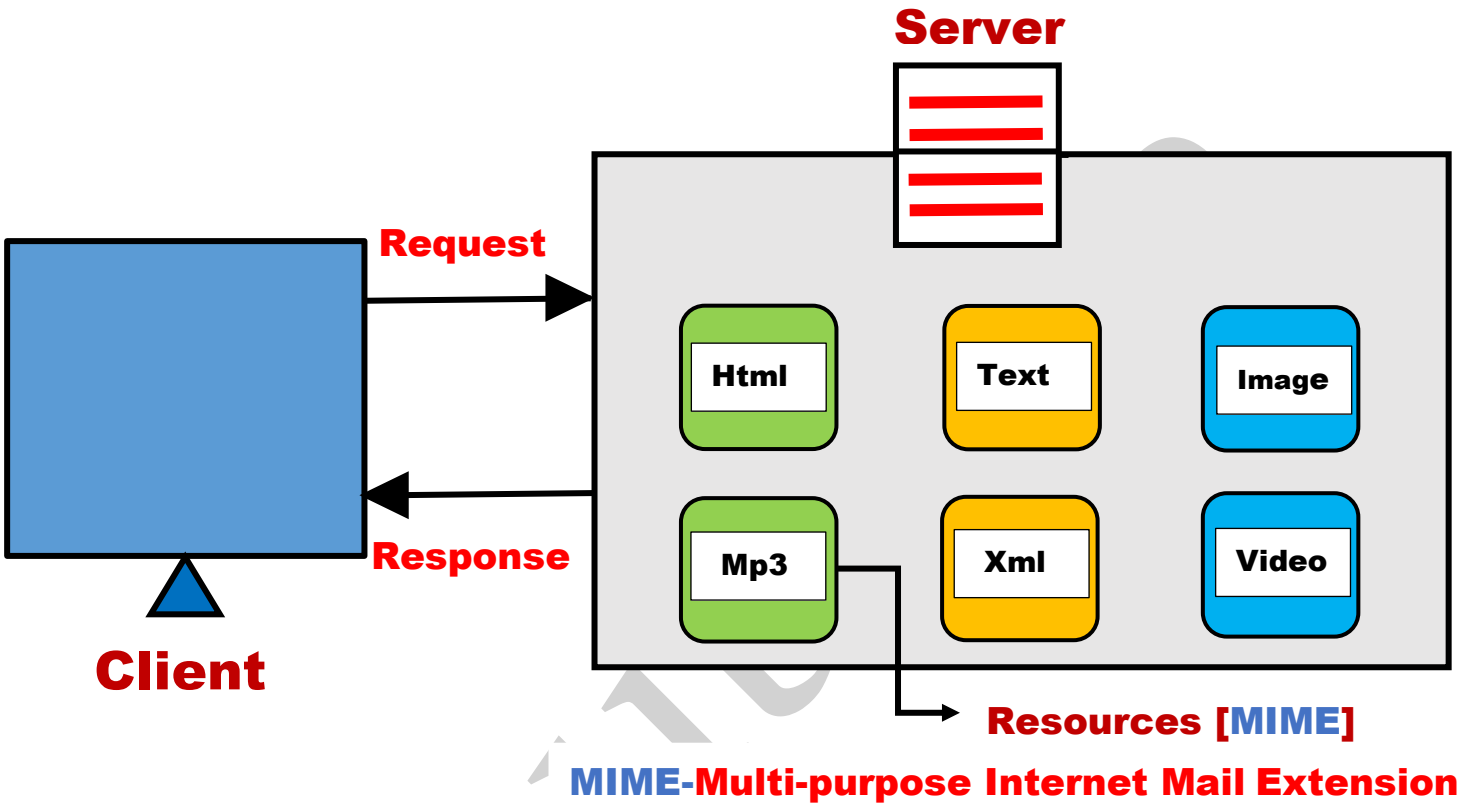
SERVLETS

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



- *The* ***different types of Server*** *are:*

a) Database Server
b) Application Server
c) Web Server

a) Database Server: *It is used to deal with* ***Data*** *(Includes Storage, Retrieval, Management and Manipulation).*

Eg: Oracle, Mysql, Ms-sql, Derby, Sybase, Informix, etc.

b) Application Server: *It is used to execute a* ***Dynamic Application or Real-time Application.***

Eg: JBOSS, IBM WebSphere, Oracle WebLogic.

- ***Apache Tomcat Server*** *internally acts as* ***Application Server.***

c) Web Server: *It is used to execute only* ***Web Application.***

Eg: Apache Tomcat, Oracle Glass Fish.

Dynamic Application: *It is used to execute all the three different types of logic such as* ***Presentation Logic, Persistence Logic and Business Logic.***

Deployment: *Making all the* ***Resources available to the Server.***

- *There are* ***two different types of Deployment*** *present namely:*

a) Manual Deployment

b) Automated Deployment

a) Manual Deployment: *In this case, all the resources are made* ***available to the Server manually.***

b) 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.***

- *Whenever a* ***Web Application is compressed,*** *then we deploy it onto the Server in the form of* ***WAR File.***
- ***WAR*** *refers to* ***Web Archive*** *i.e. it is a* ***compressed version of Web Application.***

Note:

- ***All the Web Applications are deployed onto webapps folder of Apache Tomcat Server.***

Port Number: *It is the one which helps us to* ***get connected to a particular Server.***

Note:

- ***By Default, the port number for Apache Tomcat Server is 8080.***
- ***We can directly deploy a web Application or create a War File and deploy it onto webapps folder of Apache Tomcat Server.***

Folders of Apache Tomcat Server:

- *The **Different folders of Apache Tomcat Server** are as follows:*

a) bin
b) conf
c) lib
d) logs
e) webapps
f) work

a) bin: *It contains a set of startup and shutdown batch files which is used to **start and stop the Apache Tomcat Server.***

b) conf: *It contains a set of **configurations** related to Apache Tomcat Server.*

c) lib: *It contains a set of **Libraries** in the form of **jar file** which is used to perform some additional functionalities.*

d) logs: *It is used to store all the **log messages** displayed on Server Console.*

e) webapps: *It is used to **Deploy all the Web Applications** onto Apache Tomcat Server.*

f) work: *It is used to store the data related to **Translated Servlet.***

Translated Servlet: ***Conversion of JSP into a Servlet.***

Pre-requisite for Apache Tomcat Server:

- *There are **three different pre-requisites for Apache Tomcat Server** namely:*

a) JAVA_HOME
b) CATALINA_HOME
c) Path
d) JRE_HOME (optional)

a) JAVA_HOME: *Open C Drive, Program Files and open Java and **Jdk** Folder and copy the entire path which includes all the folders **related to Jdk** into the Environment Variables.*

- *In particular, System variables present inside Environment Variables.*

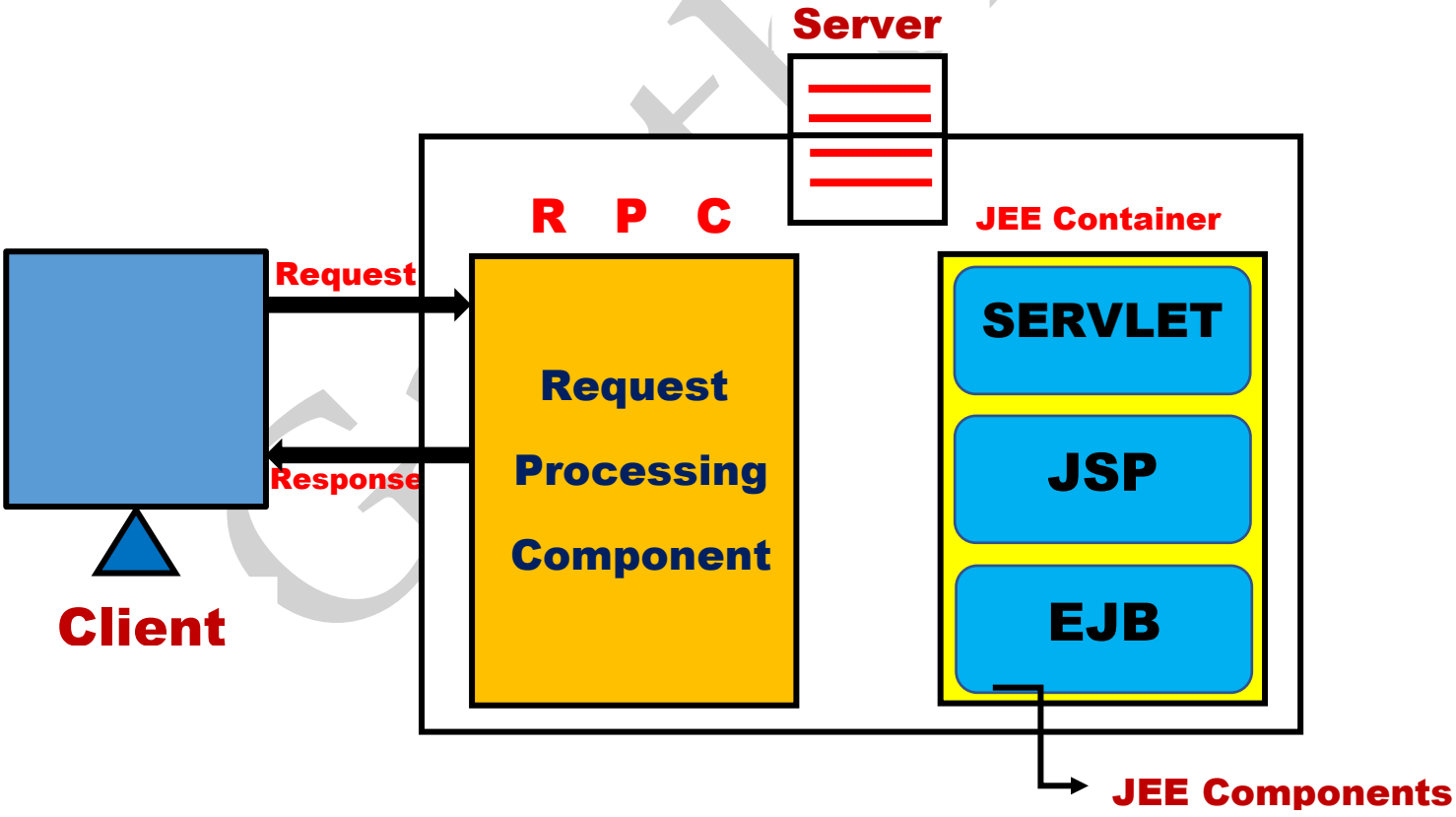
b) CATALINA_HOME: *Open the Apache Tomcat Server folder from the respective directory and copy the entire path which includes all the folders* ***related to Apache Tomcat Server.***

c) Path: *Open the Apache Tomcat Server folder from the respective directory and copy the entire path which includes* ***bin folder of Apache Tomcat Server.***

d) JRE_HOME: *Open C Drive, Program Files and open Java and Jre Folder and copy the entire path which includes all the folders* ***related to Jre.***

> ***Note:***
>
> - ***To run a JEE Application on a particular Server, the Server must mandatorily contain a JEE Container in it.***

JEE CONTAINER

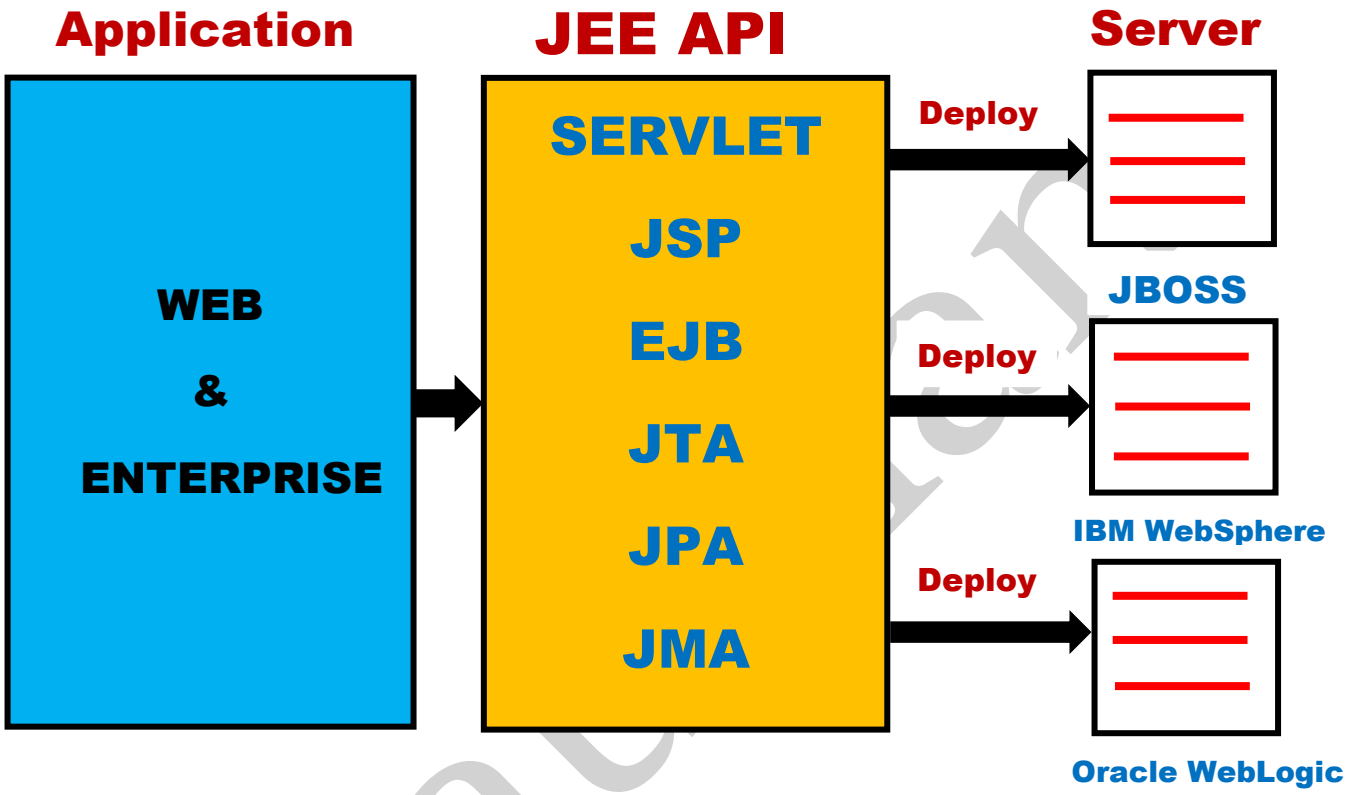


- *It is an* ***Engine*** *which is used to manage all the JEE Components such as* ***Servlet, JSP, EJB, etc.***
- ***Server*** *basically performs* ***two important tasks*** *namely:*

a) *Manages all the Resources.*
b) *Provides a Run-time Environment.*

- *Web Server is used to execute only Web Applications.*
- *Application Server is used to execute both Web and Enterprise Application (JEE Application).*

JEE APPLICATION



Definition: *It is a Specification for developing both Web and Enterprise Application which is given in the form of Abstraction API to achieve Loose Coupling between Application and the Server.*

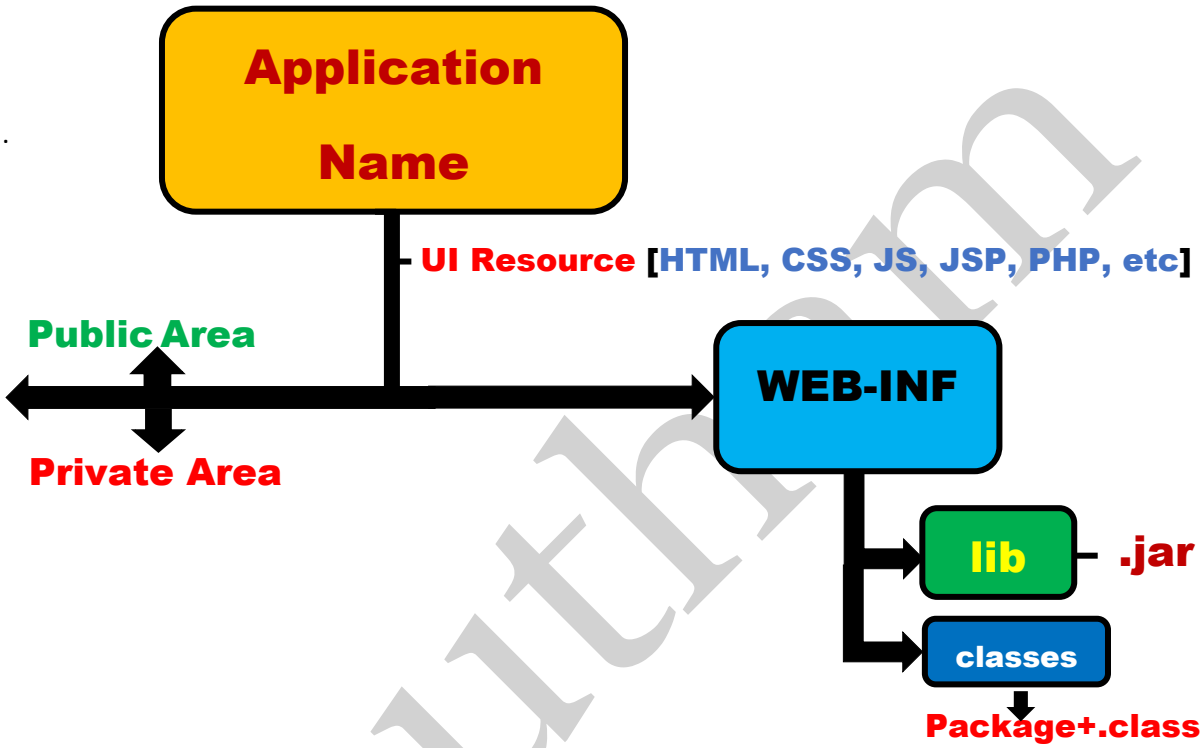
- *Few of the JEE API are:*

a) *Servlet*
b) *JSP- Java Server Page.*
c) *EJB- Enterprise Java Bean.*
d) *JTA- Java Transaction API.*
e) *JPA- Java Persistence API.*
f) *JMA- Java Mail API.*
g) *JMS-Java Message Service.*

Note:

- ***Whenever we deploy any resources onto the Server, the resources must be in the form of .class but not .java.***

Structure of JEE Application



- *Each and every Application must mandatorily have **one web.xml** present in it without which the **JEE Container** fails to load an Application where it throws **Http 404 error (Resources not Available).***
- *We can deploy **multiple applications** onto one single Server but in this case each and 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** by the JEE Container where if there is **any error present in web.xml,** then the JEE Container throws an exception called **ParseException.***
- ***ParseException** is a checked exception since it is thrown at the compilation time.*

Static v/s Dynamic Resources

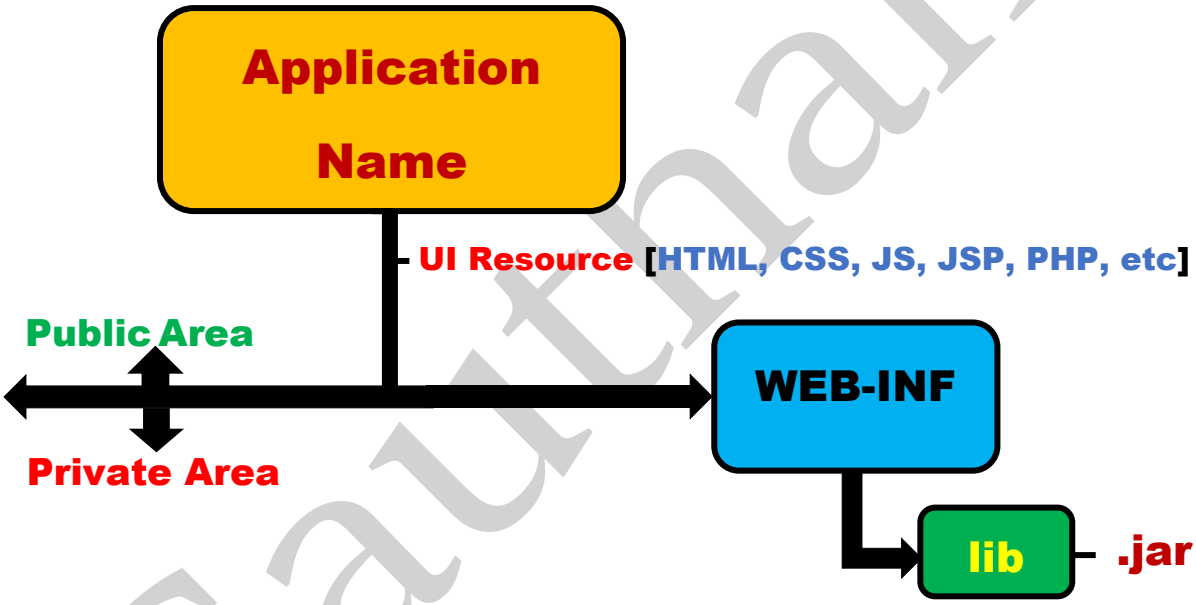
Static Resources: *Resources which are capable of dealing with only* ***Static Data (cannot be changed)*** *are known as* ***Static Resources.***

Eg: Html.

Static Application: *An Application which contains only* ***Static Resources*** *in it is known as* ***Static Application.***

Eg: Standalone Application.

Structure of Static Application



Dynamic Resources: *Resources which are capable of dealing with only* ***Dynamic Data (can be changed)*** *are known as* ***Dynamic Resources.***

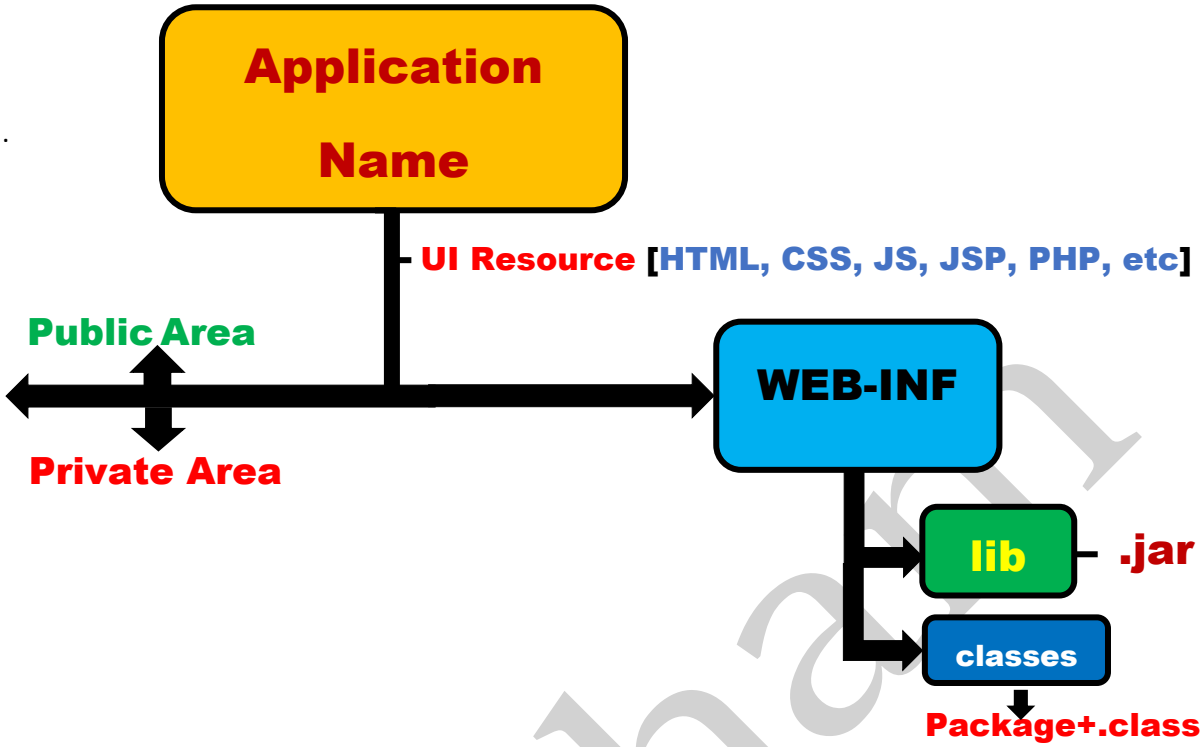
Eg: Servlet, JSP, PHP, etc.

Dynamic Application: *An Application which contains only* ***Dynamic Resources*** *in it is known as* ***Dynamic Application.***

Eg: Any Real-time Application.

- ***Dynamic Resources*** *generally deals with Database server and performs* ***three different types of logic (Presentation, Persistence, Business).***

Structure of Dynamic Application



- *There are **three different types of Logic associated with Dynamic Application** which are as follows:*

a) Presentation Logic
b) Persistence Logic
c) Business Logic

a) Presentation Logic: *It is used to **present** the contents onto an **Application.***

Technologies Involved: ***Html, Css, JavaScript, JSP, PHP, etc.***

b) 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.***

c)Business Logic: *It performs the Core functionality i.e. some set of **Calculation and Validation Operations on an Application.***

Technologies Involved: ***Servlet, Springs, etc.***

- ***Used especially in Payment Gateway.***

Note:

- ***Dynamic Application or Real-time Application* is an integration of all the three different types of Logic such as *Presentation Logic, Persistence Logic and Business Logic.***

Welcome File or Landing Page

- *A File or Page which is **automatically displayed** whenever the Client uses an Application is known as **Welcome File or Landing Page.***
- ***index** is considered to be the **default Welcome File or Landing Page** which is automatically loaded by the JEE Container whenever the Client uses an Application.*
- *We can **explicitly** make any file as **Welcome File or Landing Page** by **renaming it as index.***
- ***Multiple files** can be configured as **Welcome File or Landing Page** but the JEE Container gives the priority **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.***

Code to create a Basic Web Application

HTML Code

```
<!DOCTYPE html>
<html>
<meta charset="ISO-8859-1">
<body bgcolor="yellow">
<h1>BASIC WEB APPLICATION CREATED!!!!!!</h1>
</body>
</html>
```

XML Code

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

```
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
version="3.0">
  <display-name>Sample_Proj</display-name>
  <welcome-file-list>
    <welcome-file>home.html</welcome-file>
  </welcome-file-list>
</web-app>
```

Web.xml or Deployment Descriptor Stub

- *It is a **configuration file** in **xml format** which is used to store the information related to the **configurable resources of an Application.***
- *Each and every Application must mandatorily have only **one web.xml** without which the **JEE Container fails to load an Application** where it throws **Http 404 error** **(Resources Not Available).***
- ***Xml version used in general is 1.0.***

Note:

- ***By default, the root tag for an xml file is <web-app>.***
- ***All the custom tags/user-defined tags are transformated into 8-bit Unicode format based on which all the configurations made in web.xml are understood by the JEE Container.***

Syntax:

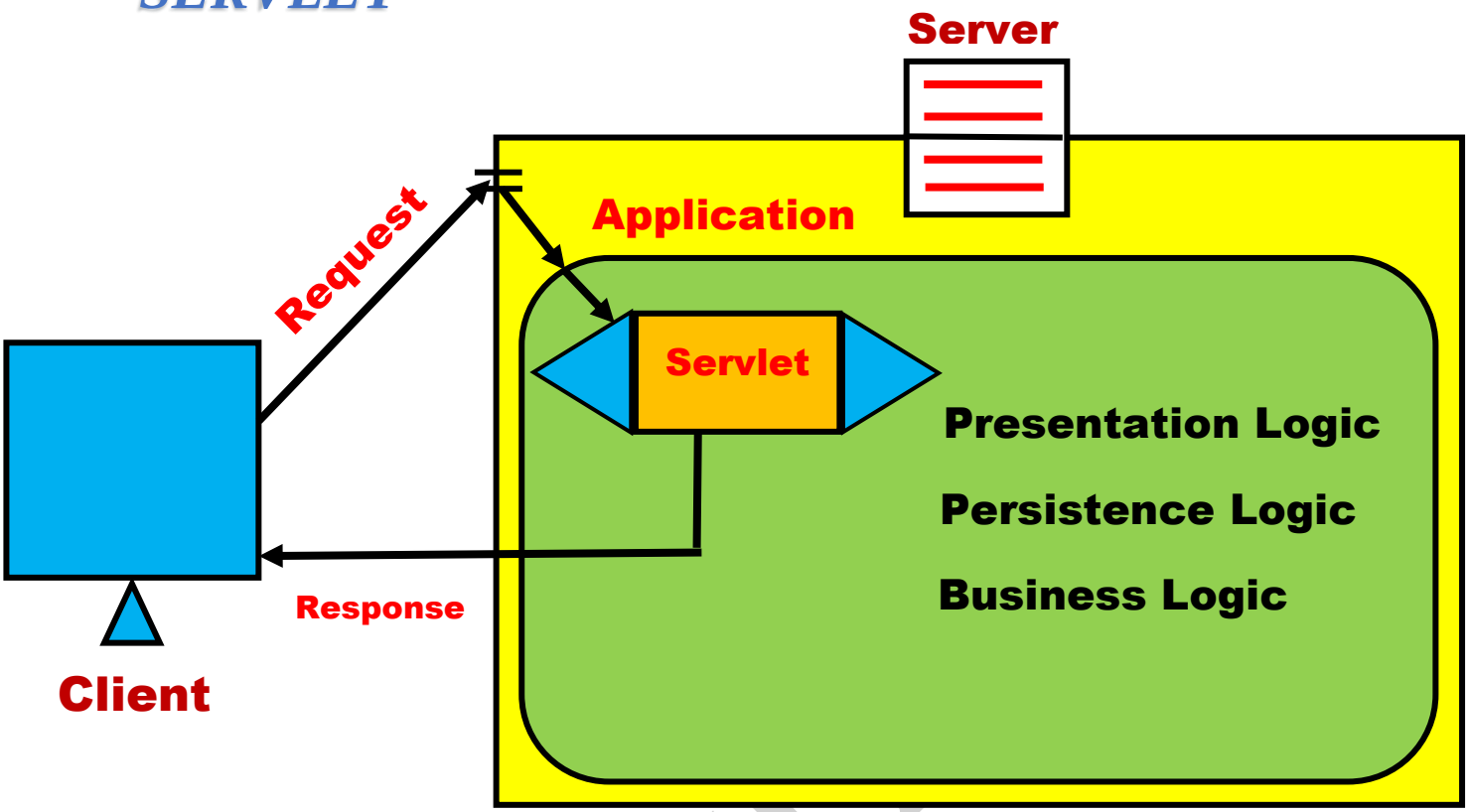
```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
version="3.0">


<! ALL DOCUMENTS ARE HERE!>


</web-app>
```

SERVLET



Definition: ***Servlet is a Server-side Java Program which performs all the three different types of Logic such as Presentation Logic, Persistence Logic and Business Logic along with which process the Http Client Request and get back some Http Response.***

- *There are* ***two different types of Servlet*** *present namely:*

a) GenericServlet
b) HttpServlet

Note:

- ***All the Applications within the Server are managed by the JEE Container.***
- ***All the Servlets within an Application are managed by the Servlet Container.***
- ***Servlet-api.jar is used to provide all the properties related to Servlet.***
- ***In case of Servlet, the properties are automatically Imported.***

a) GenericServlet:

- *Since it is **not specific to any Protocol or independent of Protocol,** hence the name **GenericServlet.***
- ***GenericServlet** does not support **Session.***

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

- ***GenericServlet** is an **Abstract Class** present in javax.servlet package.*
- ***GenericServlet** contains **three methods** in it out of which one is an **abstract method** and the 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.

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

- *Whenever we override **service( ),** it throws an exception called **ServletException and IOException.***
- *The other two methods present in **GenericServlet** are named as **init( )** and **destroy( )** Where **overriding these two methods are optional** since these are concrete methods.*

Syntax:

*+ void **service**(ServletRequest req, ServletResponse resp) throws **ServletException, IOException***

Writing a GenericServlet:

- ***Write a Servlet Class which extends an Abstract class called GenericServlet as follows:***

Note:

- ***Select src folder to create any source code for a Servlet***

```
public class OurServlet extends javax.servlet.GenericServlet
{
  @Override
public void service(ServletRequest req, ServletResponse resp)
    throws ServletException, IOException
    {

        SERVLET LOGIC TO BE WRITTEN HERE


    }
}
```

b) HttpServlet:

- *Since it is **specific** to a particular type of protocol called **Http Protocol,** hence the name **HttpServlet.***
- ***HttpServlet** supports **Session.***
- ***HttpServlet** is an **Abstract class** present in javax.servlet.http package.*
- ***HttpServlet** contains only **concrete methods** without any abstract methods in it.*
- *In case of **HttpServlet,** we need to override the respective concrete method called **doXXX( )** for a **particular type of Http Request.***
- *Whenever we override **doXXX( ),** it throws an **exception called ServletException, IOException.***

Syntax:

 # void doXXX(HttpServletRequest req, HttpServletResponse resp)
*throws **ServletException, IOException***

- *There are **8 different types of Http Request** present namely:*
  - ***POST***
  - ***GET***
  - ***PUT***
  - ***DELETE***
  - ***TRACE***
  - ***OPTION***

- ***HEAD***
- ***CONNECT***

Writing a HttpServlet:

- ***Write a Servlet Class which extends an Abstract class called HttpServlet as follows:***

```
public class OurServlet extends javax.servlet.http.HttpServlet
{
@Override
 protected void doXXX(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException
   {
      SERVLET LOGIC TO BE WRITTEN HERE
   }
}
```

Note:

- ***In case of HttpServlet, it is not a good practice to override the service( ) since HttpServlet depends upon the type of Http Request.***

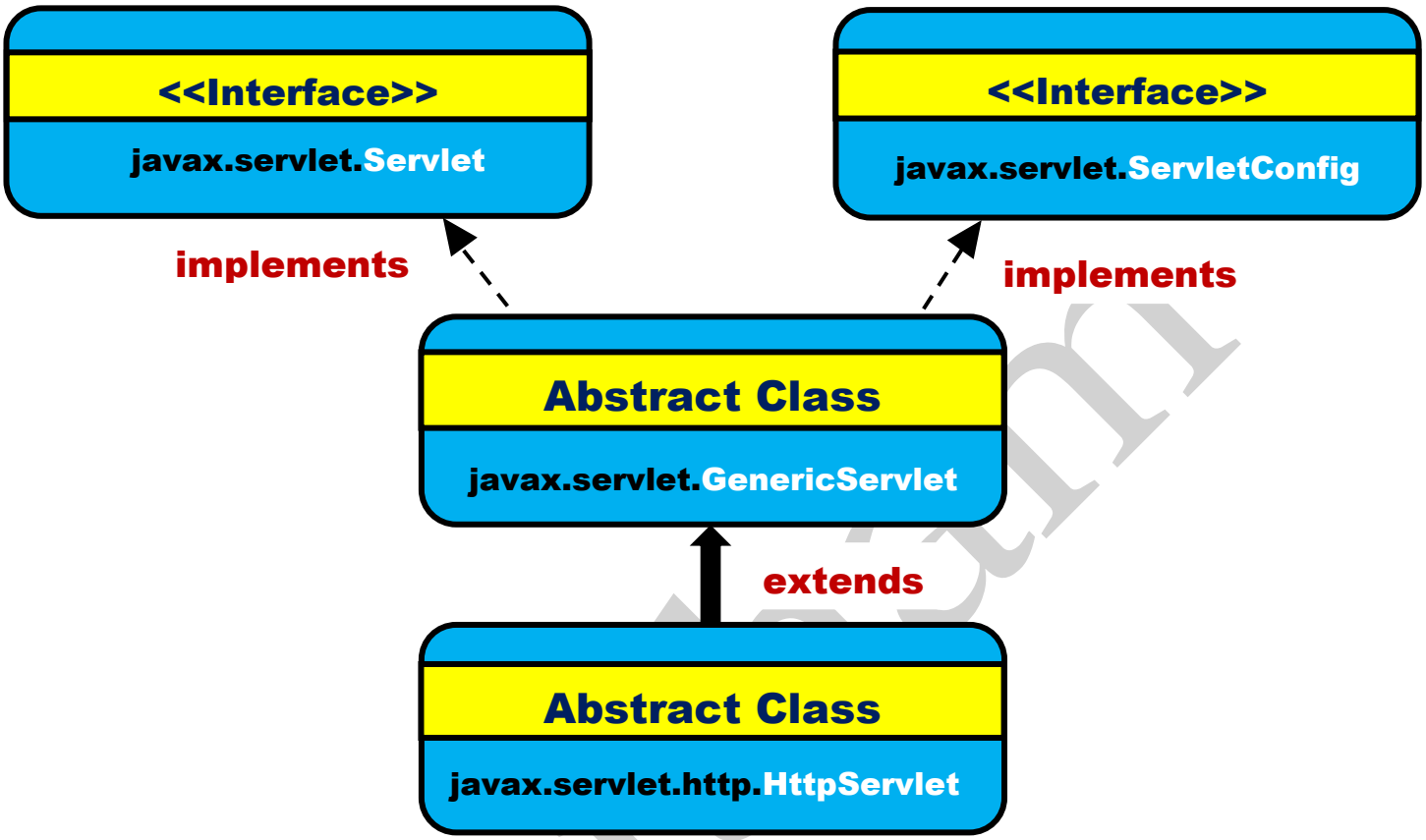
- *Web Resources can be accessed based on **Unique URL Pattern.***
- *Since Servlet is also a Web Resource, it has to be accessed based on **Unique URL Pattern.***
- *Any Resource can always be configured in **two different ways** namely:*

a) Web.xml
b) Annotation

Note:

- ***To configure any resources using Annotation, it is mandatory to use JEE 3.x version or above.***

SERVLET HIERARCHY



Note:

- ***To configure any resources using Annotation, the JEE Version to be used must be 3.0 or above.***

Criteria for a Servlet:

- *There are **three different criteria present for a Servlet namely:***

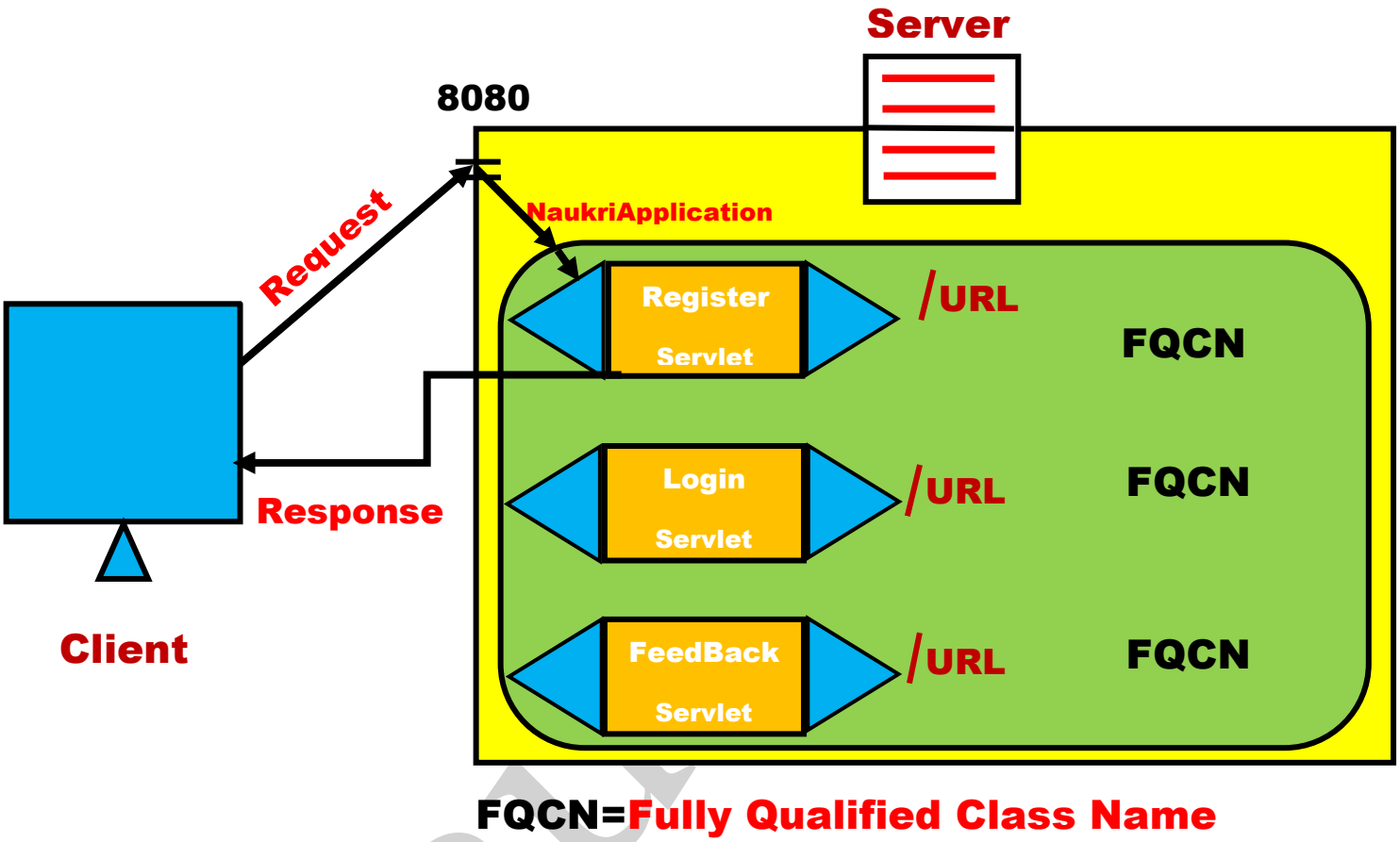
a) Create Servlet
b) Configure Servlet
c) Deploy Servlet

Steps to create a Development Environment:

1) *Open Eclipse in **JEE Perspective.***
2) *Open **Navigator.***
3) *Create a new **Dynamic Web Project** and name the Project.*
4) *Create an appropriate **package structure** under **src** folder.*

5) *Add the **Servlet-api.jar** into **lib** folder.*
6) *Generate a **web.xml** **(Deployment Descriptor Stub).***

Configuration of a Servlet in web.xml:



FQCN=Fully Qualified Class Name

- *Each and every **Servlet** must be mandatorily configured with **three different properties in web.xml namely:***

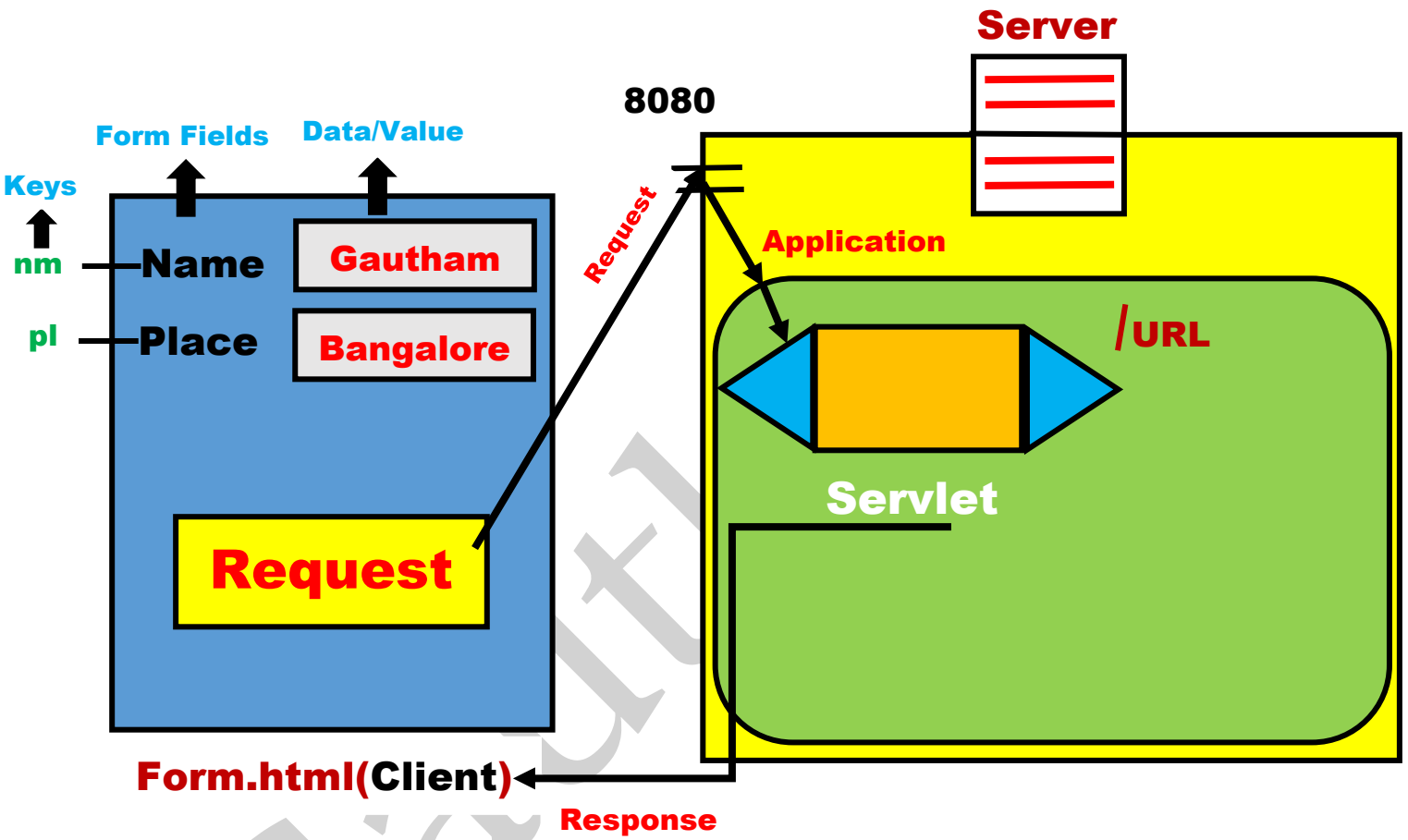
a) Servlet Name
b) URL Pattern
c) Fully Qualified Class Name

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>UniqueName</servlet-name>
<servlet-class>FullyQualifiedClassName</servlet-class>
</servlet>
</web-app>

UI/FORM DATA



Definition: ***The Data which is entered by the enducer (client) on a Form Page and is submitted to the Server in the form of Key and Value Pair is known as UI/FORM DATA.***

- *All the keys associated with Form Page are represented as* ***Unique Identifier*** *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.***
- *The* ***UI/Form Data*** *can be fetched by using* ***getParameter( ).***

+ ***String getParameter(String key)***

*a) In this method, the **key** is taken as an **Argument.***

*b) If the **key** **is present,** then the method returns **associated value.***

*c) If the **key** **is not present,** then the method returns **NULL.***

> ***Note:***
>
> ❖ ***Data present in Request Object is UI/Form Data.***

- *Once a **Request is made**, the data are carried to the **Server** as a part of **Request Object** in the form of **Key and Value Pair** which is displayed in the **URL.***

Code for UI/FORM Data using web.xml

HTML Code

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

Servlet Source Code

```
package org.btm.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>User Details: "+name+"from "+place+"</h1>"
             +"</body></html>");// PRESENTATION LOGIC //
             out.close( );
       }
}
```

XML Code

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
version="3.0">
  <display-name>Ui_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.btm.uiApp.FirstServlet</servlet-class>
  </servlet>
</web-app>
```

Note:

- ***Xml is Case-sensitive Language and also Strictly Defined.***

Code for UI/Form Data using Annotation

HTML code

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

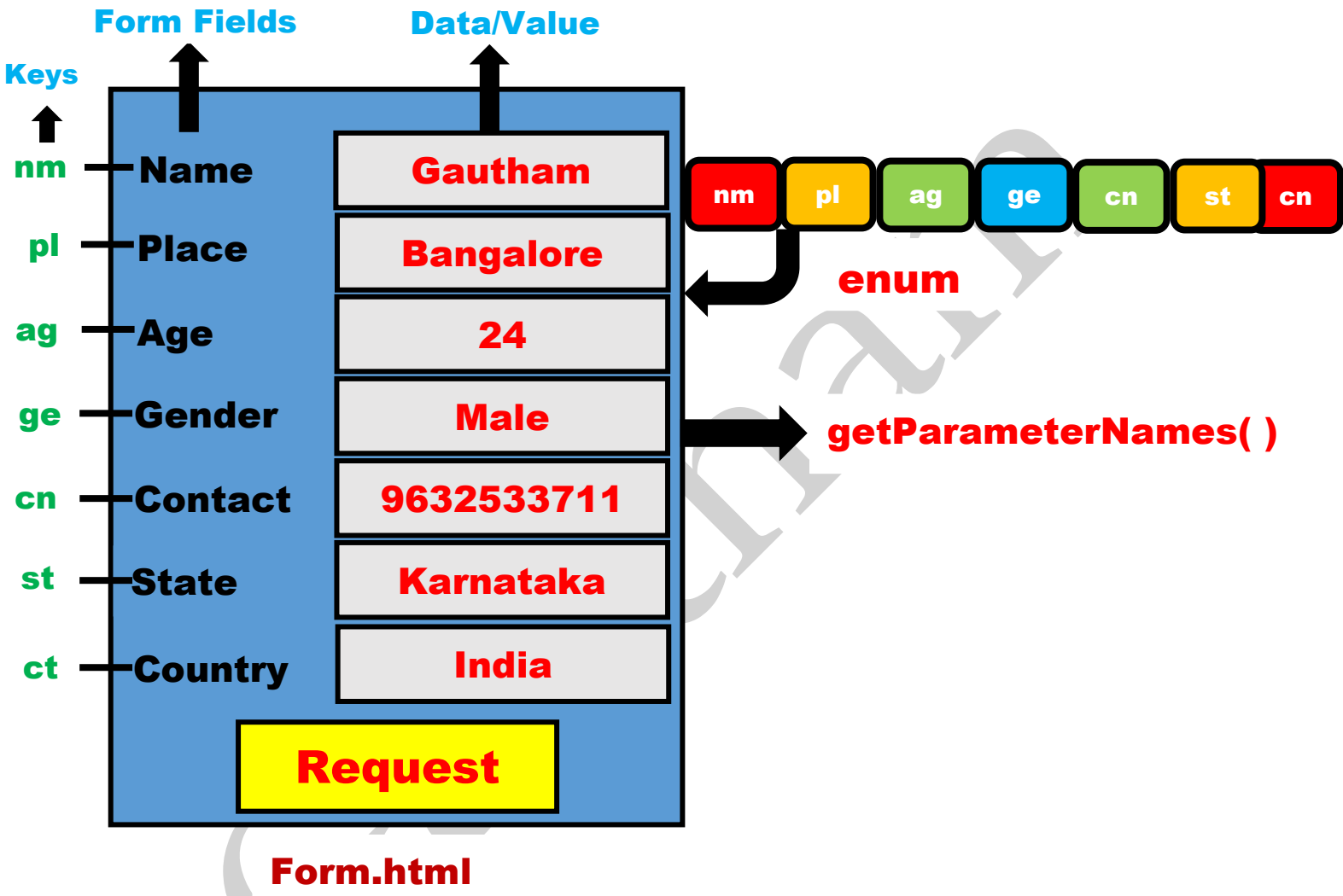
Servlet Source Code

```
package org.btm.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 name=req.getParameter("nm");
             String place=req.getParameter("pl");

             PrintWriter out=resp.getWriter( );
             out.println("<html><body bgcolor='yellow'>"
             +"<h1>User Details: "+name+"from "+place+"</h1>"
             +"</body></html>");// PRESENTATION LOGIC //
             out.close( );
```

```
        }
}
```

ENUMERATION



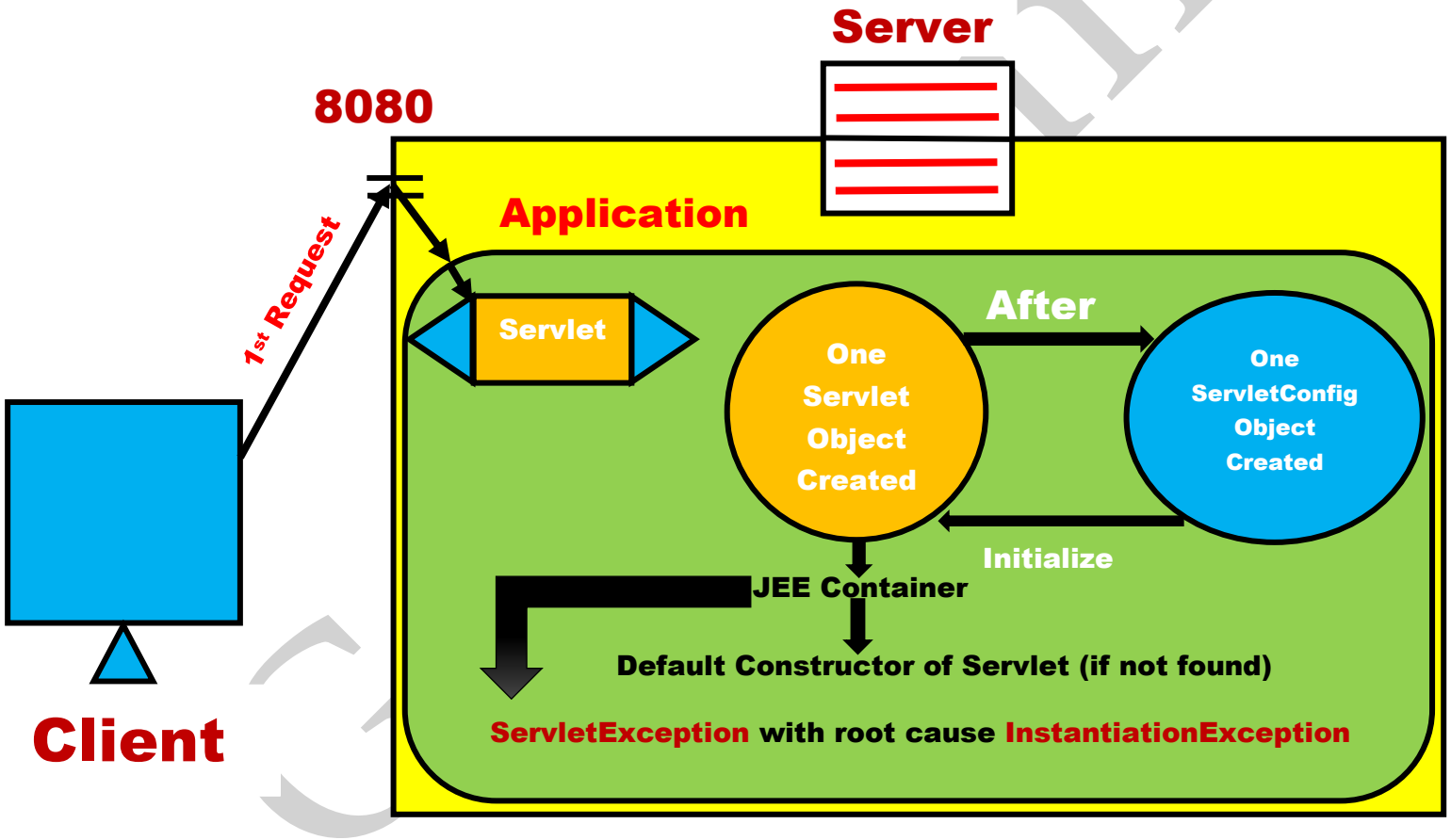
Definition: ***Enumeration is a group of Fixed Data.***

- ***getParameterNames( )*** *is used to fetch* ***all the Keys*** *associated with Form Page at once.*
- ***getParameterNames( )*** *can be used whenever there are* ***n number of form fields*** *present on a Form Page.*
- ***getParameterNames( )*** *returns an* ***Enumeration of String type*** *which contains all the keys associated with Form Page.*

Syntax:

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

SERVLET LIFECYCLE



- *Servlet gets a life and starts its Lifecycle only when a Servlet Object is created.*

***Definition:** Servlet Lifecycle depicts or represents the events or phases from Servlet Object Creation until Servlet Object Destruction.*

- *There are four different lifecycle phases present for a servlet which are as follows:*

a) Instantiation/Object Creation Phase
b) Initialization Phase
c) Service Phase
d) Destruction Phase

a) Instantiation/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 the **default constructor of Servlet** and now Servlet Lifecycle begins.*
- *If the JEE Container does not find the default constructor of Servlet, then the **JEE Container** throws an exception called **ServletException** with a root cause **InstantiationException** (Object not Created).*
- *Immediately **after the Servlet Object Creation,** one **ServletConfig** object is created by the JEE Container which is used to initialize the resources of Servlet Object.*
- *Hence, the scope of **ServletConfig** is always limited to that particular **Servlet Object.***

b) Initialization Phase:

- *In this phase, **Servlet Object has to be Initialized.***
- ***ServletObject** has to be initialized by using **init(ServletConfig)** which takes ServletConfig as a parameter which is used to initialize the resources of Servlet Object.*
- ***init(ServletConfig)** is always called by the JEE Container **only once.***
- ***If this phase fails,** then the JEE Container throws an exception called **ServletException.***

c) Service Phase:

- *In this phase, **service( )** is called which is responsible for **processing each and every client request.***
- *In this phase, the JEE Container creates **one request and one response object** for each and every client request including the **first client request.***
- *By default, **Servlet is multi-threaded** but it can be made **single threaded in two different ways namely:***

a) ***By writing a Servlet Class which implements a marker interface by name called SingleThreaded Model.***
b) ***By making service( ) as Synchronized.***

- *By default, **service( ) is also multi-threaded** i.e. one thread is created for **every client request and service( ) will be executed.***
- *Whenever a client makes a second or subsequent client request to the same Servlet once again, only **service( )** will be executed but the **Servlet object will not be created once again.***
- ***service( )** is called by the JEE Container for **multiple times.***
- ***If this phase fails,** then the JEE Container throws an exception called **ServletException.***

d) Destruction Phase:

- *In this phase, the **destroy( )** is called by the JEE Container to **close all the Costly Resources.***
- ***destroy( )** is called by the JEE Container **only once.***
- ***If this phase fails,** then the performance of an **application decreases.***

Situations of destroy( ):

- ***destroy( )** is called by the JEE Container only once but in **two different situations namely:***

a) ***Whenever we close an Application, destroy( ) is called by the JEE Container to close all the Costly Resources related to that Application.***
b) ***Whenever we redeploy an Application, destroy( ) is called to close all the previously used Costly Resources related to that Application.***

Code for Servlet Lifecycle using web.xml

HTML Code

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

```
Place :<input type="text" name="pl">
<br></br>
<input type="submit" value="Request">
</form>
</body>
</html>
```

Servlet Source Code

```
package org.btm.lifecycleApp;
import java.io.*;
import javax.servlet.*;
public class FirstServlet extends GenericServlet
{
	public FirstServlet( )
	{
		System.out.println("Servlet Object is Created");
	}
	@Override
    public void init(ServletConfig config) throws ServletException
	{
		System.out.println("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>User Details: "+name+"from "+place+"</h1>"
		+"</body></html>");// PRESENTATION LOGIC //
		out.close( );
		System.out.println("service( ) is executed");
	}
	@Override
```

```
        public void destroy( )
        {
                System.out.println("Closed All Costly Resources");
        }
}
```

XML Code

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
version="3.0">
  <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.btm.lifecycleApp.FirstServlet</servlet-class>
  </servlet>
</web-app>
```

Code for Servlet Lifecycle using Annotation

HTML Code

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

```
Place :<input type="text" name="pl">
<br></br>
<input type="submit" value="Request">
</form>
</body>
</html>
```

Servlet Source Code

```
package org.btm.lifecycleApp;
import java.io.*;
import javax.servlet.*;
import javax.servlet.annotation.WebServlet;
@WebServlet("/fs")
Public class FirstServlet extends GenericServlet
{
        public FirstServlet( )
        {
                System.out.println("Servlet Object is Created");
        }
        @Override
      public void init(ServletConfig config) throws ServletException
        {
                System.out.println("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>User Details: "+name+"from "+place+"</h1>"
                +"</body></html>");// PRESENTATION LOGIC //
                out.close( );
                System.out.println("service( ) is executed");
```

```
    }
    @Override
    public void destroy( )
    {
        System.out.println("Closed All Costly Resources");
    }
}
```

LOAD ON STARTUP

- ***Servlet Object*** *can always be created in* ***two different ways*** *namely:*

a) Whenever a client makes a first request to a Servlet.

b) In case of Load-on-Startup.

- ***<load-on-startup>*** *is a tag which is a* ***sub-tag of <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.***
- *In case of* ***Load-on-Startup,*** *only* ***service( )*** *will be executed even for the* ***first client request.***
- ***Load-on-Startup*** *must always be configured with a* ***positive integer value*** *but the JEE Container gives the priority based on* ***lowest positive integer value.***
- *Whenever two Servlets are configured with the* ***same positive integer*** *value, then* ***sequential execution takes place.***

Note:

- ***Whenever Load-on-Startup is configured with a negative integer value, then the Servlet Object is created based on first client request.***

Code for Servlet Lifecycle in case of load-on-startup using web.xml

HTML code

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

Servlet Source Code

```
package org.btm.lifecycleApp;
import java.io.*;
import javax.servlet.*;
public class FirstServlet extends GenericServlet
{
        public FirstServlet( )
        {
                System.out.println("Servlet Object is Created");
        }
        @Override
      public void init(ServletConfig config) throws ServletException
        {
                System.out.println("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>User Details: "+name+"from "+place+"</h1>"
            +"</body></html>");// PRESENTATION LOGIC //
            out.close( );
            System.out.println("service( ) is executed");
        }
        @Override
        public void destroy( )
        {
            System.out.println("Closed All Costly Resources");
        }
}
```

XML Code

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
version="3.0">
  <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.btm.lifecycleApp.FirstServlet</servlet-class>
  <load-on-startup>5</load-on-startup>
  </servlet>
</web-app>
```

Code for Servlet Lifecycle in case of load-on-startup using Annotation

HTML Code

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

Servlet Source Code

```
package org.btm.lifecycleApp;
import java.io.*;
import javax.servlet.*;
import javax.servlet.annotation.WebServlet;
@WebServlet(urlPatterns="/fs", loadOnStartup=5)
public class FirstServlet extends GenericServlet
{
	public FirstServlet( )
	{
		System.out.println("Servlet Object is Created");
	}
	@Override
    public void init(ServletConfig config) throws ServletException
	{
		System.out.println("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>User Details: "+name+"from "+place+"</h1>"
            +"</body></html>");// PRESENTATION LOGIC //
            out.close( );
            System.out.println("service( ) is executed");
      }
      @Override
      public void destroy( )
      {
            System.out.println("Closed All Costly Resources");
      }
}
```


ServletConfig

- ***ServletConfig*** *is an* ***Interface*** *which is present in* ***javax.servlet package.***
- *Since it is an Interface an* ***implementation object*** *can be created by the JEE Container by using a factory or helper method called* ***getServletConfig( )*** ***immediately after the Servlet Object Creation.***
- *Only one implementation object of* ***ServletConfig*** *is created for a* ***particular Servlet Object.***
- *The data present in* ***ServletConfig*** *object are* ***init/initialization parameters*** *which is made available only for that* ***particular Servlet Object.***
- *Hence, the scope of* ***ServletConfig Object*** *is limited to that* ***particular Servlet Object.***
- ***init/initialization parameters*** *are the data in the form of key and value pair which is used to* ***initialize the resources of Servlet Object.***

- *The **init/initialization parameters** present in ServletConfig 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.***
- *Whenever there are **n number of init/initialization parameters** declared, then we use **getInitParameterNames( ).***

Syntax:

ServletConfig config=getServletConfig( );

config.getInitParameter(key);

ServletContext

- ***ServletContext** is an **Interface** present in **javax.servlet package.***
- *Since it is an **Interface**, an implementation object is created by the JEE Container by using the factory/helper method called **getServletContext( ) at the time of Application Loading.***
- *Only **one implementation Object** of **ServletContext** is created by the JEE Container for **one entire Application** which has many **references shared throughout the Application.***
- *The data present in **ServletContext** are **context parameters** and **attributes** which is made **available throughout the Application.***
- *Hence, the scope of **ServletContext** **is throughout the Application.***
- ***context parameters** are the data in the form of key and value pair which is made **available throughout the Application.***
- *The context parameters present in **ServletContext** 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.***
- *Whenever there are **n number of context parameters** declared, then we use **getInitParameterNames( ).***

Syntax:

ServletContext context=getServletContext( );

context.getInitParameter(key);

Parameter

- ***It is the data in the form of key and value pair where key must be a Unique String and the value can also be a String.***

Eg:

UI/Form Data

Syntax: req.getParameter(Key);

init/initialization parameters

Syntax: config.getInitParameter(key);

Context parameters

Syntax: context.getInitParameter(key);

Attribute

- ***It is the data in the form of key and value pair where key must be a Unique String and the value can be any Object.***
- *Attribute can be associated with request, session and context object.*
- *Attribute cannot be associated with config object.*
- *Attribute can be added into the Scope by using* ***setAttribute( ).***
  + void setAttribute(String key, Object obj)
- *Attribute can be fetched back from the Scope by using* ***getAttribute( ).***
  + void getAttribute(String key)

Http Post

- ***Post request is used to post some contents or data from the Client to the Server.***
- *Post request deals with* ***Unlimited data.***

- *Post request is **non-idempotent.***
- *Post request cannot be **Bookmarked.***
- *In case of **Post request, the data are carried to the Server** as a part of **Http Request Body** which is not displayed even to the **Enducer**(**Client**). Hence the data are **Secured.***
- *Whenever we deal with **'N' number of data,** then the type of request is Post.*

Code to Dynamically save the Data into the Database Server using Post Request

HTML Code

```
<!DOCTYPE html>
<html>
<meta charset="ISO-8859-1">
<body bgcolor="cyan">
<form action="fs" method="post">
<table>
<tr>
<td>Id: </td>
<td><input type="text" name="i"></td>
</tr>
<tr>
<td>Name: </td>
<td><input type="text" name="nm"></td>
</tr>
<tr>
<td>Dept: </td>
<td><input type="text" name="dp"></td>
</tr>
<tr>
<td>Perc: </td>
<td><input type="text" name="pr"></td>
</tr>
<tr>
<td></td>
```

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

Servlet Source Code

```
package org.btm.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, IOException
        {
            String sid=req.getParameter("i");
            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'>"
          +"<h1>Student Details are: "+name+"from "+dept+
           " Department"+"</h1>"+ "</body></html>");
            // PRESENTATION LOGIC //
            out.close( );
```

```
                Connection con=null;
                PreparedStatement pstmt=null;
        String qry="insert into btm.student values(?,?,?,?)";
try {
            Class.forName("com.mysql.jdbc.Driver");
      con=DriverManager.getConnection("jdbc:mysql://localhost:
                        3306?user=root&password=root");
                    pstmt=con.prepareStatement(qry);
// SET THE VALUE FOR PLACEHOLDER BEFORE
                            EXECUTION //
                    pstmt.setInt(1,id);
                    pstmt.setString(2,name);
                    pstmt.setString(3,dept);
                    pstmt.setDouble(4,perc);
                    pstmt.executeUpdate( );
                    // PERSISTENCE 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( );
                        }
```

```
                    }
                }
            }
}
```

XML Code

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
version="3.0">
  <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.btm.postApp.FirstServlet</servlet-class>
  </servlet>
</web-app>
```

Http Get

- ***Get request is used to get some contents or resources from the Server.***
- *Get request deals with only* ***Limited Data*** ***i.e. 1024 Characters.***
- *Get request is* ***Idempotent.***
- *Get request can be* ***Bookmarked.***
- *In case of* ***Get request, the data are carried to the Server*** *as a part of* ***Request Object*** *in the form of* ***Key and Value pair*** *which is displayed to the* ***Enducer(Client)*** *in the* ***URL.*** *Hence, the data are* ***not Secured.***
- *Whenever the type of request is* ***not mentioned or configured,*** *then by* ***default the type of request is Get request.***

- *Whenever we deal with a **Link,** then the type of request is **Get request.***

Code to Fetch the Data from the Database Server using Get Request

HTML Code

```
<!DOCTYPE html>
<html>
<meta charset="ISO-8859-1">
<body bgcolor="cyan">
<form action="fs" method="get">
Id:<input type="text" name="i">
<br></br>
<input type="submit" value="Get">
</form>
</body>
</html>
```

Servlet Source Code

```
package org.btm.getApp;
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet
{
	@Override
protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException
	{
		String sid=req.getParameter("i");
		int id=Integer.parseInt(sid);

		Connection con=null;
		PreparedStatement pstmt=null;
		ResultSet rs=null;
		String qry="select * from btm.student where id=?";
```

```
        try {
                Class.forName("com.mysql.jdbc.Driver");
con=DriverManager.getConnection("jdbc:mysql://localhost:
                        3306?user=root&password=root");
                pstmt=con.prepareStatement(qry);
            // SET THE VALUE FOR PLACEHOLDER BEFORE
                                                EXECUTION //
                pstmt.setInt(1,id);
                rs=pstmt.executeQuery( );
                // PERSISTENCE LOGIC //
                PrintWriter out=resp.getWriter( );
                if(rs.next( ))
                {
                        String name=rs.getString(2);
                        String dept=rs.getString(3);
                out.println("<html><body bgcolor='yellow'>"
            +"<h1>Student Details are "+name+"from "+dept+
             " Department"+"</h1>"+"</body></html>");
                // PRESENTATION LOGIC //
                                out.close( );
                }
                else
                {
                        out.println("<html><body bgcolor='pink'>"
                                +"<h1>No Data Found!!!!</h1>"
                                +"</body></html>");
                        // PRESENTATION LOGIC //
                                out.close( );
                }
        } catch (ClassNotFoundException | SQLException e) {
                e.printStackTrace( );
        }
        finally
        {
                if(rs!=null)
                  {
                        try {
```

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

XML Code

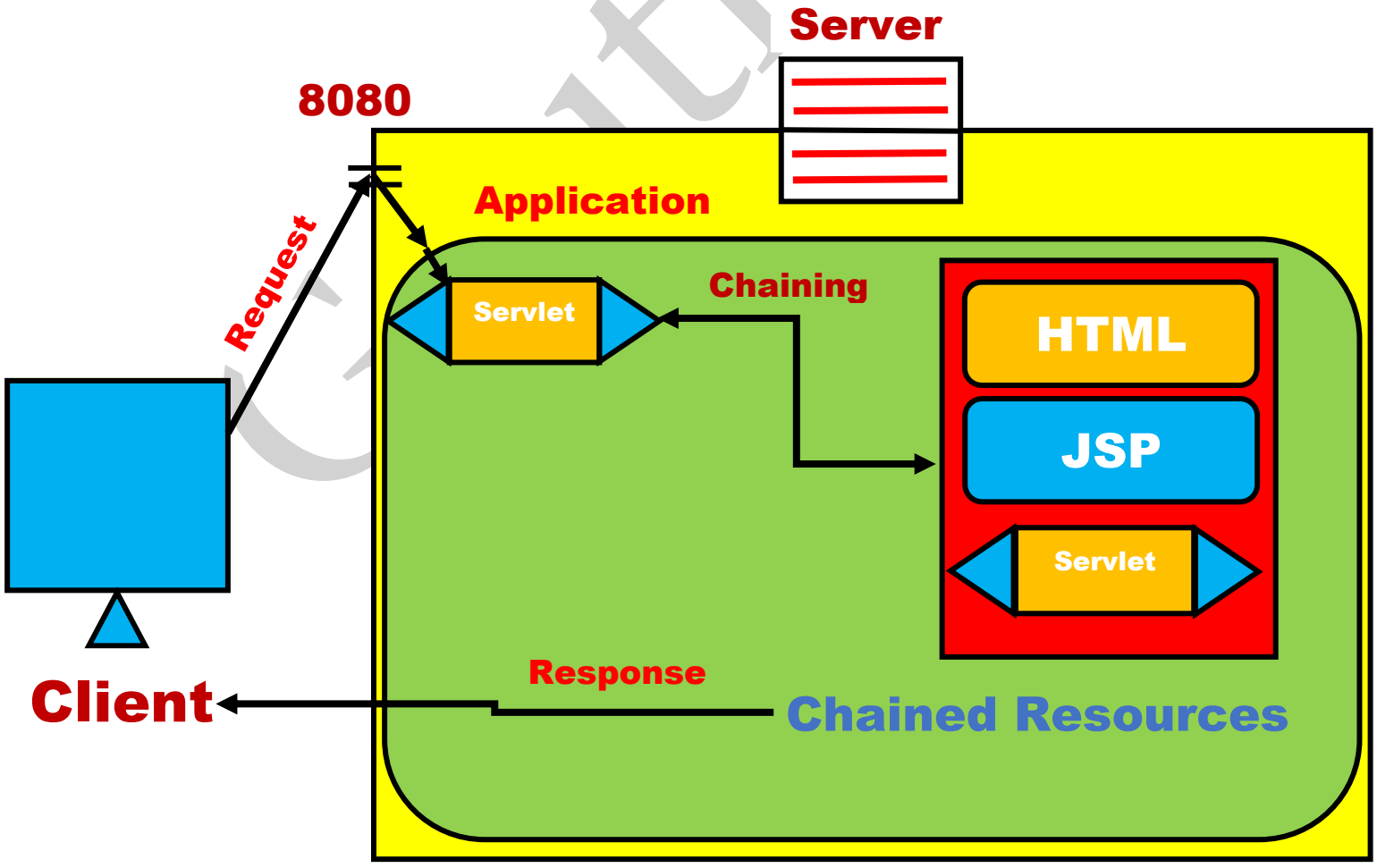
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
version="3.0">
  <display-name>Get_Proj</display-name>
  <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.btm.getApp.FirstServlet</servlet-class>
  </servlet>
</web-app>
```

SERVLET CHAINING

Definition: *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 needed to make the data present in one Servlet available to other resources which can either be a Html, JSP or another Servlet.*



- *There are **two different ways** in which we can perform **Servlet Chaining** namely:*

a) RequestDispatcher

b) sendRedirect

a) RequestDispatcher

- ***RequestDispatcher** is an **Interface** present in **javax.servlet package.***
- *Since it is an **Interface**, an implementation object can be created by the JEE Container by using a factory or helper method by name called **getRequestDispatcher( )** whenever a client makes **any request to a Servlet.***

  RequestDispatcher rd = req.getRequestDispatcher( );
- *We can get the reference of **RequestDispatcher** Interface in two different ways namely:*

  i) request(req)

  ii) context
- *Whenever we use **req.getRequestDispatcher( ),** then we can chain from one Servlet to another resources of the **same Application.***
- *Whenever we use **context.getRequestDispatcher( ),** then we can chain from one Servlet to another resources of **different Application.***
- *There are **two different methods** present in **RequestDispatcher** namely **forward( )** and **include( )***

  + void forward(ServletRequest req, ServletResponse resp)

  + void include(ServletRequest req, ServletResponse resp)
- *In case of **forward( ),** the request is **forwarded** from one Servlet to another resources(Html, JSP, Servlet) and finally the **chained resource gets back the response.***
- *In case of **include( ),** all the chained resources are **included** into main or source Servlet and finally the **main or source Servlet gets back the response.***

Note:

- ***In case of forward( ) and include( ), the URL remains the same.***
- ***Hence, both forward( ) and include( ) works at the Server Side i.e. RequestDispatcher works at the Server Side.***

b) sendRedirect

- ***sendRedirect*** *is a method of* ***HttpServletResponse*** ***Interface*** *present in* ***javax.servlet.http package.***
- *Using* ***sendRedirect( ),*** *we can chain to external resources also.*
- ***sendRedirect( )*** *can be used to chain to other resources where the* ***data are not involved.***

Note:

- ***In case of sendRedirect( ), the URL pattern changes which is displayed on the Browser.***