Servlets

→ **Server :**

Server is a software, which manages all the resources along with which process the client request and serve the client request.

→ The different types of seovess are

(1) DB Seover
(2) Application Seoves.
(3) Web Seover.

(1) **DB Seover :**

→ DB Seover is used to deal only with data's.

Ex: Oracle, my SQL, MS-SQL, Desby, MongoDB, IBm db2, Sybase, Informix etc.

(2) **Application Seover :**

→ Application Server is used to execute a dynamic Application or a realtime Application.

→ **Dynamic Application :**

→ An application which pesforms all the 3 different types of logic such as presentation Logic, Persistence logic and Bussines Logic is known as a Dynamic Application.

Ex:- JBoss, IBm WebSphere, Oracle Weblogic

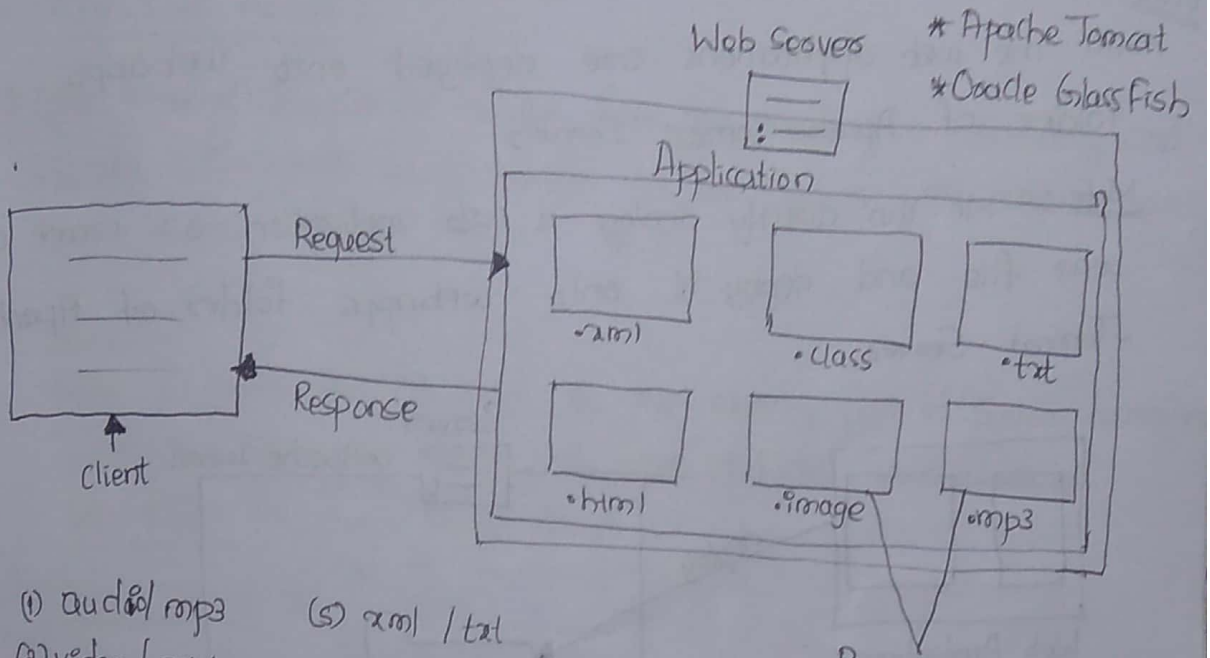
(3) **Web Seover :**

→ Web Seover is used to execute only web Applications.

Ex: Apache Tomcat, Oracle Glass Fish.

Web Server :-

→ Web Server is a software which stores and manages all the resources along with which process the client request and serve the client request.



(1) audio/mp3
(2) video / mp4
(3) html / txt
(4) image / jpeg
(5) xml / txt
(6) application / zip/pdf/json/exe

Deployment :-

→ Making all the resources available to the server is known as Deployment

→ There are 2 different types of deployment present namely
(i) Manual deployment
(ii) Automated deployment.

(i) Manual Deployment :

→ In this case, All the resources are made available to the server manually.

(ii) Automated Deployment :-

→ In this case, All the resources are made available to the server Automatically with the help of Automated tools such as ANT, Maven etc.

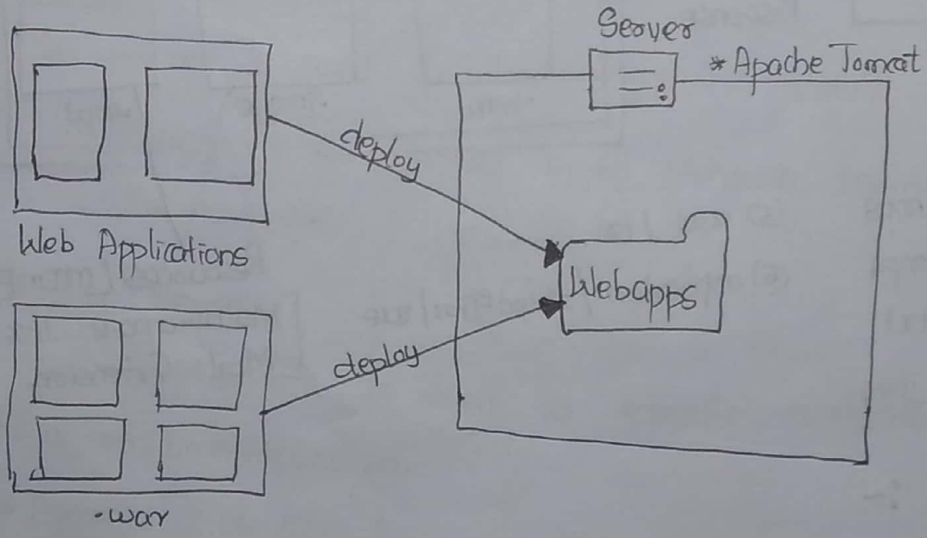
[By using Automated tools we create a Batch files. Batch files is one which can be accessed with the double click.]

Whenever a web application is compressed, then we deploy onto the server in the form of war file.
war refers to web archieve. (Compress) that means it is a compressed version of web application.

VIMP

→ All the web applications are deployed onto webapps folder of Apache Tomcat Server

Note :- We can directly deploy a web application or create a war file and deploy it onto webapps folder of Apache-Tomcat Server.



Apache Tomcat Server comes in two different varients. namely.

(i) exe varient
(ii) Zip varient.

VIMP

→ By default, the port number for Apache Tomcat Server is 8080

Pre-requisite for Apache Tomcat Server

There are 3 diff pre-requisite with respect to Apache Tomcat Server which are as follows:

(1) JAVA_HOME
(2) CATALINA_HOME
(3) path
(4) JRE_HOME (optional)

→ JAVA_HOME :

→ Open C → programfiles → java & jdk folder & copy the entire path which includes all the folders with respect to Apache Tomcat Server.

→ CATALINA_HOME :

→ Open ApacheTomcatSeever folder from the respective directory and copy the entire path which includes all the folders with respect to Apache Tomcat Seever.

→ Path :

→ put a semicolon (;) next to the existing path of System variables and copy paste the entire path which includes ~~all the folders~~ bin folder of Apache Tomcat Seever

→ JRE_HOME :

→ Open C-drive → programfile & open java & JRE folder & copy the entire path which includes all the folders withrespect to JRE.

Folders of Apache Tomcat seever

(1) bin
(2) conf
(3) lib
(4) logs
(5) webapps
(6) work.

(1) bin :— It contains a set of startup & shutdown batch files which is used to start & stop the ApacheTomcat Seever.

(2) conf :- It contains a set of configurations with respect to ApacheTomcat Seever

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

(4) webapps

(4) logs :- It is used to store all the log messages which are displayed on the seever console since seever console has limited memory

(5) webapps :- It is used deploy all the web applications onto Apache Tomcat Seever.

(6) work :- It is used to store the data with respect to translated Seevlet. [Conversion of a jsp into a seevlet is known as]

Steps to create a Basic Web Applications

(1) Create a new Dynamic Web Project
   (a) Generate a web.xml [WEB-INF]
(2) Create Resource → Home.html → Web Content
(3) Configure Resource → web.xml
(4) Deploy Resource → webapps → New Folder → WEB-INF
                                              → Home.html

home.html

```
<html>
<body bgcolor = "yellow" >
<h1> All are sleeping with eyes opened </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 >
```

Note!

→ We cannot run J2EE applications on all the Servers.

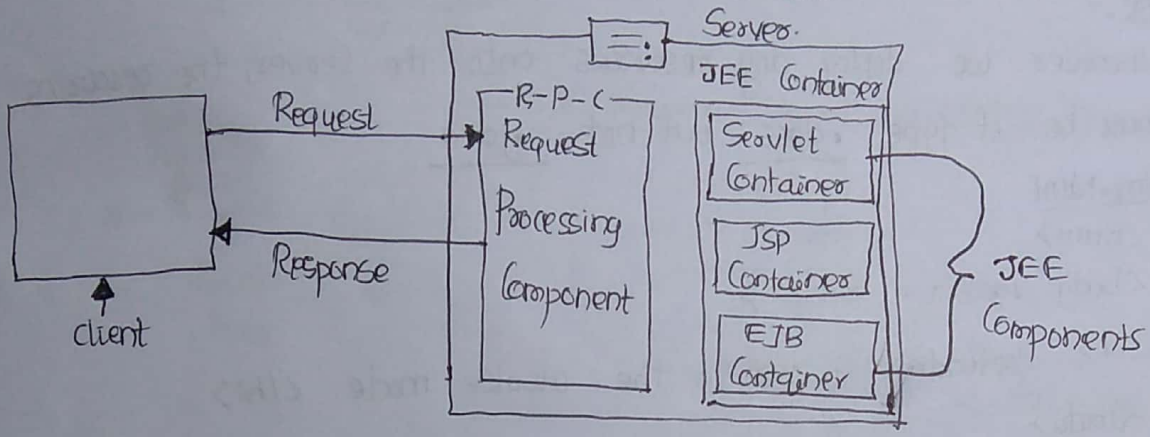
→ To run a J2EE application a particular Server, the Server must mandatorily contain a J2EE Container in it.

→ Can we run web & enterprise applications on Apache Tomcat Server?

Yes, Apache Tomcat Server version 6 and above internally refers to application Server.

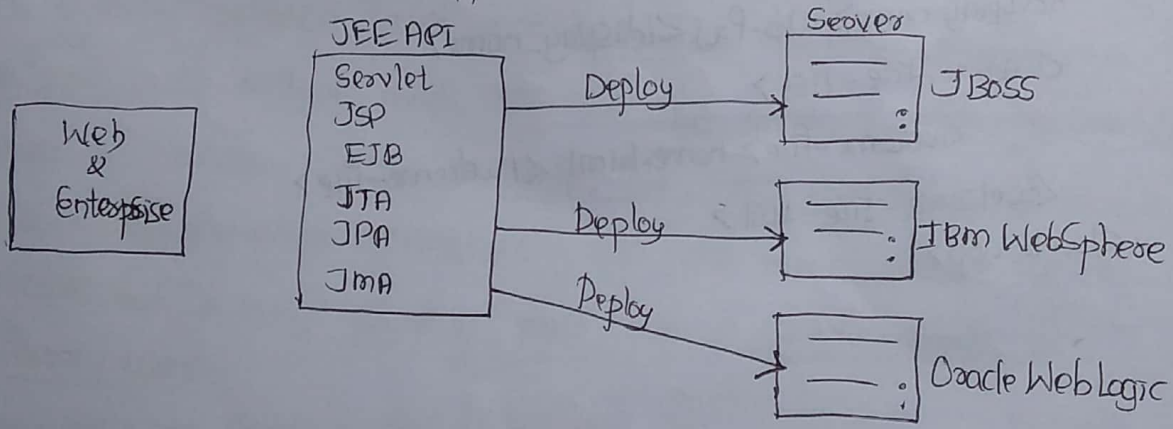
JEE Containers

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



JEE Applications

→ It is a specification for developing both web & enterprise application which is given in the form of abstraction API to achieve loose-coupling between application & the server.



Few of the JEE API's are

(1) Servlets
(2) JSP : Java Server Page
(3) EJB : Enterprise Java Bean.
(4) JTA : Java Transaction API
(5) JPA : Java Persistence API
(6) Jmp : Java Mail API

UTMP

Server Basically performs two important tasks namely
(1) manages all the resources.
(2) provides a run-time environments.

→ Web Server is used to execute only web applications.

→ Application Server is used to execute both web & enterprise application. (JEE Application)

Note :-

Whenever we deploy any resources onto the server, the resources must be of type .class but not .java

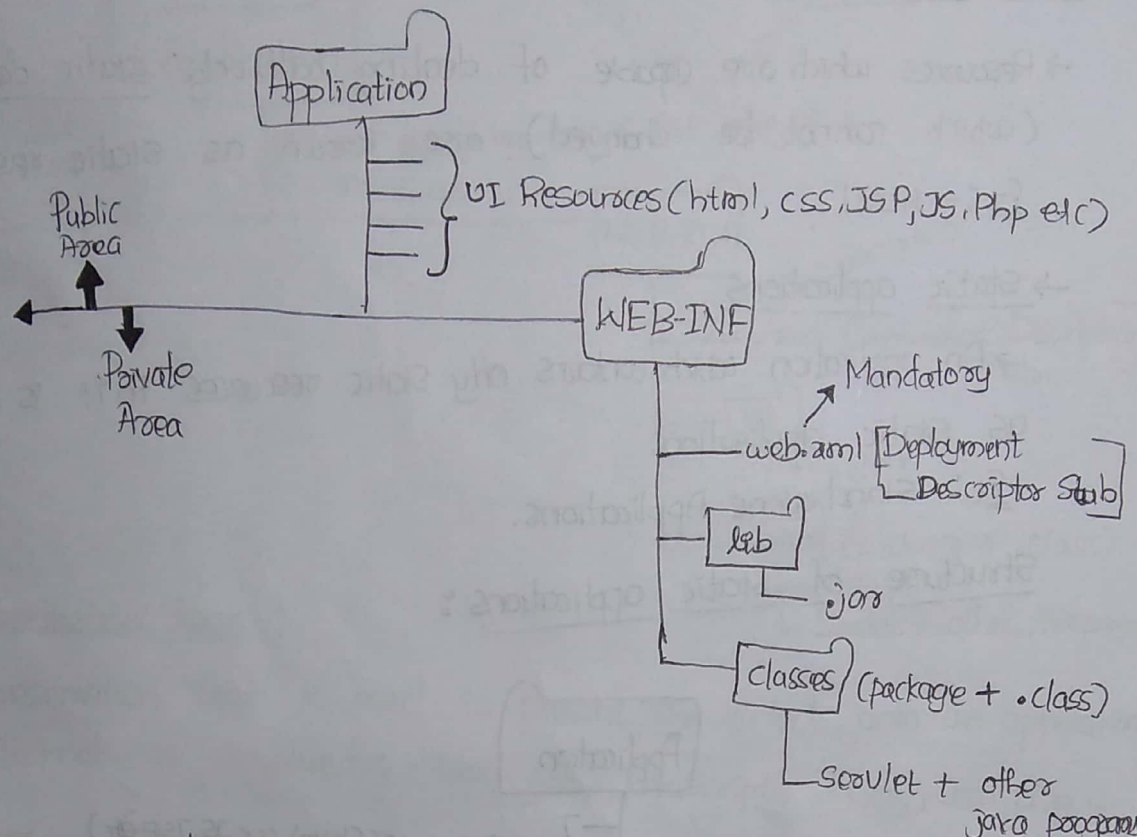
home.html

```
<html>
<body bgcolor = "yellow" >
<h1> Pretending to be in the awake mode </h1>
</body>
</html>
```

web.xml

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

Structure of JEE Applications



1→ Each & every application must mandatorily have minimum & maximum of only one web.xml without which the JEE containers fails to load an application whose it throws http: 404 Error [Resources not available]

2→ We can deploy multiple applications onto one single server. but in this case, each & every application must have a unique name or different name.

3→ Whenever we start a server, all the applications are loaded sequentially one after the another by the JEE container.

4→ At the time of application loading, the web.xml (compiled) or parsed by the JEE containers whose if there is any error present in web.xml, then JEE containers throws an exception called ParseException.

ParseException is a type of checked Exception.

Static vs dynamic Resources

→ Static resources :

→ Resources which are capable of dealing with only static data. (which cannot be changed) are known as static resources.

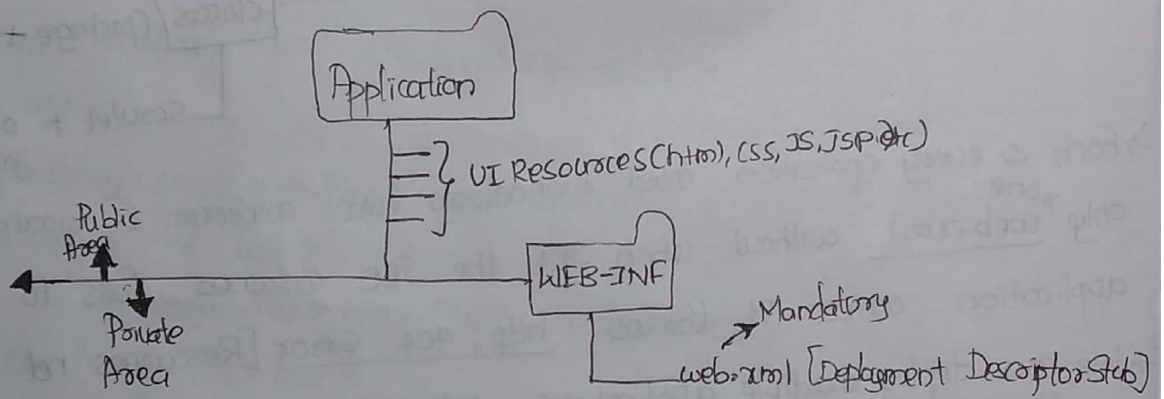
Ex :- html

→ Static applications

→ An application which contains only static resources in it is known as static applications

Ex : Stand alone Applications.

Structure of static applications :



→ Dynamic Resources :

→ Resources which are capable of dealing with only dynamic data. (which can be changed) are known as dynamic resources.

Ex :- JSP, Servlet, .php etc.

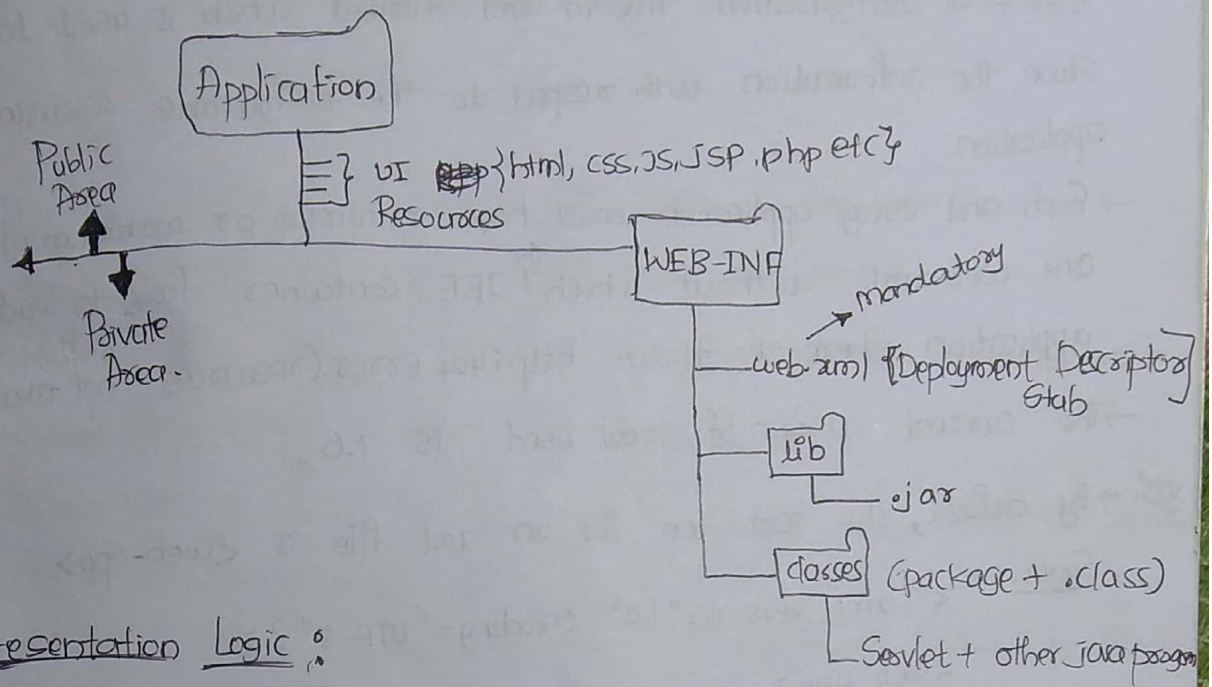
→ Dynamic resources generally deals with the DB server and can perform all the 3 diff types of logics such as presentation Logic, persistence Logic & Bussines logic

Dynamic Applications

→ An application which contains only dynamic resources in it is known as dynamic applications.

Ex : Any Real-time applications.

Structure of dynamic Application :



→ **Presentation Logic :**

→ Presentation Logic is used to present the contents onto an application

→ Technologies involved :- html, css, JavaScript, Jquery, Jsp, php etc

→ **Persistence Logic :**

→ Persist means to store. Persistence Logic is used to persist the data into the persistence system. [DataBase]

→ Technologies involved :- JDBC, SQL, hibernate etc

→ **Bussiness Logic :**

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

→ Technologies involved :- Servlet & other framework Components.

Note :

→ A dynamic application or real-time application is an integration of all the 3 different types of logic such as presentation Logic, Persistence Logic and Bussiness Logic.

Web.xml or deployment descriptor stub

→ It is a configuration file in xml format which is used to store the information with respect to the configurable resources of an application.

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

→ The current version of xml used is 1.0

VIMP → By default, the root tag for an xml file is <web-app>

Syntax :-

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
<!> ALL DOCUMENTS ARE HERE </!>
</web-app>
```

VIMP How are all the configurations made in web.xml understood by the JEE container?

→ All the custom tags or user defined tags are transformated or converted into 8-bit unicode format based on which all the configurations made in web.xml are understood by the JEE container.

UTF-8 → Unicode Transformation format - 8Bit

→ welcome-file or LandingPage

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

VIMP → index is considered to be the default welcome for the landing page which is automatically loaded by the JEE-container whenever we use an application.

→ we can explicitly make a file as 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 poiority based on the occurance.

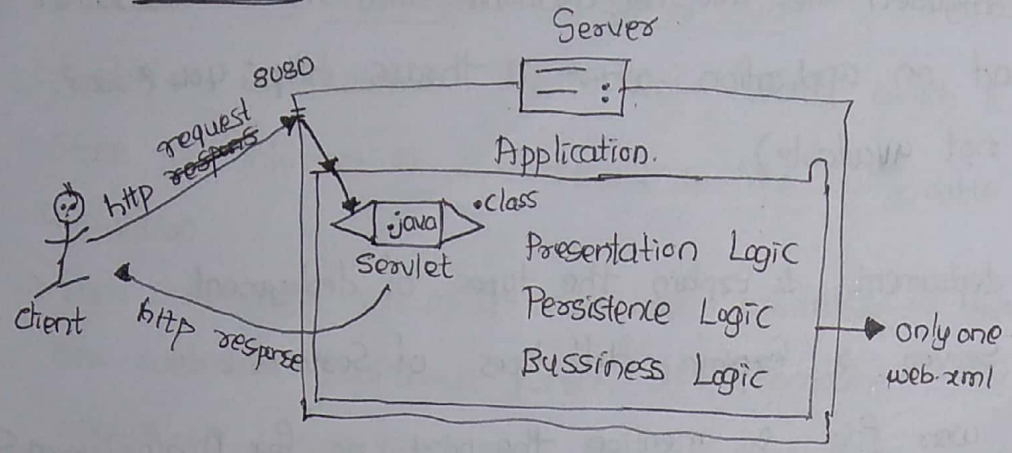
→ If the configured files are not available then the JEE Container fails to load an application where it throws http: 404 error. (resources not available).

1) define deployment & Explain the types of deployment
2) define Server & Explain diff types of Servers
3) define war file & mention the port no for Apache Tomcat Server
4) Explain the folders of Apache Tomcat Server.
5) Explain the structure of JEE Application in brief along with the diagram
6) define JEE Container & JEE application.
7) How are all the configurations made in web.xml understood by the JEE container?

index.html

```
<html>
<body bgcolor="yellow">
<h1> Ganesh is dancing with Sukhesh </h1>
</body>
</html>
```

Servlets



→ Servlet is a Server side java program which performs all the 3 different types of Logic such as presentation Logic, persistence Logics and bussiness 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 servlet present namely,

(1) GenericServlet
(2) Http Servlet

~~Generic Servlet~~

(1) GenericServlet

→ Since, It is not specific to any protocol or independent of protocol, Hence the name (called) GenericServlet.

→ GenericServlet doesnot support session.

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

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

→ GenericServlet Contains 3 different methods in it out of which one is an abstract method & other two are concrete methods.

→ The abstract method present in GenericServlet is named as service(). which has to be mandatorily overridden for two important reasons. namely

(i) Since, service() is an abstract method, overriding servie() is mandatory.

(ii) Since, service() is the only method which takes parameter called ServletRequest & ServletResponse which is responsible for processing the client request.

→ Whenever we override servie(), it throws an exception called ServletException & IOException.

```
@Override
+ void service(ServletRequest req, ServletResponse resp) throws ServletException,
  ↓                                                              IOException.
  Abstract
  method
```

→ The other two methods present in GenericServlet are init() & destroy() where overricling these methods are optional since these are concrete methods.

Writting a GenericServlet

→ Write a Servlet class which extends an abstract class GenericServlet as follows :

```
+class OurServlet extends javax.servlet.GenericServlet
  ↳ Servlet Class                ↓ abstract class
{ @Override
  + void service(ServletRequest req, ServletResponse resp) throws ServletException,
    ↓ abstract                                                    IOException
    method
  {
    Implementation
    Logic
  }
}
```

HttpServlet:

→ Since, it is specific to a particular type of protocol called HttpProtocol, Hence the name HttpServlet.

→ HttpServlet supports 'session'.

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

→ HttpServlet contains only concrete methods without any abstract methods.

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

VIMP

→ 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

} one more HttpRequest added recently i.e Patch

→ Whenever we override doXxx(), it throws an exception called ServletException & IOException.

```
@Override
# void doXxx(HttpServletRequest req, HttpServletResponse resp) -
protected   concrete method   - throws ServletException, IOException

Ex:- doPost()
     doGet()
     doPut()
```

Writing a HttpServlet:

→ Write a ServletClass which extends an abstract class called HttpServlet as follows:

```
+ class OurServlet extends javax.servlet.http.HttpServlet
{   ↳ Servlet class                          ↳ Abstract class
# void doXxx(HttpServletRequest req, HttpServletResponse resp) -
   ↳ Concrete method                - throws ServletException, IOException
{
   Implementation Logic
}
}
```

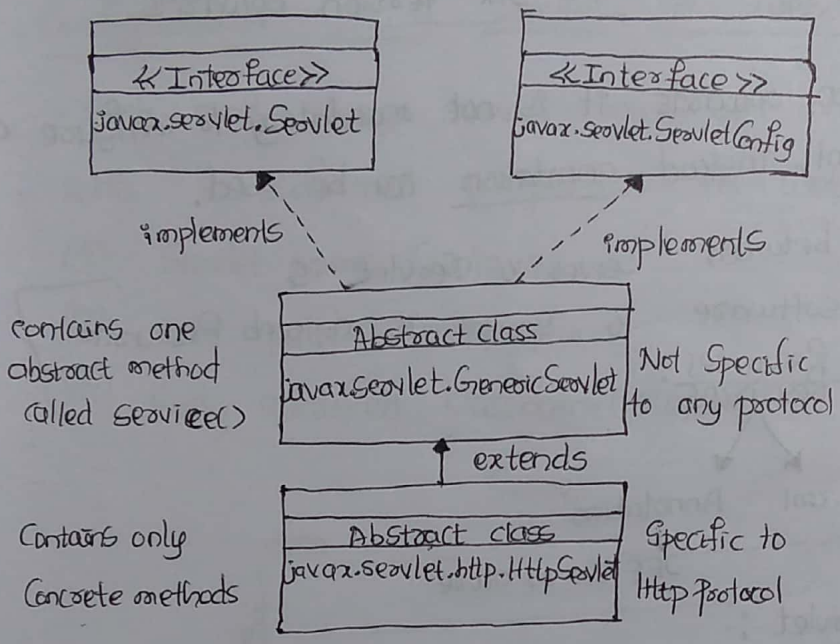
VIMP Job

Is there a service() present in HttpServlet?

→ In case of HttpServlet, service() is present as a concrete method which is not a good practice to be overridden. Since, HttpServlet depends upon the types of HttpRequest.

```
@Override
# void service(HttpServletRequest req, HttpServletResponse resp)
                    throws ServletException, IOException.
```

Servlet Heirarchy :-



List of methods :

Servlet :

init(ServletConfig)
service(ServletRequest req, ServletResponse resp)
destroy()
getServletConfig()
getServletInfo()

GenericServlet : init()

ServletConfig :

getParameterNames()
getInitParameter()
getServletContext()

HttpServlet :

doXxx(HttpServletRequest req, HttpServletResponse resp)
doPost(" _____ " _____ ")
doGet(" _____ ")
doPut(" _____ ")
doDelete(" _____ ")

Life Cycle methods of Servlet :

There are 3 different life cycle methods present for a Servlet namely,

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

→ web resources can be accessed based on unique url pattern (respective application vender provides).

→ Since, Servlet is a web resource, it has to be accessed based on the unique URL pattern.

→ Annotation is supported from JEE 3.x version onwards.

Note:

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

Major difference between Server & Servlet is
Server is a software & Servlet is a web Resource

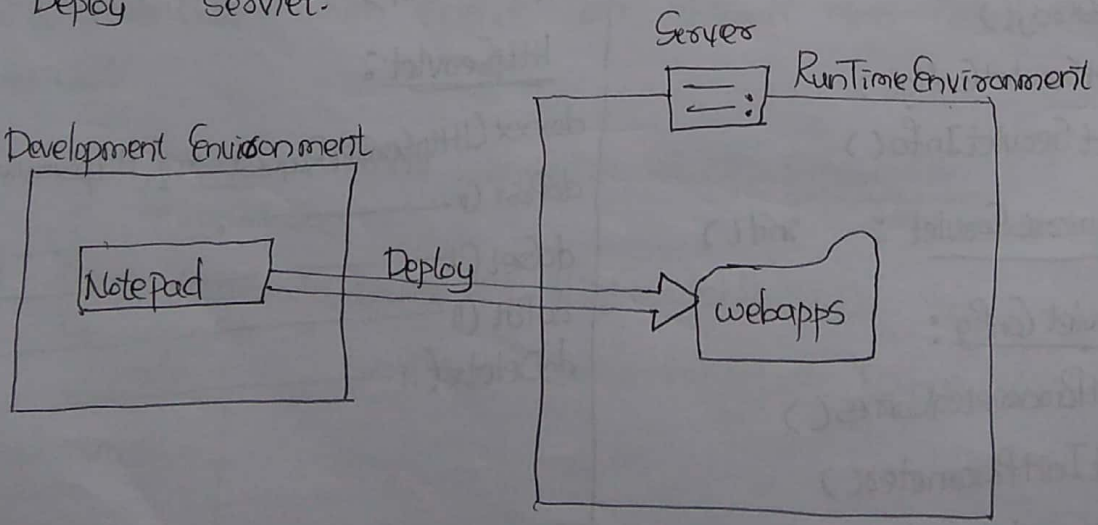
2 ways to Configure a Resources:

- web.xml
- Annotation → JEE 3.x or above

Criteria For Servlet :-

There are 3 different criteria present for a servlet namely:

1. Create Servlet.
2. Configure Servlet.
3. Deploy Servlet.



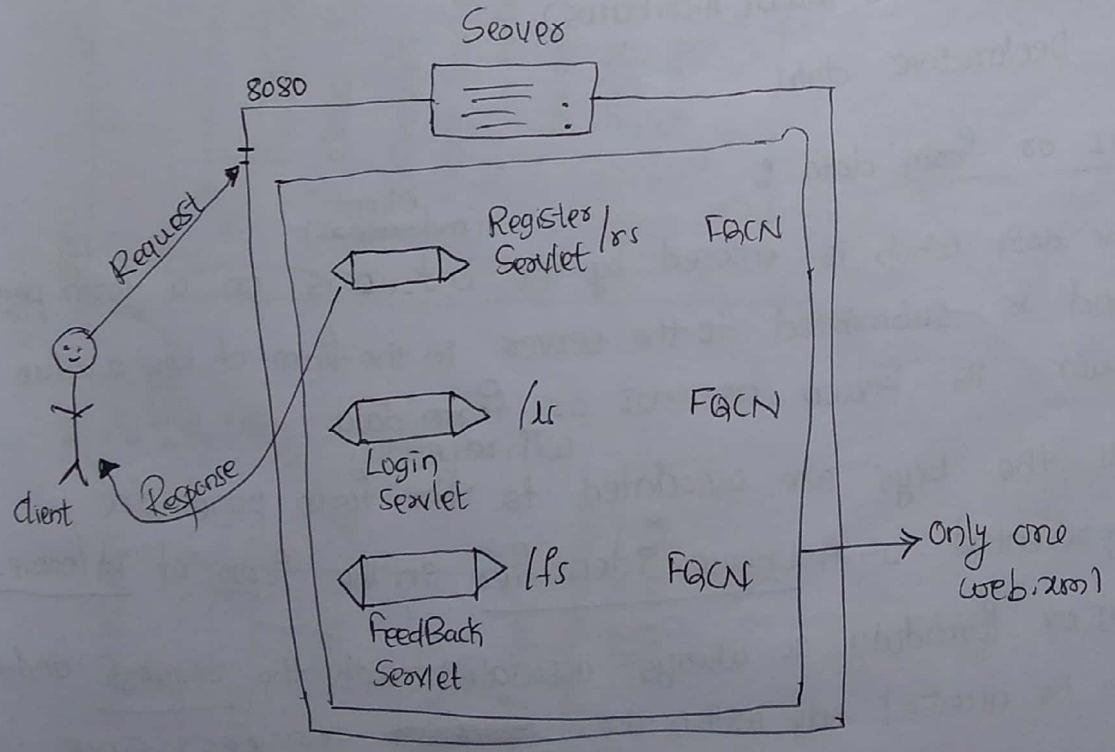
Steps to create a development Environment :

(1) Open Eclipse in JEE perspective.
(2) Open Navigator mode
(3) Create a new dynamic web project
(4) Create an appropriate package structure under Src folder.
(5) Generate a web.xml (right click on project → Java EE tools → Generate deployment descriptor stub)
(6) Add the Servlet Api.jar into the lib folder.

Configuration of a Servlet in web.xml :

→ Each and every servlet must be mandatorily configured with 3 different properties in web.xml namely -

(i) Servlet name (unique)
(ii) URL pattern (unique)
(iii) Fully qualified classname (FQCN)



Syntax :

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<web-app>
<servlet-mapping>
<servlet-name> Unique name </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 3 different types of data present namely :

(1) UI or form data

(2) programmatic data (Attributes)

(3) Declarative data

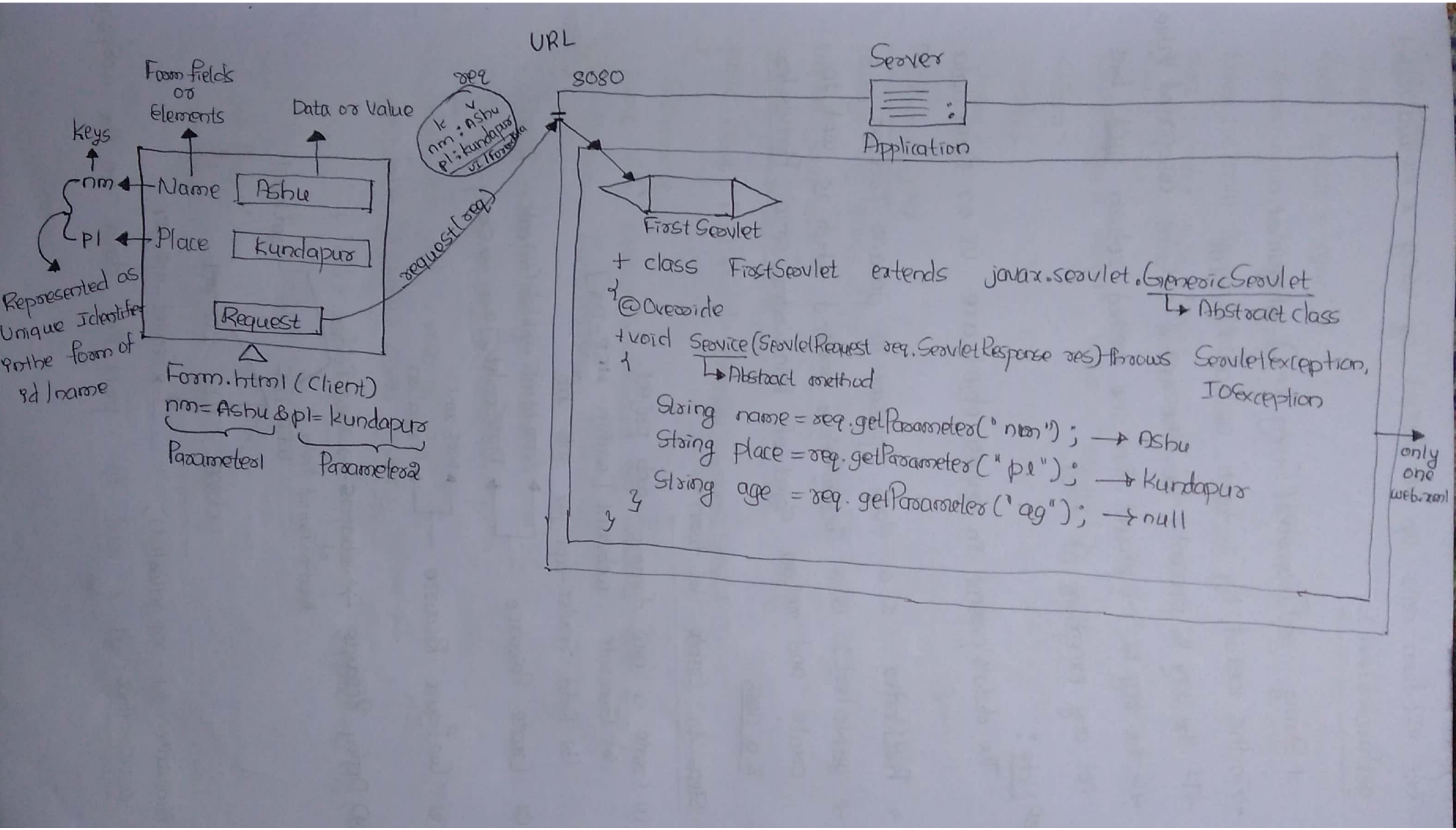
→ UI or form data :

→ The data which is entered by the client (~~end users~~) on a form page and is submitted to the server in the form of key & value pair is known as UI or form data

→ All the keys are associated with respect to ~~to the~~ form page are represented as a unique identifier in the form of id/name.

vimp

→ UI or formdata is always associated with the request and can be accessed only within the ~~service~~ service(). since service() is the only method which takes the parameter called ServletRequest & ServletResponse which is responsible for processing the client Request.


URL
Server
8080
Application
Form fields or Elements
Data or Value
Keys
nm
pl
Name
Place
Request
Ashu
kundapur
Represented as Unique Identifier for the form of id / name
req
nm : Ashu
pl : kundapur
request(req)
First Servlet
Form.html (Client)
nm=Ashu & pl= kundapur
Parameter1
Parameter2
+ class FirstServlet extends javax.servlet.GenericServlet
Abstract class
@Override
+void Service(ServletRequest req, ServletResponse res) throws ServletException, IOException
Abstract method
String name = req.getParameter("nm"); → Ashu
String place = req.getParameter("pl"); → kundapur
String age = req.getParameter("ag"); → null
only one web.xml

→ The UI/form data can be fetched by using a method called getParameter().

+ String getParameter(String key) → Argument

→ In this method, 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 return null but not any exceptions or error.

Imp Note:

The data's present in request object are UI or form data.

→ PrintWriter is a class present in java.io package.

→ getWriter() is a factory/helper method which is used to create and return object of PrintWriter since PrintWriter is a class

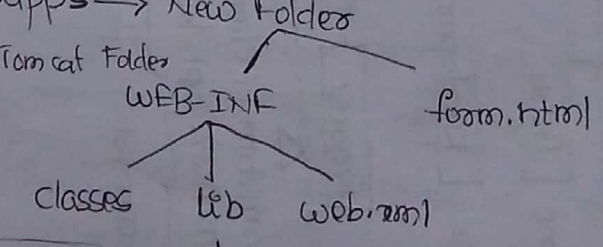
Steps to Create UI form:

(1) Create a new dynamic web project.
   (a) Generate web.xml [within WEB-INF]
   (b) Add Servlet-api.jar into lib

(2) Create Resource → form.html → Web Content
                    → FirstServlet.java → src

(3) Configure Resource → Web.xml
                       → Annotation

(4) Deploy Resource → Webapps of Apache-Tomcat Folder → New Folder



PrintWriter out = resp.getWriter();

out.println() - It is used to print the data in the web page.

form.html

```
<html>
<body bgcolor = "cyan" >
<form action = "nagaraj" >
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.jecm36.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 bgcolor = 'yellow' >"
          + "<h1> Dabba Fellow " +name + " " +place + " </h1>"
          + "</body> </html> ");
    out.flush ();
    out.close(); // presentation logic //
  }
}
```

web.xml

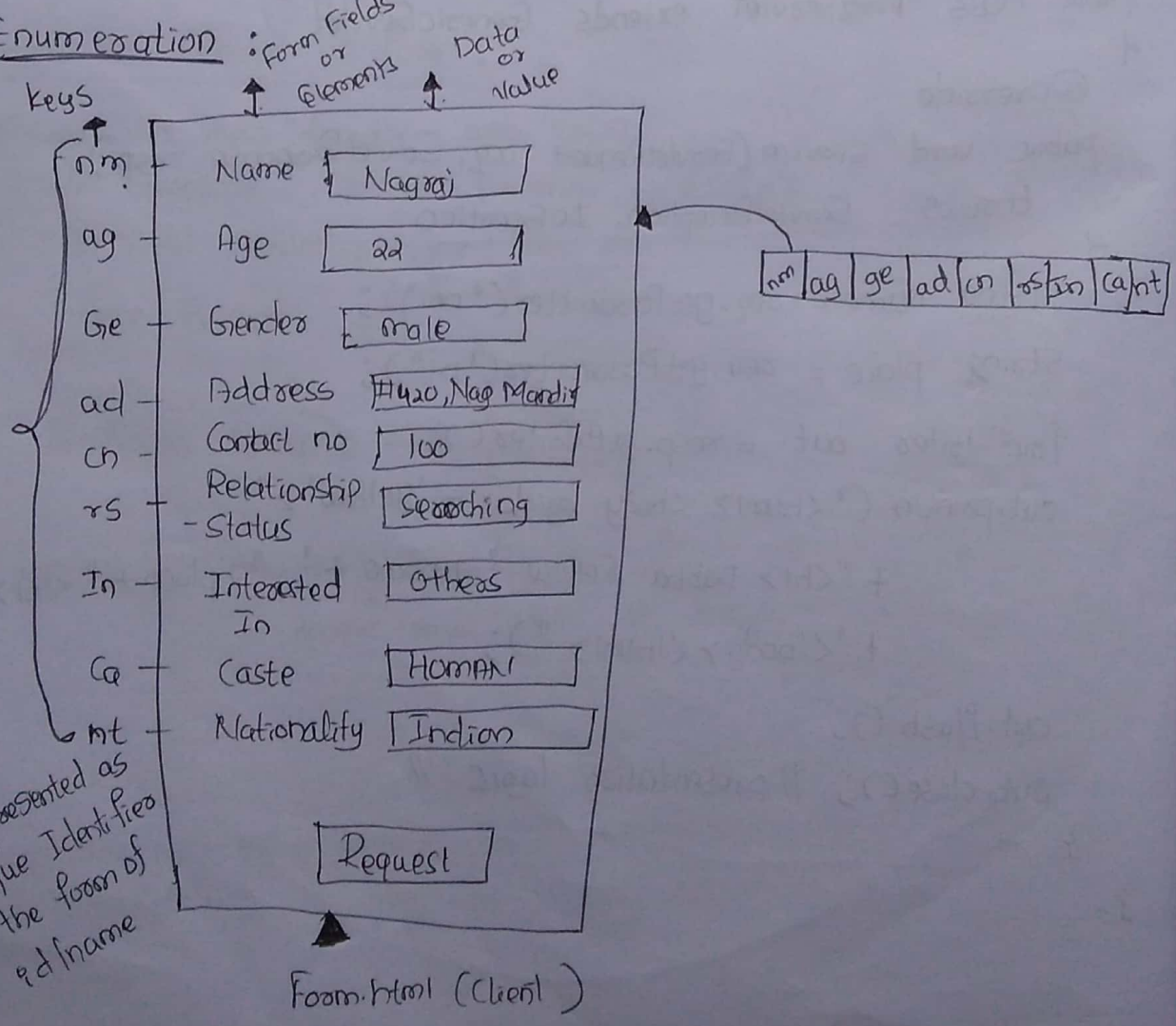
```
<?xml version="1.0" encoding="UTF-8" ?>
<web-app>
  <display-name> UI-Proj </display-name>
  <welcome-file-list>
    <welcome-file> form.html </welcome-file>
  </welcome-file-list>
  <servlet-mapping>
    <servlet-name> FirstServlet </servlet-name>
    <url-pattern> /nagaraj </url-pattern>
  </servlet-mapping>
  <servlet>
    <servlet-name> FirstServ </servlet-name>
    <servlet-class> org.jecm36.uiApp.FirstServlet </servlet-class>
  </servlet>
</web-app>
```

which is a alias name since it is to differentiate source file & servlet-name

form action

Fully Qualified classname

Enumeration : Form Fields or Elements | Data or value



→ getParameterNames() is used to fetch all the keys associated with respect to form page at once.

→ getParameterNames() can be used whenever there are 'n' numbers of form fields or form Elements present on the form page.

→ getParameterNames() returns an Enumeration of String type which contains all the keys associated with respect to the form page.

Defination 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);
  }
```

Code for UI/form data using Annotation Configuration:

index.html

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

First Servlet.java

```
package org.jecrn36.uiApp;
import java.io.*;
import javax.servlet.*; import javax.servlet.annotation.WebServlet;
@WebServlet("/fs")
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();
}
```

Servlet Life Cycle:

→ Servlet gets a life and starts its life cycle only when the Servlet object is created.

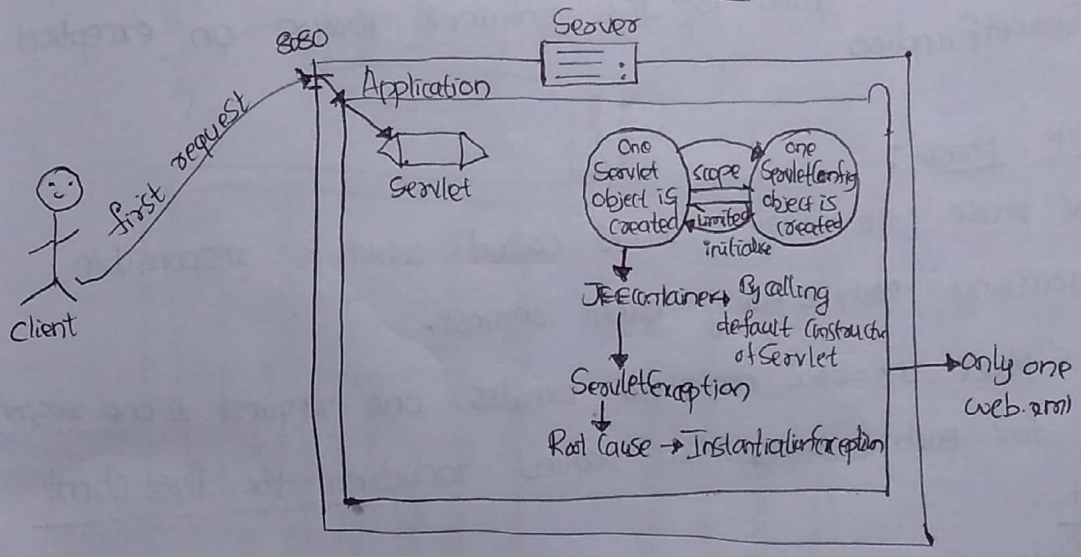
→ Servlet life cycle debits or represents the event/phases which takes place from Servlet object creation until Servlet Object destruction.

→ Note: Entire Servlet life cycle is managed by the JEE container or one of the responsibility of the JEE container is Servlet Life Cycle Management.

There are 4 different phases prent for Servlet Life Cycle namely:

1. Instantiation or object Creation phase
2. Initialisation phase
3. Service phase
4. Destruction phase.

1. Instantiation or object Creation phase:



→ In this phase, the Servlet object has to be Created

→ Whenever a client makes a first request to a Servlet, one Servlet object is created by the JEE Container by calling default Constructor of Servlet and now Servlet life cycle begins.

⇒ If the JEE container doesnot find, the default constructor of Servlet, then JEE Container throws an exception called ServletException with a root cause InstantiationException (Object not Created)

→ Immediatly, after the Servlet object creation, one ServletConfig object is created by the JEE container which is used to initialise the resources of Servlet Object.

→ Hence, Scope of ServletConfig object is limited to that particular Servlet Object.

(2) Initialization phase:

→ In this phase, the Servlet object has to be initialized. by using ServletConfig
- init() which takes ServletConfig as a parameter which is used to initialize the resources of Servlet Object

→ init() is always called by the JEE container only one.

→ If this phase fails, then the JEE container throws an exception called ServletException.

(3) Service phase:

→ In this phase, the service() is called which is responsible for processing each & every client request.

→ In this phase, the JEE container creates one request & one response object for each & every client request including the first client request.

→ By default, Servlet is multithreaded but it can be made as single threaded in two different ways namely:

(1) By writing a Servlet class which implements a marker interface by name called SingleThreadedModel.

(2) By making service() as Synchronized.

Note:

→ SingleThreadedModel is depricated interface(outdated)

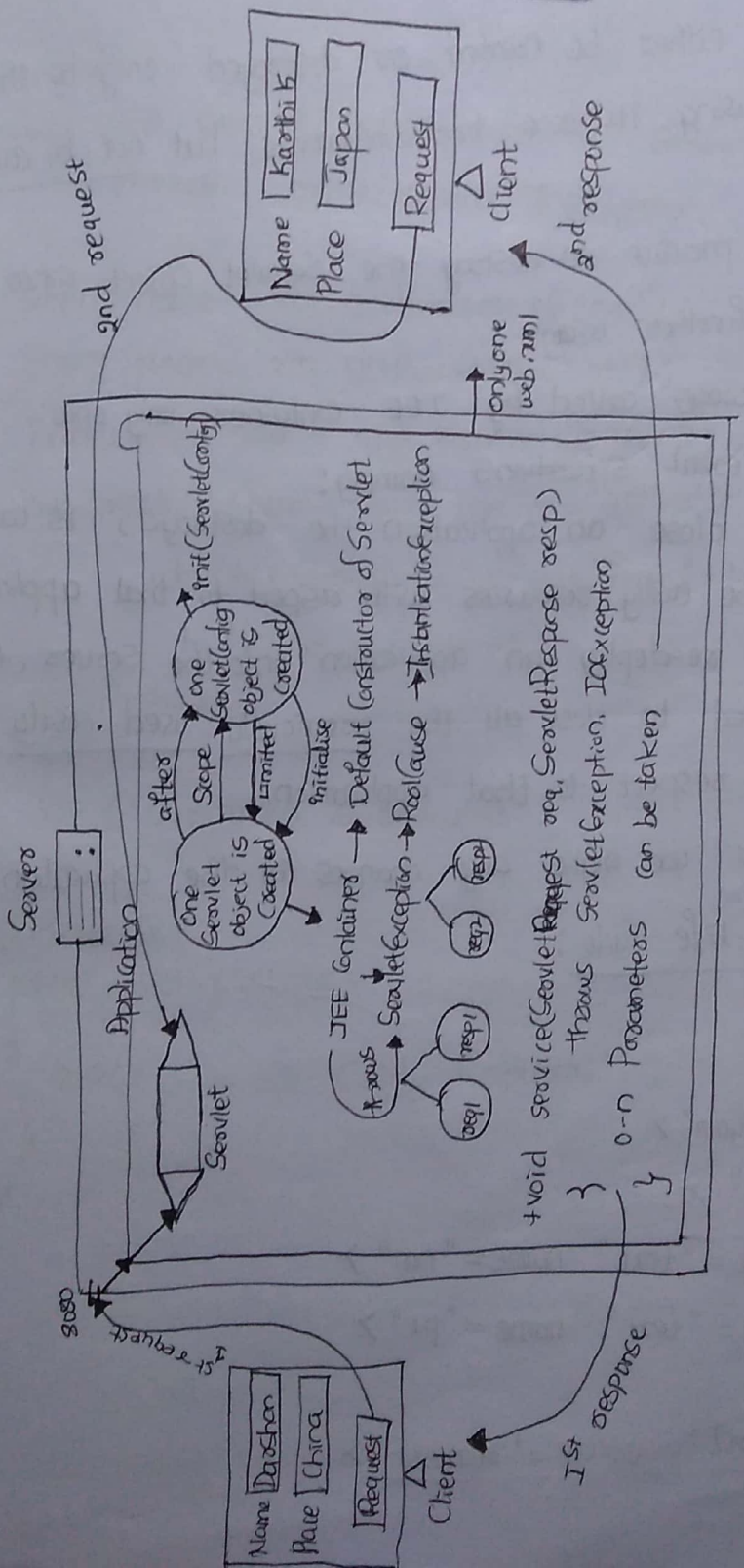
→ By default service() is multithreaded that means one thread is created for every client request and service() is executed.

→ Whereever a client makes second or subsequent client request to the same Servlet again, only service() will be executed but the Servlet object will not be created again.

→ service() is always called by the JEE container for multiple times or 'n' number of times.

→ If this phase fails then the JEE container throws an exception called ServletException.

Servlet LifeCycle Internal Architecture :



(4) Destruction phase :-

→ In this phase, the destroy() is called to close all the costly resources but not to destroy the Servlet Object.

→ destroy() is called by the JEE Container only once.

→ If this phase fails, then the JEE Container doesnot throw any error or exception instead the performance of an application decreases.

Note :

① Servlet object can either be created or destroyed only by the JEE container by using its own implementations but not by calling destroy().

② It is not a good practice to destroy the Servlet Object. Since, it is needed for the further usage.

→ Destroy() is always called by JEE Containers only once but in two different situations namely :

① whenever we close an application the destroy() is called to close all the costly resources with respect to that application.

② whenever we re-deploy an application onto the Server, the destroy() is called to close all the previously used costly resource with respect to that application.

Re-deploy → whenever we make any changes in the application.

Code for Servlet-Life-Cycle :

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

```
import java.io.*;
public class FirstServlet extends GenericServlet
{ public FirstServlet()
  {
   SOP(" Servlet object is created");
  }
 @Override
 public void init(ServletConfig config) throws ServletException.
 {
   SOP("Servlet object is initialized");
 }
 @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 bgcolor='yellow'>"
          +"<h1> Studen is "+name+"from"+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> FacV </servlet-class>
 </servlet>
 </web-app>
```

Load on Startup :-

→ Servlet gets a life and starts its life cycle only when a Servlet object is created.

→ Servlet object can always be created in two different ways namely:

(1) Whenever a client makes a first request to a Servlet, one Servlet object is created by the JEE container by calling the default constructor of Servlet.

(2) In case of Load on startup, <load-on-startup> is a tag which is a sub tag of Servlet tag (<servlet>)

→ In case of Load on Startup, the JEE container creates a Servlet object by calling the default Constructor of Servlet. at the time of Server Startup. without waiting for the first Client request. So that the delay time made by the

first client request can be avoided. which helps in increasing the performance of an application.

→ Load-on-Startup must always be configured with a positive integer value but the JEE container gives the priority based on the ~~occurance~~ Lowest +ve integer value.

→ Whenever two Servlets are configured with the same +ve integer value for Load-on-Startup, sequential execution takes place.

~~Code for Servlet LiR~~

→ Whenever a Load-on-Startup is configured with a negative integer value, then the JEE container creates a Servlet object based on first client request without throwing any error or exception.

→ Code for Servlet life cycle in case of Load-on-Startup using 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.jeerm36.lifecycleApp.FirstServlet </servlet-class>
    <load-on-startup> 3 </load-on-startup>
  </servlet>
</web-app>
```

code for Servlet life-cycle incase of annotation

⇒ we have to rename form.html to index.html

```
import javax.servlet.annotation.WebServlet;
@WebServlet(urlPatterns = "/fs", loadOnStartup = 3)
public class FirstServlet extends GenericServlet
{
        ⇒ Servlet Code ⇐
}
```

ServletConfig:

→ ServletConfig is an interface which is present in javax.servlet package.

→ Since it is an interface, an implementation object of this interface is created by JEE container by calling a factory/helper method called getServletConfig() immidially after the servlet object creation.

```
javax.servlet.ServletConfig config = getServletConfig();
```

→ only one implementation object of ServletConfig interface is created by the JEE container for that particular Servlet object.

→ The data's present in config object are init or initialization parameter which is made available only for that particular Servlet object

→ Hence scope of ServletConfig object is limited to that particular Servlet object.

init or initialization parameter:

→ It is the data in the form of key & value pair which is used to initialise the resources of Servlet object.

→ Any no of init or initialization parameter can be declared for a Config object.

→ Init or initialization parameter can be declared in two different ways namely:

① In web.xml.

② Through Annotation.

→ Declaration of init or initialization parameter in web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
<servlet>
<init-param>
<param-name> key1 </param-name>
<param-value> value1 </param-value>
</init-param>
<init-param>
<param-name> key2 </param-name>
<param-value> value2 </param-value>
</init-param>
</servlet>
</web-app>
```

Declaration of init or initialization parameter through Annotation

Note:
→ In case of web.xml the init or initialisation parameter must be mandatorly declared inside the Servlet tag since the scope of Config object is limited to that particular Servlet object.

Declaration of init or initialization parameter through Annotation

```
@WebServlet(name="FirstServ", urlPatterns="/fs",
        initParams={@WebInitParam(name="key1", value="value1"),
                    @WebInitParam(name="key2", value="value2"),
                    @WebInitParam(name="key3", value="value3")
                   })
public class FirstServlet extends GenericServlet
{
    ===
}
```

Note:-
→ In case of Annotation the init or initialisation parameter must be mandatorly declared inside the Source code of particular Servlet.

Fetching the init or initialisation parameter :

→ The init or initialisation parameter present in config object can be fetched by using a method called getInitParameter().

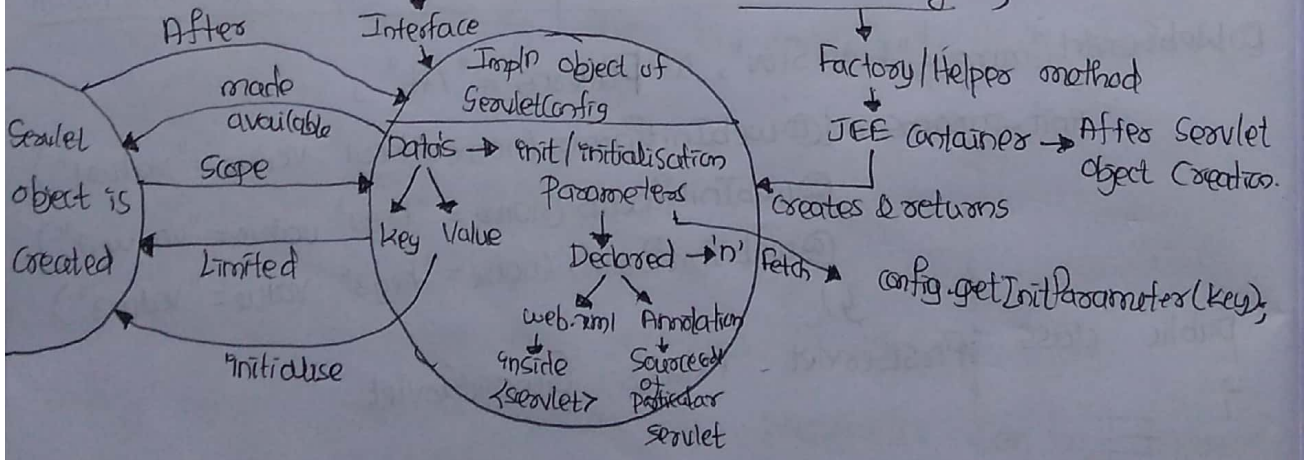
+ String getInitParameter(String key)
  ↳ Argument

config.getInitParameter("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.

→ getInitParameterNames() is used to fetch all the keys associated with respect do the Config object at once.
→ getInitParameterNames() can be used whereves there are 'n' no of init or initialization parameters declared for a Config object.
→ getInitParameterNames() returns an Enumeration of String type which contains all the keys associated with respect to Config object.

javax.servlet.ServletConfig config = getServletConfig();



Syntax of Enumeration of String type:

```
+ Enumeration<String> keynum = config.getInitParameterNames();
while(keynum.hasMoreElements())
{
    String key = keynum.nextElement();
    String val = config.getInitParameter(key);
}
```

ServletContext:

→ At a time of application Loading, the JEE container creates an environment for every application which is referred as Context

→ ServletContext is an interface present in javax.servlet package

→ Since it is an interface, only one implementation object of this interface is created for one entire application by the JEE container by calling a Factory/helper 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. Shared throughout the application.

→ The data's present in context object are context parameters & attributes which is made available throughout the application.

imp → Hence, the scope of Context object is throughout the application or context scope is also known as application scope.

→ The data's present in context object (Context parameters & attributes) are also known as application wide resources or application scope resources.

Context Parameters:

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

→ Any no of Context parameters can be declared for a context object.

imp → context parameter can always be declared only in web.xml.

Declaration of context parameters 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-value>
</context-param>
<servlet>
</servlet>
</web-app>
```

Note:

In case of web.xml, the context parameters must be mandatorily declared outside the Servlet tag. Since scope of context object is throughout the application.

Fetching the Context parameters

→ The context parameters present in context object can be fetched by using a method called getInitParameter().

+ String getInitParameter(String key)
  ↳ Argument

```
ctx.getInitParameter("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.
→ getInitParameterNames() is used to fetch all the keys associated with respect to context object at once.
→ getInitParameterNames() can be used whenever there are 'n' number of Context parameter declared for a context object.

time

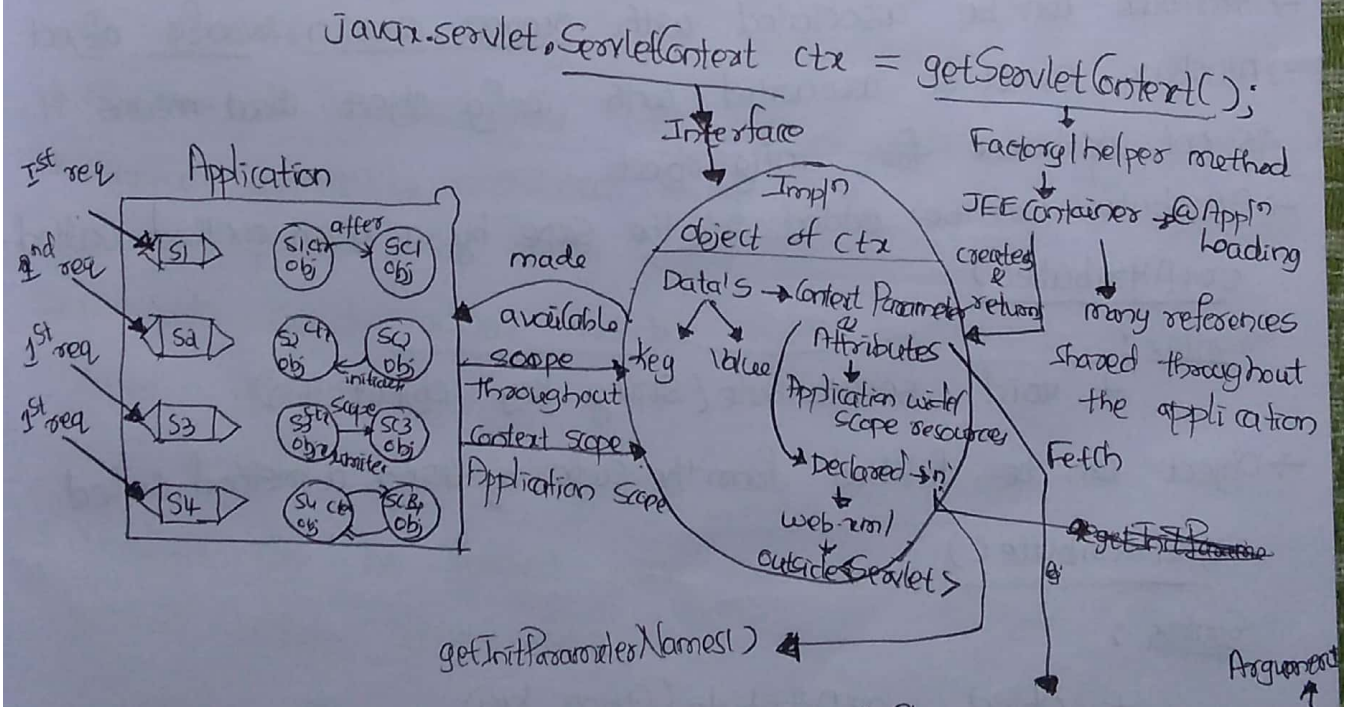
→ getInitParameterNames() returns an enumeration of String type which contains all the keys associated with respect to the context object.

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

Note:

→ The total no of Servlet objects which can be created for one entire application is 'n'. no of Servlet objects.

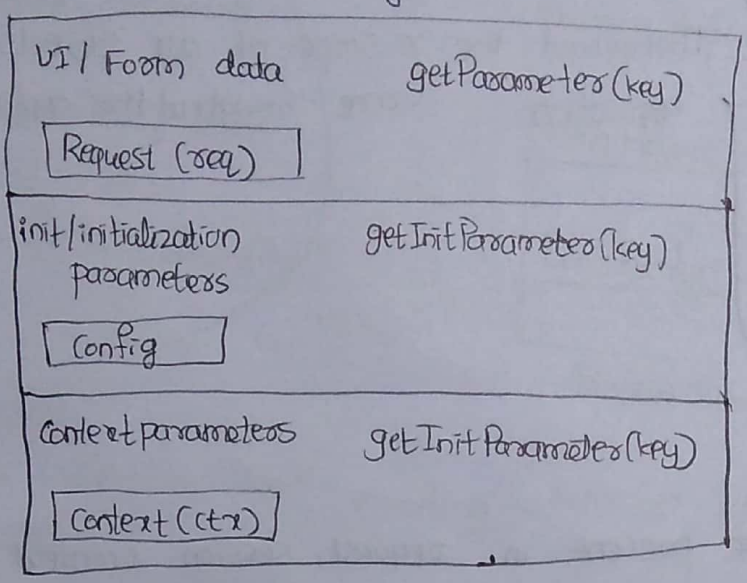
→ The total no of ServletConfig object for one entire application is 'n'.



→ Based on the reference the config & context, we differentiate init & context parameters.

Parameter :-

It is the data in the form of key & value pair where key must be unique String & value can be a any 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

Ex:- Student object, Array object, Bean object etc.

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

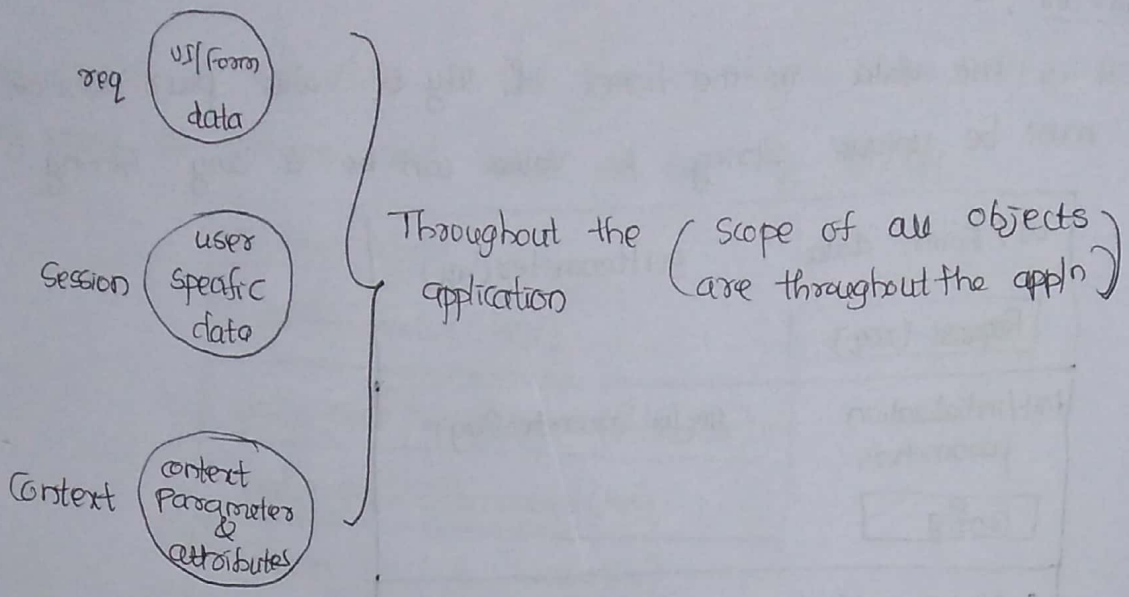
→ Attribute cannot be associated with config object i.e attribute is not applicable for config object

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

+ void setAttribute(String key, Object obj)

→ Attribute can be fetched back from the scope by using a method called getAttribute().

public Object getAttribute(String key)

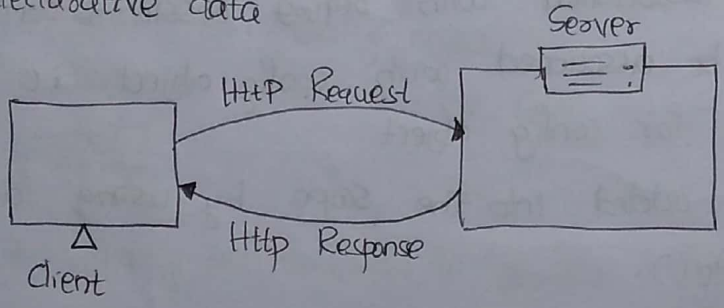


Programmatic Data :

The data's which are present in request, session & context object are known as programmatic data. Since the scope of all these objects are throughout the application

Declarative Data

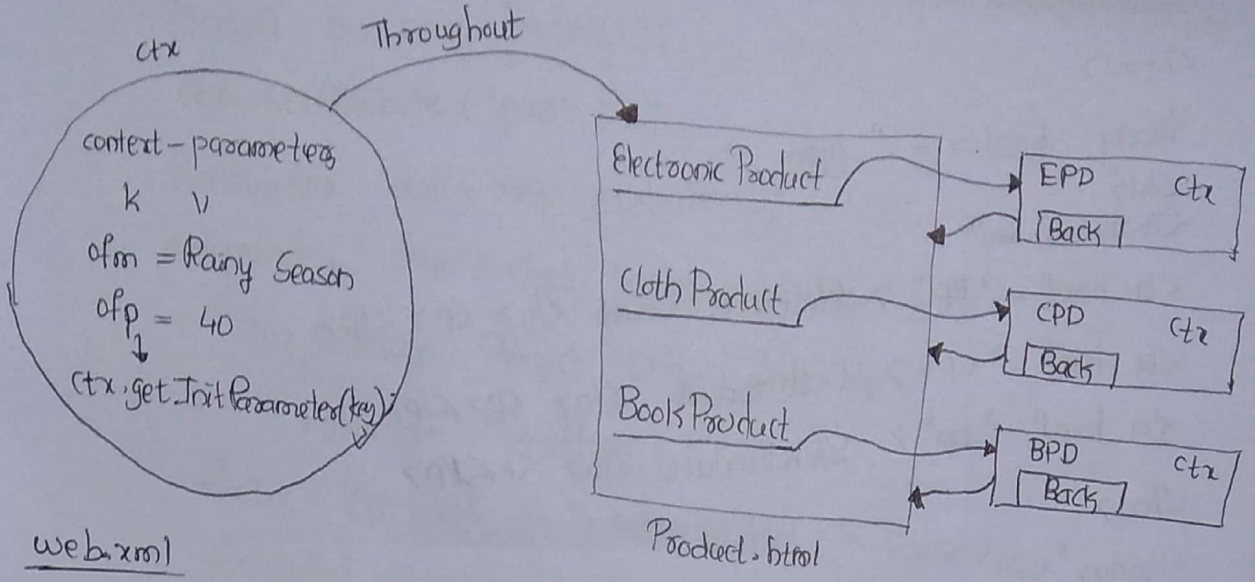
The data's which are declared in xml file are known as declarative data



Contents of HttpServletRequest

HttpServletRequest is an interface present in javax.servlet.http package

1. UI/form data
2. URL
3. Types of Request
4. Client - information
   a) IP address
   b) Browser information.(user agent)
5. Cookie → It is an ter
6. MIME(accept)



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> ofm </param-name>
  <param-value> RainySeasonSales </param-value>
  </context-param>
  <context-param>
  <param-name> ofp </param-name>
  <param-value> 40 </param-value>
  </context-param>
</web-app>
```

product.html

```
<html>
<body bgcolor = " cyan ' >
<h1>
<a href = "ep" > ElectronicProduct </a> <p> </p>
<a href = " cp" > ClothProduct </a> <p> </p>
<a href = " bp" > BookProduct </a> <p></p>
</h1>
</body>
</html>
```

Product.java

```
import java.io.*;
public class Product
{
   public String name;
   public double price;
}
```

ElectronicServlet.java

```
package org.jecm36.contextApp;
import java.io.*;
import java.servlet.*;
import javax.servlet.annotation.WebServlet;
@WebServlet("/ep")
public class ElectronicServlet extends GenericServlet
{
  @Override
  public void service(ServletRequest req, ServletResponse resp)
     throws ServletException, IoException.
  {  ServletContext ctx = getServletContext();
     String offmsg = ctx.getInitParameter("ofm");
     String offper = ctx.getInitParameter("ofp");
```

```
        // Context (ctx) into Scope //
        Product p = new Product();
        ctx.setAttribute("prd", p);
        PrintWriter out = resp.getWriter();
        out.println("<html><body bgcolor='orange'>"
        +"<h1>Electronic Product Details</h1><p>"
        +"" +offmsg+" "+offper+"%off"
        +"<br><a href=\"product.html\">Back</a>"
        +"</body></html>");
        out.flush();
        out.close();
    }
}
```

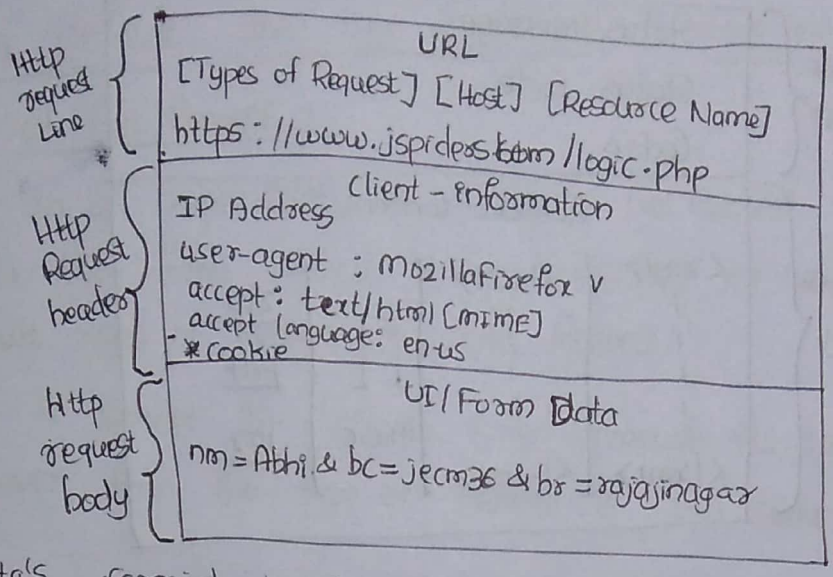
ClothServlet.java

```
import javax.servlet.annotation.WebServlet;
@WebServlet("/cp")
public class ClothServlet extends GenericServlet
{
@Override
public void service(ServletRequest req, ServletResponse resp)
        throws ServletException, IOException.
{
    ServletContext ctx = getServletContext();
    String offmsg = ctx.getInitParameter("ofm");
    String offper = ctx.getInitParameter("ofp");
    PrintWriter out = resp.getWriter();
    out.println("<html><body bgcolor='pink'>"
    +"<h1>Cloth Product Details</h1><p>"
    +"" +offmsg+" "+offper+"%off"
    +"<br><a href=\"product.html\">Back</a>"
    +"</body></html>");
    out.flush();
    out.close();
}
}
```

BookServlet.java

```
import javax.servlet.annotation.WebServlet;
@WebServlet("/bp")
public class BookServlet extends GenericServlet
{
  @Override
  public void service(ServletRequest req, ServletResponse resp)
      throws ServletException, IOException
  {
    ServletContext ctx = getServletContext();
    String offmsg = ctx.getInitParameter("ofm");
    String offper = ctx.getInitParameter("ofp");
    // Fetch the Context(ctx) object back from Scope
    Product pd = (Product) ctx.getAttribute("prd");
    PrintWriter out = resp.getWriter();
    out.println("<html> <body bgcolor='yellow'>"
    + "<h1> Book Product Details </h1> <p>"
    + " " +offmsg+ " " +offper+ "% off "
    + "<br> <a href=\"product.html\"> Back </a> "
    + "</body> </html>");
    out.flush();
    out.close();
  }
}
```

Module of HttpServletRequest :



→ The data's carried to servers as a part of Http request body & the data's are not displayed even to end users. Hence the data's are secured.

Types of HttpRequest

→ There are 8 diff types of HttpRequest present namely

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

Contents of HttpServletResponse

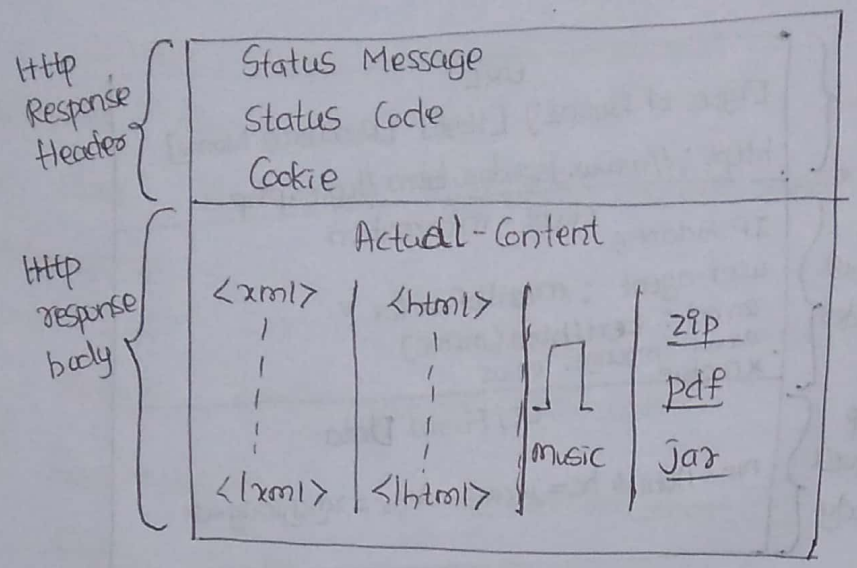
HttpServletResponse is an interface present in javax.servlet.http package.

(1) Status message
(2) Status code
(3) Cookie
(4) Actual-Content

(1) Status Message & Status Code

*** / **	Status code	StatusMessage
4xx	400	Client Error
5xx	500	Server Error
2xx	200	Success Message.

Module of HttpServletResponse:



Http Post:

→ Post request is used to post some contents or data from the client to the server.

→ Post request deals with unlimited data.

imp → Post request is non-idempotent. → [Not Specific to any resources]

→ Post request cannot be bookmarked [saved]

imp → In case of Post request, the data's are carried to server as a part of HttpRequest Body which is not displayed even to the endusers (client) Hence the data's are secured.

→ Whenever we deal with 'n' no 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 only limited data i.e 1024 characters.

imp → Get request is idempotent. [Specific to a resources]

→ Get request can be bookmark

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

→ Whenever we deal with a link and the type of request is Get request.

→ Whenever the type of request is not mentioned or configured, then by default the type of Request is Get request.

Situations of Get Request :

(1) Clicking on a hyperlink is of type Get Request.
(2) Whenever the form method is not mentioned or configured then by default type of request is Get Request.
(3) Whenever a Servlet is directly called through its URL pattern in the browser, then the type of request is Get request.

Http Put :

→ Put request is similar to that of Post request but it is used to update the existing resources.

Http Delete :

→ Delete request is used do delete the contents or data from the server.