body>
</html>
```

FirstServlet.java

```
Package org.jecm32.uiapp;
import java.io.*;
import javax.servlet.*;
import javax.servlet.annotation.webservlet;
@ WebServlet ("/poo")
public class FirstServlet extends GenericServlet
{
  @ Override
  public void service (ServletRequest req, ServletResponse
                       resp) throws ServletException, IOException
  {
  String fname = req.getParameter("fn");
  String lname = req.getParameter ("ln");
  PrintWriter out = resp.getWriter();
```

```
out.println("<html><body bgcolor = "yellow">"
+ "<h1> welcome "+ fname + " " + lname + "
</h1>" + "</body></html>");

out.flush();
out.close();
}
}
```

Enumeration:-



→ getParameterNames() is used to fetch all the key associated with form page @ once.

→ getParameterNames() can be used when ever there are 'n' no of form fields @ form elements present on form page.

VVimp:- JOB

getParameterNames() returns an enumeration of String type which contains all the keys associated with Form page.

Definition for enumeration:

Enumeration is a group of fixed Data.

```
Enumeration<String> keynum = req.getParameterNames();
while (keynum.hasMoreElements())
{
  String key = keynum.nextElement();
  String val = req.getParameter(key);
}
```


SERV

SERVLET LIFECYCLE

* Servlet gets a life and starts its lifecycle only when a Servlet object is created

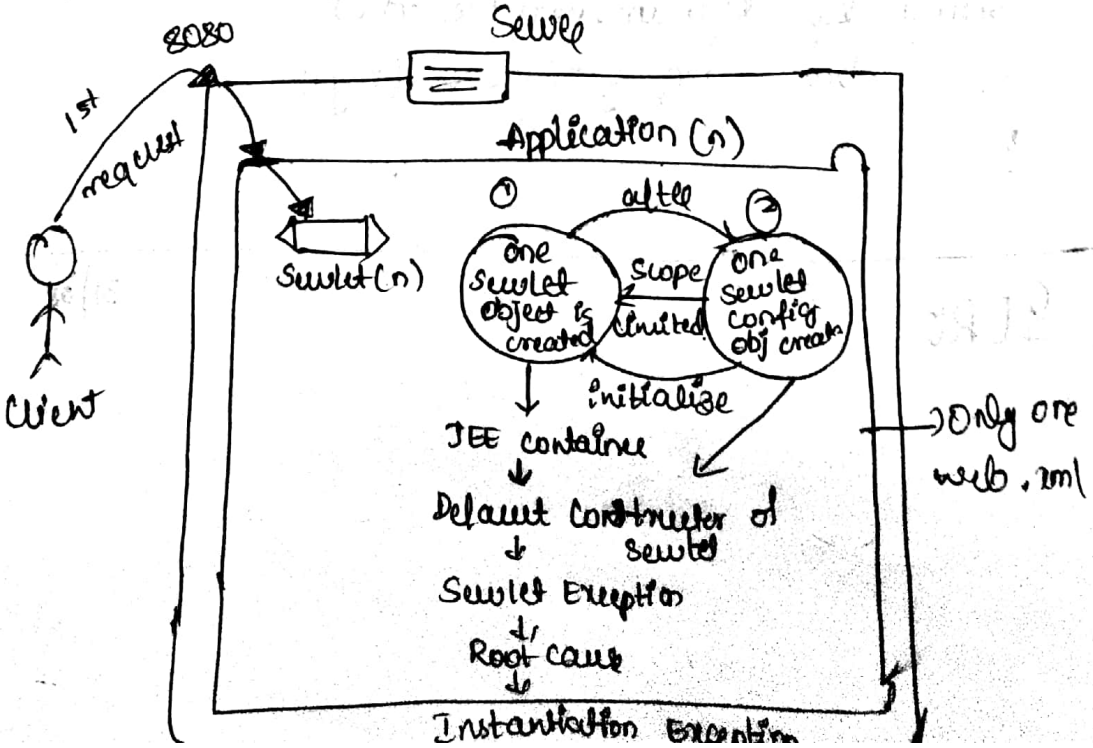
Servlet lifecycle depicts or represents the event or phase which takes phase from Servlet object creation until Servlet object destruction.

Note: Entire Servlet lifecycle is managed by JEE container [or] one of the responsibility of the JEE container is Servlet Lifecycle management.

There are 4 different lifecycle phase for a servlet namely :-

i) Instantiation | object creation phase
ii) Initialization phase
iii) Service phase
iv) Destruction phase

* **Instantiation | object creation phase:**



→ In this phase the Servlet object has to be created

→ whenever a client makes a first request to a Servlet, 1st Servlet object is created by the JEE container by calling default constructor of Servlet and now Servlet lifecycle begins.

→ If JEE container doesn't find the default constructor, then JEE container throws an exception called Servlet Exception with a root cause instantiation Exception [object not created].

→ Immediately after the Servlet object creation, one Servlet config object is created by JEE container which is used to initialize the resources of Servlet object.

→ Hence, the scope of Servlet Config object is always limited to that particular Servlet object.

Note:
SingleThreadedModel is a depricated Interface (outdated)

→ By default service() is multithreaded that means one thread is created for every client request and service() is executed.

→ whenever a client makes 2nd or subsequent client request to the same servlet again, only service() is executed by the servlet object is not created again.

→ service() is called by the JEE container for multiple times (or) 'n' no of times.

→ If this phase fails, then the JEE container throws an Exception called ServletException.
(Client request processing fails)

Destruction phase

In this phase, the destroy() is called to close all the costly resources but not to destroy the servlet object.

Vimp
→ destroy() is called by JEE container only once

Vimp:
→ If this phase fails, then the JEE container doesn't throw any error (or) Exception instead the performance of an appl° decreases

Note:- Servletobject can either be created or destroyed only by the JEE container but using

it's own implementation

Note:- ***

It is not a good practice to destroy servlet object since it is needed for further usage.

Situation of destroy()

→ There are 2 different situation in which destroy() can be called namely

a) whenever we close an application, destroy() is called to close all the costly resources w.r.to that application.

b) whenever we redeploy an application onto the server, destroy method is called to close all the previously used costly resources

Code for servlet lifecycle

~~web.xml~~ form.html

```
<html>
<body bgcolor = "cyan">
<form action = "fs">
Name : <input type = "text" name = "nm">
Place : <input type = "text" name = "pl">
<br><br>
<input type = "submit" value = "Request">
</form>
</body>
</html>
```

FirstServlet

```
package org.raj.lifecycleApp;
import java.io.*;
import javax.servlet.*;
public class FirstServlet extends GenericServlet
{
  public FirstServlet()
  {
   SOP("servlet object Is Created");
  }
  @override
  public void init(ServletConfig config) throws
                                ServletException
  {
   SOP("servlet object IS Initialised");
  }
  @override
  public void service(ServletRequest req, ServletResp
                                              resp) thro
                  ServletException, IOException
  {
   String name = req.getParameter("nm");
   String place = req.getParameter("pl")
   PrintWriter out = resp.getWriter();
out.println("<html><body bgcolor='yellow'>"
  +"<h1>Studen Details are "+name+" "+place+"
    </h1>"
    + "</body></html>");
```

```
out.flush();
out.close();
SOP("service() is Executed");
}
@Override
public void destroy()
{
  SOP("close all Costly Resources");
}
}
```

Web.xml

```
<?xml version = "1.0" encoding = "UTF-8"?>
<web-app>
<display-name> lifecycle-proj </display-name>
<welcome-file-list>
<welcome-file> form.html </welcome-file>
</welcome-file-list>
<servlet-mapping>
<servlet-name> FirstServ </servlet-name>
<url-pattern> /fs </url-pattern>
</servlet-mapping>
<servlet>
<servlet-name> FirstServ </servlet-name>
<servlet-class> org.raj.lifecycleApp.FirstServlet
</servlet-class>
</servlet>
</web-app>
```

Load-on-startup

→ Servlet gets a life & starts it's lifecycle only when Servlet object is created.

→ Servlet lifecycle can always begin in 2 different cases namely.

→ Whenever a client makes a 1st Request to a Servlet, one Servlet object is created by the JEE container by calling the default constructor of Servlet & now Servlet life cycle begins

II way in case of load-on-startup

→

→ <load-on-startup> is a tag which is a sub tag of <servlet> tag

→ In case of load-on-startup the JEE container create Servlet-object by calling the default constructor of Servlet at the time of Server Startup without waiting for the 1st client request. So that the delay time made by the 1st client request can be avoided which helps in increasing the performance of an application.

→ Load-on-startup must always be configured with a +ve Integer value. but the JEE container give the priority based on lowest positive Integer value.

Int → Whenever two Servlets are configured with the same +ve Integer value for load-on-startup, Sequential execution takeplace.

Note: JOB:
Whenever load on startup is configured with a negative integer value, then the JEE container ~~with~~ creates a Servlet object based on 1st client Request. But it doesn't throw any war @ Exception.

Code for Servlet Life Cycle In case of loadonstartup using xml configuration

```
<web-app>
<display-name> Lifecycle-Proj </display-name>
<welcome-file-list>
<welcome-file> form.html </welcome-file>
<welcome-file-list>
<servlet-mapping>
<servlet-name> Firstserv </servlet-name>
<url-pattern> /fs </url-pattern>
</servlet-mapping>
<servlet>
<servlet-name> Firstserv </servlet-name>
<servlet-class> org.raj.lifecycleApp.Firstservlet </servlet-class>
<load-on-startup> 2 </load-on-startup>
</servlet>
</web-app>
```

Note: In case of web.xml, the init or initialization parameter must mandatorily be declared inside the <servlet> tag since the scope of config object is limited only to that particular Servlet object.

Declaration of init (or) initialization parameter through annotation

```
@ WebServlet (name = "FirstServ", urlpattern = "/fs",
    init-params = {
@ WebInitParam (name = "key1" value = "value1"),
@ WebInitParam (name = "key2" value = "value2"),
@ WebInitParam (name = "key3" value = "value3")
    })
public class FirstServlet extends javax.servlet.
                                  GenericServlet
{

}
```

Fetching the init (or) initialization parameter

* The init (or) initialization parameter which is present in config object can be fetched by using a method called

getInitParameter method.

→ String getInitParameter(String key)
↳ Argument

* In this method, the key is taken as an argument.
* If the key is present, then the method returns associated value
* If the key is not present, then the method returns null but not any exception or error.

javax.servlet.ServletConfig config = getServletConfig();

Syntax:- config.getInitParameter(key)

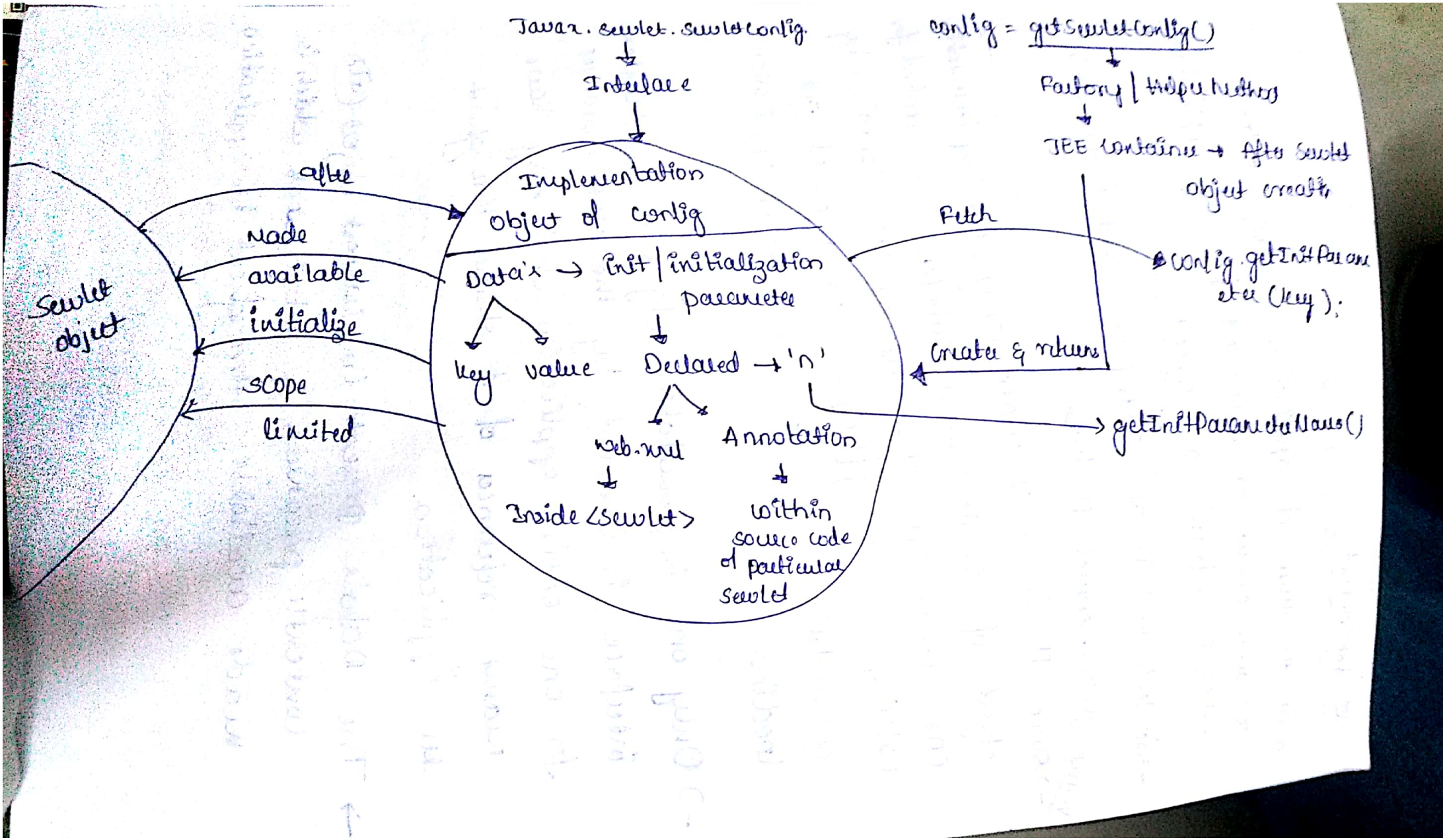
* getInitParameterNames() is used to fetch all the keys associated with the config object at once.
* getInitParameterNames() can be used whenever there are 'n' no of init (or) initialization parameters declared for a config object.

Note

getInitParameterNames() returns an enumeration of StringType which contains all the keys associated with Config object

+ Enumeration <String> keynum = config.getInitParameter
Names();

while (keynum.hasMoreElements())
{
String key = keynum.nextElement();
String val = config.getInitParameter(key);
}



ServletContext

→ At the time of application loading the JEE container creates an environment for every application which is referred as Context.

→ ServletContext is an Interface which is present in javax.servlet Package.

→ Since it is an Interface only, one implementation object of ServletContext interface is created for one entire application by the JEE Container by calling a factory/helper method called getServletContext() at the time of application loading

→ Only one implementation object of ServletContext interface is Created by the JEE container for one entire application which may have many references of it shared throughout the application.

→ The Data's present in context object (all) context parameters & attributes which is made available throughout the application.

Note: Hence the scope of Context object is through the application & context scope is also known as application scope.

→ The Data's present in Context object are known as application wide resources & application scope resources.

Context Parameters

→ It is the data in the form of key & value pair which is made available throughout the application.

→ 'n' number of context parameter can be declared for a context object

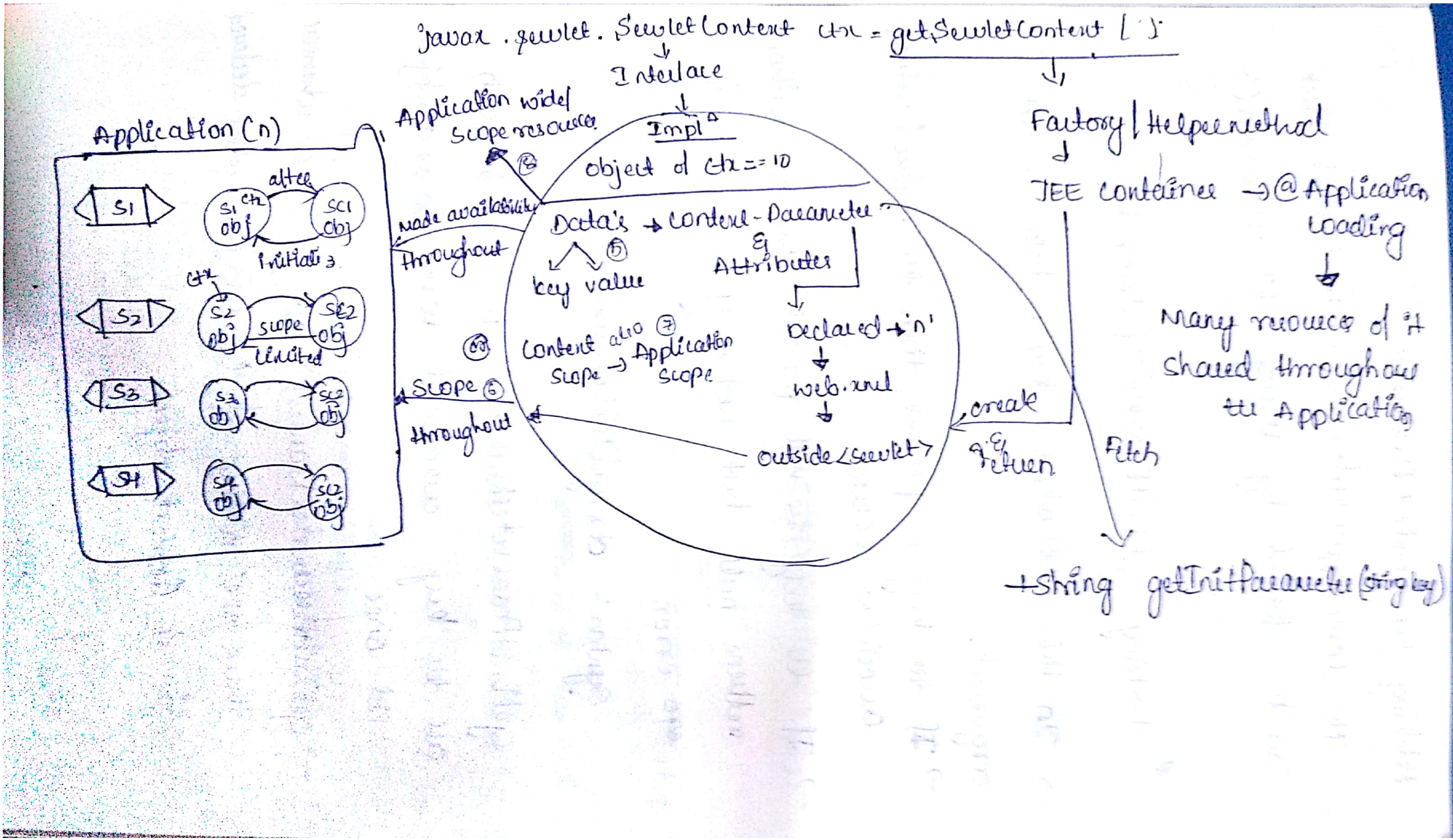
Imp Context parameter can always be declared only in web.xml.

① Declaration of context parameter in web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
<context-param>
<param-name> key1 </param-name>
<param-value> value1 </param-value>
</context-param>
```

```
<context-param>
<param-name>key2</param-name>
<param-value>value2</param-name>
</context-param>
<servlet>
</servlet>
</web-app>
```

Note:- In case of web.xml the context parameter must be mandatorily declared outside the <servlet> tag. Since the scope of context object is throughout the application.



Fetching the Context Parameter

The Context parameter present in Context object can be fetched by using getInitParameter()

&

+ String getInitParameter(String key)

→ In this method, the key is taken as an argument.

→ If the key is present, then the method returns associated value

→ If the key is not present, then the method returns null but not any exception or error.

Syntax:- ctx.getInitParameter(key);

→ getInitParameterNames() is used to fetch all the keys associated with to context object at once

→ getInitParameterNames() can be used whenever there are 'n' no of Context parameters declared for a Context object.

Note: Vinp

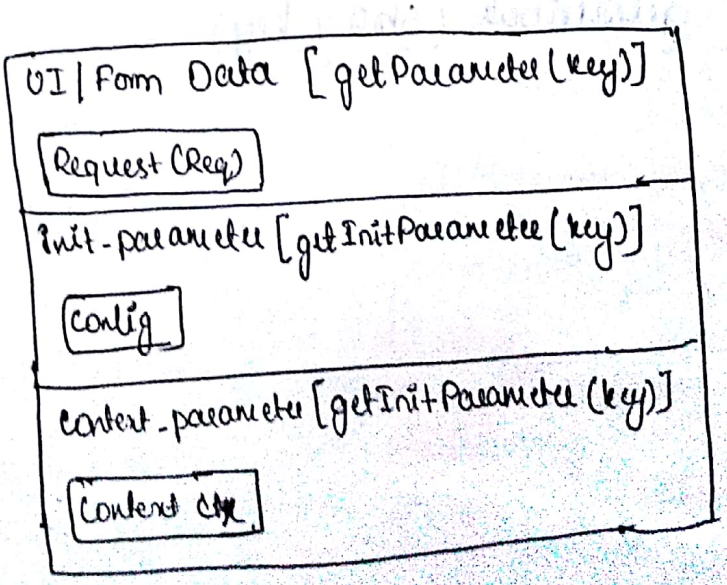
getInitParameterNames() returns an enumeration of String type which contains all the keys associated with the context object

```
→Enumeration<String> keynum = ctx.getInitParameter
                              Name();
while (keynum.hasMoreElements())
{ String key = keynum.nextElement();
  String val = ctx.getInitParameter(key);
}
```

11/9/18

Parameter:

Parameter is the data in the form of key and value pair where key must be unique string and the value can also be a string.



Attribute:

It is the data in the form of key & value pair where key must be unique string & the value can be any object.

eg:- student object, Array object, Bean object.

→ Attribute can be associated with request, session & context object.

→ Attribute cannot be associated with config object that means attribute is not applicable for config object.

→ Attribute can be added into the scope by using a method called setAttribute().

+ void setAttribute (String key, Object obj)

→ Attribute can be fetch back from the scope by using getAttribute()

+ Object getAttribute (String key)

product.html

```
<html>
<body bgcolor = "red">
<h1>
<a href = "ep" >Electronic Product </a><p></p>
<a href = "cp"> Cloth Product </a><p></p>
<a href = "bp" > BookProduct </a>
</h1>
</body>
</html>
```

product.java

```
Package org.jeem30.contextApp;
public class Product
{
  public String name;
  public double price;
}
```

ElectronicServlet.java

```
package org.jeem30.contextApp;
import java.io.*;
import javax.servlet.*;
import javax.servlet.annotation.WebServlet;
@WebServlet("/ep")
public class ElectronicServlet extends GenericServlet
{
  @Override
```

```
public void service (ServletRequest req, ServletRespons
    resp) throws ServletException, SQLException
{
 ServletContext ctx = getServletContext();
 String offmsg = ctx.getInitParameter("ofm");
 String offperc = ctx.getInitParameter("ofp");
 Product P = new Product();
 ctx.setAttribute("prdnm", P); // Add ctx obj into
                                  scope
PrintWriter out = resp.getWriter();
 out.println("<html><body bgcolour = 'Pink'>"
+"<h1> Electronic Product Details </h1><P></P>"+
" "+offmsg+" "+offperc+"% Off "+"<br> <anhf> =
  <a href=\"product.html\">Back </a>"+"</body>
    </html>");
 out.flush();
 out.close();
}
}
```

Cloth Servlet.java

—//—

Book Servlet.java

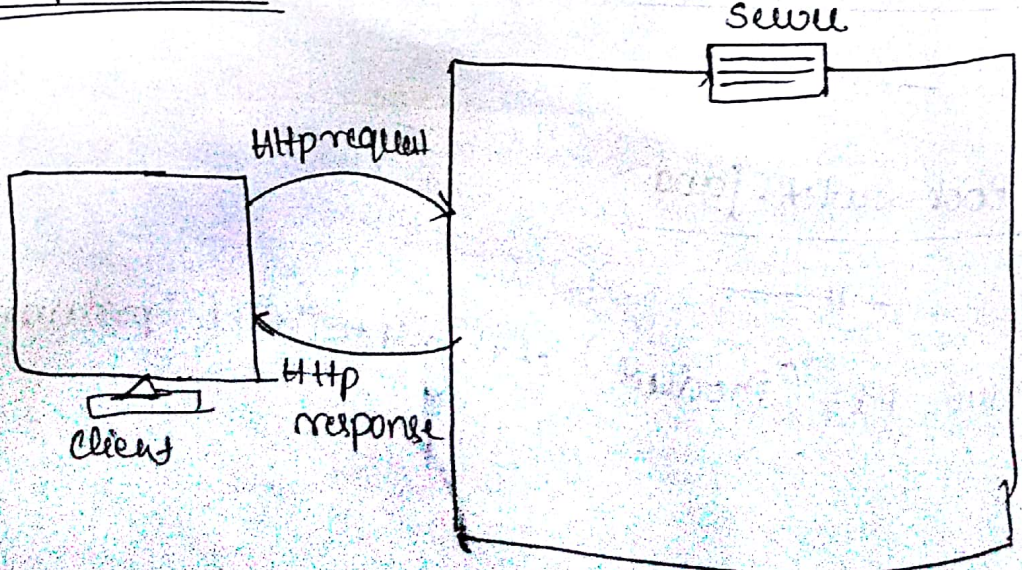
—//—

```
Product Pd = (Product) ctx.getAttribute("prdnm");
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
<display-name> context.proj </display-name>
<welcome-file-list>
<welcome-file> Product.html </welcome-file>
</welcome-file-list>
<context-param>
<param-name> efm </param-name>
<param-value> Rainy season Sale </param-value>
</context-param>
<context-param>
<param-name> efp </param-name>
<param-value> efm </param-value>
</context-param>
</web-app>
```

Http Servlet:-



→ ServletRequest → Interface → javax.servlet

contents of ServletRequest :

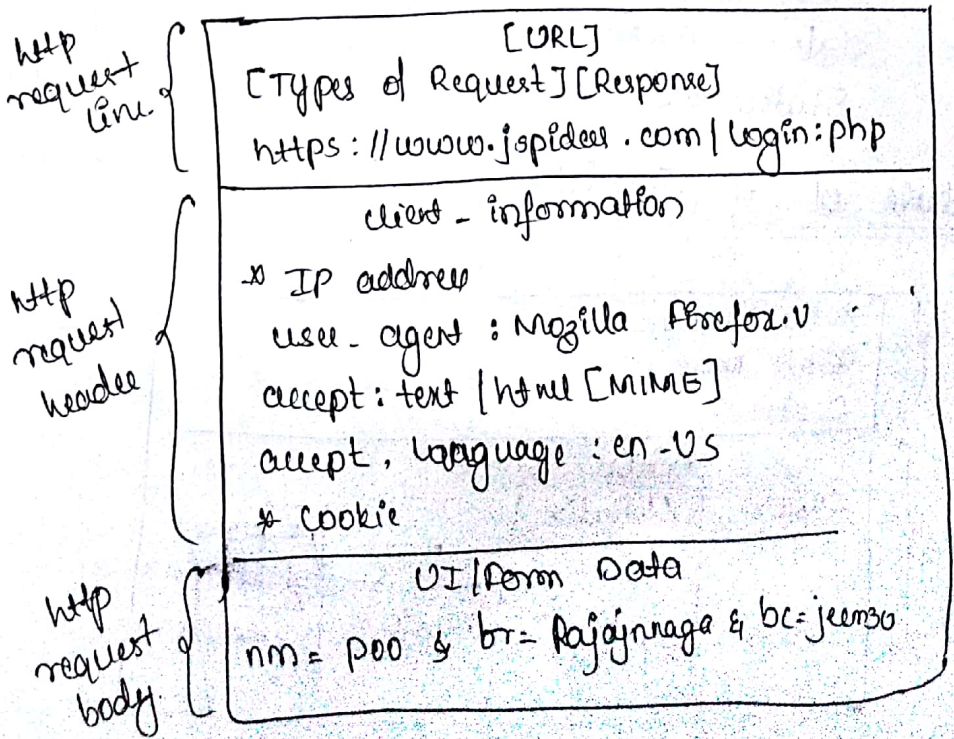
1) UI | Form data
2) URL

→ HttpServletRequest → Interface → javax.servlet.http

contents of HttpServletRequest :

1) UI | Form Data
2) URL
3) Type of Request
4) Client - information
   a) IP - address
   b) Browser Information (user - agent)
5) Cookie
6) MIME Type (accept)

Module of HttpServletRequest



Type of HttpRequest

There are 8 different types of HttpRequest provided namely.

① Post ② Get ③ Put ④ Delete ⑤ Trace

⑥ option ⑦ head ⑧ connect

Contents of HttpServlet Response

① Status Message

② Status Code

③ Cookie

④ Actual Content

Status code and Status Message :-

1XX - 100 → Success Message } Status message

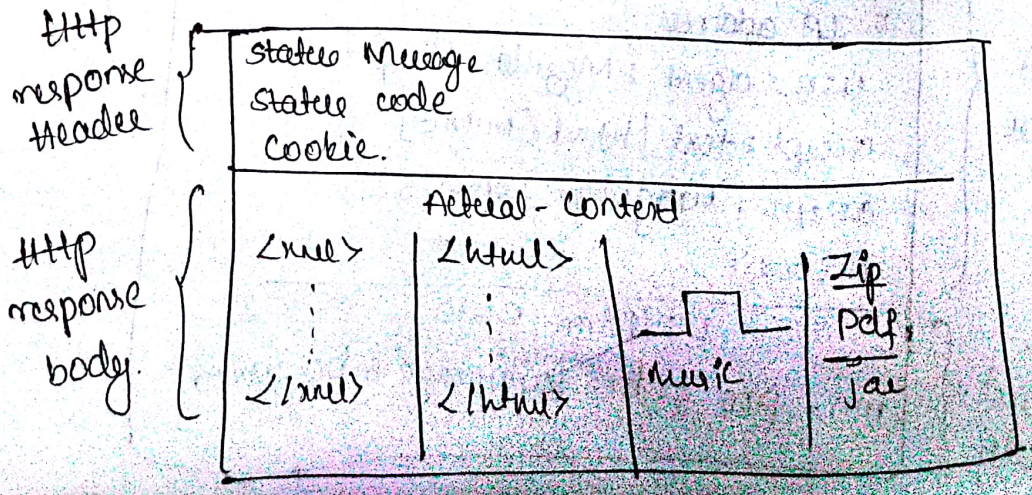
4XX - 404 → Client Error

5XX - 500 → Server Error

↓

Status code

Modeule of HttpServletResponse :-



Http post

→ post request is used to post some contents / data from client to the server

→ Post request deals with unlimited Data

imp → Post request is non-idempotent (not specific to any resource)

→ post request cannot be bookmark (saved)

imp → In case of post request, the data's are carried to the server as a part of http request body which is not displayed even to the end user (client). Hence the data's are secured.

→ whenever we deal with 'n' no of data's, then the type of request is post request.

Http get :-

→ get request is used to get the contents (or) resources from the server.

→ get request deals with limited data ie 1024 character.

* → Get request can be bookmark

→ get request is idempotent

→ In case of get request the data's are carried to the server as a part of request object in the form of key and value pair which is displayed in a URL, hence, the data's are not secured.

→ whenever we deal with the link, then the type of request is get request

Note: whenever the type of request is not mentioned or configured, then by default the type of request is getrequest.

Situations of getrequest

→ Clicking on a hyperlink is of type getrequest

→ whenever the form method or form type is not mentioned or configured, then by default the type of request is get request.

→ whenever a Servlet is directly called through it's URL pattern in the Browser, then the typed request is getRequest.

Http Put:-

put request is similar to that of post request but it is used to update Existing resources.

Http delete:-

Delete request is used to delete the contents of resources from the server.

Code to save the data's into the database server dynamically in a secured manner. entered by the user on a web page

student.html

```
<html>
<body bgcolor = "#00ff66 ">
<form action = "fs" method = "post">
<table>
<tr>
<td> Id : </td>
<td><input type = "text" name = "i "></td>
</tr>
<tr>
<td><input ty pe = "text" name = "nm"></td>
</tr>
<tr>
<td><input type = "text" name = "DP "></td>
</tr>
<tr>
<td><input type = "text" name = "pr"> </td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><input type = "submit" value = "Post"></td>
</tr>
```

```
</table>
</form>
</body>
</html>
```

FirstServlet

```
package org.raj.postApp;
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet
{
 @override
 protected void doPost(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOExcep
 {
  String sid = req.getParameter("id");
  int id = Integer.parseInt(sid);
  String name = req.getParameter("nm");
  String dept = req.getParameter("dp");
  String sperc = req.getParameter("pr");
  double perc = Double.parseDouble(sperc);

  PrintWriter out = resp.getWriter();
  out.println("<html><body bgcolor="yellow">"
+ "<h> Student Details are "+name+" "+dept+
```

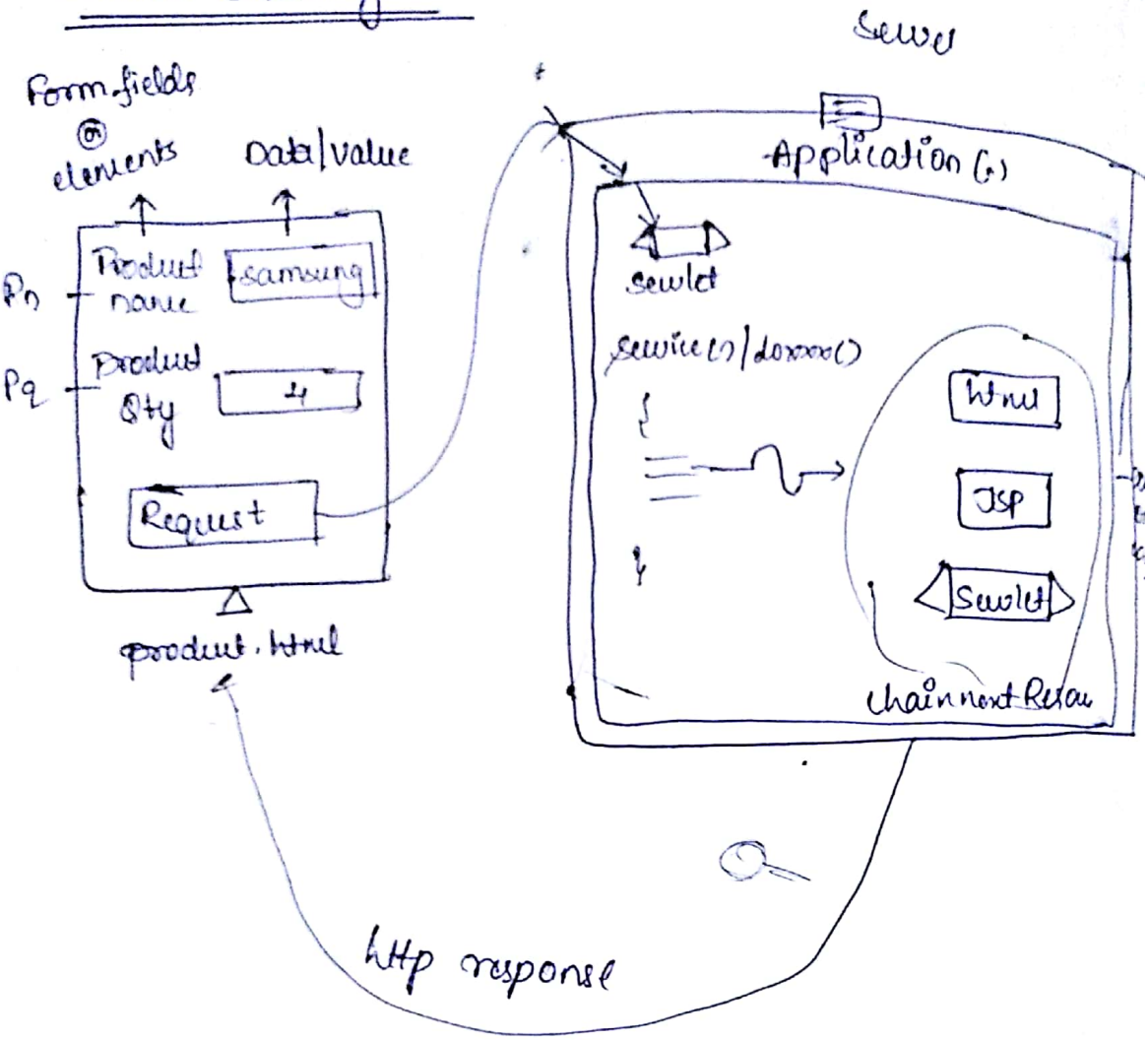
```
out.flush();
out.close();

Connection con = null;
PreparedStatement pstmt = null;
String qry = "insert into jecm30.student values(?,?,?,?)";

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

con = DriverManager.getConnection("jdbc:mysql://localhost:
   3306?user=root&password=dinga");

pstmt = con.prepareStatement(qry);
pstmt.setInt(1,id);
Pstmt.setString(2,name);
pstmt.setString(3,dept);
pstmt.setDouble(4,perc);
pstmt.executeUpdate(); // Resistance logic//
} catch (ClassNotFoundException | SQLException e) {
    e.printStackTrace();
  }
finally
{
  if (pstmt != null)
  {
    try {
        pstmt.close();
      }
```

```
        catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if (con != null)
    {
        try {
            con.close();
        }
        catch (SQLException e) {
            e.printStackTrace();
        }
        SOP("close all costly resources");
    }
}
```

web

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
<display-name> Post_Proj </display-name>
<welcome-file-list>
<welcome-file>student.html</welcome-file>
</welcome-file-list>
<servlet-mapping>
<servlet-name> FirstServ </servlet-name>
<url-pattern> /fs </url-pattern>
</servlet-mapping>
<servlet>
<servlet-name> FirstServ </servlet-name>
<servlet-class> org.raj.postApp.FirstServlet </servlet-class>
</servlet>
</web-app>
```

Servlet Chaining




```
double price = 40000.00;
double totalSa = (price * pq);
```

→ Whenever a client makes any request to a Servlet, Servlet can execute its logic either through service() (or) doXxx(). & then chain (or) communicate with another resource which can either be html, JSP (or) another Servlet. This type of Servlet to resource communication is known as Servlet chaining.

Definition for Servlet chaining:-

Chaining from one Servlet to another resource which can either be html, JSP, (or) another Servlet is known as Servlet chaining.

Need for Servlet Chaining

Servlet chaining is used to make the data's present in one Servlet available to another resources, which can either be Html, JSP (or) another Servlet.

Note: Servlet chaining will always begin whenever a client makes any request to a Servlet.

→ Servlet chaining can always be perform in 2 different ways.

a) RequestDispatcher

b) sendRedirect

a) RequestDispatcher :-

→ RequestDispatcher is interface which is present in javax.servlet package.

→ Since it is an interface and implementation object is created by the JEE container by calling a factory / Helper method called getRequestDispatcher() whenever a client makes any request to Servlet

→ RequestDispatcher is used to dispatch the request from one servlet to another resource which can either be Html, jsp, (or) another Servlet

Syntax :- (Servlet chaining)

Syntax :-

→ javax.servlet.RequestDispatcher rd = req.getRequestDispatcher ("home.html");
↳ Name of html Resource

→ javax.servlet.RequestDispatcher rd = req.getRequestDispatcher ("index.jsp");
↳ Name of JSP Resource

imp

javax.servlet.RequestDispatcher rd = req.getRequestDispatcher ("Servlet");
↳ URL Pattern for Servlet

address of implicit obj.

→ we can get the reference of RequestDispatcher interface in two different ways namely

① req
② context

javax.servlet.RequestDispatcher rd = req.getRequestDispatcher ("Servlet");
↳ URL patte

Imp

→ whenever we use request.getRequestDispatcher we can chain from one Servlet to another resource (html, JSP & Servlet) of the same application.

netbanking

→ whenever we use context.getRequestDispatcher we can chain from one Servlet to another Resource of Different application.

→ Note :- Imp

RequestDispatcher works at the Server Side

There are 2 different methods present in RequestDispatcher Interface namely.

a) forward method()

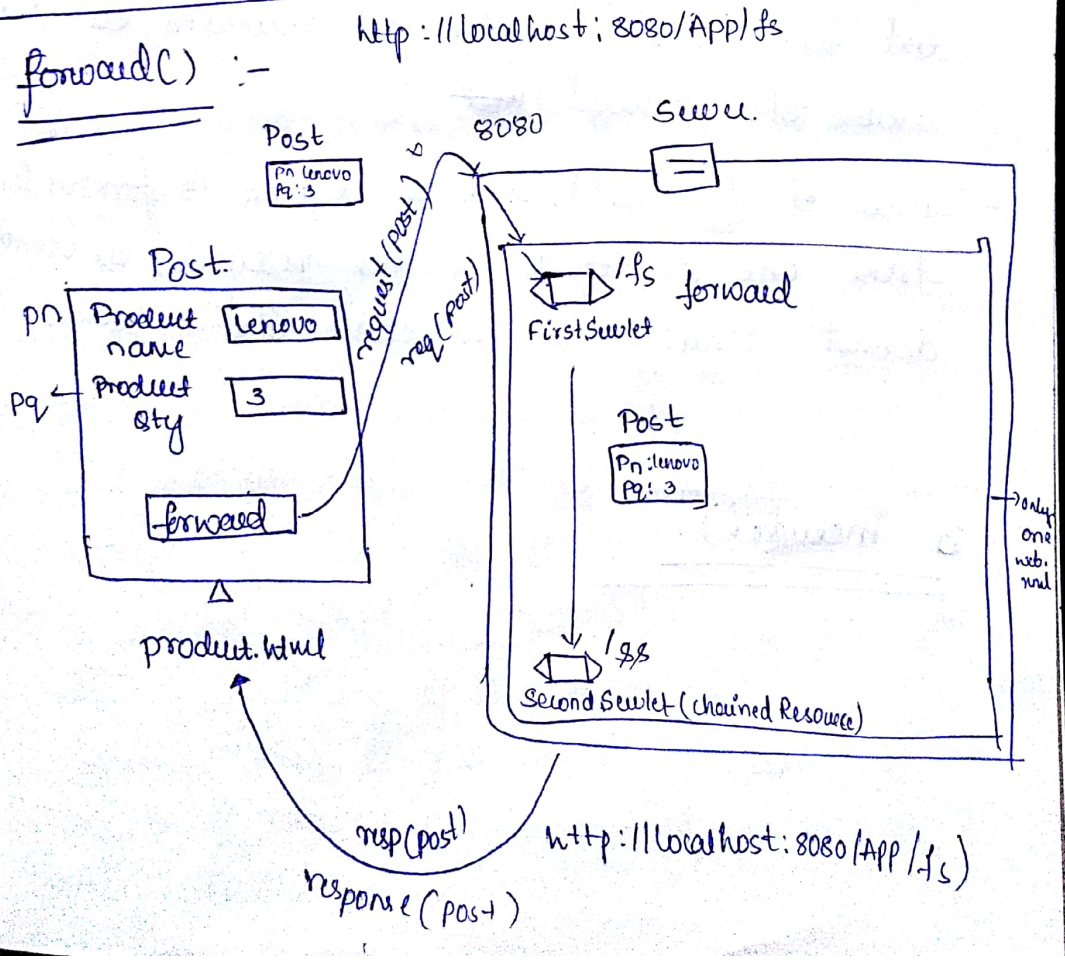
b) include method()

+ void forward (ServletRequest req, ServletResponse resp)

+ void include (ServletRequest req, ServletResponse resp).

---

forward() :-



→ Since forward is a method of request Dispatcher Interface it works at the server side.

→ forward() is used to forward the request from one Servlet to another resource which can either be html, JSP or another Servlet. (Servlet chaining)

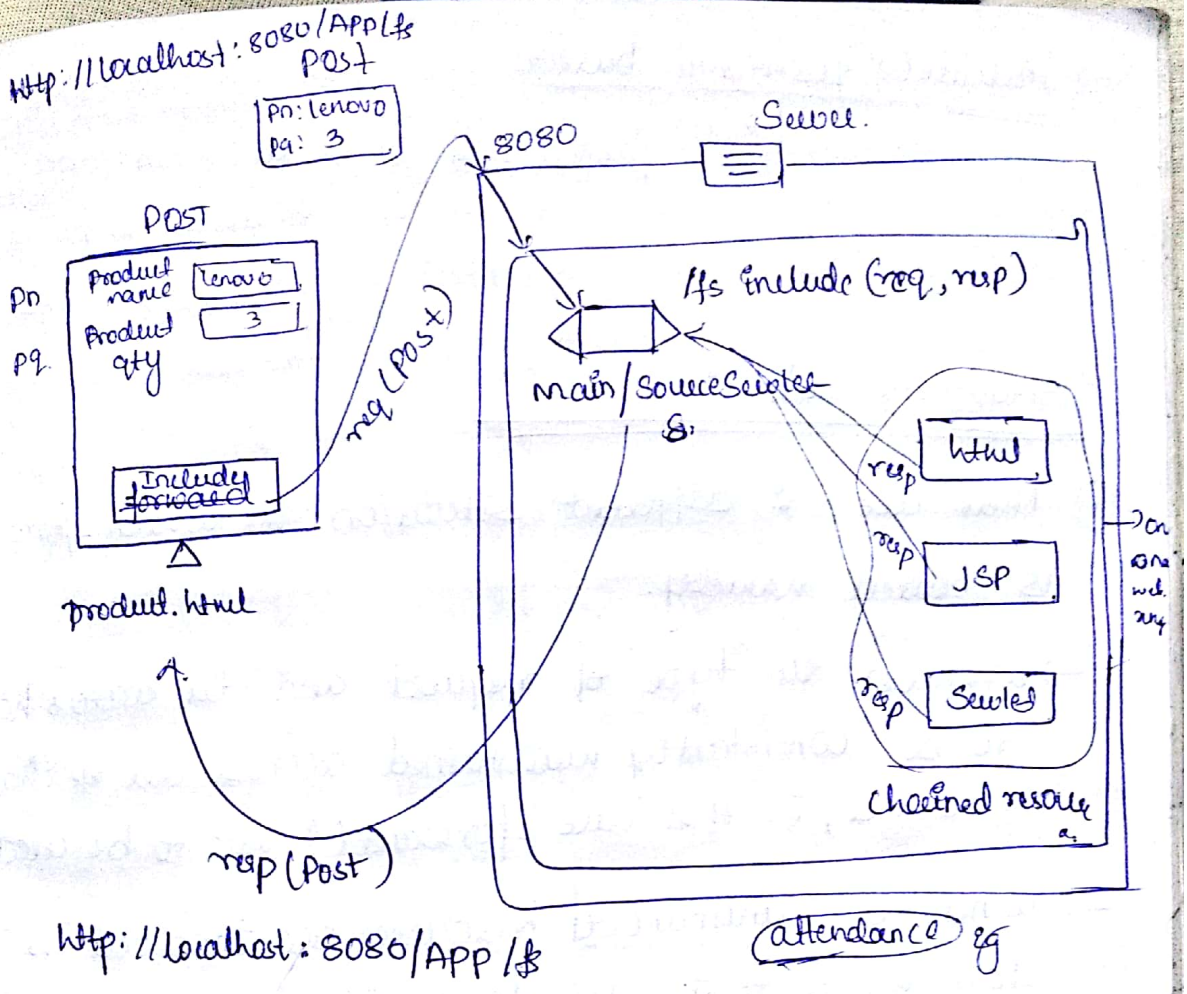
→ In case of forward(), from one Servlet to another resource the same existing request is forwarded across multiple chain resources. That means the same request is navigated.

*** → The type of request and the data's are consistently maintained across all the chain resources.

*** → The name of the chain resource are the url pattern of the chain resource is not displayed to the end user (client). Hence the url remains the same incase of forward() ~~method~~.

→ Incase of forward(), once the request is forwarded from one Servlet to another resource, the controller doesnot comeback to the same Servlet again.

2. include() :-



→ Since include is a method of RequestDispatcher interface, it works at the Server side.

Include method works like a response merger. (or) combine

→ In case of include(), the contents of all the chained resources are included into the main/source Servlet. Where the response of all the chained resources are combined (or) merge into one single response and returned back to the client.

→ In case of include(), the main/source Servlet is the one which takes a request as well as gets back the response. Hence, the url remains the same.

→ Any no of resources can be included into the main (or) Source Servlet.

→ include() is positional based

↳ The 1st seen chain resource content will be 1st included to the main/source servlet then the remain resource as per the position. (sequentially)

Situations for forward()

① There are 2 different situation in which forward() is called namely.

→ whenever the type of request and the data is has to be consistantly maintained accross all the chained resources, in that case forward() has to be used.

→ whenever minimum of resources are involved in chaining, in that case forward() has to be used.

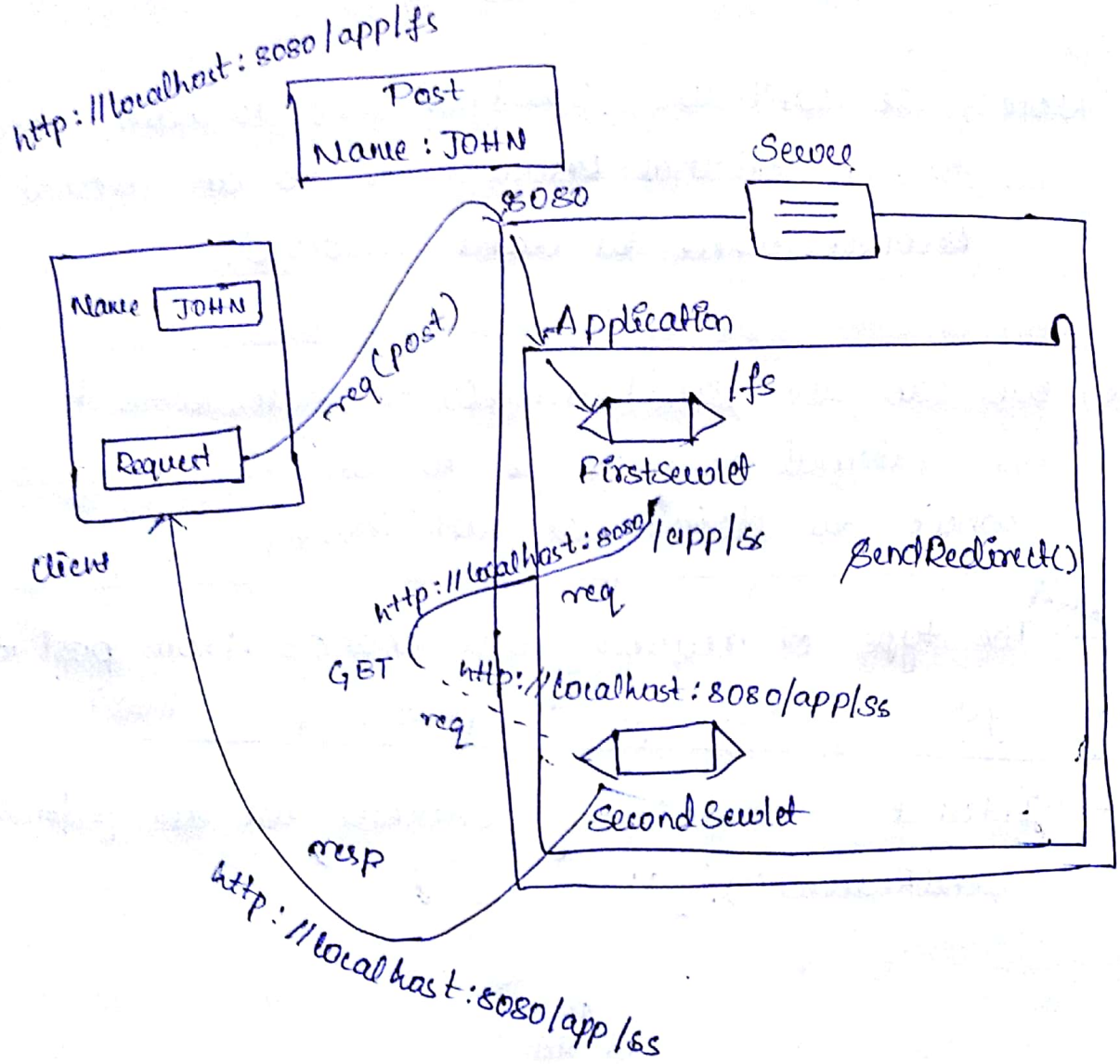
Situation for Include()

① whenever there are 'n' no of resources involved in chaining, in that case include() has to be used.

sendRedirect():-

Fb a/c → login → us ps → Login validation

SendRedirect() :-



→ sendRedirect is a method of the HttpServletResponse interface which is present in javax.servlet.http package

~~using send redirect~~

→ Using SendRedirect() we can chain to external resources also.

→ The change in the URL pattern is displayed in the browser.

→ Data inconsistency that means the data present in the FirstServlet may not be available to the ChainedResource.

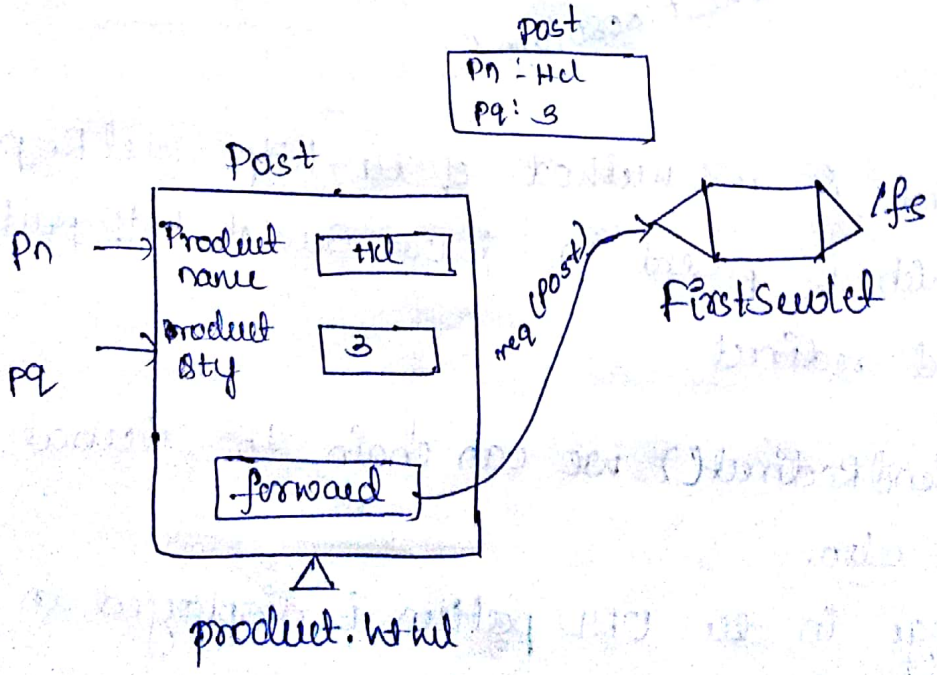
→ sendRedirect() work at the client side. since the Redirection happens in the Browser URL.

Note :- 1) we can't use sendRedirect() to chain either to an external Resource or to an internal Resource, where the data involved.

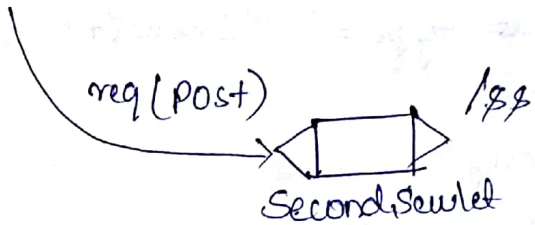
⇒ we can use sendRedirect() to chain either to an external resource or to an internal resource. where the data's are not involved.

→ The type of request will change from post to get, but not get to post. by JEE container.

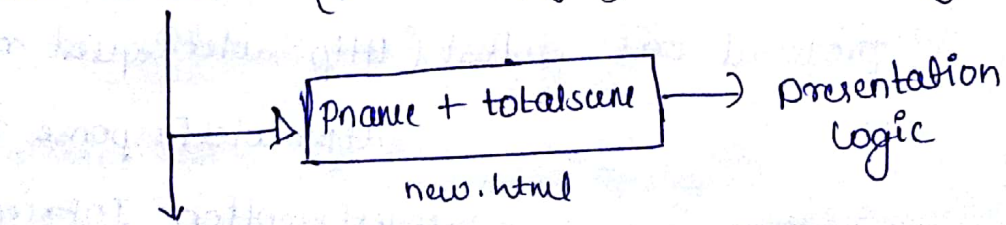
→ ~~[Internet is mandatory whenever we are performing sendRedirect().]~~ X

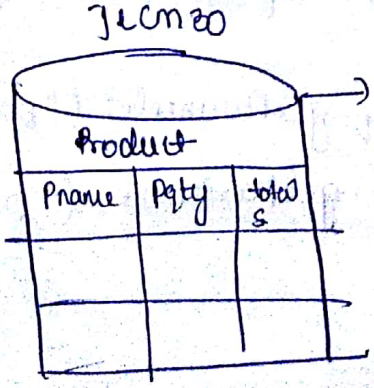


```
+ class FirstServlet ↑ HttpServlet
{
  doPost(req, resp)   (post)
  {
    String pname = req.getParameter("pn");
    String pqty = req.getParameter("pq");
    req.setAttribute("prdnm", pname);
    req.setAttribute("prdqt", pqty);

RequestDispatcher rd = req.getRequestDispatcher("ss");
  rd.forward(req, resp);   (post)
  }
}
```




```
+ class SecondServlet ↑ HttpServlet
{
  doPost(req, resp)   (Post)   ctrl space
  {
    String pname = (String) req.getAttribute("prdnm");
    double price = 40000.00;   (pqty)
    double totalsum = (price * pqty)  → Business logic
```





→ persistance logic

Forward() eg

```
<!DOCTYPE html>
<html>
<meta charset = "ISO-8859-1 ">
<body bgcolor = "cyan ">
<form action = " fs " method = "post ">
Product Name : <input type = "text " name = " pn ">
Product Quantity : <input type = "text " name = "pq">
< br > < /br >
<input type = "submit" value = " Forward ">
</form>
</body>
</html>
```

FirstServlet

```
package org.raj.chainApp;
import java.io.*;
public class FirstServlet extends HttpServlet
{
  @Override
  protected void doPost (HttpServletRequest req,
                         HttpServletResponse resp) throws
                  ServletException, IOException
  {
     String pname = req.getParameter ("pn");
     String pqty = req.getParameter ("pq");
```

```
        //ADD REQUEST object into scope //
        req.setAttribute("prdnm", pname);
        req.setAttribute("prdqt", pqty);
    RequestDispatcher rd = req.getRequestDispatcher("ss");
     rd.forward(req, resp);
    }
}
```

Second.Servlet

```
package org.raj.chainApp;
import java.io.*;
public class SecondServlet extends HttpServlet
{
  @Override
protected void doPost(HttpServletRequest req, HttpServlet
            Response resp) throws ServletException, IOException
{
String pname = (String) req.getAttribute("prdnm");
String pqty = (String) req.getAttribute("prdqt");
 int qty = Integer.parseInt(pqty);
double price = 40000.00;
double totalscore = (price*qty); // Business logic //

PrintWriter out = resp.getWriter();
out.println("<html><body bgcolor = 'yellow'>"
+"<h1> Product details are "+pname+" "+total
                                    Sum+"</h1>"
    + "</body></html>");
out.flush();
out.close(); // presentation logic //
```

```
Connection con = null;
PreparedStatement pstmt = null;
String qry = "insert into jecm30.product values(?,?,?)";

try{
  class.forName("com.mysql.jdbc.Driver");
con = DriverManager.getConnection("jdbc:mysql://localhost
      :3306/?user=root&password=dinga");
pstmt = con.prepareStatement(qry);
pstmt.setString(1, pname);
pstmt.setInt(2, qty);
pstmt.setDouble(3, totalsum);
pstmt.executeUpdate(); // Persistance logic //
} catch(ClassNotFoundException | SQLException e){
      e.printStackTrace();

}
  finally
  {
    if(pstmt != null)
    {
      try{
        pstmt.close();
      } catch(SQLException e) {
          e.printStackTrace();
      }
    }
```

```
        if (con != null)
        {
            try {
                con.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
 <display-name> chain_Proj </display-name>
 <welcome-file-list>
 <welcome-file> product.html </welcome-file>
 </welcome-file-list>
 <servlet-mapping>
 <servlet-name> Firstserv </servlet-name>
 <url-pattern> /fs </url-pattern>
 </servlet-mapping>
 <servlet>
 <servlet-name> Firstserv </servlet_name>
 <servlet-class> org.raj.chainApp.Firstservlet </servlet-class>
 </servlet>
 <servlet-mapping>
 <servlet-name> SecondServ </servlet-name>
 <url-pattern> /ss </url-pattern>
```

</servlet-mapping>
<servlet>
<servlet-name> secondservlet </servlet-name>
<servlet-class> org.raj.chainApp.SecondServlet
</servlet-class>
</servlet>
</web-app>

include()

htmlfile

```
<!DOCTYPE html>
<html>
<meta charset = "ISO-8859-1">
<body bgcolor = "cyan">
<form action = "fs" method = "post">
Product Name : <input type = "text" name = "pn">
product Quantity : <input type = "text" name = "pq">
<br><br>
<input type = "submit" value = "include">
</form>
</body>
</html>
```

FirstServlet

```
Package org.raj.includeApp;
import java.io.*;
public class FirstServlet extends HttpServlet
```

```
{
@override
protected void doPost (HttpServletRequest req,
    HttpServletResponse resp) throws ServletException,
                                     IOException

{
  String pname = req.getParameter ("pn");
  String pqty = req.getParameter ("pq");
// ADD REQUEST OBJECT INTO SCOPE //
  req.setAttribute ("prdnm", pname);
  req.setAttribute("prdqt", pqty);

RequestDispatcher rd = req.getRequestDispatcher ("ss");
rd.include (req, resp);

double totalsum = (double) req.getAttribute ("tsum");
PrintWriter out = resp.getWriter ();
out.println ("<html><body bgcolor='yellow'>"
  + "<h1>product Details are "+pname+"
    "+totalsum+"</h1>"
    + "</body></html>");

out.flush();
out.close(); // Presentation logic//

}
}
```

SecondServlet

```
package org.raj.includeApp;
import java.io.*;
public class SecondServlet extends HttpServlet
{
  @Override
protected void doPost(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOExceptio
{
  String pname = (String) req.getAttribute("pdm");
  String pqty = (String) req.getAttribute("pdqt");
  int qty = Integer.parseInt(pqty);
  double price = 40000.00;
  double totalsum = (price * qty); // Business logic //
  req.setAttribute("tsum", totalsum);

  Connection con=null;
  PreparedStatement pstmt = null;
  String qry = "insert into jecm30.product values(???)";
  try{
    Class.forName("com.mysql.jdbc.Driver");
con = DriverManager.getConnection("jdbc:mysql://localhost:
        3306?user=root&password=dinga")
  pstmt = con.prepareStatement(qry);
  pstmt.setString(1, pname);
  pstmt.setInt(2, qty);
```

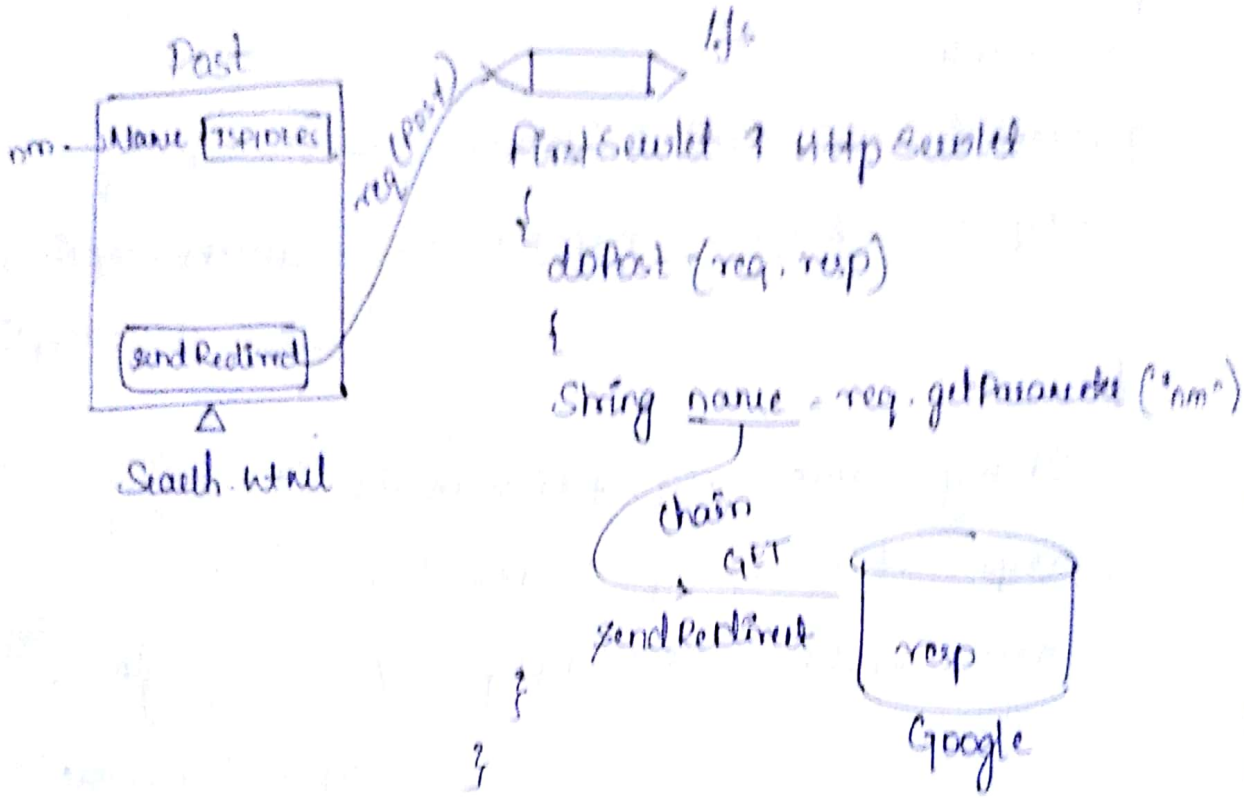
```
        pstmt.setDouble(3, totalsum);
        pstmt.executeUpdate(); // Persistance logic //
    } catch (UserNotFoundException | SQLException e) {
        e.printStackTrace();
    }
    finally
    {
        if (pstmt != null)
        {
            try{
                pstmt.close();
            } catch (SQLException e){
                e.printStackTrace();
            }
        }
        if (con != null)
        {
            try{
                con.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

web

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
<display-name> chain_Proj </display-name>
<welcome-file-list>
```

```
<welcome-file>product.html</welcome-file>
</welcome-file-list>
<servlet-mapping>
<servlet-name>FirstServ</servlet-name>
<url-pattern>/fs</url-pattern>
</servlet-mapping>
<servlet>
<servlet-name>FirstServ</servlet-name>
<servlet-class>org.raj.includeApp.FirstServlet
</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>SecondServ</servlet-name>
<servlet-pattern>/ss</url-pattern>
</servlet-mapping>
<servlet>
<servlet-name>Secondserv</servlet-name>
<servlet-class>org.raj.includeApp.Secondserv
</servlet-class>
</servlet>
</web-app>
```

code for sendRedirect()



~~web~~ html

```
<html>
<body bgcolor : "yellow">
<form action = "fs" method = "post">
Name: <input type = "text" name = "nm">
<br></br>
<input type = "submit" value = "SendRedirect">
</form>
</body>
</html>
```

```
package org.raj.chainApp;
import java.io.*;
public class FirstServlet extends HttpServlet
{
@Override
protected void doPost(HttpServletRequest req,
  HttpServletResponse resp) throws ServletException,
                                   IOException
{
  String name = req.getParameter("nm");
  resp.setContentType("text/html");
  resp.sendRedirect("https://www.google.co.in
                     /#q=" + name);
  }
}
```

xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
<display-name>chain-proj</display-name>
<welcome-file-list>
<welcome-file>search.html</welcome-file>
</welcome-file-list>
<servlet-mapping>
<servlet-name>FirstServ</servlet-name>
<url-pattern>/fs</url-pattern>
</servlet-mapping>
```

```
<servlet>
<servlet-name> FirstServ </servlet-name>
<servlet-class> org.raj.chainApp.FirstServlet </servlet-class>
</servlet>
</webapp>
```

Cookie

Definition:- Cookie is a temporary storage area used to store small amount of data & information which is created by the server and it is stored on the client side (browser).

Need for Cookie:-

(1) Cookie is needed for session tracking (all activities / records b/w st & staff)

→ cookie is created by the server, added into the response header and only then the response is rendered back to the client

→ Cookie is created by the server & stored & then on the hard disk of client system (browser).

*** → Once the cookie is enabled, whenever a client makes second request to the server, the cookies travel as a part of HttpRequestHeader which is not displayed even to the end user. Hence, the data's are secured

→ There are 2 different type of cookie namely
1) Persistant cookie (Store)
2) Non - Persistant Cookie

1) Persistant cookie:

These are the cookies which are read by the client & stored either on the harddisk or client system (Browser) for a particular time mentioned by the server

→ These cookies exist even after restarting the system but only until the time mentioned by the server elapses (or) expires.

Non-Persistant cookie:-

→ These are the cookies which are stored in the Browser cache memory & destroyed immediately once the Browser is closed.

Advantages of cookies:-

→ Cookies can be used for session tracking (or) session management.

→ cookie can be used to store non-sensitive data or information. since it is stored on the client side (Browser)

Disadvantages of cookies:-

→ cookie can be disabled by the client or browser at any point of time.

→ cookie can't be used to store sensitive data or information since it stored on the client side (Browser)

→ cookie can be used to store only limited data or information

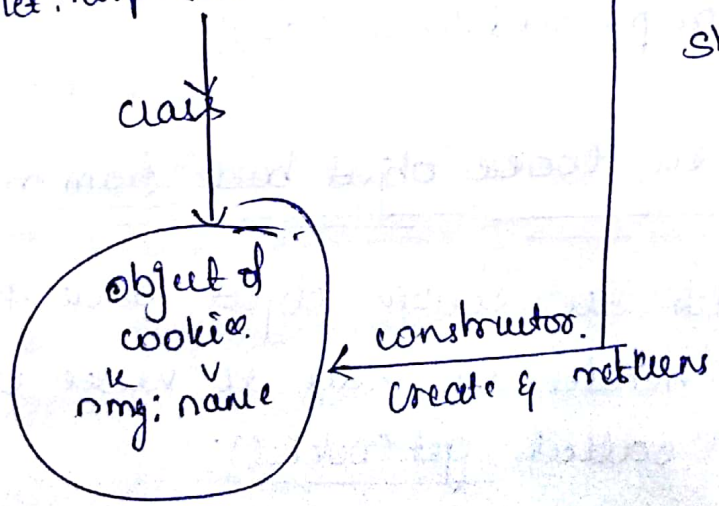
Note:- In case of programming language cookie is a class which is present in javax.servlet.http package

1) creating an object of cookie class

Since, cookie is a class present in javax.servlet.http package & object of this class can be created by using a cookie constructor which has a parameterised constructor as follows.

cookie(string name, string value)

javax.servlet.http.Cookie ck = new Cookie (string name, string value);



Eg:- Cookie ck = new Cookie ("nmg", name);

2) set the age for cookie object:

***→ Based on the age of the cookie object, the cookie is concluded as persistent or non-persistant.

→ The age of for cookie object can be set by using a method called setMaxAge();

+ void setMaxAge(int seconds);

Eg:- ck.setMaxAge(100);

3) Add the cookie object into response header:

→ until the cookie object is added into the response header, the response is not rendered back to the client.

→ The cookie object can be added into the response header by using a method called addCookie();

+ void addCookie(cookie ck)

Eg:- resp.addCookie(ck);

4) Fetch the cookie object back from response header

→ To fetch the cookie object back from the response header, we have to make use of a method called getCookie();

getCookie(); return a cookie array

Eg:- Cookie c[] = req.getCookie();

get

*** getCookie() always returns a cookie array

Session :-

Http Session:

→ Any activity which takesplace b/w start tym and stop tym is considered to be session.

→ Since session is specific to a particular type of protocol called Http protocol Hence, the name HttpSession.

→ HttpSession is an interface which is present in javax.servlet.http package.

→ Since it is an interface, & an implementation object of this interface can be created by the jee container by calling a factory/helper method called getSession().

*** → Only one implementation object of HttpSession Interface is created by Jee container for a particular client (or) user. along with unique session id. who differentiate b/w the different session object.

→ The data's present in session object (or) user specific data which is limited (or) restricted to a particular client (or) user.

→ One session obj

Advantages of session:

→ Session object can store unlimited data (or) no. of data

Disadvantages of session:

→ Session object will be destroyed immediatly once the browser is closed.

* The scope of session object is throughout the application.

Note:- (job + offer)

1) whenever the age of cookie object is configured with a negative integer value then the cookie object will be destroyed immediately once the browser is closed. Hence, concluded as non-persistance cookie

2) whenever the age of Cookie object is configured with a positive integer value, then the cookie object will be destroyed once the time mentioned by the server elapses (or) Expires. Hence, concluded as persistant cookie.

cookie (code)

form.html

```
<html>
<body><meta charset = "ISO-8859-1">
<body bgcolor = "cyan">
<form action = "fs" method = "post">
Name : <input type = "text" name = "nm">
<br> <br>
<input type = "submit" value = "Request">
</form>
</body>
</html>
```

FirstServlet

```
package org.raj.cookieApp;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet
{
  @Override
  protected void doPost (HttpServletRequest req,
HttpServletResponse resp) throws ServletException,
                                   IOException.
{
  String name = req.getParameter ("nm");
  Cookie ck = new Cookie ("nmg", name);
```

```
ck.setMaxAge(800);
resp.addCookie(ck);
PrintWriter out = resp.getWriter();
out.print("<html>");
out.print("<body>");
out.print("<form action='ss'>");
out.print("<input type='submit' value=
        'NextServlet'>");
out.print("</form>");
out.print("</body>");
out.print("</html>");
out.flush();
out.close();
}
}
```

SecondServlet

```
package org.raj.cookieApp;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SecondServlet extends HttpServlet
{
@Override
protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws ServletException,
                                 IOException
```

```
{
    Cookie c[] = req.getCookie();
    PrintWriter out = resp.getWriter();
    out.println("<html><body bgcolor='yellow'>"
+"<H1>Cookie object value "+c[0].getvalue()+
    "</H1>"+"</body></html>");
    out.flush();
    out.close();
  }
}
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
<display-name>Cookie-proj</display-name>
<welcome-file-list>
<welcome-file>form.html</welcome-file>
</welcome-file-list>
<servlet-mapping>
<servlet-name>Firstsew</servlet-name>
<url-pattern>/fs</url-pattern>
</servlet-mapping>
<servlet>
<servlet-name>Firstsew</servlet-name>
<servlet-class>org.raj.cookieApp.Firstservlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>Secondsew</servlet-name>
<url-pattern>/ss</url-pattern>
</servlet-mapping>
```

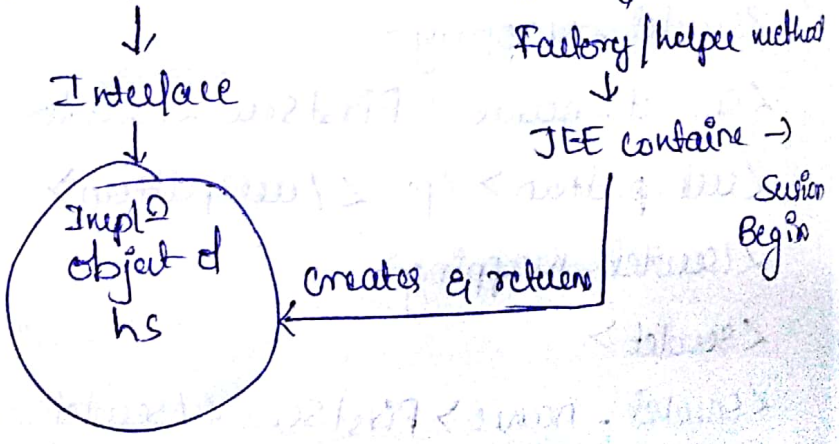
```
<servlet>
<servlet-name>SecondServ</servlet-name>
<servlet-class>org.raj.cookieApp.SecondServlet
</servlet-class>
</servlet>
</web-app>
```

Session

① Creating an implementation object of http session interface

Since http session is an interface present in javax.servlet.http Package, an implementation object of this interface is created by the JEE container for every client when user coherence the session begins

```
javax.servlet.http.HttpSession hs = req.getSession();
```



Eg:- HttpSession hs = req.getSession();

③ set the age for session object

The age for session object can be set by using a method called setMaxInactiveInterval()

+void setMaxInactiveInterval(int seconds)

Eg:- hs.setMaxInactiveInterval(500);

④ Add the session object into the scope

A session object can be added into the scope by using method called setAttribute().

+void setAttribute(String key, object obj)

Eg:- hs.setAttribute("nmg", name);

⑤ fetch the session object back from the scope.

A session object can be fetched back from the scope by using getAttribute()

+object getAttribute(String key)

Eg:- hs.getAttribute("nmg");

We can explicitly destroy a session object at any part of time by using a method called invalidate().

+void invalidate()

Eg:- hs.invalidate();

Note: whenever the age of session object is configured either with a positive integer value (or) with negative integer value, the session object will be destroyed immediately once the Browser is closed (or) Browsing session ends.

Note: whenever we deal with a session object, it is not mandatory to have a fixed no of data's init. Since session object deals with unlimited data.

form.html

```
<!Doctype html>
<html>
<meta charset = "ISO-8859-1">
<body bgcolor = "cyan">
<form action = "fs" method = "post">
Name : <input type = "text" name = "nm">
<br></br>
<input type > "submit" value = "Request">
</form>
</body>
</html>
```

FirstServlet

```
package org.raj.cookieApp;
import java.io.*;
public class FirstServlet extends HttpServlet
{
  @Override
  protected void doPost(HttpServletRequest req,
HttpServletResponse resp) throws ServletException,
                                  IOException.
{
  String name = req.getParameter("nm");
  HttpSession hs = req.getSession();
  hs.setMaxInactiveInterval(800);
  //hs.invalidate();
 hs.setAttribute("nmg", name);
 PrintWriter out = resp.getWriter();
  out.print("<html>");
  out.print("<body>");
  out.print("<form action='ss'>");
 out.print("<input type='submit' value='NextServlet'>");
 out.print("</form>");
out.print("</html>");
 Out.flush();
 out.close();
}
}
```

Second Servlet

```
package org.raj.cookieApp;
import java.io.*;
public class SecondServlet extends HttpServlet
{
@Override
protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws ServletException,
IOException
{
HttpSession hs = req.getSession(false);
String name = (String) hs.getAttribute("nmg")
PrintWriter out = resp.getWriter();
out.println("<html><body bgcolor='yellow'>
+ "<h1>Session object value "+name+" "+hs.getId()
+ "</h1>"
+ "</body></html>");
out.flush();
out.close();
}
}
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<welcome-file-list>
<welcome-file>form.html</welcome-file>
```

```
</welcome-file-list>
<servlet-mapping>
<servlet-name> FirstServ </servlet-name>
<url-pattern> /fs </url-pattern>
</servlet-mapping>
<servlet>
<servlet-name> FirstServ </servlet-name>
<servlet-class> org.raj.SessionApp.FirstServlet </servlet-class>
</servlet>
<servlet-mapping>
<servlet-name> SecondServ </servlet-name>
<url-pattern> /ss </url-pattern>
</servlet-mapping>
<servlet>
<servlet-name> SecondServ </servlet-name>
<servlet-class> org.raj.SessionApp.SecondServlet
                                    </servlet-class>
</servlet>
</web-app>
```

Note:

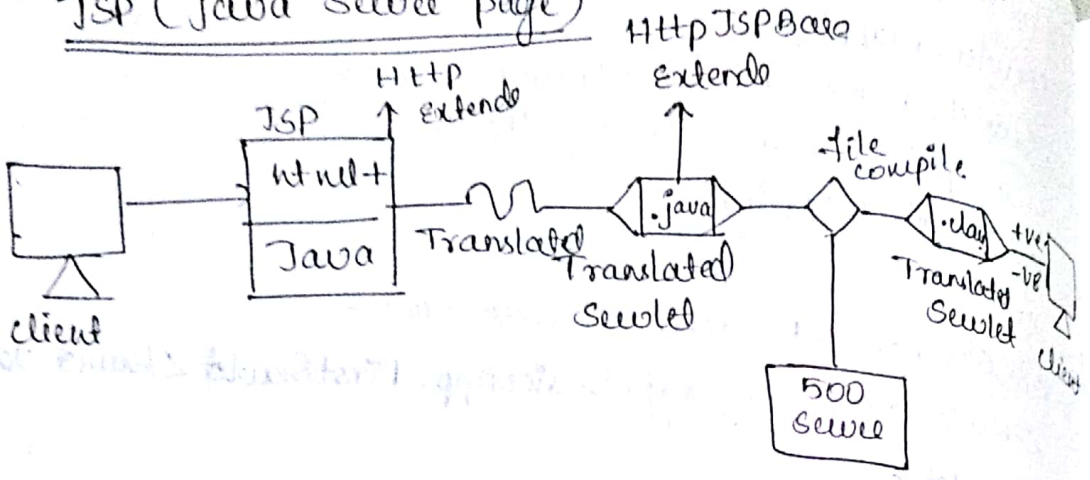
1) when we set the age of Session object is configured either with a +ve integer (or) -ve integer value, result must be same i.e the session object will be destroyed immediately once browser is closed (or) Browsing Session Ends.

2) when ever we deal with a Session object, it is not mandatory to have a fixed no of data's in it.

Since, Session object deals with unlimited data

→ Dynamic resources

JSP (Java Server page)



Why JSP is known as "High level Abstraction of Servlet"

∴ JSP is internally a "Servlet"

- i.e the translation / conversion of JSP into a Servlet by JEE container for the 1st client request is known as High level Abstraction of servlet.
- when ever a client makes a 1st request to a JSP, the JSP is translated / converted into Servlet by the JEE container which is known as translated Servlet of type .java.
- On successful compilation a translated servlet a .class is generated & response is rendered back to client.
- If compilation fails, then the JEE container throws an error with 500 (ServerError)
- Hence, Each & every JSP is "internally a Servlet"

Specification of JSP:-

- There 3 diff specifications present in JSP. namely
  1) By default, JSP extends HttpJ

2) By default, all the translated servlets extends a class by the name HttpJspBase class

3) By default all the translated servlet are stored in the work folder of Apache-Tomcat.

[CATALINA_HOME/WORK]

JSP is case sensitive, JSP is strictly defined language

Life cycle phases of JSP:

There are 5 diff life cycle phases of JSP present. namely.

1) Translation phase
2) Instantiation / object creation phase
3) Initialize phase
4) Service phase
5) Distruction phase

Life cycle methods of JSP:

1) _jspInit() → @override
2) _jspService(HttpServletRequest req, HttpServletResponse resp)
3) _jspdestroy() → @override

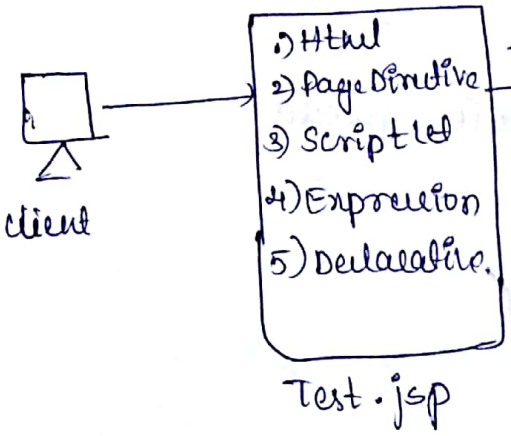
* In case of JSP, _jspInit(), _jspdestroy() is overridden But overriding _jspService() is not a good practice, since it depends upon the type of HttpRequest.

→ -jspInit(), -jspdestroy() can be overridden as follows

```
<%!
  -jspInit()
  {
  //Initialize all the
        Resources
  }
%>
```

```
<%!
  -jspdestroy()
  {
  // close all the costly
        Resources
  }
%>
```

Contents of JSP



naming convention followed by

Test_jsp.java →

```
+class Test_jsp extends HttpJspBase
{
  -jstInit()
{
}
-jspService(hreq, hresp)
{
  ①
  ②
  ③
  ④
}
-jspdestroy()
{
  ⑤
}
```

WWP

Note: The naming convention followed by Apache tomcat Server to store all the translated Servlets into the workfolder of Apache.Tomcat server is named as

filename_jsp.java

② most of the contents of jsp fits into -jspService()

③ JSP is used to give only presentation logic

JSP) why is Jsp used to give a presentation logic instead of html?

→ Jsp is used to give a presentation logic for 2 imp reasons namely.

1) since Jsp is a dynamic resource

2) code simplification.

→ Page directive: intern it is considered default content present on any Jsp file

→ Scriptlet → The area which is used write a java code on a jsp file

make
tomcat
server
↓
filename - jsp.java

eg: <%
Java code
%>

Incase of Jsp. Scriptlet is represented as

Expression:- It is the one which is used to represent or print the Java code

eg: <%= %> in case of Jsp Expression is represented as

criteria for a Jsp

→ There are 3 diff criteria present for a Jsp

1) we cannot write one Jsp element inside other Jsp element

<% <% %> %>

2) we cannot write any html code inside the Jsp element

<% <html> %>

3) we can write any Jsp element for any no of times

<% %>
<% %>

Scopes of Jsp

There are 4 diff scopes present in Jsp namely

1) page - "pageScope" of Jsp → (page directive) → available only when on a Jsp

2) Request

3) Session

4) application (servlet context)

Implicit object of JSP (Imp)

There are 9 different implicit object present in JSP which as follows:-

1) request → refers to HttpServletRequest
2) response → HttpServletResponse
3) config → ServletConfig → getServletConfig()
4) session → HttpSession → getSession()
5) application → ServletContext → getServletContext()

} implicit object of JSP

6) page → object
7) out → JSPWriter
8) exception → Throwable
9) page context. → pagecontext.

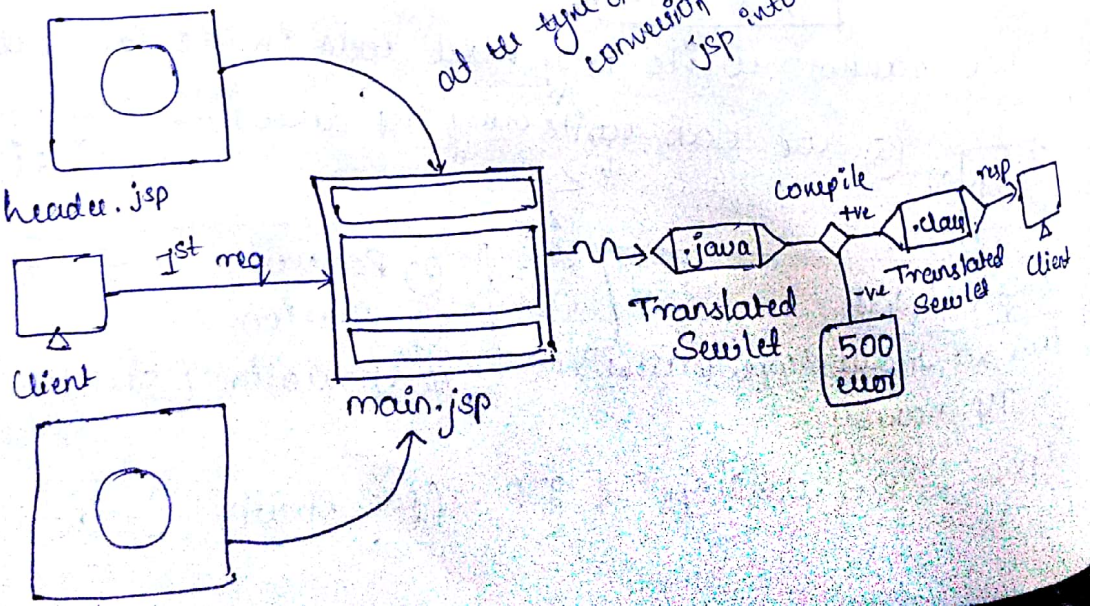
Session from trained

Directives of JSP:-

→ There are 3 different directives present in JSP namely

1) Page directive
2) Include directive
3) Taglib directive

2) Include directive



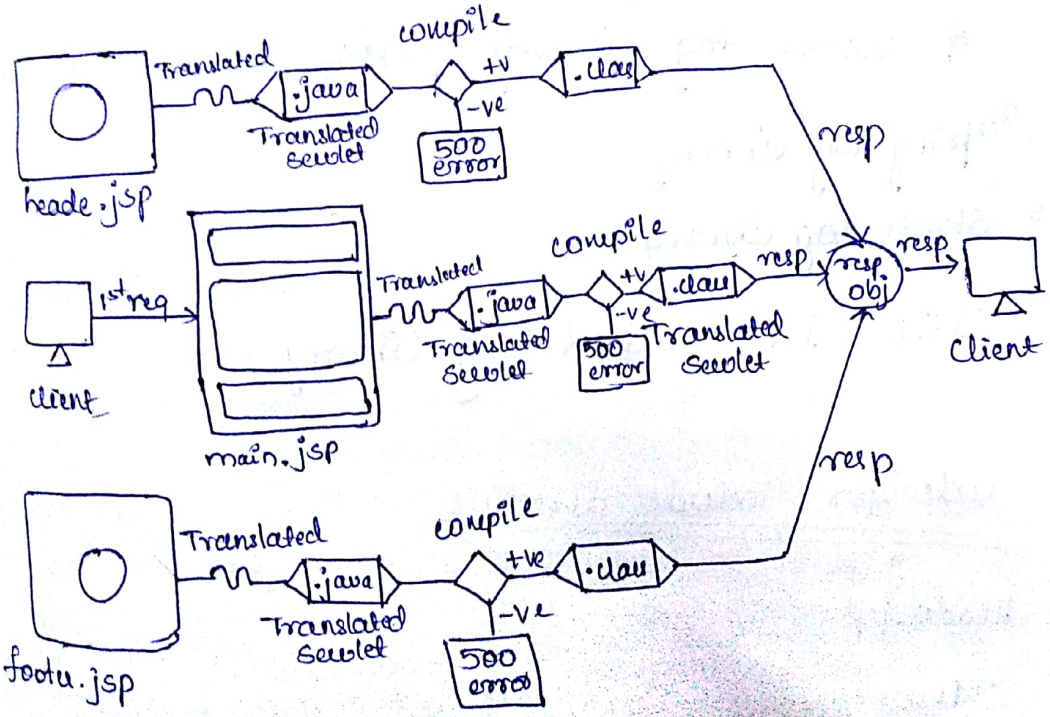
, Include directive works at the translation phase.

→ In this case, the contents of all the including pages are included into the main JSP file & then the final content is translated into one single translated servlet of type .java & on succesfull compilation a translated servlet of .class is generated & the response is rendered back to the client.

→ If the compilation fails then the jee container throws an error with 500 code ie server error

→ This is a good choice if there are minimum no of modifications involved to the JSP files.

JSP include @ Include standard action



① Include standard action works @ the runtime

② In this case, the contents of all the JSP files are separately translated into individual

translated servlet which of type .java & on successful compilation a translated servlet of .class is generated at the runtym all the responses are combined into one single response object and only one single response is rendered back to the client.

→ This is a good choice if there are frequent modification made to the JSP files.

Taglib directive:-

→ It is used to refer an external library of tags or custom tags.

→ most of the frame work comes with a set of custom tag library namely

a) Spring tag library
b) Struts tag library
c) JSTL - JSP standard Tag library.

Code for include directive

header.jsp

```
<html>
<body>
<font color="yellow" size="6">
Home || Notifications || FriendRequests || Search || chats
</font> </body> </html>
```

Footer.jsp

```
<html>
<body>
<font color = "yellow" size = "4">
CopyRight @ 2018 || TSC Apply || Contact us || Feedback
|| about us.

</font>
</body>
</html>
```

main.jsp

```
<html>
<body bgcolor = "black">
<%@ include file = "header.jsp" %>
<p></p> <p></p>
<font color = "white" size = "10">
JEE is Ending !!
</font>
<p></p><p></p>
<%@ include file = "footer.jsp" %>
</body>
</html>
```

web.xml

```
<?xml version = "1.0" encoding = "UTF-8"?>
<web-app>
<display name> JSP-proj </display-name>
<welcome-file
```

Code for include standard action:-

main.jsp

```
<html>
<body bgcolor = "black">
<jsp:include page = "header.jsp"></jsp:include>
<p></p><p></p>
<font color = "white" size = "10">
J2EE is Ending!:
</font>
<p></p> <p></p>
<jsp:include page = "footer.jsp" ></jsp:include>
</body>
</html>
```

Code to display date & time using JSP

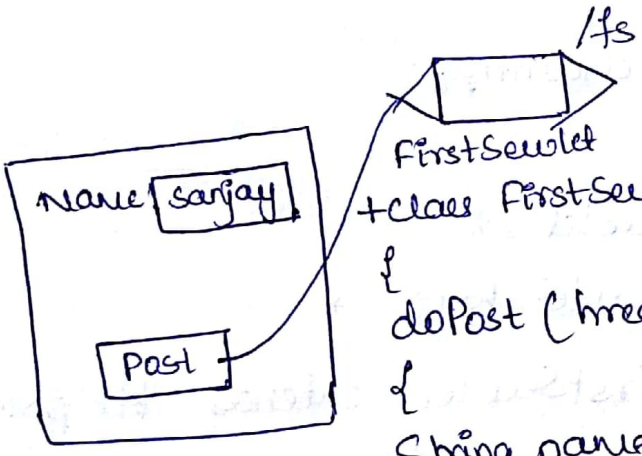
~~web.xml~~ date.jsp

```
<%@ page import = "java.util.Date" %>
<html>
<body bgcolor = "black">
<%
    Date d = new Date();
%>
<font color = "white" size = "10"><%= d %>
</font>
</body>
</html>
```

~~date.jsp~~ web.xml

```
<?xml version = "1.0" encoding = "UTF-8"?>
<web-app>
<display-name>jsp-proj</display_name>
<welcome-file-list>
<welcome-file>date.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

code for Servlet to JSP chaining




```
+class FirstServlet extends HttpServlet
{
doPost(hreq, hresp)
{
String name = req.getParameter("nm");
req.setAttribute("nmg", name);
RequestDispatcher rd = req.getRequestDispatcher("index.jsp");
rd.forward(req, resp);
}
}
```

```
String name = (String) req.
        getAttribute("nmg");
<%= name>
```

index.jsp

form.html

```
<html>
<body bgcolor = "cyan">
<form action = "fs" method = "post">
Name : <input type = "text" name = "nm">
<br> <br>
<input type = "submit" value = "post">
</form>
</body>
</html>
```

FirstServlet

```
package org.raj.chainApp;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet
{
  @override
protected void doPost (HttpServletRequest req,
    HttpServletResponse resp) throws ServletException
                                    IOException.
  {
     String name = req.getParameter ("nm");
     req.setAttribute ("nmg", name);
RequestDispatcher rd = req.getRequestDispatcher
  rd.forward (req, resp);          ("index.jsp")
}
```

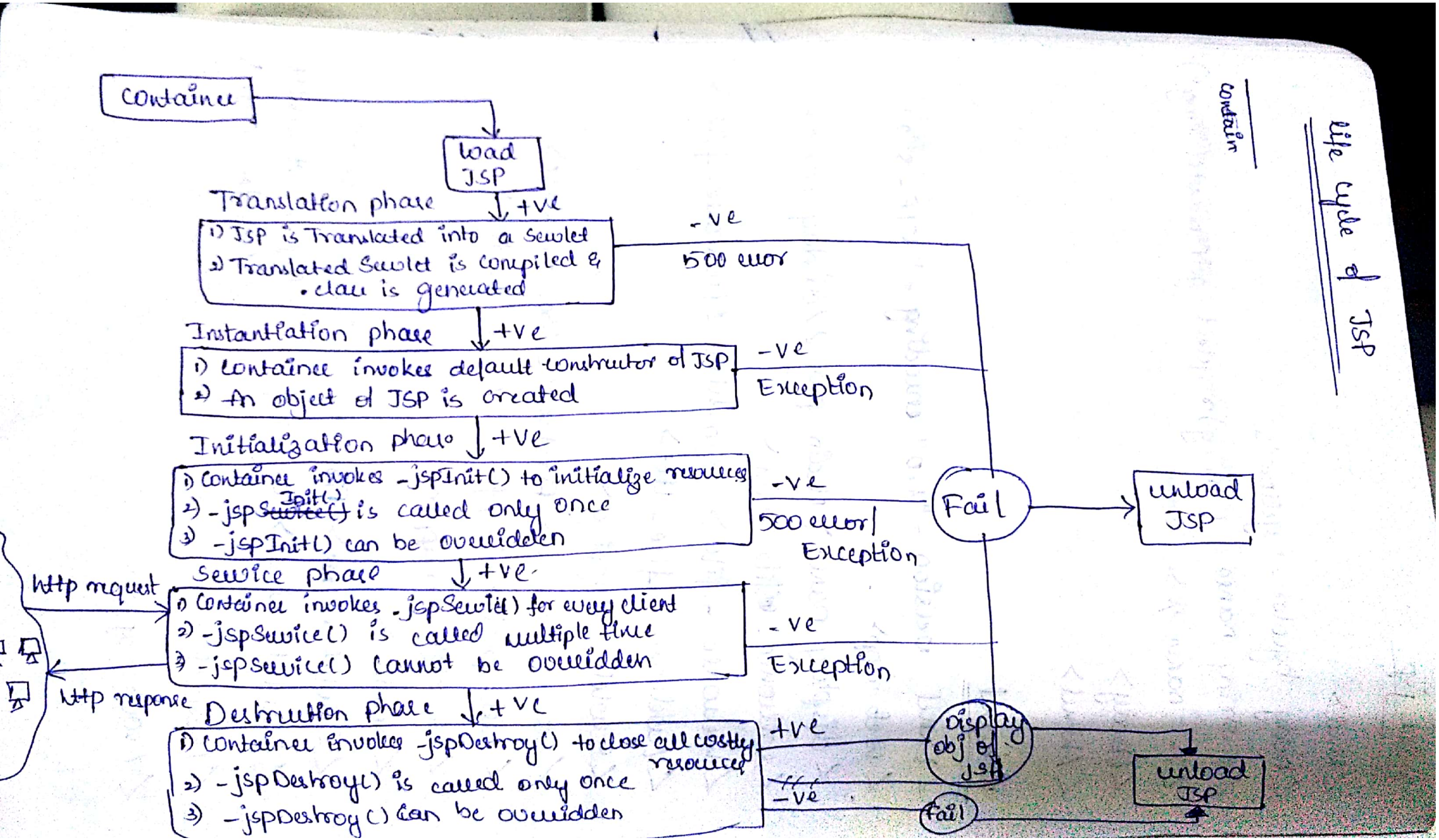
index.jsp

```
<html>
<body bgcolor = "yellow">
<%. string name = (string)request.getAttribute("nmg");
<%= name %>
</body>
</html>
```

web.xml

```
<?xml version = "1.0" encoding = "UTF-8"?>
<web-app>
<display name> chain-proj </display-name>
<welcome-file-list>
<welcome-file> form.html </welcome-file>
</welcome-file-list>
<servlet-mapping>
<servlet-name> Firstserv </servlet name>
<url-pattern> /fs </url-pattern>
</servlet-mapping>
<servlet>
<servlet-name> Firstserv </servlet-name>
<servlet-class> org.raj.chainApp.FirstServlet
</servlet-class>
</servlet>
</web-app>
```

life cycle of JSP



Note:

By default Jsp is multithreaded but it can be made single threaded by using below command

<% @ Page is ThreadSafe = "false" %>